第3章

应用数组的游戏开发

空战游戏中能否有10台敌机、反弹球消砖块中能否有30个待消除砖块、flappy bird中能否有5个柱子同时出现？在学习数组之前以上目标是很难实现的。本章利用数组的知识进一步改进游戏，实现更复杂的效果。

在前两章的基础上，学习本章前需要掌握的新语法知识：数组的定义、数组作为函数的参数。

3.1 生命游戏

假设有int Cells[50][50]，即有50×50个小格子，每个小格子里面生命存活（值为1）或者死亡（值为0），通过把所有元素的生命状态输出可以显示出相应的图案。

通过这个例子可以体会二维数组在游戏开发中的应用，实现所有数据的存储，并将画面显示、数据更新的代码分离，便于程序的维护和更新。本节游戏的最终代码参看“\随书资源\第3章\3.1生命游戏.cpp”，效果如图3-1所示。

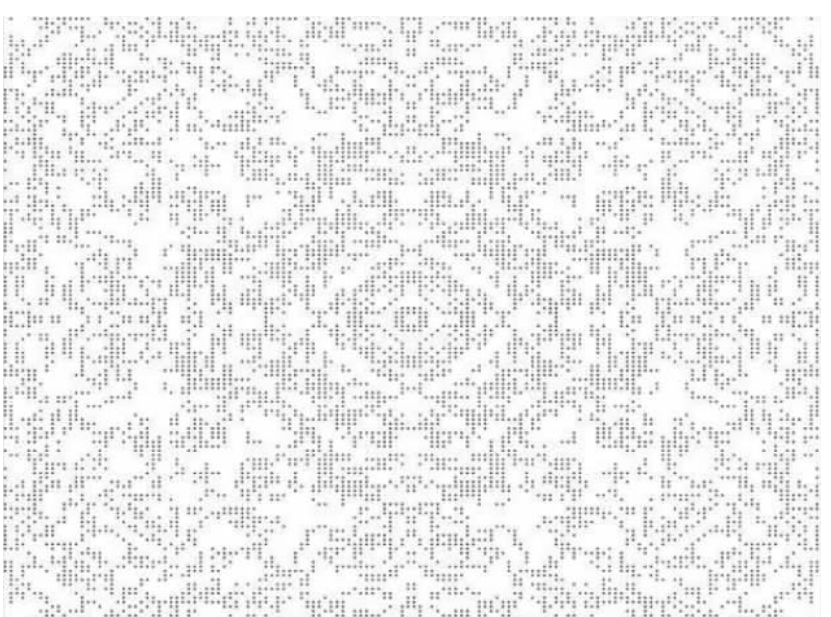

图3-1 生命游戏效果

3.1.1 游戏的初始化

第一步利用第 2 章的游戏框架进行初始化，输出静态的生命状态，如图 3-2 所示。二维数组 int cells[High][Width]记录所有位置细胞的存活状态，值为 1 表示生、值为 0 表示死。

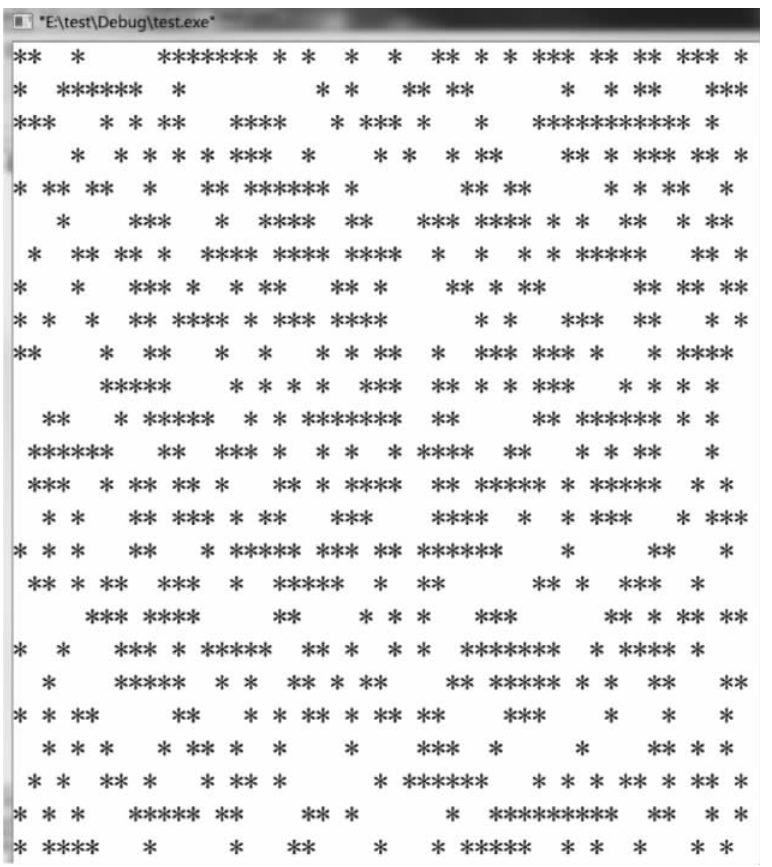

图 3-2 生命游戏的初始化效果

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
#include <time.h>

#define High 25                          // 游戏画面尺寸
#define Width 50

// 全局变量
int cells[High][Width];                  // 所有位置细胞生 1 或死 0

void gotoxy(int x,int y)                 // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
```

```
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                          // 数据的初始化
{
    int i,j;
    for (i = 0;i < High;i++)            // 随机初始化
        for (j = 0;j < Width;j++)
        {
            cells[i][j] = rand() % 2;
        }
}

void show()                             // 显示画面
{
    gotoxy(0,0);                        // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i <= High;i++)
    {
        for (j = 0;j <= Width;j++)
        {
            if (cells[i][j] == 1)
                printf(" * ");          // 输出活的细胞
            else
                printf(" ");            // 输出空格
        }
        printf("\n");
    }
    sleep(50);
}

void updateWithoutInput()               // 与用户输入无关的更新
{
}

void updateWithInput()                  // 与用户输入有关的更新
{
}

int main()
{
    startup();                          // 数据的初始化
    while (1)                           // 游戏循环执行
    {
        show();                         // 显示画面
        updateWithoutInput();           // 与用户输入无关的更新
        updateWithInput();              // 与用户输入有关的更新
    }
```

```
    return 0;
}
```

3.1.2 繁衍或死亡

每个矩阵方格可以包含一个有机体，不在边上的有机体有 8 个相邻方格。生命游戏演化的规则如下：

1. 如果一个细胞周围有 3 个细胞为生，则该细胞为生（即该细胞若原先为死，则转为生；若原先为生，则保持不变）。

2. 如果一个细胞周围有两个细胞为生，则该细胞的生死状态保持不变。

3. 在其他情况下该细胞为死（即该细胞若原先为生，则转为死；若原先为死，则保持不变）。

依照上面的规则让细胞进行繁衍或死亡，得到不断变化的图案。注意利用了中间变量数组 NewCells 来保存下一帧的存亡数据，具体原因请读者分析体会。

```
void startup()                          // 数据的初始化
{
    int i,j;
    for (i = 0;i < High;i++)            // 初始化
        for (j = 0;j < Width;j++)
            cells[i][j] = 1;
}

void updateWithoutInput()               // 与用户输入无关的更新
{
    int NewCells[High][Width];          // 下一帧的细胞情况
    int NeibourNumber;                  // 统计邻居的个数
    int i,j;
    for (i = 1;i < = High - 1;i++)
    {
        for (j = 1;j < = Width - 1;j++)
        {
            NeibourNumber = cells[i - 1][j - 1] + cells[i - 1][j] + cells[i - 1][j + 1]
                + cells[i][j - 1] + cells[i][j + 1] + cells[i + 1][j - 1] + cells[i + 1][j]
+ cells[i + 1][j + 1];
            if (NeibourNumber == 3)
                NewCells[i][j] = 1;
            else if (NeibourNumber == 2)
                NewCells[i][j] = cells[i][j];
            else
                NewCells[i][j] = 0;
        }
    }

