



```
public PointInPolygonExample() {
    appWidth = 640;
    appHeight = 640;
    appTitle = "Point In Polygon Example";
    appBackground = Color.WHITE;
    appFPSColor = Color.BLACK;
}
@Override
protected void initialize() {
    super.initialize();
    // polygon points and lists of point inside
    // and outside the polygon
    poly = new ArrayList<Vector2f>();
    polyCpy = new ArrayList<Vector2f>();
    inside = new ArrayList<Vector2f>();
    outside = new ArrayList<Vector2f>();
    mousePos = new Vector2f();
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    mousePos = getWorldMousePosition();
    // draw polygon for algorithm testing
    if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
        winding = !winding;
    }
    if( mouse.buttonDownOnce( MouseEvent.BUTTON1 ) ) {
        poly.add( mousePos );
    }
    if( mouse.buttonDownOnce( MouseEvent.BUTTON3 ) ) {
        poly.clear();
    }
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    // see if the mouse is inside the polygon
    selected = pointInPolygon( mousePos, poly, winding );
    // test random points against the polygon
    Random rand = new Random();
    inside.clear();
    outside.clear();
    for( int i = 0; i < MAX_POINTS; ++i ) {
```



```
    float x = rand.nextFloat() * 2.0f - 1.0f;
    float y = rand.nextFloat() * 2.0f - 1.0f;
    Vector2f point = new Vector2f( x, y );
    if( pointInPolygon( point, poly, winding ) ) {
        inside.add( point );
    } else {
        outside.add( point );
    }
}
}
private boolean pointInPolygon( Vector2f point, List<Vector2f> poly,
    boolean winding ) {
    // point in polygon algorithm
    int inside = 0;
    if( poly.size() > 2 ) {
        Vector2f start = poly.get( poly.size() - 1 );
        boolean startAbove = start.y >= point.y;
        for( int i = 0; i < poly.size(); ++i ) {
            Vector2f end = poly.get( i );
            boolean endAbove = end.y >= point.y;
            if( startAbove != endAbove ) {
                float m = (end.y - start.y) / (end.x - start.x);
                float x = start.x + (point.y - start.y) / m;
                if( x >= point.x ) {
                    if( winding ) {
                        inside += startAbove ? 1 : -1;
                    } else {
                        inside = inside == 1 ? 0 : 1;
                    }
                }
            }
            startAbove = endAbove;
            start = end;
        }
    }
    return inside != 0;
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    // render instructions
    g.drawString( "Winding: " + (winding?"ON":"OFF"), 20, 35 );
    String mouse =
```



```
String.format( "Mouse: (%.2f,%.2f)", mousePos.x, mousePos.y );
g.drawString( mouse, 20, 50 );
g.drawString( "Left-Click to add points", 20, 65 );
g.drawString( "Right-Click to clear points", 20, 80 );
g.drawString( "Space Bar to toggle winding", 20, 95 );
Matrix3x3f view = getViewportTransform();
// draw test polygon
if( poly.size() > 1 ) {
    polyCpy.clear();
    for( Vector2f vector : poly ) {
        polyCpy.add( view.mul( vector ) );
    }
    g.setColor( selected ? Color.GREEN : Color.BLUE );
    Utility.drawPolygon( g, polyCpy );
}
// draw inside point blue, outside points red
g.setColor( Color.BLUE );
for( Vector2f vector : inside ) {
    Vector2f point = view.mul( vector );
    g.fillRect( (int)point.x, (int)point.y, 1, 1 );
}
g.setColor( Color.RED );
for( Vector2f vector : outside ) {
    Vector2f point = view.mul( vector );
    g.fillRect( (int)point.x, (int)point.y, 1, 1 );
}
}
public static void main( String[] args ) {
    launchApp( new PointInPolygonExample() );
}
```

## 7.2 使用轴对齐边界框进行相交测试

轴对齐边界框（AABB）是边与 x 轴和 y 轴平行的一个矩形，如图 7.6 所示。

由于是矩形不是旋转的，因此它是很容易用于相交测试的一种形状。矩形可以仅仅使用一个最小值和一个最大值来表示，如图 7.7 所示。注意，AABB 是不能旋转的。

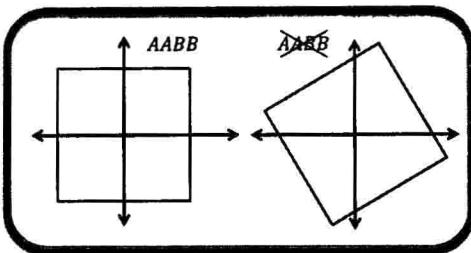


图 7.6 轴对齐边界框

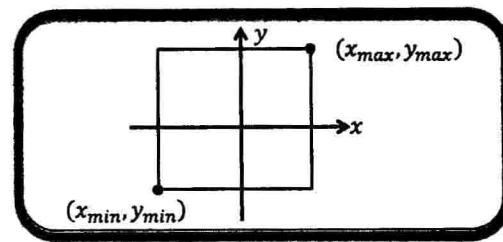


图 7.7 AABB 表示

变换和绘制一个 AABB 很简单。

```
private void drawAABB( Graphics g, Vector2f min, Vector2f max ) {
    Matrix3x3f view = getViewportTransform();
    Vector2f topLeft = new Vector2f( min.x, max.y );
    topLeft = view.mul( topLeft );
    Vector2f bottomRight = new Vector2f( max.x, min.y );
    bottomRight = view.mul( bottomRight );
    int rectX = (int)topLeft.x;
    int rectY = (int)topLeft.y;
    int rectWidth = (int)( bottomRight.x - topLeft.x );
    int rectHeight = (int)( bottomRight.y - topLeft.y );
    g.drawRect( rectX, rectY, rectWidth, rectHeight );
}
```

由于该形状是不能旋转的，因此你只需要检查 x 和 y 的最小值和最大值之间是否有一个点。检查一个点是否在 AABB 中，就是这么简单。

```
private boolean pointInAABB( Vector2f p, Vector2f min, Vector2f max ) {
    return p.x > min.x && p.x < max.x &&
           p.y > min.y && p.y < max.y;
}
```

检查两个 AABB 是否重叠也很简单。如果没有任何值重叠，那么，就没有相交，如图 7.8 所示。

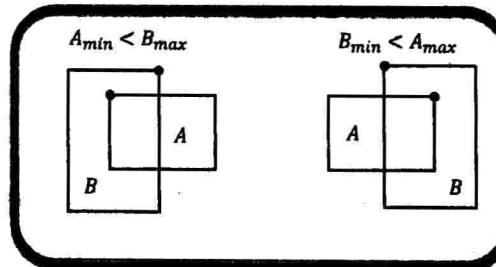


图 7.8 AABB 相交



```
private boolean intersectAABB( Vector2f minA, Vector2f maxA,
    Vector2f minB, Vector2f maxB ) {
    if( minA.x > maxB.x || minB.x > maxA.x ) return false;
    if( minA.y > maxB.y || minB.y > maxA.y ) return false;
    return true;
}
```

## 7.3 使用圆形测试相交

对于相交测试来说，圆形也是很有用的形状。圆形可以表示为一个圆点和一个半径，如图 7.9 所示。

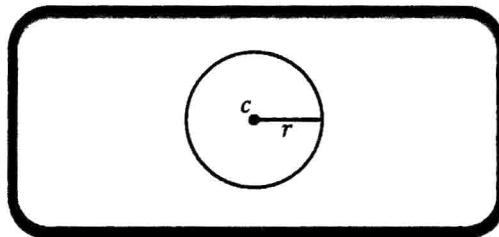


图 7.9 圆形的表示

可以绘制一个圆形，以便将其延展到窗口的大小，具体方式与绘制 AABB 相同。

```
private void drawOval( Graphics g, Vector2f center, float radius ) {
    Matrix3x3f view = getViewportTransform();
    Vector2f topLeft =
        new Vector2f( center.x - radius, center.y + radius );
    topLeft = view.mul( topLeft );
    Vector2f bottomRight =
        new Vector2f( center.x + radius, center.y - radius );
    bottomRight = view.mul( bottomRight );
    int circleX = (int)topLeft.x;
    int circleY = (int)topLeft.y;
    int circleWidth = (int)(bottomRight.x - topLeft.x);
    int circleHeight = (int)(bottomRight.y - topLeft.y);
    g.drawOval( circleX, circleY, circleWidth, circleHeight );
}
```

要检查一个点是否在圆形中，应将该点与圆心的距离和半径进行比较，如图 7.10 所示。



如果半径比这个距离大，则该点位于圆形之中。

由于求平方根的方法可能较慢，检查距离的平方会更容易一些。

$$\text{if } a < b \text{ then } a^2 < b^2$$

后面示例中的距离比较，大多采用的是平方后的距离。如下面这个方法所示：

```
private boolean pointInCircle( Vector2f p, Vector2f c, float r ) {
    Vector2f dist = p.sub( c );
    return dist.lenSqr() < r*r;
}
```

该测试检查两个圆形是否重叠，这和点的测试非常类似。如果到圆心的距离比两个圆的半径之和要大，两个圆不会重合，如图 7.11 所示。

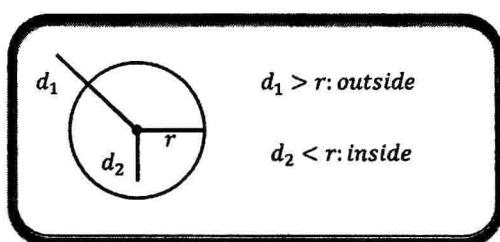


图 7.10 圆中的点

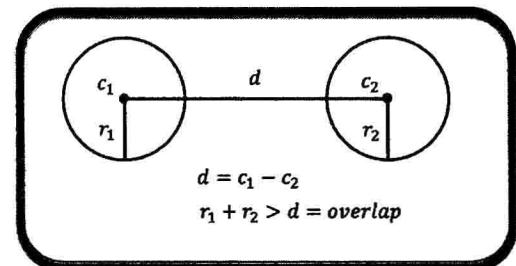


图 7.11 圆的重合

测试圆的重合的代码，如下所示：

```
private boolean intersectCircle(
    Vector2f c0, float r0, Vector2f c1, float r1 ) {
    Vector2f c = c0.sub( c1 );
    float r = r0 + r1;
    return c.lenSqr() < r*r;
}
```

## 圆和 AABB 的重合

要判断一个圆和一个 AABB 是否重合，需要计算圆到 AABB 的距离。圆心要么在 AABB 之外，要么在 AABB 之内。如果圆心小于 AABB 的最小值，它们之间的距离就是 AABB 的最小值和圆心之间的差距。如果圆心大于 AABB 的最大值，它们之间的距离就是 AABB 的最大值和圆心的差距。如果圆心大于 AABB 最小值但是小于 AABB 最大值，那么圆心位于



AABB 之中，并且不需要再进行测试，如图 7.12 所示。使用每个 x 和 y 计算距离，找到 AABB 上最近的点，这可以用于测试圆半径距离，就像前面示例中所做的那样。

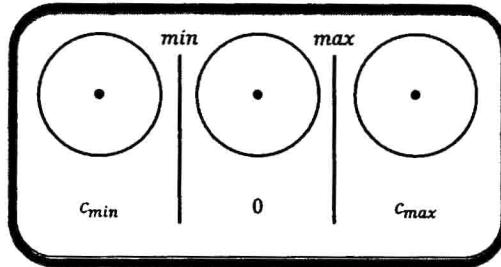


图 7.12 圆和 AABB 的情况

该算法可以编写如下：

```
private boolean intersectCircleAABB(  
    Vector2f c, float r, Vector2f min, Vector2f max ) {  
    float dx = 0.0f;  
    if( c.x < min.x ) dx = c.x - min.x;  
    if( c.x > max.x ) dx = c.x - max.x;  
    float dy = 0.0f;  
    if( c.y < min.y ) dy = c.y - min.y;  
    if( c.y > max.y ) dy = c.y - max.y;  
    float d = dx*dx + dy*dy;  
    return d < r*r;  
}
```

可以将 dx 和 dy 变量组合到一个单个的变量中，在同一行执行平方和相加计算，从而优化这段代码。优化后的版本如下所示：

```
private boolean intersectCircleAABB(  
    Vector2f c, float r, Vector2f min, Vector2f max ) {  
    float d = 0.0f;  
    if( c.x < min.x ) d += (c.x - min.x)*(c.x - min.x);  
    if( c.x > max.x ) d += (c.x - max.x)*(c.x - max.x);  
    if( c.y < min.y ) d += (c.y - min.y)*(c.y - min.y);  
    if( c.y > max.y ) d += (c.y - max.y)*(c.y - max.y);  
    return d < r*r;  
}
```

位于 `javagames.intersection` 包中的 `OverlapExample`，如图 7.13 所示，绘制了两个圆和两个 AABB。这些形状可以点击或拖拽。使用屏幕鼠标坐标到世界坐标的转换来做到这些，



就像前面测试以检查鼠标位置一样。

如果形状相交，将它们绘制为蓝色；否则，将它们绘制为黑色。本章前面介绍的绘制一个 AABB 和一个圆形的方法，用来绘制冲突形状，以便它们可以根据窗口的大小而缩放。给出的所有重叠测试，圆形和圆形之间的，AABB 和 AABB 之间的，圆形和 AABB 之间的，都可以用来测试任何形状是否与其他形状重叠。AABB 表示为一个最小值、一个最大值以及一个位置。圆形表示为一个圆心和一个半径。

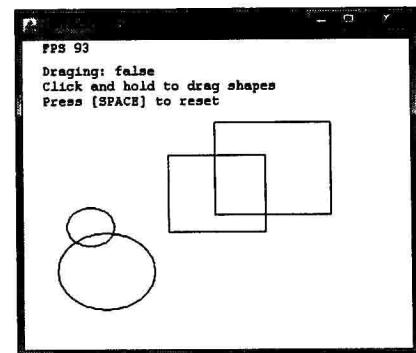


图 7.13 重叠的示例

```
package javagames.intersection;  
import java.awt.*;  
import java.awt.event.*;  
import javagames.util.*;  
  
public class OverlapExample extends SimpleFramework {  
    // mouse variables  
    private Vector2f mousePos;  
    private Vector2f mouseDelta;  
    private boolean clicked;  
    private boolean dragging;  
    // AABB variables  
    private Vector2f min0, max0;  
    private Vector2f min0Cpy, max0Cpy;  
    private Vector2f rect0Pos;  
    private boolean rect0Collision;  
    private boolean rect0Moving;  
    private Vector2f min1, max1;  
    private Vector2f min1Cpy, max1Cpy;  
    private Vector2f rect1Pos;  
    private boolean rect1Collision;  
    private boolean rect1Moving;  
    // circle variables  
    private Vector2f c0, c0Pos;  
    private float r0;  
    private boolean circle0Collision;  
    private boolean circle0Moving;  
    private Vector2f c1, c1Pos;  
    private float r1;  
    private boolean circle1Collision;  
    private boolean circle1Moving;
```



```
public OverlapExample() {
    appHeight = 640;
    appWidth = 640;
    appTitle = "Overlap Example";
    appBackgroundColor = Color.WHITE;
    appFPSColor = Color.BLACK;
}
protected void initialize() {
    super.initialize();
    mousePos = new Vector2f();
    min0 = new Vector2f( -0.25f, -0.25f );
    max0 = new Vector2f( 0.25f, 0.25f );
    min1 = new Vector2f( -0.3f, -0.3f );
    max1 = new Vector2f( 0.3f, 0.3f );
    r0 = 0.25f;
    r1 = 0.125f;
    reset();
}
private void reset() {
    rect0Pos = new Vector2f();
    rect1Pos = new Vector2f( 0.25f, 0.5f );
    c0Pos = new Vector2f( -0.60f, -0.60f );
    c1Pos = new Vector2f( 0.6f, 0.6f );
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    // reset objects on spacebar
    if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
        reset();
    }
    // convert screen coordinates to world coordinates
    // for intersection testing
    Vector2f pos = getWorldMousePosition();
    mouseDelta = pos.sub( mousePos );
    mousePos = pos;
    clicked = mouse.buttonDownOnce( MouseEvent.BUTTON1 );
    dragging = mouse.buttonDown( MouseEvent.BUTTON1 );
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    // calculate the AABB minimum and maximum values
```



```
Matrix3x3f mat = Matrix3x3f.translate(
    rect0Pos.x, rect0Pos.y
);
min0Cpy = mat.mul( min0 );
max0Cpy = mat.mul( max0 );
mat = Matrix3x3f.translate(
    rect1Pos.x, rect1Pos.y
);
min1Cpy = mat.mul( min1 );
max1Cpy = mat.mul( max1 );
// position the circles
mat = Matrix3x3f.translate(
    c0Pos.x, c0Pos.y
);
c0 = mat.mul( new Vector2f() );
mat = Matrix3x3f.translate(
    c1Pos.x, c1Pos.y
);
c1 = mat.mul( new Vector2f() );
// test for click and drag of objects
if( clicked && pointInAABB( mousePos, min0Cpy, max0Cpy ) ) {
    rect0Moving = true;
}
if( clicked && pointInAABB( mousePos, min1Cpy, max1Cpy ) ) {
    rect1Moving = true;
}
if( clicked && pointInCircle( mousePos, c0Pos, r0 ) ) {
    circle0Moving = true;
}
if( clicked && pointInCircle( mousePos, c1Pos, r1 ) ) {
    circle1Moving = true;
}
rect0Moving = rect0Moving && dragging;
if( rect0Moving ) {
    rect0Pos = rect0Pos.add( mouseDelta );
}
rect1Moving = rect1Moving && dragging;
if( rect1Moving ) {
    rect1Pos = rect1Pos.add( mouseDelta );
}
circle0Moving = circle0Moving && dragging;
if( circle0Moving ) {
    c0Pos = c0Pos.add( mouseDelta );
```



```
        }

        circle1Moving = circle1Moving && dragging;
        if( circle1Moving ) {
            c1Pos = c1Pos.add( mouseDelta );
        }

        rect0Collision = false;
        rect1Collision = false;
        circle0Collision = false;
        circle1Collision = false;
        // perform intersection testing
        if( intersectAABB( min0Cpy, max0Cpy, min1Cpy, max1Cpy ) ) {
            rect0Collision = true;
            rect1Collision = true;
        }
        if( intersectCircle( c0, r0, c1, r1 ) ) {
            circle0Collision = true;
            circle1Collision = true;
        }
        if( intersectCircleAABB( c0, r0, min0Cpy, max0Cpy ) ) {
            circle0Collision = true;
            rect0Collision = true;
        }
        if( intersectCircleAABB( c0, r0, min1Cpy, max1Cpy ) ) {
            circle0Collision = true;
            rect1Collision = true;
        }
        if( intersectCircleAABB( c1, r1, min0Cpy, max0Cpy ) ) {
            circle1Collision = true;
            rect0Collision = true;
        }
        if( intersectCircleAABB( c1, r1, min1Cpy, max1Cpy ) ) {
            circle1Collision = true;
            rect1Collision = true;
        }
    }

    private boolean pointInAABB( Vector2f p, Vector2f min, Vector2f max ) {
        return p.x > min.x && p.x < max.x &&
               p.y > min.y && p.y < max.y;
    }

    private boolean pointInCircle( Vector2f p, Vector2f c, float r ) {
        Vector2f dist = p.sub( c );
        return dist.lenSqr() < r*r;
    }
```



```
private boolean intersectAABB(
    Vector2f minA, Vector2f maxA, Vector2f minB, Vector2f maxB ) {
    if( minA.x > maxB.x || minB.x > maxA.x ) return false;
    if( minA.y > maxB.y || minB.y > maxA.y ) return false;
    return true;
}
private boolean intersectCircle(
    Vector2f c0, float r0, Vector2f c1, float r1 ) {
    Vector2f c = c0.sub( c1 );
    float r = r0 + r1;
    return c.lenSqr() < r*r;
}
private boolean intersectCircleAABB(
    Vector2f c, float r, Vector2f min, Vector2f max ) {
    float d = 0.0f;
    if( c.x < min.x ) d += (c.x - min.x)*(c.x - min.x);
    if( c.x > max.x ) d += (c.x - max.x)*(c.x - max.x);
    if( c.y < min.y ) d += (c.y - min.y)*(c.y - min.y);
    if( c.y > max.y ) d += (c.y - max.y)*(c.y - max.y);
    return d < r*r;
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    // render instructions
    g.drawString( "Draging: " + dragging, 20, 35 );
    g.drawString( "Click and hold to drag shapes", 20, 50 );
    g.drawString( "Press [SPACE] to reset", 20, 65 );
    // render objects
    g.setColor( rect0Collision ? Color.BLACK : Color.BLUE );
    drawAABB( g, min0Cpy, max0Cpy );
    g.setColor( rect1Collision ? Color.BLACK : Color.BLUE );
    drawAABB( g, min1Cpy, max1Cpy );
    g.setColor( circle0Collision ? Color.BLACK : Color.BLUE );
    drawOval( g, c0, r0 );
    g.setColor( circle1Collision ? Color.BLACK : Color.BLUE );
    drawOval( g, c1, r1 );
}
// draw the AABB
private void drawAABB( Graphics g, Vector2f min, Vector2f max ) {
    Matrix3x3f view = getViewportTransform();
    Vector2f topLeft = new Vector2f( min.x, max.y );
    topLeft = view.mul( topLeft );
```



```
Vector2f bottomRight = new Vector2f( max.x, min.y );
bottomRight = view.mul( bottomRight );
int rectX = (int)topLeft.x;
int rectY = (int)topLeft.y;
int rectWidth = (int)( bottomRight.x - topLeft.x );
int rectHeight = (int)( bottomRight.y - topLeft.y );
g.drawRect( rectX, rectY, rectWidth, rectHeight );
}
// draw the circle
private void drawOval( Graphics g, Vector2f center, float radius ) {
    Matrix3x3f view = getViewportTransform();
    Vector2f topLeft =
        new Vector2f( center.x - radius, center.y + radius );
    topLeft = view.mul( topLeft );
    Vector2f bottomRight =
        new Vector2f( center.x + radius, center.y - radius );
    bottomRight = view.mul( bottomRight );
    int circleX = (int)topLeft.x;
    int circleY = (int)topLeft.y;
    int circleWidth = (int)(bottomRight.x - topLeft.x);
    int circleHeight = (int)(bottomRight.y - topLeft.y);
    g.drawOval( circleX, circleY, circleWidth, circleHeight );
}
public static void main( String[] args ) {
    launchApp( new OverlapExample() );
}
}
```

