CHAPTER

18

第 18 章
合金弹头

本章要点

- 开发射击类游戏的基本方法
- 游戏的界面分解和分析
- 游戏界面组件的分析和实现
- 怪物的移动和发射子弹
- 实现角色移动、跳跃、发射子弹等行为
- 检测子弹是否命中目标
- 实现游戏的绘图工具类
- 管理游戏资源
- 继承 SurfaceView 实现游戏主组件
- 掌握 SurfaceView 的绘图机制
- 使用多线程实现游戏动画
- 实现游戏的 Activity

本章将会介绍一款经典的射击类游戏：合金弹头，合金弹头游戏要求玩家控制自己的角色不断前行，并发射子弹去射击沿途遇到的各种怪物，同时还要躲避怪物发射的子弹。当然，由于完整的合金弹头游戏涉及的地图场景很多，而且怪物种类也很多，因此本章对该游戏进行了适当的简化。本游戏只实现了一个地图场景，并将之设置为无限地图，并且只实现了 3 种类型的怪物，但只要读者真正掌握了本章的内容，当然也就能从一个地图扩展为多个地图；也可以从 3 种类型的怪物扩展出多种类型的怪物了。

对于 Android 学习者来说，学习开发这个小程序难度适中，而且能很好地培养学习者的学习乐趣。开发者需要从程序员的角度来看待玩家面对的游戏界面，游戏界面上的每个怪物、每个角色、每颗子弹、每个能与玩家交互的东西，都应该在程序中通过类的形式来定义它们，这样才能更好地用面向对象的方式来解决问题。

18.1 合金弹头游戏简介

合金弹头是一款早期风靡一时的射击类游戏，这款游戏的节奏感非常强，让大部分男同胞充满童年回忆。实际上，现在也非常流行将一些早期游戏移植到手机平台上，以充分满足广大玩家的怀旧情怀。

图 18.1 显示了合金弹头的游戏界面。

图 18.1 合金弹头

这款游戏的玩法很简单，玩家控制角色不断地向右前进，角色可通过跳跃来躲避敌人（也可统称为怪物）发射的子弹和地上的炸弹，玩家也可控制角色发射子弹来打死右边的各种敌人。对于完整的合金弹头游戏，它会包含很多“关卡”，每个关卡都是一种地图，每个关卡都包含了大量不同的怪物。但本章由于篇幅关系，只做了一种地图，而且这种地图是“无限循环”的——也就是说，玩家只能一直向前去消灭不同的怪物，无法实现“通关”。

提示：

如果读者有兴趣把这款游戏改成包含很多关卡的游戏，那就需要准备大量的背景图片。然后为不同的地图加载不同的背景图片，让地图不要无限循环即可，并为不同地图使用不同的怪物。

18.2 开发游戏界面组件

在开发游戏之前，首先需要从程序员的角度来分析游戏界面，并逐步实现游戏界面上的各种组件。

18.2.1 游戏界面分析

对于图 18.1 所示的游戏界面，从普通玩家的角度来看，他会看到游戏界面上有受玩家控制移动、跳跃、发射子弹的角色，还有不断发射子弹的敌人、地上有炸弹、天空中有正在爆炸的飞机……乍看上去会给人眼花缭乱的感觉。

如果从程序员的角度来看，游戏界面大致可分为如下组件。

- **游戏背景**：只是一张静止的图片。
- **角色**：该角色可以站立、走动、跳跃、射击。
- **怪物**：怪物类代表了游戏界面上所有的敌人，包括拿枪的敌人、地上的炸弹、天空中的飞机……虽然这些怪物的图片不同、发射的子弹不同，攻击力也可能不同，但这些只是实例与实例之间的差异，因此程序只要为怪物定义一个类即可。
- **子弹**：不管是角色发射的子弹还是怪物发射的子弹，都可归纳为子弹类。虽然不同子弹的图片不同，攻击力不同，但这些只是实例与实例之间的差异，因此程序只要为子弹定义一个类即可。

