# Langton’s Ant

软件1班 202430551743 严嘉鹏

1. 实验目的

调整了开题报告中的模块分类和工作计划，初步敲定了游戏模式：自由模式，闯关模式，趣味模式，对抗模式，合作模式，以及最后尝试利用Qt来实现GUI。本次报告以完成自由模式为主。

1. 基本游戏规则（参考最强大脑）

正方形棋盘，含黑白格子和一只蚂蚁，若蚂蚁在黑格上，则将黑变白，左转90度并前进一格;若蚂蚁在白格上，则将白变黑，右转90度并前进一格。题面是一个蚂蚁已走过若干步的蚁行图，要求根据最后蚂蚁的位置和朝向来逆推出棋盘的初始状态。

三、自由模式

自由模式的主要设置目的是为了帮助用户熟悉游戏规则以便进行接下来的游戏，它的功能：用户输入棋盘的大小，然后电脑随机生成一个棋盘，用户再输入蚂蚁要走的步数、初始位置和初始朝向，然后程序检验有效后给出蚁行图和蚂蚁最后的位置和朝向供用户作答，用户做对游戏结束，做错后输入1再做一遍，输入2查看答案，如果用户连续输入3次无效的蚂蚁初始位置和朝向，那么程序会自动提示一个合理的位置和朝向，用户须按照该提示填入后作答，如果电脑也没找到合适的蚂蚁初始位置和朝向，直接输出该棋盘无解，重新生成一个棋盘供用户作答。

1. 实验内容

分为3个文件写该模块，分别为freedomMode.h，freedomMode.cpp，freedomModemain.cpp，实验内容将阐述写代码过程时的思考和相关代码。

1. freedomMode.h:

这一部分代码是最先搭框架时要写的，首先明确有哪些类和对象，以及需要用到哪些功能函数，以代码和注释形式呈现。

#ifndef FREEDOM\_MODE\_H

#define FREEDOM\_MODE\_H**//头文件保护**

#include <iostream>

using namespace std;

enum Direction {

UP,

DOWN,

LEFT,

RIGHT

};

void TurnRight(Direction& heading);

void TurnLeft(Direction& heading);

class freedomMode {**//自由模式设置成一个类**

private:

int size;**//棋盘边长**

int steps;

bool\* board;**//打算用一维数组表示二维棋盘,动态数组用于存储不同大小的棋盘**

bool\* finalBoard;

bool\* initialBoard;

void initializeBoard(bool\* board);

void copyBoard(bool\* src, bool\* dest);**//复制棋盘，用来模拟更新蚂蚁走过棋盘时不影响到原棋盘**

bool moveAnt(int& x, int& y, Direction& heading);**//将一维数组索引和二维坐标x,y相互转化,比较坐标和边长易得蚂蚁是否跑出了该数组**

int getIndex(int x, int y);**//将二维坐标转化为一维数组索引**

public:

freedomMode(int size, int steps);**//随机生成棋盘**

~freedomMode();

bool isValidInitialPosition(int x, int y, Direction heading);**//判断该位置是否越界**

void generateFinalBoard(int& x, int& y, Direction& heading);**//生成棋盘**

void printFinalBoard();//打印蚁行图

bool checkSolution(bool\* userBoard);**//检查用户是否作答正确**

void printInitialBoard();**//打印初始状态棋盘**

bool suggestInitialPosition(int& x, int& y, Direction& heading);**//用户多次输入无效时电脑给提示**

};

#endif

1. freedomMode.cpp：

有了头文件中打好的框架后，在这个cpp文件里要做的就是写好函数的定义来实现相应的功能，由于功能较多，这里只写出部分函数及其代码：

复制函数，避免对该数组操作后无法还原（模拟蚂蚁走过步数后的棋盘），传入两个一维动态数组，赋值即可，这里也可以直接使用复制构造函数来实现。

void freedomMode::copyBoard(bool\* src, bool\* dest) {

for (int i = 0; i < size \* size; ++i) {

dest[i] = src[i];

}

}

模拟函数，用于模拟蚂蚁移动过程中位置的变化和对格子颜色的改变，还用来判断前进一格后数组是否越界。

bool freedomMode::moveAnt(int& x, int& y, Direction& heading) {

int index = getIndex(x, y);

if (board[index]) {

TurnLeft(heading);

board[index] = false;

}

else {

TurnRight(heading);

board[index] = true;

