课程设计一

控制台小游戏 - 吃豆豆

时间：2019.09.17 - 2019.10.13

唐金麟 17 级 计算机

联系方式：TangJinlin@smail.nju.edu.cn

目录

控制台小游戏——吃豆豆

一、概述

主要内容：

以 吃豆豆大作战 为基础参考，实现了运行在 Windows 控制台下的一个吃豆豆小游戏。玩家目标是吃掉所有豆豆，途中，若玩家撞到怪兽则游戏结束。游戏中存在超级豆子，吃到后，所有怪兽静止，且玩家进入无敌状态（可撞怪兽使之回到初始位置）。玩家可使用键盘方向键进行选择、控制移动，回车键确定，Esc 键返回，空格键暂停。

已实现的目标：

- 地图支持自定义编辑（可编辑地图中的：空地、墙、豆子、超级豆子、小怪初始位置及数量、吃豆人初始位置）
- 可选择游戏难度（即设定不同的小怪移动速度）
- 怪兽可追踪吃豆人位置，往吃豆人的所在位置靠近
- 可查看历史记录 TOP 10，并且可对游戏记录进行回放
- 在回放的过程中，支持按下方向键后，退出回放模式，在当前情形下，操纵吃豆人，进入正常的游戏模式，继续游戏
- 有用户友好的 UI 界面（比如：吃到超级豆子后吃豆人变彩色闪烁，且下方显示进度条倒计时；选择界面高亮显示选项）

二、主要类的设计

1）依次介绍各个类的设计：

①Position 类

由于是在控制台下实现本游戏，所以显示界面的所有元素都是字符，而一些较为美观，且符合游戏中的元素的特殊字符一般都是 2 个英文字符的宽度，所以这里界面设计的主要思想使将整个控制台运行窗口看成一个二维的画布，横坐标以每两个英文字符的宽度为单位，纵坐标则以每行为单位。

而有了画布之后，画布上的每一个点其实可以看作一个数据结构，这就是 Position 类的意义。

数据成员及成员函数：

```
11  class Position
12  {
13  protected:
14      string key;//该点的字符
15      int x, y;  //坐标
16      int color; //颜色
```

```
27      //设定字符
28      void setKey(const string str);
29      //设定显示坐标
30      void setXY(int a, int b);
31      //在指定坐标处显示
32      void print();
33      //清空该坐标处的字符
34      void clear();
```

通过这个类，可以轻松的实现在控制台中特定坐标的某个点输出特定颜色的特定字符（比如："■"）。

②MapPos 类

在 Position 类基础上，可以采用继承的方式，定义出 MapPos 类：表示地图中的某个点。相比 Position 类，MapPos 类多了一个成员变量用于标识这个点的类型属性（空白、墙、豆子、超级豆子），同时也多了与类型相关的两个方法。

```
15      //地图中的某点的类型：空白、墙、豆子、超级豆子
16      enum MAP_POS_TYPE { SPACE, WALL, PEAN, SUPER_PEAN };
17      class MapPos: public Position
18      {
19          int type; //地图中的某点的类型
20      public:
21          //设定地图中某点的类型
22          void setType(int t);
23          int getType() { return type; }
24      };
```

③Map 类

将 MapPos 对象构成的一个二维数据，封装成一个整体，这就是用于表示游戏地图的一个类，Map 类。类中，还有一些与游戏运行相关的成员变量在其中。

数据成员：

```
34  class Map
35  {
36      //地图中的每个点，看作一个个不同类型的元素
37      MapPos points[MAP_SIZE][MAP_SIZE];
38      //玩家得分
39      int scores;
40      //胜利目标得分
41      int target_scores;
42      //超级豆子带来的冷却效果时间
43      int freezeTime;
```

对外的提供接口：

```
51      //初始化生成地图(使用指定的文件)
52      void init(const char* filepath, Pacman &pacman, vector<Ghost> &ghosts);
53      //某点图形重新输出
54      void renew(int x, int y) { points[x][y].print(); }
55      //返回某点(x,y)的类型,以判断是否可达/计分
56      int goXY(int x, int y) { return points[x][y].getType(); }
57      //吃掉(x,y)处的豆子
58      void delPean(int x, int y);
59      //检查(x,y)位置是否超出地图范围,以防止数组越界
60      bool oK(int x, int y);
61      //吃掉(x,y)处的超级豆子
62      void delSuperPean(int x, int y);
63      //找到A点可以到达B点的路径中的下一步的方向
64      int findDir(Position &A, Position&B);
65  };
```