从上面介绍不难看出，开发这款游戏，主要就是实现上面的角色、怪物和子弹3个类。

18.2.2 实现“怪物”类

由于不同怪物之间会存在各种差异，那么此处就需要为怪物类定义相应的实例变量来记录这些差异。不同怪物之间可能存在如下差异。

- 怪物的类型。
- 代表怪物位置的*X*、*Y*坐标。
- 标识怪物是否已经死亡的旗标。
- 绘制怪物图片左上角的*X*、*Y*坐标。
- 绘制怪物图片右下角的*X*、*Y*坐标。
- 怪物发射的所有子弹。（有的怪物不会发射子弹）
- 怪物未死亡时所有的动画帧图片和怪物死亡时所有的动画帧图片。

提示：

本程序并未把怪物的所有动画帧图片直接保存在怪物实例中，本程序将会专门使用一个工具类来保存所有角色、怪物的所有动画帧图片。

为了让游戏界面的角色、怪物都能“动起来”，程序的实现思路是这样的：程序会专门启动一条独立的线程，这条线程负责控制角色、怪物不断地更换新的动画帧图片——因此程序需要为怪物增加一个成员变量来记录当前游戏界面正在绘制怪物动画的第几帧，而负责动画的独立线程只要不断地调用怪物的绘制方法即可——实际上该绘制方法每次只是绘制一张静态图片（这张静态图片是怪物动画的其中一帧）。

下面是怪物类的成员变量部分。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 定义代表怪物类型的常量（如果程序还需要增加更多怪物，只需在此处添加常量即可）
public static final int TYPE_BOMB = 1;
public static final int TYPE_FLY = 2;
public static final int TYPE_MAN = 3;
// 定义怪物类型的成员变量
private int type = TYPE_BOMB;
// 定义怪物X、Y坐标的成员变量
private int x = 0;
private int y = 0;
// 定义怪物是否已经死亡的旗标
private boolean isDie = false;
// 绘制怪物图片左上角的X坐标
private int startX = 0;
```

```
    // 绘制怪物图片左上角的 Y 坐标
    private int startY = 0;
    // 绘制怪物图片右下角的 X 坐标
    private int endX = 0;
    // 绘制怪物图片右下角的 Y 坐标
    private int endY = 0;
    // 该变量用于控制动画刷新的速度
    int drawCount = 0;
    // 定义当前正在绘制怪物动画的第几帧的变量
    private int drawIndex = 0;
    // 用于记录死亡动画只绘制一次，不需要重复绘制
    // 每当怪物死亡时，该变量会被初始化为等于死亡动画的总帧数
    // 当怪物的死亡动画帧播放完成时，该变量的值变为 0
    private int dieMaxDrawCount = Integer.MAX_VALUE;
    // 定义怪物射出的子弹
    private List<Bullet> bulletList = new ArrayList<>();
```

上面的成员变量即可记录该怪物实例的各种状态。实际上以后程序要升级，比如为怪物增加更多的特征，如怪物可以拿不同的武器，怪物可以穿不同的衣服，怪物可以具有不同的攻击力……这些都可考虑定义成怪物的成员变量。

下面是怪物类的构造器，该构造器只要传入一个 type 参数即可，该 type 参数告诉系统，该怪物是哪种类型的怪物。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
    public Monster(int type)
    {
        this.type = type;
        // -------下面代码根据怪物类型来初始化怪物 X、Y 坐标------
        // 如果怪物是炸弹（TYPE_BOMB）或敌人（TYPE_MAN）
        // 怪物的 Y 坐标与玩家控制的角色的 Y 坐标相同
        if (type == TYPE_BOMB || type == TYPE_MAN)
        {
            y = Player.Y_DEFALUT;
        }
        // 如果怪物是飞机，根据屏幕高度随机生成怪物的 Y 坐标
        else if (type == TYPE_FLY)
        {
            y = ViewManager.SCREEN_HEIGHT * 50 / 100
                - Util.rand((int) (ViewManager.scale * 100));
        }
        // 随机计算怪物的 X 坐标。
        x = ViewManager.SCREEN_WIDTH + Util.rand(ViewManager.SCREEN_WIDTH >> 1)
            - (ViewManager.SCREEN_WIDTH >> 2);
    }
```

