《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 大二软件工程 | 姓名 | 丁彦君 | 学号 | 2329403126 |
| 实验布置日期 | | 2024.9.3 | | 提交  日期 | 2024.9.11 | | 成绩 |  |

课程实践实验1：生命游戏的模拟

## 问题描述及要求

模拟生命游戏在一个无边界矩形网格上的演化,每个单元格有两个状态：活的（x）或死的（□）。状态根据邻居的存活单元数目更新：活的单元需有2或3个活邻居才存活，死的单元需有恰好3个活邻居才能复活。要求程序可以逐代显示单元格状态变化。

## 二、概要设计

**（1）对实验内容的理解**：

该程序根据特定的生命规则（基于周围8个邻居的存活状态）来更新每个单元的状态。程序需要支持显示每一代的状态变化，并且能够根据用户的选择，从键盘输入或文件读取初始的网格配置（用“□”表示死单元，“\*”表示活单元）。通过模拟这些状态变化，用户可以观察到生命游戏随时间的动态演化过程。

**（2）功能**：

使用二维数组来模拟各个细胞的状态。

**（3）程序运行的界面设计**：

先介绍游戏玩法和规则，然后通过相应输入输出对应的结果，并在屏幕下方提示是否继续进行更迭，不进行更迭就结束运行，进行更迭就展示下一阶段的状态，并再次询问是否再次更迭，循环往复。在实验规则的基础上我将死亡单元由原来的空格换成□，使得状态更加显目，视觉体验感更强。