## 7.4 使用分隔轴方法

使用分隔轴方法来测试两个物体是否重叠，这比前面的测试更为复杂，但是，它可以处理在任何位置旋转的任何凸面形状。尽管这种方法本身不能给出确切的冲突点，但是它对于测试物体是否重叠非常有用。

其思路很简单：如果在两个凸面多边形之间存在一个分隔的轴，那么，这两个物体没有重叠。换句话说，如果可以在两个图形之间画出一条直线，那么，它们没有重叠，如图 7.14 所示。

如果你搜索该理论的严格的数学证明，会在网上找到很多例子。为此，图 7.14 给出了

充分的证明。记住，这只对于凸面图形有效。如果有任何内角大于 180 度，该多边形就是凹面多边形，分隔轴测试将会无效。图 7.15 展示了一个由于凹面多边形而测试失败的例子。

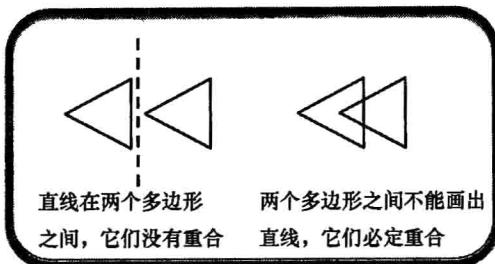


图 7.14 分隔轴方法

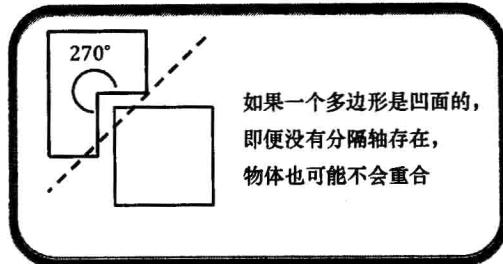


图 7.15 凹面多边形问题

如果你有一个凹面多边形，它总是可以分解为多个凸面多边形来进行测试，如图 7.16 所示。

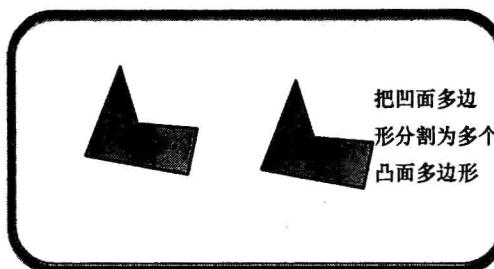


图 7.16 分割凹面多边形

很容易看到如何在两个物体之间画出一条直线，但是，这个测试需要在代码中用数学计算来实现。好在不仅一般的情况会很容易，还有一些优化可以应用于对称的形状。

要测试图形是否重叠，应按照如下步骤进行：对于两个图形中的每一个线段，创建一个垂直的正规化向量，并且使用点积将所有的点都投射到正规化向量上，如图 7.17 所示。

一旦点都投射到了该垂直正规化向量上，如果投射重叠了，那么，多边形是重叠的。如果投射不包含任何的重叠区域，那么，图形没有重叠，如图 7.18 所示。

此外，如下面的示例所示，如果物体是对称地进行投射，那么，可以将每个物体的半径与两个物体中心之间的距离进行比较。这个测试就像是测试两个圆的重叠。这个概念如图 7.19 所示。

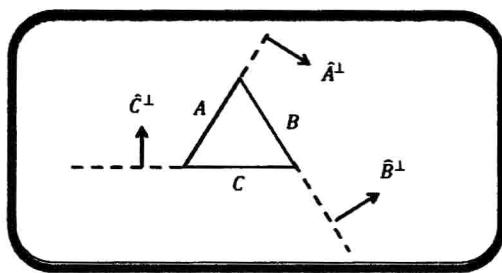


图 7.17 投射线段

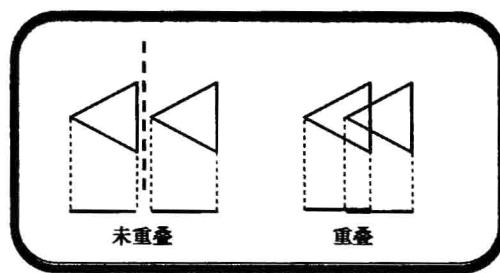


图 7.18 重叠的投射

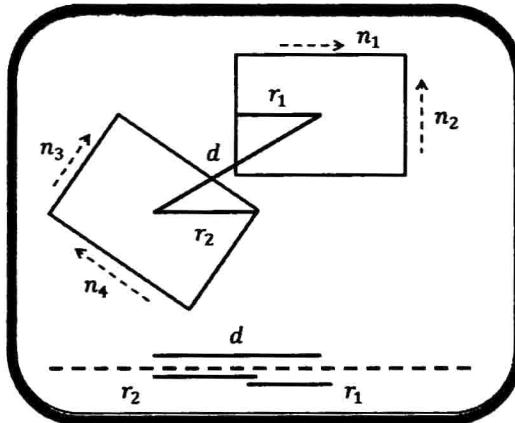


图 7.19 矩形重叠

上面的距离( $d$ )，以及( $r_1$ )与( $r_2$ )的加和，用来判断物体是否重叠。如果任何一个测试失败，那么，物体都没有重叠。假设物体不会彼此曲折吻合，如果物体之间不存在分隔轴，那么，它们是相交的。这个方法很好的一点是，它对于二维以上的情况也有效。在前面的图中，唯一没有重叠的轴是( $n_3$ )。

## 7.5 使用线段一线段的重叠方法

使用中点公式，可以找到每条线段的中心。

$$midpt = \frac{P_0 - P_1}{2}$$

每条线段的中心到其任何一个端点的距离，都是这条线段的半径。同样的半径，当正



规范化为长度 1 的时候，它就是用于投射的轴。由于每条线段的半径投射到自身之上的时候总是为 0，因此仅仅将其他线段的半径与投射的中心距离相比较即可，如图 7.20 所示。

*if  $|\hat{n}_1 \cdot (C_1 - C_2)| > |\hat{n}_1 \cdot r_2|$ : no overlap*

*if  $|\hat{n}_2 \cdot (C_1 - C_2)| > |\hat{n}_2 \cdot r_1|$ : no overlap*

由于该比较只是使用距离，而距离总是正值，因此，使用绝对值确保不会计算出负值来。

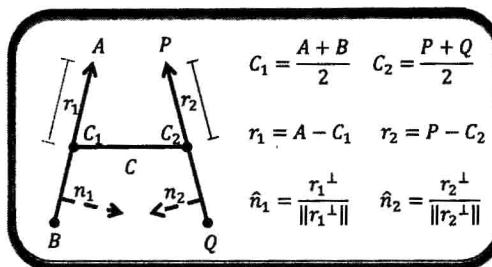


图 7.20 线段和线段的重叠

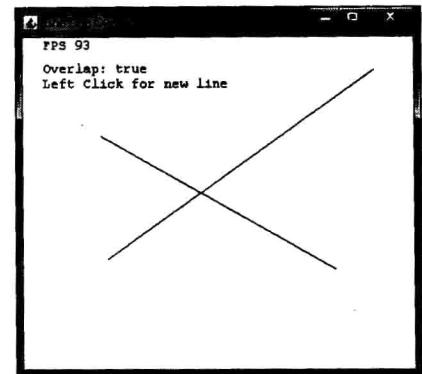


图 7.21 线段和线段之间重叠的示例

位于 `javagames.intersection` 的 `LineLineOverlapExample`，如图 7.21 所示，它存储了两条线段，P 和 Q，第一条线段为 `start`，第二条线段为 `end`。当点击鼠标左键的时候，设置了 `start` 点，`end` 点随着鼠标指针而确定。

在 `updateObject()` 方法中，我们使用分隔轴的方法检查了两条线段，看看它们是否重叠。使用这两条线段找到中点、半径以及每个正规化轴。然后，将中心点之间的投射距离与两条线段的半径比较。如果这个距离大于半径，那么，没有重叠。`render()` 方法绘制了两条线段。如果它们重叠，线段绘制为蓝色；否则，线段绘制为黑色。

```
package javagames.intersection;
import java.awt.*;
import java.awt.event.MouseEvent;
import javagames.util.*;

public class LineLineOverlapExample extends SimpleFramework {
    private Vector2f P, Q;
    private Vector2f start, end;
    boolean overlap = false;
    public LineLineOverlapExample() {
        appWidth = 640;
```



```
appHeight = 640;
appSleep = 10L;
appTitle = "Line Line Overlap";
appBackground = Color.WHITE;
appFPSColor = Color.BLACK;
}

@Override
protected void initialize() {
    super.initialize();
    P = new Vector2f( -0.6f, 0.4f );
    Q = new Vector2f( 0.6f, -0.4f );
    start = new Vector2f( 0.8f, 0.8f );
    end = new Vector2f();
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    end = getWorldMousePosition();
    if( mouse.buttonDownOnce( MouseEvent.BUTTON1 ) ) {
        start = new Vector2f( end );
    }
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    overlap = lineLineOverlap( P, Q, start, end );
}
private boolean lineLineOverlap(
    Vector2f A, Vector2f B, Vector2f P, Vector2f Q ) {
    Vector2f C0 = A.add( B ).div( 2.0f );
    Vector2f C1 = P.add( Q ).div( 2.0f );
    Vector2f C = C0.sub( C1 );
    Vector2f r0 = A.sub( C0 );
    Vector2f r1 = P.sub( C1 );
    Vector2f N0 = r0.perp().norm();
    Vector2f N1 = r1.perp().norm();
    float abs1 = Math.abs( N0.dot( C ) );
    float abs2 = Math.abs( N0.dot( r1 ) );
    if( abs1 > abs2 ) return false;
    abs1 = Math.abs( N1.dot( C ) );
    abs2 = Math.abs( N1.dot( r0 ) );
    if( abs1 > abs2 ) return false;
    return true;
}
```



```
}

@Override
protected void render( Graphics g ) {
    super.render( g );
    g.drawString( "Overlap: " + overlap, 20, 35 );
    g.drawString( "Left Click for new line", 20, 50 );
    g.setColor( overlap ? Color.BLUE : Color.BLACK );
    Matrix3x3f view = getViewportTransform();
    Vector2f v0 = view.mul( P );
    Vector2f v1 = view.mul( Q );
    g.drawLine( (int)v0.x, (int)v0.y, (int)v1.x, (int)v1.y );
    v0 = view.mul( start );
    v1 = view.mul( end );
    g.drawLine( (int)v0.x, (int)v0.y, (int)v1.x, (int)v1.y );
}
public static void main( String[] args ) {
    launchApp( new LineLineOverlapExample() );
}
}
```

## 7.6 使用矩形—矩形的重叠方法

要测试矩形之间的重叠，使用同样的方法。需要针对每个矩形计算中心、宽度和高度以及两个轴向量，如图 7.22 所示。

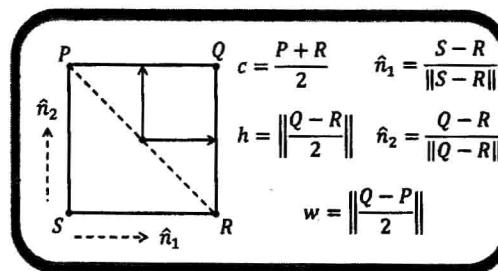


图 7.22 矩形重叠计算

要将半径投射到轴上，指向一角的单个向量不能使用，因为旋转会导致投射不正确，如图 7.23 所示。要投射正确的半径，必须总是分别投射宽度和高度。

RectRectOverlapExample 如图 7.24 所示, 它位于 javagames.intersection 包中, 保存了一个基本矩形的 4 个点, 示例中所测试的两个矩形都复制了这个矩形。每一次矩形重叠测试, 都有点、位置和旋转(弧度表示)的一个数组。相交变量保存了分隔轴测试的状态。

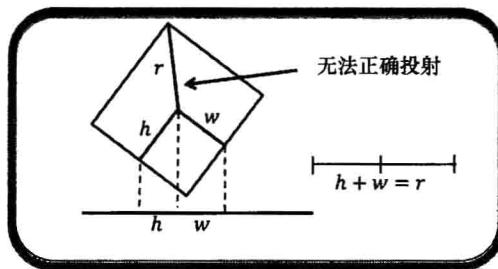


图 7.23 矩形半径投射

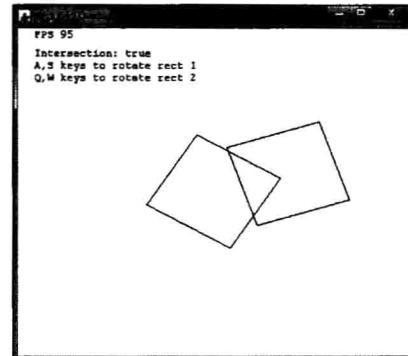


图 7.24 矩形—矩形重叠示例

processInput()方法使用鼠标位置移动第二个矩形。A 键和 S 键将旋转第一个矩形, Q 键和 W 键旋转第二个矩形。

updateObjects()方法复制每个矩形, 然后将其变换和旋转, 以便可以对其进行重叠测试。

rectRectIntersection()方法计算每个矩形的中心、4 个点的轴的值, 以及每个矩形的宽度和高度。宽度和高度的加和与每个中心的距离进行比较。如果它们都不重叠, 那么, 物体没有重叠。

render()方法将矩形转换到屏幕坐标, 并且如果重叠的话, 将其绘制为蓝色, 否则将其绘制为黑色。

```
package javagames.intersection;
import java.awt.*;
import java.awt.event.KeyEvent;
import javagames.util.*;

public class RectRectOverlapExample extends SimpleFramework {
    private Vector2f[] rect;
    private Vector2f[] rect0;
    private Vector2f rect0Pos;
    private float rect0Angle;
    private Vector2f[] rect1;
    private Vector2f rect1Pos;
    private float rect1Angle;
    private boolean intersection;
```



```
public RectRectOverlapExample() {
    appWidth = 640;
    appHeight = 640;
    appSleep = 10L;
    appTitle = "Rect Rect Overlap";
    appBackground = Color.WHITE;
    appFPSColor = Color.BLACK;
}
@Override
protected void initialize() {
    super.initialize();
    // set up rectangles for testing
    rect = new Vector2f[] {
        new Vector2f( -0.25f, 0.25f ),
        new Vector2f( 0.25f, 0.25f ),
        new Vector2f( 0.25f, -0.25f ),
        new Vector2f( -0.25f, -0.25f ),
    };
    rect0 = new Vector2f[ rect.length ];
    rect0Pos = new Vector2f();
    rect0Angle = 0.0f;
    rect1 = new Vector2f[ rect.length ];
    rect1Pos = new Vector2f();
    rect1Angle = 0.0f;
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    // convert mouse coordinate for testing
    rect1Pos = getWorldMousePosition();
    // rotate rectangles
    if( keyboard_KeyDown( KeyEvent.VK_A ) ) {
        rect0Angle += (float)( Math.PI / 4.0 * delta );
    }
    if( keyboard_KeyDown( KeyEvent.VK_S ) ) {
        rect0Angle -= (float)( Math.PI / 4.0 * delta );
    }
    if( keyboard_KeyDown( KeyEvent.VK_Q ) ) {
        rect1Angle += (float)( Math.PI / 4.0 * delta );
    }
    if( keyboard_KeyDown( KeyEvent.VK_W ) ) {
        rect1Angle -= (float)( Math.PI / 4.0 * delta );
    }
}
```



```
}

@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    // translate objects
    Matrix3x3f mat = Matrix3x3f.identity();
    mat = mat.mul( Matrix3x3f.rotate( rect0Angle ) );
    mat = mat.mul( Matrix3x3f.translate( rect0Pos ) );
    for( int i = 0; i < rect.length; ++i ) {
        rect0[i] = mat.mul( rect[i] );
    }
    mat = Matrix3x3f.identity();
    mat = mat.mul( Matrix3x3f.rotate( rect1Angle ) );
    mat = mat.mul( Matrix3x3f.translate( rect1Pos ) );
    for( int i = 0; i < rect.length; ++i ) {
        rect1[i] = mat.mul( rect[i] );
    }
    // test for intersection
    intersection = rectRectIntersection( rect0, rect1 );
}

private boolean rectRectIntersection( Vector2f[] A, Vector2f[] B ) {
    // separating axis intersection algorithm
    Vector2f N0 = A[0].sub( A[1] ).div( 2.0f );
    Vector2f N1 = A[1].sub( A[2] ).div( 2.0f );
    Vector2f CA = A[0].add( A[2] ).div( 2.0f );
    float D0 = N0.len();
    float D1 = N1.len();
    N1 = N1.div( D1 );
    N0 = N0.div( D0 );
    Vector2f N2 = B[0].sub( B[1] ).div( 2.0f );
    Vector2f N3 = B[1].sub( B[2] ).div( 2.0f );
    Vector2f CB = B[0].add( B[2] ).div( 2.0f );
    float D2 = N2.len();
    float D3 = N3.len();
    N2 = N2.div( D2 );
    N3 = N3.div( D3 );
    Vector2f C = CA.sub( CB );
    float DA = D0;
    float DB = D2 * Math.abs( N2.dot( N0 ) );
    DB += D3 * Math.abs( N3.dot( N0 ) );
    if( DA + DB < Math.abs( C.dot( N0 ) ) ) return false;
    DA = D1;
    DB = D2 * Math.abs( N2.dot( N1 ) );
```



```
DB += D3 * Math.abs( N3.dot( N1 ) );
if( DA + DB < Math.abs( C.dot( N1 ) ) ) return false;
DA = D2;
DB = D0 * Math.abs( N0.dot( N2 ) );
DB += D1 * Math.abs( N1.dot( N2 ) );
if( DA + DB < Math.abs( C.dot( N2 ) ) ) return false;
DA = D3;
DB = D0 * Math.abs( N0.dot( N3 ) );
DB += D1 * Math.abs( N1.dot( N3 ) );
if( DA + DB < Math.abs( C.dot( N3 ) ) ) return false;
return true;
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    // render instructions
    g.drawString( "Intersection: " + intersection, 20, 35 );
    g.drawString( "A,S keys to rotate rect 1", 20, 50 );
    g.drawString( "Q,W keys to rotate rect 2", 20, 65 );
    // draw rectangles
    g.setColor( intersection ? Color.BLUE : Color.BLACK );
    Matrix3x3f view = getViewportTransform();
    for( int i = 0; i < rect0.length; ++i ) {
        rect0[i] = view.mul( rect0[i] );
    }
    Utility.drawPolygon( g, rect0 );
    for( int i = 0; i < rect1.length; ++i ) {
        rect1[i] = view.mul( rect1[i] );
    }
    Utility.drawPolygon( g, rect1 );
}
public static void main( String[] args ) {
    launchApp( new RectRectOverlapExample() );
}
}
```

## 7.7 优化测试

总是查看图形以及测试，看看是否有办法简化数学计算，这一点很重要。例如，如果



使用了 AABB 形状，则有些信息可以直接编码到测试中。AABB 的正规化向量的长度总是为 1，并且总是在 x 和 y 轴上，如图 7.25 所示。

使用这些标准向量的任何点积，总是如下所示：

$$P * n_1 = n_x * (1) + P_y * (0) = P_x$$

$$P * n_2 = n_x * (0) + P_y * (1) = P_y$$

在这个例子中，甚至不需要计算标准向量，就可以使用 P 点的 x 和 y 值。像这样的优化可以简化代码，加速计算，并且防止引入 bug。总是要留意诸如此类的情况。

如果使用 AABB 矩形进行分隔轴测试，那么，垂直正规化向量就是 x 轴和 y 轴。这些正规化向量之间的点积以及矩形的边，简化为最小的和最大的(x,y)坐标，如图 7.26 所示。

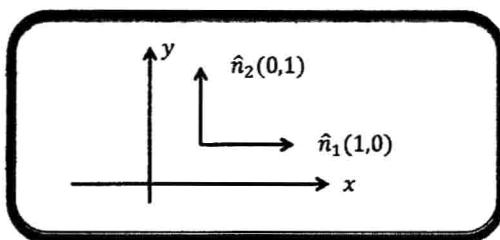


图 7.25 优化

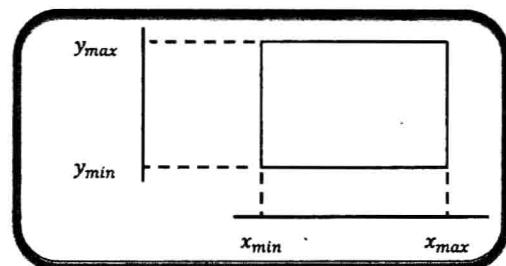


图 7.26 AABB 分隔轴

重叠矩形测试就是这样实现的。你甚至可以在不知道分隔轴测试的情况下就直接使用它。

## 7.8 资源和延伸阅读

Haines, Eric, "Point in Polygon Strategies," *Graphics Gems IV*, ed. Paul Heckbert, Academic Press, pp. 24–46, 1994.

