```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <termios.h>

#define N       4 // 表达图形的数组
#define BASE    7 // 7种图形
#define CHANGE  4 // 4种变换

#define ROW     20// 游戏背景
#define COL     14

#define BAND    2 // 边界和停落的图形
#define BLOCK   1 // 运动图形


// 图形结构
struct blocks {
    int space[N][N];
}blocks[BASE][CHANGE];

// 棋盘数组 // 前两行后两行  前两列  后两列都是边界
int els[ROW][COL];
int row, col; // 方块在棋盘中对应的行列

// 枚举
enum {BLACK, RED, GREEN, YELLOW, BLUE, PURPLE, DARKGREEN};
// 移动方向
enum {UP, LEFT=1, RIGHT, DOWN};

int colour; // 当前方块颜色
int colour_arr[ROW][COL]; // 存储每一个方块的颜色
int colour_next;

int base; // 当前方块类型 BASE
int change; // 当前方块四种变换的一种
int next_change;
int base_next; // 下一个方块类型

struct termios new, old;

void initBlocks();
```

```c
void showBg();
void selectColour(int select);
void writeInBlock();
void printBlock();
void run();
void execCmd(int cmd);
void judgeLast();
int judgeMove(int move);
void debug();
void nextBlock();
void judgeLine();
int judgeChange();
int judgeOver();

void alarmHandler(int s)
{
    alarm(1);
    // debug();
    if (judgeMove(DOWN) == 1) {
        execCmd(DOWN);
    } else {
        judgeLast(); // 停留棋盘
        alarm(0);
        run();
    }
}

int main(void)
{
    char c;

    // 信号使得图形自由下落
    signal(SIGALRM, alarmHandler);

    // 按键设置    不回显  关闭缓存
    // 初始化所有图形
    initBlocks();
    // 画背景
    showBg();

    // 关闭光标显示
    printf("\033[?25l");

    // 随机产生方块图形
    srand(time(NULL));
    base_next = rand() % BASE;
    // 随机产生当前方块的颜色
    colour_next = rand() % 7;
    next_change = next_change % CHANGE;
```

```
    run();

    // 关闭标准输入回显和缓存
    tcgetattr(0, &old);
    new = old;
    new.c_lflag = new.c_lflag & ~(ICANON | ECHO);

    tcsetattr(0, TCSANOW, &new);

    while (1) {
        c = getchar();
        switch (c) {
            case 'q':
            case 'Q':
                alarm(0);
                printf("\033[10;20 游戏结束");
                break;
            case 'a':
            case 'A':
                // 左移
                if (judgeMove(LEFT)) {
                    execCmd(LEFT);
                }
                break;
            case 'd':
            case 'D':
                // 右移动
                if (judgeMove(RIGHT))
                    execCmd(RIGHT);
                break;
            case 's':
                // 加速下落
                if (judgeMove(DOWN)) {
                    execCmd(DOWN);
                } else {
                    judgeLast(); // 停留棋盘
                    alarm(0);
                    run();
                }
                break;
            case 'w':
                if (judgeChange()) {
                    execCmd(UP);
                }
                break;
        }
    }
```

```
    return 0;
}

/*
 游戏运行
 产生新图形
 */
void run()
{

    alarm(1);
    // 方块初始行列
    row = 2;
    col = 5;

    base = base_next;
    colour = colour_next;
    change = next_change;
    // 随机产生方块图形
    srand(time(NULL));
    base_next = rand() % BASE;
    // 随机产生当前方块的颜色
    colour_next = rand() % 7;
    next_change = next_change % CHANGE;


    // 判断游戏是否结束
    if (judgeOver()) {
        printf("\033[15;20H游戏结束!!!");
        tcsetattr(0, TCSANOW, &old);
        exit(0); // 进程终止
    }

    writeInBlock(); // 方块写入棋盘

}



/*
 初始化图形
 */
void initBlocks()
{
    int row, col;
    int temp[N][N] = {};
    int base, change;
```

```
    // blocks[0]存方块 blocks[1] 存z blocks[2] 存反z
    for (row = 1; row <= 2; row ++) {
        for (col = 1; col <= 2; col ++) {
            // 方块
            blocks[0][0].space[row][col] = 1;
        }
    }
    // z 和  反z
    for (col = 0; col < 2; col ++) {
        // z
        blocks[1][0].space[1][col] = 1;
        blocks[1][0].space[2][col+1] = 1;
        // 反z
        blocks[2][0].space[1][col+1] = 1;
        blocks[2][0].space[2][col] = 1;
    }

    // blocks[3]存7 blocks[4]反7 blocks[5] 存| blocks[6] 存土
    for (row = 1; row < 4; row ++) {
        // 7
        blocks[3][0].space[row][1] = 1;
        blocks[3][0].space[1][0] = 1;

        // 反7
        blocks[4][0].space[row][1] = 1;
        blocks[4][0].space[1][2] = 1;
    }

    // |
    for (row = 0; row < 4; row ++) {
        blocks[5][0].space[row][1] = 1;
    }
    // 土
    for (col = 0; col < 3; col ++) {
        blocks[6][0].space[1][1] = 1;
        blocks[6][0].space[2][col] = 1;
    }

    for (base = 0; base < BASE; base ++) {
        for (change = 0; change < CHANGE-1; change ++) {