下面是初代的操作步骤：

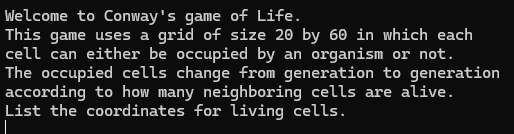


图1：屏幕提示

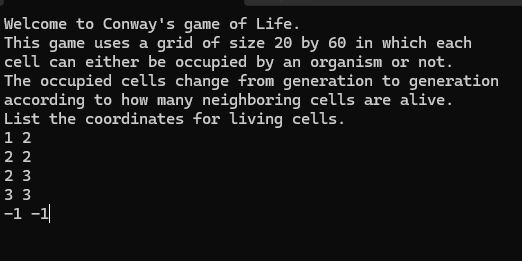


图2：键盘输入

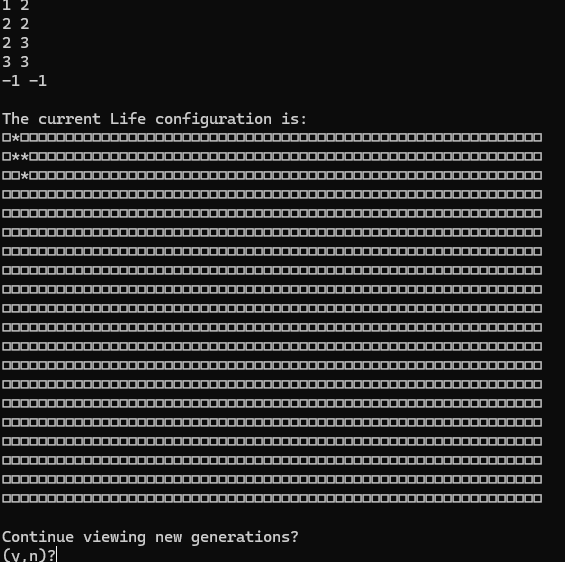


图3：运行结果



图4：用户选择是否继续运行



图5- 1继续运行迭代的结果

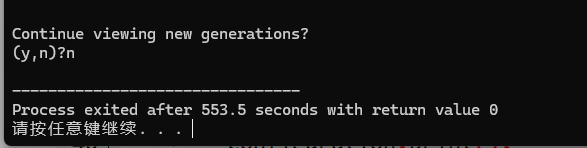
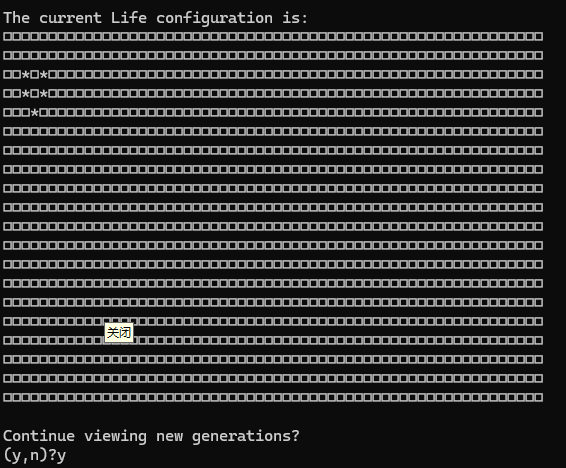
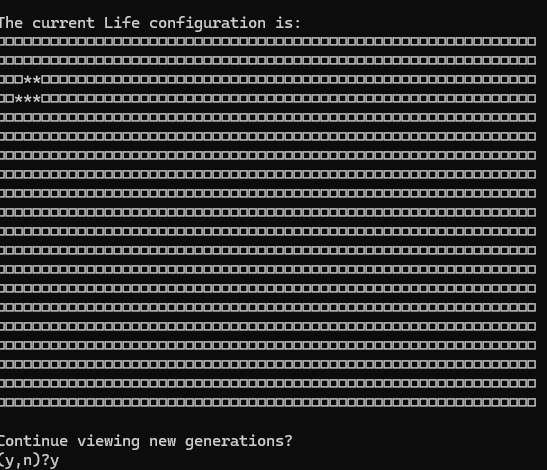
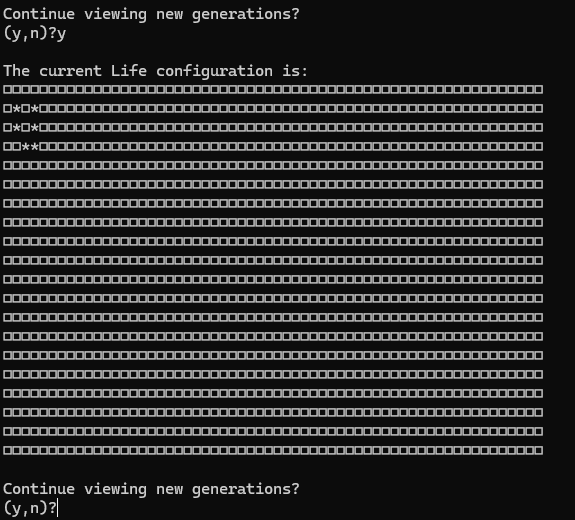
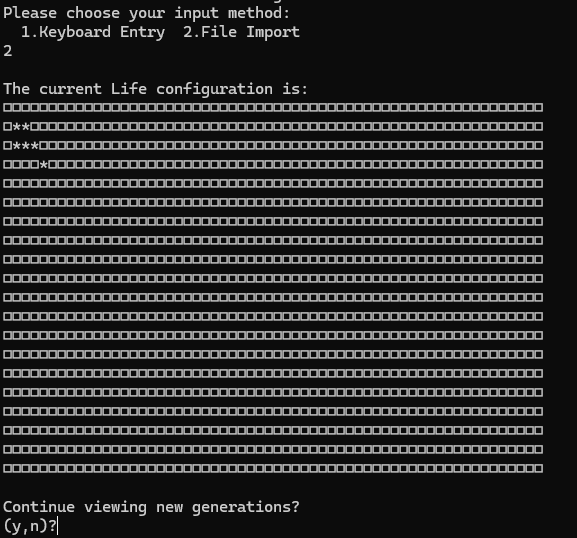
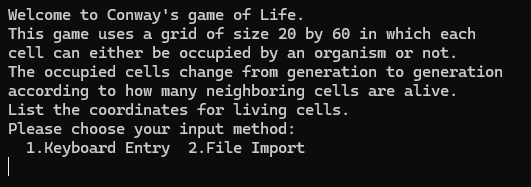