Akenine-Moller, Tomas, and Eric Haines, *Real-Time Rendering*, 2<sup>nd</sup> Edition, A K Peters, pp. 598–599, 600, 2002.

Arvo, James, "A Simple Method for Box-Sphere Intersection Testing," *Graphic Gems*, ed. Andrew S. Glassner, Academic Press, pp. 335–339.

Ericson, Christer, *Real-Time Collision Detection*, Focal Press, p. 184, 2004.



# 第8章

## 游戏原型

本章基于你在前面各章中已经学到的所有内容，介绍如何创建一个原型太空游戏。尽管它还有很多元素缺失，因而不能称之为一款完整的游戏，但我们已经有了足够的可用工具来创建一个可以工作的原型。

在初次学习编程的时候，我遇到的一个问题是，有众多的示例使用各种编程概念，如循环、变量、集成和多态，但是，并没有太多的示例真正做某件事情。这些示例要么太简单了，只是无用的代码片段，要么是极其复杂、编写糟糕的程序，其中还掺入了所有的计算机科学理论，而不是只用到完成任务所需的那些知识。本章的这款原型游戏并不会试图涵盖所有内容，相反，它试图在良好的类设计和容易理解的代码之间取得平衡。

就像任何其他的游戏一样，前面的任何一章中都没有介绍首要的问题。当飞船从屏幕上飞走的时候，会发生什么事情？

### 8.1 创建一个多边形包装类

制作一款 2D 太空飞船游戏时，首先要解决的大问题是，当飞船到达屏幕边缘时，会发生什么事情。一个解决方案是，将飞船保持在屏幕的中央，而移动其周围的环境。没有对加载文件的任何支持，也没有关卡编辑器的话，要做到这点似乎有点难。另一个选择是，当飞船到达屏幕边界的时候，将其弹回。这样做似乎显得很奇怪。第三种选择是，让飞船从一端到另一端折返。让我们采用第三种方法。

还有很多方法可以解决这一问题。一种方法是，让飞船完全离开屏幕，然后让其在另一端返回，如图 8.1 所示。

尝试这一想法似乎很简单。我们使用围绕飞船的一个边界框来测试飞船何时离开了屏幕，并且将其移动到另外一端。这种方法有两个问题：一个问题是，可能会使得飞船在飞

行中变成只有很小的一块可见，从而很难碰到它；第二个问题是，飞船会持续地处在屏幕之外。这并不经常发生，但是，当多边形的大部分都在屏幕之外的时候，可能很难撞击到该物体。由于存在这些问题，因此与一直等待直到整个物体离开屏幕相比，折返物体似乎是更好的选择。

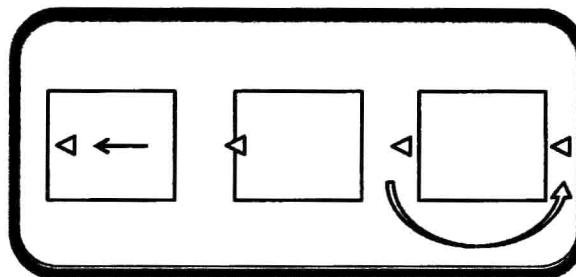


图 8.1 折返多边形

折返物体以使得离开屏幕一端的任何部分从屏幕的另一端返回，也需要担心风险。当你将一个物体从一端折返到另一端的时候，同一物体可能会有 4 个副本，如图 8.2 所示。

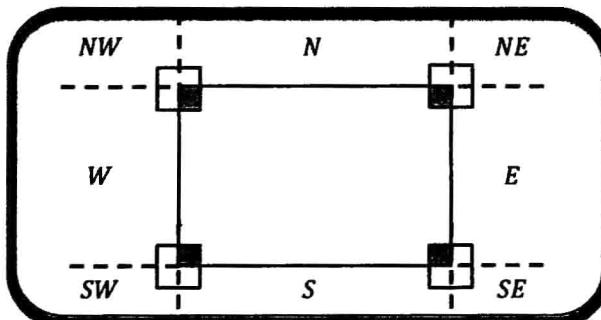


图 8.2 折返物体

这使得不仅需要绘制 4 个副本，而且所有 4 个副本都需要进行碰撞检查。如果只对第一个物体进行碰撞检查，位于其对角的两个物体可能就不会记录碰撞，即便它们的绘制物体有重叠，如图 8.3 所示。

创建一个渲染列表，其中包含了每个物体的副本，这可以解决该问题。物体可以绘制，并且整个列表可用来检查碰撞。物体折返的时候不会遇到问题，因为一旦物体的中心到了游戏世界的边界之外，就会通过游戏世界的宽度和高度来调整其位置以将其折返。应该总是将折返位置放置在边界之中，这样做物体就不会跑到屏幕之外，如图 8.4 所示。

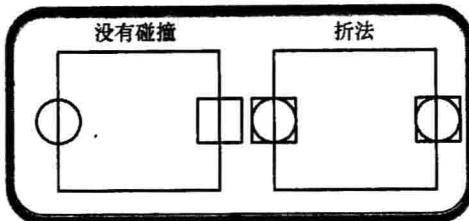


图 8.3 折返的副本

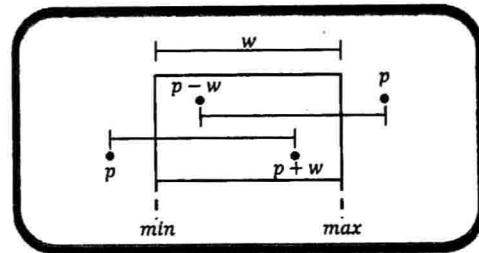


图 8.4 多边形迁移

```
if( p < min )
    p += w;
if( p > max )
    p -= w;
```

一旦物体的位置折返了，通过计算最小和最大( $x,y$ )值来得到物体的AABB，如图 8.5 所示。

边界框判断物体是否在屏幕的边界之外，以及是否需要将其折返到另一端。

位于 `javagames.prototype` 包中的 `PolygonWrapper` 类，包含了折返多边形及位置的方法。构造方法接受世界坐标表示的屏幕的宽度和高度，并且计算最小和最大向量值。

`hasLeftWorld()`方法根据最小和最大世界坐标来检查给定的位置。如果给定的位置在世界的边界之外，`wrapPosition()`方法返回一个折返位置。通过世界的宽度和高度来调整该折返位置，以便新的位置不会在世界的边界之外。要折返的多边形的边界，通过如下方式计算：

```
// PolygonWrapper.java
private Vector2f getMin( Vector2f[] poly ) {
    Vector2f min = new Vector2f( Float.MAX_VALUE, Float.MAX_VALUE );
    for( Vector2f v : poly ) {
        min.x = Math.min( v.x, min.x );
        min.y = Math.min( v.y, min.y );
    }
    return min;
}
private Vector2f getMax( Vector2f[] poly ) {
    Vector2f max = new Vector2f( -Float.MAX_VALUE, -Float.MAX_VALUE );
    for( Vector2f v : poly ) {
        max.x = Math.max( v.x, max.x );
        max.y = Math.max( v.y, max.y );
    }
}
```

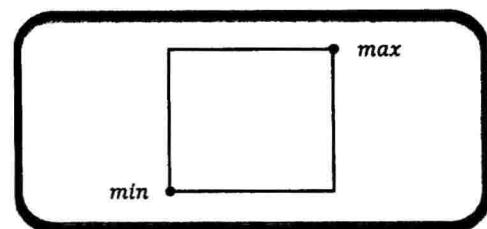


图 8.5 最小和最大 AABB



```
    }
    return max;
}
```

`wrapPolygon()`方法计算最小和最大值，并且使用这些值来判断物体是否需要向北、南、东或西折返。如果物体向两个主要方向折返，例如，向南和向东，那么，它也需要向北和向西折返。四个主要方向的每一个都要进行测试。使用变换方法，复制每个折返多边形，并且将其添加到渲染列表：

```
// PolygonWrapper.java
private Vector2f[] transform( Vector2f[] poly, Matrix3x3f mat ) {
    Vector2f[] copy = new Vector2f[ poly.length ];
    for( int i = 0; i < poly.length; ++i ) {
        copy[i] = mat.mul( poly[i] );
    }
    return copy;
}
```

通过加上或减去世界的宽度和高度，在8个方向上折返多边形，如图8.6所示。

`PolygonWrapper`代码如下所示：

```
package javagames.prototype;
import java.util.List;
import javagames.util.Matrix3x3f;
import javagames.util.Vector2f;

public class PolygonWrapper {
    private float worldWidth;
    private float worldHeight;
    private Vector2f worldMin;
    private Vector2f worldMax;
    public PolygonWrapper( float worldWidth, float worldHeight ) {
        this.worldWidth = worldWidth;
        this.worldHeight = worldHeight;
        worldMax = new Vector2f( worldWidth / 2.0f, worldHeight / 2.0f );
        worldMin = worldMax.inv();
    }
    public boolean hasLeftWorld( Vector2f position ) {
        return position.x < worldMin.x || position.x > worldMax.x ||
               position.y < worldMin.y || position.y > worldMax.y;
    }
    public Vector2f wrapPosition( Vector2f position ) {
```

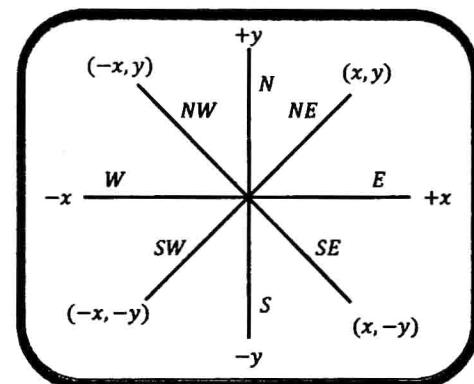


图8.6 指南针方向



章 原型

```
Vector2f wrapped = new Vector2f( position );
if( position.x < worldMin.x ) {
    wrapped.x = position.x + worldWidth;
} else if( position.x > worldMax.x ) {
    wrapped.x = position.x - worldWidth;
}
if( position.y < worldMin.y ) {
    wrapped.y = position.y + worldHeight;
} else if( position.y > worldMax.y ) {
    wrapped.y = position.y - worldHeight;
}
return wrapped;
}

public void wrapPolygon( Vector2f[] poly, List<Vector2f[]> renderList ) {
    Vector2f min = getMin( poly );
    Vector2f max = getMax( poly );
    boolean north = max.y > worldMax.y;
    boolean south = min.y < worldMin.y;
    boolean west = min.x < worldMin.x;
    boolean east = max.x > worldMax.x;
    if( west ) renderList.add( wrapEast( poly ) );
    if( east ) renderList.add( wrapWest( poly ) );
    if( north ) renderList.add( wrapSouth( poly ) );
    if( south ) renderList.add( wrapNorth( poly ) );
    if( north && west ) renderList.add( wrapSouthEast( poly ) );
    if( north && east ) renderList.add( wrapSouthWest( poly ) );
    if( south && west ) renderList.add( wrapNorthEast( poly ) );
    if( south && east ) renderList.add( wrapNorthWest( poly ) );
}

private Vector2f getMin( Vector2f[] poly ) {
    Vector2f min = new Vector2f( Float.MAX_VALUE, Float.MAX_VALUE );
    for( Vector2f v : poly ) {
        min.x = Math.min( v.x, min.x );
        min.y = Math.min( v.y, min.y );
    }
    return min;
}

private Vector2f getMax( Vector2f[] poly ) {
    Vector2f max = new Vector2f( -Float.MAX_VALUE, -Float.MAX_VALUE );
    for( Vector2f v : poly ) {
        max.x = Math.max( v.x, max.x );
        max.y = Math.max( v.y, max.y );
    }
}
```



```
    return max;
}

private Vector2f[] wrapNorth( Vector2f[] poly ) {
    return transform( poly, Matrix3x3f.translate( 0.0f, worldHeight ) );
}

private Vector2f[] wrapSouth( Vector2f[] poly ) {
    return transform( poly, Matrix3x3f.translate( 0.0f, -worldHeight ) );
}

private Vector2f[] wrapEast( Vector2f[] poly ) {
    return transform( poly, Matrix3x3f.translate( worldWidth, 0.0f ) );
}

private Vector2f[] wrapWest( Vector2f[] poly ) {
    return transform( poly, Matrix3x3f.translate( -worldWidth, 0.0f ) );
}

private Vector2f[] wrapNorthWest( Vector2f[] poly ) {
    return transform(
        poly, Matrix3x3f.translate( -worldWidth, worldHeight )
    );
}

private Vector2f[] wrapNorthEast( Vector2f[] poly ) {
    return transform(
        poly, Matrix3x3f.translate( worldWidth, worldHeight )
    );
}

private Vector2f[] wrapSouthEast( Vector2f[] poly ) {
    return transform(
        poly, Matrix3x3f.translate( worldWidth, -worldHeight )
    );
}

private Vector2f[] wrapSouthWest( Vector2f[] poly ) {
    return transform(
        poly, Matrix3x3f.translate( -worldWidth, -worldHeight )
    );
}

private Vector2f[] transform( Vector2f[] poly, Matrix3x3f mat ) {
    Vector2f[] copy = new Vector2f[ poly.length ];
    for( int i = 0; i < poly.length; ++i ) {
        copy[i] = mat.mul( poly[i] );
    }
    return copy;
}
}
```



位于 `javagames.prototype` 包中的 `ScreenWrapExample`，如图 8.7 所示，使用 `PolygonWrapper` 类来折返一个屏幕上来回移动的方块。`pos` 向量保存了方块的位置。`poly` 数组保存了方块多边形。`renderList` 保存了要折返的方块的副本，以及最初的变换多边形。折返变量是前面所介绍的 `PolygonWrapper`。

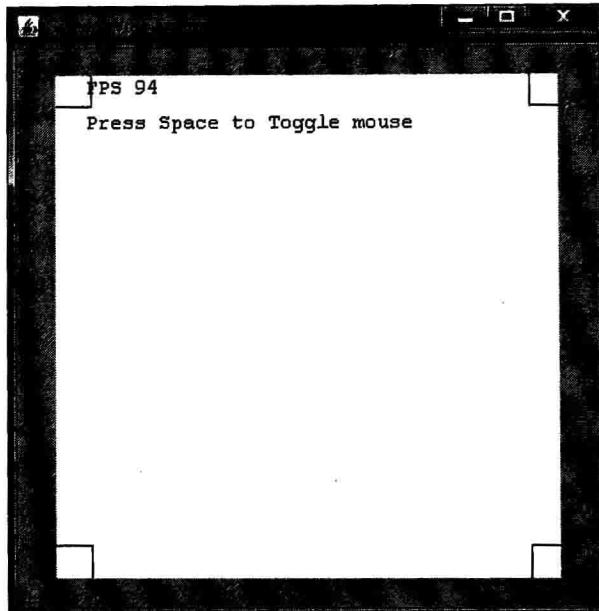


图 8.7 屏幕折返示例

`initialize()`方法创建了所有的物体并且将鼠标设置为相对移动，从而不需要鼠标离开屏幕物体就可以在屏幕上折返。如果使用绝对位置的话，`processInput()`方法使用鼠标位置来移动方块；如果使用相对移动的话，它通过将当前位置和相对移动相加来移动方块。该方法还使用空格键来切换相对鼠标移动。

`transform()`方法创建了对象的一个副本，并且通过给定的矩形来变换它。`updateObjects()`方法清理了渲染列表，将方块折返并变换，然后使用 `PolygonWrapper` 将物体的任何副本添加到渲染列表中。`render()`方法将渲染列表中的所有物体绘制到屏幕上。

```
package javagames.prototype;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import javagames.util.*;
```



```
public class ScreenWrapExample extends SimpleFramework {
    private Vector2f pos;
    private Vector2f[] poly;
    private ArrayList<Vector2f[]> renderList;
    private PolygonWrapper wrapper;
    public ScreenWrapExample() {
        appBorderScale = 0.9f;
        appWidth = 640;
        appHeight = 640;
        appMaintainRatio = true;
        appTitle = "Screen Wrap Example";
        appBackground = Color.WHITE;
        appFPSColor = Color.BLACK;
    }
    @Override
    protected void initialize() {
        super.initialize();
        mouse.setRelative( true );
        renderList = new ArrayList<Vector2f[]>();
        wrapper = new PolygonWrapper( appWorldWidth, appWorldHeight );
        poly = new Vector2f[] {
            new Vector2f( -0.125f, 0.125f ),
            new Vector2f( 0.125f, 0.125f ),
            new Vector2f( 0.125f, -0.125f ),
            new Vector2f( -0.125f, -0.125f ),
        };
        pos = new Vector2f();
    }
    @Override
    protected void processInput( float delta ) {
        super.processInput( delta );
        if( mouse.isRelative() ) {
            Vector2f v = getRelativeWorldMousePosition();
            pos = pos.add( v );
        } else {
            pos = getWorldMousePosition();
        }
        if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
            mouse.setRelative( !mouse.isRelative() );
            if( mouse.isRelative() ) {
                pos = new Vector2f();
            }
        }
    }
}
```



```
    }
    @Override
    protected void updateObjects( float delta ) {
        super.updateObjects( delta );
        renderList.clear();
        pos = wrapper.wrapPosition( pos );
        Vector2f[] world = transform( poly, Matrix3x3f.translate( pos ) );
        renderList.add( world );
        wrapper.wrapPolygon( world, renderList );
    }
    private Vector2f[] transform( Vector2f[] poly, Matrix3x3f mat ) {
        Vector2f[] copy = new Vector2f[ poly.length ];
        for( int i = 0; i < poly.length; ++i ) {
            copy[i] = mat.mul( poly[i] );
        }
        return copy;
    }
    @Override
    protected void render( Graphics g ) {
        super.render( g );
        g.drawString( "Press Space to Toggle mouse", 20, 35 );
        Matrix3x3f view = getViewportTransform();
        for( Vector2f[] toRender : renderList ) {
            for( int i = 0; i < toRender.length; ++i ) {
                toRender[i] = view.mul( toRender[i] );
            }
            Utility.drawPolygon( g, toRender );
        }
    }
    public static void main( String[] args ) {
        launchApp( new ScreenWrapExample() );
    }
}
```

## 8.2 创建一个原型小行星

PrototypeAsteroid 类位于 javagames.prototype 包中，它表示一个穿越太空的陨石。在创建的时候，使用了一个随机的速率和旋转。Java 的随机数生成器只能返回 0 到 1 之间的浮



点数，因此，要创建在任意范围内分布的随机数，需要一些额外的步骤。例如，要返回-3 到 7 之间的随机数，应按照如下步骤进行。

1. 用最大值减去最小值，计算随机数之间的差距。
2. 生成从 0 到 1 的一个随机浮点数。
3. 将随机数乘以差距值。
4. 通过加上最小值来迁移范围。

这些步骤听起来有些令人混淆，实际上并非如此。

```
private float getRandomFloat( float min, float max ) {  
    float rand = new Random().nextFloat();  
    return rand * (max - min) + min;  
}
```

要得到一个随机的整数，也可以采用相同的步骤。由于随机数生成器返回范围从 0 到计数值减去 1 的一个数字，因此该方法需要略作调整。

```
private float getRandomRadians( int minDegree, int maxDegree ) {  
    int rand = new Random().nextInt( maxDegree - minDegree + 1 );  
    return (float)Math.toRadians( rand + minDegree );  
}
```

getRandomRotationDelta()方法返回了 (5,45) 到 (-5,-45) 之间的一个角度（弧度表示），如图 8.8 所示。

```
private float getRandomRotationDelta() {  
    float radians = getRandomRadians( 5, 45 );  
    return new Random().nextBoolean() ? radians :  
-radians;  
}
```

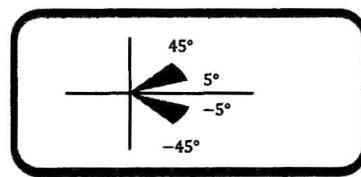


图 8.8 随机旋转增量

还可以使用 **setter** 和 **getter** 方法来访问小行星的某些属性。

```
public void setPolygon( Vector2f[] polygon );  
public void setPosition( Vector2f position );  
public Vector2f getPosition();  
public void setSize( PrototypeAsteroid.Size size );  
public PrototypeAsteroid.Size getSize()
```