}

switch (heading) {

case UP:

if (x - 1 < 0) return false;

x--;

break;

case DOWN:

if (x + 1 >= size) return false;

x++;

break;

case LEFT:

if (y - 1 < 0) return false;

y--;

break;

case RIGHT:

if (y + 1 >= size) return false;

y++;

break;

}

return true;

}

位置判断函数：如果moveAnt函数判断出前进一格后位置未越界，蚂蚁就继续走完剩下的步数，如果一直都没越界说明用户输入有效，接下来可以生成蚁行图，反之输出无效要提示用户重新输入。

bool freedomMode::isValidInitialPosition(int x, int y, Direction heading) {

copyBoard(board, finalBoard);

for (int i = 0; i < steps; ++i) {

if (!moveAnt(x, y, heading)) {

copyBoard(finalBoard, board);

return false;

}

}

copyBoard(finalBoard, board);

return true;

}

蚁行图生成函数：如果用户输入有效，就用这个函数来输出蚁行图。

void freedomMode::generateFinalBoard(int& x, int& y, Direction& heading) {

copyBoard(initialBoard, board);

for (int i = 0; i < steps; ++i) {

moveAnt(x, y, heading);

}

copyBoard(board, finalBoard);

}

建议函数：若用户连续3次输入无效，电脑会自己找出合适的蚂蚁初始位置和朝向，使用srand(),rand()函数模拟随机，并设着最大尝试次数，如果超过最大尝试次数仍未找到有效输入，则直接认为该棋盘在用户输入的步数下，无论哪种蚂蚁的初始位置和朝向都无解。

bool freedomMode::suggestInitialPosition(int& x, int& y, Direction& heading) {

const int MAX\_ATTEMPTS = 1000; // 最大尝试次数

for (int attempt = 0; attempt < MAX\_ATTEMPTS; ++attempt) {

x = rand() % size;

y = rand() % size;

heading = static\_cast<Direction>(rand() % 4);

if (isValidInitialPosition(x, y, heading)) {

return true;

}

}

return false;

}

（3）freedomModemain.cpp：这个函数主要用于人机交互，本来想利用Qt来生成可视化界面来方便用户作答，可是在VS上部署Qt的过程遇到了很多困难，自身对Qt也缺乏足够的了解，所以只能从长计议，先按照自由模式要实现的功能在黑框框里进行你问我答的简单人机交互。采用了很多if else嵌套和while的嵌套来实现。

int main() {

int size=0;

int steps=0;

while (true) {

cout << "请输入棋盘的边长: ";

cin >> size;

cout << "电脑已生成本次棋盘" << endl;

cout << "请输入要走的步数: ";

cin >> steps;

freedomMode game(size, steps);

int x=0, y=0;

Direction heading;

int invalidAttempts = 0;

while (true) {

if (invalidAttempts >= 3) {

if (!game.suggestInitialPosition(x, y, heading)) {

cout << "该棋盘无解，电脑将重新要求您输入棋盘边长生成新的棋盘。" << endl;

break;

}

cout << "电脑给出的合理初始位置和朝向如下：" << endl;

cout << "初始位置: (" << x << ", " << y << ")" << endl;

cout << "初始朝向: ";

switch (heading) {

case UP:

cout << "UP";

break;

case DOWN:

cout << "DOWN";

break;

case LEFT:

cout << "LEFT";

break;

case RIGHT:

cout << "RIGHT";

break;

}

cout << endl;

cout << "请使用电脑给出的提示重新输入蚂蚁的初始位置 (x y) 和朝向 (0: UP, 1: DOWN, 2: LEFT, 3: RIGHT): ";

cin >> x >> y;

int headingInput=0;

cin >> headingInput;

heading = static\_cast<Direction>(headingInput);

}

else {

cout << "请输入蚂蚁的初始位置 (x y): ";

cin >> x >> y;

cout << "请输入蚂蚁的初始朝向 (0: UP, 1: DOWN, 2: LEFT, 3: RIGHT): ";

int headingInput;

cin >> headingInput;

heading = static\_cast<Direction>(headingInput);

}

if (game.isValidInitialPosition(x, y, heading)) {

cout << "您的输入有效，蚁行图如下：" << endl;

int finalX = x, finalY = y;