这里，Map 类的**主要功能**是：

①从地图文件中，读取地图布局，初始化 MapPos 对象构成的二维数组，以及初始化吃豆人的位置，小怪的位置及数量。

②同时，提供了一系列与地图有关的操作，比如地图中 A 点到 B 点的路径寻找（寻路算法，小怪移动策略中会使用到），移动位置的合法性检查等等。

寻路算法说明：这里寻找地图中 A 点到 B 的路径，从本质上来说，可以看成一个走迷宫的过程，也就是要找出一条路径。寻路算法有很多种，较为著名的有 A star 算法，而这里的地图规模不大，对效率的要求不是非常高，所以这里采用了实现较为简单的，也易于理解的 BFS 方法，就可以找出 A 点到 B 点的最短路径，即：从 A 的周围出发，一层一层向外扩散的搜索，直到搜索到 B 点（使用一个队列记录邻居中可达的点，执行广度优先搜索，并且记录下每个点的前一个点，便于再搜到目的地时回退）。

④Ghost 类

这个类表示游戏中的怪兽。这个也继承自 Position，因为从某种意义上来说，小怪也是二维画布上的一个点，是需要作为一个点，在此画布上输出显示的（同理，表示吃豆人的 Pacman 类也是这样）。

数据成员：

因为这是个会在二维画布上会移动的“点“，所以相比 Position 类，此类多了几个成员变量，用于表示移动方向和初始位置（用途：在吃豆人无敌状态下被碰到，小怪要回到初始位置）。

```
6   class Ghost:public Position
7   {
8       //表示移动方向
9       DIRECTION go;
10      //初始位置坐标
11      int init_x, init_y;
```

注：这里为了提高可读性，上下左右四个方向使用了枚举类型 DIRECTION 来表示。

```
9   //上下左右
10  enum DIRECTION { UP, DOWN, LEFT, RIGHT };
```

提供给外部的接口：

主要的功能是：朝某个方向移动，以及检测自己是否与吃豆人相碰。

```
14      //在(x,y)位置放置小怪
15      Ghost(int x, int y);
16      //小怪朝某个方向移动，且在Map中要检测移动是否合法（比如，墙无法前进）
17      int move(Map &map,Pacman &pacman);
18      int move(int dir,Map &map, Pacman &pacman);//重载：指定方向的移动
19      //碰撞检测：若不在冷冻期，则GameOver；否则，该小怪回到初始位置
20      bool hit(Pacman &pacman, Map &map);
21  };
```

⑤Pacman 类

表示游戏中的主角：吃豆人。与 Ghost 类相像，也是继承自 Position 类，也有提供给外部用朝某个方向移动的接口，用于实现吃豆人在地图上的移动。

```
4   class Pacman:public Position
5   {
6       DIRECTION go;
7   public:
8       friend class Game;
9       Pacman() {};
10      //在(x,y)位置放置吃豆人
11      void init(int x, int y);
12      //pacman朝某个方向移动，且在Map中要检测移动是否合法（比如，墙无法前进）
13      void move(int dir, Map &map);
14      //返回吃豆人当前所在的位置坐标
15      void getXY(int &a, int &b);
16  };
```

⑥RecordItem 类

游戏中加入了历史记录回放的功能，而实现回放功能的关键数据结构就是 RecordItem 类。此类的一个对象，表示一局游戏的游戏记录，记录下了本局游戏的时间戳、分数、一些游戏配置参数（小怪移动速度、数量）、总共的时间周期个数，以及吃豆人和小怪的移动方向。有了这些数据，就可以做到复现某一局游戏，只要按照 RecordItem 类的对象给定的参数配置游戏参数，然后在运行游戏的每个时钟周期，给出吃豆人和小怪的移动方向，即可做到回放某一局游戏。

数据成员：

```
12  class RecordItem //一条记录
13  {
14      string name;//记录名（时间戳）
15      int score;//分数
16      int speed;//小怪移动速度
17      int ghost_num;//小怪数（3~5个）
18      int steps_num;//总共经历的时间周期
19      vector<int> steps;//记录的每个时钟周期时吃豆人/小怪的移动方向
```

⑦Record 类

有了 RecordItem 类还不便于实现完整的“历史记录“功能模块，所以这里还需要有一个专门负责管理历史记录条目的模块，即 Record 类。Record 类数据成员较为简单，就是一个

RecordItem 对象构成的 vector 容器，Record 类的主要功能体现在它对外提供的一系列接口，

主要的功能有：①从文件读取历史记录条目，以及将历史记录信息存入文件；②增加/删除一条历史记录；③展示历史记录。

```
28  class Record //记录管理器
29  {
30      const char *filepath; //存放历史记录的文件名
31      vector<RecordItem> items; //历史记录条目
32  public:
33      friend class Game;
34      Record():filepath("game.record"){};
35      //从文件中读取历史记录信息
36      void read();
37      //将历史信息存入文件
38      void save();
39      //增加一条历史记录信息
40      void add(int the_score,const vector<int> &steps,int ghost_num,int speed);
41      //删除指定下标的历史记录条目
42      void del(int x);
43      //展示历史记录
44      void show();
45  };
```