大小是一个枚举类型的值：

```
public class PrototypeAsteroid {  
    public enum Size {  
        Large,
```



```
    Medium,  
    Small;  
}  
//...  
}
```

update()方法负责调整小行星的位置和旋转。注意，除了调整位置和旋转，**PolygonWrapper**类用来折返多边形的位置。

draw()方法负责用给定的视口矩阵和Graphics对象绘制多边形。要绘制填充的多边形，需要给Utility类添加两个方法：

```
package javagames.util;  
import java.awt.*;  
import java.util.List;  
  
public class Utility {  
    // ... Other methods left out  
    // ... New methods are below  
    public static void fillPolygon( Graphics2D g, Vector2f[] polygon ) {  
        Polygon p = new Polygon();  
        for( Vector2f v : polygon ) {  
            p.addPoint( (int)v.x, (int)v.y );  
        }  
        g.fill( p );  
    }  
    public static void fillPolygon( Graphics2D g, List<Vector2f> polygon ) {  
        Polygon p = new Polygon();  
        for( Vector2f v : polygon ) {  
            p.addPoint( (int)v.x, (int)v.y );  
        }  
        g.fill( p );  
    }  
}
```

pointInPolygon()方法和前面所讨论的方法相同。contains()方法接受一个点，如果**PolygonWrapper**所复制的任何多边形包含该点的话，它返回true。注意，**PrototypeAsteroid**中没有实际的模型代码。那是编辑器和工厂的工作，我们将在后面介绍。

```
package javagames.prototype;  
import java.awt.*;  
import java.util.*;  
import javagames.util.*;
```



```
public class PrototypeAsteroid {  
    public enum Size {  
        Large,  
        Medium,  
        Small;  
    }  
    private PolygonWrapper wrapper;  
    private Size size;  
    private float rotation;  
    private float rotationDelta;  
    private Vector2f[] polygon;  
    private Vector2f position;  
    private Vector2f velocity;  
    private ArrayList<Vector2f[]> renderList;  
    public PrototypeAsteroid( PolygonWrapper wrapper ) {  
        this.wrapper = wrapper;  
        renderList = new ArrayList<Vector2f[]>();  
        velocity = getRandomVelocity();  
        rotationDelta = getRandomRotationDelta();  
    }  
    private Vector2f getRandomVelocity() {  
        float angle = getRandomRadians( 0, 360 );  
        float radius = getRandomFloat( 0.06f, 0.3f );  
        return Vector2f.polar( angle, radius );  
    }  
    private float getRandomRadians( int minDegree, int maxDegree ) {  
        int rand = new Random().nextInt( maxDegree - minDegree + 1 );  
        return (float)Math.toRadians( rand + minDegree );  
    }  
    private float getRandomRotationDelta() {  
        float radians = getRandomRadians( 5, 45 );  
        return new Random().nextBoolean() ? radians : -radians;  
    }  
    private float getRandomFloat( float min, float max ) {  
        float rand = new Random().nextFloat();  
        return rand * (max - min) + min;  
    }  
    public void setPolygon( Vector2f[] polygon ) {  
        this.polygon = polygon;  
    }  
    public void setPosition( Vector2f position ) {  
        this.position = position;  
    }  
}
```



```
public Vector2f getPosition() {
    return position;
}
public void setSize( Size size ) {
    this.size = size;
}
public Size getSize() {
    return size;
}
public void update( float time ) {
    position = position.add( velocity.mul( time ) );
    position = wrapper.wrapPosition( position );

    rotation += rotationDelta * time;
    renderList.clear();
    Vector2f[] world = transformPolygon();
    renderList.add( world );
    wrapper.wrapPolygon( world, renderList );
}
private Vector2f[] transformPolygon() {
    Matrix3x3f mat = Matrix3x3f.rotate( rotation );
    mat = mat.mul( Matrix3x3f.translate( position ) );
    return transform( polygon, mat );
}
private Vector2f[] transform( Vector2f[] poly, Matrix3x3f mat ) {
    Vector2f[] copy = new Vector2f[ poly.length ];
    for( int i = 0; i < poly.length; ++i ) {
        copy[i] = mat.mul( poly[i] );
    }
    return copy;
}
public void draw( Graphics2D g, Matrix3x3f view ) {
    for( Vector2f[] poly : renderList ) {
        for( int i = 0; i < poly.length; ++i ) {
            poly[i] = view.mul( poly[i] );
        }
        g.setColor( Color.LIGHT_GRAY );
        Utility.fillPolygon( g, poly );
        g.setColor( Color.BLACK );
        Utility.drawPolygon( g, poly );
    }
}
public boolean contains( Vector2f point ) {
```



```
for( Vector2f[] polygon : renderList ) {
    if( pointInPolygon( point, polygon ) ) {
        return true;
    }
}
return false;
}

private boolean pointInPolygon( Vector2f point, Vector2f[] polygon ) {
    boolean inside = false;
    Vector2f start = polygon[ polygon.length - 1 ];
    boolean startAbove = start.y >= point.y;
    for( int i = 0; i < polygon.length; ++i ) {
        Vector2f end = polygon[i];
        boolean endAbove = end.y >= point.y;
        if( startAbove != endAbove ) {
            float m = (end.y - start.y) / (end.x - start.x);
            float x = start.x + (point.y - start.y) / m;
            if( x >= point.x ) {
                inside = !inside;
            }
        }
        startAbove = endAbove;
        start = end;
    }
    return inside;
}
}
```

## 8.3 创建一个原型编辑器

现在是时候创建一些多边形了。尽管有可能可以猜到每个点，但要创建 9 个小行星形状，手动进行的话，工作量还是很大的。就像大多数程序一样，我也很懒。编写一个编辑器来创建多边形，这样会容易很多。位于 `javagames.prototype` 包中的 `PrototypeEditor`，如图 8.9 所示。

如果模型存储为文件，它们将在运行时加载。然而，做这些事情所需的代码还没有介绍过，因此，编辑器将会作弊。当按下空格键的时候，编辑器会产生出能够粘贴到其他源文件中的代码。本书的第二部分将会介绍使用真正的文件，因此，你不必为此而担心。



`polygon` 列表保存了模型的点。`mousePos` 是鼠标的当前位置，布尔变量 `closed` 所保存的值，确定了在绘制时多边形的最后一个点是否和第一个点相连。

`processInput()`方法处理编辑器的交互。当按下鼠标左键时，会给多边形添加一个点。鼠标中键用于切换 `closed` 变量。鼠标右键从多边形删除点。C 键清除整个多边形，空格键打印出多边形代码。`printPolygon()` 方法向控制台打印出当前的多边形，以便可以将其粘贴到一个文本文件中或复制到其他的代码文件中。打印到调试控制台的输出如下所示：

```
Vector2f[] v = new Vector2f[] {  
    new Vector2f(-0.2895149f, 0.3865415f),  
    new Vector2f(-0.47104853f, -0.28012514f),  
    new Vector2f(-0.14241004f, 0.15805948f),  
    new Vector2f(-0.13615024f, -0.30516434f),  
    new Vector2f(0.28012514f, -0.2832551f),  
    new Vector2f(0.17370892f, 0.21752739f),  
    new Vector2f(-0.19248825f, 0.40219092f),  
    new Vector2f(-0.18622851f, 0.25195616f),  
};
```

`render()`方法显示常用的指令，并且绘制多边形以及轴线。`drawAxisLines()`方法在编辑器中绘制两条线，表示 x 轴和 y 轴。

`drawPolygon()`方法负责渲染当前模型。如果多边形只是一个点，那么，绘制单个的点。接下来，如果多边形有多个点，那么所有的点（除了最后一个点以外）都绘制。如果多边形是闭合的，那么，最后一个点连接到第一个点。如果多边形不是闭合的，那么，从最后一个点到当前鼠标位置绘制一条线。`drawPoint()`和 `drawLine()`方法都是标准版本。

```
package javagames.prototype;  
import java.awt.*;  
import java.awt.event.*;  
import java.util.ArrayList;  
import javagames.util.*;  
  
public class PrototypeEditor extends SimpleFramework {  
    private ArrayList<Vector2f> polygon;  
    private Vector2f mousePos;  
    private boolean closed;
```

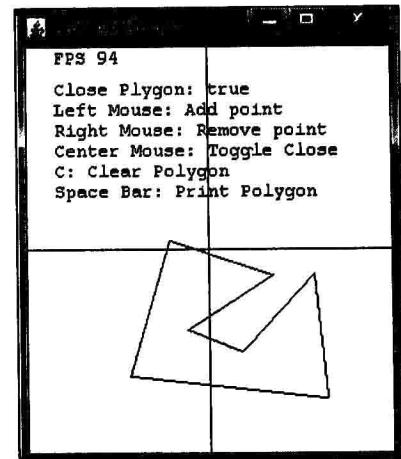


图 8.9 原型编辑器



```
public PrototypeEditor() {
    appTitle = "Prototype Editor 1.0";
    appWidth = 640;
    appHeight = 640;
    appBackground = Color.WHITE;
    appFPSColor = Color.BLACK;
}
protected void initialize() {
    super.initialize();
    polygon = new ArrayList<Vector2f>();
}
protected void processInput( float delta ) {
    super.processInput( delta );
    mousePos = getWorldMousePosition();
    if( mouse.buttonDownOnce( MouseEvent.BUTTON1 ) ) {
        polygon.add( mousePos );
    }
    if( mouse.buttonDownOnce( MouseEvent.BUTTON2 ) ) {
        closed = !closed;
    }
    if( mouse.buttonDownOnce( MouseEvent.BUTTON3 ) ) {
        if( !polygon.isEmpty() ) {
            polygon.remove( polygon.size() - 1 );
        }
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_C ) ) {
        polygon.clear();
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
        printPolygon();
    }
}
private void printPolygon() {
    System.out.println( "Vector2f[] v = new Vector2f[] { " );
    for( Vector2f v : polygon ) {
        System.out.print( " new Vector2f(" );
        System.out.print( v.x + "f, " );
        System.out.print( v.y + "f)" );
        System.out.println( "," );
    }
    System.out.println( ");" );
}
protected void render( Graphics g ) {
```



```
super.render( g );
g.drawString( "Close Polygon: " + closed, 20, 35 );
g.drawString( "Left Mouse: Add point", 20, 50 );
g.drawString( "Right Mouse: Remove point", 20, 65 );
g.drawString( "Center Mouse: Toggle Close", 20, 80 );
g.drawString( "C: Clear Polygon", 20, 95 );
g.drawString( "Space Bar: Print Polygon", 20, 110 );

drawAxisLines( g );
drawPolygon( g );
}

private void drawAxisLines( Graphics g ) {
    g.setColor( Color.BLUE );
    Vector2f left = new Vector2f( appWorldWidth / 2.0f, 0.0f );
    Vector2f right = new Vector2f( -left.x, 0.0f );
    drawLine( g, left, right );
    Vector2f top = new Vector2f( 0.0f, appWorldHeight / 2.0f );
    Vector2f bottom = new Vector2f( 0.0f, -top.y );
    drawLine( g, top, bottom );
}

private void drawPolygon( Graphics g ) {
    g.setColor( Color.BLACK );
    if( polygon.size() == 1 ) {
        drawPoint( g, polygon.get( 0 ) );
    }
    for( int i = 0; i < polygon.size() - 1; ++i ) {
        drawLine( g, polygon.get(i), polygon.get( i+1 ) );
    }
    if( closed && polygon.size() > 1 ) {
        Vector2f P = polygon.get( polygon.size() - 1 );
        Vector2f S = polygon.get( 0 );
        drawLine( g, S, P );
    }
    if( !(polygon.isEmpty() || closed) ) {
        Vector2f P = polygon.get( polygon.size() - 1 );
        Vector2f S = mousePos;
        drawLine( g, S, P );
    }
}
private void drawPoint( Graphics g, Vector2f v ) {
    Matrix3x3f view = getViewportTransform();
    Vector2f s = view.mul( v );
    g.drawRect( (int)s.x, (int)s.y, 1, 1 );
```



```

    }
    private void drawLine( Graphics g, Vector2f v0, Vector2f v1 ) {
        Matrix3x3f view = getViewportTransform();
        Vector2f S = view.mul( v0 );
        Vector2f P = view.mul( v1 );
        g.drawLine( (int)S.x, (int)S.y, (int)P.x, (int)P.y );
    }
    public static void main( String[] args ) {
        launchApp( new PrototypeEditor() );
    }
}
}

```

## 8.4 用原型小行星工厂生产小行星

位于 `javagames.prototype` 包中的 `PrototypeAsteroidFactory`, 包含了 3 个大的、3 个中型的和 3 个小的小行星, 它们都是使用编辑器并粘贴到代码中设计的。给定一个位置, 这 3 个方法生成一个随机数来选择所返回的小行星:

```

createLargeAsteroid( Vector2f position )
createMediumAsteroid( Vector2f position )
createSmallAsteroid( Vector2f position )

```

第一个方法 `mirror()`, 用来沿着 `x` 轴和 `y` 轴翻转模型, 以便模型有多个独特的版本。

```

// PrototypeAsteroidFactory.java
private Vector2f[] mirror( Vector2f[] polygon ) {
    Vector2f[] mirror = new Vector2f[ polygon.length ];
    float x = rand.nextBoolean() ? 1.0f : -1.0f;
    float y = rand.nextBoolean() ? 1.0f : -1.0f;
    Matrix3x3f mat = Matrix3x3f.scale( x, y );
    for( int i = 0; i < polygon.length; ++i ) {
        mirror[i] = mat.mul( polygon[i] );
    }
    return mirror;
}

```

就像在执行视口矩阵乘法的时候使用矩阵来翻转 `y` 轴一样, 通过乘以-1 来翻转 `x` 或 `y` 轴的值, 以缩放模型。

```

package javagames.prototype;
import java.util.Random;

```



```
import javagames.prototype.PrototypeAsteroid.Size;
import javagames.util.Matrix3x3f;
import javagames.util.Vector2f;

public class PrototypeAsteroidFactory {
    private static final Vector2f[][] LARGE = {
        { // Large 0
            new Vector2f(-0.029733956f, 0.283255100f),
            new Vector2f(-0.183098610f, 0.111111104f),
            new Vector2f(-0.230046930f,-0.057902932f),
            new Vector2f(-0.092331770f,-0.139280080f),
            new Vector2f( 0.117370844f,-0.142410040f),
            new Vector2f( 0.161189320f,-0.048513293f),
            new Vector2f( 0.151799680f, 0.067292630f),
            new Vector2f( 0.195618150f, 0.129890440f),
            new Vector2f( 0.017214417f, 0.158059480f),
        }, { // Large 1
            new Vector2f(-0.001763641f, 0.325800420f),
            new Vector2f(-0.082892360f, 0.220339000f),
            new Vector2f(-0.227513200f, 0.065913380f),
            new Vector2f(-0.206349200f,-0.141242860f),
            new Vector2f(-0.061728360f,-0.080979230f),
            new Vector2f( 0.061728418f,-0.167608260f),
            new Vector2f( 0.192239940f,-0.092278720f),
            new Vector2f( 0.167548480f, 0.126177010f),
            new Vector2f( 0.107583820f, 0.269303200f),
        }, { // Large 2
            new Vector2f( 0.176838760f,-0.107981205f),
            new Vector2f(-0.070422530f,-0.076682330f),
            new Vector2f(-0.220657290f,-0.123630640f),
            new Vector2f(-0.273865400f, 0.048513293f),
            new Vector2f(-0.186228510f, 0.086071970f),
            new Vector2f(-0.214397490f, 0.223787190f),
            new Vector2f(-0.026604056f, 0.148669780f),
            new Vector2f( 0.104851365f, 0.220657290f),
            new Vector2f( 0.211267590f, 0.032863855f),
        },
    };
    private static final Vector2f[][] MEDIUM = {
        { // Medium 0
            new Vector2f(-0.045383394f, 0.186228510f),
            new Vector2f(-0.167449180f, 0.123630700f),
            new Vector2f(-0.067292630f, 0.039123654f),
        },
    };
}
```



```
    new Vector2f(-0.107981205f, -0.073552370f),
    new Vector2f( 0.057902932f, -0.073552370f),
    new Vector2f( 0.133020280f,  0.098591566f),
}, { // Medium 1
    new Vector2f(-0.023474216f,  0.189358350f),
    new Vector2f(-0.107981205f,  0.107981205f),
    new Vector2f(-0.129890440f, -0.098591566f),
    new Vector2f( 0.020344257f, -0.120500800f),
    new Vector2f( 0.139280080f, -0.001564979f),
    new Vector2f( 0.076682330f,  0.092331770f),
    new Vector2f(-0.007824719f,  0.095461670f),
}, { // Medium 2
    new Vector2f(-0.064162790f,  0.158059480f),
    new Vector2f(-0.173708920f,  0.126760600f),
    new Vector2f(-0.142410040f,  0.023474216f),
    new Vector2f(-0.039123654f,  0.029733956f),
    new Vector2f( 0.010954618f, -0.035993695f),
    new Vector2f( 0.117370844f,  0.023474216f),
    new Vector2f( 0.117370844f,  0.120500800f),
    new Vector2f(-0.001564979f,  0.092331770f),
},
};

private static final Vector2f[][] SMALL = {
    { // Small 0
        new Vector2f(-0.048513293f,  0.057902990f),
        new Vector2f(-0.073552430f, -0.042253494f),
        new Vector2f( 0.004694819f, -0.035993695f),
        new Vector2f( 0.042253494f,  0.026604056f),
        new Vector2f(-0.001564979f,  0.082942130f),
    }, { // Small 1
        new Vector2f( 0.067292690f,  0.007824719f),
        new Vector2f(-0.029733956f, -0.076682330f),
        new Vector2f(-0.067292630f, -0.042253494f),
        new Vector2f(-0.061032890f,  0.082942130f),
        new Vector2f( 0.032863855f,  0.111111104f),
    }, { // Small 2
        new Vector2f(-0.007824719f,  0.089201870f),
        new Vector2f(-0.114241004f,  0.001564979f),
        new Vector2f(-0.004694819f, -0.067292690f),
        new Vector2f( 0.039123654f, -0.039123654f),
        new Vector2f(-0.014084518f,  0.020344317f),
    },
};
```



```
private PolygonWrapper wrapper;
private Random rand;
public PrototypeAsteroidFactory( PolygonWrapper wrapper ) {
    this.wrapper = wrapper;
    this.rand = new Random();
}

public PrototypeAsteroid createLargeAsteroid( Vector2f position ) {
    PrototypeAsteroid asteroid = new PrototypeAsteroid( wrapper );
    asteroid.setPosition( position );
    asteroid.setPolygon( getRandomAsteroid( LARGE ) );
    asteroid.setSize( Size.Large );
    return asteroid;
}

public PrototypeAsteroid createMediumAsteroid( Vector2f position ) {
    PrototypeAsteroid asteroid = new PrototypeAsteroid( wrapper );
    asteroid.setPosition( position );
    asteroid.setPolygon( getRandomAsteroid( MEDIUM ) );
    asteroid.setSize( Size.Medium );
    return asteroid;
}

public PrototypeAsteroid createSmallAsteroid( Vector2f position ) {
    PrototypeAsteroid asteroid = new PrototypeAsteroid( wrapper );
    asteroid.setPosition( position );
    asteroid.setPolygon( getRandomAsteroid( SMALL ) );
    asteroid.setSize( Size.Small );
    return asteroid;
}

private Vector2f[] getRandomAsteroid( Vector2f[][][] asteroids ) {
    return mirror( asteroids[ rand.nextInt( asteroids.length ) ] );
}

private Vector2f[] mirror( Vector2f[] polygon ) {
    Vector2f[] mirror = new Vector2f[ polygon.length ];
    float x = rand.nextBoolean() ? 1.0f : -1.0f;
    float y = rand.nextBoolean() ? 1.0f : -1.0f;
    Matrix3x3f mat = Matrix3x3f.scale( x, y );
    for( int i = 0; i < polygon.length; ++i ) {
        mirror[i] = mat.mul( polygon[i] );
    }
    return mirror;
}
```

RandomAsteroidExample 位于 javagames.prototype 包中，如图 8.10 所示，它使用 PolygonWrapper、PrototypeAsteroid 和 PrototypeAsteroid Factory，来生成在屏幕上随机飞来飞去的陨石。

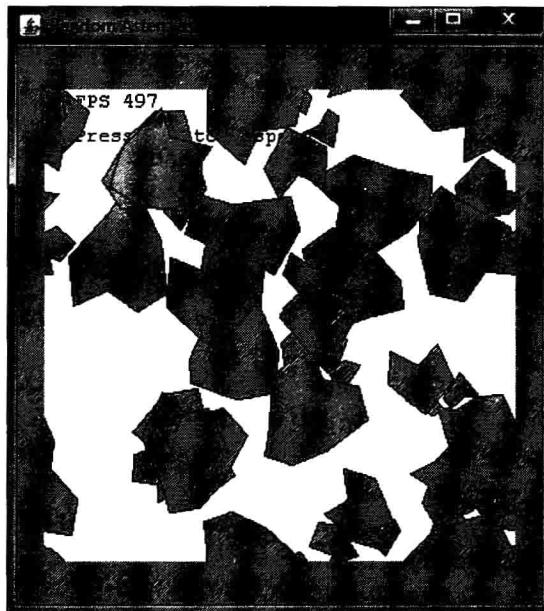


图 8.10 随机的小行星示例

getRandomAsteroid()方法创建了一个随机的位置，然后选择一个随机的大小：small、medium 或 large。记住，工厂的每一个方法都从 3 种不同的多边形模型中针对每个大小选取一个模型，并且为每个模型生成 4 个镜像的版本。每个多边形有 4 个镜像，每个大小有 3 个形状，共有 3 种不同的大小，因此，共有 36 种不同的模型。

还应该注意，由于 PrototypeAsteroid 使用了新的 Utility.fillPolygon()方法，Graphics 对象强制转型为一个 Graphics2D 对象。

```
// RandomAsteroidExample.java
protected void render( Graphics g ) {
    super.render( g );
    g.drawString( "Press ESC to respawn", 20, 35 );
    Matrix3x3f view = getViewportTransform();
    for( PrototypeAsteroid asteroid : asteroids ) {
        asteroid.draw( (Graphics2D)g, view );
    }
}
```