Direction finalHeading = heading;

game.generateFinalBoard(finalX, finalY, finalHeading);

game.printFinalBoard();

cout << "蚂蚁走完所有步数的最终位置: (" << finalX << ", " << finalY << ")" << endl;

cout << "蚂蚁走完所有步数的最终朝向: ";

switch (finalHeading) {

case UP:

cout << "UP";

break;

case DOWN:

cout << "DOWN";

break;

case LEFT:

cout << "LEFT";

break;

case RIGHT:

cout << "RIGHT";

break;

}

cout << endl;

bool\* userBoard = new bool[size \* size];

while (true) {

cout << "请输入原始棋盘 (0 表示白格, 1 表示黑格):" << endl;

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

cin >> userBoard[i \* size + j];

}

}

if (game.checkSolution(userBoard)) {

cout << "回答正确！游戏结束" << endl;

delete[] userBoard;

return 0;

}

else {

cout << "回答错误！" << endl;

int choice;

cout << "输入 1 重新作答该题，输入 2 电脑直接给出初始棋盘答案: ";

cin >> choice;

if (choice == 1) {

continue;

}

else if (choice == 2) {

game.printInitialBoard();

cout << "游戏结束" << endl;

delete[] userBoard;

return 0;

}

}

}

}

else {

cout << "您的输入无效，请重新输入蚂蚁的初始位置和朝向" << endl;

invalidAttempts++;