⑧Game 类

总体来说，整个游戏有四个模块：吃豆人、小怪、地图、历史记录。将四个模块整合到一起，就构成了整个游戏的数据结构，即 Game 类。而整个游戏的运行，简而言之，就是对四个模块对外提供的接口进行调用。

```
16  class Game
17  {
18      //四个模块：吃豆人、小怪、地图、历史记录
19      Pacman pacmanX;
20      vector<Ghost> ghosts;
21      Map mapX;
22      Record record;
23
24      //小怪移动速度调节
25      int GHOST_SPEED,speed_value;
26      //已过的时钟周期的个数
27      int time_counter;
28      //记录每个时钟周期时吃豆人/小怪的移动方向
29      vector<int> steps;
30      //标识是否处于回放状态
31      bool play_flag;
```

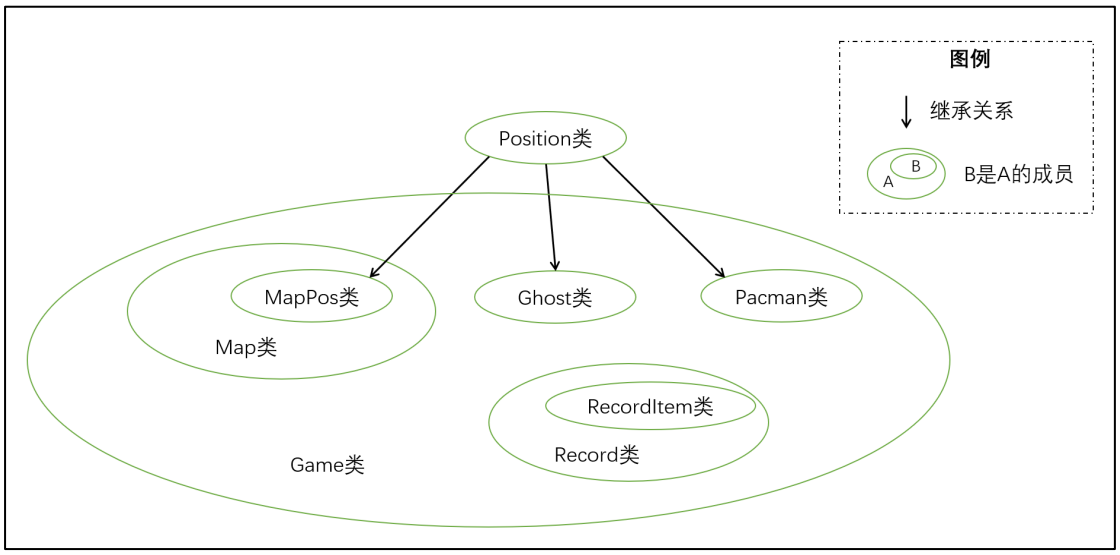
对外的接口，以及一些私有的成员函数：

```
33      public:
34          Game();
35          //游戏初始化
36          void init();
37          //游戏初始界面
38          int start();
39          //游戏配置选择
40          bool settings();
41          //历史记录展示
42          int show_record();
43          //游戏循环
44          bool loop();
45          //游戏回放
46          void play(int x);
```

```
48      private:
49          //游戏暂停
50          void pause();
51          //游戏结束
52          bool game_over();
53          //游戏胜利
54          bool game_win();
55          //分数等相关游戏信息的UI
56          void infoUI();
57          //帮助信息UI绘制
58          void helpUI();
59          //暂停后，重新绘制所有界面
60          void refresh();
61      };
```

对外的接口：主要是提供给外部使用（在 main 函数中被调用），用于启动整个游戏逻辑，包括了初始界面的显示，游戏开始，历史记录展示等功能。
私有的成员函数：主要用于实现一些不需要在外部被调用的函数的功能，比如各类 UI 的绘制，游戏暂停、失败、胜利逻辑。

2）综上所述，各个类之间的关系如下图所示：



三、程序的功能特点和运行操作方法

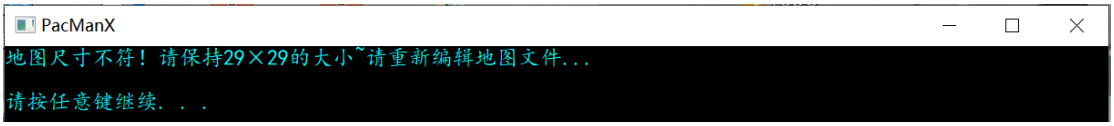
首先，对程序目录下的各文件进行说明：
①PacManX.exe
程序可执行文件。采用 Visual Studio 2017 在“Debug”“x86”配置下，“项目”-“属性”-“C/C++”-“代码生成”设置页面中，运行库选择多线程调试（/MTd）（静态编译），生成。
②map.txt
地图文件。必须存在于 PacManX.exe 同目录下，否则游戏无法读取地图。

