第3章

控制程序设计

在前面两章，所有的程序都是顺序结构，即程序从 main()方法开始进入，然后逐条指令开始执行，直到所有指令都执行完。但是，就像生活一样，不是所有的事情都能按照计划执行，许多事情经常会随着条件的变化面临着各种选择。程序亦是如此，也会根据不同条件执行相应的行为。例如，在“贪吃蛇”游戏中，按下不同的按键，“贪吃蛇”的运动方向随之而改变。本章主要介绍选择和循环控制结构程序设计的实现方法。

3.1 选择控制结构语句

3.1.1 if 语句

最常见的选择语句是 if 语句。if 语句用于选择是否执行一个行为，执行过程是先计算条件表达式的值，如果条件表达式为真，则执行其后的语句；否则，就直接跳过其后的语句。Java 语言提供了 3 种形式的 if 语句。

1. 单分支结构

单分支结构基本形式如下：

```
if(条件表达式){
    语句;
}
```

单分支结构非常简单，if 后面括号里的条件表达式值为真，就执行{}里的语句；否则，直接越过{}里的语句。{}里的语句是复合语句，相当于一个整体。{}里的语句可以是多条，也可以只有 1 条，当只有 1 条语句的时候，可以省略{}。


视频讲解

【例 3.1】 编写程序，实现通过按键控制方块运动，即当按下“w”键时，方块向上运动，如图 3.1 所示。

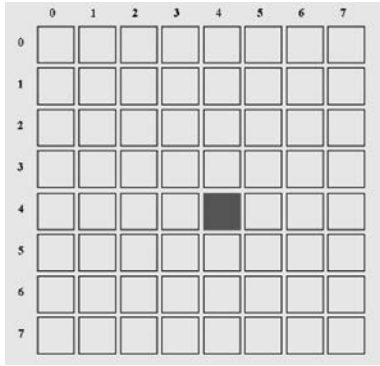

图 3.1　按键控制方块运动

为了实现按键控制方块运动，需要获得按键输入信息，然后根据判断信息执行相应的行为。Screen 类的 getKey()方法可以获得按键的输入信息，然后根据键值执行相应的操作。代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        int row = screen.getRandom(SIZE);
        int col = screen.getRandom(SIZE);
        Cell cell = new Cell(row,col);
        screen.add(cell);
        char key;
        while(true) {
            screen.delay();
            key = screen.getKey();
            if(key == 'w') {                    // 按键为 w,向上运动
                cell.moveUp();
            }
        }
    }
}
```

运行程序，在键盘上按“w”键时，方块向上运动。有时候不小心将键盘上大写键打开了，测试程序的时候，按下“w”键，方块并没有向上运动，这个错误不易发现，所以将代码修改成按下“W”键也能向上运动，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        int row = screen.getRandom(SIZE);
        int col = screen.getRandom(SIZE);
        Cell cell = new Cell(row,col);
        screen.add(cell);
        char key;
        while(true) {
```

```
            screen.delay();
            key = screen.getKey();
            if(key == 'w' || key == 'W') {
                cell.moveUp();
            }
        }
    }
}
```

运行程序，当按下“W”键的时候，方块向上运动。

2. 双分支结构

除了上述单分支选择结构，Java 语言提供了 if-else 双分支语句，即在两条语句之中进行选择，格式如下：

```
if(条件表达式){
    语句 1;
}
else{
    语句 2;
}
```

if-else 语句执行过程是：如果满足条件表达式，则执行语句 1；否则执行语句 2。语句 1 和语句 2 均可以由多条语句组成。另外，需注意 else 语句不能单独使用，必须与 if 配对使用。