    for (i = 1;i < = High - 1;i++)
        for (j = 1;j < = Width - 1;j++)
            cells[i][j] = NewCells[i][j];
}
```

3.1.3　小结

读者可以进一步修改生命游戏的规则，实现更复杂的效果。

思考题：

1. 让某块区域有水源，即在某块区域生命更容易生存、繁衍。
2. 实现按＋键生命游戏加速演化显示、-键减速、Esc 键暂停、R 键重新开始。
3. 实现捕食者、猎物组成的生命游戏，分别用不同的字符显示。

3.2　用数组实现反弹球消砖块

本节利用数组知识进一步改进反弹球消砖块游戏，实现多个待消砖块的效果，如图 3-3 所示。本节游戏的最终代码参看“\随书资源\第 3 章\ 3.2 反弹球.cpp”。

3.2.1　反弹球

第一步实现小球反弹的效果，如图 3-4 所示。其中，二维数组 int canvas[High][Width]存储游戏画布中的所有元素，0 输出空格，1 输出小球'0'；小球坐标为(ball_x，ball_y)，则 canvas[ball_x][ball_y] = 1，数组的其他元素值为 0。在 updateWithoutInput()函数中小球更新位置时先将原来位置所在数组元素值设为 0，再将新位置所在元素值设为 1。

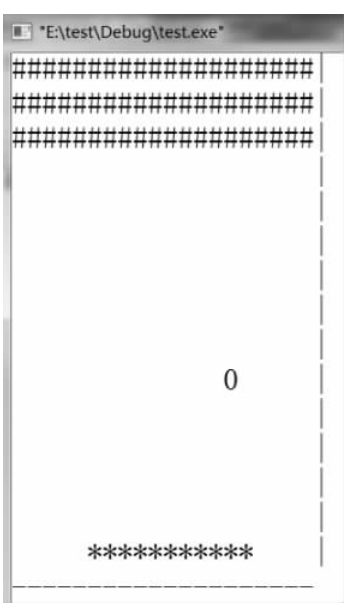

图 3-3　反弹球消砖块游戏效果

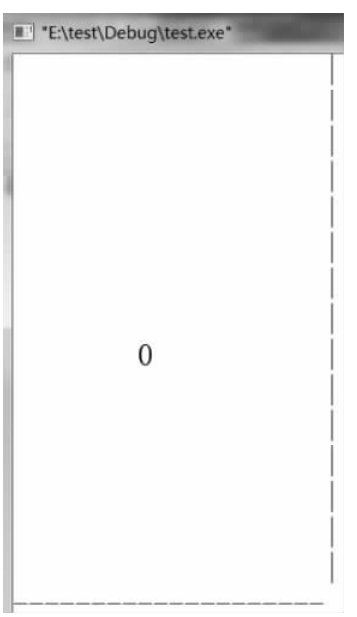

图 3-4　小球反弹效果

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <cwindow.h>
```

```
#define High 15                          // 游戏画面尺寸
#define Width 20

// 全局变量
int ball_x,ball_y;                       // 小球的坐标
int ball_vx,ball_vy;                     // 小球的速度
int canvas[High][Width] = {0};           // 二维数组存储游戏画布中对应的元素
                                         // 0 为空格,1 为小球 0

void gotoxy(int x,int y)                 // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                           // 数据的初始化
{
    ball_x = 0;
    ball_y = Width/2;
    ball_vx = 1;
    ball_vy = 1;
    canvas[ball_x][ball_y] = 1;
}

void show()                              // 显示画面
{
    gotoxy(0,0);                         // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");             // 输出空格
            else if (canvas[i][j] == 1)
                printf("0");             // 输出小球 0
        }
        printf("|\n");                   // 显示右边界
    }
    for (j = 0;j < Width;j++)
        printf("-");                     // 显示下边界
}

void updateWithoutInput()                // 与用户输入无关的更新
{
    canvas[ball_x][ball_y] = 0;

    ball_x = ball_x + ball_vx;
    ball_y = ball_y + ball_vy;

    if ((ball_x == 0)||(ball_x == High - 1))
```

```
            ball_vx = -ball_vx;
        if ((ball_y == 0)||(ball_y == Width - 1))
            ball_vy = -ball_vy;

        canvas[ball_x][ball_y] = 1;

        sleep(50);
    }

    void updateWithInput()                    // 与用户输入有关的更新
    {
    }

    int main()
    {
        startup();                            // 数据的初始化
        while (1)                             // 游戏循环执行
        {
            show();                           // 显示画面
            updateWithoutInput();             // 与用户输入无关的更新
            updateWithInput();                // 与用户输入有关的更新
        }
        return 0;
    }
```

3.2.2　增加挡板

第二步类似 2.2 节增加挡板，当二维数组 canvas[High][Width]中的元素值为 2 时输出挡板'*'。当在 updateWithInput()函数中控制挡板移动时每帧仅移动一个单位，同样需要先将原来位置所在数组元素值设为 0，再将新位置所在元素值设为 2，效果如图 3-5 所示。

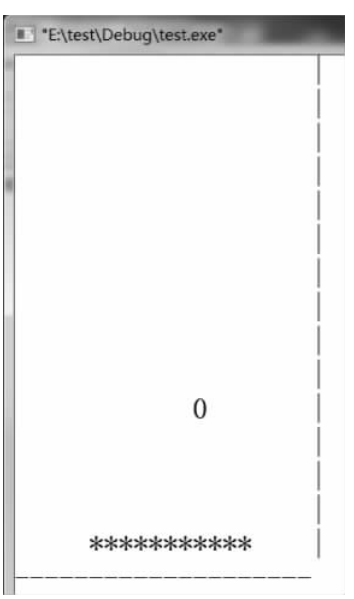

图 3-5　增加挡板效果

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <cwindow.h>

#define High 15                          // 游戏画面尺寸
#define Width 20

// 全局变量
int ball_x,ball_y;                       // 小球的坐标
int ball_vx,ball_vy;                     // 小球的速度
int position_x,position_y;               // 挡板的中心坐标
int ridus;                               // 挡板的半径大小
int left,right;                          // 挡板的左右位置
int canvas[High][Width] = {0};           // 二维数组存储游戏画布中对应的元素
// 0 为空格,1 为小球 0,2 为挡板 *

void gotoxy(int x,int y)                 // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                           // 数据的初始化
{
    ball_x = 0;
    ball_y = Width/2;
    ball_vx = 1;
    ball_vy = 1;
    canvas[ball_x][ball_y] = 1;

    ridus = 5;
    position_x = High - 1;
    position_y = Width/2;
    left = position_y - ridus;
    right = position_y + ridus;

    int k;
    for (k = left;k <= right;k++)
        canvas[position_x][k] = 2;
}

void show()                              // 显示画面
{
    gotoxy(0,0);                         // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
```

```
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");              // 输出空格
            else if (canvas[i][j] == 1)
                printf("0");              // 输出小球 0
            else if (canvas[i][j] == 2)
                printf(" * ");            // 输出挡板 *
        }
        printf("|\n");                    // 显示右边界
    }
    for (j = 0;j < Width;j++)
        printf(" - ");                    // 显示下边界
    printf("\n");
}

void updateWithoutInput()                 // 与用户输入无关的更新
{
    if (ball_x == High - 2)
    {
        if ( (ball_y >= left) && (ball_y <= right) )          // 被挡板挡住
        {
            printf("\a");                 // 响铃
        }
        else                              // 没有被挡板挡住
        {
            printf("游戏失败\n");
            system("pause");
            exit(0);
        }
    }

    canvas[ball_x][ball_y] = 0;

    ball_x = ball_x + ball_vx;
    ball_y = ball_y + ball_vy;

    if ((ball_x == 0)||(ball_x == High - 2))
        ball_vx = - ball_vx;
    if ((ball_y == 0)||(ball_y == Width - 1))
        ball_vy = - ball_vy;

    canvas[ball_x][ball_y] = 1;

    sleep(50);
}

void updateWithInput()                    // 与用户输入有关的更新
{
    char input;
    if(kbhit())                           // 判断是否有输入
```

```
    {
        input = getch();                    // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a' && left > 0)
        {
            canvas[position_x][right] = 0;
            position_y--;                   // 位置左移
            left = position_y - ridus;
            right = position_y + ridus;
            canvas[position_x][left] = 2;
        }
        if (input == 'd' && right < Width - 1)
        {
            canvas[position_x][left] = 0;
            position_y++;                   // 位置右移
            left = position_y - ridus;
            right = position_y + ridus;
            canvas[position_x][right] = 2;
        }
    }
}

int main()
{
    startup();                              // 数据的初始化
    while (1)                               // 游戏循环执行
    {
        show();                             // 显示画面
        updateWithoutInput();               // 与用户输入无关的更新
        updateWithInput();                  // 与用户输入有关的更新
    }
    return 0;
}
```

3.2.3 消砖块

第三步增加砖块,当二维数组 canvas[High][Width]中的元素值为 3 时输出挡板'#'。由于采用了数组,在 startup()中可以很方便地初始化多个砖块。在 updateWithoutInput()中判断小球碰到砖块后对应数组元素值由 3 变为 0,即该砖块消失,效果如图 3-6 所示。

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <cwindow.h>

#define High 15                             // 游戏画面尺寸
#define Width 20

// 全局变量
int ball_x,ball_y;                          // 小球的坐标
int ball_vx,ball_vy;                        // 小球的速度
```

