



**数据结构设计性实验报告**

**课程名称 数据结构**

**题目名称 池塘夜降彩色雨**

**学生学院 计算机学院**

**专业班级 计算机科学与技术21(3)**

**学 号 3121005043**

**学生姓名 林睿鹏**

**指导教师 苏 庆**

**2022 年 12 月 20 日**

1. **需求分析**

设计一个程序，并且不调用图形化相关的库(不使用windows.h)演示下雨的情景，雨滴入水出现层层水波，水波逐渐消失，除此之外再增加一个风雨飘摇、电闪雷鸣的情景

1. 输入的形式和输入值的范围

输入的形式为字符，输入值的范围为{0,1,2,3,ENTER}

1. 输出的形式

输出形式：画面出现下雨的效果，且入水有水波，还会显示落入水中雨滴的颜色

1. 程序能达到的功能

①雨滴从窗口顶层随机位置出现，用户输入一次ENTER键，往下落一行，有风状态同时会往左侧移动一列

②落水时，产生水波，且水圈的数目随机，逐渐消失

③每出现落水的雨滴，都会打印出当前落水的雨滴的颜色

④有风状态下，雨滴呈倾斜下落，如被风吹动一般，于此同时，天空中的随机位置出现闪电

⑤雨滴的密度是随机变化的，有时雨滴稀疏，有时倾盆大雨

1. 测试程序，包括正确的输入及其输出结果和含有错误的输入及其输出结果

**正确输入**: {0,1,2,3,ENTER}

输出结果：输入1，可观察微风时雨滴的下落；输入2，观察风雨飘摇、电闪雷鸣的情景；输入ENTER：雨滴下落一行；输入0+ENTER，返回初始界面；输入0，程序结束

**错误输入**：非指定的输入

**输出结果**：不做出响应或直接结束程序

2、**概要设计**

数据类型：

1. 一些初始的定义

#define OK 1

#define ERROR -1

#define TRUE 2

#define FALSE -2

#define OVERFLOW 0

#define ROW 40 // 画面的宽度

#define COLUMN 80 // 画面的高度

typedef int Status; // 函数执行状态

typedef char RainType; // 雨滴形状

typedef char WaterType; // 水的形状

typedef char\* Thunder; // 雷电

(2)雨滴的数据结构

typedef struct RainNode{

RainType rainType = '|'; // 雨滴形状

int x = 0; // x的坐标

int y = 0; // y的坐标

int offsetY = 0; // y的偏置量，用来处理风吹的情况

int waterRingNum = 0; // 产生的水圈的数目

Color color = RED; // 枚举类型，指明颜色

int tag = 0; // 为0表明无雨，为1表明由于，为2表明斜的雨出界了

} \*EveryRain;

(3)水面的数据结构

typedef struct waterNode {

WaterType w; // 水波的形状

int move\_times = 0; // 水波移动的偏移量

int tag = 0; // 为0表明为正常的水，为其他的则表明为水波

} \*Water;