            // 保存变换前图形
            for (row = 0; row < N; row ++) {
                for (col = 0; col < N; col ++) {
                    temp[row][col] = blocks[base][change].space[row]
[col];
                }
            }
            // 变换
            for (row = 0; row < N; row ++) {
```

```
                for (col = 0; col < N; col ++) {
                    blocks[base][change+1].space[row][col] = temp[N-
1-col][row];
                }
            }
        }
    }

}


// 展示背景
void showBg()
{
    int i, j, k;

    printf("\033[2J"); // 清屏
    // ROW COL
    printf("\033[5;10H\033[45m--====================----------------
---\033[0m\n");

    for (i = 0; i < ROW-4; i++) {
        printf("\033[%d;10H\033[45m||\033[0m", 6+i);
        for (j = 0; j < COL-4; j++) {
            printf("  ");
        }
        printf("\033[45m||");
        if (i == 6 || i == 10) {
            printf("\033[45m---------------||\033[0m\n");
        } else
            printf("\033[0m\t\t\033[45m||\033[0m\n");
    }
    printf("\033[22;10H\033[45m--====================--------------
----\033[0m\n");


    // 初始化棋盘数组边界
    // 前两行  + 后两行
    for (i = 0, j = ROW-2; i < 2 && j < ROW; i++, j++) {
        for(k = 0; k < COL; k++) {
            els[i][k] = BAND;
            els[j][k] = BAND;
        }
    }

    // 前两列 + 后两列
    for (i = 2; i < ROW-2; i++) {
        for (j = 0, k = COL-2; j < 2 && k < COL; j++, k++) {
            els[i][j] = BAND;
```

```
            els[i][k] = BAND;
        }
    }
}

// 选择颜色
void selectColour(int select)
{

    // enum {BLACK, RED, GREEN, YELLOW, BLUE, PURPLE, DARKGREEN};
    switch (select){
        case BLACK:
            printf("\033[40m");
            break;
        case RED:
            printf("\033[41m");
            break;
        case GREEN:
            printf("\033[42m");
            break;
        case YELLOW:
            printf("\033[43m");
            break;
        case BLUE:
            printf("\033[44m");
            break;
        case PURPLE:
            printf("\033[45m");
            break;
        case DARKGREEN:
            printf("\033[46m");
            break;
    }

}

/*
 打印棋盘
 */
void printBlock()
{
    int i, j;

    printf("\033[6;12H");
    for (i = 2; i < ROW-2; i++) {
        for (j = 2; j < COL-2; j++) {
            if (els[i][j] == BLOCK) {
                // 正在下落的方块
                selectColour(colour);
```

```
                printf("[]\033[0m");
            } else if (els[i][j] == BAND) {
                // 已经下落完停留的方块
                selectColour(colour_arr[i][j]);
                printf("[]\033[0m");
            } else
                printf("  ");
        }
        printf("\n\033[11C"); // 让每一行起始列都从12列开始
    }
    nextBlock();
}

// 方块写入棋盘
void writeInBlock()
{
    int i, j;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (els[row+i][col+j] != BAND) {
                // 防止覆盖了已经下落的方块
                els[i+row][j+col] = blocks[base][change].space[i]
[j];
            }
        }
    }
    printBlock();
}

/*
 判断方块能否运动
 能－－－》返回1
 不能----> 返回2
 参数:move
    1 left
    2 right
    3 down
    enum {LEFT=1, RIGHT, DOWN};
 */
int judgeMove(int move)
{
    // 移动的行列
    int f_row ,f_col;
    int i, j;

    f_row = f_col = 0;

    switch (move) {
```

```
        case LEFT:
            f_col = -1;
            break;
        case RIGHT:
            f_col = 1;
            break;
        case DOWN:
            f_row = 1;
            break;
        default:
            break;
    }
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (els[row+i][col+j] == 1) {
                if (els[row+i+f_row][col+j+f_col] == BAND)
                    return 0;
            }
        }
    }

    return 1;
}

/*
 消除上一个状态
 */
void cancelState(int state)
{
    int i, j;
    int last_row, last_col;

    last_row = last_col = 0;

    switch (state) {
        case UP:
            break;
        case LEFT:
            last_col = 1;
            break;
        case RIGHT:
            last_col = -1;
            break;
        case DOWN:
            last_row = -1;
            break;
        default:
            break;
    }
```

```
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if ( els[i+row+last_row][j+col+last_col]!= BAND)
                els[i+row+last_row][j+col+last_col] = 0;
        }
    }

    printBlock();
}

/*
 方块停留棋盘 ??????????????????????????????
