C程序设计

E3. 2048游戏设计与实现

厦门大学信息学院软件工程系

黄炜 副教授

作业要求

- 题目
  - 设计并实现一个2048游戏。
- 要求
  - 需求分析
  - 程序设计
  - 程序实现
  - 程序测试与维护

纳尼？只有这一句话？
大部分需求往往只有一句话。

需求分析

- 需求是不明确的
  - 大部分的需求一开始都是不明确的。
  - 由需求分析人员和项目委托方共同完成。
  - 最终将明确的需求作为项目合同的附件，按期责功。
- 方法
  - 通过搜索相近的产品，进一步细化需求。
  - 向项目委托方进一步确认。

由于需求分析有专门的课程，作业提供完整的需求。

游戏规则

- 题目要求

1、2048有一个4*4的棋盘。

2、开始时棋盘内随机出现两个数字，出现的数字仅可能为2或4。

3、玩家可以选择上下左右四个方向，棋盘内的棋子按该方向位移。

a) 若有棋子在移动方向的一侧（如：右侧）为空，则将其移动到该位。

b) 若有棋子在移动方向的一侧（如：右侧）为同样的数字，则将其与该侧数字相加。不可将两个以上数字相加。

c) 若该方向无棋子满足a)或b)条件，则不移动棋子。

4、每有效移动一步，棋盘的空位（无数字处）随机出现一个数字（为2或4）。

5、若四个方向都满足第3条c)规则，则判负。

6、若出现2048，则判胜。

从游戏规则到功能设计

- 数据结构： 4*4的整型矩阵（ 或者16个变量 ）
  - 来源：1、2048有一个4*4的棋盘。
  - 不要去考虑太多的问题（ 如*N***N* ），否则就没完没了了！
- 功能1：初始化棋盘（ 实现方式为函数 ）
  - 来源：2、开始时棋盘内随机出现两个数字，可能为2或4。
  - 初步想法：清零、生成随机数决定2或者4，赋值。
- 功能2：显示棋盘
  - 初步想法：两个for循环，打印数组，注意对齐问题

从游戏规则到功能设计

- 功能3：上下左右位移棋盘
  - 来源：3、玩家可以选择上下左右四个方向，棋盘内的棋子按该方向位移。
  - 不要去考虑太多的问题（如棋子相同时是否合并、上下左右是否合在一个函数搞定），否则就没完没了了！
- 功能3.1：按行整理棋盘（用于左、右位移）
  - 来源：若有棋子在移动方向的一侧（如：右侧）为空，则将其移动到该位。

从游戏规则到功能设计

- 功能3.2：按行合并棋子（用于左、右位移）
  - 来源：若有棋子在移动方向的一侧（如：右侧）为同样的数字，则将其与该侧数字相加。不可将两个以上数字相加。
- 功能3.3、3.4：按列整理棋盘、按列合并棋子
- 功能4：随机选择空位，随机增加一个棋子
  - 来源：4、每有效移动一步，棋盘的空位（无数字处）随机出现一个数字（为2或4）。
  - 仍然不要考虑细节（随机怎么实现），否则就没完没了了。

从游戏规则到功能设计

- 功能4：判决胜负
  - 来源：5、若四个方向都满足第3条c)规则，则判负。
  - 来源：6、若出现2048，则判胜。
  - 仍然不要考虑细节（如：结束以后如何重新开始）