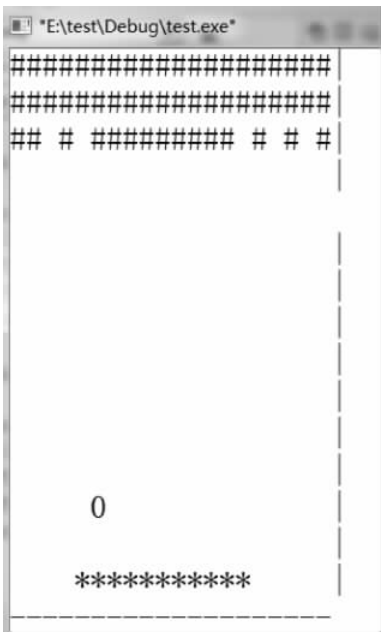

图 3-6　消砖块效果

```
int position_x,position_y;                    // 挡板的中心坐标
int ridus;                                    // 挡板的半径大小
int left,right;                               // 挡板的左右位置
int canvas[High][Width] = {0};                // 二维数组存储游戏画布中对应的元素
// 0 为空格,1 为小球 0,2 为挡板 * ,3 为砖块 #

void gotoxy(int x,int y)                      // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                                // 数据的初始化
{
    ridus = 5;
    position_x = High - 1;
    position_y = Width/2;
    left = position_y - ridus;
    right = position_y + ridus;

    ball_x = position_x - 1;
    ball_y = position_y;
    ball_vx = -1;
    ball_vy = 1;
    canvas[ball_x][ball_y] = 1;
```

```
    int k,i;
    for (k = left;k <= right;k++)          // 挡板
        canvas[position_x][k] = 2;

    for (k = 0;k < Width;k++)              // 加几排砖块
        for (i = 0;i < High/4;i++)
            canvas[i][k] = 3;
}

void show()                                // 显示画面
{
    gotoxy(0,0);                           // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");               // 输出空格
            else if (canvas[i][j] == 1)
                printf("0");               // 输出小球 0
            else if (canvas[i][j] == 2)
                printf("*");               // 输出挡板 *
            else if (canvas[i][j] == 3)
                printf("#");               // 输出砖块 #
        }
        printf("|\n");                     // 显示右边界
    }
    for (j = 0;j < Width;j++)
        printf("-");                       // 显示下边界
    printf("\n");
}

void updateWithoutInput()                  // 与用户输入无关的更新
{
    if (ball_x == High - 2)
    {
        if ( (ball_y >= left) && (ball_y <= right) )                    // 被挡板挡住
        {
        }
        else                               // 没有被挡板挡住
        {
            printf("游戏失败\n");
            system("pause");
            exit(0);
        }
    }

    static int speed = 0;
    if (speed < 7)
```

```
        speed++;
    if (speed == 7)
    {
        speed = 0;

        canvas[ball_x][ball_y] = 0;
        // 更新小球的坐标
        ball_x = ball_x + ball_vx;
        ball_y = ball_y + ball_vy;
        canvas[ball_x][ball_y] = 1;

        // 碰到边界后反弹
        if ((ball_x == 0)||(ball_x == High - 2))
            ball_vx = -ball_vx;
        if ((ball_y == 0)||(ball_y == Width - 1))
            ball_vy = -ball_vy;

        // 碰到砖块后反弹
        if (canvas[ball_x - 1][ball_y] == 3)
        {
            ball_vx = -ball_vx;
            canvas[ball_x - 1][ball_y] = 0;
            printf("\a");
        }
    }
}

void updateWithInput()                    // 与用户输入有关的更新
{
    char input;
    if(kbhit())                           // 判断是否有输入
    {
        input = getch();                  // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a' && left > 0)
        {
            canvas[position_x][right] = 0;
            position_y--;                 // 位置左移
            left = position_y - ridus;
            right = position_y + ridus;
            canvas[position_x][left] = 2;
        }
        if (input == 'd' && right < Width - 1)
        {
            canvas[position_x][left] = 0;
            position_y++;                 // 位置右移
            left = position_y - ridus;
            right = position_y + ridus;
            canvas[position_x][right] = 2;
        }
    }
}
```

```
int main()
{
    startup();                          // 数据的初始化
    while (1)                           // 游戏循环执行
    {
        show();                         // 显示画面
        updateWithoutInput();           // 与用户输入无关的更新
        updateWithInput();              // 与用户输入有关的更新
    }
    return 0;
}
```

3.2.4 小结

应用数组可以更方便地记录复杂的数据，实现更复杂的显示、逻辑判断与控制。

思考题：

1. 按空格键发射新的小球。
2. 尝试实现接金币的小游戏。

3.3 空 战 游 戏

本节利用数组进一步改进空战游戏，并实现多台敌机、发射散弹等效果，如图 3-7 所示。读者可以先尝试逐步实现，再参考代码“\随书资源\第 3 章\ 3.3 空战游戏.cpp”。

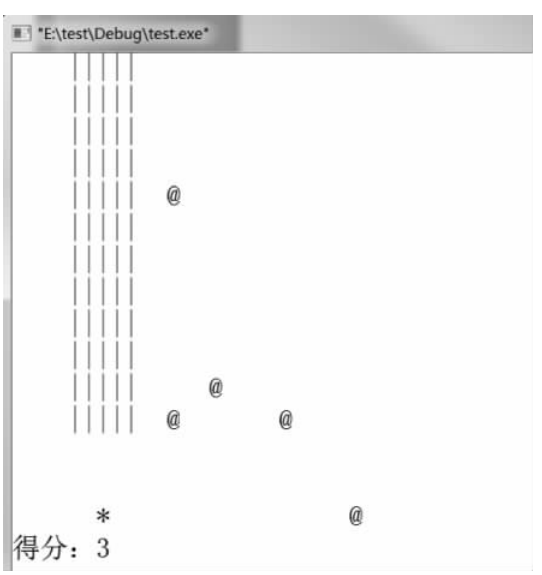

图 3-7 空战游戏效果

3.3.1 飞机的显示与控制

第一步实现飞机的显示和控制。在二维数组 int canvas[High][Width]中存储游戏画面数据,元素值为 0 输出空格,为 1 输出飞机'*','飞机移动的实现和 3.2 节中反弹球的移动类似。

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

#define High 25                          // 游戏画面尺寸
#define Width 50

// 全局变量
int position_x,position_y;               // 飞机的位置
int canvas[High][Width] = {0};           // 二维数组存储游戏画布中对应的元素
                                         // 0 为空格,1 为飞机 *

void gotoxy(int x,int y)                 // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                           // 数据的初始化
{
    position_x = High/2;
    position_y = Width/2;
    canvas[position_x][position_y] = 1;
}

void show()                              // 显示画面
{
    gotoxy(0,0);                         // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");             // 输出空格
            else if (canvas[i][j] == 1)
                printf("*");             // 输出飞机 *
        }
        printf("\n");
    }
```

```
}

void updateWithoutInput()                    // 与用户输入无关的更新
{
}

void updateWithInput()                       // 与用户输入有关的更新
{
    char input;
    if(kbhit())                              // 判断是否有输入
    {
        input = getch();                     // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a')
        {
            canvas[position_x][position_y] = 0;
            position_y--;                    // 位置左移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'd')
        {
            canvas[position_x][position_y] = 0;
            position_y++;                    // 位置右移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'w')
        {
            canvas[position_x][position_y] = 0;
            position_x--;                    // 位置上移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 's')
        {
            canvas[position_x][position_y] = 0;
            position_x++;                    // 位置下移
            canvas[position_x][position_y] = 1;
        }
    }
}

int main()
{
    startup();                               // 数据的初始化
    while (1)                                // 游戏循环执行
    {
        show();                              // 显示画面
        updateWithoutInput();                // 与用户输入无关的更新
        updateWithInput();                   // 与用户输入有关的更新
    }
    return 0;
}
```

3.3.2　发射子弹

第二步实现发射子弹的功能，当二维数组 canvas[High][Width]中的元素值为 2 时输出子弹'|'。改进后玩家可以连续按键，在画面中会同时显示多发子弹，如图 3-8 所示。

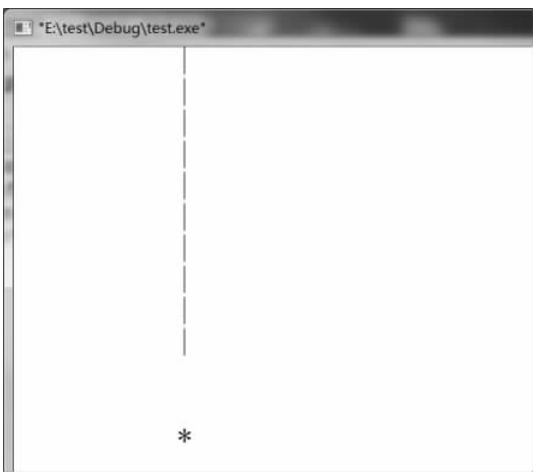

图 3-8　发射多发子弹效果

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

#define High 25                          // 游戏画面尺寸
#define Width 50

// 全局变量
int position_x,position_y;               // 飞机的位置
int canvas[High][Width] = {0};           // 二维数组存储游戏画布中对应的元素
                                         // 0 为空格,1 为飞机 * ,2 为子弹|,3 为敌机@

void gotoxy(int x,int y)                 // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                           // 数据的初始化
{
    position_x = High/2;
    position_y = Width/2;
    canvas[position_x][position_y] = 1;
}
```

```
void show()                                    // 显示画面
{
    gotoxy(0,0);                               // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");                   // 输出空格
            else if (canvas[i][j] == 1)
                printf("*");                   // 输出飞机*
            else if (canvas[i][j] == 2)
                printf("|");                   // 输出子弹|
        }
        printf("\n");
    }
}

void updateWithoutInput()                      // 与用户输入无关的更新
{
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 2)             // 子弹向上移动
            {
                canvas[i][j] = 0;
                if (i > 0)
                    canvas[i - 1][j] = 2;
            }
        }
    }
}

void updateWithInput()                         // 与用户输入有关的更新
{
    char input;
    if(kbhit())                                // 判断是否有输入
    {
        input = getch();                       // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a')
        {
            canvas[position_x][position_y] = 0;
            position_y--;                      // 位置左移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'd')
        {
```

```
            canvas[position_x][position_y] = 0;
            position_y++;                    // 位置右移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'w')
        {
            canvas[position_x][position_y] = 0;
            position_x--;                    // 位置上移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 's')
        {
            canvas[position_x][position_y] = 0;
            position_x++;                    // 位置下移
            canvas[position_x][position_y] = 1;
        }
        else if (input == ' ')               // 发射子弹
        {
            canvas[position_x-1][position_y] = 2;  // 发射子弹的初始位置在飞机的正上方
        }
    }
}

int main()
{
    startup();                               // 数据的初始化
    while (1)                                // 游戏循环执行
    {
        show();                              // 显示画面
        updateWithoutInput();                // 与用户输入无关的更新
        updateWithInput();                   // 与用户输入有关的更新
    }
    return 0;
}
```

3.3.3　击中敌机

第三步增加一个下落的敌机，当二维数组 canvas[High][Width]中的元素值为 3 时输出敌机'@'，加入击中敌机、敌机撞击我机的功能，如图 3-9 所示。

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

#define High 15                              // 游戏画面尺寸
#define Width 25

// 全局变量
int position_x,position_y;                   // 飞机的位置
```

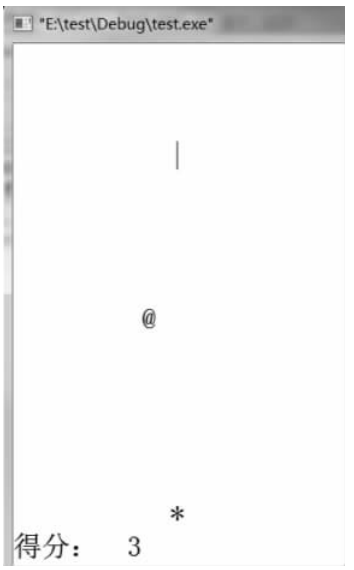

图 3-9 增加敌机和得分显示

