《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 软件工程 | 姓名 | 梅子羽 | 学号 | 2327406107 |
| 实验布置日期 | | 2024.9.3 | | 提交  日期 | 2024.9.3 | | 成绩 |  |

课程实践实验1：生命游戏的模拟

## 一、问题描述及要求

生命游戏在一个无边界的矩形网格上进行，这个矩形网格中的每个单元可被占据，或者不被占据。被占据的单元称为活的，不被占据的单元称为死的。

哪一个单元是活的是根据其周围活的邻居单元的数目而一代一代地发生变化的。一代一代转换的具体规则如下：

（i）给定单元的邻居单元指的是与它在垂直、水平或对角方向上相接的8个单元。

（ii）如果一个单元是活的，则如果它具有2个或3个活的邻居单元，则此单元在下一代还是活的。（iii）如果一个单元是活的，则如果它具有0个或1个、4个或4个以上的活的邻居单元，则此单元在下一代会因为孤独或拥塞而死亡。

（iv）如果一个单元是死的，则如果它具有恰好有3个活的邻居，则此单元在下一代会复活，否则该单元在下一代仍然是死的。

（1）要求编写程序，模拟任意一个初始输入配置以及代代更替的不同状态并进行显示。

（2）修改以上程序,要求生成一个网格时，用“空格”和“x”分别表示网格中每一个死的单元和活的单元，并且可根据用户选择从键盘或者从文件读入初始配置。

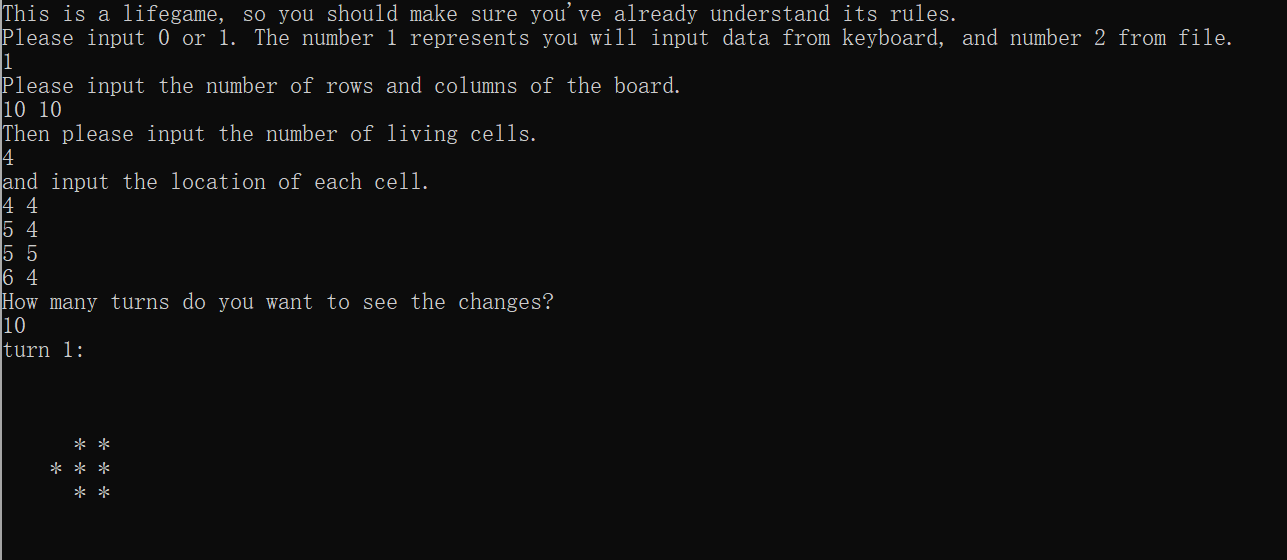
## 二、概要设计

生命游戏是很经典的一道数学问题，它最初于1970年10月在《科学美国人》杂志中马丁·葛登能的“数学游戏”专栏出现。作为一道思考性的题目，我们的目标并不是解出具体答案，而是利用计算机的特性来探索其中的规律。