【例 3.2】 编写程序，实现按“W”键控制方块向上运动，按其他键控制方块向下运动。


视频讲解

按下“W”键向上运动，按下其他键则向下运动，这是非常明显的二选一情况，所以可以使用 if-else 语句实现要求，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        int row = screen.getRandom(SIZE);
        int col = screen.getRandom(SIZE);
        Cell cell = new Cell(row,col);
        screen.add(cell);

        char key;
        while(true) {
            screen.delay();
            key = screen.getKey();
            if(key == 'w' || key == 'W') {
                cell.moveUp();
            }
            else{
                cell.moveDown();
            }
        }
    }
}
```

运行程序，当按下“W”键时，方块向上运动；按下其他键时，方块向下运动。

3. 多分支结构

在程序中，经常会遇到面临很多选择的时候，例如按下不同的按键控制方块向不同方向运动，Java 语言提供了 if-else if-else 语句，格式如下：

```
if(条件表达式 1){
    语句 1;
}
else if(条件表达式 2){
    语句 2;
}
…
else{
    语句 n;
}
```

该语句执行过程是：如果满足条件表达式 1 的时候，则执行语句 1，否则计算条件表达式 2；如果满足条件表达式 2，则执行语句 2，否则计算下一个条件表达式；直到所有条件均不满足，则执行 else 所对应的语句 n。


视频讲解

【例 3.3】 编写程序，实现按“W”键控制方块向上运动，按“S”键控制方块向下运动，按其他键控制方块斜向左上运动。

根据题意可知，这是多种选择的情况，可以使用 if-else if-else 语句实现功能，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        int row = screen.getRandom(SIZE);
        int col = screen.getRandom(SIZE);
        Cell cell = new Cell(row,col);
        screen.add(cell);
        char key;
        while(true) {
            screen.delay();
            key = screen.getKey();
            if(key == 'w' || key == 'W') {
                cell.moveUp();
            }
            else if(key == 's' || key == 'S'){
                cell.moveDown();
            }
            else{
                cell.moveLeft();
                cell.moveUp();
            }
        }
    }
}
```

运行程序，当按下“W”键时，方块向上运动；按下“S”键时，方块向下运动；按下其他按键时，

方块斜向左上运动。

if 语句用于选择是否执行一个行为，if-else 语句用于在两个语句之间进行选择，if-else if-else 语句用于在多个语句之间进行选择。

3.1.2　switch 语句

对于在多个选项中选择，不仅可以用 if-else if-else 语句来完成，Java 语言还提供了一种更为方便的 switch 语句，它的格式如下：

```
switch(表达式){
    case 常量表达式 1:
        语句 1;
        break;
    case 常量表达式 2:
        语句 2;
        break;
    ...
    default:语句 n;
}
```

该语句执行过程是：先计算表达式的值，如果表达式的值与某个常量表达式的值相等，则执行其后控制的语句；如果所有的常量表达式都与表达式的值不相等，则执行 default 后的语句。

需要注意的是：

(1) case 后面必须是常量表达式，不能包含变量，并且每个常量表达式的值都不相同。

(2) default 语句可以缺省。如果省略了 default 语句，当表达式的值与所有的常量表达的值都不相等，则什么也不执行。

(3) break 语句可以终止其所在的 switch 语句的执行。switch 语句执行的原理是遇到匹配项之后，开始执行其后的语句，如果没有遇到 break 语句，会一直执行到 switch 语句结束为止。

【例 3.4】 使用 switch 语句编写程序，实现按“w”键控制方块向上运动，按“s”键控制方块向下运动，按“a”键控制方块向左运动，按键“d”控制方块向右运动。


视频讲解

使用 switch 语句，代码如下：

```
public class Main {
  public static void main(String[ ] args){
    final int SIZE = 8;
    Screen screen = new Screen(SIZE);
    int row = screen.getRandom(SIZE);
    int col = screen.getRandom(SIZE);
    Cell cell = new Cell(row,col);
    screen.add(cell);
    char key;
    while(true) {
        screen.delay();
        key = screen.getKey();
        switch(key) {
            case 'w' :
```

```
                cell.moveUp();
                break;
            case 's':
                cell.moveDown();
                break;
            case 'a':
                cell.moveLeft();
                break;
            case 'd':
                cell.moveRight();
                break;
        }
    }
  }
}
```