```
int enemy_x,enemy_y;                    // 敌机的位置
int canvas[High][Width] = {0};          // 二维数组存储游戏画布中对应的元素
                                        // 0 为空格,1 为飞机 * ,2 为子弹|,3 为敌机@
int score;                              // 得分

void gotoxy(int x,int y)                // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                          // 数据的初始化
{
    position_x = High - 1;
    position_y = Width/2;
    canvas[position_x][position_y] = 1;
    enemy_x = 0;
    enemy_y = position_y;
    canvas[enemy_x][enemy_y] = 3;
    score = 0;
}

void show()                             // 显示画面
{
    gotoxy(0,0);                        // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
```

```
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");            // 输出空格
            else if (canvas[i][j] == 1)
                printf("*");            // 输出飞机*
            else if (canvas[i][j] == 2)
                printf("|");            // 输出子弹|
            else if (canvas[i][j] == 3)
                printf("@");            // 输出敌机@
        }
        printf("\n");
    }
    printf("得分: %3d\n",score);
    sleep(20);
}

void updateWithoutInput()               // 与用户输入无关的更新
{
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 2)
            {
                if ((i == enemy_x) && (j == enemy_y))                    // 子弹击中敌机
                {
                    score++;            // 分数加 1
                    canvas[enemy_x][enemy_y] = 0;
                    enemy_x = 0;        // 产生新的飞机
                    enemy_y = rand() % Width;
                    canvas[enemy_x][enemy_y] = 3;
                    canvas[i][j] = 0;   // 子弹消失
                }

                // 子弹向上移动
                canvas[i][j] = 0;
                if (i > 0)
                    canvas[i - 1][j] = 2;
            }
        }
    }

    if ((position_x == enemy_x) && (position_y == enemy_y))              // 敌机撞到我机
    {
        printf("失败!\n");
        Sleep(3000);
        system("pause");
        exit(0);
```

```
    }

    if (enemy_x > High)                    // 敌机跑出显示屏幕
    {
        canvas[enemy_x][enemy_y] = 0;
        enemy_x = 0;                       // 产生新的飞机
        enemy_y = rand() % Width;
        canvas[enemy_x][enemy_y] = 3;
        score--;                           // 减分
    }

    static int speed = 0;
    if (speed < 10)
        speed++;
    if (speed == 10)
    {
        // 敌机下落
        canvas[enemy_x][enemy_y] = 0;
        enemy_x++;
        speed = 0;
        canvas[enemy_x][enemy_y] = 3;
    }
}

void updateWithInput()                     // 与用户输入有关的更新
{
    char input;
    if(kbhit())                            // 判断是否有输入
    {
        input = getch();                   // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a')
        {
            canvas[position_x][position_y] = 0;
            position_y--;                  // 位置左移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'd')
        {
            canvas[position_x][position_y] = 0;
            position_y++;                  // 位置右移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'w')
        {
            canvas[position_x][position_y] = 0;
            position_x--;                  // 位置上移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 's')
        {
            canvas[position_x][position_y] = 0;
```

```
            position_x++;                    // 位置下移
            canvas[position_x][position_y] = 1;
        }
        else if (input == ' ')               // 发射子弹
        {
            canvas[position_x-1][position_y] = 2;  // 发射子弹的初始位置在飞机的正上方
        }
    }
}

int main()
{
    startup();                               // 数据的初始化
    while (1)                                // 游戏循环执行
    {
        show();                              // 显示画面
        updateWithoutInput();                // 与用户输入无关的更新
        updateWithInput();                   // 与用户输入有关的更新
    }
    return 0;
}
```

3.3.4　多台敌机

第四步利用数组 enemy_x[EnemyNum],enemy_y[EnemyNum]存储多台敌机的位置，可以实现同时出现多台敌机的效果。

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

#define High 15                              // 游戏画面尺寸
#define Width 25
#define EnemyNum 5                           // 敌机的个数

// 全局变量
int position_x,position_y;                   // 飞机的位置
int enemy_x[EnemyNum],enemy_y[EnemyNum];     // 敌机的位置
int canvas[High][Width] = {0};               // 二维数组存储游戏画布中对应的元素
                                             // 0 为空格,1 为飞机 *,2 为子弹|,3 为敌机@
int score;                                   // 得分

void gotoxy(int x,int y)                     // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
```

```
}

void startup()                                  // 数据的初始化
{
    position_x = High - 1;
    position_y = Width/2;
    canvas[position_x][position_y] = 1;
    int k;
    for (k = 0;k < EnemyNum;k++)
    {
        enemy_x[k] = rand() % 2;
        enemy_y[k] = rand() % Width;
        canvas[enemy_x[k]][enemy_y[k]] = 3;
    }
    score = 0;
}

void show()                                     // 显示画面
{
    gotoxy(0,0);                                // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");                    // 输出空格
            else if (canvas[i][j] == 1)
                printf("*");                    // 输出飞机*
            else if (canvas[i][j] == 2)
                printf("|");                    // 输出飞机|
            else if (canvas[i][j] == 3)
                printf("@");                    // 输出飞机@
        }
        printf("\n");
    }
    printf("得分: %3d\n",score);
    sleep(20);
}

void updateWithoutInput()                       // 与用户输入无关的更新
{
    int i,j,k;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 2)
            {
                for (k = 0;k < EnemyNum;k++)
                {
```

```
                if ((i == enemy_x[k]) && (j == enemy_y[k]))          // 子弹击中敌机
                {
                    score++;            // 分数加 1
                    canvas[enemy_x[k]][enemy_y[k]] = 0;
                    enemy_x[k] = rand() % 2;                         // 产生新的飞机
                    enemy_y[k] = rand() % Width;
                    canvas[enemy_x[k]][enemy_y[k]] = 3;
                    canvas[i][j] = 0;  // 子弹消失
                }
            }
            // 子弹向上移动
            canvas[i][j] = 0;
            if (i > 0)
                canvas[i - 1][j] = 2;
        }
    }
}

static int speed = 0;
if (speed < 20)
    speed++;

for (k = 0;k < EnemyNum;k++)
{
    if ((position_x == enemy_x[k]) && (position_y == enemy_y[k]))        // 敌机撞到我机
    {
        printf("失败!\n");
        Sleep(3000);
        system("pause");
        exit(0);
    }

    if (enemy_x[k]> High)                // 敌机跑出显示屏幕
    {
        canvas[enemy_x[k]][enemy_y[k]] = 0;
        enemy_x[k] = rand() % 2;         // 产生新的飞机
        enemy_y[k] = rand() % Width;
        canvas[enemy_x[k]][enemy_y[k]] = 3;
        score -- ;                       // 减分
    }

    if (speed == 20)
    {
        // 敌机下落
        for (k = 0;k < EnemyNum;k++)
        {
            canvas[enemy_x[k]][enemy_y[k]] = 0;
            enemy_x[k]++;
            speed = 0;
            canvas[enemy_x[k]][enemy_y[k]] = 3;
        }
```

```
        }
    }
}

void updateWithInput()                          // 与用户输入有关的更新
{
    char input;
    if(kbhit())                                 // 判断是否有输入
    {
        input = getch();                        // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a')
        {
            canvas[position_x][position_y] = 0;
            position_y--;                       // 位置左移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'd')
        {
            canvas[position_x][position_y] = 0;
            position_y++;                       // 位置右移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'w')
        {
            canvas[position_x][position_y] = 0;
            position_x--;                       // 位置上移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 's')
        {
            canvas[position_x][position_y] = 0;
            position_x++;                       // 位置下移
            canvas[position_x][position_y] = 1;
        }
        else if (input == ' ')                  // 发射子弹
        {
            canvas[position_x-1][position_y] = 2;  // 发射子弹的初始位置在飞机的正上方
        }
    }
}

int main()
{
    startup();                                  // 数据的初始化
    while (1)                                   // 游戏循环执行
    {
        show();                                 // 显示画面
        updateWithoutInput();                   // 与用户输入无关的更新
        updateWithInput();                      // 与用户输入有关的更新
    }
    return 0;
}
```

3.3.5　发射散弹

第五步实现发射宽度为 BulletWidth 的散弹，如图 3-10 所示。当积分增加后，散弹半径增大、敌机移动的速度加快。

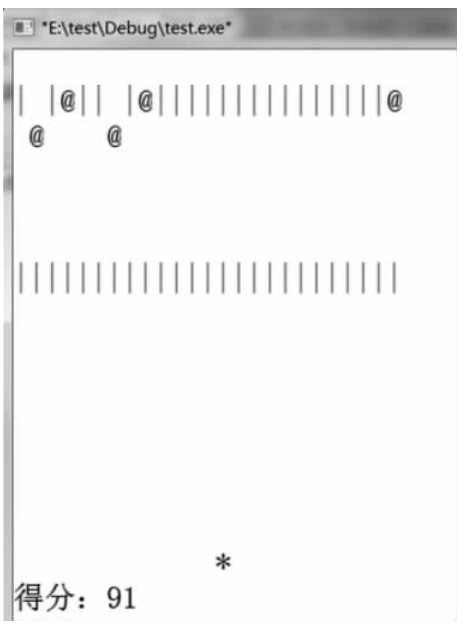

图 3-10　发射散弹效果

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

#define High 15                                  // 游戏画面尺寸
#define Width 25
#define EnemyNum 5                               // 敌机的个数

// 全局变量
int position_x,position_y;                       // 飞机的位置
int enemy_x[EnemyNum],enemy_y[EnemyNum];         // 敌机的位置
int canvas[High][Width] = {0};                   // 二维数组存储游戏画布中对应的元素
                                                 // 0 为空格,1 为飞机 * ,2 为子弹|,3 为敌机@
int score;                                       // 得分
int BulletWidth;                                 // 子弹的宽度
int EnemyMoveSpeed;                              // 敌机的移动速度

void gotoxy(int x,int y)                         // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
```

