Against Pandemic 大作业报告

组名： 南外废柴理科生

组员：

- 陈思雨 人文社科96 2019012633

  游戏构思，功能函数设计，代码、参数调试，后期可玩性测试及优化

- 葛依然 人文社科98 2019012668

  游戏构思，主要代码编写，游戏整体框架搭建，图形界面绘制、人机交互界面的实现

环境：Visual Studio 2017

源代码见https://github.com/PandemicProject/Pandemic

Against Pandemic 大作业报告
- 简介
- 设计思想
  - 问题分析
    - 首先分析“疫”部分。
    - 其次分析”战“部分
    - 综合部分
  - 基本思路
- 框架
- 功能
  - 图形绘制
  - 用户交互
- 调试和解决办法
  - 中文输入
  - 多余空格
  - 格式不符
  - 拼写错误
- 不足
- 心得体会
- TIPS：

简介

游戏“战疫公司”的设计灵感来源于一度火爆的游戏“瘟疫公司”。结合当下全球疫情肆虐、各国努力抗疫的背景，本小组萌生了制作一款”反向版瘟疫公司“游戏的想法。于是”战疫公司“——一款以玩家自主决策、分配资金，帮助初始国战胜疫情的游戏，就此诞生。

设计思想

问题分析

实现本游戏所涉及的问题可以分成两大类——“战”和“疫”。即 ”战“——玩家如何发布指令、影响疫情发展；”疫“——病毒如何在人群中“传染”、疫情现状如何体现这两大问题。

首先分析“疫”部分。

- ”病毒如何在人群中传染“
  此问题可以转化为”如何确定每日感染人数“。解决此问题可以使用如下两种方法：

  a. 直接设定感染率，辅以随机函数实现
  b. 仿照现实感染原因，通过“距离”判断，辅以一定的感染率和随机函数

  在经过方法调研并结合疫情实际情况，我们决定用方法2，并通过**“动态点阵图”**实现感染情况可视化和感染判断。
- “如何体现疫情现状”
  此问题相当于对游戏涉及数据的选择性呈现。结合上一个问题的实现思路，以及“战”部分的功能操作，我们决定采用**“点阵图颜色”+“数据”**呈现的方式。根据现实中对疫情的新闻报道，分为**确诊感染者(infected)”**，**“密切接触者(exposed)”**，**“健康者(healthy)”**三大类，以及后期可以通过指令激活的**“隔离监测者(quarantine)**"。

其次分析”战“部分

- “玩家如何发布指令”
  此问题相当于“如何实现交互功能”。 即通过循环、条件判断函数的组合，实现 **“玩家输入指令--激发相应函数--读取指令信息--作出判断”** 这一过程。
- “玩家的指令如何对疫情发展造成影响”
  此问题涉及玩家可以采取指令的种类、每种指令的影响力度，在代码编写时引申为**函数的设计、调用**，以及**变量之间的影响变化**。

综合部分

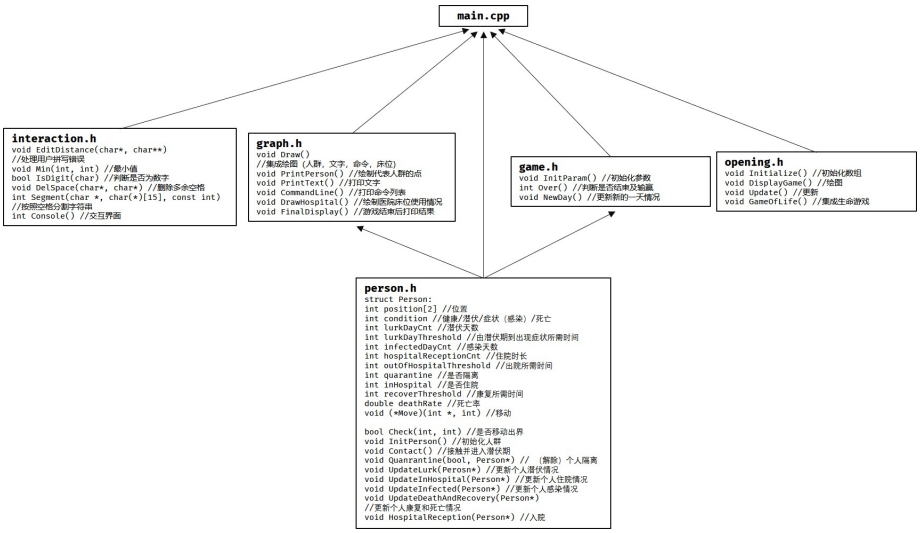
结合以上分析，我们发现以**”人“**作为对象，通过每一个”人“不同状态（感染/隔离/确诊/健康）的录入，加和后即为一个国家的疫情现状。之后，所有的”人“作为一个整体，其感染机率、死亡率、日产能等又统一受到玩家（即国家）指令的影响，最终通过可视化反映为疫情现状。在这样的粗略框架下，游戏涉及的主要问题得以解决。

