# GameCenterHUSTMSE2102DWJ报告

1.各个模块及实现

## AutoRank 自动排名 分数计录系统

1. 数据结构设计
2. 使用scoreblock保存数据块
3. 使用vector型变量 s\_vScoreList 保存数据列表
4. 算法和程序流程
5. 读写分数数据：通过二进制的方法对分数数据逐个存放
6. 通过重载scoreblock类的比较符号使用std::sort实现排序功能
7. 模块间接口描述
8. 输出分数文件

static bool WriteFile(const char\* path);

path : 输出文件地址

返回值: 文件异常时返回false

1. 写入分数文件

static bool ReadFile(const char\* path);

path : 输出文件地址

返回值: 文件异常时返回false

1. 输出分数排名

static void PrintList(int layer);

layer: 输出时各行预输出的制表符数

1. 插入分数块

static void Insert(const scoreblock& obj);

static void Insert(const scoreblock& obj);\

1. 快速输入并输出文件，计入分数

static void FastIOScore(unsigned long score, const char\* path);

1. 系统实现

将数据以字符串的形式读取实现对其二进制形式的读取

1. 需求分析
2. 实现游戏程序需要的排行榜
3. 以二进制形式存储减少空间占用
4. 增加读取速度与效率
5. 防止玩家篡改排名数据
6. 实例代码



## Control 控制模块 实现计时与按键监听

1. 数据结构设计
2. Class timer: 计时器
3. Struct keycheck 独立按键监听
4. 算法和程序流程
5. 计时器: 保存开始计时与获取的时间，计算两次的时间差
6. 按键监听：通过每帧更新与监听按键，区分出松开 长按 按下 短按等状态
7. 模块间接口描述:

timer：

    timer();

    //设置偏移时间

    bool SetOffset(int iOffset);

    //清除时间

    bool Clean();

    //是否计时中

    bool GetIsRunning();

    //开始计时

    bool Start();

    //结束计时

    bool Stop();

    //重新计时

    void Reset();

    //读取时间

    //停止时返回记录时间的负数

    clock\_t Read();

    //确保timer为开启状态，否则返回-1 一般情况返回时差

clock\_t SleepTill(clock\_t ctSleepTime);

keycheck:

    //gvKey 检测按键的虚拟键位码

    keycheck(DWORD gvKey);

    //默认初始化，以255（不存在）为检测键

    keycheck();

    //获取状态 0松开 1按下 2持续按下 3长时间按下(由LongPress设置)

    int getstat();

    //gvKey 检测按键的虚拟键位码

    void reset(DWORD gvKey);

    static unsigned int iLongPress;

    //设置longpress

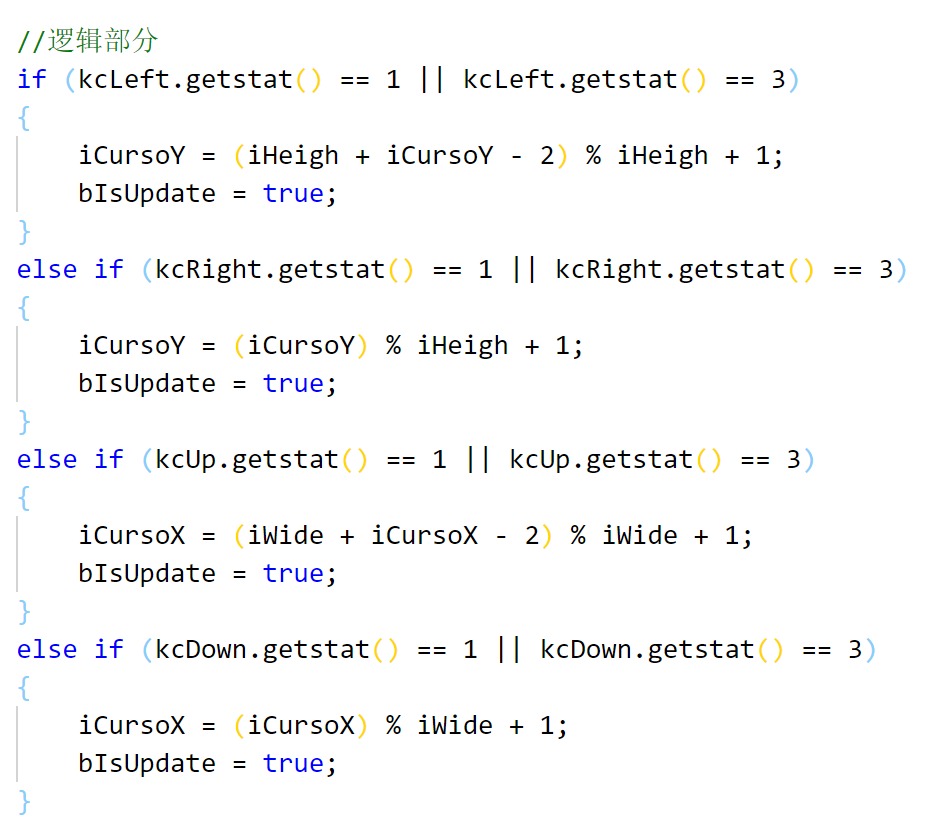
    static unsigned int SetLongPress(unsigned int newarg);