从上面的粗体字代码可以看出，程序在创建怪物实例时，不仅负责初始化怪物的 type 成员变量，还会根据怪物类型来设置怪物的 *X*、*Y* 坐标。

- 如果怪物是炸弹和拿枪的人（都在地面上），那么它们的 *Y* 坐标与角色默认的 *Y* 坐标（在地面上）相同。如果怪物是飞机，那么怪物的 *Y* 坐标是随机计算的。
- 不管什么怪物，它的 *X* 坐标都是随机计算的。

上面程序中还用到一个 Util 工具类，该工具类仅仅包含一个计算随机数的方法。下面是该工具类的代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Util.java

```
    public class Util
    {
```

```
    public static Random random = new Random();
    // 返回一个 0～range 的随机数
    public static int rand(int range)
    {
        // 如果 range 为 0，直接返回 0
        if (range == 0)
            return 0;
        // 获取一个 0～range 之间的随机数
        return Math.abs(random.nextInt() % range);
    }
}
```

前面已经介绍了绘制怪物动画的思路：程序将会采用后台线程来控制不断地绘制怪物动画的下一帧，但实际上每次绘制的只是怪物动画的某一帧。下面是绘制怪物的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 绘制怪物的方法
public void draw(Canvas canvas)
{
    if (canvas == null)
    {
        return;
    }
    switch (type)
    {
        case TYPE_BOMB:
            // 死亡的怪物用死亡图片
            drawAni(canvas, isDie ? ViewManager.bomb2Image : ViewManager.bombImage);
            break;
        case TYPE_FLY:
            // 死亡的怪物用死亡图片
            drawAni(canvas, isDie ? ViewManager.flyDieImage : ViewManager.flyImage);
            break;
        case TYPE_MAN:
            // 死亡的怪物用死亡图片
            drawAni(canvas, isDie ? ViewManager.manDieImage : ViewManager.manImgae);
            break;
        default:
            break;
    }
}
// 根据怪物的动画帧图片来绘制怪物动画
public void drawAni(Canvas canvas, Bitmap[] bitmapArr)
{
    if (canvas == null)
    {
        return;
    }
    if (bitmapArr == null)
    {
        return;
    }
    // 如果怪物已经死亡，且没有播放过死亡动画
    // (dieMaxDrawCount 等于初始值表明未播放过死亡动画)
    if (isDie && dieMaxDrawCount == Integer.MAX_VALUE)
    {
        // 将 dieMaxDrawCount 设置为与死亡动画的总帧数相等
        dieMaxDrawCount = bitmapArr.length;  // ⑤
    }
    drawIndex = drawIndex % bitmapArr.length;
    // 获取当前绘制的动画帧对应的位图
    Bitmap bitmap = bitmapArr[drawIndex];  // ①
```