看着大佬一气呵成，

好方……

好的架构是进化来的，

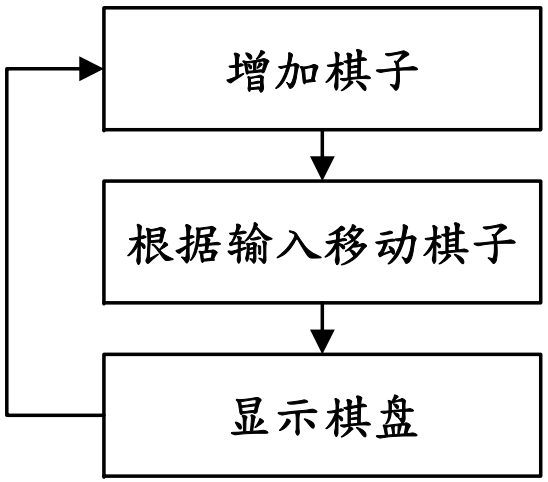
不是设计来的

模块化设计

- 自顶向下
  - 顶部架构设计，再一步步细化
- 自底向上
  - 先收集核心代码的片段，一步步构造
- 双管齐下
  - 两边同时进行

走出第一步

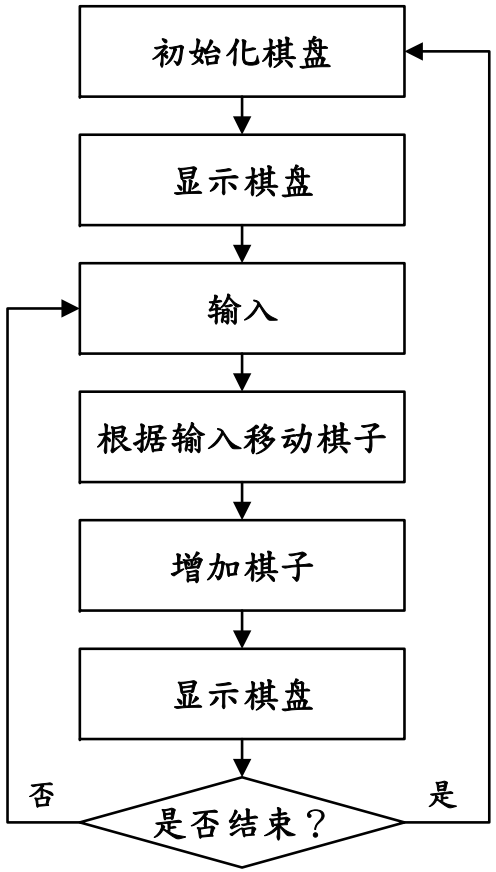
- 总体设计和对应的代码框架




```
int main()
{
    while (1)
    {
        // add a number
        // move by input
        // print new matrix
    }
    return 0;
}
```

走出第一步

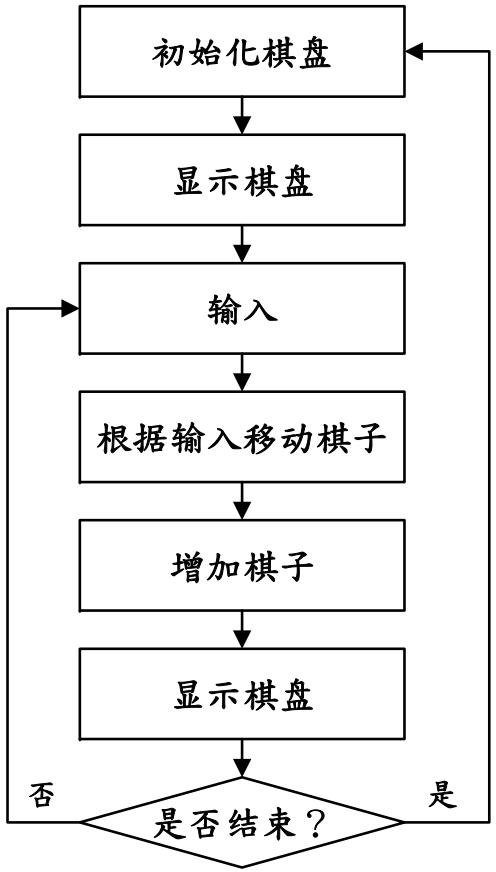
- 完善框架




```
int main()
{
    // initialize board
    // display board
    while (1)
    {
        // get input
        // move by input
        // fill blank
        // display board
        // if it is over, break
    }
    return 0;
}
```

走出第一步

- 完善框架




```
int main(int argc, char *argv[]) {
    initBoard();
    drawBoard();
    while (1) {
        c=getInput();
        suc = moveByDirect(c);
        addRandom();
        drawBoard();
        if (gameEnded()) {
            showResult();
            break;
        }
    }
    return 0;
}
```

```c
#include <stdio.h>

void initBoard() { printf("initBoard\n"); }
void drawBoard() { printf("drawBoard\n"); }
char getInput() { printf("getInput\n"); return 0; }
void addRandom() { printf("addRandom\n"); }
int gameEnded() { printf("gameEnded\n"); return 1; }
void showResult() { printf("showResult\n"); }
void moveByDirect(char c) { printf("moveByDirect(%c)\n", c); }

int main(int argc, char *argv[])
{
    initBoard();
    drawBoard();
```

```
    while (1)
    {
        char c = getInput();
        moveByDirect(c);
        addRandom();
        drawBoard();
        if (gameEnded())
        {
            showResult();
            break;
        }
    }
    return 0;
}
```