根据题目不难将问题拆分成四个部分来编写对应的代码段落，一是根据用户输入确定矩形网格大小；二是初始化矩形网格的状态，这其中还要区分用户数据的键盘输入或文件导入；

三是实时更新矩形网格的状态，这又分为呈现单元矩阵的对应数组变化和相邻计数矩阵的对应数组变化；四是格式化，图形化输出单元矩形网格。

若要完整的串联起四部分，合理的语言引导是必不可少的。故笔者也在控制台的程序界面上呈现了必要的文字说明。



综上，总体的设计思路已然呈现。秉持功能化，模块化编写代码的原则，我将定义Configuration类来处理此问题，将所述的四个部分内容的声明和编写分别以.h和.cpp文件形式呈现，最后在main函数中实现对用户的交互。其中涉及到的主要数据结构即为二维向量，剩余皆为基础代码结构。

## 三、详细设计

这里着重分析一下怎么实现单元的代际更迭的。

为了每轮的实时结果呈现，必须有一个二维向量记录单元的演变情况，而单元怎么演变，则还需要一个计数的二维向量，每个格子中的数表示与此格相邻的活单元数量。通过遍历记录单元的二维向量，对存在活单元的网格，相邻的8个网格对应的相邻计数二维向量中的数都加1，最后根据相邻计数的二维向量和游戏规则，得到下一回初始的记录单元的二维向量。

但要注意，每次相邻计数的二维向量都必须初始化为0。

针对边缘上的活单元，笔者考虑到遍历越界的情况，特别将两个二维向量都开大了一圈，避免在处理九宫格问题时的越界报错问题。外侧的数据并不需要过多在意，但最好还是同内侧同样操作，数据呈现是不涵盖在内即可。