```
        if (bitmap == null || bitmap.isRecycled())
        {
            return;
        }
        int drawX = x;
        // 对绘制怪物动画帧位图的 X 坐标进行微调
        if (isDie)
        {
            if (type == TYPE_BOMB)
            {
                drawX = x - (int) (ViewManager.scale * 50);
            }
            else if (type == TYPE_MAN)
            {
                drawX = x + (int) (ViewManager.scale * 50);
            }
        }
        // 对绘制怪物动画帧位图的 Y 坐标进行微调
        int drawY = y - bitmap.getHeight();
        // 绘制怪物动画帧的位图
        Graphics.drawMatrixImage(canvas, bitmap, 0, 0, bitmap.getWidth(),
            bitmap.getHeight(), Graphics.TRANS_NONE, drawX, drawY, 0,
                Graphics.TIMES_SCALE);
        startX = drawX;
        startY = drawY;
        endX = startX + bitmap.getWidth();
        endY = startY + bitmap.getHeight();
        drawCount++;
        // 后面的 6、4 用于控制人、飞机发射子弹的速度
        if (drawCount >= (type == TYPE_MAN ? 6 : 4))  // ③
        {
            // 如果怪物是人，只在第 3 帧才发射子弹
            if (type == TYPE_MAN && drawIndex == 2)
            {
                addBullet();
            }
            // 如果怪物是飞机，只在最后一帧才发射子弹
            if (type == TYPE_FLY && drawIndex == bitmapArr.length - 1)
            {
                addBullet();
            }
            drawIndex++;  // ②
            drawCount = 0;  // ④
        }
        // 每播放死亡动画的一帧，dieMaxDrawCount 减 1
        // 当 dieMaxDrawCount 等于 0 时，表明死亡动画播放完成，MonsterManger 会删除该怪物
        if (isDie)
        {
            dieMaxDrawCount--;  // ⑥
        }
        // 绘制子弹
        drawBullet(canvas);
    }
```

上面代码包含两个方法，其中 draw(Canvas canvas)方法只是简单地对怪物类型进行了判断，并针对不同怪物类型使用不同的怪物动画。

draw(Canvas canvas)方法总是调用 drawAni(Canvas canvas, Bitmap[] bitmapArr)方法来绘制怪物，调用后者时根据怪物类型的不同、怪物是否死亡将会传入不同的位图数组——每个位图数组就代表一组动画帧的所有位图。

drawAni()方法中的①号粗体字代码就是根据 drawIndex 来获取当前帧对应的位图，而程序

执行 drawAni()方法时，②号粗体字代码可以控制 drawIndex 自加一次，这样即可保证下次调用 drawAni()方法时就会绘制动画的下一帧。

drawAni()方法还涉及一个 drawCount 变量，这个变量是控制动画刷新速度的计数器——程序在③号粗体字代码处进行了控制：只有当 drawCount 计数器的值大于 6（对于其他类型的怪物，该值为 4）时才会调用 drawIndex++，这意味着当怪物类型是 TYPE_MAN 时，drawAni()方法至少调用 6 次之后才会将 drawIndex 加 1（即绘制下一帧位图）；当怪物是其他类型时，drawAni()方法至少调用 4 次之后才会将 drawIndex 加 1（即绘制下一帧位图）——这是因为程序中控制动画刷新的线程的刷新频率是固定的，即如后台线程控制每隔 40ms 调用一次怪物的 drawAni()方法，但如果每隔 40ms 就更新一次动画帧的话，那么游戏界面上的所有怪物“动”的速度都是一样的（每隔 40ms 刷新一次），而且它们都动得非常快。为了解决这个问题，程序就需要使用 drawCount 来控制不同怪物实际每隔多少毫秒更新一次动画帧。对于上面代码来说，如果怪物类型是 TYPE_MAN，则只有当 drawCount 计数器大于 6 时，才会更新一次动画帧，这意味着实际上每隔 240ms 才会更新一次动画帧；如果是其他类型的怪物，那么只有当 drawCount 计数器大于 4 时，才会更新一次动画帧，这意味着实际上每隔 160ms 才会更新一次动画帧。

提示：

如果游戏中还有更多类型的怪物，且这些怪物的动画帧具有不同的更新速度，那么程序还需要进行更细致的判断。

drawAni()方法还涉及一个 dieMaxDrawCount 变量，这个变量用于控制怪物的死亡动画只会被绘制一次——在怪物临死之前，程序都必须播放怪物的死亡动画，该动画播放完成，该怪物就应该从地图上删除。当怪物已经死亡（isDie 为真）且还未绘制死亡动画的任何帧时（dieMaxDrawCount 等于初始值），程序在⑤号粗体字代码处将 dieMaxDrawCount 设置为与死亡动画的总帧数相等，程序每次调用 drawAni()方法时，⑥号粗体字代码都会把 dieMaxDrawCount 减 1，当 dieMaxDrawCount 变为 0 时，表明该怪物的死亡动画的所有帧都绘制完成，接下来程序即可将该怪物从地图上删除了——在后面的 MonsterManager 类中将会看到程序根据怪物的 dieMaxDrawCount 为 0 来从地图上删除怪物的代码。

Monster 还包含了 startX、startY、endX、endY 四个变量，这四个变量就代表了怪物当前帧所覆盖的矩形区域，因此，如果程序需要判断该怪物是否被子弹打中，只要子弹出现在该矩形区域内，即可判断怪物被子弹打中了。下面是判断怪物是否被子弹打中的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 判断怪物是否被子弹打中的方法
public boolean isHurt(int x, int y)
{
    return x >= startX && x <= endX
        && y >= startY && y <= endY;
}
```