RandomAsteroidExample 代码如下所示：

```
package javagames.prototype;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.*;
import javagames.prototype.PrototypeAsteroid.Size;
import javagames.util.*;

public class RandomAsteroidExample extends SimpleFramework {
    private PrototypeAsteroidFactory factory;
    private ArrayList<PrototypeAsteroid> asteroids;
    private Random rand;
    public RandomAsteroidExample() {
        appBorderScale = 0.9f;
        appWidth = 640;
        appHeight = 640;
        appMaintainRatio = true;
        appSleep = 1L;
        appTitle = "Random Asteroids";
        appBackground = Color.WHITE;
        appFPSColor = Color.BLACK;
    }
    @Override
    protected void initialize() {
        super.initialize();
        rand = new Random();
        asteroids = new ArrayList<PrototypeAsteroid>();
        PolygonWrapper wrapper =
            new PolygonWrapper( appWorldWidth, appWorldHeight );
        factory = new PrototypeAsteroidFactory( wrapper );
        createAsteroids();
    }
    private void createAsteroids() {
        asteroids.clear();
        for( int i = 0; i < 42; ++i ) {
            asteroids.add( getRandomAsteroid() );
        }
    }
    private PrototypeAsteroid getRandomAsteroid() {
        float x = rand.nextFloat() * 2.0f - 1.0f;
        float y = rand.nextFloat() * 2.0f - 1.0f;
        Vector2f position = new Vector2f( x, y );
    }
}
```



```
Size[] sizes = Size.values();
Size randomSize = sizes[ rand.nextInt( sizes.length ) ];
switch( randomSize ) {
    case Small: return factory.createSmallAsteroid( position );
    case Medium: return factory.createMediumAsteroid( position );
    case Large:
        default: return factory.createLargeAsteroid( position );
    }
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    if( keyboard_KeyDownOnce( KeyEvent.VK_ESCAPE ) ) {
        createAsteroids();
    }
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    for( PrototypeAsteroid asteroid : asteroids ) {
        asteroid.update( delta );
    }
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    g.drawString( "Press ESC to respawn", 20, 35 );
    Matrix3x3f view = getViewportTransform();
    for( PrototypeAsteroid asteroid : asteroids ) {
        asteroid.draw( (Graphics2D)g, view );
    }
}
public static void main( String[] args ) {
    launchApp( new RandomAsteroidExample() );
}
}
```

## 8.5 原型 Bullet 类

PrototypeBullet 代码位于 javagames.prototype 包中，这是一个最简单的原型游戏源代



码。除了绘制圆以便可以调整屏幕大小外，没有其他任何值得讨论的内容。

```
package javagames.prototype;  
import java.awt.*;  
import javagames.util.*;  
  
public class PrototypeBullet {  
    private Vector2f velocity;  
    private Vector2f position;  
    private Color color;  
    private float radius;  
    public PrototypeBullet( Vector2f position, float angle ) {  
        this.position = position;  
        velocity = Vector2f.polar( angle, 1.0f );  
        radius = 0.006f;  
        color = Color.GREEN;  
    }  
    public Vector2f getPosition() {  
        return position;  
    }  
    public void draw( Graphics2D g, Matrix3x3f view ) {  
        g.setColor( color );  
        Vector2f topLeft = new Vector2f(  
            position.x - radius, position.y + radius );  
        topLeft = view.mul( topLeft );  
        Vector2f bottomRight = new Vector2f(  
            position.x + radius, position.y - radius );  
        bottomRight = view.mul( bottomRight );  
        int circleX = (int)topLeft.x;  
        int circleY = (int)topLeft.y;  
        int circleWidth = (int)(bottomRight.x - topLeft.x);  
        int circleHeight = (int)(bottomRight.y - topLeft.y);  
        g.fillOval(  
            circleX, circleY, circleWidth, circleHeight  
        );  
    }  
    public void update( float time ) {  
        position = position.add( velocity.mul( time ) );  
    }  
}
```



## 8.6 原型 Ship 类

PrototypeShip 代码位于 javagames.prototype 包中，这也是一目了然的。构造方法为移动飞船设置了一些常量，并且直接编码了模型的点。还有 set\*() 和 get\*() 方法用于销毁状态、角度、加速等，还有一些方法能够向左或向右旋转飞船。

launchBullet() 方法返回一个新的 PrototypeBullet 对象，该对象转换为飞船的突起部分。

```
// PrototypeShip.java
public PrototypeBullet launchBullet() {
    Vector2f bulletPos = position.add( Vector2f.polar( angle, 0.0325f ) );
    return new PrototypeBullet( bulletPos, angle );
}
```

isTouching() 方法检查渲染列表中的每一艘飞船与给定的小行星物体之间的碰撞。最酷的部分是 updatePosition() 方法，它使得飞船来回飞行。首先，要更新速度：

$$V_1 = V_0 + at$$

加速度在 setThrusting() 方法中设置。如果飞船没有向前移动，那么加速度为 0 并且速度保持不变。注意，新的加速度向量使用当前的角度，这会慢慢改变飞船航行的方向。

```
Vector2f accel = Vector2f.polar(angle, curAcc);
```

接下来，新的速度保持在最大值之下。如果没有执行这一步，飞船会持续加速并且最终会比子弹移动得还要快。注意这里使用了最小值，因此当最大速度不再比实际速度小的时候，最大速度为 1.0 并且保持不变。只有在最大速度小于实际值的时候，速度才会受到有效的限制。

```
float maxSpeed = Math.min( maxVelocity / velocity.len(), 1.0f );
velocity = velocity.mul( maxSpeed );
```

接下来，应用了摩擦力。即便太空中没有摩擦力，我还是想给飞船添加摩擦力，以便它最终能够减速。如果飞船没有加速，摩擦力将会使其慢下来。

```
float slowDown = 1.0f - friction * time;
velocity = velocity.mul( slowDown );
```

最后，更新位置，然后折返。



```
position = position.add( velocity.mul( time ) );
position = wrapper.wrapPosition( position );
```

PrototypeShip 代码如下所示：

```
package javagames.prototype;
import java.awt.*;
import java.util.ArrayList;
import javagames.util.*;

public class PrototypeShip {
    private float angle;
    private float acceleration;
    private float friction;
    private float maxVelocity;
    private float rotationDelta;
    private float curAcc;
    private Vector2f position;
    private Vector2f velocity;
    private PolygonWrapper wrapper;
    private boolean damaged;
    private Vector2f[] polyman;
    private ArrayList<Vector2f[]> renderList;
    public PrototypeShip( PolygonWrapper wrapper ) {
        this.wrapper = wrapper;
        friction = 0.25f;
        rotationDelta = (float)Math.toRadians( 180.0 );
        acceleration = 1.0f;
        maxVelocity = 0.5f;
        velocity = new Vector2f();
        position = new Vector2f();
        polyman = new Vector2f[] {
            new Vector2f( 0.0325f, 0.0f ),
            new Vector2f( -0.0325f, -0.0325f ),
            new Vector2f( 0.0f, 0.0f ),
            new Vector2f( -0.0325f, 0.0325f ),
        };
        renderList = new ArrayList<Vector2f[]>();
    }
    public void setDamaged( boolean damaged ) {
        this.damaged = damaged;
    }
    public boolean isDamaged() {
```



```
        return damaged;
    }

    public void rotateLeft( float delta ) {
        angle += rotationDelta * delta;
    }

    public void rotateRight( float delta ) {
        angle -= rotationDelta * delta;
    }

    public void setThrusting( boolean thrusting ) {
        curAcc = thrusting ? acceleration : 0.0f;
    }

    public void setAngle( float angle ) {
        this.angle = angle;
    }

    public PrototypeBullet launchBullet() {
        Vector2f bulletPos = position.add( Vector2f.polar( angle, 0.0325f ) );
        return new PrototypeBullet( bulletPos, angle );
    }

    public void update( float time ) {
        updatePosition( time );
        renderList.clear();
        Vector2f[] world = transformPolygon();
        renderList.add( world );
        wrapper.wrapPolygon( world, renderList );
    }

    private Vector2f[] transformPolygon() {
        Matrix3x3f mat = Matrix3x3f.rotate( angle );
        mat = mat.mul( Matrix3x3f.translate( position ) );
        return transform( polyman, mat );
    }

    private void updatePosition( float time ) {
        Vector2f accel = Vector2f.polar( angle, curAcc );
        velocity = velocity.add( accel.mul( time ) );
        float maxSpeed = Math.min( maxVelocity / velocity.len(), 1.0f );
        velocity = velocity.mul( maxSpeed );
        float slowDown = 1.0f - friction * time;
        velocity = velocity.mul( slowDown );
        position = position.add( velocity.mul( time ) );
        position = wrapper.wrapPosition( position );
    }

    private Vector2f[] transform( Vector2f[] poly, Matrix3x3f mat ) {
        Vector2f[] copy = new Vector2f[ poly.length ];
        for( int i = 0; i < poly.length; ++i ) {
```



```
        copy[i] = mat.mul( poly[i] );
    }
    return copy;
}
public void draw( Graphics2D g, Matrix3x3f view ) {
    g.setColor( new Color( 50, 50, 50 ) );
    for( Vector2f[] poly : renderList ) {
        for( int i = 0; i < poly.length; ++i ) {
            poly[i] = view.mul( poly[i] );
        }
        g.setColor( Color.DARK_GRAY );
        Utility.fillPolygon( g, poly );
        g.setColor( isDamaged() ? Color.RED : Color.GREEN );
        Utility.drawPolygon( g, poly );
    }
}
public boolean isTouching( PrototypeAsteroid asteroid ) {
    for( Vector2f[] poly : renderList ) {
        for( Vector2f v : poly ) {
            if( asteroid.contains( v ) ) {
                return true;
            }
        }
    }
    return false;
}
}
```

FlyingShipExample 位于 javagames.prototype 包中，如图 8.11 所示，它使用飞船和子弹代码来测试飞船在屏幕上的来回飞行。向左箭头和向右箭头会旋转飞船，向上箭头使飞船向前移动，而空格键发射子弹。

注意 updateObject()方法中 Java 的魔力。如果在遍历子弹的集合时试图移除子弹，那么，该列表将会抛出一个 ConcurrentModificationException。生成该列表的一个副本并且将其从最初的列表中移除，这样就不会抛出异常。

```
// FlyingShipExample.java
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
}
```



图 8.11 飞行的飞船的示例



```
ship.update( delta );
ArrayList<PrototypeBullet> copy =
    new ArrayList<PrototypeBullet>( bullets );
for( PrototypeBullet bullet : copy ) {
    bullet.update( delta );
    if( wrapper.hasLeftWorld( bullet.getPosition() ) ) {
        bullets.remove( bullet );
    }
}
}
```

FlyingShipExample 如下所示：

```
package javagames.prototype;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import javagames.util.*;

public class FlyingShipExample extends SimpleFramework {
    private PrototypeShip ship;
    private PolygonWrapper wrapper;
    private ArrayList<PrototypeBullet> bullets;
    public FlyingShipExample() {
        appBorderScale = 0.9f;
        appWidth = 640;
        appHeight = 640;
        appMaintainRatio = true;
        appSleep = 1L;
        appTitle = "Flying Ship Example";
    }
    @Override
    protected void initialize() {
        super.initialize();
        bullets = new ArrayList<PrototypeBullet>();
        wrapper = new PolygonWrapper( appWorldWidth, appWorldHeight );
        ship = new PrototypeShip( wrapper );
    }
    @Override
    protected void processInput( float delta ) {
        super.processInput( delta );
        if( keyboard_KeyDown( KeyEvent.VK_LEFT ) ) {
            ship.rotateLeft( delta );
        }
        if( keyboard_KeyDown( KeyEvent.VK_RIGHT ) ) {
```



```
    ship.rotateRight( delta );
}
if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
    bullets.add( ship.launchBullet() );
}
ship.setThrusting( keyboard.keyDown( KeyEvent.VK_UP ) );
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    ship.update( delta );
    ArrayList<PrototypeBullet> copy =
        new ArrayList<PrototypeBullet>( bullets );
    for( PrototypeBullet bullet : copy ) {
        bullet.update( delta );
        if( wrapper.hasLeftWorld( bullet.getPosition() ) ) {
            bullets.remove( bullet );
        }
    }
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    g.drawString( "Rotate: Left/Right Arrow", 20, 35 );
    g.drawString( "Thrust: Up Arrow", 20, 50 );
    g.drawString( "Fire: Space Bar", 20, 65 );
    Matrix3x3f view = getViewportTransform();
    ship.draw( (Graphics2D)g, view );
    for( PrototypeBullet b : bullets ) {
        b.draw( (Graphics2D)g, view );
    }
}
public static void main( String[] args ) {
    launchApp( new FlyingShipExample() );
}
}
```

## 8.7 编写原型游戏

原型游戏如图 8.12 所示，位于 javagames.prototype 包中，它使用了我们目前为止所见

过的所有技术。尽管这只是一个原型，并且目前还没有成为一款完整的游戏，但我已经展示了足够的工具来让一些功能奏效。如果要等到最后再制作一款游戏，可能需要等太长的时间。



图 8.12 原型游戏

该原型游戏使用了我们在本章前面所介绍的如下的类。

- PolygonWrapper
- PrototypeShip
- PrototypeAsteroid
- PrototypeAsteroidFactory
- PrototypeBullet

当你在本章末尾尝试编译和运行代码之前，确保已经创建了这些类。

initialize()方法为原型创建了所有这些对象，包括创建了一些星星作为背景。如下的代码创建了星星，并且创建了颜色的一个数组。Color 类接受 3 个值：red、blue 和 green，这 3 个值在 0 到 1 之间。将每种颜色值设置为相同的值，将会产生灰色的阴影。本书稍后会详细介绍颜色，但现在，只要使用 Java 语言所提供的 java.awt.Color 类就行了。

```
// PrototypeGame.java
private void createStars() {
```



```
stars = new Vector2f[ STAR_COUNT ];
colors = new Color[ STAR_COUNT ];
for( int i = 0; i < stars.length; ++i ) {
    float x = rand.nextFloat() * 2.0f - 1.0f;
    float y = rand.nextFloat() * 2.0f - 1.0f;
    stars[i] = new Vector2f( x, y );
    float color = rand.nextFloat();
    colors[i] = new Color( color, color, color );
}
}
```

当创建小行星的时候，注意 `getAsteroidStartPosition()` 方法。和产生各种大小的随机小行星的示例不同，这个方法只创建一个较大的随机小行星，并且使用如下代码将其放置到一个圆圈中，以使得它们不会在飞船之上产生。

```
// PrototypeGame.java
private Vector2f getAsteroidStartPosition() {
    float angle = (float) Math.toRadians( rand.nextInt( 360 ) );
    float minimum = appWorldWidth / 4.0f;
    float extra = rand.nextFloat() * minimum;
    float radius = minimum + extra;
    return Vector2f.polar( angle, radius );
}
```

前面的代码在圆圈中放置了新的多边形，该圆圈是屏幕的四分之一大，如图 8.13 所示。

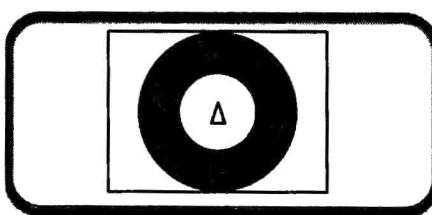


图 8.13 随机小行星的放置

这会防止这种现象发生：新产生的小行星出现于飞船的顶部，玩家还没来得及开火，就碰到它并爆炸了。这是游戏程序员所面对的各种挑战的一个很好的例子。只有在原型游戏开始运行时，这个问题才会变得明显。不管游戏多么简单，总是会有奇怪的问题需要解决。

`processInput()` 方法使用向左键和向右键来旋转飞船，向上键会激活加速动作，空格键会发射子弹，`Escape` 键会重新产生小行星。

当子弹击中小行星时，不仅小行星会从渲染列表中删除，而且如果小行星不是太小的话，它会分裂成两块更小的小行星。

`updateShip()` 方法检查碰撞。如果飞船被击中，会设置毁灭标志。尽管在真实的游戏中，当飞船被击中时，游戏会重新启动，但我们还没有介绍玩家生命或游戏结束状态的概念，因此，目前当飞船被击中时，将其绘制为红色。

`render()` 方法绘制星星、所有的小行星、子弹、飞船以及常用的帧速率和指令。还有



一些新的代码，它们会开启抗锯齿功能，以使线条绘制得更为平滑。第 10 章将会介绍抗锯齿。

```
Graphics2D g2d = (Graphics2D)g;
g2d.setRenderingHint(
    RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON
);
```

`drawStars()`、`drawShip()`、`drawAsteroids()`和`drawBullets()`方法负责绘制原型中的各种物体。`PrototypeGame` 的代码如下所示：

```
package javagames.prototype;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.*;
import javagames.prototype.PrototypeAsteroid.Size;
import javagames.util.*;

public class PrototypeGame extends SimpleFramework {
    private static final int STAR_COUNT = 1500;
    private PrototypeShip ship;
    private PolygonWrapper wrapper;
    private PrototypeAsteroidFactory factory;
    private ArrayList<PrototypeBullet> bullets;
    private ArrayList<PrototypeAsteroid> asteroids;
    private Random rand;
    private Vector2f[] stars;
    private Color[] colors;
    public PrototypeGame() {
        appBorderScale = 0.9f;
        appWidth = 640;
        appHeight = 640;
        appMaintainRatio = true;
        appSleep = 1L;
        appTitle = "Prototype Game";
    }
    @Override
    protected void initialize() {
        super.initialize();
        // create game objects
        rand = new Random();
        bullets = new ArrayList<PrototypeBullet>();
```



```
asteroids = new ArrayList<PrototypeAsteroid>();
wrapper = new PolygonWrapper( appWorldWidth, appWorldHeight );
ship = new PrototypeShip( wrapper );
factory = new PrototypeAsteroidFactory( wrapper );
createStars();
createAsteroids();
}
// this creates the random stars for the background
private void createStars() {
    stars = new Vector2f[ STAR_COUNT ];
    colors = new Color[ STAR_COUNT ];
    for( int i = 0; i < stars.length; ++i ) {
        float x = rand.nextFloat() * 2.0f - 1.0f;
        float y = rand.nextFloat() * 2.0f - 1.0f;
        stars[i] = new Vector2f( x, y );
        float color = rand.nextFloat();
        colors[i] = new Color( color, color, color );
    }
}
// create the random asteroids
private void createAsteroids() {
    asteroids.clear();
    for( int i = 0; i < 4; ++i ) {
        Vector2f position = getAsteroidStartPosition();
        asteroids.add( factory.createLargeAsteroid( position ) );
    }
}
// create random position for an asteroid
private Vector2f getAsteroidStartPosition() {
    float angle = (float)Math.toRadians( rand.nextInt( 360 ) );
    float minimum = appWorldWidth / 4.0f;
    float extra = rand.nextFloat() * minimum;
    float radius = minimum + extra;
    return Vector2f.polar( angle, radius );
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    // fly the ship
    if( keyboard_KeyDown( KeyEvent.VK_LEFT ) ) {
        ship.rotateLeft( delta );
    }
    if( keyboard_KeyDown( KeyEvent.VK_RIGHT ) ) {
```



```
        ship.rotateRight( delta );
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
        bullets.add( ship.launchBullet() );
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_ESCAPE ) ) {
        createAsteroids();
    }
    ship.setThrusting( keyboard.keyDown( KeyEvent.VK_UP ) );
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    updateAsteroids( delta );
    updateBullets( delta );
    updateShip( delta );
}
private void updateAsteroids( float delta ) {
    for( PrototypeAsteroid asteroid : asteroids ) {
        asteroid.update( delta );
    }
}
private void updateBullets( float delta ) {
    ArrayList<PrototypeBullet> copy =
        new ArrayList<PrototypeBullet>( bullets );
    for( PrototypeBullet bullet : copy ) {
        updateBullet( delta, bullet );
    }
}
// check for bullet collisions
private void updateBullet( float delta, PrototypeBullet bullet ) {
    bullet.update( delta );
    if( wrapper.hasLeftWorld( bullet.getPosition() ) ) {
        bullets.remove( bullet );
    } else {
        ArrayList<PrototypeAsteroid> ast =
            new ArrayList<PrototypeAsteroid>( asteroids );
        for( PrototypeAsteroid asteroid : ast ) {
            if( asteroid.contains( bullet.getPosition() ) ) {
                bullets.remove( bullet );
                asteroids.remove( asteroid );
                spawnBabies( asteroid );
            }
        }
    }
}
```



```
        }
    }

// create smaller asteroids when one is broken apart
private void spawnBabies( PrototypeAsteroid asteroid ) {
    if( asteroid.getSize() == Size.Large ) {
        asteroids.add(
            factory.createMediumAsteroid( asteroid.getPosition() ) );
        asteroids.add(
            factory.createMediumAsteroid( asteroid.getPosition() ) );
    }
    if( asteroid.getSize() == Size.Medium ) {
        asteroids.add(
            factory.createSmallAsteroid( asteroid.getPosition() ) );
        asteroids.add(
            factory.createSmallAsteroid( asteroid.getPosition() ) );
    }
}

// update the ship object
private void updateShip( float delta ) {
    ship.update( delta );
    boolean isHit = false;
    for( PrototypeAsteroid asteroid : asteroids ) {
        if( ship.isTouching( asteroid ) ) {
            isHit = true;
        }
    }
    ship.setDamaged( isHit );
}

@Override
protected void render( Graphics g ) {
    // render instructions
    super.render( g );
    g.drawString( "Rotate: Left/Right Arrow", 20, 35 );
    g.drawString( "Thrust: Up Arrow", 20, 50 );
    g.drawString( "Fire: Space Bar", 20, 65 );
    g.drawString( "Press ESC to respawn", 20, 80 );
    Graphics2D g2d = (Graphics2D)g;
    g2d.setRenderingHint(
        RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON
    );
    // draw game objects
```



```
Matrix3x3f view = getViewportTransform();
drawStars( g2d, view );
drawAsteroids( g2d, view );
drawBullets( g2d, view );
drawShip( g2d, view );
}

private void drawStars( Graphics2D g, Matrix3x3f view ) {
    for( int i = 0; i < stars.length; ++i ) {
        g.setColor( colors[i] );
        Vector2f screen = view.mul( stars[i] );
        g.fillRect( (int)screen.x, (int)screen.y, 1, 1 );
    }
}

private void drawShip( Graphics2D g, Matrix3x3f view ) {
    ship.draw( g, view );
}

private void drawAsteroids( Graphics2D g, Matrix3x3f view ) {
    for( PrototypeAsteroid asteroid : asteroids ) {
        asteroid.draw( g, view );
    }
}

private void drawBullets( Graphics2D g, Matrix3x3f view ) {
    for( PrototypeBullet b : bullets ) {
        b.draw( g, view );
    }
}

public static void main( String[] args ) {
    launchApp( new PrototypeGame() );
}
}
```

## 8.8 资源和延伸阅读

“How to Finish a Game,” <http://makegames.tumblr.com/post/1136623767/finishinga-game>.