}

}

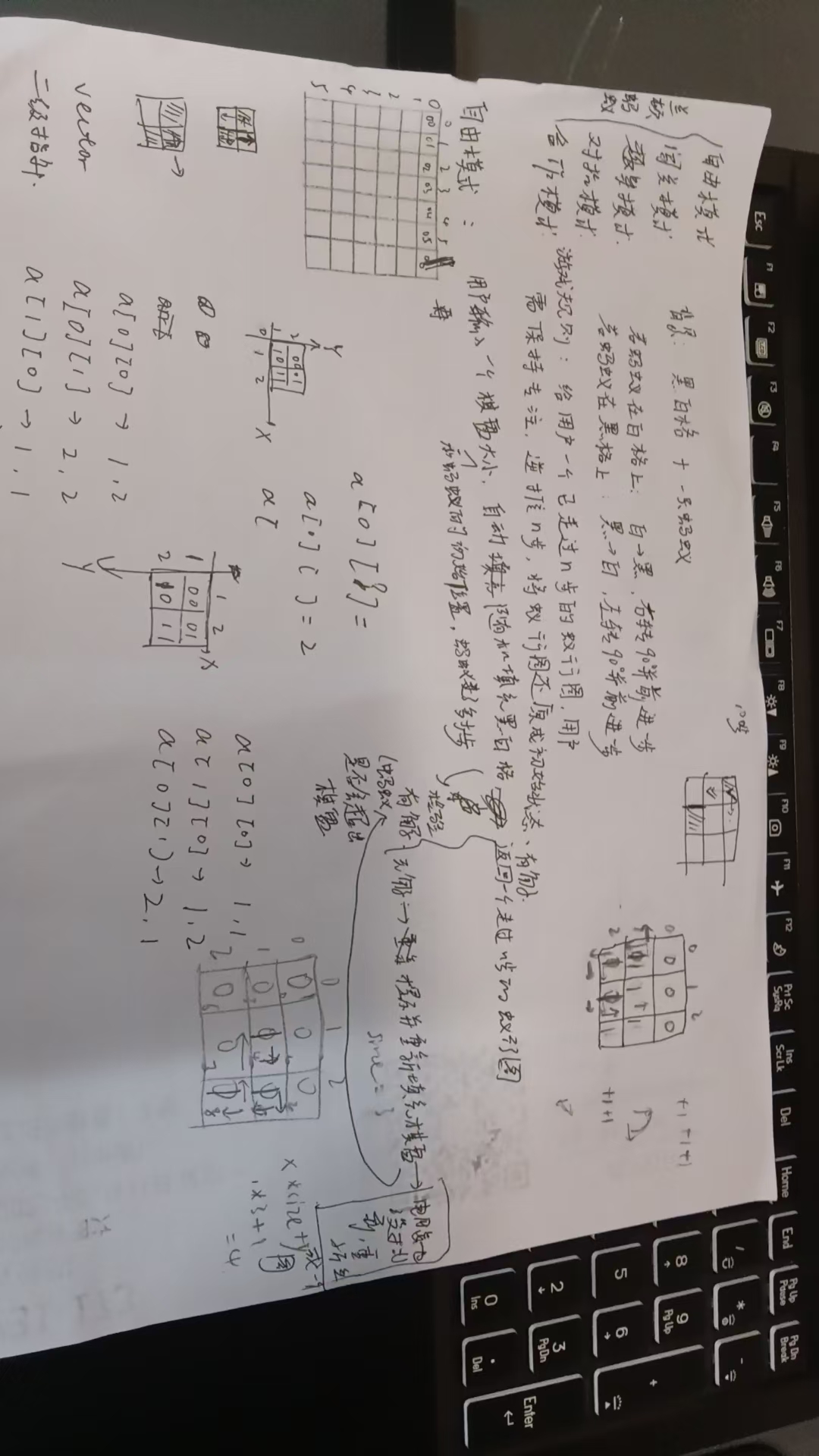
}

return 0;