(4)存储当前屏幕上全部雨水的数据结构

typedef struct SqQueue {

EveryRain elem; // 存储空间的基址

int front; // 队头位标

int rear; // 队尾位标，指示队尾元素的下一位置

int maxSize; // 最大长度

} RainSqQueue; // 雨水的循环队列

(5)屏幕的数据结构

typedef struct ScreenNode{

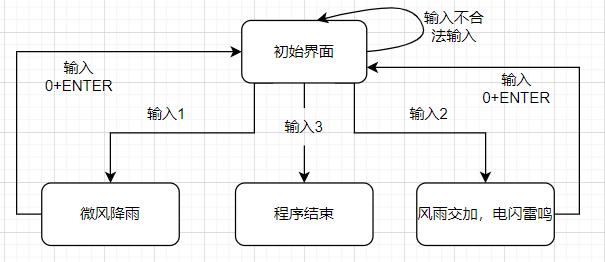
RainSqQueue rain; // 存放存在的全部雨滴的循环队列

RainType (\*screenIndex)[COLUMN]; // 对应屏幕上的坐标，负责显示内容，数组指针

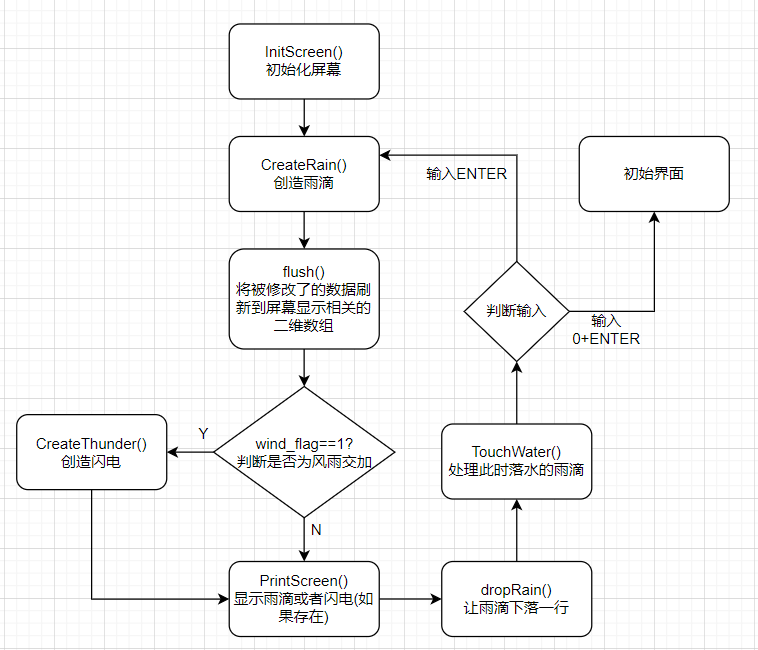
Water water; // 屏幕下方的水

} Screen; // 屏幕

主程序流程：



各程序板块的调用关系：



补充：

由于本次课设为图形化界面，我觉得仍有必要补充我的思路和设计。

首先，我没有调用window.h的库，所以为了整体效果，我需要用户做出一定的输入才能出现理想的界面。

1. 我设计了一个screenIndex二数组，对应当前画面的坐标系，它的作用就是存放内容，并能体现出画面的改变，如雨滴下落，rain队列的雨滴元素x坐标改变，然后再把队列的所有元素根据x、y坐标的对应关系，存放screenIndex坐标数组里面去，所以打印该二维数组，会发现全部雨滴的x下移了
2. 每当用户按下ENTER，都会在上界的随机位置产生雨滴，全部雨滴也会同时下落一行，这个时候把修改了的值放到screenIndex坐标数组去，再打印出来，就实现了模拟雨滴下落的过程。这就是代码最底层最核心的思想。
3. 详细设计

**引入的库函数**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include "Rain.h"

**基本操作**

void InitScreen(Screen &S); // 初始化屏幕状态

Status CreateRain(Screen& S, RainType type, int en\_times, int offsetY); // 产生雨滴，随机选取rain数组的元素补充雨滴

Status dropRain(Screen &S, int wind\_flag); // 让雨下落

void flush(Screen S); // 刷新一次，将新的内容预存到显示数组中，包括雨滴，打雷，风，水波

void PrintScreen(Screen S, RainNode\* fallIntoWater, int last\_index\_rain); // 刷新一次屏幕

int TouchWater(Screen& S, RainNode \*fallIntoWater); // 触水时会发生的一系列反应

void CreateWaterRing(Screen &S, RainNode e); // 产生水花

void CheckWaterRing(Water water, int op); // 在每次打印水之前，检查水波

void CreateThunder(Screen& S); // 产生雷电

Status EnQueue\_Sq(RainSqQueue& Q, RainNode e); // 入队

Status DeQueue\_Sq(RainSqQueue& Q); // 出队

Status QueueEmpty\_Sq(RainSqQueue Q); // 给队列判空

void freeScreen(Screen& S); // 将开辟的空间free掉

int getRand(int min, int max); //产生随机数，左闭右闭区间

**伪代码**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include "Rain.h"

void InitScreen(Screen& S)

// 初始化屏幕状态

// 输入：数据类型为Screen的变量S的引用

// 输出：初始化S的一些成员变量

S.rain.elem <- (EveryRain)malloc(sizeof(RainNode) \* ROW \* COLUMN + 1); // 难死了这几个初始化

S.rain.front <- 0;

S.rain.rear <- 0;

S.rain.maxSize <- ROW \* COLUMN + 1;

S.screenIndex <- (RainType(\*)[COLUMN])malloc(sizeof(RainType) \* ROW \* COLUMN); // 1维数组\*COLUMN

S.water <- (Water)malloc(sizeof(waterNode) \* COLUMN);

if S.rain.elem = NULL || S.screenIndex = NULL || S.water = NULL

return;

for i <- 0 to i < ROW do

for j <- 0 to j < COLUMN do

S.screenIndex[i][j] = 32;

for i <- 0 to i < COLUMN do

S.water[i].w <- '~';

S.water[i].move\_times <- 0;

S.water[i].tag <- 0;

void freeScreen(Screen& S)

// 释放S开辟的内存空间

// 输入：数据类型为Screen的变量S的引用

// 输出：利用free，释放S中成员变量利用malloc开辟的空间

// free(S.rain.elem);S.rain.elem = NULL；// 太妙了，此处不能free

// 因为结构体内存对齐S地址=S.rain的地址=S.rain.elem的地址，这里free就会把整个S给free了

// 此处为败笔，还望以后能学到如何解决此问题

free(S.screenIndex); // 指向数组的指针，直接释放即可

free(S.water);

S.screenIndex <- NULL;

S.water <- NULL;

// 产生雨滴，随机选取rain数组的元素补充雨滴

Status CreateRain(Screen& S, RainType type, int en\_times, int offsetY)