接下来为怪物实现发射子弹的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 根据怪物类型获取子弹类型，不同怪物发射不同的子弹
// return 0 代表这种怪物不发射子弹
public int getBulletType()
{
    switch (type)
```

```
        {
            case TYPE_BOMB:
                return 0;
            case TYPE_FLY:
                return Bullet.BULLET_TYPE_3;
            case TYPE_MAN:
                return Bullet.BULLET_TYPE_2;
            default:
                return 0;
        }
    }
    // 定义发射子弹的方法
    public void addBullet()
    {
        int bulletType = getBulletType();
        // 如果没有子弹
        if (bulletType <= 0)
        {
            return;
        }
        // 计算子弹的 X、Y 坐标
        int drawX = x;
        int drawY = y - (int) (ViewManager.scale * 60);
        // 如果怪物是飞机，重新计算飞机发射的子弹的 Y 坐标
        if (type == TYPE_FLY)
        {
            drawY = y - (int) (ViewManager.scale * 30);
        }
        // 创建子弹对象
        Bullet bullet = new Bullet(bulletType, drawX, drawY, Player.DIR_LEFT);
        // 将子弹添加到该怪物发射的子弹集合中
        bulletList.add(bullet);
    }
```

怪物发射子弹的方法是 addBullet()，该方法需要调用 getBulletType()方法来判断该怪物所发射的子弹类型（不同怪物可能需要发射不同的子弹），如果 getBulletType()方法返回 0，即代表这种怪物不发射子弹。

一旦确定了这种怪物发射子弹的类型，程序就可根据不同怪物计算子弹的初始 *X*、*Y* 坐标——基本上，子弹的 *X*、*Y* 坐标保持与怪物当前的 *X*、*Y* 坐标相同，再进行适当微调即可。程序最后两行粗体字代码创建了一个 Bullet 对象（子弹实例），并将新的 Bullet 对象添加到 bulletList 集合中。

当怪物发射了子弹之后，程序还需要绘制该怪物的所有子弹。下面是绘制怪物发射的所有子弹的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
    // 更新角色的位置：将角色的 X 坐标减少 shift 距离（角色左移）
    // 更新所有子弹的位置：将所有子弹的 X 坐标减少 shift 距离（子弹左移）
    public void updateShift(int shift)
    {
        x -= shift;
        for (Bullet bullet : bulletList)
        {
            if (bullet == null)
            {
                continue;
            }
            bullet.setX(bullet.getX() - shift);
        }
    }
```

```
    // 绘制子弹的方法
    public void drawBullet(Canvas canvas)
    {
        // 定义一个 deleteList 集合，该集合保存所有需要删除的子弹
        List<Bullet> deleteList = new ArrayList<>();
        Bullet bullet = null;
        for (int i = 0; i < bulletList.size(); i++)
        {
            bullet = bulletList.get(i);
            if (bullet == null)
            {
                continue;
            }
            // 如果子弹已经越过屏幕
            if (bullet.getX() < 0 || bullet.getX() > ViewManager.SCREEN_WIDTH)
            {
                // 将需要清除的子弹添加到 deleteList 集合中
                deleteList.add(bullet);
            }
        }
        // 删除所有需要清除的子弹
        bulletList.removeAll(deleteList);  // ⑦
        // 定义代表子弹的位图
        Bitmap bitmap;
        // 遍历该怪物发射的所有子弹
        for (int i = 0; i < bulletList.size(); i++)
        {
            bullet = bulletList.get(i);
            if (bullet == null)
            {
                continue;
            }
            // 获取子弹对应的位图
            bitmap = bullet.getBitmap();
            if (bitmap == null)
            {
                continue;
            }
            // 子弹移动
            bullet.move();
            // 绘制子弹的位图
            Graphics.drawMatrixImage(canvas, bitmap, 0, 0, bitmap.getWidth(),
                bitmap.getHeight(), bullet.getDir() == Player.DIR_RIGHT ?
                Graphics.TRANS_MIRROR : Graphics.TRANS_NONE,
                bullet.getX(), bullet.getY(), 0, Graphics.TIMES_SCALE);
        }
    }
```