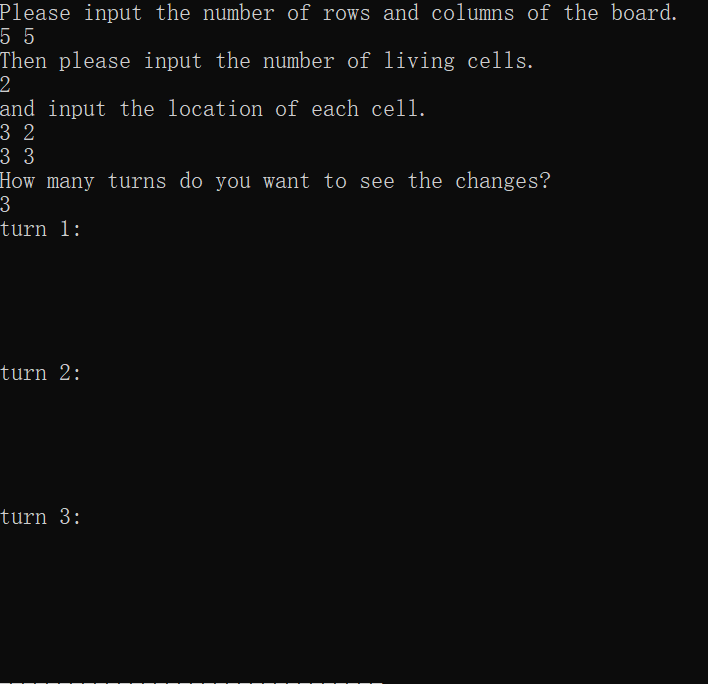
## 四、实验结果

笔者选取了四组典型的输入输出。

第一组：键盘输入 5\*5网格 2个活细胞 位于（3,2）（3,3）查看2轮情况

针对输入交互，主函数逻辑

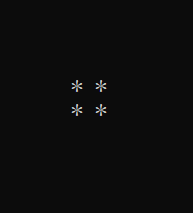
正确输出：均为

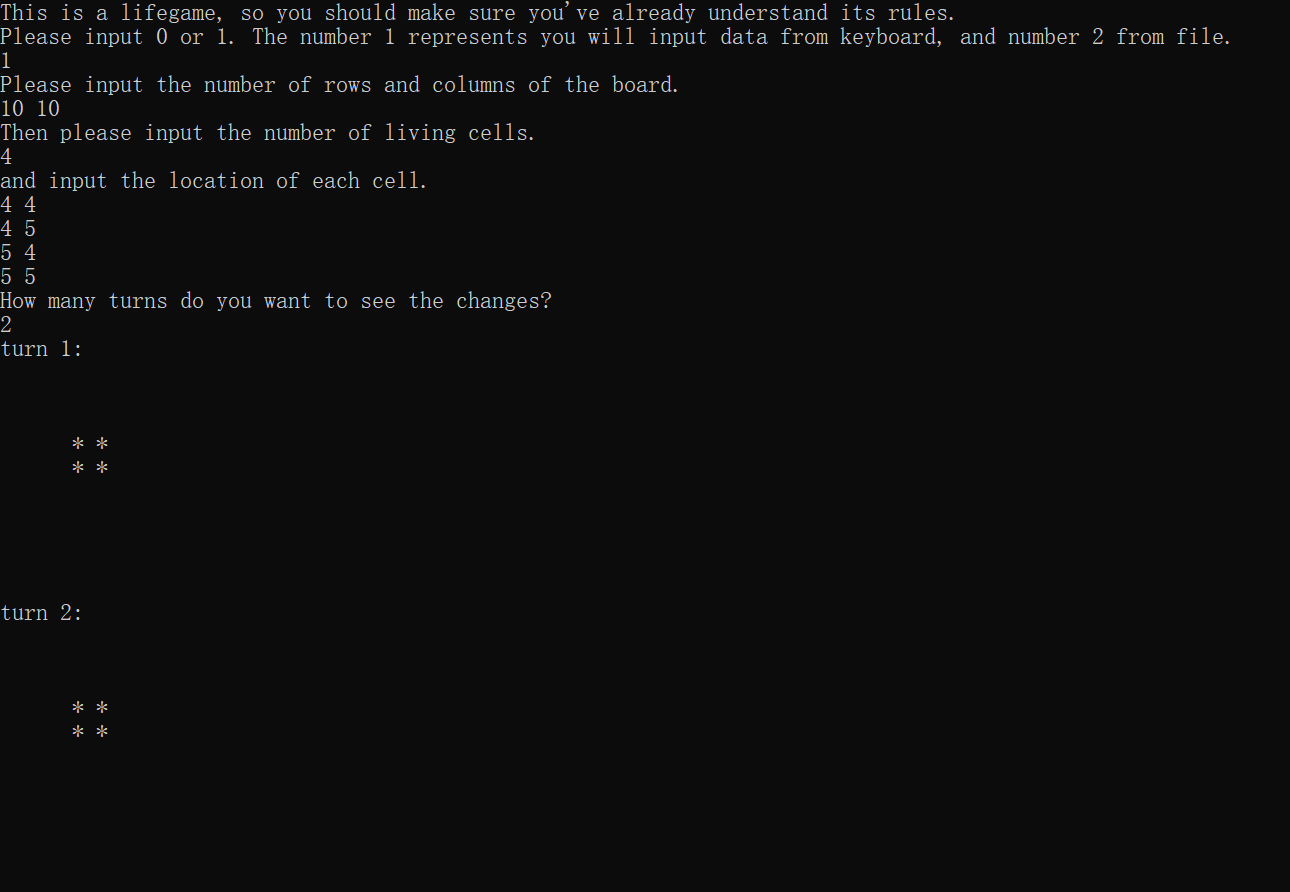
实际输出：

测试结论：通过

第二组：键盘输入 10\*10网格 4个活细胞 位于（4,4）（4,5）（5,4）（5,5）查看2轮情况

针对输入交互，主函数逻辑

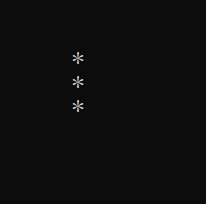
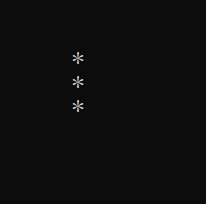
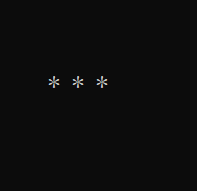
正确输出：均为