// 产生雨滴

// 输入：数据类型S的引用；滴形状type；en\_times 雨滴密度；offsetY 雨滴往左偏移的数量

// 输出：S中的结构体循环队列Rain入队了随机数目的新元素雨滴；成功返回OK，否则返回ERROR

// 随机选取下标，随机产生波纹数目，随机产生颜色

if NULL = S.rain.elem return ERROR;

// 随机选取坐标

for j <- 0 to j < COLUMN do

n <- getRand(1, 80) % en\_times;

if 0 = n // 被选中

var e;

e.rainType <- type; // 不同情况，雨滴不同

e.x <- 0;

e.y <- j; // 对应第一行

e.offsetY <- offsetY; // 对应第一行

e.waterRingNum <- getRand(1, 3);

e.color <- (Color)getRand(0, 6); // 此处强制类型转换

e.tag <- 1;

EnQueue\_Sq(S.rain, e); // 入队

return OK;

/\*

让雨滴下落

\*/

Status dropRain(Screen& S, int wind\_flag)

// 让雨滴下落一行

// 输入：数据类型为Screen的变量S的引用；wind\_flag 判断是否为有风的状态

// 输出：S中循环队列rain的全部元素的成员变量x++；成功返回OK，否则返回ERROR

if NULL = S.rain.elem return ERROR;

var Q = S.rain;

// 遍历队列，然后让他们的y坐标下移

for j <- Q.front to j != Q.rear do

Q.elem[j].x++;

// 此处的wind\_flag的设置是为了后期补充其他功能时更方便

if wind\_flag = 1

// 偏移量大于0，且y大于等于0，才减1，也就是说，y最小为-1

if Q.elem[j].offsetY > 0 && Q.elem[j].y >= 0

Q.elem[j].y--;

Q.elem[j].offsetY--;

// 此处有一个极难发现的bug，那就是执行完上面的if之后，y变为-1，同时接触水面，但是tag未修改，导致产生了water[-1]

if Q.elem[j].y < 0

Q.elem[j].tag <- 2;

j = (j + 1) % Q.maxSize;

return OK;

/\*

刷新一次，将新的内容预存到显示数组中，包括雨滴，打雷，风，水波

\*/

void flush(Screen S)

// 刷新一次，存入当前的新数据

// 输入：数据类型为Screen的变量S

// 输出：将S中的用以屏幕显示的screenIndex二维数组初始化为空，再将S的rain队列中全部元素将以x、y都对应的方式存入screenIndex数组中

for i <- 0 to i < ROW do

For j <- 0 to j < COLUMN do

S.screenIndex[i][j] <- 32;

var Q = S.rain;

for j = Q.front to j != Q.rear do

x <- Q.elem[j].x;

y <- Q.elem[j].y;

// 处理y为-1的情况

y < 0 ? S.screenIndex[x][0] <- 32 : S.screenIndex[x][y] <- Q.elem[j].rainType;

j=(j+1)%Q.maxSize

/\*

打印出屏幕当前所有内容

\*/

void PrintScreen(Screen S, RainNode\* fallIntoWater, int last\_index\_rain)

// 打印出屏幕当前的应该显示的内容

// 输入：数据类型为Screen的变量S；fallIntoWater 落入水中的雨滴的数组； last\_index\_rain 落入水中雨滴的数目，对应fallIntoWater的最后一位有效元素的下标

// 输出：将S中的screenIndex的内容打印到屏幕上，water数组的内容打印到屏幕上，以及落入水中雨滴的颜色

// 打印屏幕内容

for i <- 0 to i < ROW do

for j <- 0 to j < COLUMN do

printf("%c", S.screenIndex[i][j]);

printf("\n");

// 打印水

CheckWaterRing(S.water, 1);

for i <- 0; i < COLUMN do

printf("%c", S.water[i].w);

printf("\n");

CheckWaterRing(S.water, -1); // 还原水滴

// 打印雨滴信息

if fallIntoWater = NULL return;

printf("从左至右，下面为落入水里的雨滴的信息:\n");

for i <- 0 to i < last\_index\_rain do

printf("第%d滴雨的颜色为: ", i + 1);

switch (fallIntoWater[i].color)

{

case RED:

printf("RED\n");

break;

case ORANGE:

printf("ORANGE\n");

break;

case YELLOW:

printf("YELLOW\n");

break;

case GREEN:

printf("GREEN\n");

break;

case CYAN:

printf("CYAN\n");

break;

case BLUE:

printf("BLUE\n");

break;

case PURPLE:

printf("PURPLE\n");

break;

default:

break;

}

}

/\*

触碰到水时，会抹去存在，并且产生水花

\*/

int TouchWater(Screen &S, RainNode\* fallIntoWater)

// 检查落水的雨滴，产生水波

// 输入：数据类型为Screen的变量S的引用；fallIntoWater 落入水中的雨滴数组

// 输出：创建水圈，将入水的雨滴元素从S的rain队列中出队，再存入数组fallIntoWater；返回值 落入水中雨滴的数目，对应fallIntoWater的最后一位有效元素的下标