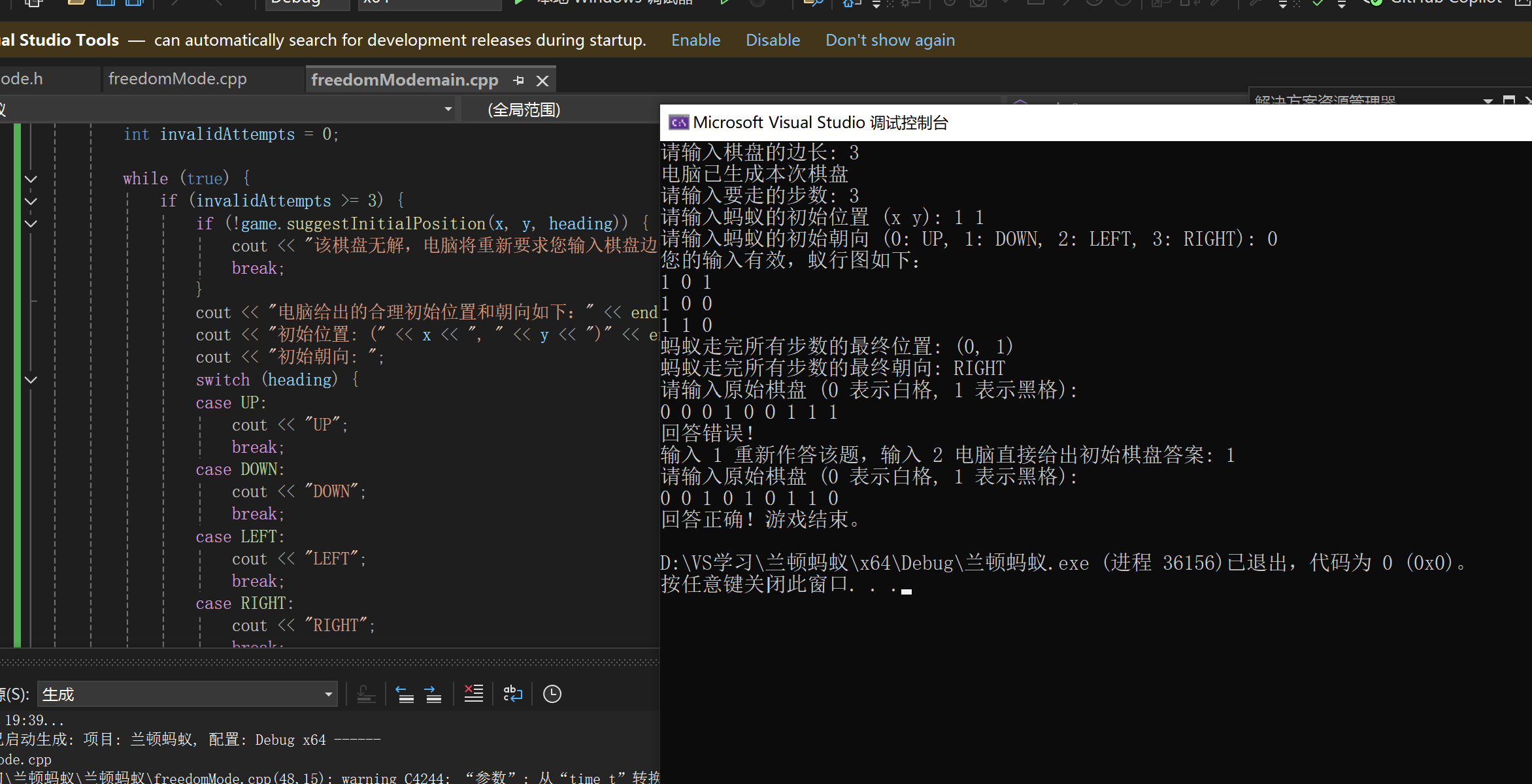
}

1. 实验记录和运行结果

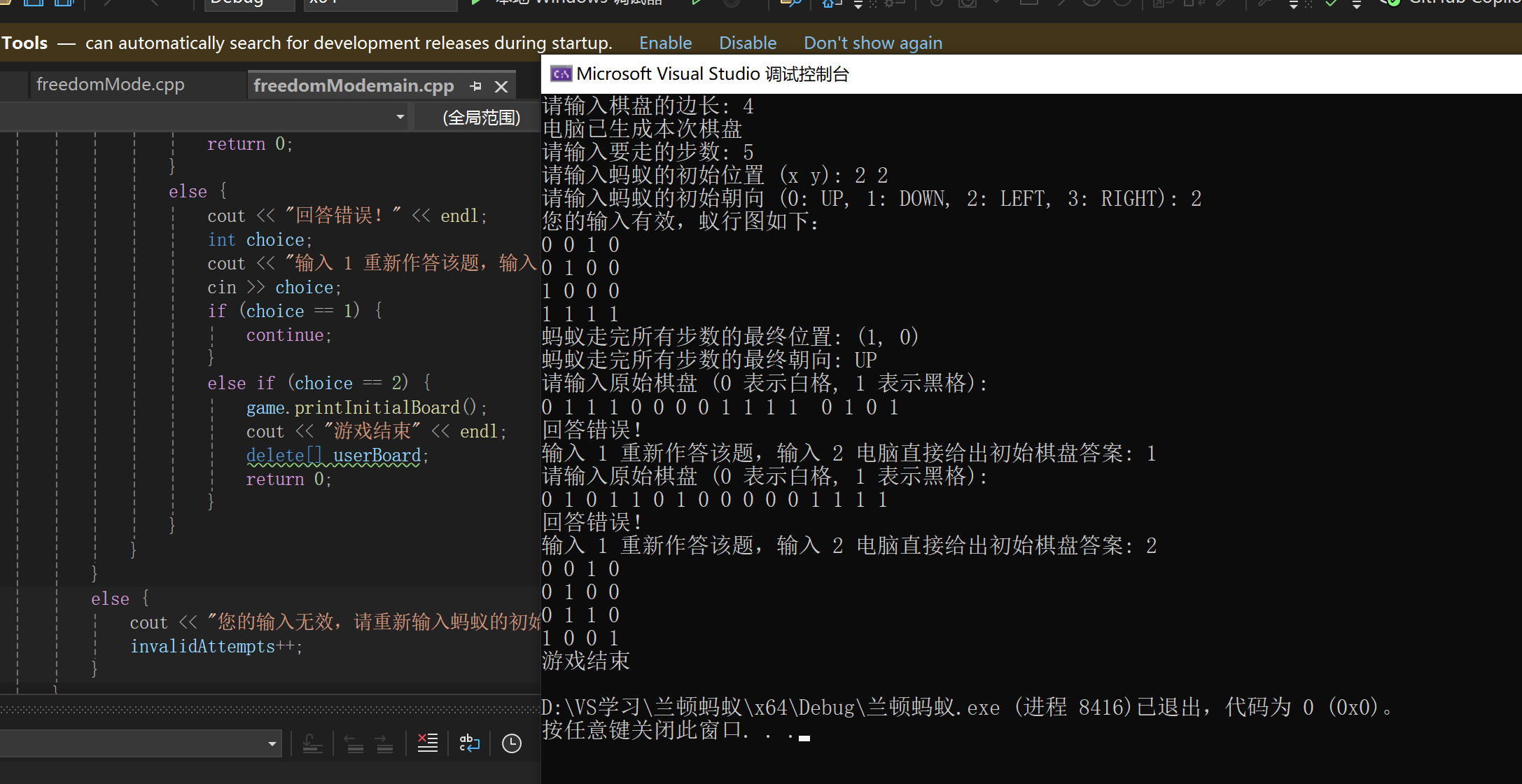
实验开始前的草稿



回答正确与错误界面



多次错误后电脑给出答案



用户多次输入无效后电脑给出提示