图5- 2结束运行的结果

接下来是进阶版的操作步骤（仅展示不同之处）：



最后全部消亡。

**（4）设计思路**：

1.明确游戏网格规模。

2.确定游戏输入输出规则并显示在屏幕上。

3.根据输入的内容初始化单元的状态，并将其打印到屏幕上。

4.让玩家选择是否进行下一步迭代操作。

5.如果不迭代，则结束游戏进程；若迭代，则输出下一次的单元状态，以此类推。

6.设置边界条件，简化算法，提升游戏可操作性和可读性。

7.结合上述3,4,5规定一个类用来存放函数。

**（5）程序结构设计**：

1.首先需要创建一个项目用于存放源文件和代码。

2.然后创建一个主函数用于整体操作。

3.接着创建类（类名：life）：在life.h中引入类的定义，在life.cpp中添加各种函数的具体内容。

4.在主程序中按照设计思路的顺序编写函数。

## 三、详细设计

**（1）各类函数介绍：**

**instructions(); 描述游戏规则**

**initialize(); 初始化存活单元的状态**

**print(); 打印输出当前单元的状态 （存活/死亡）**

**neighbor\_count(); 统计周围活的邻居单元的数目**

**update(); 进行存活/死亡单元间的转换**

**user\_says\_yes(); 供用户选择是否进行下一步迭代**

**（2）关键算法的设计：**

输入

初始化数组

开始

**2.文件**

**1.键盘**

输出

迭代

y/n ?