// 遍历最后一行位置的雨滴，当drop发生后，产生水花，并抹掉它们在坐标和循环数组的存在

// 抹去存在，直接除去循环数组里面的就好了，因为我的flush操作会刷新一遍显示数组

// 从队头往后遍历COLUMN次(一行最多COLUMN个)，当x坐标最后一行的，就把tag置为0，并且出栈

the\_last\_rain <- 0;

pre\_front <- 0; // 记录加1前的front值

for i <- 0 to i < COLUMN do

if S.rain.elem[S.rain.front].x = ROW

RainNode e <- S.rain.elem[S.rain.front];

pre\_front <- S.rain.front;

DeQueue\_Sq(S.rain); // 删除这些落水的，此处的出栈需要对S，不能对局部变量

// 当tag为1时才产生水波，如果为2，即出界，不产生水波

if e.tag = 1

CreateWaterRing(S, e);// 蹦出水花，此处传过去的参数也要是S，不能是局部变量

S.rain.elem[pre\_front].tag <- 0;

fallIntoWater[the\_last\_rain++] <- e;// 检查落水雨滴的颜色

return the\_last\_rain; // 返回数组的最后一个下标

// 产生水波

void CreateWaterRing(Screen &S, RainNode e)

// 实现产生水波

// 输入：数据类型为Screen的变量S的引用；e 雨滴

// 输出：S中water数组对应e的坐标y的元素被设置为出现水波的状态

// 考虑到边界条件

y <- e.y; // e下落时的y坐标

if y < 0 return; // 勇敢if，不怕溢出

S.water[y].move\_times <- S.water[y].tag = e.waterRingNum;

/\*

在每次打印水之前，检查水波

@param op 为1时负责检查随波，为-1时负责还原水波

\*/

void CheckWaterRing(Water water, int op)

// 检查水面状态，查看哪里出现了水波

// 输入：water 水面数组；op 对应操作，1为检查，-1为复位

// 输出：当参数op为-1时，水面数组全部复位；当op为1时，让对应位置出现水波

// 注意边界条件

for i <- 0 to i < COLUMN do

if op = 1

if water[i].tag != 0

offsetLeft <- i - (water[i].move\_times - water[i].tag + 1);

offsetRight <- i + (water[i].move\_times - water[i].tag + 1);

if offsetLeft >= 0

water[offsetLeft].w <- '(';

if offsetRight <= COLUMN - 1

water[offsetRight].w <- ')';

water[i].tag--; // 减一次，下次就会偏移更远

else if op = -1

if water[i].tag = 0

water[i].w <- '~';

water[i].move\_times <- 0;

// 创造闪电

void CreateThunder(Screen& S)

// 创造闪电

// 输入：数据类型为Screen的变量S的引用

// 输出：将随机位置产生的闪电存入S的screenIndex数组，以便打印到屏幕上

// 设定雷电的单位长度

len <- ROW / 8;

// 从边界往后遍历，随机选取点产生雷电

for i <- len + 1 do i < COLUMN do

i <- i + len + 2;

num <- getRand(1, 10);// 1 / 5的概率

if num % 5 = 0

// 产生雷电

indexX1 <- 0; // x坐标

indexY1 <- i;

indexX2 <- 0;

indexY2 <- 0;

indexX3 <- 0;

indexY3 <- 0;

for offset <- 0 to offset < len do

S.screenIndex[indexX1++][indexY1 - offset] <- '/';

indexX2 <- indexX1 - 1;

indexY2 <- indexY1 - offset;

for offset <- 0 to offset < len do

S.screenIndex[indexX2][indexY2 + offset + 1] <- '\_';

indexX3 <- indexX2;

indexY3 <- indexY2 + offset + 1;

for offset <- 0 to offset < len do

S.screenIndex[++indexX3][indexY3 - offset] <- '/';

Status EnQueue\_Sq(RainSqQueue& Q, RainNode e)

// 入队

// 输入：Q 存放雨滴的循环队列；e 雨滴元素

// 输出：e入队到Q队列去，成功返回OK，否则返回ERROR

// 若当前队列不满，在队列的尾元素之后，插入元素 e 为新的队列尾元素

if Q.front = (Q.rear + 1) % Q.maxSize return ERROR;

Q.elem[Q.rear] <- e;

Q.rear <- (Q.rear + 1) % Q.maxSize;

return OK;

Status DeQueue\_Sq(RainSqQueue& Q)

// 出队

// 输入：Q 存放雨滴的循环队列

// 输出：让Q队头的元素出队；成功返回OK，否则返回ERROR

// 若队列不空，则删除队列Q中的头元素

if TRUE = QueueEmpty\_Sq(Q) return ERROR; // 队空报错

Q.front <- (Q.front + 1) % Q.maxSize; // Q.front循环加1

return OK;

Status QueueEmpty\_Sq(RainSqQueue Q) // 判断队列Q是否空，若空则返回TRUE，否则FALSE

// 判断队列是否为空

// 输入：Q 存放雨滴的循环队列

// 输出：为空返回TRUE，不为空返回FALSE

return Q.front = Q.rear ? TRUE : FALSE;