运行程序，“w”“s”“a”“d”键可以分别控制方块上、下、左、右运动。

使用 swtich 语句的时候，要根据情况需要，判断是否要加上 break 语句。例如打开大写键，按键依然能控制方块上、下、左、右运动。使用 swtich 语句实现的话，代码如下：

```
public class Main {
  public static void main(String[ ] args){
    final int SIZE = 8;
    Screen screen = new Screen(SIZE);
    int row = screen.getRandom(SIZE);
    int col = screen.getRandom(SIZE);
    Cell cell = new Cell(row,col);
    screen.add(cell);

    char key;
    while(true) {
        screen.delay();
        key = screen.getKey();
        switch(key) {
            case 'w' :
            case 'W' :
                cell.moveUp();
                break;
            case 's':
            case 'S':
                cell.moveDown();
                break;
            case 'a':
            case 'A':
                cell.moveLeft();
                break;
            case 'd':
            case 'D':
                cell.moveRight();
                break;
        }
    }
  }
}
```

运行程序，即使打开大写键，“W”“S”“A”“D”键依然能控制方块上、下、左、右运动。当按下按键“W”的时候，case 'w' 选项后的语句为空，程序会自动往下执行，执行 case 'W' 选项对应的程序段，所以即使大写键开启，输入的键值是“W”也能控制方块向上运动。

如果给每一个 case 选项的语句段都增加 break 语句，则需要增加不少重复的代码。对于不同条件，执行相同行为的场景下，通过 switch 语句和 break 语句的结合使用，可以写出简洁的代码。

对于 switch 语句中漏掉 break 语句这种情况时有发生，Java 14 版本采用了新的方式规避这个问题，使用简化的“case L ->”模式匹配语法，作用于不同范围并控制执行流，代码为：

```
public class Main {
  public static void main(String[ ] args){
    final int SIZE = 8;
    Screen screen = new Screen(SIZE);
    int row = screen.getRandom(SIZE);
    int col = screen.getRandom(SIZE);
    Cell cell = new Cell(row,col);
    screen.add(cell);

    char key;
    while(true) {
        screen.delay();
        key = screen.getKey();
        switch(key) {
                case 'w','W' -> {
                    cell.moveUp();
                }
                case 's','S' -> {
                    cell.moveDown();
                }
                case 'a','A' ->{
                    cell.moveLeft();
                }
                case 'd','D' ->{
                    cell.moveRight();
                }
        }
    }
  }
}
```

将重复的 case 标签常量合并在一个 case 常量表达式中，用逗号分隔，箭头指向语句即可，执行完语句后退出 switch 语句。

虽然 switch 语句有时候比 if-else 语句简洁，逻辑关系一目了然，程序可读性好，但是使用范围较窄，比如选择条件是非常大的范围时，就不适合用 switch 语句。如表达变量 i 是 100～550 的整数时，使用 if 语句非常简单，代码如下：

```
if( i > 100 && i < 550 )
```

而使用 switch 语句将会非常麻烦，需要设置几百个 case 选项，因此对于初学者来说，熟练掌握 if 语句即可。

3.2 循环控制结构语句

循环是指反复执行某一过程，在3.1节中的案例使用的while语句就是一种经典的循环结构。循环结构是结构化程序的基本结构之一，它与顺序结构、选择结构共同构成各种复杂的程序。换言之，所有结构化程序都是由这三种基本结构组成。Java语言提供了while语句、do-while语句和for语句来实现循环结构。

3.2.1 while语句

while语句是常见的循环语句，while语句的格式如下：

```
while(条件表达式){
    循环体语句;
}
```

while语句的执行过程是：当圆括号里的条件表达式为真时，执行循环体语句，然后再判断表达式里的值，如果为真，继续执行循环体语句，如此重复执行，直到表达式的值为假的时候结束循环。