 */
void judgeLast()
{
    int i, j;

    for (i = 2; i < ROW-2; i++) {
        for (j = 2; j < COL-2; j++) {
            if (els[i][j] == BLOCK) {
                // 运动的方块
                els[i][j] = BAND;
                colour_arr[i][j] = colour;
            }
        }
    }

    // 判断是否消行
    judgeLine();

    printBlock();
}

/*
 执行动作
 */
void execCmd(int cmd)
{
    switch (cmd) {
        case UP:
            change = (change + 1) % CHANGE;
            cancelState(cmd);
            break; // ??
        case LEFT:
            col --;
            cancelState(cmd);
            break;
        case RIGHT:
            col ++;
```

```
            cancelState(cmd);
            break;
        case DOWN:
            row ++;
            // printf("当前图形在row:%d, col:%d\n", row, col);
            cancelState(cmd);
            break;
        default:
            break;
    }

    writeInBlock();
}

/*
 调试模块
 */
void debug()
{
    int i, j;

    printf("\033[5;58H");

    for(i = 0; i < ROW; i++) {
        for (j = 0; j < COL; j++) {
            printf("%d\033[0m", els[i][j]);
        }
        printf("\n\033[57C");
    }

}


/*
 显示下一个图形
 */
void nextBlock()
{
    int i, j;

    printf("\033[7;38H");

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (blocks[base_next][next_change].space[i][j]) {
                selectColour(colour_next);
                printf("[]\033[0m");
            } else
                printf("  ");
```

```
        }
        printf("\n\033[37C");
    }
}

void execCancelLine(int r)
{
    int i, j;

    for (i = r; i > 2; i--) {
        for (j = 2; j < COL-2; j++) {
            els[i][j] = els[i-1][j];
        }
    }

    for (j = 2; j < COL-2; j++) {
        els[2][j] = 0; // 棋盘最上方
    }

}

/*
 消行模块
 */
void judgeLine()
{
    int i, j;
    int count = 0;
    int t;

    for (i = ROW-3; i >= 2; i--) {
        count = 0;
        for (j = 2; j < COL-2; j++) {
            if (els[i][j] == BAND)
                count ++;
        }
        if (count == COL-4) {
            // 一行满了
            t = i;
            execCancelLine(t);
            // 加分  判断是否升级
            i = t + 1; // 重落下来哪一行继续判断

        }
    }
}


/*
```

```
 旋转
 */
int judgeChange()
{
    int i, j;
    int tmp;
    int cnt = 0;

//  debug();

    tmp = (change + 1) % CHANGE;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (blocks[base][tmp].space[i][j]) {
                if (els[row+i][col+j] != 2) {
                    // 当前下标i,j方块不重叠棋盘边界/已停落方块
                    cnt ++;
                    printf("\033[30;12Hcnt:%d", cnt);
                    if (cnt == 4) // 旋转的新图形不会重叠
                        return 1;
                }
            }

        }
    }

    return 0;
}

/*
 判断游戏是否结束
 */
int judgeOver()
{
    int i, j;

    // 第三行有图形
    for (j = 2; j < COL-2; j++) {
        // 第三行
        if (els[2][j] == BAND)
            return 1;
    }

    // 下一个图形进入不了棋盘
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (blocks[base][change].space[i][j] == 1) {
                if (els[row+i][col+j] == 2)
```

```
                    return 1;
                }
            }
        }

        return 0;
    }
```