③game.record

游戏历史记录文件。可有可无，若不存在，则代表当前无任何的历史记录信息，下次完成一局游戏后会自动创建。

游戏支持的地图编辑功能是通过编辑 map.txt 文件来完成的。经过测试，这里使用 win10 自带的记事本（或写字板）或者 VS 2017 的编辑器打开显示效果较好，而用 Notepad++打开显示效果不好（默认字体下，显示字符的宽度不标准，会导致排列不整齐）。

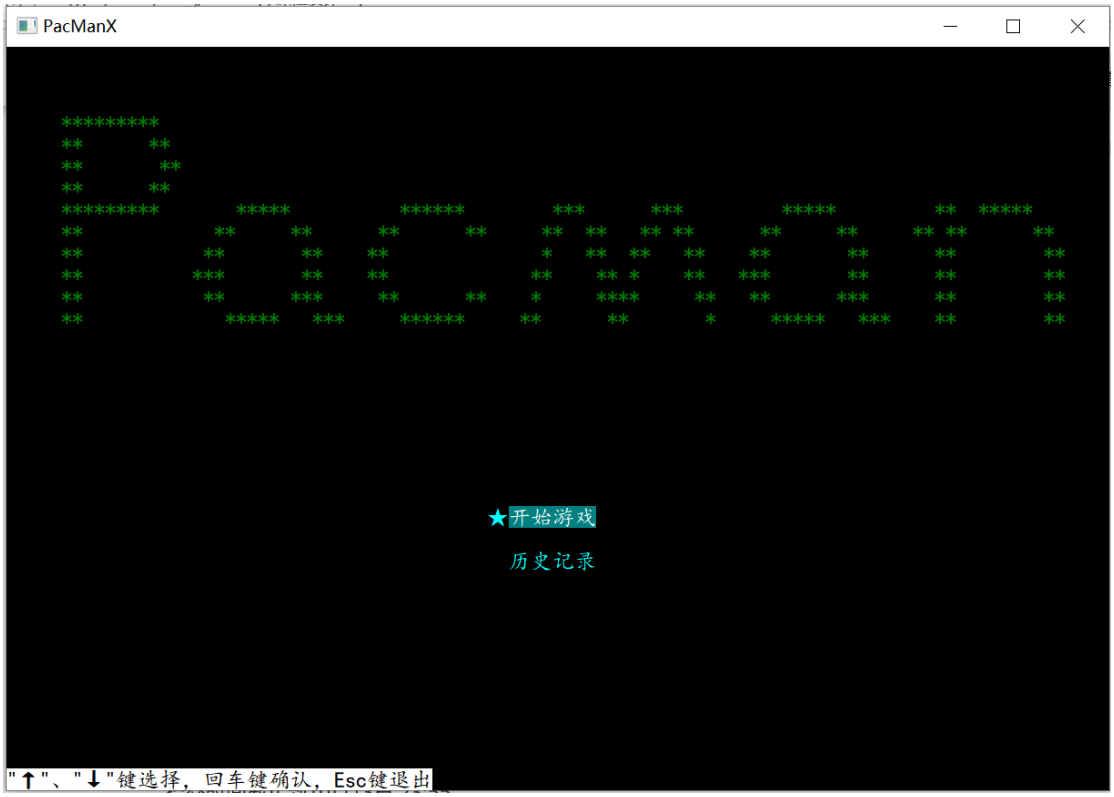
打开后，文件内容如下，只需要将对应的表示各类元素的字符复制编辑到对应位置，即可编辑出自定义的地图。若地图文件格式不符合要求，则游戏会在选择完难度，初始化游戏地图的时候，输出对应的错误提示信息，如下：