基本思路

1. 搭建游戏基本框架，编写胜负判定、初始化、退出等函数。
2. 编写“人”为对象的代码文件。其中解决“动态点阵图”下的感染状态判定，是否进入医院、是否康复。
3. 编写交互指令代码文件。规定每一个指令的作用，同时配备交互界面弹出，输入报错。

4. 编写“动态点阵图”呈现代码文件。打印每一个“人”的“位置”（一个点代表一个人），设定颜色区分感染状态。
5. 将以上文件整合，按需求在主文件中调用。设计参数。
6. 反复测试调整，优化代码。根据测试情况修改参数及部分功能函数内容、胜负条件等，提高游戏可玩性。

框架



功能

图形绘制

图形绘制采用Easy Graphics Engine图形库提供的 `<graphic.h>`

- 开始界面

  因为生命游戏的发明者也是在新冠疫情中去世，所以写了最基础的生命游戏作为该游戏的开始界面

- 主体

  - 人群

    每个点代表一个人，其背后是一个结构体。点的颜色代表状态：

    - 健康：白色
    - 潜伏期：绿色
    - 确认感染：红色
    - 死亡：不显示

    每个人的位置每天会更新一次，隔离者和死亡者除外

  - 床位

    床位占用情况用进度条表示，白色部分代表已占用床位

  - 文字

    文字部分每天更新天数（Day）、 健康人数（# healthy）、潜伏期人数（# exposed）、确诊感染人数（# infected）、 死亡人数（# dead）、隔离人数（# quarantine）空余床位数 (Vacant Hospital Bed)、钱（Money Left）、剩余口罩量（Mask Left）、药品开发

天数倒计时（Days Requierd for medicine）和疫苗开发剩余天数（Days Required for vaccine）。

用户交互

交互采用Easy Graphics Engine 图形库提供的 `<sys_edit.h>` 实现用户键盘输入命令。命令列表及其含义与影响如下：

- **build [number]**: 建医院，共[number]个床位

  注：消耗游戏币
- **research [number1] [number2]**: 分别向药物研究和疫苗开发投入[number1]，[number2]个游戏币以加速开发
- **mask**: 要求人们戴口罩

  注：传染率降低，每日口罩消耗增加，口罩供应不足会导致传染率增加
- **quarantine**: 将感染的人隔离

  注：隔离的人不会带来经济效益
- **distance**: 要求人们保持社交距离

  注：移动意愿降低，人均经济效益减少
- **work**: 要求人们复工

  注：移动意愿增加（人与人之间的距离更可能小于安全社交距离），人均口罩产量和经济效益增加
- **rm mask**: 要求人们摘掉口罩

  注：传染率增加，每日口罩消耗降低
- **rm quarantine**: 解除隔离

  注：恢复隔离的人的经济效益
- **rm work**: 要求人们不再外出工作

  注：移动意愿降低，人均经济效益降低
- **quit**: 停止输入命令，继续更新疫情状态
- **new**: 重新开始
- **exit**: 退出游戏

调试和解决办法

主要的问题是用户输入不可控，很可能出现中文输入，空白输入，拼写错误，多余空格，大小写混乱，不符合命令格式等问题，因此这一块需要错误处理等措施。

中文输入

首先通过判断输入中每一位的ascii码是否小于0得到输入中是否有中文，如果有则抛出异常。

多余空格

- 末尾空格：从后往前遍历，空格转为‘\0’，遇到字母或数字停止
- 开头空格：从前往后遍历，若为空格，指针指向下一个字符，遇到字母或数字停止
- 词与词间多余空格（同时处理大小写）：

```
▷ A: 用户输入剔除首尾空格后的结果，B：初始化新字符数组
i <- 0
for j <- 0 to length[A] - 1 do
    ▷ 若A[j]为大写字母则转成小写
    if A[j] = ' ' and A[j + 1] = ' '  then
        continue
     end if
     B[i] <- A[j]
     i <- i + 1
end for
return B
```