while语句和if语句，两者结构非常相似，两者的差别在于if语句只判断一次，而while语句会反复判断，直到条件不满足为止。


视频讲解

【例3.5】 编写程序，使用while语句，实现“贪吃蛇”不断向下运动，一直运动到屏幕底部撞到墙，游戏结束，如图3.2所示。

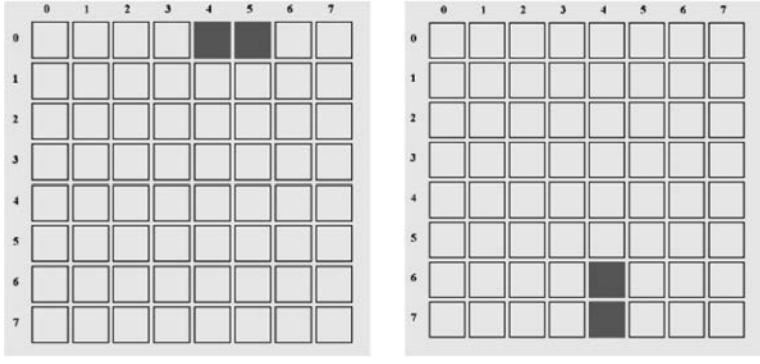

图3.2 “贪吃蛇”向下运动

“贪吃蛇”可以不断向下运动的条件就是：“蛇头”没有达到屏幕底部。解决了判断条件，接下来就是循环体的内容，主要包括两部分：延时和向下运动。代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        Cell head = new Cell(0,4);
```

```
        Cell body = new Cell(0,5);
        screen.add(head);
        screen.add(body);
        while(head.getRow()< SIZE - 1) {
            screen.delay();
            body.moveTo(head);
            head.moveDown();
        }
    }
}
```

运行程序，“贪吃蛇”不断向下运动，直到运动到屏幕最底部为止。

3.2.2　do-while 语句

除了 while 语句，Java 语言还提供了 do-while 语句实现循环结构，do-while 语句的格式如下：

```
do{
    循环体语句;
}while(条件表达式);
```

do-while 语句的执行过程是：先执行循环体语句，再进行判断，如果条件表达式为真，继续执行循环体语句，如此反复，直到条件表达式的值为假，结束循环。

【例 3.6】 编写程序，使用 do-while 循环语句，实现“贪吃蛇”不断向下运动，直到运动到屏幕底部。


视频讲解

使用 do-while 语句，循环执行的条件是判断“蛇头”是否运动到屏幕的底部。循环体的内容主要包括两部分，延时和向下运动，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        Cell head = new Cell(0,4);
        Cell body = new Cell(0,5);
        screen.add(head);
        screen.add(body);

        do{
            screen.delay();
            body.moveTo(head);
            head.moveDown();
        }while(head.getRow() < SIZE - 1);
    }
}
```

运行程序，“贪吃蛇”不断向下运动，直到运动到屏幕最底部。

通常情况下，对于同一问题，既可以使用 while 语句，也可以使用 do-while 语句，两者可以互换，运行结果相同，但是两者的差别为：do-while 语句是先执行后判断，所以 do-while 语句至少执行循环体一次，而 while 语句是先判断后执行，所以可能一次也不执行循环体。因此，如果 while 语句的条件表达式一开始就为假，两个循环语句的结果是不同的。

3.2.3 for 语句

例 3.5、例 3.6 的程序都是重复执行固定次数的循环，包含如下三部分：

(1) 初始化数值。

(2) 比较数值与目标值的关系。

(3) 循环递增数值。

在 while 语句中，这三部分内容放在不同的位置，初始化数值语句放在循环外，比较数值语句放在 while 语句中括号里面，而递增数值语句放在循环体语句内。当循环体里语句较多时，有时候会遗忘递增数值，从而导致循环无限执行。

Java 语言还提供了 for 循环语句，将初始化、判断、更新数值三部分内容组合在一起，形成更加紧凑的形式。for 循环语句基本格式如下：

```
for (表达式 1; 表达式 2; 表达式 3){
    循环语句;
}
```