// 左闭右闭区间，产生随机数

int getRand(int min, int max)

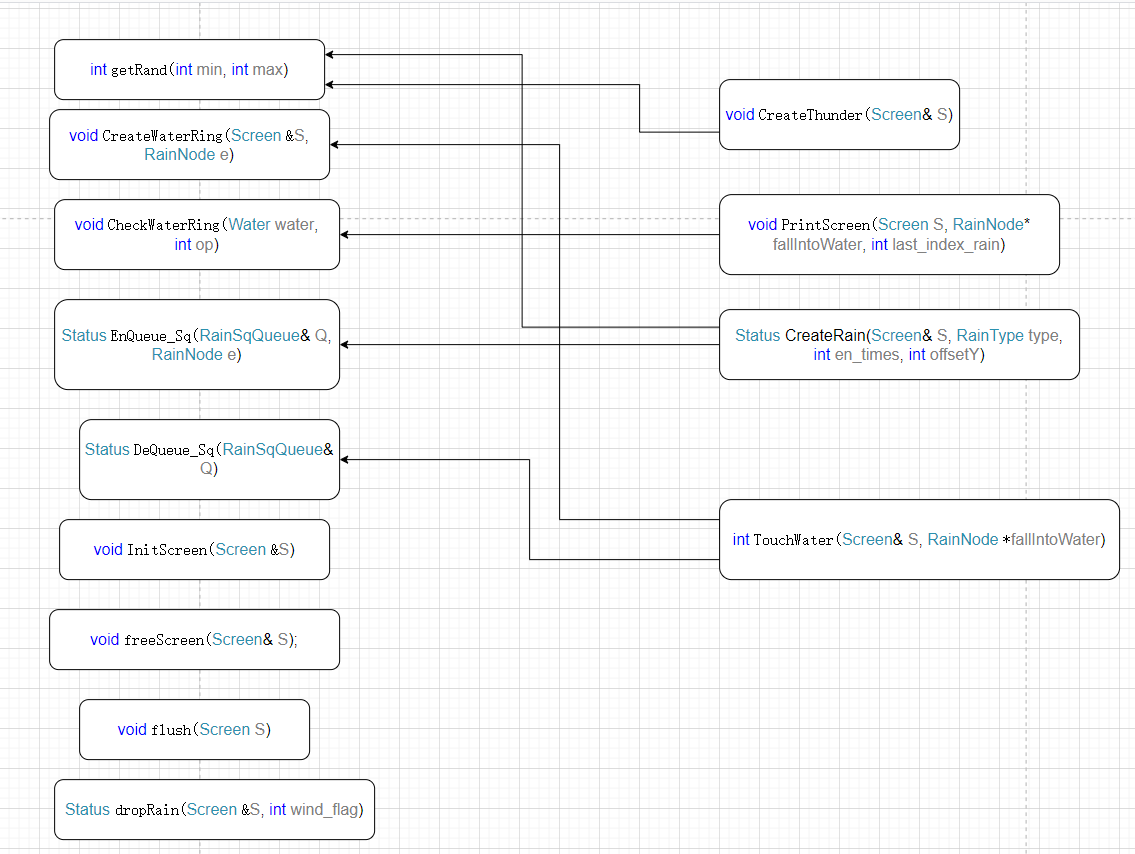
// 得到随机数，左闭右闭

// 输入：min 下界；max 上界

// 输出：返回一个[min,max]的随机整数

return (rand() % (max - min + 1)) + min;

函数和过程的调用流程图



4、**调试分析**

(1)过程遇到的问题

①实验中不知道怎么逐渐地显示水波，后面我想到了当雨滴落入水时，将对应坐标y的水滴设置一个move\_times，即记录产生的水波数目，然后每次flush的时候，这个下标为y的water元素就会让左侧的water元素形状变为‘(’，右侧变为‘)’(模仿水波)，具体让距离多远的water变就根据move\_times来，具体see code

②初始化也难到我了，我没想过原来二维数组初始化这么要命啊

③碰撞如何处理，这里我不知道怎么处理倾斜的雨滴碰到左边的边界的情况。

本来我想的是，干脆从记录当前所有雨滴的队列删除，但是队列只能操作头尾。

然后我就干脆，直接把它所在的位置设置为空格，具体就是当碰撞到边界时，不删除这个雨滴，让它也跟着下降(y=0)，但是它出现的位置设置为空格，这样实现碰撞消失了的效果。我有担心过会不会它覆盖了其他雨滴或者其他雨滴覆盖它，但是经我仔细，发现这是不会，因为当前的所有雨滴都是同步运作的，但我按下ENTER，它们全部下降一行，虽然画面很乱，但是每个雨滴的先后位置关系都是严格区分的，所以，不会说它插队到了前一滴雨的位置，而把它覆盖了。