上面程序中的 updateShift(int shift)方法负责将怪物所有的子弹全部左移 shift 距离，这是因为界面上角色会不断地向右移动，角色会产生一个 shift 偏移，所以程序就需要将怪物（包括它的所有子弹）全部左移 shift 距离，这样才会产生逼真的效果。

上面程序中的粗体字代码使用 deleteList 集合收集所有越过屏幕的子弹，然后⑦号粗体字代码负责删除 deleteList 集合包含的所有子弹——这样即可把所有越过屏幕的子弹删除掉。

接下来程序采用循环遍历了该怪物发射的所有子弹，先获取子弹对应的位图，然后调用子弹的 move()方法控制子弹移动。上面方法中的最后一行粗体字代码负责绘制子弹位图。

Monster 类还需要定义一个方法，用于判断怪物的子弹是否打中角色，如果打中角色，则删除该子弹。下面是该方法的代码。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

```
// 判断子弹是否与玩家控制的角色碰撞（判断子弹是否打中角色）
public void checkBullet()
{
    // 定义一个 delBulletList 集合，该集合保存打中角色的子弹，它们将要被删除
    List<Bullet> delBulletList = new ArrayList<>();
    // 遍历所有子弹
    for (Bullet bullet : bulletList)
    {
        if (bullet == null || !bullet.isEffect())
        {
            continue;
        }
        // 如果玩家控制的角色被子弹打到
        if (GameView.player.isHurt(bullet.getX(), bullet.getX()
            , bullet.getY(), bullet.getY()))
        {
            // 子弹设为无效
            bullet.setEffect(false);
            // 将玩家的生命值减 5
            GameView.player.setHp(GameView.player.getHp() - 5);
            // 将子弹添加到 delBulletList 集合中
            delBulletList.add(bullet);
        }
    }
    // 删除所有打中角色的子弹
    bulletList.removeAll(delBulletList);
}
```

18.2.3 实现怪物管理类

由于游戏界面上会出现很多个怪物，因此程序需要额外定义一个怪物管理类来专门负责管理怪物的随机产生、死亡等行为。

为了有效地管理游戏界面上所有活着的怪物和已死的怪物（保存已死的怪物是为了绘制死亡动画），为怪物管理类定义如下成员变量。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 保存所有死掉的怪物，保存它们是为了绘制死亡动画，绘制完后清除这些怪物
public static final List<Monster> dieMonsterList = new ArrayList<>();
// 保存所有活着的怪物
public static final List<Monster> monsterList = new ArrayList<>();
```

接下来在怪物管理类中定义一个随机生成怪物的工具方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 随机生成并添加怪物的方法
public static void generateMonster()
{
    if (monsterList.size() < 3 + Util.rand(3))
    {
        // 创建新怪物
        Monster monster = new Monster(1 + Util.rand(3));
        monsterList.add(monster);
    }
}
```

前面已经指出，当玩家控制游戏界面的角色不断地向右移动时，程序界面上的所有怪物、怪物的子弹都必须不断地左移，因此程序需要在 MonsterManager 类中定义一个控制所有怪物