急需实现的函数

- 函数列表

函数名	解决
initBoard	未完成
addRandom	未完成
drawBoard	未完成
getInput	未完成
moveByDirect(char)	未完成
gameEnded	未完成
showResult	未完成

急需实现的函数

- 急需实现的函数
  - 初始化棋盘：initBoard();
  - 画棋盘：drawBoard();
- 引发的问题：数据结构
  - 声明尺寸4*4的int数组为棋盘
- 原因
  - 程序应先有初步的感受

初始化棋盘：initBoard();

- 肯定想得到的
  - 棋盘清空
  - 增加2个棋子：addRandom();两次
    - 这里可以先暂不实现
- 可能暂时想不到的
  - 得分清零

```c
#include <stdio.h>
#define SIZE 4
int board[SIZE][SIZE] = { 0 };

void addRandom() { printf("addRandom\n"); }

void cleanBoard() { printf("cleanBoard\n"); }

void initBoard()
{
    cleanBoard();
    addRandom();
    addRandom();
}
```

棋盘清空：cleanBoard();

- 详细流程
  - 遍历第一维度（x：0→SIZE）
  - 对每个第一维度，遍历第二维度（y：0→SIZE）
  - 对每个遍历，将当前格子设置为0（b[x][y]=0）

```
void cleanBoard() {
    int x, y;
    for (x = 0; x<SIZE; x++)
    {
        for (y = 0; y<SIZE; y++)
        {
            board[x][y] = 0;
        }
    }
}

void initBoard()
{
    cleanBoard();
    addRandom();
    addRandom();
}
```

画棋盘：drawBoard();

- 详细流程
  - 棋盘打印：两个for循环，printf语句
  - 对0点的处理：if语句

```
void drawBoard()
{
    int x, y;
    printf(" ***** Game 2048 ***** \n");
    for (y = 0; y<SIZE; y++)
    {
        for (x = 0; x<SIZE; x++)
        {
            printf("[%8d]", board[x][y]);
        }
        printf("\n");
    }
    printf("\n");
}
```

```
void drawBoard()
{
    int x, y;
    printf(" ***** Game 2048 ***** \n");
    for (y = 0; y<SIZE; y++)
    {
        for (x = 0; x<SIZE; x++)
        {
            if (board[x][y] != 0)
                printf("[%8d]", board[x][y]);
            else
                printf("[%8s]", " ");
        }
        printf("\n");
    }
    printf("\n");
}
```

需要实现的函数

- 回顾函数列表

函数名	解决
initBoard	已完成
initBoard >> cleanBoard	已完成
initBoard >> addRandom	未完成
drawBoard	已完成
getInput	未完成
moveByDirect(char)	未完成
gameEnded	未完成
showResult	未完成

急需实现的函数

- 急需实现的函数
  - 增加棋子：addRandom();
- 原因
  - 没有棋子则无从开始
- 困难
  - 随机数不知道怎么实现时，可先不随机
  - 即：在第一个空格处放置棋子
    - 方便实现，方便使用

```
void addRandom() // TODO: it is not random
{
    int x, y;

    for (x = 0; x < SIZE; x++)
    {
        for (y = 0; y < SIZE; y++)
        {
            if (board[x][y] == 0)
            {
                board[x][y] = 2;
                break;
            }
        }
        if (y < SIZE)
            break;
    }
}
```

需要实现的函数

- 回顾函数列表

函数名	解决
initBoard	已完成
initBoard >> cleanBoard	已完成
initBoard >> addRandom	简化替代
drawBoard	已完成
getInput	未完成
moveByDirect(char)	未完成
gameEnded	未完成
showResult	未完成