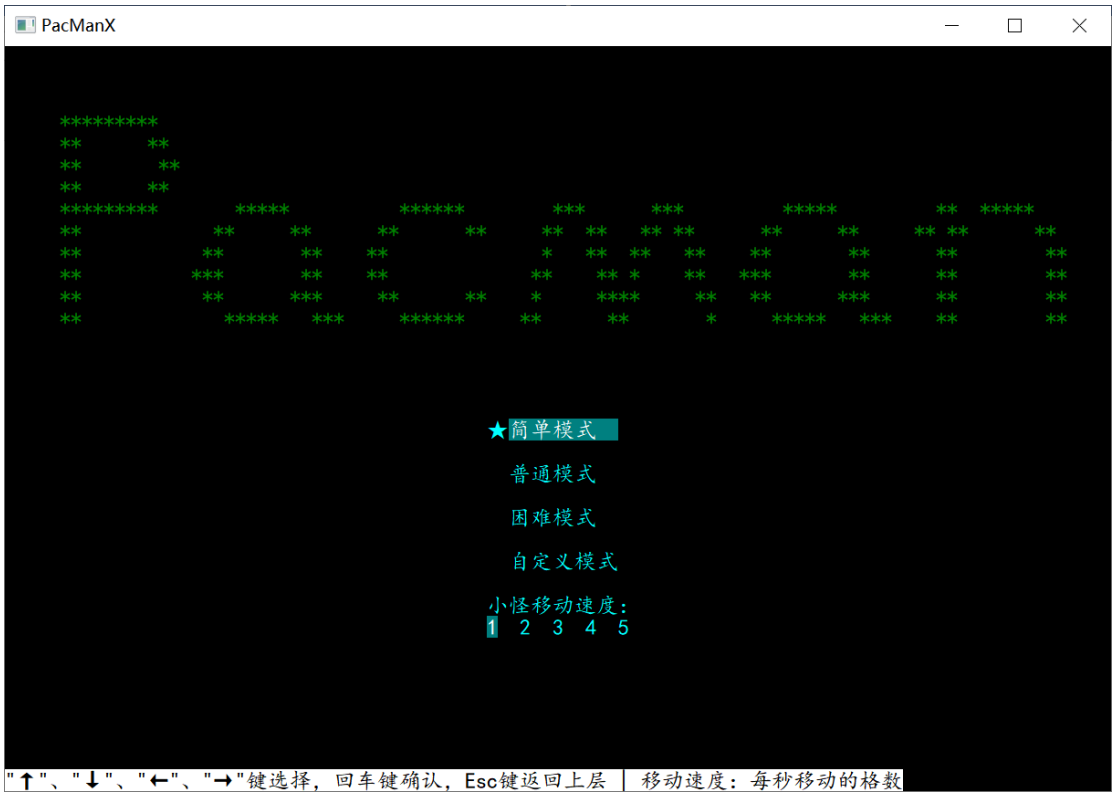
在地图文件正确的情况下，即可正常开始游戏。

说明：推荐打开.exe 文件后，窗口上方标题栏处右键，进入“属性”-“字体”设置页面，设置字体为“楷体”，不勾选粗体，大小为 16（或者 18），以便获得较好的显示效果。

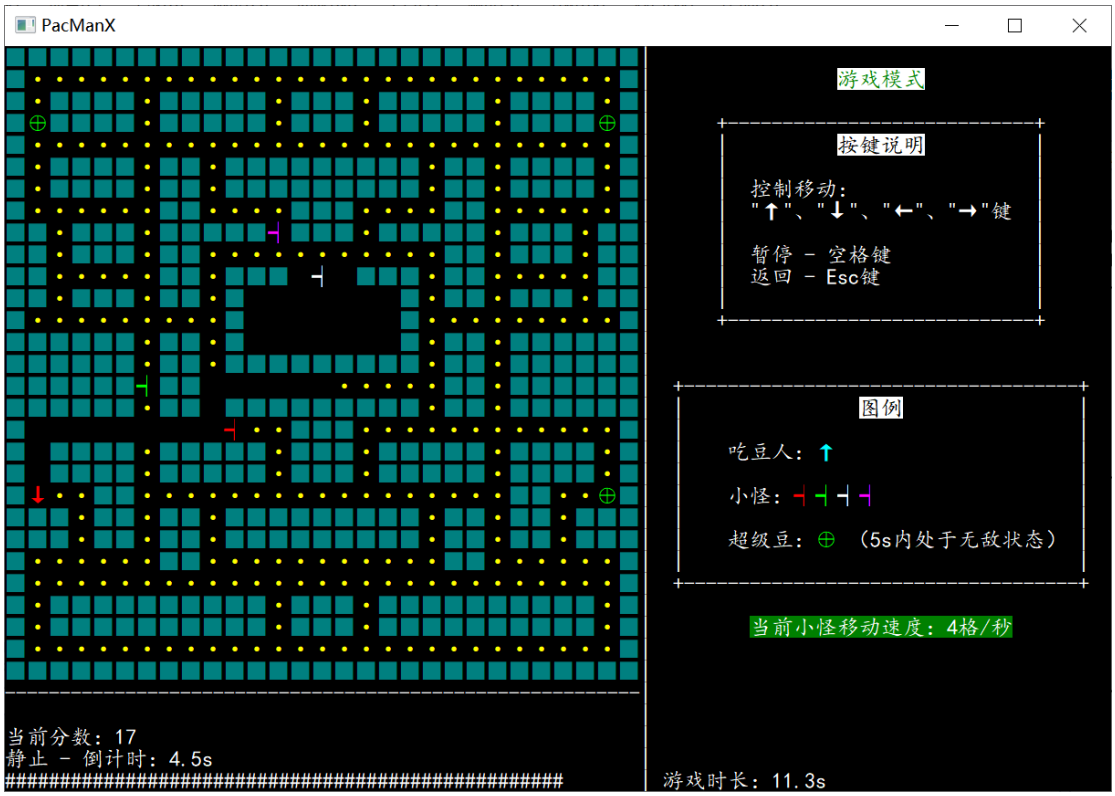
首先，这是初始界面，可上下键选择。(标题为“Pacman”，自己使用'*'目测绘制而成)