**y**

**n**

结束运行

## 四、实验结果

**测试输入（图1）：共计三组数据(22/23/24 55/56 87 88 97 98）（-1-1代表结束输入）**

**测试目的：通过不同的网格排布研究其迭代规律。**

**输出结果：（图2,3,4,5）结果显示，第一种呈交替往复，第二种是直接消亡，第三种是稳定不变。**

**错误原因：在代码编译时遇到（Id returned 1 exit status）的问题，经过长期摸索发现是编译时32位和64位不兼容导致的。**

**测试结论：通过（见下图演示）**

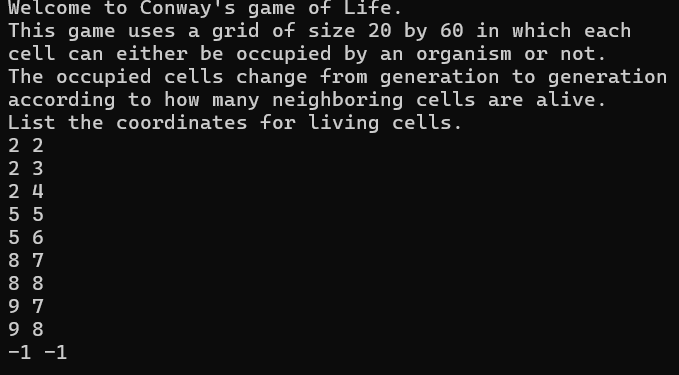
****

图1

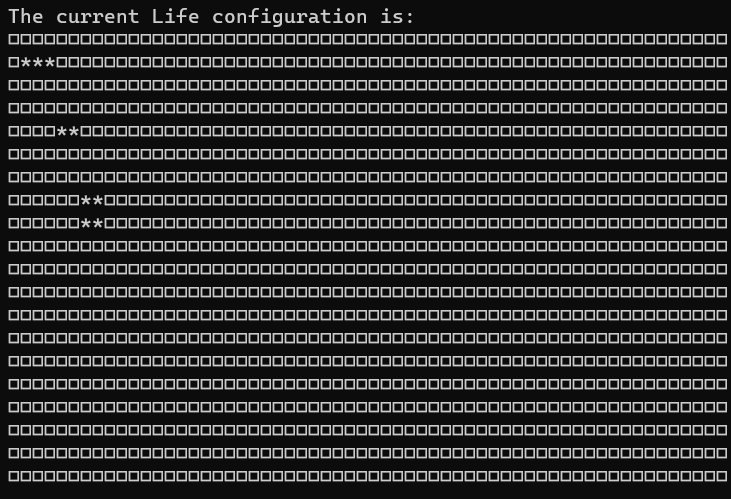
****

图2

****

图3

****

图4

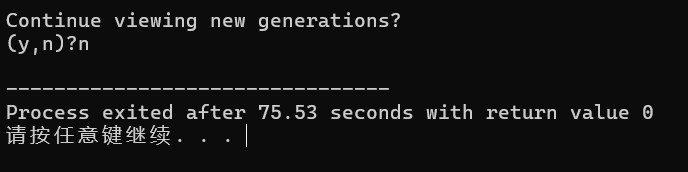
****

图5

## 五、实验分析与探讨

**1.测试结果的分析**

**(1)测试策略解释**

**测试方法**：

使用多种测试用例（见上述图例）

**测试目标**：

确保模拟能够正确地按照生命游戏的规则运行，验证算法在各种情况下的表现

**(2)结果分析**

**结果概述**：

总结测试中得到的结果，描述不同配置下模拟的表现。比如是否存在某些状态（稳定不变/交替/逐渐消亡）

**时空复杂度分析**：

**时间复杂度**：

与网格的大小以及迭代次数有关。例如，若每次迭代需要遍历整个网格，则时间复杂度为 O(N×M×T);里面还有多重循环操作，每层循环叠加O(N).

**空间复杂度**：

空间复杂度为 O(N×M)，因为需要存储整个网格的状态。

**2.实验设计和实现过程中的问题**

**(1)实验设计**

**问题描述**：

刚开始不会合理运用类和项目，代码都堆叠在一起，经常把自己写绕进去。

**解决方法**：

学会创建项目，使用类，在类.h中定义类，在类.cpp中添加函数，使条理清晰。

**(2)实现过程中遇到的问题**

**问题描述**：

q1.开始时项目内类名用的Life,运行时报错。

q2.编写代码时经常不小心将；和：混用。

q3.将python的一些代码逻辑误用在c++里，比如if后面不用()而是直接用空格代替，一行代码写完忘记加；等等。

q4.代码二次编译时与其他程序冲突，出现Id returned 1 exit status这种错误，编译不通过。

**解决方法**：

q1.没查出来问题，后来还是重新写换成小写的life才成功的。

q2,q3.这两种都是粗心导致的，一次次的编译，找出错误的地方进行修改。

q4.这个问题比较棘手，代码一点问题没有，但是就是编译不通过，我的是项目1，但是编译出来不知道从哪冒出来一个collect2.exe，导致我直接用项目里面的 项目.exe 可以运行，但是改变代码不起作用，最后还是重新创建项目转移代码才成功的，然后在csdn上找到可能问题出在之前运行的未关闭隐藏起来，另一个说法是3位和64位存在冲突。

## 六、小结

**1.已完成内容：**

1. 实现了生命游戏的基本规则，包括细胞的生死状态转换。
2. 设计了界面用于显示细胞网格和模拟结果。
3. 编写了更新细胞状态的算法，并确保其符合生命游戏的规则。
4. 优化了算法和代码，增加了一些输入的框架和限制条件，使得程序更加人性化，避免了不必要的问题。
5. 使输入方式更加多元化，既允许用户键盘输入，也允许用户导入文件（ans.txt）。

**2.未完成内容：**

1. 性能未完全优化，特别是在大规模网格下的效率提升。
2. 界面交互功能，比如动态调整网格大小未能实现。

**3.程序局限性：**

大规模网格可能导致性能问题，处理速度较慢。

**4.展望与建议：**

**(1)进一步学习与完善方向：**

探索不同的优化算法，进一步简化代码，提高运行效率。

**(2)感想与收获：**

刚开始面对这个题目时我是一头雾水的，尽管懂得项目实现的大致步骤和方向，但不知从何处下手。但是通过一周多的潜心研究，从学会创建项目和类开始，到构思框架，实现函数，到后期优化代码，并添加文件导入这一多元化的功能，一路以来收获颇多。我想，正是在这一次次的试错和改进的过程中，我才能提高自己的代码能力和处理、解决问题的能力。

不怕苦，肯埋下头来好好敲代码，不被冗繁的bug所击败，是我这次实践的最大收获。在今后的课程实践中，我也将继续挑战自己，追求卓越，朝着更好的方向努力下去。

## 附录：源代码

1、**实验环境：Dev-cpp**

2、

（1）main.cpp

#include "life.h"