需要实现的函数

- 获取输入：getInput()
  - 最简单的思路：利用getchar()函数从键盘获取输入
- 按输入移动棋子：moveByDirect(char)
  - 先实现向上移动棋子，后实现其它方向
  - 其它移动
    - 左移动：顺时针转棋盘1次，上移动，顺时针转棋盘3次
    - 下移动：顺时针转棋盘2次，上移动，顺时针转棋盘2次
    - 右移动：顺时针转棋盘3次，上移动，顺时针转棋盘1次

```c
char getInput()
{
    printf("Please input a command: ");
    return getchar();
}
```

```
void moveLeft() { printf("moveLeft\n"); }
void moveRight() { printf("moveRight\n"); }
void moveUp() { printf("moveUp\n"); }
void moveDown() { printf("moveDown\n"); }

void moveByDirect(char c) {
    printf("moveByDirect(%c)\n", c);
    switch (c) {
    case 'a':
        moveLeft();  break;
    case 'd':
        moveRight(); break;
    case 'w':
        moveUp();    break;
    case 's':
        moveDown();  break;
    }
}
```

```
void moveLeft() { printf("moveLeft\n"); }
void moveRight() { printf("moveRight\n"); }
void moveUp() { printf("moveUp\n"); }
void moveDown() { printf("moveDown\n"); }

void moveByDirect(char c) {
    printf("moveByDirect(%c)\n", c);
    switch (c) {
    case 'a':
        moveLeft();  break;
    case 'd':
        moveRight(); break;
    case 'w':
        moveUp();    break;
    case 's':
        moveDown();  break;
    }
}
```

```
void rotateBoard() { printf("rotateBoard\n"); }

void moveUp() { printf("moveUp\n"); }

void moveLeft()
{
    rotateBoard();
    moveUp();
    rotateBoard();
    rotateBoard();
    rotateBoard();
}
```

```
void moveDown()
{
    rotateBoard();
    rotateBoard();
    moveUp();
    rotateBoard();
    rotateBoard();
}

void moveRight()
{
    rotateBoard();
    rotateBoard();
    rotateBoard();
    moveUp();
    rotateBoard();
}
```

需要实现的函数

- 回顾函数列表

函数名	解决
initBoard	已完成
initBoard >> addRandom	简化替代
drawBoard	已完成
getInput	已完成
moveByDirect(char)	已完成
moveByDirect(char) >> moveUp()	未完成
moveByDirect(char) >> moveDown/Left/Right()	已完成
moveByDirect(char) >> rotate()	未完成
gameEnded	未完成
showResult	未完成

移动棋盘：moveUp()

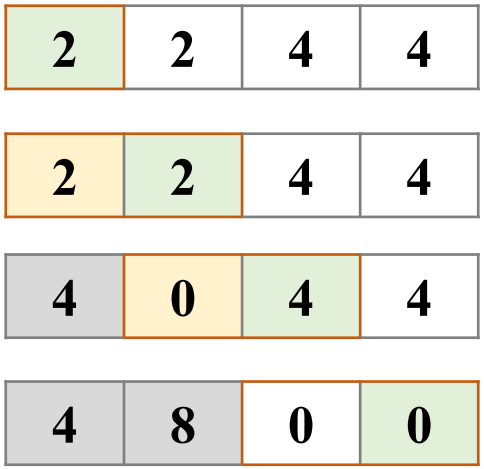
- 向上移动详细流程
  - 循环：*N*列分别向上移动（sideArray()）

```
void slideArray(int arr[], unsigned int length)
{
    printf("slideArray(arr, %u)\n", length);
}

void moveUp()
{
    int x;
    for (x = 0; x<SIZE; x++)
    {
        slideArray(board[x], SIZE);
    }
}
```

移动棋盘列：slideArray()