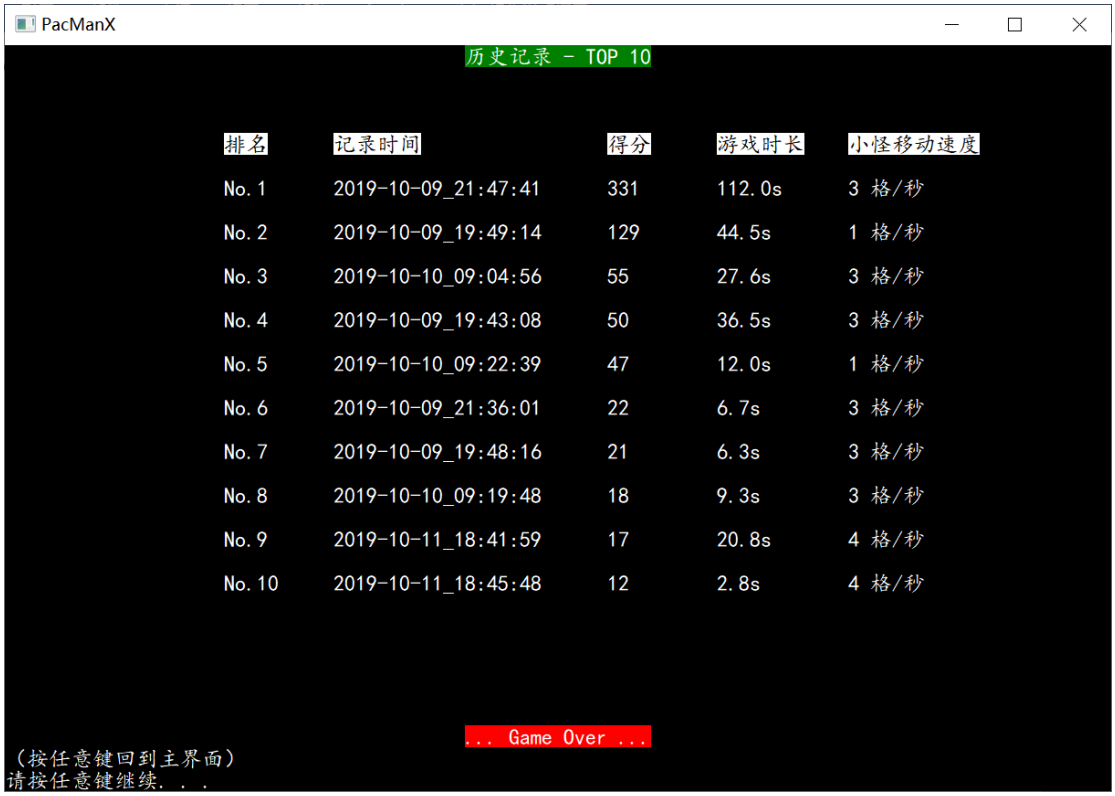
选择“开始游戏”后，进入难度选择界面。难度有三个固定模式：简单、普通、困难模式。分别对应自定义模式中小怪移动速度 1、3、5 的层次。



回车确认后，即可进入正式的游戏界面。游戏界面，右侧是一些帮助信息，下方显示当前分数。吃到超级豆子后，吃点人会彩色闪烁，下方出现倒计时和进度条。

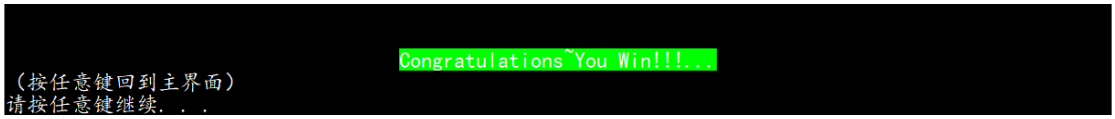


游戏结局分为胜利（吃完所有豆子）和失败（撞上小怪），两种结局下都会记录下本局游戏，并写入 game.record（如果可以进入分数 TOP 10 的话），之后可进行回放查看。