```
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                                  // 数据的初始化
{
    position_x = High - 1;
    position_y = Width/2;
    canvas[position_x][position_y] = 1;
    int k;
    for (k = 0;k < EnemyNum;k++)
    {
        enemy_x[k] = rand() % 2;
        enemy_y[k] = rand() % Width;
        canvas[enemy_x[k]][enemy_y[k]] = 3;
    }
    score = 0;
    BulletWidth = 0;
    EnemyMoveSpeed = 20;
}

void show()                                     // 显示画面
{
    gotoxy(0,0);                                // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");                    // 输出空格
            else if (canvas[i][j] == 1)
                printf("*");                    // 输出飞机*
            else if (canvas[i][j] == 2)
                printf("|");                    // 输出子弹|
            else if (canvas[i][j] == 3)
                printf("@");                    // 输出飞机@
        }
        printf("\n");
    }
    printf("得分: %d\n",score);
    Sleep(20);
}

void updateWithoutInput()                       // 与用户输入无关的更新
{
    int i,j,k;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
```

```
        if (canvas[i][j] == 2)
        {
            for (k = 0;k < EnemyNum;k++)
            {
                if ((i == enemy_x[k]) && (j == enemy_y[k]))              // 子弹击中敌机
                {
                    score++;              // 分数加 1
                    if (score % 5 == 0 && EnemyMoveSpeed > 3)  // 达到一定积分后敌机变快
                        EnemyMoveSpeed -- ;
                    if (score % 5 == 0)   // 达到一定积分后子弹变厉害
                        BulletWidth++;
                    canvas[enemy_x[k]][enemy_y[k]] = 0;
                    enemy_x[k] = rand() % 2;                             // 产生新的飞机
                    enemy_y[k] = rand() % Width;
                    canvas[enemy_x[k]][enemy_y[k]] = 3;
                    canvas[i][j] = 0;  // 子弹消失
                }
            }
            // 子弹向上移动
            canvas[i][j] = 0;
            if (i > 0)
                canvas[i - 1][j] = 2;
        }
    }
}

static int speed = 0;
if (speed < EnemyMoveSpeed)
    speed++;

for (k = 0;k < EnemyNum;k++)
{
    if ((position_x == enemy_x[k]) && (position_y == enemy_y[k]))        // 敌机撞到我机
    {
        printf("失败!\n");
        Sleep(3000);
        system("pause");
        exit(0);
    }

    if (enemy_x[k]> High)                 // 敌机跑出显示屏幕
    {
        canvas[enemy_x[k]][enemy_y[k]] = 0;
        enemy_x[k] = rand() % 2;          // 产生新的飞机
        enemy_y[k] = rand() % Width;
        canvas[enemy_x[k]][enemy_y[k]] = 3;
        score -- ;                        // 减分
    }

    if (speed == EnemyMoveSpeed)
    {
        // 敌机下落
        for (k = 0;k < EnemyNum;k++)
        {
```

```
                canvas[enemy_x[k]][enemy_y[k]] = 0;
                enemy_x[k]++;
                speed = 0;
                canvas[enemy_x[k]][enemy_y[k]] = 3;
            }
        }
    }
}

void updateWithInput()                          // 与用户输入有关的更新
{
    char input;
    if(kbhit())                                 // 判断是否有输入
    {
        input = getch();                        // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a' && position_y > 0)
        {
            canvas[position_x][position_y] = 0;
            position_y--;                       // 位置左移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'd' && position_y < Width - 1)
        {
            canvas[position_x][position_y] = 0;
            position_y++;                       // 位置右移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 'w')
        {
            canvas[position_x][position_y] = 0;
            position_x--;                       // 位置上移
            canvas[position_x][position_y] = 1;
        }
        else if (input == 's')
        {
            canvas[position_x][position_y] = 0;
            position_x++;                       // 位置下移
            canvas[position_x][position_y] = 1;
        }
        else if (input == ' ')                  // 发射子弹
        {
            int left = position_y - BulletWidth;
            int right = position_y + BulletWidth;
            if (left < 0)
                left = 0;
            if (right > Width - 1)
                right = Width - 1;
            int k;
            for (k = left;k <= right;k++)       // 发射子弹
                canvas[position_x - 1][k] = 2;          // 发射子弹的初始位置在飞机的正上方
        }
    }
}
```

```
int main()
{
    startup();                          // 数据的初始化
    while (1)                           // 游戏循环执行
    {
        show();                         // 显示画面
        updateWithoutInput();           // 与用户输入无关的更新
        updateWithInput();              // 与用户输入有关的更新
    }
    return 0;
}
```

3.3.6　小结

本节的空战游戏是不是更有趣了？大家实现这个接近 200 行代码的程序会较好地掌握语法知识、锻炼逻辑思维。

思考题：

1. 增加敌机 boss，其形状更大、血量更多。
2. 尝试让游戏更有趣，敌机也发射子弹。

3.4　贪　吃　蛇

本节实现一个经典的小游戏——贪吃蛇，如图 3-11 所示。读者可以先自己尝试，主要难点是小蛇数据如何存储、如何实现转弯的效果、吃到食物后如何增加长度。本节游戏的最终代码参看“\随书资源\第 3 章\ 3.4 贪吃蛇.cpp”。

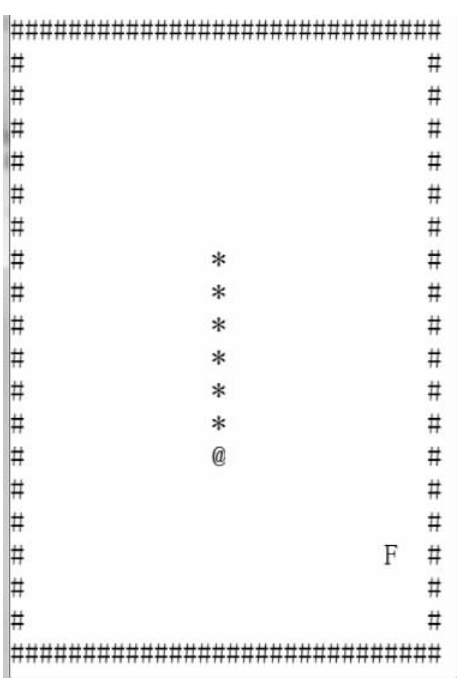

图 3-11　贪吃蛇游戏效果

3.4.1 构造小蛇

第一节在画面中显示一条静止的小蛇，如图 3-12 所示。对于二维数组 canvas[High][Width]的对应元素，值为 0 输出空格，值为－1 输出边框＃，值为 1 输出蛇头@，值为大于 1 的正数输出蛇身＊。在 startup()函数中初始化蛇头在画布的中间位置(canvas[High/2][Width/2] = 1;)，蛇头向左依次生成 4 个蛇身(for (i=1;i<=4;i++) canvas[High/2][Width/2-i] = i+1;)，元素值分别为 2、3、4、5。

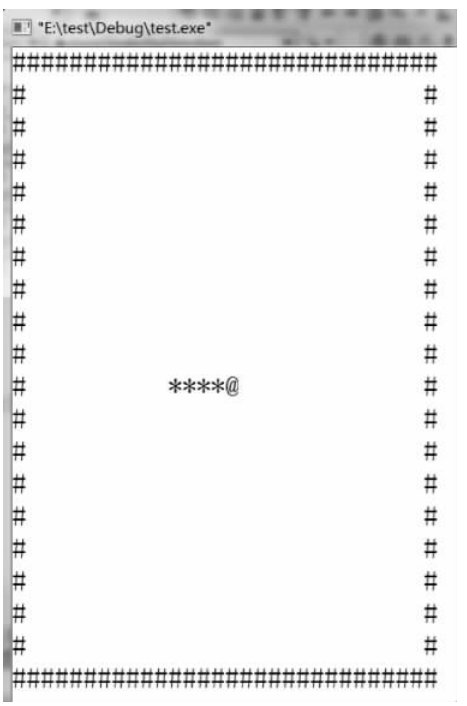

图 3-12 静止的小蛇效果

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

#define High 20                              // 游戏画面尺寸
#define Width 30

// 全局变量
int canvas[High][Width] = {0};               // 二维数组存储游戏画布中对应的元素
    // 0 为空格，-1 为边框＃，1 为蛇头@，大于 1 的正数为蛇身＊

void gotoxy(int x,int y)                     // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
```

```
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

void startup()                                  // 数据的初始化
{
    int i,j;

    // 初始化边框
    for (i = 0;i < High;i++)
    {
        canvas[i][0] = -1;
        canvas[i][Width-1] = -1;
    }
    for (j = 0;j < Width;j++)
    {
        canvas[0][j] = -1;
        canvas[High-1][j] = -1;
    }

    // 初始化蛇头位置
    canvas[High/2][Width/2] = 1;
    // 初始化蛇身,画布中的元素值分别为 2、3、4、5 等
    for (i = 1;i <= 4;i++)
        canvas[High/2][Width/2-i] = i+1;
}

void show()                                     // 显示画面
{
    gotoxy(0,0);                                // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");                    // 输出空格
            else if (canvas[i][j] == -1)
                printf("#");                    // 输出边框#
            else if (canvas[i][j] == 1)
                printf("@");                    // 输出蛇头@
            else if (canvas[i][j]> 1)
                printf("*");                    // 输出蛇身*
        }
        printf("\n");
    }
}

void updateWithoutInput()                       // 与用户输入无关的更新
{
```

```
}

void updateWithInput()                          // 与用户输入有关的更新
{
}

int main()
{
    startup();                                  // 数据的初始化
    while (1)                                   // 游戏循环执行
    {
        show();                                 // 显示画面
        updateWithoutInput();                   // 与用户输入无关的更新
        updateWithInput();                      // 与用户输入有关的更新
    }
    return 0;
}
```