格式不符

用**try catch**语句尽量捕捉可能出现的错误，以build命令为例：

首先将删除多余空格后的用户输入按空格切分，储存在二维数组command中，每行为一个词或数，并返回词和数的总数。由command[0]可知用户输入的命令为“build”。接着判断词和数的总数是否为2，command[1]是否为数字，command[1]是否大于1位且首位为0，所需资金是否超出现有钱数。若出现上述情况中任意一种，则抛出异常。

```
try
{
    if (cnt != 2) //cnt为总词数
    {
        throw false;
    }
    //IsDigit()判断是否每个字符均为数字
    if (!IsDigit(command[1]))
    {
        throw false;
    }
    if (strlen(command[1]) > 1 && command[1][0] == '0')
    {
        throw 0;
    }
    int num = atoi(command[1]);
    if (num * costPerBed < money)
    {
        throw 'f';
    }
    //略过主体部分
}
catch (bool)
```

```
{
    printf("Invalid input. Format: build [number].");
}
catch (int)
{
    printf("The number begins with a zero.");
}
catch (char)
{
    printf("Hospital Construction failed. Not enough money.");
}
```

拼写错误

这一部分参考了gdb地输出。首席按遍历所有命令，采用动态规划计算用户输入相较于每个命令的**最小编辑距离**，将距离最小的词作为对用户输入的猜测然后输出。

```
function Min(a, b)
    if a < b then
        return a
    else
        return b
    end if
end function

▷ cmdList为1 × 12的字符指针型数组，全局，为命令列表
▷ input为字符指针，re为字符二级指针
function EditDistance(input, re)
    flag <- 0
    now <- 0xff
    dp[0..14][0..14]
    for k <- 0 to 11 do
        target <- cmdList[k]
        ▷ 在target和input前插入一个0分别存入_in和command
        for i <- 0 to length[_in] do
            dp[i][0] <- i
        end
        for i <- 0 to length[command] do
            dp[0][i] <- i
        end
        for i <- 0 to length[_in] do
            for j <- 0 to length[command]
                f <- _in[i] = command[j]
                f <- 1 - tmp
                tmp = Min(dp[i][j-1]+1, dp[i-1][j-1]+f)
                dp[i][j] = Min(dp[i-1][j]+1, tmp)
```

```
            end for
        end for
        if dp[m-1][n-1] <= now then
            *re = target
            now = dp[m-1][n-1]
        end if
    end for
end function
```

不足

- 模型缺乏现实意义
- 很多函数的时间复杂度和空间复杂度都非常高，如Contact()的循环套循环
- 模块搭建有待完善
- 代码鲁棒性不够

心得体会

陈思雨：我印象最深的其实是代码运行的环境配置。因为我之前一直使用的是Dev C++编译，但是这次考虑到需要使用ege库，所以依次下载了VSC，VS2019，结果配置调试中一直没有成功，运行时经常报错。在各种Debug和设置调试上花了很长时间，最后才发现原因是和葛依然同学的代码原编写平台不同，安装了VS2017版的才成功运行。这段反复失败纠错的经历提醒了我代码环境的重要性，以及在未来团队合作时，要注意尽量达到环境设置一致，可以节省很多时间QAQ... 除此之外，看着我们的想法最后能成为有一定可玩性的游戏，也是非常有成就感的一件事。在反复调参测试的过程中，我也第一次切身体会到了游戏开发的不易——如何控制难易程度，使玩家有“挑战感”带来的纠结时，也有获胜的“希望”。这些都是我在本次大作业中印象深刻的收获。

葛依然：一开始是因为想到SEIR模型才萌生了做这个的念头，但是后来上手的时候才意识到SEIR模型似乎不适用于时间离散的情况，因此改用了现在这个模式。写代码的过程中可以切实地体会到计算机并不会像人那样“I feel like this is the right way”或者“Maybe I should do this or that”，而是只会一丝不苟地遵循代码地逻辑，这其实也是在反向逼迫自己先把逻辑搞清楚再上手，有时候两个步骤换一下顺序就会有很大区别。最后在交互方面，不能要求用户每一次输入都完美地符合标准，而是要去提高代码自身地鲁棒性，从而不至于用户输入时一点点疏忽程序就卡死。

TIPS：

一些玩本游戏的小攻略：

- 使用rm work指令要慎重，因为是全面停工停产，有可能会加速游戏失败
- 选择build hospital的时机及数量很重要。床位一旦建立，无论是否有人入住都会花钱
- research经费分配时注意结合现状
- 不要忽视infected数量增长，有可能会比dead更“可怕”
- 积极的指令实施、人为“干预”疫情往往能取得好的效果