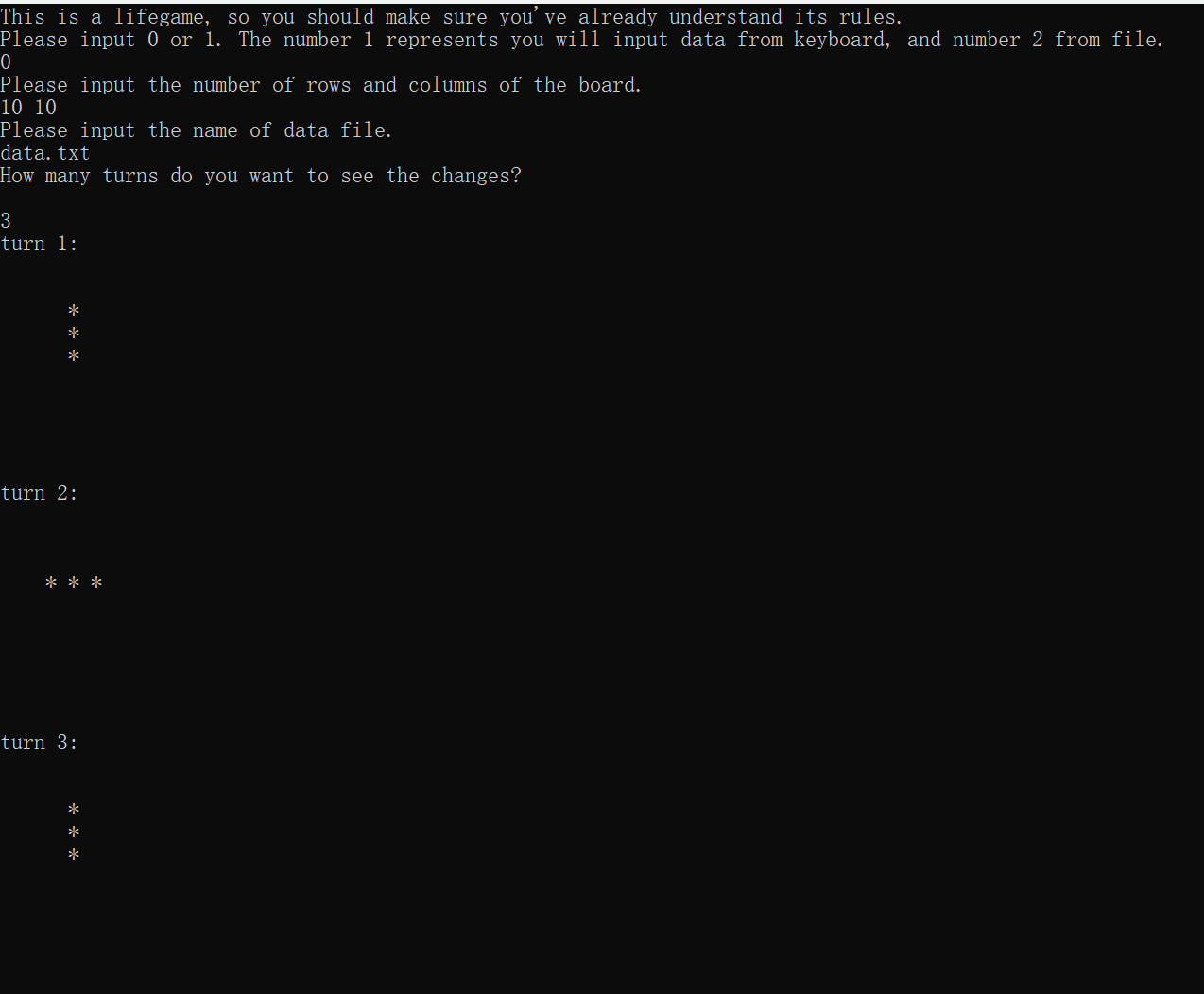
实际输出：

测试结论：通过

第三组：文件输入 10\*10网格 3个活细胞 位于（4,3）（4,4）（4,5）查看3轮情况

针对输入交互，主函数逻辑

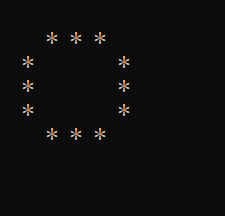
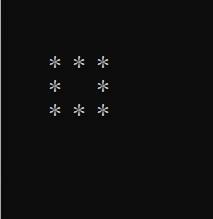
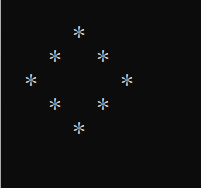
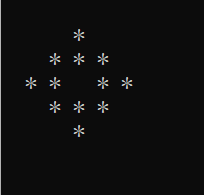
 正确输出： 

实际输出： 

测试结论：通过

第四组：键盘输入 10\*10网格 5个活细胞 位于（4,3）（3,4）（4,4）（5,4）（4,5）查看4轮情况

/ 针对输入交互，主函数逻辑

 正确输出：   

实际输出： 

测试结论：通过

## 五、实验分析与探讨

测试主要针对输入的键盘与文件交互和主函数的逻辑检验，因为本次实验的算法性较弱，性能上的分析就从简，主要就是双重循环的n方时间复杂度。其实这个实验的样例很难覆盖所有情况，因为单元的演化情况多样，但最终都会趋于稳定，不论是数量为零还是动态平衡。每组数据都会有一些独特的，难以解释的规律，最可惜的是我并没有找到可以无限繁殖下去的初始单元数据，但这个情况也是相对的，因为没有规定矩阵大小和初始活单元数量，姑且算做一个小未知吧。