及其子弹不断左移的方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 更新怪物与子弹的坐标的方法
public static void updatePosistion(int shift)
{
    Monster monster = null;
    // 定义一个集合，保存所有将要被删除的怪物
    List<Monster> delList = new ArrayList<>();
    // 遍历怪物集合
    for (int i = 0; i < monsterList.size(); i++)
    {
        monster = monsterList.get(i);
        if (monster == null)
        {
            continue;
        }
        // 更新怪物、怪物所有子弹的位置
        monster.updateShift(shift);  // ①
        // 如果怪物的 X 坐标越界，将怪物添加到 delList 集合中
        if (monster.getX() < 0)
        {
            delList.add(monster);
        }
    }
    // 删除 delList 集合中的所有怪物
    monsterList.removeAll(delList);
    delList.clear();
    // 遍历所有已死的怪物的集合
    for (int i = 0; i < dieMonsterList.size(); i++)
    {
        monster = dieMonsterList.get(i);
        if (monster == null)
        {
            continue;
        }
        // 更新怪物、怪物所有子弹的位置
        monster.updateShift(shift);  // ②
        // 如果怪物的 X 坐标越界，将怪物添加到 delList 集合中
        if (monster.getX() < 0)
        {
            delList.add(monster);
        }
    }
    // 删除 delList 集合中的所有怪物
    dieMonsterList.removeAll(delList);
    // 更新玩家控制的角色的子弹坐标
    GameView.player.updateBulletShift(shift);
}
```

上面程序中的①号粗体字代码处于循环体之内，该循环将会控制把所有活着的怪物及其子弹全部都左移 shift 距离，如果移动之后的怪物的 *X* 坐标超出了屏幕范围，程序就会清除该怪物；②号粗体字代码同样处于循环体之内，②号代码的处理方式与①号代码的处理方式几乎是一样的，只是②号代码负责处理的是界面上已死的怪物。

上面程序中的最后一行粗体字代码则负责将玩家发射的所有子弹都左移 shift 距离——这也是必要的，原因与怪物发射的子弹都需要左移 shift 距离一样。

接下来要为 MonsterManager 实现一个新的方法，该方法可用于检查界面上的怪物是否将要死亡，将要死亡的怪物将会从 monsterList 集合中删除，并添加到 dieMonsterList 集合中，然

后程序将会负责绘制它们的死亡动画。

下面为 MonsterManager 类增加一个 checkMonster()方法。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 检查怪物是否将要死亡的方法
public static void checkMonster()
{
    // 获取玩家发射的所有子弹
    List<Bullet> bulletList = GameView.player.getBulletList();
    if (bulletList == null)
    {
        bulletList = new ArrayList<>();
    }
    Monster monster = null;
    // 定义一个 delList 集合，用于保存将要死亡的怪物
    List<Monster> delList = new ArrayList<>();
    // 定义一个 delBulletList 集合，用于保存所有将要被删除的子弹
    List<Bullet> delBulletList = new ArrayList<>();
    // 遍历所有怪物
    for (int i = 0; i < monsterList.size(); i++)
    {
        monster = monsterList.get(i);
        if (monster == null)
        {
            continue;
        }
        // 如果怪物是炸弹
        if (monster.getType() == Monster.TYPE_BOMB)
        {
            // 角色被炸弹炸到
            if (GameView.player.isHurt(monster.getStartX()
                , monster.getEndX(), monster.getStartY(), monster.getEndY()))
            {
                // 将怪物设置为死亡状态
                monster.setDie(true);
                // 播放爆炸音效
                ViewManager.soundPool.play(
                    ViewManager.soundMap.get(2), 1, 1, 0, 0, 1);
                // 将怪物（爆炸的炸弹）添加到 delList 集合中
                delList.add(monster);
                // 玩家控制的角色的生命值减 10
                GameView.player.setHp(GameView.player.getHp() - 10);
            }
            continue;
        }
        // 对于其他类型的怪物，则需要遍历角色发射的所有子弹
        // 只要任何一个子弹打中怪物，即可判断怪物即将死亡
        for (Bullet bullet : bulletList)
        {
            if (bullet == null || !bullet.isEffect())
            {
                continue;
            }
            // 如果怪物被角色的子弹打到
            if (monster.isHurt(bullet.getX(), bullet.getY()))
            {
                // 将子弹设为无效
                bullet.setEffect(false);
                // 将怪物设为死亡状态
                monster.setDie(true);
                // 如果怪物是飞机
```

```
                if(monster.getType() == Monster.TYPE_FLY)
                {
                    // 播放爆炸音效
                    ViewManager.soundPool.play(
                        ViewManager.soundMap.get(2), 1, 1, 0, 0, 1);
                }
                // 如果怪物是人
                if(monster.getType() == Monster.TYPE_MAN)
                {
                    // 播放惨叫音效
                    ViewManager.soundPool.play(
                        ViewManager.soundMap.get(3), 1, 1, 0, 0, 1);
                }
                // 将怪物（被子弹打中的怪物）添加到 delList 集合中
                delList.add(monster);
                // 将打中怪物的子弹添加到 delBulletList 集合中
                delBulletList.add(bullet);
            }
        }
        // 将 delBulletList 包含的所有子弹从 bulletList 集合中删除
        bulletList.removeAll(delBulletList);
        // 检查怪物子弹是否打到角色
        monster.checkBullet();
    }
    // 将已死亡的怪物（保存在 delList 集合中）添加到 dieMonsterList 集合中
    dieMonsterList.addAll(delList);
    // 将已死亡的怪物（保存在 delList 集合中）从 monsterList 集合中删除
    monsterList.removeAll(delList);
}
```