表达式 1 通常是为控制循环的变量赋初值；表达式 2 是判断循环是否执行的条件表达式；表达式 3 一般用来更新控制循环的变量，其后无分号。

for 语句执行过程如下：

(1) 执行“表达式 1”，为控制循环的变量赋初值。

(2) 判断“表达式 2”，如果它的值为真，则执行循环体内的语句，否则转到第(5)步。

(3) 执行“表达式 3”，更新变量。

(4) 重复执行步骤(2)和(3)。

(5) 结束循环。

for 语句是 while 语句的一种变体，比 while 语句使用起来更加灵活。


视频讲解

【例 3.7】 编写程序，使用 for 循环语句，实现“贪吃蛇”不断向下运动，直到运动到屏幕底部。

使用 for 语句实现功能，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        Cell head = new Cell(0,4);
        Cell body = new Cell(0,5);
        screen.add(head);
        screen.add(body);

        for(int row = head.getRow(); row < SIZE - 1; row ++ ) {
            screen.delay();
            body.moveTo(head);
            head.moveDown();
        }
    }
}
```

运行程序，“贪吃蛇”不断向下运动，直到运动到屏幕最底部为止。

与 while 循环语句相比，for 循环语句结构更为紧凑，不容易遗忘更新变量，尤其是多重循环的时候，优势更加明显，因此 for 循环语句的使用非常广泛。

3.2.4　三种循环的比较

通过例 3.5～例 3.7 可知，一般情况下，对于同一问题，while、do-while、for 循环语句都可以处理，三者可以相互替代。

for 与 while 循环语句是“当型”循环，即先判断条件后执行循环体。do-while 循环语句是“直到型”循环，即先执行循环体，后判断条件。因此，for 与 while 循环语句可能一次也不执行循环体的内容，而 do-while 循环语句至少执行一次循环体的内容。

如何选择哪一种循环来解决问题？如果循环次数明确的时候，使用 for 循环更为方便，而循环次数不确定的时候，while 循环语句更容易理解。如果第一次循环肯定会执行，可以使用 do-while 循环语句。

for 循环语句比 while 和 do-while 循环语句功能更强大，使用更灵活。for 循环语句中的三个表达式可以部分省略或全部省略，但是两个分号不能省略，而且语句的位置可以灵活多变。

例如：

```
for(int count = 0; count < 6; count ++ ){
}
```

可以写成如下格式：

```
for(int count = 0; count < 6;){
        count ++ ;
}
```

或者

```
int count = 0;
for( ; count < 6; count ++ ){
}
```

还有其他多种写法，不一一列举，所以 for 语句非常灵活。

3.2.5　嵌套循环语句

一个循环体内包含着另一个循环结构，称为嵌套循环。嵌套循环里还可以继续嵌套循环，循环层次可以不断叠加。三种循环语句均可以相互嵌套。

【例 3.8】 编写程序，实现“打砖块”游戏中“砖墙”功能，将屏幕上面 3 行都填满方块，如图 3.3 所示。

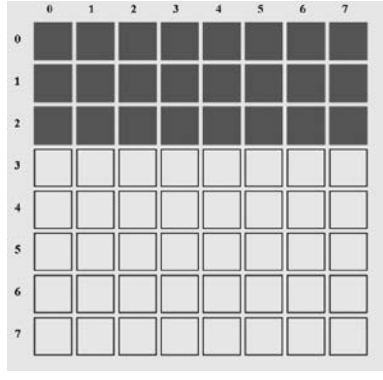

图 3.3　屏幕上 3 行填满方块

视频讲解

要将屏幕的上 3 行都填满方块，可以使用双重循环实现，内层循环的作用是将屏幕中的某一行都填满方块，外层循环的作用是控制行，实现从第 0 行到 2 行逐行填满方块，代码

如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);

        for(int row = 0; row < 3; row ++ ) {
            for(int col = 0; col < SIZE; col ++ ) {
                Cell cell = new Cell(row,col);
                screen.add(cell);
            }
        }
    }
}
```

运行程序，屏幕上 3 行都填满了方块。在阅读嵌套循环的时候，就像看钟表一样，内层循环像秒钟转了一圈，外层循环像分钟转一格。

对于例 3.8 也可以使用两个 while 循环嵌套实现，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);

        int row = 0;
        while(row < 3) {
            int col = 0;
            while(col < SIZE) {
                Cell cell = new Cell(row,col);
                screen.add(cell);
                col ++ ;
            }
            row ++ ;
        }
    }
}
```

对比 while 和 for 循环语句，可以看出，for 循环语句的结构更为紧凑，而且不容易遗忘变量的更新。

3.2.6 break 语句和 continue 语句

1. break 语句

在循环中，有时候需要提前结束循环，就像游戏进行到一半时，想提前结束游戏，Java 语言提供 break 语句，可以提前结束循环。

break 语句的一般格式如下：