1. 系统实现

timer：使用ctime中的clock()获取时间

keycheck：使用 windowsAPI中的GetAsyncKeyState()获取虚拟键为状态

1. 需求分析
2. 时间记录准确
3. 按键获取准确无冲突 (getch缺点)
4. 可以获得无法用ascii表示的字符
5. 可以分析出多种按键状态(松开 长按 按下 短按等状态)
6. 示例代码



## Conif 控制台交互模块 实现选项式用户交互

1. 数据结构设计

class conif 控制台下的用户交互选择界面

1. 算法和程序流程
2. 在构造时先计算出各行选项要达到居中效果时所需的空格
3. 将输入的内容预先载入缓冲区(字符串数组)中 减少延迟感
4. 绘制界面时将每行分为 边框行 空行 单字符行 标题行 选项行 四类逐行输出
5. 使用keycheck监听用户输入
6. 模块间接口描述

    //创建一个界面

    //TotalOpt  总共选项数

    //OptName   各个选项的名称

    //Title     标题名称

    //ScreenW   界面宽度

    //SelectCh  选中提示符

    //BoardCh   边框字符

    //UpVk      上键的虚拟键位码

    //DownVk    下键的虚拟键位码

    //EnterVk   选择键的虚拟键位码

    conif(int TotalOpt, char (\*OptName)[MAXOPTLEN],

     char\* Title = NULL,

     int ScreenW = 27,

     char SelectCh = '\*',

     char BoardCh = '#',

     DWORD UpVk = VK\_UP,

     DWORD DownVk = VK\_DOWN,

     DWORD EnterVk = VK\_RETURN);

    //显示界面

    //返回选中的选项

    int Display();

1. 系统实现

如果有有效输入 则 更新画面

更新按键状态 - 读取输入

直到用户输入回车则退出否则循环

1. 需求分析

直接输入命令过于生硬 采用上下选择更加人性化

1. 实现效果



## Mine 扫雷模块 实现扫雷游戏

1. 数据结构设计

class MineMap 承担扫雷游戏的类

原始地图，记录 地雷即空地周围的地雷数

std::vector<std::vector<int> > MineMap ::vOriginalMap;

状态地图，呈现给玩家地图

std::vector<std::vector<char> > MineMap ::vStateMap;

1. 算法和程序流程
2. 创建扫雷地图:

.在两种地雷地图中设置边界

.按顺序放置地雷块 在随即交换两个块(总块数次)实现打乱效果

.根据地雷放置统计每个格周围的地雷数

.寻找一个周围0格最多的位置预先触碰

1. 触碰时自动触碰为0的格

.通过递归直到被触碰的格不为0

1. 将输入的内容预先载入缓冲区(std::stringstream ssTmpInput;)中 减少延迟感
2. 遍历状态地图，根据每格信息及是否被选中打印游戏界面
3. 模块间接口描述

    //构造函数

    MineMap(int wide, int heigh, int mine\_num);

    //绘制地雷状态地图

    void Print\_Original();

    //绘制地图状态地图

    void Print\_State();

    //判断玩家是否失败

    bool IsDead();

    //判断玩家是否胜利

    bool IsWin();

    //还有可能是玩家退出游戏

    //开始游戏

    bool Play();