3.4.2 小蛇的移动

实现小蛇的移动是贪吃蛇游戏的难点。图 3-13 列出了小蛇分别向右、向上运动后对应二维数组元素值的变化，从中我们可以得出实现思路。

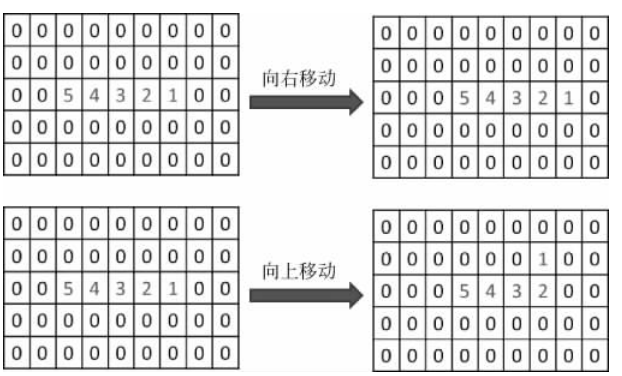

图 3-13 小蛇移动前后的效果

假设小蛇元素为 54321，其中 1 为蛇头、5432 为蛇身、最大值 5 为蛇尾。首先将所有大于 0 的元素加 1，得到 65432；将最大值 6 变为 0，即去除原来的蛇尾；再根据对应的移动方向将 2 对应方向的元素由 0 变成 1；如此即实现了小蛇的移动。小蛇向上移动的对应流程如图 3-14 所示。

本游戏的第二步为定义变量 int moveDirection 表示小蛇的移动方向，值为 1、2、3、4 分别表示小蛇向上、下、左、右方向移动，小蛇的移动在 moveSnakeByDirection()函数中实现。

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
```

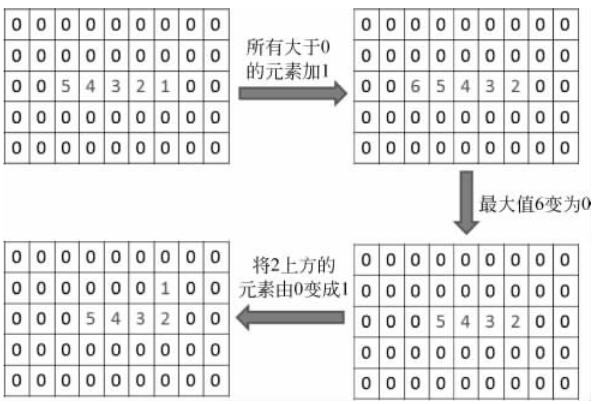

图 3-14　小蛇向上移动的流程

```
#define High 20                          // 游戏画面尺寸
#define Width 30

// 全局变量
int moveDirection;                       // 小蛇移动的方向,上、下、左、右分别用 1、2、3、4 表示
int canvas[High][Width] = {0};           // 二维数组存储游戏画布中对应的元素
            // 0 为空格 0,-1 为边框 #,1 为蛇头@,大于 1 的正数为蛇身 *

void gotoxy(int x,int y)                 // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
    SetConsoleCursorPosition(handle,pos);
}

// 移动小蛇
// 第一步扫描数组 canvas 的所有元素,找到正数元素都加 1
// 找到最大元素(即蛇尾巴),把其变为 0
// 找到等于 2 的元素(即蛇头),根据输出的上下左右方向把对应的另一个像素值设为 1(新蛇头)
void moveSnakeByDirection()
{
    int i,j;
    for (i = 1;i < High - 1;i++)
        for (j = 1;j < Width - 1;j++)
            if (canvas[i][j]> 0)
                canvas[i][j]++;

    int oldTail_i,oldTail_j,oldHead_i,oldHead_j;
    int max = 0;

    for (i = 1;i < High - 1;i++)
        for (j = 1;j < Width - 1;j++)
            if (canvas[i][j]> 0)
```

```
            {
                if (max < canvas[i][j])
                {
                    max = canvas[i][j];
                    oldTail_i = i;
                    oldTail_j = j;
                }
                if (canvas[i][j] == 2)
                {
                    oldHead_i = i;
                    oldHead_j = j;
                }
            }

    canvas[oldTail_i][oldTail_j] = 0;

    if (moveDirection == 1)          // 向上移动
        canvas[oldHead_i - 1][oldHead_j] = 1;
    if (moveDirection == 2)          // 向下移动
        canvas[oldHead_i + 1][oldHead_j] = 1;
    if (moveDirection == 3)          // 向左移动
        canvas[oldHead_i][oldHead_j - 1] = 1;
    if (moveDirection == 4)          // 向右移动
        canvas[oldHead_i][oldHead_j + 1] = 1;
}

void startup()                       // 数据的初始化
{
    int i,j;

    // 初始化边框
    for (i = 0;i < High;i++)
    {
        canvas[i][0] = -1;
        canvas[i][Width - 1] = -1;
    }
    for (j = 0;j < Width;j++)
    {
        canvas[0][j] = -1;
        canvas[High - 1][j] = -1;
    }

    // 初始化蛇头位置
    canvas[High/2][Width/2] = 1;
    // 初始化蛇身,画布中的元素值分别为 2、3、4、5 等
    for (i = 1;i <= 4;i++)
        canvas[High/2][Width/2 - i] = i + 1;

    // 初始小蛇向右移动
    moveDirection = 4;
}
```

```
void show()                          // 显示画面
{
    gotoxy(0,0);                     // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");         // 输出空格
            else if (canvas[i][j] == -1)
                printf("#");         // 输出边框#
            else if (canvas[i][j] == 1)
                printf("@");         // 输出蛇头@
            else if (canvas[i][j]> 1)
                printf("*");         // 输出蛇身*
        }
        printf("\n");
    }
    sleep(100);
}

void updateWithoutInput()            // 与用户输入无关的更新
{
    moveSnakeByDirection();
}

void updateWithInput()               // 与用户输入有关的更新
{
}

int main()
{
    startup();                       // 数据的初始化
    while (1)                        // 游戏循环执行
    {
        show();                      // 显示画面
        updateWithoutInput();        // 与用户输入无关的更新
        updateWithInput();           // 与用户输入有关的更新
    }
    return 0;
}
```

3.4.3　玩家控制小蛇移动

第三步的实现比较简单,在 updateWithInput()函数中按 a、s、d、w 键改变 moveDirection 的值,然后调用 moveSnakeByDirection()实现小蛇向不同方向的移动,如图 3-15 所示。

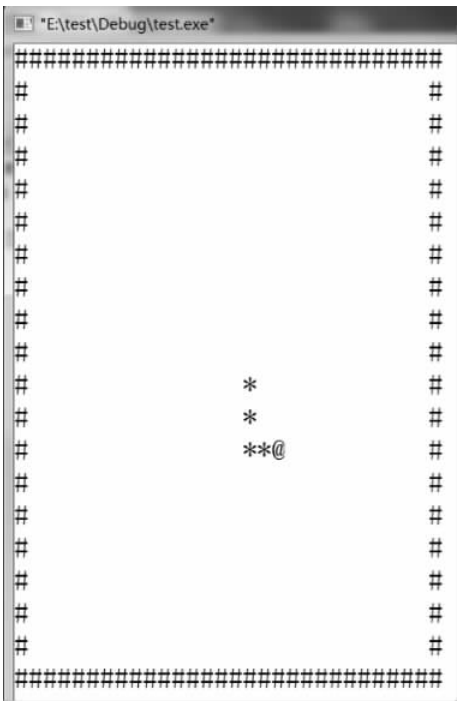

图 3-15　小蛇移动的效果

```
void updateWithInput()                  // 与用户输入有关的更新
{
    char input;
    if(kbhit())                         // 判断是否有输入
    {
        input = getch();                // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a')
        {
            moveDirection = 3;          // 位置左移
            moveSnakeByDirection();
        }
        else if (input == 'd')
        {
            moveDirection = 4;          // 位置右移
            moveSnakeByDirection();
        }
        else if (input == 'w')
        {
            moveDirection = 1;          // 位置上移
            moveSnakeByDirection();
        }
        else if (input == 's')
        {
            moveDirection = 2;          // 位置下移
```

```
                moveSnakeByDirection();
            }
        }
    }
```

3.4.4　判断游戏失败

第四步判断游戏失败，当小蛇和边框或自身发生碰撞时游戏失败，如图 3-16 所示。

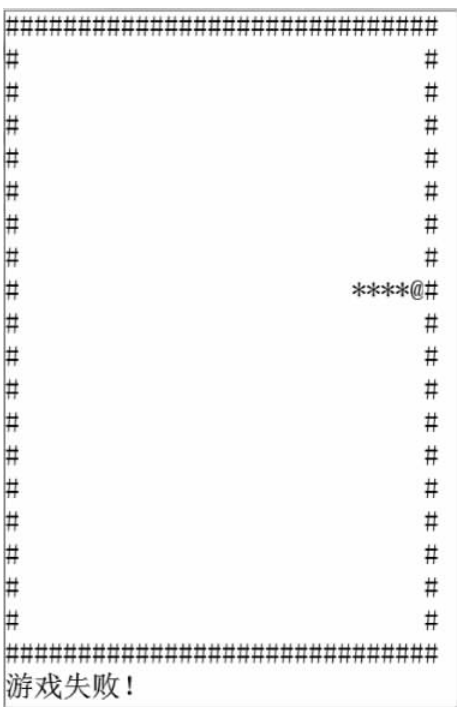

图 3-16　游戏失败效果

```
void moveSnakeByDirection()
{
    int i,j;
    for (i = 1;i < High - 1;i++)
        for (j = 1;j < Width - 1;j++)
            if (canvas[i][j]> 0)
                canvas[i][j]++;
    int oldTail_i,oldTail_j,oldHead_i,oldHead_j;
    int max = 0;
    for (i = 1;i < High - 1;i++)
        for (j = 1;j < Width - 1;j++)
            if (canvas[i][j]> 0)
            {
                if (max < canvas[i][j])
                {
                    max = canvas[i][j];
                    oldTail_i = i;
```