```
break;
```

【例 3.9】　编写程序，实现按“W”键控制方块向上运动，按“S”键控制方块向下运动，按“K”键结束游戏。


视频讲解

相比例 3.3 和例 3.4 用按键控制方块运动的任务，该任务多了新功能，即按“K”键结束游戏。需要提前结束循环，则可以使用 break 语句，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        int row = screen.getRandom(SIZE);
        int col = screen.getRandom(SIZE);
        Cell cell = new Cell(row,col);
        screen.add(cell);
        char key;
        while(true) {
            screen.delay();
            key = screen.getKey();
            if(key == 'w' || key == 'W') {
                cell.moveUp();
            }
            if(key == 's' || key == 'S'){
                cell.moveDown();
            }
            if(key == 'k' || key == 'K'){
                break;
            }
        }
    }
}
```

运行程序，按键“W”控制方块向上运动，按键“S”控制方块向下运动，按键“K”控制游戏结束。

2. continue 语句

continue 语句也可以提前结束循环，不过它只结束本次循环，即跳过本次循环剩下的部分，进入下一次循环。

continue 语句的一般格式如下：

```
continue;
```

【例 3.10】　“赛车类”游戏中会出现一排障碍物，中间只留下狭窄的空隙供赛车通过，如图 3.4 所示。

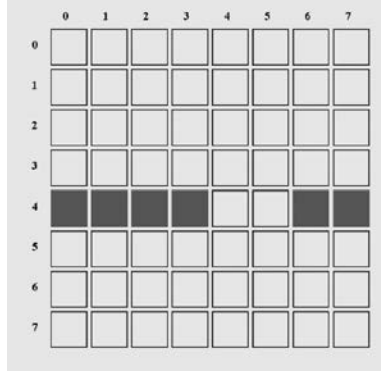

图 3.4　赛车障碍物


视频讲解

对于障碍物来说，除了“空隙”外，该行其他位置都应添加方块。因此，可以使用循环语句逐一添加方块，“空隙”的位置使用 continue 语句，跳过本次循环，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
```

```
            int row = screen.getRandom(SIZE);
            int gapCol = screen.getRandom(SIZE - 1);
            for(int col = 0; col < SIZE; col ++ ) {
                if(col == gapCol || col == gapCol + 1) {
                    continue;
                }
                Cell cell = new Cell(row,col);
                screen.add(cell);
            }
        }
    }
```

运行程序，显示如图 3.4 所示的界面。对于例 3.10，也可以不使用 continue 语句实现，代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        int row = screen.getRandom(SIZE);
        int gapCol = screen.getRandom(SIZE - 1);

        for(int col = 0; col < SIZE ; col ++ ) {
            if(col! = gapCol && col ! = gapCol + 1) {
                Cell cell = new Cell(row,col);
                screen.add(cell);
            }
        }
    }
}
```

通过案例可知，程序设计非常灵活，同一个问题可以有多种解决办法。continue 语句和 break 语句的区别是：break 语句是终止整个循环过程，而 continue 语句只结束本次循环，而且 continue 语句只能用在循环语句之中。

视频讲解

3.3 综合案例：按键控制“贪吃蛇”运动

编写程序，实现按键控制“贪吃蛇”运动，按键“w”控制“贪吃蛇”向上运动，按键“s”控制“贪吃蛇”向下运动，按键“a”控制“贪吃蛇”向左运动，按键“d”控制“贪吃蛇”向右运动，如图 3.5 所示。