- 实现并不唯一
- 思路：遍历每个元素，找该元素应放置的位置
  - 查找的起止位置
    - 起始位置默认为0，在遇到“翻倍”合并时前进
    - 终止位置和遍历的当前位置相同
  - 处理规则
    - 目标相同：翻倍；目标为0：翻倍
    - 当前位置：设置为0



```
unsigned int findTarget(int array[SIZE], unsigned int x,
unsigned int stop)
{
    printf("findTarget(arr, %u, %u)\n", x, stop);
    return 0;
}
```

```
void slideArray(int array[], unsigned int length) {
    uint8_t x, t, stop = 0;
    for (x = 0; x < SIZE; x++) {
        if (array[x] != 0) {
            t = findTarget(array, x, stop);
            if (t != x) {
                if (array[t] == 0)
                    array[t] = array[x];
                else if (array[t] == array[x]) {
                    array[t]*=2;
                    stop = t + 1;
                }
                array[x] = 0;
            }
        }
    }
}
```

目标位置：findTarget()

- 先暗中观察
  - 遍历x到stop上的每一个元素，注意是反序
  - 跳过0，找到第一个非0元素
  - 如果全是0，找到最后的一个0元素

```
int findTarget(int array[SIZE], int x, int stop)
{
    int t;
    for (t = x - 1; t >= stop; t--)
    {
        if (array[t] != 0 && array[t] != array[x])
            return t + 1;
    }
    return t + 1;
}
```

局部模块的测试

- 做个小测试
  - 举一些常见的例子，进行测试

```
int main(int argc, char *argv[]) {
    int a[][SIZE] = {
        { 0,0,0,2 },{ 0,0,2,0 },{ 0,2,0,0 },{ 2,0,0,0 },
        { 2,0,0,2 },{ 2,0,2,0 },{ 2,2,0,0 },{ 0,2,0,2 },
        { 2,0,4,2 },{ 2,4,0,2 },{ 2,0,4,4 },{ 4,4,0,2 },
        { 8,0,4,2 },{ 8,4,2,4 },{ 8,4,4,2 },{ 2,4,8,1 },
        { 0,0,0,0 },{ 2,4,2,4 },{ 2,2,4,4 },{ 4,2,2,4 } };
    for (int i = 0; i<20; i++) {
        printf("[%d]", i);
        for (int x = 0; x<SIZE; x++)
            printf("%3d ", a[i][x]);
        slideArray(a[i], SIZE);
        printf(" => ");
        for (int x = 0; x<SIZE; x++)
            printf("%3d ", a[i][x]);
        printf("\n");
    }
}
```

顺时针旋转棋盘： rotateBoard()

- 循环的条件（先观察）
  - 每次交换是4个位置：0,0; 0,1; 0,2; 1,1，交换4次
  - 循环起止？？（继续观察）
    - 外层：*i*: 0→*N*/2-1
    - 内层：*j*: *i*→*N*-*i*-2

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

3,0	2,0	1,0	0,0
3,1	2,1	1,1	0,1
3,2	2,2	1,2	0,2
3,3	2,3	1,3	0,3

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

```
void rotateBoard()
{
    int i, j, n = SIZE;
    int tmp;
    for (i = 0; i<n / 2; i++)
    {
        for (j = i; j<n - i - 1; j++)
        {
            tmp = board[i][j];
            board[i][j] = board[j][n - i - 1];
            board[j][n - i - 1] = board[n - i - 1][n - j - 1];
            board[n - i - 1][n - j - 1] = board[n - j - 1][i];
            board[n - j - 1][i] = tmp;
        }
    }
}
```

需要实现的函数

- 回顾函数列表

函数名	解决
initBoard	已完成
initBoard >> addRandom	简化替代
drawBoard	已完成
getInput	已完成
moveByDirect(char)	已完成
gameEnded	未完成
showResult	未完成

判断程序是否结束：gameEnded();

- 判断什么时候程序结束
  - 已经到达2048则结束
  - 还有空位就没结束： **hasEmpty()**
  - 如果还有成对的数字也没结束： **findPairDown()**
  - 需要判断两个方向旋转棋盘： **rotateBoard()**
    - 注意要翻转回来

```
int hasEmpty() { printf(“hasEmpty\n"); return 1; }
int findPairDown() { printf("findPairDown\n"); return 0; }
int hasWon() { printf("hasWon\n"); return 0; }

int gameEnded()
{
    if (hasWon()) return 2;
    if (!hasEmpty()) return 0;
    if (findPairDown()) return 0;
    rotateBoard();
    int isEnded = 1;
    if (findPairDown())
        isEnded = 0;
    rotateBoard();
    rotateBoard();
    rotateBoard();
    return isEnded;
}
```