④free的时候，太坑了，我free一个结构体中的结构体的成员变量，结果VS直接崩溃。然后我调试了一晚上才发现，靠，最外层的结构体的地址=嵌套的结构体的地址=内层结构体的成员变量的地址，相当于我直接free了这个最外层的结构体，这样搞好像有问题，毕竟最外层结构体只是个局部变量，而且free后我还访问了它的成员变量。

⑤说到局部变量这也太坑了，我不太敢多用指针，大多都是引用(虽然底层也是指针)，所以有时候我得好好区分这家伙到底是局部变量还是真能修改函数外的变量的值。苦不堪言。

⑥有时候想了一些很巧的法子来让某个模块的代码变得很简单、很巧妙，但是后期维护或者加功能的时候又不得不把“包装”拆开，最后还是回到了最规范的写法。所以写代码要考虑到后期补充和维护，不要一味追求巧。

(2)算法的时空分析和改进设想

我存入雨滴利用了循环队列，一开始我只想开辟存在的所有雨滴数量的空间，但是后面发现扩容很难，于是直接开辟了画面坐标那么大的队列，出于两点，我利用了define定义画面行和列的大小，即用户可以实现轻易改变画面的大小；二，虽然一开始浪费了很多空间，但是随机程序执行(雨滴下落的越来越多)，那些空间也被利用上了。复杂度的话，我遍历基本上用for，一是我设置的都是顺序表存储结构，遍历快，二是简单，虽然很多遍历利用双重for都是O(n^2)的确有点捞就是了。

(3)经验和体会

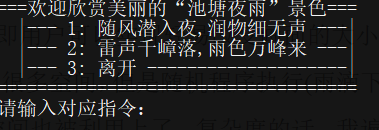
前面已经谈了经验了，这里站在宏观的角度审视这次课设以及谈谈最宝贵的体会。代码实现不难，但难的是创意和勇气。好比我怎么实现图形化的界面呢？明明以前最多用printf打印文字或者\*，今天却要用它来模拟下雨。我没学过图形化编程啊，这个题目我肯定不能选......诸如此类的想法害我迟迟不敢踏出那一步，只敢留在原地保守地换着更高级的语言、框架来CRUD。总结，本次实验我没学到很多新的知识，编程经验的提高也不多，但是至少，它教会了我去想、去做！