上面这个方法的判断逻辑非常简单，程序把怪物分为两类进行处理。

- 如果怪物是地上的炸弹，只要炸弹炸到角色，炸弹也就即将死亡。上面程序中第一行粗体字代码处理了怪物是炸弹的情形。
- 对于其他类型的怪物，程序则需要遍历角色发射的子弹，只要任意一颗子弹打中了怪物，即可判断怪物即将死亡。上面程序中第二行粗体字代码正是遍历玩家所发射的子弹的代码。

最后 MonsterManager 还需要定义一个绘制所有怪物的方法。该方法的实现逻辑也非常简单，程序只要分别遍历该类的 dieMonsterList 和 monsterList 集合，并将集合中所有怪物绘制出来即可。对于 dieMonsterList 集合中的怪物，它们都是将要死亡的怪物，因此只要将它们所有的死亡动画帧都绘制一次，接下来就应该清除这些怪物了——Monster 实例的 dieMaxDrawCount 成员变量为 0 时就代表所有死亡动画帧都绘制了一次。

下面是该 drawMonster()方法的代码，该方法就负责绘制所有怪物。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

```
// 绘制所有怪物的方法
public static void drawMonster(Canvas canvas)
{
    Monster monster = null;
    // 遍历所有活着的怪物，绘制活着的怪物
    for (int i = 0; i < monsterList.size(); i++)
    {
        monster = monsterList.get(i);
        if (monster == null)
        {
            continue;
        }
        // 绘制怪物
```

```
            monster.draw(canvas);
        }
        List<Monster> delList = new ArrayList<>();
        // 遍历所有已死亡的怪物，绘制已死亡的怪物
        for (int i = 0; i < dieMonsterList.size(); i++)
        {
            monster = dieMonsterList.get(i);
            if (monster == null)
            {
                continue;
            }
            // 绘制怪物
            monster.draw(canvas);
            // 当怪物的 getDieMaxDrawCount()返回 0 时，则表明该怪物已经死亡
            // 且该怪物的死亡动画所有帧都播放完成，将它们彻底删除
            if (monster.getDieMaxDrawCount() <= 0)  // ③
            {
                delList.add(monster);
            }
        }
        dieMonsterList.removeAll(delList);
    }
```

上面程序中的第一行粗体字代码负责遍历所有活着的怪物，并将它们绘制出来；第二行粗体字代码则负责遍历所有将要死亡的怪物，并将它们绘制出来。程序中③号粗体字代码检查该怪物的 dieMaxDrawCount 是否为 0，如果为 0，则表明该怪物已死亡、且该怪物的死亡动画所有帧都播放完成，应该将它们彻底删除。

18.2.4 实现“子弹”类

本游戏的子弹类比较简单，本游戏中的子弹不会产生爆炸效果。对子弹的处理思路是：只要子弹打中目标，子弹就会自动消失。正因为本游戏的子弹类比较简单，因此子弹类只需要定义如下属性即可。

- 子弹的类型。
- 子类的 *X*、*Y* 坐标。
- 子弹的射击方向（向左或向右）。
- 子弹在垂直方向（*Y* 方向）上的加速度。

基于上面分析，程序为 Bullet 类定义了如下成员变量。

程序清单：codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

```
// 定义代表子弹类型的常量（如果程序还需要增加更多子弹，只需在此处添加常量即可）
public static final int BULLET_TYPE_1 = 1;
public static final int BULLET_TYPE_2 = 2;
public static final int BULLET_TYPE_3 = 3;
public static final int BULLET_TYPE_4 = 4;
// 定义子弹的类型
private int type;
// 子弹的 X、Y 坐标
private int x;
private int y;
// 定义子弹射击的方向
private int dir;
// 定义子弹在 Y 方向上的加速度
private int yAccelate = 0;
// 子弹是否有效
private boolean isEffect = true;
```