判断留有空位：countEmpty()

- 判断空位数量
  - 两层循环遍历
  - 如果当前位置为空，返回1
  - 否则，返回0
- 往下判断是否有相邻成对的棋子： findPairDown()
  - 两层循环遍历（有一层因为避免溢出要调整起止）
  - 如果有相邻成对的棋子，返回1
  - 否则，返回0

```
int hasEmpty() {
    int x, y;
    for (x = 0; x<SIZE; x++)
        for (y = 0; y<SIZE; y++)
            if (board[x][y] == 0)
                return 1;
    return 0;
}

int findPairDown() {
    int x, y;
    for (x = 0; x<SIZE; x++)
        for (y = 0; y<SIZE - 1; y++)
            if (board[x][y] == board[x][y + 1])
                return 1;
    return 0;
}
```

```
int hasWon()
{
    int x, y;
    for (x = 0; x<SIZE; x++)
        for (y = 0; y<SIZE; y++)
            if (board[x][y] == WIN)
                return 1;
    return 0;
}
```

修正输入函数：getInput ()

- 初步方便连续测试，处理回车

```
char getInput()
{
    printf("Please input a command: ");
    char ch = getchar();
    while (getchar() != '\n');
    return ch;
}
```

```
char getInput()
{
    printf("Please input a command: ");
    char ch;
    while (ch = getchar())
    {
        if (ch == 'a' || ch == 's' || ch == 'd' || ch == 'w'
|| ch == '\n')
            break;
    }
    if (ch != '\n')
        while (getchar() != '\n');
    return ch;
}
```

输出结果：showResult()

- 输出GAME OVER
- 或者判断是否达到2048
  - 达到，输出YOU WIN
  - 否则，输出GAME OVER

```
void showResult(int mode)
{
    if (mode == 1)
        printf("GAME OVER\n");
    else if (mode == 2)
        printf("YOU WIN\n");
}
```

```
int main(int argc, char *argv[]) {
    initBoard();
    drawBoard();
    while (1) {
        char c = getInput();
        int mode;
        moveByDirect(c);
        addRandom();
        drawBoard();
        if (mode = gameEnded()) {
            showResult(mode);
            break;
        }
    }
    return 0;
}
```

需要实现的函数

- 回顾函数列表

函数名	解决
initBoard	已完成
initBoard >> addRandom	简化替代
drawBoard	已完成
getInput	已完成
moveByDirect(char)	已完成
gameEnded	已完成
showResult	已完成

需要实现的函数

- 回顾函数列表

函数名	解决
initBoard	已完成
initBoard >> addRandom	已完成
drawBoard	已完成
getInput	已完成
moveByDirect(char)	已完成
gameEnded	已完成
showResult	已完成

进一步改进程序

- 问题：处理用户的不当输入
  - 用户输入的时候，可能会有多余的输入
  - 错误的输入应忽略，不能增加棋子

```
int slideArray(int array[], int length) {
    int x, t, stop = 0, success = 0;
    ...
                array[x] = 0;
                success = 1;
            }
        }
    return success;
}

int moveUp() {
    int success = 0;
    ...
        if (slideArray(board[x], SIZE))
            success = 1;
    }
    return success;
}
```

```
int moveLeft()
{
    int success;
    rotateBoard();
    success = moveUp();
    ...
    return success;
}

int moveDown()
{
    int success;
    ...
    success = moveUp();
    ...
    return success;
}
```

```
int moveRight() {
    int success;
    ...
    success = moveUp();
    rotateBoard();
    return success;
}
```