```
                    oldTail_j = j;
                }
                if (canvas[i][j] == 2)
                {
                    oldHead_i = i;
                    oldHead_j = j;
                }
            }
        canvas[oldTail_i][oldTail_j] = 0;
        int newHead_i,newHead_j;
        if (moveDirection == 1)          // 向上移动
        {
            newHead_i = oldHead_i - 1;
            newHead_j = oldHead_j;
        }
        if (moveDirection == 2)          // 向下移动
        {
            newHead_i = oldHead_i + 1;
            newHead_j = oldHead_j;
        }
        if (moveDirection == 3)          // 向左移动
        {
            newHead_i = oldHead_i;
            newHead_j = oldHead_j - 1;
        }
        if (moveDirection == 4)          // 向右移动
        {
            newHead_i = oldHead_i;
            newHead_j = oldHead_j + 1;
        }

        // 小蛇是否和自身撞或者和边框撞,游戏失败
        if (canvas[newHead_i][newHead_j]> 0 || canvas[newHead_i][newHead_j] == - 1)
        {
            printf("游戏失败!\n");
            exit(0);
        }
        else
            canvas[newHead_i][newHead_j] = 1;
    }
```

3.4.5 吃食物增加长度

第五步实现吃食物增加长度的功能,当二维数组 canvas[High][Width]的元素值为－2 时输出食物数值'F',如图 3-17 所示。当蛇头碰到食物时长度加 1。

其实现思路和 3.4.2 节中小蛇的移动类似,只需保持原蛇尾,不将最大值变为 0 即可。图 3-18 所示为小蛇向上移动吃到食物的对应流程。

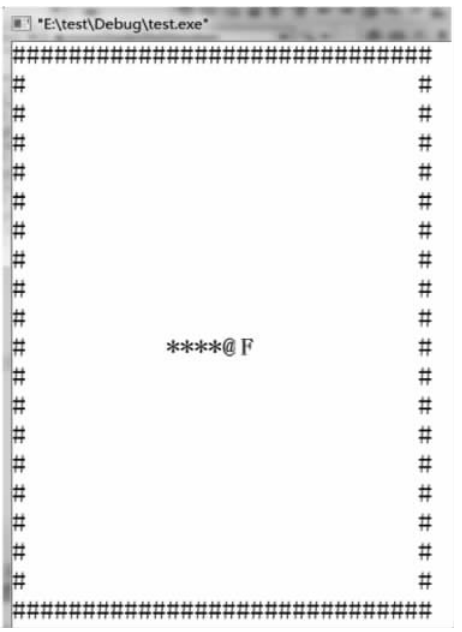

图 3-17　增加食物效果

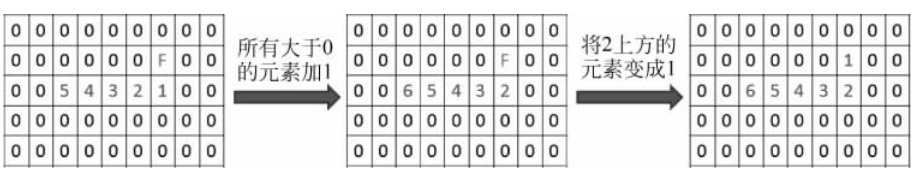

图 3-18　小蛇向上移动吃到食物的对应流程

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>

#define High 20                          // 游戏画面尺寸
#define Width 30

// 全局变量
int moveDirection;                       // 小蛇移动位置,上下左右分别用 1、2、3、4 表示
int food_x,food_y;                       // 食物的位置
int canvas[High][Width] = {0};           // 二维数组存储游戏画布中对应的元素
    // 0 为空格 0,-1 为边框 #,-2 为食物 F,1 为蛇头@,大于 1 的正数为蛇身 *

void gotoxy(int x,int y)                 // 将光标移动到(x,y)位置
{
    HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos;
    pos.X = x;
    pos.Y = y;
```

```
    SetConsoleCursorPosition(handle,pos);
}

// 移动小蛇
// 第一步扫描数组 canvas 的所有元素,找到正数元素都加 1
// 找到最大元素(即蛇尾巴),把其变为 0
// 找到等于 2 的元素(即蛇头),根据输出的上下左右方向把对应的另一个像素值设为 1(新蛇头)
void moveSnakeByDirection()
{
    int i,j;
    for (i = 1;i < High - 1;i++)
        for (j = 1;j < Width - 1;j++)
            if (canvas[i][j]> 0)
                canvas[i][j]++;

    int oldTail_i,oldTail_j,oldHead_i,oldHead_j;
    int max = 0;

    for (i = 1;i < High - 1;i++)
        for (j = 1;j < Width - 1;j++)
            if (canvas[i][j]> 0)
            {
                if (max < canvas[i][j])
                {
                    max = canvas[i][j];
                    oldTail_i = i;
                    oldTail_j = j;
                }
                if (canvas[i][j] == 2)
                {
                    oldHead_i = i;
                    oldHead_j = j;
                }
            }

    int newHead_i,newHead_j;

    if (moveDirection == 1)          // 向上移动
    {
        newHead_i = oldHead_i - 1;
        newHead_j = oldHead_j;
    }
    if (moveDirection == 2)          // 向下移动
    {
        newHead_i = oldHead_i + 1;
        newHead_j = oldHead_j;
    }
    if (moveDirection == 3)          // 向左移动
    {
        newHead_i = oldHead_i;
        newHead_j = oldHead_j - 1;
```

```
    }
    if (moveDirection == 4)          // 向右移动
    {
        newHead_i = oldHead_i;
        newHead_j = oldHead_j + 1;
    }

    // 如果新蛇头吃到食物
    if (canvas[newHead_i][newHead_j] == -2)
    {
        canvas[food_x][food_y] = 0;
        // 产生一个新的食物
        food_x = rand() % (High - 5) + 2;
        food_y = rand() % (Width - 5) + 2;
        canvas[food_x][food_y] = -2;

        // 原来的旧蛇尾留着,长度自动加 1
    }
    else                             // 否则,原来的旧蛇尾减掉,保持长度不变
        canvas[oldTail_i][oldTail_j] = 0;

    // 小蛇是否和自身撞或者和边框撞,游戏失败
    if (canvas[newHead_i][newHead_j]> 0 || canvas[newHead_i][newHead_j] == -1)
    {
        printf("游戏失败!\n");
        sleep(2000);
        system("pause");
        exit(0);
    }
    else
        canvas[newHead_i][newHead_j] = 1;
}

void startup()                       // 数据的初始化
{
    int i,j;

    // 初始化边框
    for (i = 0;i < High;i++)
    {
        canvas[i][0] = -1;
        canvas[i][Width - 1] = -1;
    }
    for (j = 0;j < Width;j++)
    {
        canvas[0][j] = -1;
        canvas[High - 1][j] = -1;
    }

    // 初始化蛇头位置
    canvas[High/2][Width/2] = 1;
```

```
    // 初始化蛇身,画布中的元素值分别为 2、3、4、5 等
    for (i = 1;i <= 4;i++)
        canvas[High/2][Width/2 - i] = i + 1;

    // 初始小蛇向右移动
    moveDirection = 4;

    food_x = rand() % (High - 5) + 2;
    food_y = rand() % (Width - 5) + 2;
    canvas[food_x][food_y] = -2;
}

void show()                          // 显示画面
{
    gotoxy(0,0);                     // 光标移动到原点位置,以下重画清屏
    int i,j;
    for (i = 0;i < High;i++)
    {
        for (j = 0;j < Width;j++)
        {
            if (canvas[i][j] == 0)
                printf(" ");         // 输出空格
            else if (canvas[i][j] == -1)
                printf("#");         // 输出边框#
            else if (canvas[i][j] == 1)
                printf("@");         // 输出蛇头@
            else if (canvas[i][j]> 1)
                printf("*");         // 输出蛇身*
            else if (canvas[i][j] == -2)
                printf("F");         // 输出食物 F
        }
        printf("\n");
    }
    sleep(100);
}

void updateWithoutInput()            // 与用户输入无关的更新
{
    moveSnakeByDirection();
}

void updateWithInput()               // 与用户输入有关的更新
{
    char input;
    if(kbhit())                      // 判断是否有输入
    {
        input = getch();             // 根据用户的不同输入来移动,不必输入回车
        if (input == 'a')
        {
            moveDirection = 3;       // 位置左移
            moveSnakeByDirection();
```

```
        }
        else if (input == 'd')
        {
            moveDirection = 4;      // 位置右移
            moveSnakeByDirection();
        }
        else if (input == 'w')
        {
            moveDirection = 1;      // 位置上移
            moveSnakeByDirection();
        }
        else if (input == 's')
        {
            moveDirection = 2;      // 位置下移
            moveSnakeByDirection();
        }
    }
}

int main()
{
    startup();                      // 数据的初始化
    while (1)                       // 游戏循环执行
    {
        show();                     // 显示画面
        updateWithoutInput();       // 与用户输入无关的更新
        updateWithInput();          // 与用户输入有关的更新
    }
    return 0;
}
```

3.4.6　小结

本节用 C 语言实现了经典的贪吃蛇游戏，大家是不是很有成就感？

思考题：

1. 增加道具，吃完可以加命或减速。
2. 尝试实现双人版贪吃蛇游戏(可参考 5.4 节中内容)。

3.5　版本管理与团队协作

在实现复杂的游戏程序时往往需要开发多个版本，比如 3.4 节中的贪吃蛇需要 5 个步骤逐步完善。随着程序越来越复杂，多个版本代码的保存、比较、回溯、修改是开发中不可缺少的功能。另外，复杂的游戏可以由两三名同学合作开发，如何实现高效的多人协作将是迫切需要解决的问题。

3.5.1　SVN 简介

Subversion(SVN)是一个常用的代码版本管理软件，可以选择 VisualSVN Server 服务

器和 TortoiseSVN 客户端搭配使用。VisualSVN 的官方下载地址为“http://subversion.apache.org/packages.html”，TortoiseSVN 客户端的官方下载地址为“http://tortoisesvn.net/downloads.html”。对于 SVN 的安装与配置可以查看官网中的帮助，或在线搜索相应教程。

配置完成后，可以在 VisualSVN 服务器中建立账号，创建代码仓库。利用 TortoiseSVN 客户端可以在本地计算机下载服务器上最新版本的代码，如图 3-19 所示。