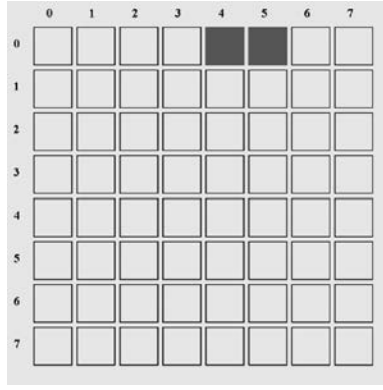

图 3.5 “贪吃蛇”运动

“贪吃蛇”的运动规律，主要由两部分组成：“蛇头”和“蛇身”，“蛇头”运动规律与运动方向有关系，“蛇身”运动规律与运动方向无关，无论什么运动方向，“蛇身”的每一部分都会移动到与之相邻方块的位置。

代码如下：

```
public class Main {
    public static void main(String[ ] args){
        final int SIZE = 8;
        Screen screen = new Screen(SIZE);
        Cell head = new Cell(0,4);
        Cell body = new Cell(0,5);
        screen.add(head);
        screen.add(body);

        char key;
        while(true) {
            screen.delay();
            key = screen.getKey();
            if(key == 'w') {
                body.moveTo(head);
                head.moveUp();
            }
            if(key == 's') {
                body.moveTo(head);
                head.moveDown();
            }
            if(key == 'a') {
                body.moveTo(head);
                head.moveLeft();
            }
            if(key == 'd') {
                body.moveTo(head);
                head.moveRight();
            }
        }
    }
}
```

提示：计算机相比于人最擅长的就是做重复的事，很多软件设计的目的就是解决重复劳动的问题。

习题

3.1　下列程序段，关于循环执行的次数，说法正确的是_____。

```
int k = 5 ;
while(k > 1){
    k-- ;
}
```

A. 循环执行了 5 次　　B. 循环执行了 0 次

C. 循环执行了 4 次　　D. 循环执行了 3 次

3.2　执行以下程序，sum 的值为_____。

```
int sum = 0;
int i ;
for( i = 0; i <= 5; i++ ){
    sum + = i;
}
```

A. 10　　B. 15　　C. 21　　D. 28

3.3　编写程序，在屏幕上显示杨辉三角图形，如图 3.6 所示。

第 0 行添加 1 个方块；

第 1 行添加 2 个方块；

…

第 N 行添加 N+1 个方块。

3.4　编写程序，在屏幕上显示倒杨辉三角图形，如图 3.7 所示。

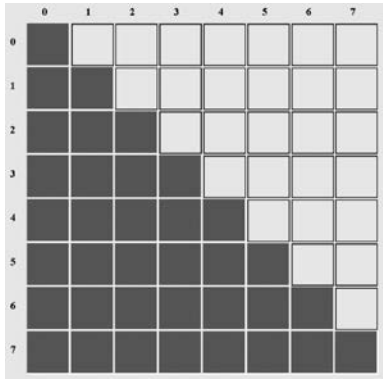

图 3.6　杨辉三角

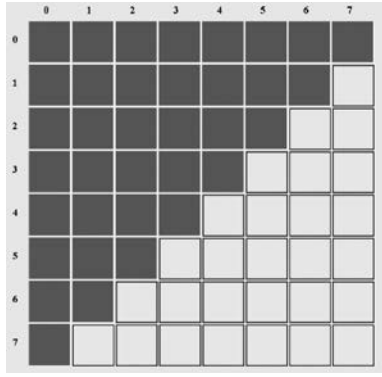

图 3.7　倒杨辉三角

3.5　编写程序，按键“W”“S”“A”“D”控制“Z”形“俄罗斯方块”（如图 3.8 所示）上、下、左、右运动。

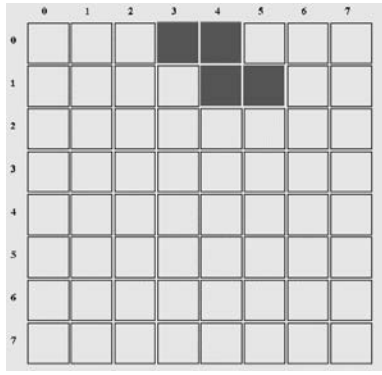

图 3.8　“Z”形“俄罗斯方块”示意图