# 第2部分

# 提高技能



# 第 9 章

## 文件和资源

使用文件有很多的优点，但是，最重要的是：

- 能够创建、读取、更新和删除数据，而不需要重新编译软件；
- 能够在软件运行期间保存和恢复信息。

对于游戏编程来说，理解如何读取和写入文件是必需的，因为音频、图像、对话、3D 模型、关卡数据以及游戏所需的任何其他内容，都是存储在文件中的。

本章介绍如何读取和写入属性文件以完成简单的任务，还介绍了如何用 XML 文件存储复杂的数据。本章中还有专门一节，介绍如何使用打包到 Java 存档文件(\*.jar)中的资源。一旦有了音频和视频文件，理解如何存放它们以及如何在游戏内部访问它们，就很重要。

### 9.1 理解 Java 如何处理文件和目录

在软件读取和写入文件之前，重要的是理解 Java 如何操作文件和目录。有经验的 Java 开发者可以跳过这一节，然后，简单地浏览一下后面关于文件和资源的信息。有一些有趣的问题需要考虑。

File 类可以表示一个文件或一个目录。创建一个文件的方式有很多种：

- File(URI uri)
- File(String pathname)
- File(String parent, String child)
- File(File parent, String child)

File 类中大多数方法的作用都一目了然，并且目前我们需要做的只是认识它们，因此，遍历文件树是一个不错的例子，如图 9.1 所示。



	WireFrameTriangle.class	12/29/10 8:33 AM
	WireFrameTriangle3D\$1.class	12/29/10 8:33
	WireFrameTriangle3D\$2.class	12/29/10 8:33
	WireFrameTriangle3D.class	12/29/10 8:33 AM
	build.xml	4/25/10 12:01 PM
+	docs	3/4/11 5:53 PM
+	junit	3/4/11 5:53 PM
+	lib	2/17/12 4:01 PM
	software-rendering.jar	5/1/10 7:35 AM
+	portfolio	2/17/12 4:01 PM
	index.html	5/1/10 7:35 AM
+	jnlp	2/17/12 4:01 PM
	3d-rendering-first-try.jnlp	5/1/10 7:35 AM
	3d-rendering-second-try.jnlp	5/1/10 7:35 AM
	3d-walking.jnlp	5/1/10 7:35 AM
		0 KB
		0 KB
		0 KB
		13 KB
		12 KB
		0 KB
		...

图 9.1 目录树

位于 `javagames.filesandres` 包中的 `FilesAndDirectories` 示例，遍历硬盘上的一个目录并且显示每个文件和目录的相关信息。`main()`方法调用 `runTest()`方法，后者包含了遍历文件树的代码。`runTest()`方法首先为硬盘上的每个目录创建一个新的文件对象，然后调用 `displayInfo()`方法，后者把文件的相关信息打印到控制台，然后递归地遍历剩下的文件。

使用如下的方法，来收集每个文件的相关信息。

- `canExecute()`——文件是否是可执行的。
- `canRead()`——文件是否是可读的。
- `canWrite()`——文件是否是可写的。
- `isHidden()`——文件是否是隐藏文件。
- `isDirectory()`——如果它是一个目录，为 `True`；否则为 `False`。
- `lastModified()`——最近一次修改文件的日期和时间。
- `length()`——文件的大小，以字节为单位。
- `getName()`——文件名。

还有其他一些方法可供使用。请参阅 Java 文档了解更多信息。

`StringBuilder` 类用来格式化文件信息。`lastModified` 值包含在一个 `Date()`类中，并且使用 `SimpleDateFormat()`来格式化。文件的长度转换为 KB，并且用 `DecimalFormat()`类格式化为包含逗号的形式。最后，读取、写入和执行等文件属性都附加到末尾。文件信息打印到屏幕上之后，使用 `listFiles()`方法来显示任何子文件夹的所有信息。`FilesAndDirectories` 代码如下所示：

```
package javagames.filesandres;
import java.io.File;
```



```
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class FilesAndDirectories {
    public FilesAndDirectories() {
    }
    public void runTest() {
        // list files and folders
        String dir = "C:\\temp";
        File file = new File( dir );
        displayInfo( 0, file );
    }
    private void displayInfo( int depth, File file ) {
        // Name, Date, Size, Attr
        boolean executable = file.canExecute();
        boolean readable = file.canRead();
        boolean writable = file.canWrite();
        boolean hidden = file.isHidden();
        boolean directory = file.isDirectory();
        long lastModified = file.lastModified();
        long length = file.length();
        String name = file.getName();
        // create ASCII file structure
        StringBuilder buf = new StringBuilder();
        for( int i = 0; i < depth; ++i ) {
            buf.append( "|" );
        }
        if( directory ) {
            buf.append( "+" );
        }
        if( name.isEmpty() ) {
            buf.append( "." );
        } else {
            buf.append( name );
        }
        // add modification date
        buf.append( "\\t\\t" );
        Date date = new Date( lastModified );
        buf.append( new SimpleDateFormat().format( date ) );
        buf.append( "\\t\\t" );
        // add file size in kilobytes
        long kb = length / 1024;
```



```
DecimalFormat format = new DecimalFormat();
format.setGroupingUsed( true );
buf.append( format.format( kb ) );
buf.append( " KB" );
// add read, write, execute attribute flags
buf.append( "\t\t" );
if( hidden ) buf.append( "." );
if( readable ) buf.append( "R" );
if( writable ) buf.append( "W" );
if( executable ) buf.append( "X" );
// print everything to the command line
System.out.println(buf.toString());
File[] children = file.listFiles();
if( children != null ) {
    for( File child : children ) {
        displayInfo( depth + 1, child );
    }
}
}
public static void main( String[] args ) {
    new FilesAndDirectories().runTest();
}
}
```

## 9.2 理解输入/输出流

在 Java 中，`InputStream` 是用来处理多种数据来源的一个接口。`InputStream` 可能是文件、数组、网络套接字以及外围设备。下面列出了可以用来创建不同资源的各种输入源。

- 属性: `InputStream`、`Reader`。
- 图像: `File`、`InputStream`、`ImageInputStream`、`URL`。
- 音频: `InputStream`。
- XML: `InputStream`、`Reader`、`File`、`InputSource`。

尽管有各种不同的类可以用来读取和写入数据，但 `InputStream` 是唯一一个通用的类。即便使用 `Class` 和 `ClassLoader`，也可以使用 `InputStream` 访问类路径资源。

有了所有这些依赖于 `InputStream` 的类，它应该很容易使用。然而，由于异常处理模式与很多早期的 Java API 相关，使用 `InputStream` 可能有些麻烦并且容易出错。



由于流在使用完之后总是应该关闭，因此即便存在一个错误，基本的模式也应该如下所示：

```
InputStream in = null;
try {
    in = [make new stream]
    while( [has more data] ) {
        [process data]
    }
} catch( IOException ex ) {
    [handle error or rethrow]
} finally {
    try {
        in.close();
    } catch( Exception ex ) { }
}
```

不仅创建和读取一个 `InputStream` 导致的异常必须要捕获，而且还要关闭流。

`read()`方法返回一个整数，因此，记得将其转换为正确的数据类型。当 `read()`方法返回 -1 时，则不再有更多数据可用。读取字符串使用一个 `Reader` 接口，它包装了一个 `InputStream`。模式是相同的，只不过每个 `readLine()`方法返回单个的一行文本，而不是返回单个的字节。

这段示例代码读取的是 `lorem-ipsum.txt` 文件，它包含了如下的文本：

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia
deserunt mollit anim id est laborum.
```

位于 `javagames.filesandres` 包中的 `ReadingDataFromFiles` 代码，是从文件读取字节和字符串的一个小示例。

```
package javagames.filesandres;
import java.io.*;
public class ReadingDataFromFiles {
    public void runTest() {
        String path = "./res/assets/text/lorem-ipsum.txt";
        File file = new File( path );
```



```
    readInBytes( file );
    readInStrings( file );
}

public void readInBytes( File file ) {
    System.out.println();
    System.out.println();
    System.out.println( "*****" );
    System.out.println( "Reading in bytes" );
    System.out.println();
    InputStream in = null;
    try {
        in = new FileInputStream( file );
        int next = -1;
        while( (next = in.read()) != -1 ) {
            System.out.print( (char)next );
        }
    } catch( IOException ex ) {
        ex.printStackTrace();
    } finally {
        try {
            in.close();
        } catch( IOException ex ) { }
    }
}
public void readInStrings( File file ) {
    System.out.println();
    System.out.println();
    System.out.println( "*****" );
    System.out.println( "Reading in strings" );
    System.out.println();
    Reader reader = null;
    try {
        reader = new FileReader( file );
        BufferedReader buf = new BufferedReader( reader );
        String line = null;
        while( (line = buf.readLine()) != null ) {
            System.out.println( line );
        }
    } catch( IOException ex ) {
        ex.printStackTrace();
    } finally {
        try {
            reader.close();
        }
    }
}
```



```
        } catch( IOException ex ) { }
    }
}

public static void main( String[] args ) {
    new ReadingDataFromFiles().runTest();
}
}
```

使用 `OutputStream` 类写文件和使用 `InputStream` 类似。其模式如下：

```
OutputStream out = null;
try {
    out = [open output stream]
    [write out data]
} catch( FileNotFoundException fex ) {
    [handle error]
} catch( IOException ioex ) {
    [handle error]
} finally {
    try {
        out.close()
    } catch( Exception ex ) { }
}
```

写字节很简单，只需要用一个文件打开一个 `FileOutputStream` 并写字节。注意，这里专门捕获 `FileNotFoundException`。这是一个可选的捕获，因为 `FileNotFoundException` 是一个 `IOException`，但有的时候，以不同的方式处理这两种错误更好。

然而，Java 喜欢把事情搞得困难。尽管可以用 `BufferedReader` 读取字符串，但不能使用 `BufferedReader` 写出整个字符串。要写出带有换行的整个字符串，需要使用 `PrintWriter` 类。这个类有很多的方法，用于将字符串写到一个文件中。不要试图找到 `PrintReader` 类，根本没有这个类。

好消息是很多人已经投入这场战斗，因此，快速地在互联网上搜索下，总是能够找到答案。位于 `javagames.filesandres` 包中的 `WritingDataToFiles`，是将数据写入到文件的一个示例。

```
package javagames.filesandres;
import java.io.*;
import java.util.Random;

public class WritingDataToFiles {
    private Random rand = new Random();
    public void runTest() {
        writeOutBytes( "./res/assets/text/byte-file.bin" );
    }
}
```



```
        writeOutStrings( "./res/assets/text/string-file.txt" );
    }

    public void writeOutBytes( String fileName ) {
        System.out.println();
        System.out.println();
        System.out.println("*****");
        File file = new File( fileName );
        OutputStream out = null;
        try {
            out = new FileOutputStream( file );
            for( int i = 0; i < 1000; ++i ) {
                out.write( rand.nextInt( 256 ) );
            }
        } catch( FileNotFoundException fex ) {
            fex.printStackTrace();
        } catch( IOException ioex ) {
            ioex.printStackTrace();
        } finally {
            try {
                out.close();
                System.out.println("Wrote: " + file.getPath());
            } catch( Exception ex ) { }
        }
    }

    public void writeOutStrings( String fileName ) {
        System.out.println();
        System.out.println();
        System.out.println("*****");
        // We read with a BufferedReader, and write with
        // Expect a BufferedWriter, but no...
        // Damn Java, a PrintWriter.
        // Is there a PrintReader? No. Why? No idea.
        String[] strings = {
            "Lorem ipsum dolor sit amet, consectetur adipisicing elit,", "sed do eiusmod tempor incididunt ut labore et dolore magna", "aliqua. Ut enim ad minim veniam, quis nostrud exercitation", "ullamco laboris nisi ut aliquip ex ea commodo consequat.", "Duis aute irure dolor in reprehenderit in voluptate velit", "esse cillum dolore eu fugiat nulla pariatur. Excepteur sint", "occaecat cupidatat non proident, sunt in culpa qui officia", "deserunt mollit anim id est laborum.", };
        File file = new File( fileName );
        PrintWriter out = null;
```



```
try {
    out = new PrintWriter( file );
    for( String string : strings ) {
        out.println( string );
    }
} catch( FileNotFoundException fex ) {
    fex.printStackTrace();
} finally {
    try {
        out.close();
        System.out.println("Wrote: " + file.getPath());
    } catch( Exception e ) { }
}
}

public static void main( String[] args ) {
    new WritingDataToFiles().runTest();
}
}
```

## 9.3 创建 Resources.jar 文件进行测试

Java 存档文件是重命名为\*.jar 的\*.zip 文件。这些文件可以保存数据、资源、代码、其他的存档文件、本地库或者程序所需的任何其他内容。由于能够运行 Java 的所有操作系统也可以对\*.zip 文件执行文件操作，因此，这是很便于使用的一种格式。在本书后面，使用 ANT 构建工具，所有的图像和音频文件都将存储到一个存档文件中。然而，一旦文件存储到了存档文件，就需要以一种特殊的方式来访问它们。即便不考虑存档文件，加载图像和声音也已经足够困难了，因此，本节将详细介绍使用\*.jar 文件中的资源的过程。

要学习如何从存档文件提取数据，首先需要创建一个存档文件。如果想要了解关于这些文件的更多信息，参阅 9.8 “资源和延伸阅读” 部分。

在下面的示例中，Java 的 jar 命令行工具用来创建测试文件，该文件添加到了 classpath 中。这个文件需要使用 JAR 文件中的资源进行测试。



记住 jar.exe (Windows) 或 jar.sh (Linux) 文件需要位于系统路径中，才能供存档文件使用。该程序位于和 javac 程序相同的位置。



要创建在后面示例中使用的 Java 存档文件，应在本地硬盘上创建如下的结构：

```
+ javagames (folder)
+ filesandres (folder)
  Test1.txt
  Test2.txt
  Test3.txt
```

在每个文本文件中，添加标题和“Lorem ipsum”文本。

Test1

```
-----
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Duis id ante quis justo ultricies euismod. Vestibulum ante
ipsum primis in faucibus orci luctus et ultrices posuere
cubilia Curae; Donec urna nibh, feugiat vitae facilisis ut,
iaculis id metus. Suspendisse eu enim vitae ligula auctor
dictum eu et orci. Pellentesque habitant morbi tristique senectus
et netus et malesuada fames ac turpis egestas. Phasellus
elementum rhoncus ante eu fringilla. Integer volutpat dolor
sed tellus pulvinar id laoreet metus sollicitudin.
Pellentesque vitae vehicula justo. Donec dolor mi,
mattis ut posuere eget, bibendum non velit. Etiam eget felis
diam, non venenatis neque. Integer nec libero quis libero
rhoncus volutpat. Duis volutpat ultricies erat, at tincidunt
lectus facilisis sit amet.
```

将命令行导航到包含了顶级 `javagames` 文件夹的文件夹中，并且运行如下的命令：

```
jar -cf resources.jar.
```

Java 存档文件工具可以用于打包应用程序。在本书后面，我们将学习 ANT 构建工具，它使得这类任务变得容易。`-cf` 标志实际上是两个标志：`c` 是一个创建标志，`f` 是一个文件名标志。`resources.jar` 文件是要创建的文件的名字，`.`(点)是当前的目录，也就是要添加到存档文件中的文件所在的位置。这里使用当前目录，是因为没有指定目录信息。

在后面的代码实例中，位于 `resources.jar` 中的文件将通过类路径而加载。确保在所使用的开发环境中，`resources.jar` 文件添加到了类路径中，否则的话，将无法找到文件。

参阅开发环境文档，将 `resources.jar` 文件添加到类路径中。如果通过命令行来运行它，看上去如下所示：

```
java -cp resources.jar class.to.Run
```



## 9.4 将资源放到类路径上

使用文件的一个问题是，你必须知道文件的路径才能找到它。绝对文件路径如下所示：

```
C:\Users\Java\Game
```

如果用户没有 C 盘，或者如果他使用其他操作系统的话，这个路径会无效。相对路径也比较类似，但没有这么明显的问题。

```
./relative/path/to/file.txt
```

如果 Java 应用程序在正确的文件夹中启动的话，使用相对文件路径是有效的，但是，如果应用程序从另一个位置启动，将会出错。例如，创建一个脚本，它从另一个目录启动程序，这将会破坏相对路径。尽管只允许用户从当前目录启动应用程序也是可以接受的，但还有一种更容易的方式。

大多数使用过 Java 一段时间的人，都熟悉类路径，这是运行应用程序所需的所有目录和 JAR 文件的一个神奇列表。我们不仅可以把 JAR 文件和目录添加到类路径，而且可以访问 JAR 文件和包含资源的文件夹。当我们把资源包含到类路径中的时候，应用程序从何处启动是无关紧要的。在任何操作系统中的任何文件系统中，文件的路径都是相同的。

先不要高兴，重要的是要理解打开类路径资源的不同方法之间的差异。要从类路径加载资源，有两个不同的类可供使用。

- `ClassLoader.getSystemResourceAsStream()`
- `Class.getResourceAsStream()`

这两个方法看起来很相似，因此，很容易搞混。由于它们几乎相同，因此，必须仔细确保对正确的资源使用正确的方法。

如下是容易混淆的地方。`ClassLoader` 和 `Class` 都可以使用绝对路径加载资源，但是，只有 `Class` 可以加载相对资源，如图 9.2 所示。

如果资源的完整路径是 `path/to/file.txt`，那么 `ClassLoader` 和 `Class` 都可以加载它。但是，由于 `Class` 也可以相对于自身加载文件，因此如果是绝对路径

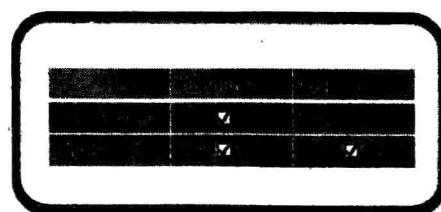


图 9.2 Class Loader 资源



的话，Class 需要在路径的开头处有一个/。

```
ClassLoader - "path/to/file.txt"
Class - "/path/to/file.txt"
```

如果在用于 Class 路径的开头处漏掉了/，将会找不到资源，因为它将会相对自身来查找而不是从根目录来查找。使用 Class 的时候，还有另一个问题。获取类对象的常规方式是，使用 getClass()方法。如下的方式可以很好地工作：

```
getClass().getResourceAsStream( "resource" );
```

然而，getClass()方法的问题在于，它总是返回最深的子类，即便在父类中也是如此。如果最深的子类创建于不同的包中，那么，在父类中加载的资源将无法找到。要防止这种情况发生，在使用相对资源加载的时候，应该使用静态类。这样一来，不管如何创建子类，都能够毫无问题地工作。

```
ClassName.class.getResourceAsStream( "resource" );
```

位于 javagames.filesandres 包中的 ClasspathResources 示例，展示了加载资源的不同方式。记住，应创建 resources.jar 文件并手动将其添加到类路径，以使资源在运行时可用。

```
package javagames.filesandres;
import java.io.*;

public class ClasspathResources {
    public ClasspathResources() {
    }
    public void runTest() {
        /*
         * ClassLoader uses absolute path. There is
         * NO '/' at the beginning of the path!
         */
        System.out.println();
        System.out.println("*****");
        System.out.println("ClassLoader - Absolute Path");
        System.out.println();
        InputStream in = ClassLoader.getSystemResourceAsStream(
            "javagames/filesandres/Test1.txt" );
        printResource( in );
        /*
         * getClass() instead of class loader.
         * Can be relative...
         */
    }
}
```