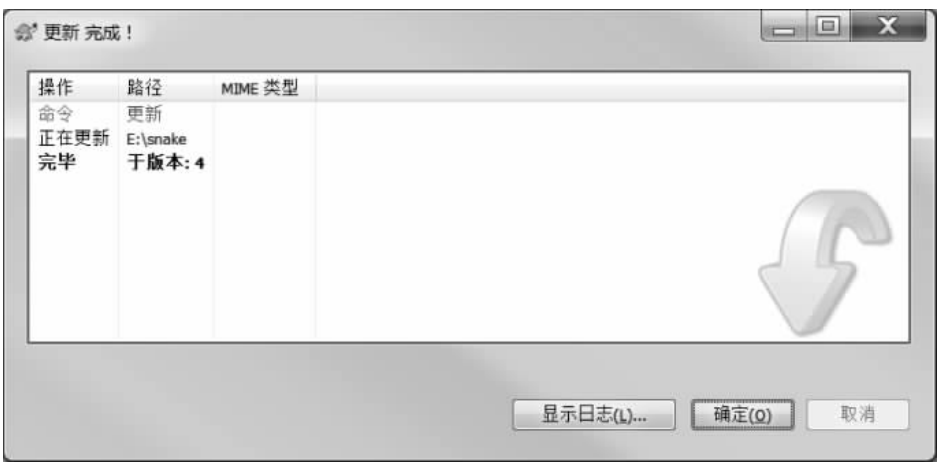

图 3-19　TortoiseSVN 更新代码版本

在本地修改代码后可以用 TortoiseSVN 提交最新代码，VisualSVN 服务器会自动更新，如图 3-20 所示。

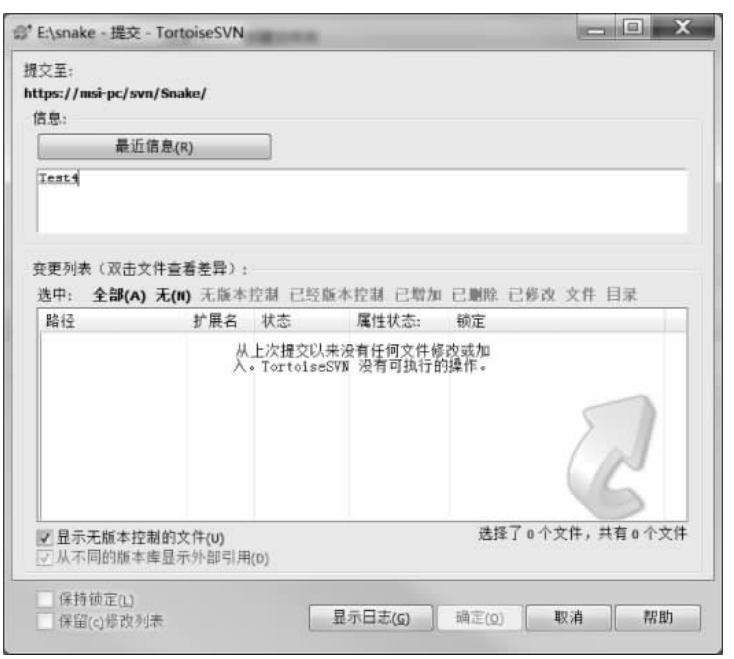

图 3-20　TortoiseSVN 提交最新代码

用户也可以查看日志，能够看到实现对应版本代码的用户账号、修改时间、备注等信息，也可以随时切换到任一版本的代码，如图 3-21 所示。

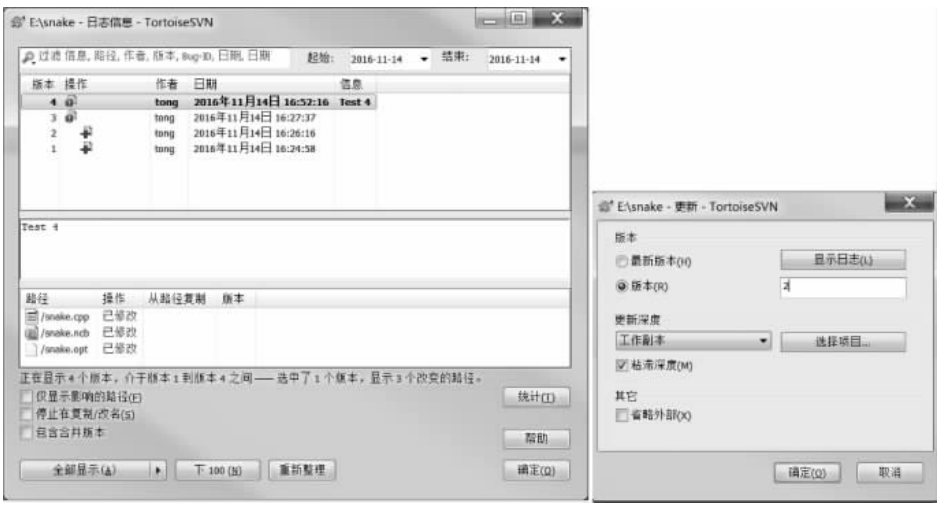

图 3-21　查看日志与切换版本

使用 SVN 可以方便地比较不同版本代码之间的差别，图 3-22 中的阴影部分为当前版本修改的代码。

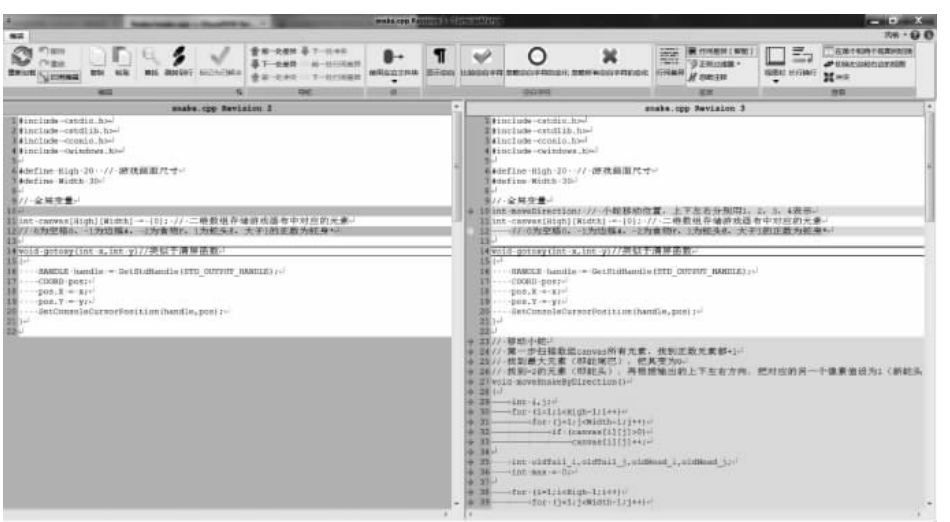

图 3-22　比较不同版本代码间的差别

用户也可以在 SVN 中分配多个账号，以方便团队协作开发，例如多人修改与更新、查看不同作者的工作进度、合并多人修改的代码等。在具体配置时可以使用同一网段的计算机搭建内网服务器，也可以采用阿里云、腾讯云等搭建外网 SVN 服务器。

3.5.2　开发实践

本节尝试开发一个勇闯地下 100 层的小游戏，并用 SVN 进行版本管理，如图 3-23 所

示。小人'0'可以站在板'----'上左右移动，板会随机出现且一直上升，如果小人掉落到最下方或碰到最上方，游戏失败。

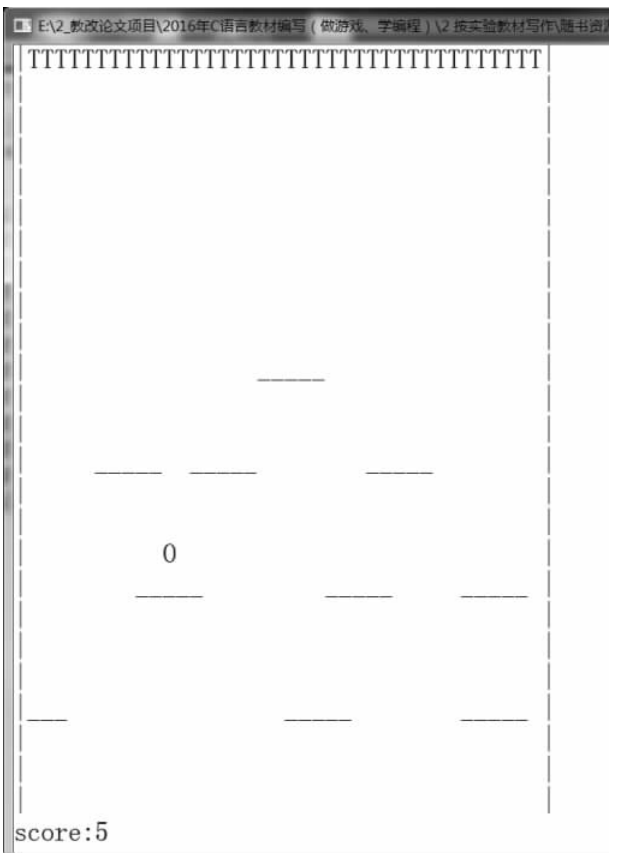

图 3-23　勇闯地下 100 层游戏效果

读者可以按照以下思路分步骤实现：

1. 一块板的上升。
2. 多块随机板的上升。
3. 小人随着板上升。
4. 小人的左右移动。
5. 小人的重力感下落。
6. 死亡的判断。
7. 记录分数。
8. 随着分数增加难度上升。

其代码可参考“\随书资源\第 3 章\3.5 勇闯地下 100 层\”。

3.5.3　小结

用好 SVN 可以方便地进行代码的版本管理与团队协作，读者可以在游戏开发中实践体会。初学者也可以使用码云等在线代码托管平台。