本次实验主要练习了二维向量的代码编写，设计到其定义，遍历，更改数据等基础操作，笔者确实没有想到其他更加简便直观的解决办法了，但在此次实验中出现的代码调试等诸多问题，在提醒着笔者，需要对二维向量的操作有更深入的了解，比如字符向量和字符串向量之间的区别联系问题，因为笔者老是在c++的字符与字符串上出现问题。道阻且长，笔者仍需努力！

补充说明：输入文件必须存于项目文件夹下，格式为每排x y，x代表行，y代表列，中间用空格隔开。

## 附录：源代码

1.实验环境：Dev-C++ 5.11集成开发环境，C++11标准。

2、

（1）configuration.h

//configuration.h

#ifndef \_\_CONFIGURATION\_H

#define \_\_CONFIGURATION\_H

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

class Configuration

{

private:

vector<vector<char> > board;

vector<vector<int> > count;

public:

void iniBoard(int, int);

void iniFromFile(string&);

void iniFromKeyboard(int, int);

void boardUpdate();

void boardPrint()const;

};

#endif

（2）configuration.cpp

//configuration.cpp

#include "configuration.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

vector<vector<int> > count1;

void Configuration::iniBoard(int x, int y)

{

vector<vector<char> > board1(x + 2, vector<char>(y + 2, ' '));

vector<vector<int> > temp(x + 2, vector<int>(y + 2, 0));

count = temp;

board = board1;

count1 = count;

}

void Configuration::iniFromKeyboard(int x, int y)

{

board[x][y] = '\*';

}

void Configuration::iniFromFile(string &filename)

{

ifstream ifs;

ifs.open(filename, ios::in);

char ct[5];

while( ifs.getline(ct, 5) )

{

int x = int(ct[0]) - 48;

int y = int(ct[2]) - 48;

board[x][y] = '\*';

}

ifs.close();

}

void Configuration::boardUpdate()

{

for(int i = 1; i <= board.size() - 2; i ++)

{

for(int j = 1; j <= board[0].size() - 2; j ++)

{

if(board[i][j] == '\*')

{

count[i - 1][j - 1] ++;

count[i - 1][j] ++;

count[i - 1][j + 1] ++;

count[i][j - 1] ++;

count[i][j + 1] ++;

count[i + 1][j - 1] ++;

count[i + 1][j] ++;

count[i + 1][j + 1] ++;

}

}

}

for(int i = 1; i <= board.size() - 2; i ++)

{

for(int j = 1; j <= board[0].size() - 2; j ++)

{

if(board[i][j] == '\*')

{

if(count[i][j] != 2 && count[i][j] != 3)

board[i][j] = ' ';

}

else

{

if(count[i][j] == 3)

board[i][j] = '\*';

}

}

}

count = count1;

}

void Configuration::boardPrint() const

{

for(int i = 1; i <= board.size() - 2; i ++)

{

for(int j = 1; j <= board[0].size() - 2; j ++)

{

cout << board[i][j] << ' ';

}

cout << endl;

}

}

（3）main.cpp

//main.cpp

#include "configuration.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

int main()

{

bool flag; string filename;

int x, y, number, x1, y1, turns;

Configuration c1;

cout << "This is a lifegame, so you should make sure you've already understand its rules.\n";

cout << "Please input 0 or 1. The number 1 represents you will input data from keyboard, and number 2 from file.\n";

cin >> flag;

cout << "Please input the number of rows and columns of the board.\n";

cin >> x >> y;

c1.iniBoard(x, y);

if(flag == 1)

{

cout << "Then please input the number of living cells.\n";

cin >> number;

cout << "and input the location of each cell.\n";

while(number --)

{

cin >> x1 >> y1;

c1.iniFromKeyboard(x1, y1);

}

}

else

{

cout << "Please input the name of data file.\n";

cin >> filename;

c1.iniFromFile(filename);

}

cout << "How many turns do you want to see the changes?\n";

cin >> turns;

for(int i = 1; i <= turns; i ++)

{

cout << "turn " << i << ":\n";

c1.boardUpdate();

c1.boardPrint();

}

return 0;

}