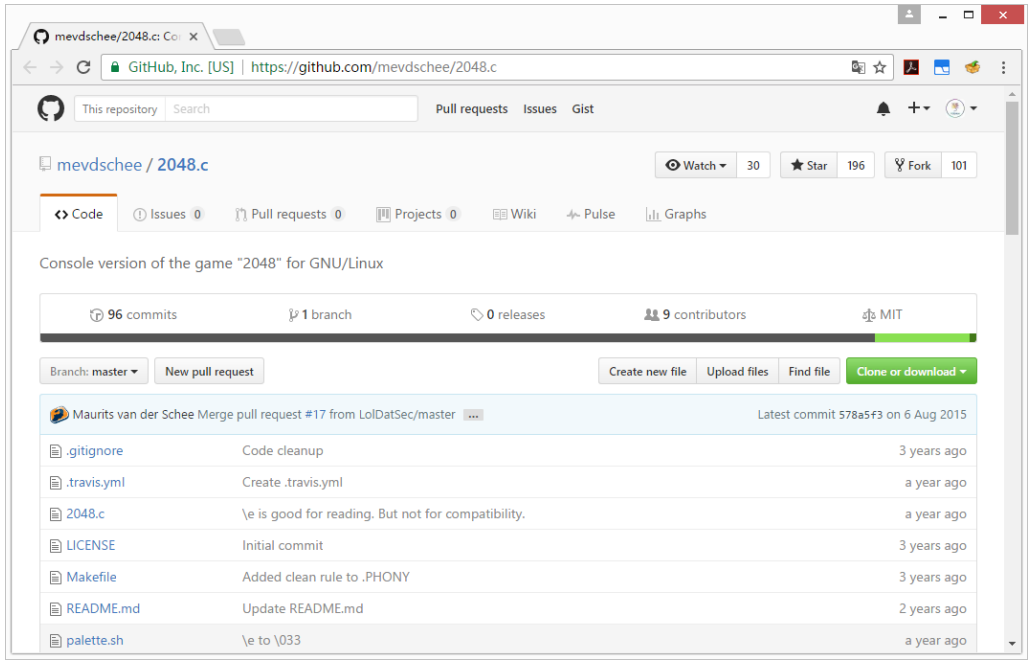
```
int moveByDirect(char c) {
    int success = 0;
    ...
        success = moveLeft();
    ...
        success = moveRight();
    ...
        success = moveUp();
    ...
        success = moveDown();
        break;
    }
    return success;
}
```

居中显示棋子：drawBoard()

```
void drawBoard()
{
    int x, y;
    printf(" ***** Game 2048 ***** \n");
    for (y = 0; y < SIZE; y++)
        for (x = 0; x < SIZE; x++)
            if (board[x][y] != 0)
                printf("[%8d]", board[x][y]);
            else
                printf("[%8s]", " ");
        printf("\n");
    printf("\n");
}
```

高级专题：开源软件

- 本课件的软件源自于
  - https://github.com/mevdschee/2048.c
- 编译环境
  - **Linux GCC**
- 开源软件还需提供
  - README
  - Makefile
  - LICENSE，.travis.yml等



[Makefile]

```
CFLAGS += -std=c99

.PHONY: all clean

all: 2048

clean:
    rm -f 2048
```

[.travis.yml]

```
language: c
compiler:
  - gcc
  - clang
# Change this to your needs
script: make
```

```
[README.md]
2048.c
======

![screenshot](http://www.leaseweblabs.com/wp-
content/uploads/2014/03/20481.png)

Console version of the game "2048" for GNU/Linux

### Gameplay

You can move the tiles in four directions using the arrow
keys: up, down, left, and right. All numbers on the board
will slide into that direction until they hit the wall and if
they bump into each other then two numbers will be combined
into one if they have the same value. Each number will only
be combined once per move. Every move a new number 2 or 4
appears. If you have a 2048 on the board you have won, but
you lose once the board is full and you cannot make a move.
```

````
### Requirements

- C compiler
- GNU/Linux

### Installation

```
wget https://raw.githubusercontent.com/mevdschee/2048.c/master/2048.c
gcc -o 2048 2048.c
./2048
```

### Running

The game supports different color schemes. This depends on
ANSI support for 88 or 256 colors. If there are not enough
colors supported the game will fallback to black and white
````

(still very much playable). For the original color scheme run:

```
./2048
```

For the black-to-white color scheme (requires 256 colors):

```
./2048 blackwhite
```

For the blue-to-red color scheme (requires 256 colors):

```
./2048 bluered
```

````
### Contributing

Contributions are very welcome. Always run the tests before
committing using:

```
$ ./2048 test
All 13 tests executed successfully
```
````

[LICENSE]
The MIT License (MIT)

C程序设计

E3.

THANK YOU

厦门大学信息学院软件工程系

黄炜 副教授