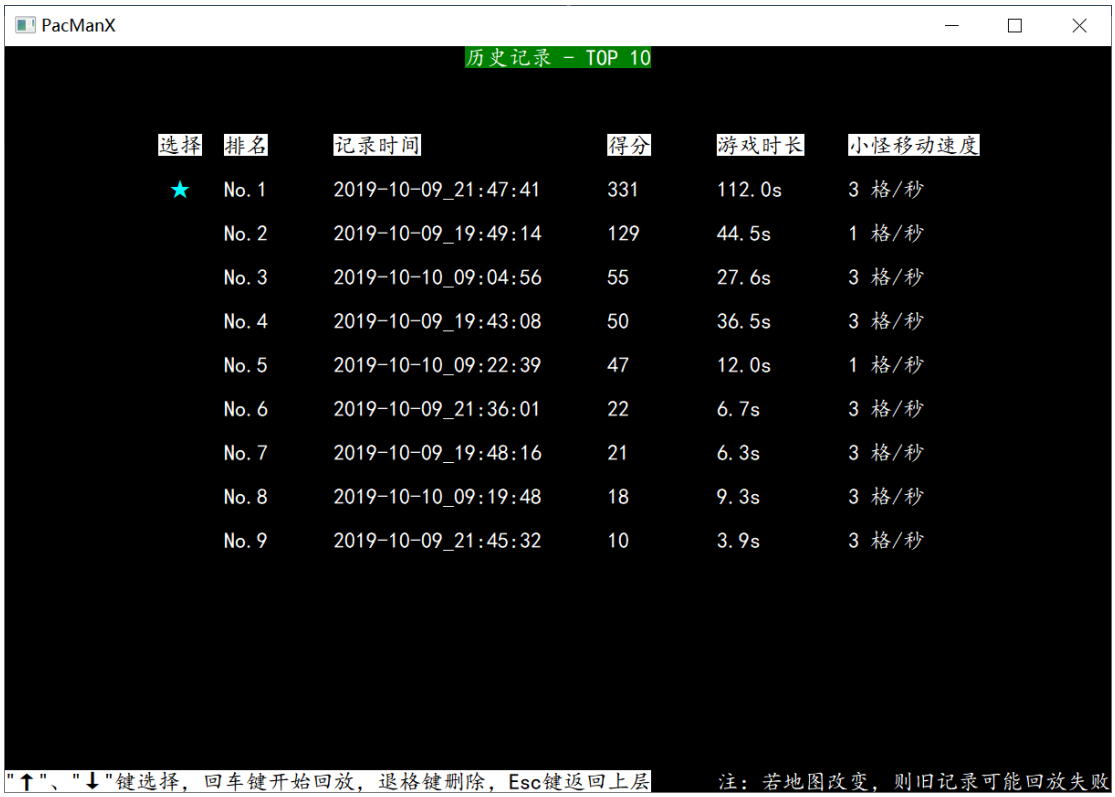


（此为失败结局的展示界面，展示出 TOP 10 分数排行榜）

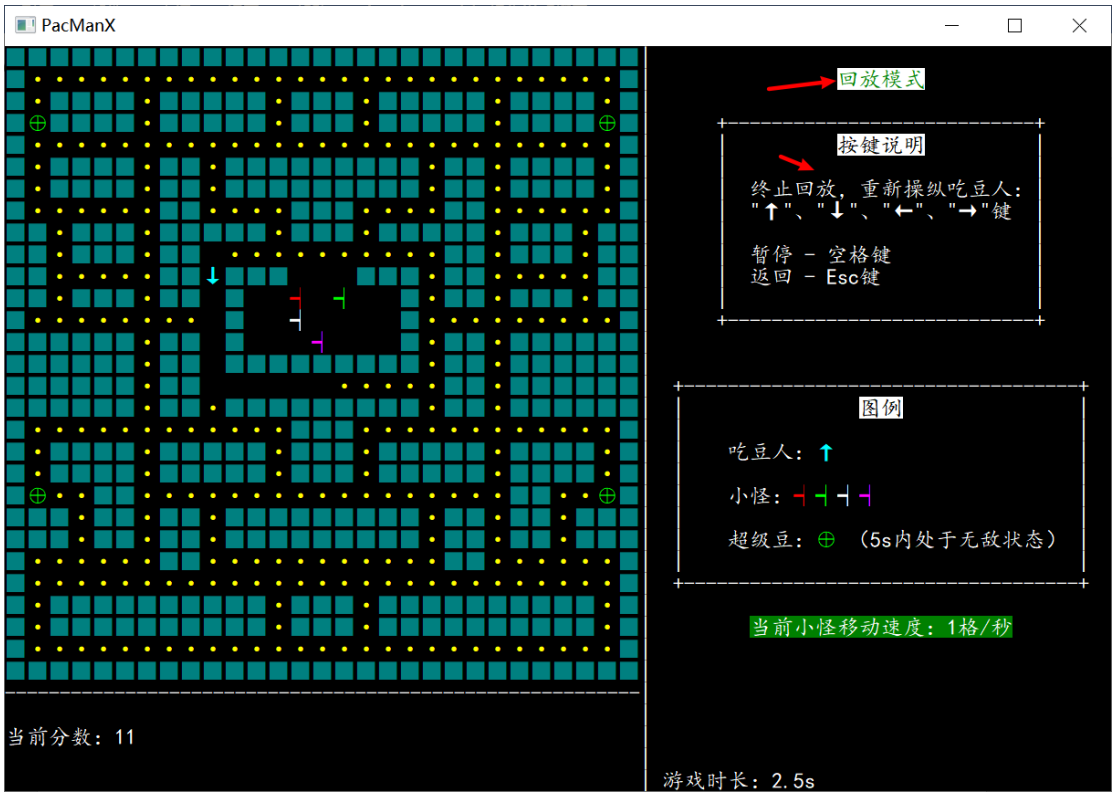
胜利结局与此失败类似，仅仅只是界面下方的色块不一样，如下：



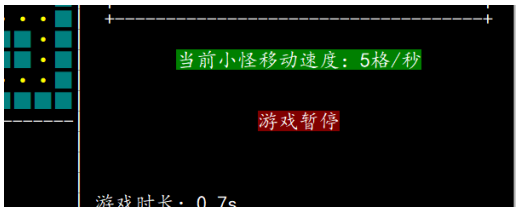
若选择“历史记录”，则可查看 game.record 文件中保留的历史记录信息。其中，对每条记录都可以按下回车键进行回放，或者按下退格键删除。（历史记录最多可保留分数排名前十的记录，下图中因为截图时只进行了 9 局游戏，所以只有 9 条）

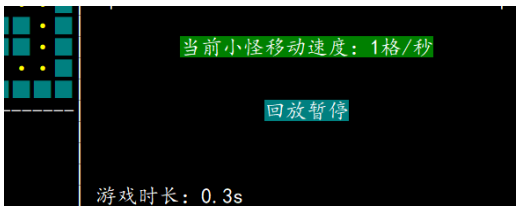


进入回放后，界面与正常游戏模式的界面稍有差异：左侧的一些提示信息变更为回放模式下的提示（下图中红色箭头处的两处）。而在回放模式下，只要按下方向键，就可以在回放到位置，继续操纵吃豆人，进入正常的游戏模式，这个机制有利于玩家在对某一局游戏的失败不满意时，可以在回放的时候随时弥补当初的遗憾。



在游戏模式和回放模式下，随时可按下 Esc 键返回游戏主界面。也可按下空格键，暂停游戏/回放，两种模式下，暂停提示稍有不同，都是在右下角出现提示色块，如下图。





四、代码实现中值得一提的地方

1）任意位置输出及颜色控制

参考网络上一些已有的控制台小游戏，有两个很基本的功能：在指定位置输出文本，以及输出一段指定颜色（包括前景色和背景色）的文本。所以，利用 Windows.h 中的相关 API 将以上两个功能，封装成了以下两个函数，并在之后多次使用：（位于 win_tools.cpp 文件中）

①设置输出光标到指定位置。

```
3    //设置光标位置
4    void Goto_XY(const int x, const int y)
5    {
6        COORD position;
7        position.X = x;
8        position.Y = y;