1. **用户使用说明**

该程序操作简单，用户只需要根据提示输入即可。

1. **测试结果**

初始界面



在当前界面输入不合理的内容，程序仍会保持当前初始界面

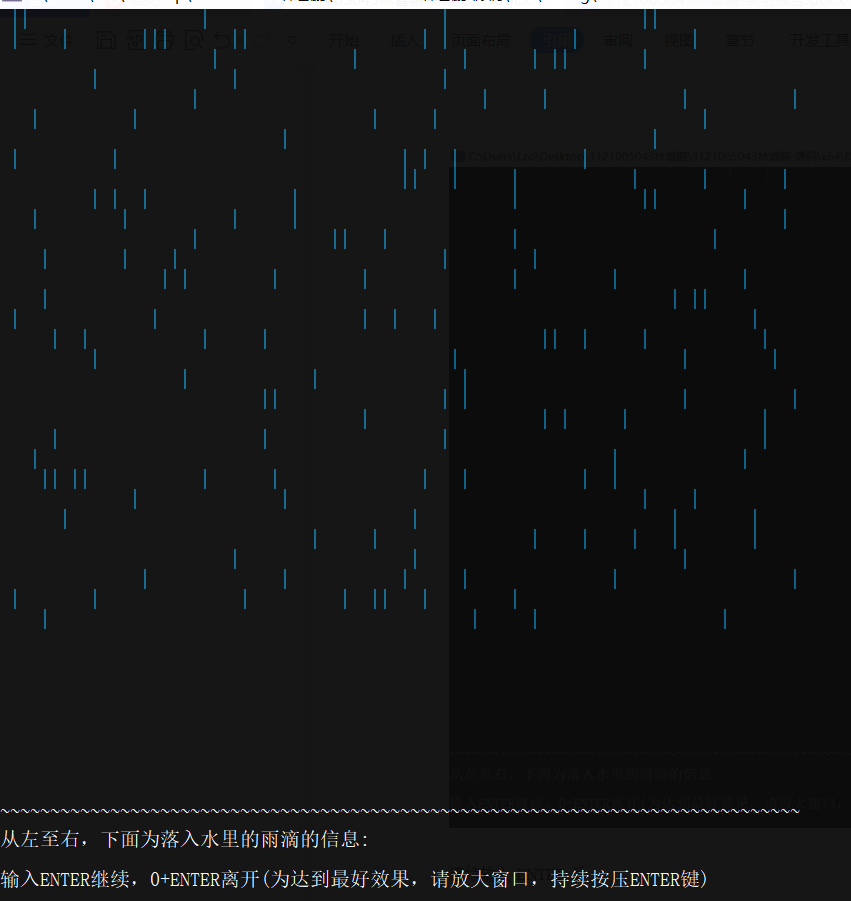
1. 输入1

进入“无风之雨”界面

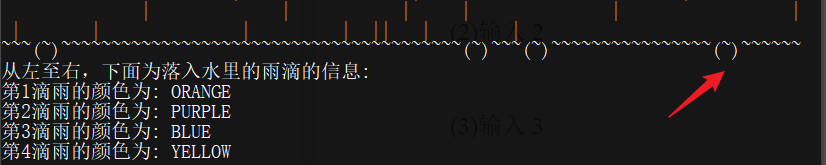


连续输入ENTER后，出现下落的雨滴

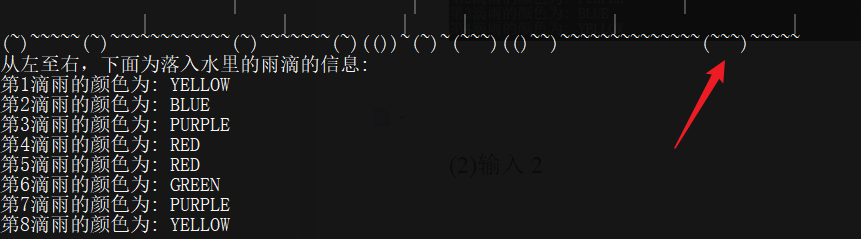
注：程序实现了密度变化，注意此时的雨滴密度



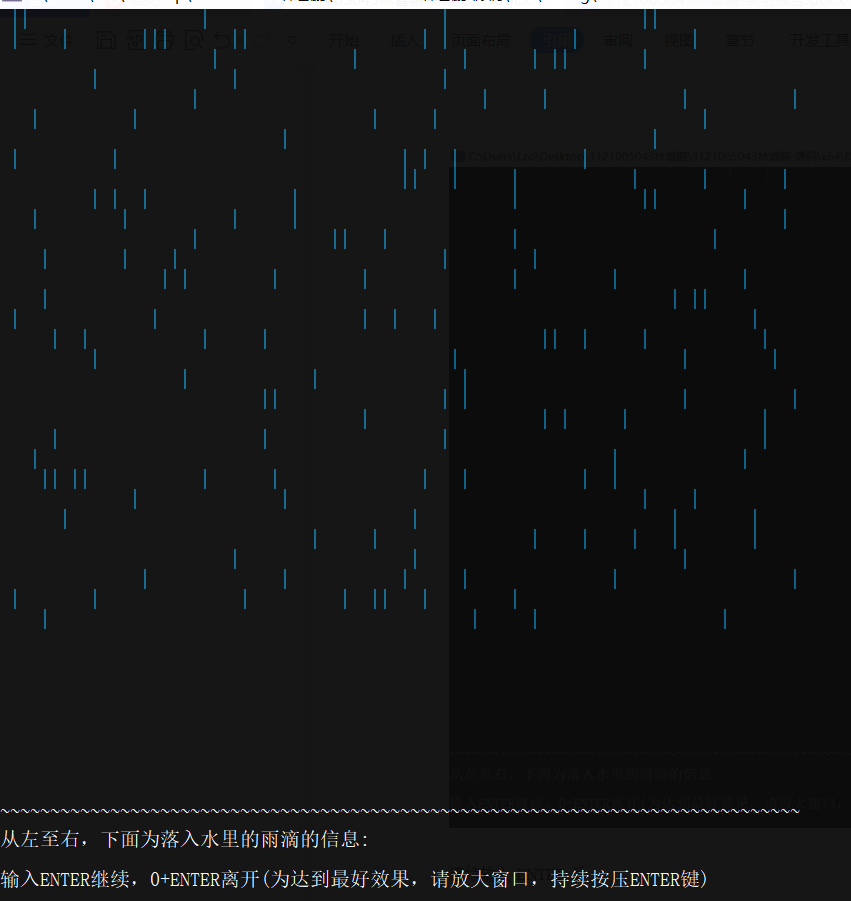
雨滴入水时，会产生水波，并且告知用户雨滴颜色



下面的图片是上面图片的雨滴往下落一行的情景，可以看到水波的确是逐层扩散