#include <iostream>

using namespace std;

bool user\_says\_yes()

{

int c;

bool initial\_response = true;

do {

if (initial\_response)

cout << "(y,n)?" << flush;

else

cout << "Respond with either y or n:" << flush;

do {

c = cin.get();

}while(c == '\n' || c == ' ' || c == '\t');

initial\_response = false;

}while (c != 'y' && c != 'Y' && c != 'n' && c != 'N');

return (c == 'y' || c == 'Y');

}

void instructions( )

{

cout << "Welcome to Conway's game of Life." << endl;

cout << "This game uses a grid of size "

<< maxrow << " by " << maxcol << " in which each" << endl;

cout << "cell can either be occupied by an organism or not." << endl;

cout << "The occupied cells change from generation to generation"<< endl;

cout << "according to how many neighboring cells are alive."<< endl;

}

int main()

{

life configuration;

instructions();

configuration.initialize();

configuration.print();

cout << "Continue viewing new generations?" << endl;

while (user\_says\_yes())

{

configuration.update();

configuration.print();

cout << "Continue viewing new generations?" << endl;

}

}

（2）life.h

#ifndef LIFE\_H

#define LIFE\_H

const int maxrow = 20,maxcol = 60;

class life

{

public:

void initialize();

void print();

void update();

private:

int grid[maxrow + 2][maxcol + 2];

int neighbor\_count(int row,int col);

};

#endif

（3）life.cpp

#include "life.h"

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

int life::neighbor\_count(int row,int col)

{

int i,j,count = 0;

for(i = row - 1;i <= row + 1;i++)

{

for(j = col - 1;j <= col + 1;j++)

count += grid[i][j];

}

count -= grid[row][col];

return count;

}

void life::initialize()

{

int row,col;

int n;

for(row = 0;row <= maxrow + 1;row++)

{

for(col = 0;col <= maxcol + 1;col++)

grid[row][col] = 0;

}

cout << "List the coordinates for living cells." << endl;

cout << "Please choose your input method:" << endl;

cout << " 1.Keyboard Entry ";

cout << "2.File Import" << endl;

cin >> n;

if(n == 1)

{

cin >> row >> col;

while(row != -1 || col != -1)

{

if(row == 0 || row > 20)

{

cout << "Row " << row << " is out of range" << endl;

cout << "Please enter the coordinates again." << endl;

cin >> row >> col;

}

if(row >= 1 && row <= maxrow)

{

if(col >= 1 && col <= maxcol)

grid[row][col] = 1;

else

cout << "Column " << col << " is out of range" << endl;

cout << "Please enter the coordinates again." << endl;

cin >> row >> col;

}

}

}

else

{

std::ifstream file("ans.txt");

if (!file)

{

std::cerr << "Unable to open file\n";

return;

}

int row, col;

while (file >> row >> col)

{

if (row == -1 && col == -1)

{

break;

}

if (row < 1 || row > maxrow)

{

std::cout << "Row " << row << " is out of range" << std::endl;

continue;

}

if (col < 1 || col > maxcol)

{

std::cout << "Column " << col << " is out of range" << std::endl;

continue;

}

grid[row][col] = 1;

}

file.close();

}

}

void life::print()

{

int row,col;

cout << "\nThe current Life configuration is:" << endl;

for(row = 1;row <= maxrow;row++)

{

for(col = 1;col <= maxcol;col++)

{

if(grid[row][col] == 1)

cout << "\*";

else

cout << "□";

}

cout << endl;

}

cout << endl;

}

void life::update()

{

int row,col,new\_grid[maxrow + 2][maxcol + 2];

for(row = 1;row <= maxrow;row++)

for(col = 1;col <= maxcol;col++)

switch(neighbor\_count(row,col))

{

case 2:

new\_grid[row][col] = grid[row][col];

break;

case 3:

new\_grid[row][col] = 1;

break;

default:

new\_grid[row][col] = 0;

}

for(row = 1;row <= maxrow;row++)

for(col = 1; col <= maxcol;col++)

grid[row][col] = new\_grid[row][col];

}