```
System.out.println();
System.out.println("*****");
System.out.println("getClass() - Relative path");
System.out.println();
in = getClass().getResourceAsStream( "Test2.txt" );
printResource( in );
/*
 * getClass() can also use the absolute path,
 * but it needs a '/' at the start of the path.
 */
System.out.println();
System.out.println("*****");
System.out.println("getClass() - Absolute path");
System.out.println();
in = getClass().getResourceAsStream(
    "/javagames/filesandres/Test3.txt" );
printResource( in );
/*
 * Because getClass() always returns the subclass,
 * if a subclass is created in another package,
 * the relative path may not be correct.
 * However, using an absolute path doesn't allow
 * packages to be moved around. Use the static class,
 * which also works in static methods.
 */
System.out.println();
System.out.println("*****");
System.out.println("getClass() - Absolute path");
System.out.println();
in = ClasspathResources.class.getResourceAsStream( "Test3.txt" );
printResource( in );
/*
 * Either ClassLoader or Class will return null
 * for unknown resources
 */
in = getClass().getResourceAsStream( "fat/finger/mistake" );
if( in == null ) {
    System.out.println();
    System.out.println( "*****" );
    System.out.println( "Got a null back!!!!" );
}
in = ClassLoader.getSystemResourceAsStream( "fat/finger/mistake" );
if( in == null ) {
```



```
        System.out.println( "Got another null back!!!!" );
    }
}

private void printResource( InputStream in ) {
    try {
        InputStreamReader reader = new InputStreamReader( in );
        BufferedReader buf = new BufferedReader( reader );
        String line = null;
        while( (line = buf.readLine()) != null ) {
            System.out.println(line);
        }
    } catch( IOException e ) {
        e.printStackTrace();
    } finally {
        try {
            in.close();
        } catch( Exception e ) { }
    }
}

public static void main( String[] args ) {
    new ClasspathResources().runTest();
}
```

## 9.5 制作资源加载工具包

尽管从 JAR 内部加载资源对于部署应用程序来说很好，但在开发过程中，这很可怕。如果每次资源文件中有一处修改，就必须创建一个新的 JAR 文件并将其添加到类路径中，然后才能进行测试，这样开发过程会变得很慢。好在，有很多方法能够处理在开发过程中从类路径加载资源的问题。

- 将 JAR 文件添加到类路径。每次做出修改的时候，都需要重新构建和部署一个新的 JAR 文件，这很痛苦，而且 JAR 文件中的资源不能修改，这也很痛苦。
- 把资源文件夹添加到类路径中。这种方法要好一点，但是，资源文件夹的文件夹结构和 JAR 文件必须一致。此外，在设置一个新项目的时候，我使用了这个方法但是忘记添加文件夹，这会带来麻烦而且不容易发现问题。
- 从文件系统和类路径加载资源。这需要额外的代码来尝试从两个不同的地方加载



资源，但不需要在项目中进行设置就可以工作。

本书采用了第三种方法，因为它不需要在开发环境中进行任何额外的设置。

位于 `javagames.util` 包中的 `ResourceLoader` 类，用于从类路径和文件系统加载资源。

```
package javagames.util;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;

public class ResourceLoader {
    public static InputStream load(
        Class<?> clazz, String filePath, String resPath ) {
        // try the resource first
        InputStream in = null;
        if( !( resPath == null || resPath.isEmpty() ) ) {
            in = clazz.getResourceAsStream( resPath );
        }
        if( in == null ) {
            // try the file path
            try {
                in = new FileInputStream( filePath );
            } catch( FileNotFoundException e ) {
                e.printStackTrace();
            }
        }
        return in;
    }
}
```

位于 `javagames.filesandres` 包中的 `ResourceLoaderExample` 代码，使用 `ResourceLoader` 从文件系统和类路径加载文件。确保在示例中调整文件路径，以便与你系统上的文件系统相匹配。

```
package javagames.filesandres;
import java.io.*;
import javagames.util.ResourceLoader;

public class ResourceLoaderExample {
    public ResourceLoaderExample() {
    }
    public void runTest() {
        Class<?> clazz = ResourceLoaderExample.class;
        // load absolute resource
```



```
String filePath = "not/used";
String resPath = "/javagames/filesandres/Test1.txt";
InputStream in = ResourceLoader.load( clazz, filePath, resPath );
printResource( in );
// load relative resource
filePath = "not/used";
resPath = "Test2.txt";
in = ResourceLoader.load( clazz, filePath, resPath );
printResource( in );
// load absolute file path
filePath = "C:/Book/res/assets/lib/javagames/filesandres/Test3.txt";
resPath = "/not/available";
in = ResourceLoader.load( clazz, filePath, resPath );
printResource( in );
// load relative file path
filePath = "res/assets/lib/javagames/filesandres/Test3.txt";
resPath = "/not/available";
in = ResourceLoader.load( clazz, filePath, resPath );
printResource( in );
// error with both is null
filePath = "fat/finger";
resPath = "fat/finger/too";
in = ResourceLoader.load( clazz, filePath, resPath );
printResource( in );
}
private void printResource( InputStream in ) {
try {
    InputStreamReader reader = new InputStreamReader( in );
    BufferedReader buf = new BufferedReader( reader );
    String line = null;
    while( (line = buf.readLine()) != null ) {
        System.out.println(line);
    }
} catch( IOException e ) {
    e.printStackTrace();
} finally {
    try {
        in.close();
    } catch( Exception e ) { }
}
}
public static void main( String[] args ) {
    new ResourceLoaderExample().runTest();
}
```



}  
}

## 9.6 利用 Java 属性

Java 包含了一个非常有用的类，可以用来处理简单的名—值对属性。Properties 类可以读取包含一个文件，其中包含了名—值对组成的行。Properties 类通过 System 对象而可用，适用于系统范围内的属性，但是，已经创建的任何属性类都可以使用。

```
name1=value1  
name2=value2
```

位于 javagames.filesandres 包中的 PrintSystemProperties 示例，打印输出所有的系统属性值，然后突出显示一些有用的系统属性。

```
package javagames.filesandres;  
import java.util.Properties;  
  
public class PrintSystemProperties {  
    public static void main( String[] args ) {  
        Properties system = System.getProperties();  
        for( Object key : system.keySet() ) {  
            System.out.println(  
                key + "=" + system.getProperty( key.toString() )  
            );  
        }  
        System.out.println();  
        System.out.println( "----- Some Cool Props -----" );  
        System.out.println( "java.version=" +  
            System.getProperty( "java.version" ) );  
        System.out.println( "os.name=" + System.getProperty( "os.name" ) );  
        System.out.println( "user.country=" +  
            System.getProperty( "user.country" ) );  
        System.out.println( "user.language=" +  
            System.getProperty( "user.language" ) );  
        System.out.println(  
            "user.home=" + System.getProperty( "user.home" ) );  
        System.out.println( "user.dir=" + System.getProperty( "user.dir" ) );  
        System.out.println(
```



```
    "user.name=" + System.getProperty( "user.name" ) );  
}  
}
```

系统属性有用的原因之一是，Java 包含了-D 命令行标志，它允许在命令行上定义系统属性。该标志的格式是-Dname=value。要设置一个命令行属性，直接包含所需的那么多个-D 标志即可。如下面的代码段所示，当传递该标志的时候，它们不需要放置在命令的最后，因为类名后面带有一个 JAR 名称的任何文本，都会作为字符串值传递给 main()方法。

```
C:\> java -Dname1=value1 -Dname2=value2 -jar CoolGame.jar  
"Don't put the -D stuff here. It won't work."
```

可以很容易地获取属性，因为 System 类在任何地方都可用。有两个方法可以获取属性值：

```
String prop = System.getProperty( "name" );  
String default = System.getProperty( "name", "default" );
```

第一个方法只需要给其传递一个属性名，并且，如果该属性不可用的话，会返回 null。第二个方法包含了一个默认值，如果没有找到该属性的话，就返回这个默认值。使用属性需要注意一个陷阱，一旦设置了属性，仍然可以覆盖它们。在加载多个属性文件和系统属性的时候，必须要小心，因为最近一次的属性定义将会擦除之前的属性值。

位于 javagames.filesandres 包中的 SavingPropertyFiles 示例，创建了一个名为 testing.properties 的简单文件。这个文件中有 3 个属性：prop1、prop2 和 override。还有一些以#或以!字符开头的注释。

```
package javagames.filesandres;  
import java.io.*;  
  
public class SavingPropertyFiles {  
    public SavingPropertyFiles() {  
    }  
    public void saveFile() {  
        File file = new File( "testing.properties" );  
        PrintWriter out = null;  
        try {  
            out = new PrintWriter( file );  
            out.println( "# This is a comment" );  
            out.println( "prop1=fileValue1" );  
            out.println( "# This is another comment" );  
            out.println( "prop2=fileValue2" );  
            out.println( "# This can be overriden from the" );  
        } catch ( IOException e ) {  
            e.printStackTrace();  
        } finally {  
            if ( out != null ) {  
                out.close();  
            }  
        }  
    }  
}
```



```
        out.println( "! command line, or not..." );
        out.println( "override=fileoverride" );
    } catch( FileNotFoundException fex ) {
        fex.printStackTrace();
    } finally {
        try {
            out.close();
            System.out.println("Wrote: " + file.getPath());
        } catch( Exception e ) { }
    }
}
public static void main( String[] args ) {
    new SavingPropertyFiles().saveFile();
}
}
```

PropertyFileExample 使用了前面示例中 testing.properties 文件。代码首先查找一个名为 load.props.first 的属性。通过如下方式获取这个 System 属性：

```
// PropertyFileExample.java
private boolean getLoadFirstProperty() {
    try {
        String value = System.getProperty( "load.props.first", "false" );
        return Boolean.parseBoolean( value );
    } catch( Exception e ) {
        return false;
    }
}
```

如果找到该属性，会将其解析为一个布尔值。如果没有找到该属性，其默认值为 false。该属性用于判定在 testing.properties 文件之前还是之后加载 System 属性。由于最近加载的文件决定了可用的变量，因此在运行时哪个属性可用取决于加载文件的顺序。

如果 load.props.first 为 false，那么 System 属性将不会先加载，并且会覆盖 testing.properties 文件。如果 load.props.first 为 true，那么，命令行的 override 属性会被加载的文件覆盖。



尝试通过传递一个命令行 override，并且将 load.props.first 分别设置为 true 和 false 来运行该示例，看看会发生什么情况。

```
java -Dload.props.first=true -Doverride=command-line -classpath
C:\javagames\book\bin javagames.filesandres.PropertyFileExample
```



你的机器上的命令行示例应该也是类似的。

```
package javagames.filesandres;
import java.io.*;
import java.util.Properties;
import javagames.util.ResourceLoader;

public class PropertyFileExample {
    private Properties props;

    public PropertyFileExample() {
        props = new Properties();
    }

    private void loadAndPrintProperties() {
        boolean loadPropsFirst = getLoadFirstProperty();
        if( loadPropsFirst ) {
            props.putAll( System.getProperties() );
        }
        loadProperties();
        if( !loadPropsFirst ) {
            props.putAll( System.getProperties() );
        }
        printProperties();
    }

    private void loadProperties() {
        InputStream in = null;
        try {
            in = ResourceLoader.load( PropertyFileExample.class,
                "./testing.properties", "/testing.properties" );
            props.load( in );
        } catch( IOException e ) {
            e.printStackTrace();
        } finally {
            try {
                in.close();
            } catch( Exception e ) { }
        }
    }

    private boolean getLoadFirstProperty() {
        try {
            String value = System.getProperty( "load.props.first", "false" );
            return Boolean.parseBoolean( value );
        } catch( Exception e ) {
```



```
        return false;
    }
}

private void printProperties() {
    System.out.println(
        "load.props.first=" + getLoadFirstProperty()
    );

    System.out.println(
        "user.home=" + props.getProperty( "user.home", "." )
    );
    System.out.println(
        "prop1=" + props.getProperty( "prop1", "default1" )
    );
    System.out.println(
        "prop2=" + props.getProperty( "prop2", "default2" )
    );
    System.out.println(
        "prop3=" + props.getProperty( "prop3", "default3" )
    );
    System.out.println(
        "override=" + props.getProperty( "override", "defaultOverride" )
    );
}

public static void main( String[] args ) {
    PropertyFileExample example = new PropertyFileExample();
    example.loadAndPrintProperties();
}
```

## 9.7 XML 文件概览

使用 XML 文件是存储、保存和加载游戏相关信息的一种方法。尽管有很多不同的文件格式，但由于支持 XML 不需要任何第三方库，因此，本书采用该格式。

如果你不熟悉 XML 的话，Web 上有大量的图书、教程和示例介绍 XML，浏览一下其基础知识是一个不错的想法。如下是一些不同类型的 XML 标记。

XML 版本包含在每个文件的顶部。



```
<?xml version="1.0" encoding="UTF-8"?>
```

每个 XML 文件都有一个唯一的根标记：

```
<root>  
</root>
```

每个标记包含一个起始标记和结束标记。每个起始标记必须有一个相同名称的结束标记：

```
<tag></tag>
```

如果一个标记中没有内容，可以使用其简写版本。

```
<shorthand/>
```

标记自身之中可以包含文本、其他的标记，或者二者都包含：

```
<text>I'm some text</text>  
<nested><child/></nested>  
<nested-text>some text and <child/></nested-text>
```

标记也可以在其开始标记中包含属性，如下所示：

```
<attributes attr1="value1" attr2="values"/>
```

如下的字符在 XML 中是不允许的：< > & '和"。这些字符中的每一个，都通过如下的特殊字符序列来表示：

```
< = &lt;;  
> = &gt;;  
& = &amp;;  
" = &quot;;  
' = &apos;;
```

要包含不解析的文本，可以使用 CDATA 字符标记：

```
Start tag = <![CDATA[  
End tag = ]]>
```

在这些标记中，即便是特殊字符也会捕获而不进行解析。

最后，注释的形式如下：

```
<!-- comment -->
```

注释中唯一不允许的字符序列，是两个连字符--。注释可以跨越多行。

位于 javagames.filesandres 包中的 SaveXMLExample，使用 PrintWriter 类将文件内容写出，从而创建了一个 XML 文件。这段代码所创建的 sample.xml 文件，稍后用做测试文件



来解析 XML。

```
package javagames.filesandres;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class SaveXMLExample {
    public SaveXMLExample() {
    }
    public void createXMLFile() {
        File file = new File( "sample.xml" );
        PrintWriter out = null;
        try {
            out = new PrintWriter( file );
            writeXML( out );
        } catch( FileNotFoundException fex ) {
            fex.printStackTrace();
        } finally {
            try {
                out.close();
                System.out.println("Wrote: " + file.getPath());
            } catch( Exception e ) { }
        }
    }
    private void writeXML( PrintWriter out ) {
        out.println( "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" );
        // all XML must have a single root tag
        out.println( "<root>" );
        // a comment
        out.println( " <!-- This is a comment -->" );
        // and empty tag
        out.println( " <empty></empty>" );
        // shorthand for a tag with no children
        out.println( " <shorthand/>" );
        // text element
        out.println( " <text>I'm some text</text>" );
        // nested elements
        out.println( " <nested><child/></nested>" );
        // nested text with child
        out.println( " <nested-text>I'm some text <child/></nested-text>" );
        // attributes
        out.println( " <attributes attr1=\"value1\" attr2=\"value2\" />" );
    }
}
```



```
// special characters
out.println(
    " <special-chars> &lt; &gt; &apos;" +
    " " </special-chars>" );
// unparsed character data CDATA
out.println( " <cdata>" ) ;
// Starting CDATA tag
out.println( " <![CDATA[" );
out.println( "<xml><attr=\\"xml inside xml\\"/></xml>" );
// Ending CDATA tag
out.println( " ]]>" );
out.println( " </cdata>" );
// ending root tag
out.println( "</root>" );
}
public static void main( String[] args ) {
    new SaveXMLExample().createXMLFile();
}
}
```

写出到 XML 文件比解析 XML 文件要容易得多。好在，在加载游戏数据的时候，并不会涉及 XML 规范的真正复杂或令人混淆的部分。

XMLUtility 类添加到了 javagames.util 包中，它包含了一些辅助方法，用来解析 XML 文件。parseDocument()方法接受一个 InputStream 或一个 Reader，并且返回一个 Document 对象，它包含了内存中的整个 XML 文件。getElements()和 getAllElements()方法返回具有给定标记名称的元素。不同之处在于，getElements()方法只是在一个层级深度上查找该标记，而 getAllElements()返回拥有给定名称的任何标记，而不管它们在文档中位于何处。

```
package javagames.util;
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class XMLUtility {
    public static Document parseDocument( InputStream inputStream )
        throws ParserConfigurationException, SAXException, IOException {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse( new InputSource( inputStream ) );
        return document;
    }
}
```



```
}

public static Document parseDocument( Reader reader )
throws ParserConfigurationException, SAXException, IOException {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse( new InputSource( reader ) );
    return document;
}

public static List<Element> getAllElements(
    Element element, String tagName ) {
    ArrayList<Element> elements = new ArrayList<Element>();
    NodeList nodes = element.getElementsByTagName( tagName );
    for( int i = 0; i < nodes.getLength(); i++ ) {
        elements.add( (Element)nodes.item( i ) );
    }
    return elements;
}

public static List<Element> getElements(
    Element element, String tagName ) {
    ArrayList<Element> elements = new ArrayList<Element>();
    NodeList children = element.getChildNodes();
    for( int i = 0; i < children.getLength(); i++ ) {
        Node node = children.item( i );
        if( node.getNodeType() == Node.ELEMENT_NODE ) {
            String nodeName = node.getNodeName();
            if( nodeName != null && nodeName.equals( tagName ) ) {
                elements.add( (Element)node );
            }
        }
    }
    return elements;
}
}
```

位于 javagames.filesandres 包中的 LoadXMLExample 代码，解析了前面示例中的 sample.xml 文件。

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
    <!-- This is a comment -->
    <empty></empty>
    <shorthand/>
    <text>I'm some text</text>
```



```

<nested><child/></nested>
<nested-text>I'm some text <child/></nested-text>
<attributes attr1="value1" attr2="value2" />
<special-chars> &lt; &gt; &amp; &apos; &quot; </special-chars>
<cdata>
    <![CDATA[
<xml><attr="xml inside xml"/></xml>
    ]]>
</cdata>
</root>

```

你需要采取递归的方式来处理文档，因为 XML 标记可以包含其他的标记。要在不知道文档结构的情况下检查文档，使用如下模式：

```

// LoadXMLExample.java
private void inspectXML( Node node ) {
    printNode( node );
    if( node.hasAttributes() ) {
        NamedNodeMap nodeMap = node.getAttributes();
        for( int i = 0; i < nodeMap.getLength(); ++i ) {
            inspectXML( nodeMap.item( i ) );
        }
    }
    if( node.hasChildNodes() ) {
        NodeList nodeList = node.getChildNodes();
        for( int i = 0; i < nodeList.getLength(); ++i ) {
            inspectXML( nodeList.item( i ) );
        }
    }
}

```

每个节点的类型、名称和值都打印出来了。然后，使用相同的函数，递归地处理任何属性。对于任何子元素，做同样的事情。`parseXML()`方法使用 `XMLUtility` 类，使用标记名称提取出每个元素。

如下的方法，从每个元素中获取了必要的信息。

- `Element.getNodeName()`——返回了 XML 标记的名称。
- `Element.getTextContent()`——返回了开始标记和结束标记之间的文本。
- `Element.getAttribute(String name)`——给定了属性名称，如果该属性可用的话，该方法返回属性值或者 `null`。