当雨滴下落一段时间后，可以清晰看到雨滴的密度发生了变化

 变化前

 变化后

输入0 + ENTER回到主界面，测试成功。

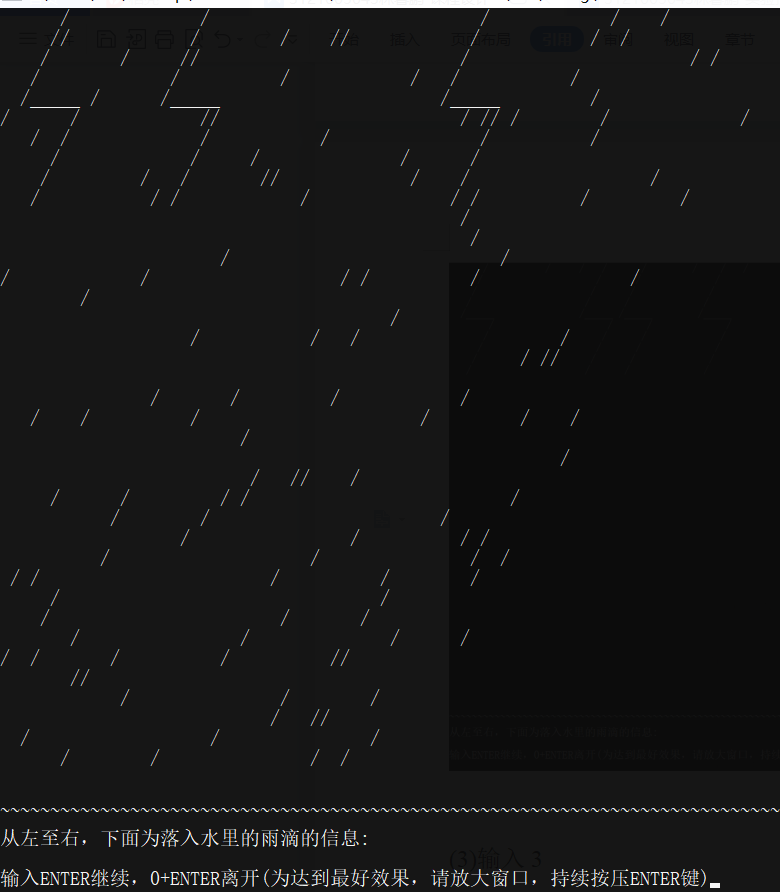
(2)输入2

进入“暴风骤雨”、“电闪雷鸣”界面

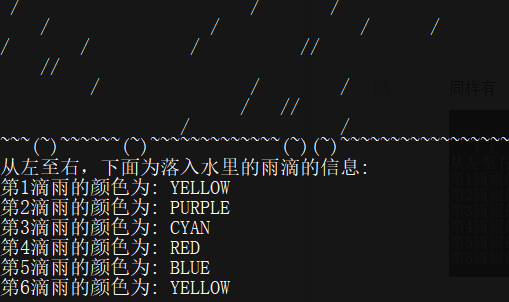
除了实现“无风之雨”的密度变化，落水有逐渐消失的水波，颜色提示，还增加了“风雨飘摇”的效果和“电闪雷鸣”

看到出现了闪电

连续输入ENTER后

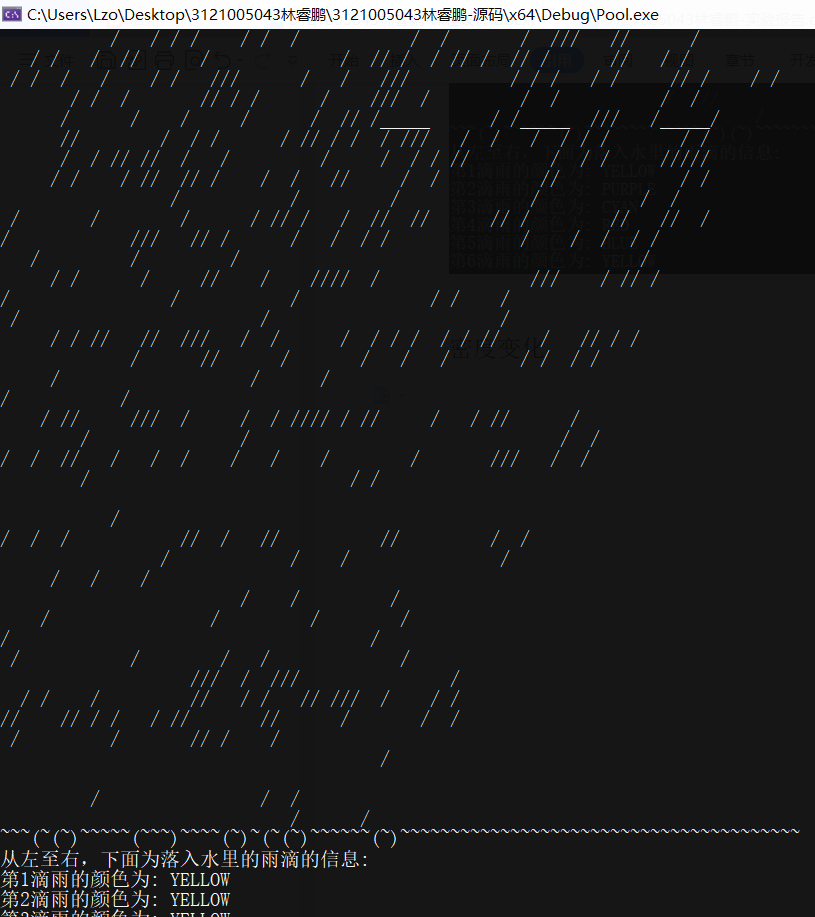
 画面里有闪电和飘飘的雨滴

同样有水波和颜色提示



密度变化

 变化前

 变化后

输入0+ENTER，返回主界面成功

(3)输入3

程序结束