1. 系统实现

如果有有效输入 则 更新画面

更新按键状态 - 读取输入

如果有有效输入 则 根据输入命令(按键)更新地图状态

直到游戏结束或退出则退出否则循环

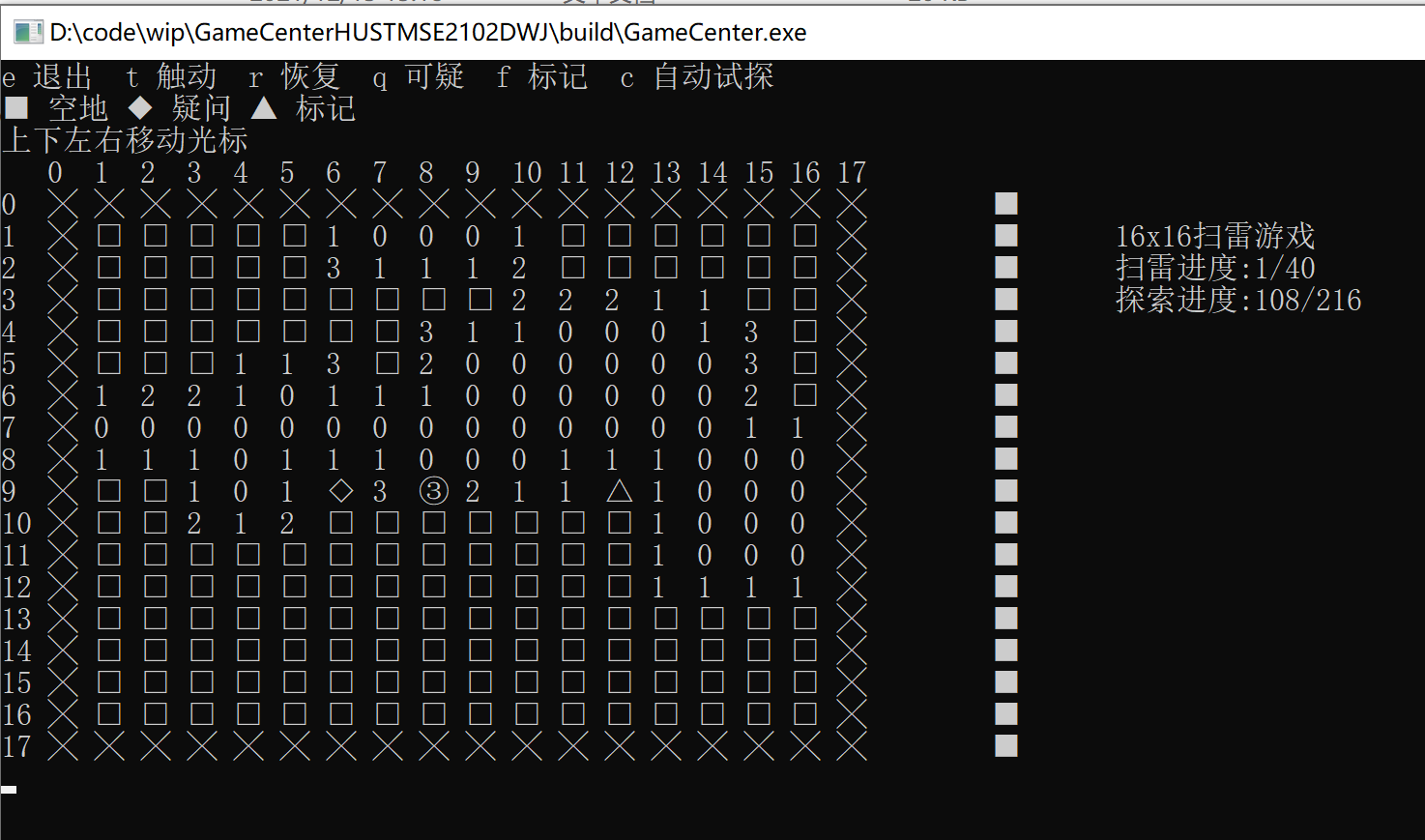
1. 需求分析

扫雷游戏如果用命令的方式需要重复输入大量坐标

使用伪光标方式可以大大减少输入难度，且由于扫雷游戏常是从一个区域开始到完成的特点，使用伪光标可以极大增加游戏体验

通过自动试探功能进一步减少繁琐操作，增加游戏体验

1. 游戏效果



**Snake 贪吃蛇模块 实现贪吃蛇游戏**

1. 数据结构设计

Struct SnakeBody{

COORD pos;

SnakeBody \*next;

}

用链表储存蛇的躯干信息

1. 算法和程序流程
2. 初始化蛇身长度、位置生成第一个食物
3. 循环判断是否按下键盘，存储键盘输入的指令
4. 判断经过一定时间后，执行键盘输入的指令，判断蛇的方向，判断蛇头前方一格内是否为蛇身或食物或墙壁
5. 移动蛇身

.新增一格链表头并打印蛇身，清空链表尾并打印空格

.若吃到食物则再随机生成新的食物

.若吃到蛇身或墙壁则结束游戏

1. 模块间接口描述

2. 日志

//Log.md的内容

3. 心得

1. 通过对常用的程序模块(conif/control)进行封装，能够极大的提高程序的开发效率，增加代码复用性，并方便代码维护。

2. 更新画面时通过将即将打印的内容储存后再一起输出，能够增加画面的流畅度。