注意试图获取“`child`”标记的部分。这里有两个不同的元素，带有相同的标记名，但



是，返回的可能是没有标记、一个标记或两个标记，这取决于用来获取它们的方法。有时候，你想要获取具有特定名称的所有标记，但是有时候，所有元素都有一个相同名称的标记，例如<name>，但你只想要当前元素的标记。使用 utility 类的 getElements() 和 getAllElements() 方法，这些都可以做到。

```
package javagames.filesandres;

import java.io.*;
import java.util.List;
import org.w3c.dom.*;
import org.xml.sax.*;
import javagames.util.*;
import javax.xml.parsers.ParserConfigurationException;

public class LoadXMLExample {
    public LoadXMLExample() {
    }
    public void loadFile() {
        InputStream in = null;
        try {
            in = ResourceLoader.load( LoadXMLExample.class, "sample.xml",
                "/sample/xml" );
            Document document = XMLUtility.parseDocument( in );
            System.out.println( "*****");
            System.out.println( "* INSPECT");
            inspectXML( document );
            System.out.println( "*****");
            System.out.println( "* PARSE");
            parseXML( document );
        } catch( ParserConfigurationException pex ) {
            System.out.println( "ParserConfigurationException" );
            pex.printStackTrace();
        } catch( SAXException ex ) {
            System.out.println( "SAXException" );
            ex.printStackTrace();
        } catch( IOException ex ) {
            System.out.println( "IOException" );
            ex.printStackTrace();
        } finally {
            try {
                in.close();
            } catch( IOException ex ) {
            }
        }
    }
}
```



```
        }
    }
    private void inspectXML( Node node ) {
        printNode( node );
        if( node.hasAttributes() ) {
            NamedNodeMap nodeMap = node.getAttributes();
            for( int i = 0; i < nodeMap.getLength(); ++i ) {
                inspectXML( nodeMap.item( i ) );
            }
        }
        if( node.hasChildNodes() ) {
            NodeList nodeList = node.getChildNodes();
            for( int i = 0; i < nodeList.getLength(); ++i ) {
                inspectXML( nodeList.item( i ) );
            }
        }
    }
    private void printNode( Node node ) {
        System.out.print( "Type: " + getNodeType( node ) );
        System.out.print( ", Name: " + node.getNodeName() );
        String value = node.getNodeValue() == null ?
            "" : node.getNodeValue().trim();
        System.out.println( ", Value: '" + value + "'" );
    }
    private String getNodeType( Node node ) {
        switch( node.getNodeType() ) {
            case Node.ATTRIBUTE_NODE:
                return "ATTRIBUTE_NODE";
            case Node.CDATA_SECTION_NODE:
                return "CDATA_SECTION_NODE";
            case Node.COMMENT_NODE:
                return "COMMENT_NODE";
            case Node.DOCUMENT_NODE:
                return "DOCUMENT_NODE";
            case Node.ELEMENT_NODE:
                return "ELEMENT_NODE";
            case Node.ENTITY_NODE:
                return "ENTITY_NODE";
            case Node.TEXT_NODE:
                return "TEXT_NODE";
            default:
                return "Unknown";
        }
    }
}
```



```
}

private void parseXML( Document document ) {
    Element element = document.getDocumentElement();
    List<Element> elements =
        XMLUtility.getAllElements( element, "empty" );
    System.out.println( "Element: " + elements.get( 0 ).getnodeName() );
    List<Element> shorthand =
        XMLUtility.getAllElements( element, "shorthand" );
    System.out.println( "Element: " + shorthand.get( 0 ).getnodeName() );
    List<Element> text = XMLUtility.getAllElements( element, "text" );
    System.out.println( "Text: " + text.get( 0 ).getTextContent() );
    List<Element> nested = XMLUtility.getAllElements( element, "nested" );
    System.out.println( "Element: " + nested.get( 0 ).getnodeName() );
    List<Element> elementChildren =
        XMLUtility.getElements( element, "child" );
    System.out.println( "Get-Child-Count: " + elementChildren.size() );
    List<Element> child =
        XMLUtility.getElements( nested.get( 0 ), "child" );
    System.out.println( "Get-Child-Count: " + child.size() );
    List<Element> allChildren =
        XMLUtility.getAllElements( element, "child" );
    System.out.println( "Get-AllChild-Count: " + allChildren.size() );
    List<Element> nestedText =
        XMLUtility.getElements( element, "nested-text" );
    System.out.println( "nested-text: " +
        nestedText.get( 0 ).getTextContent()
    );
    List<Element> nestedChild =
        XMLUtility.getElements( nestedText.get( 0 ), "child" );
    System.out.println( "nested-text: " +
        nestedChild.get( 0 ).getNodeName() );
    List<Element> attributes =
        XMLUtility.getElements( element, "attributes" );
    Element attrElement = attributes.get( 0 );
    System.out.println(
        "attr1: " + attrElement.getAttribute( "attr1" ) );
    System.out.println(
        "attr2: " + attrElement.getAttribute( "attr2" ) );
    List<Element> specialChars =
        XMLUtility.getElements( element, "special-chars" );
    System.out.println( "special-chars: " +
        specialChars.get( 0 ).getTextContent() );
    List<Element> cdata = XMLUtility.getElements( element, "cdata" );
```



```
        System.out.println( "cdata: " + cdata.get( 0 ).getTextContent() );
    }
    public static void main( String[] args ) {
        new LoadXMLExample().loadFile();
    }
}
```

关于使用 XML 文件，还有很多的信息，如果你感觉到这方面的知识不是很牢固，不必为此担心。在本书第 16 章和第 17 章中，完整的游戏将使用从位于类路径上的存档文件中提取出的 XML 文件，来创建和加载多边形。因此，可以耐心地等待一下。如果游戏编程很容易，那谁都能干这活儿了。

## 9.8 资源和延伸阅读

Roubtsov, Vladimir, “Smartly Load Your Properties,” 2003, <http://www.javaworld.com/javaqa/2003-08/01-qa-0808-property.html>.

“Document Object Model,” 1995, <http://docs.oracle.com/javase/tutorial/jaxp/dom/index.html>.

“How to Use JAR Files,” 1995, <http://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html>.



# 第 10 章

## 图像

Java 有各种方法来创建图像。其中一些较为常见的是  `ImageIcon`、 `BufferedImage` 和  `VolatileImage`。还有其他的一些格式和类，但是，根据本书的目的，我们只介绍  `BufferedImage` 和  `VolatileImage`。另外，由于 Java 语言已经存在很长一段时间，并且能够在多种操作系统上工作，因此一些图像格式只是为了保持向后兼容才使用，而不应该用于新的开发中。即便它们并不都是所需要的，熟悉所有不同的选择还是很好的。

### 10.1 学习 Java 中的颜色

在理解图像的不同格式之前，理解 Java 如何表示颜色是很重要的。Java 中的颜色包含红色、绿色和蓝色成分，还有一个可选的 alpha 成分，用于表示透明度。可以使用从 `0.0f` 到 `1.0f` 的浮点数来创建颜色，其中 `0.0f` 表示根本没有颜色成分，而 `1.0f` 表示该颜色成分最浓。每种颜色，不管是使用什么方法创建的，都包含一个单个的值表示红色、绿色和蓝色，还有一个 alpha 值表示一定的位数。综合起来，每种颜色的位数组合起来构成一个 32 位的单个值。但是，这个值只是颜色绘制为图像之前的表示。最终，由目标图像的格式来确定该颜色。本章后面将会介绍这些不同的格式。

```
Color( float r, float g, float b )
Color( float r, float g, float b, float a )
```

前面的方法用给定的红色、绿色和蓝色以及可选的 alpha 值构建了一个颜色，alpha 值控制对象的透明度。如果没有提供 alpha 值，它会设置为 `1.0f`，表示完全不透明（没有透明度）。

也有使用从 0-255 的整数值的构造方法。0-255 值表示可以用来表示颜色的 4 个字节中的一个。这些方法构造了和浮点数版本相同的一种颜色。



```
Color( int r, int g, int b )  
Color( int r, int g, int b, int a )
```

另一个构造方法只接受一个单个的整数。这个单个值将每个颜色的(r, g, b, a)成分包含在一起，放入一个单个的值中。

```
Color( int rgba )
```

alpha 值默认为 255 (用十六进制表示是 0xFF)。这个颜色是通过将每个整数值都位移到正确的位置来构建的。

```
int rgba = 0xFF << 24 | r << 16 | g << 8 | b;
```

考虑这个值的另一种方式是：

0xAARRGGBB

其中，每一对字母都表示该颜色对应的字节。

```
Color( int rgba, boolean hasAlpha )
```

这个构造方法允许使用一个不同的 alpha 值来产生透明度。

## 10.2 了解不同的图像类型

理解 Java 如何表示颜色，这只是迷宫的一部分。构造一个颜色时，以及使用一个 Graphics 对象将颜色绘制到图像上时，由目标图像的颜色格式最终决定写到图像上的颜色格式。

BufferedImage 类支持各种图像类型。如下的列表简单介绍了 BufferedImage 类所支持的不同的图像格式。

- TYPE\_BYTE\_BINARY——1 位、2 位或 4 位图像，带有一个 IndexColorModel 用来选择适合的颜色值。
- TYPE\_BYTE\_GRAY——每种颜色都表示为从 0 (黑色) 到 255 (白色) 的一个单个的字节。
- TYPE\_BYTE\_INDEXED——每个字节值都是颜色 256 值数组的一个索引，颜色中包括 Web 安全色和灰度值。
- TYPE\_CUSTOM——未知类型使用这个值作为图像类型。



- TYPE\_3BYTE\_BGR——每 3 个字节组表示一种颜色，其中索引 0 是蓝色，索引 1 是绿色，索引 2 是红色。
- TYPE\_4BYTE\_ABGR——每 4 个字节组表示一种颜色，其中索引 0 是 alpha，索引 1 是蓝色，索引 2 是绿色，索引 3 是红色。
- TYPE\_4BYTE\_ABGR\_PRE——这和前面的类型相同，但是，所有的颜色在存储之前都预先和 alpha 值相乘过。
- TYPE USHORT\_555\_RGB——用 5 个位表示每一种颜色，一起组成一个 16 位的值，没有 alpha 值，以此表示一幅图像，如图 10.1 所示。
- TYPE USHORT\_565\_RGB——用 5 个位表示红色和蓝色，用 6 个位表示绿色，没有 alpha 成分，如图 10.2 所示。
- TYPE USHORT\_GRAY——和 8 位灰度级别图像相同，带有一个未使用的较高的 8 位。
- TYPE\_INT\_BGR——格式为 0x00BBGGRR 的整数，没有 alpha 值。
- TYPE\_INT\_RGB——格式为 0x00RRGGBB 的整数，没有 alpha 值。
- TYPE\_INT\_ARGB——格式为 0xAARRGGBB 的整数，其中 alpha 值没有预先和 RGB 颜色值相乘。
- TYPE\_INT\_ARGB\_PRE——格式为 0xAARRGGBB 的整数，其中 alpha 值已经预先和 RGB 颜色值相乘。

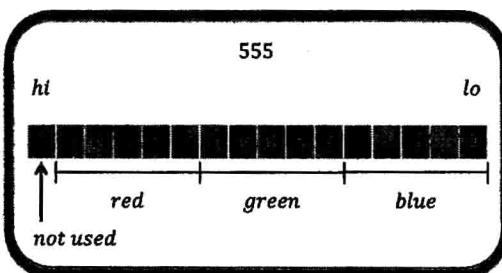


图 10.1 555 无符号 short 颜色

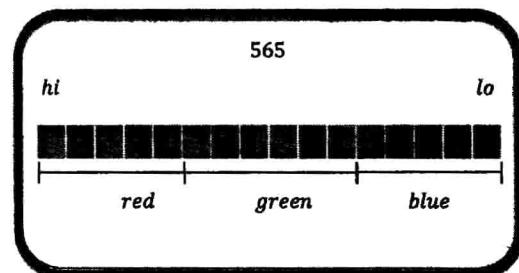


图 10.2 556 无符号 short 颜色

当保存和加载图像的时候，Java 的类中有一些非常有用的方法：

```
BufferedImage read( File input )
BufferedImage read( InputStream input )
BufferedImage read( ImageInputStream input )
BufferedImage read( URL input )
boolean write( RenderedImage img, String formatName, File output )
boolean write( RenderedImage img, String formatName,
              ImageOutputStream output )
```

```
boolean write( RenderedImage img, String formatName, OutputStream output )
```

ImageIO 类支持 JPG、GIF、BMP 和 PNG 文件。例如，要保存一个 JPG 文件，使用如下的代码：

```
ImageIO.write( image, "jpg", file );
```

图 10.3 所示的 ImageCreator 示例，位于 javagames.images 包中，它创建了一个缓冲的图像，并且以\*.gif、\*.bmp、\*.jpg 和\*.png 格式加载图像。createCustomImage()方法重新创建了一个 BufferedImage。注意，所使用的图像格式是 BufferedImage.TYPE\_INT\_RGB，并且它不支持透明度，因为并不是所有的文件格式都支持透明度。此外，还要记住，一旦完成了使用 BufferedImage 的图形对象进行绘制，别忘了释放它。

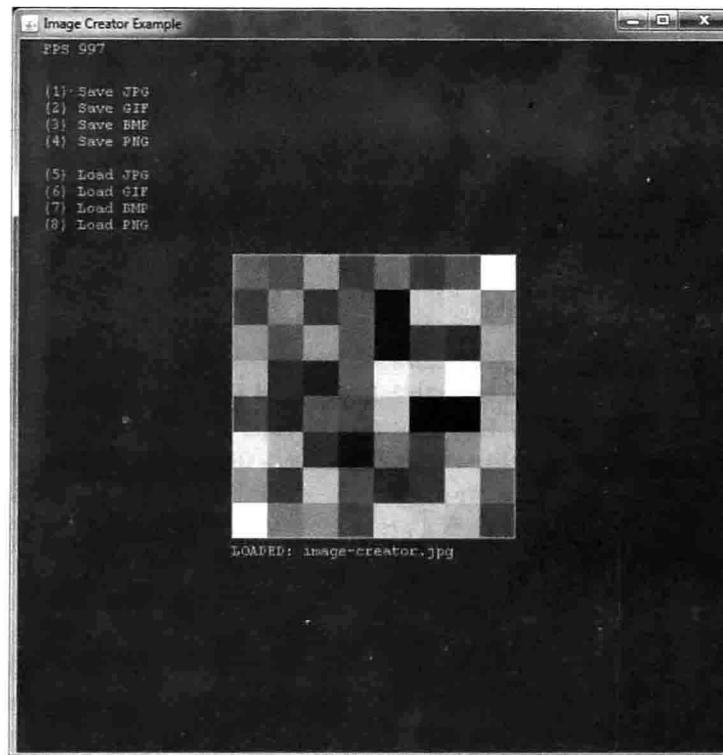


图 10.3 图像创建方法示例

```
BufferedImage img = new BufferedImage( imgWidth, imgHeight, imgFormat );
Graphics2D g = img.createGraphics();
// draw stuff here
g.dispose(); // clean up memory
```



大多数时候，在 Java 中你不用关心内存清理，但是通过一个 `BufferedImage.createGraphics()` 方法创建的 `Graphics2D` 对象是一个例外。确保当你完成使用时总是调用 `dispose()`，否则内存会很快用完。

`render()` 方法绘制了常用的帮助文本并且绘制了所创建的图像。当加载文件的时候，会显示文件信息。`createFile()` 方法使用随机颜色生成一幅新的图像，并且以给定的格式将其保存到硬盘上。`loadFile()` 方法尝试从硬盘加载已经保存的文件。如果该文件不可用，加载的文件设置为 `null`。`processInput()` 方法使用数字键 1-4 来保存文件，使用数字键 5-8 从硬盘加载文件。

```
package javagames.images;

import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.Random;
import javagames.util.SimpleFramework;
import javax.imageio.ImageIO;

public class ImageCreator extends SimpleFramework {
    private static final int IMG_WIDTH = 256;
    private static final int IMG_HEIGHT = 256;
    private static final int SQUARES = 8;
    private Random rand = new Random();
    private BufferedImage sprite;
    private String loadedFile;
    public ImageCreator() {
        appWidth = 640;
        appHeight = 640;
        appSleep = 1L;
        appTitle = "Image Creator Example";
        appBackground = Color.DARK_GRAY;
    }
    @Override
    protected void processInput( float delta ) {
        super.processInput( delta );
        if( keyboard_KeyDownOnce( KeyEvent.VK_1 ) ) {
            createFile( "jpg", "image-creator.jpg" );
        }
    }
}
```



```
if( keyboard.keyDownOnce( KeyEvent.VK_2 ) ) {
    createFile( "bmp", "image-creator.bmp" );
}
if( keyboard.keyDownOnce( KeyEvent.VK_3 ) ) {
    createFile( "gif", "image-creator.gif" );
}
if( keyboard.keyDownOnce( KeyEvent.VK_4 ) ) {
    createFile( "png", "image-creator.png" );
}
if( keyboard.keyDownOnce( KeyEvent.VK_5 ) ) {
    loadFile( "image-creator.jpg" );
}
if( keyboard.keyDownOnce( KeyEvent.VK_6 ) ) {
    loadFile( "image-creator.bmp" );
}
if( keyboard.keyDownOnce( KeyEvent.VK_7 ) ) {
    loadFile( "image-creator.gif" );
}
if( keyboard.keyDownOnce( KeyEvent.VK_8 ) ) {
    loadFile( "image-creator.png" );
}
}
private void createFile( String type, String fileName ) {
try {
    sprite = createCustomImage();
    File file = new File( fileName );
    if( !ImageIO.write( sprite, type, file ) ) {
        throw new IOException( "No '" + type + "' image writer found" );
    }
    loadedFile = "SAVED: " + fileName;
} catch( IOException ex ) {
    ex.printStackTrace();
}
}
private void loadFile( String fileName ) {
try {
    sprite = ImageIO.read( new File( fileName ) );
    loadedFile = "LOADED: " + fileName;
} catch( IOException e ) {
    e.printStackTrace();
    sprite = null;
}
}
```



```
private BufferedImage createCustomImage() {
    BufferedImage image = new BufferedImage( IMG_WIDTH, IMG_HEIGHT,
                                              BufferedImage.TYPE_INT_RGB );
    Graphics2D g2d = image.createGraphics();
    int dx = image.getWidth() / SQUARES;
    int dy = image.getHeight() / SQUARES;
    for( int i = 0; i < SQUARES; ++i ) {
        for( int j = 0; j < SQUARES; ++j ) {
            g2d.setColor( new Color( rand.nextInt() ) );
            g2d.fillRect( i*dx, j*dy, dx, dy );
        }
    }
    g2d.setColor( Color.GREEN );
    g2d.drawRect( 0, 0, image.getWidth()-1, image.getHeight()-1 );
    g2d.dispose();
    return image;
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    // draw help
    g.drawString( "", 20, 35 );
    g.drawString( "(1) Save JPG", 20, 50 );
    g.drawString( "(2) Save GIF", 20, 65 );
    g.drawString( "(3) Save BMP", 20, 80 );
    g.drawString( "(4) Save PNG", 20, 95 );
    g.drawString( "", 20, 110 );
    g.drawString( "(5) Load JPG", 20, 125 );
    g.drawString( "(6) Load GIF", 20, 140 );
    g.drawString( "(7) Load BMP", 20, 155 );
    g.drawString( "(8) Load PNG", 20, 170 );
    if( sprite != null ) {
        int x = (canvas.getWidth() - sprite.getWidth()) / 2;
        int y = (canvas.getHeight() - sprite.getHeight()) / 2;
        g.drawImage( sprite, x, y, null );
        g.drawString( loadedFile, x, y + sprite.getHeight() + 15 );
    } else {
        g.drawString(
        "ERROR - File Not Found!",
        canvas.getWidth() / 3,
        canvas.getHeight() / 3
    );
    }
}
```

```
    }
    public static void main( String[] args ) {
        launchApp( new ImageCreator() );
    }
}
```

## 10.3 执行颜色插值

颜色插值（color interpolation）是从一种颜色到另一种颜色的混合。此过程的第一部分是线性插值，并且由于公式是线性的，因此数学上并不复杂。

这个示例涉及很多有趣的概念。你可以把颜色混合背后的数学，应用于需要在一个距离之间插值的任何数值。使用这一技术来捕获单个的像素值，使你能够执行定制图像插值。此外，学习如何手动执行一种图像效果，而不依赖于隐藏的算法，这总是很有趣的事情。

给定任意两个点，如图 10.4 所示，线条上的任何 x 值对应一个 y 值，如图 10.5 所示。

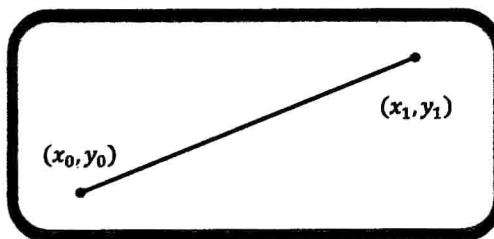


图 10.4 线段

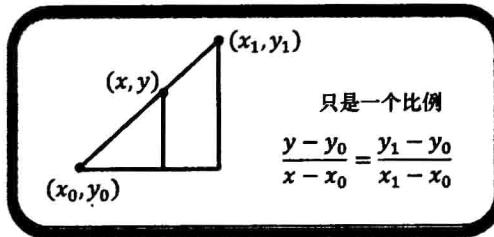


图 10.5 只是一个比例



使用如下的比例乘积求出 y:

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0}$$

如果让  $x_0 = 0$  且  $x_1 = 1$ :

$$y = y_0 + (x - 0)(y_1 - y_0 / 1 - 0)$$

$$y = y_0 + x(y_1 - y_0 / 1)$$

$$y = y_0 + x(y_1 - y_0)$$

这就是所谓的直线参数方程:

$$P = P_0 + t(P_1 - P_0)$$

这是有意义的，因为这个方程式在从  $P_0$  到  $P_1$  的线段之间，插入了一个点。

然而，混合颜色要更加复杂一些。给定一个起始颜色和一个结束颜色，以及两个点之间的像素距离，颜色就可以混合了，但是每个颜色成分必须各自混合，并且在每个阶段组合以创建特定的颜色。例如，从红色到绿色进行混合，每个颜色成分必须插值。

如图 10.6 所示，红色从 255 到 0 混合，绿色从 0 到 255 混合，蓝色没有任何操作。

当使用插值方程式混合颜色时，未知的值  $y$  是任何特定的点的颜色。要混合一个单个的颜色成分，例如，红色从 255 到 0 跨越 20 个像素，代码应该如下所示：

```
int y0 = 255; int y1 = 0;
int x0 = 0; int x1 = 20;
for( int x = x0; x < x1; ++x ) {
    int y = y0 + (x - x0) * (y1 - y0) / (x1 - x0);
    System.out.println(y);
}
```

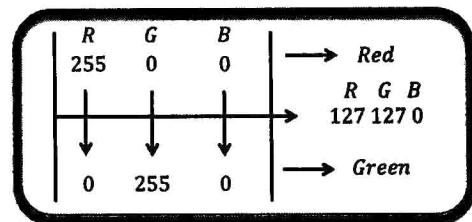


图 10.6 颜色插值

创建单个颜色并在颜色之间混合，有助于在 `BufferedImage` 中设置单个的像素值。如下的代码用来获取对一个 `BufferedImage` 像素数据的访问。

```
BufferedImage img = new BufferedImage( 400, 400,
    BufferedImage.TYPE_INT_ARGB );
WritableRaster raster = img.getRaster();
```