9        SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), position);
10   }
```

②设置文本颜色

```
31      //设置文本颜色
32      void SetColor(int colorID)
33      {
34          SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), colorID);
35      }
```

2）按键识别

按键识别的功能，使用到的函数主要是_kbhit()函数和_getch()的组合。每个时钟周期内，先利用非阻塞的_kbhit()来判断是否有按键按下，若有，则继续调用不带输入缓存的_getch()函数，识别出具体的是哪个按键。（为何选用这种方式？原因可见后文“遇到的问题及解决方案”部分）

3）游戏主循环代码的复用性增强

正常游戏模式下的游戏逻辑的实现代码包含在了 Game::loop()函数中。这个函数完成的工作就是不断地在每个时钟周期，去执行小怪移动、按键识别及响应、UI 信息更新、游戏结束判断等工作。在这个函数中，不需要做初始化工作，一些游戏相关参数（比如小怪移动速度、地图的生成、时间变量的重置等等工作）的初始化设置工作放在了 loop 函数外，也就是说，loop 函数仅仅只是按照当前的参数设置，将游戏的逻辑跑起来，完成决定小怪的移动，响应输出等等的工作。这样做的好处是极大地提高了 loop 函数代码的复用性，也就是在回放模式中（即 Game::play()函数中），只要识别到按下了方向键，立即调用 loop 函数，就可以做到，在回放到的位置的情形下，继续开始正常的游戏模式，玩家继续操纵吃豆人完成游戏。

4）小怪移动策略

这里在 Map 类中，已实现了 findDir() 方法，所以每一个小怪其实都是可以准确地知道下一步该怎么走，才可以用最短路在地图上走到玩家的位置。而经过自身的体验发现，如果小怪每次都按正确的方向移动，那么玩家将很快被小怪追上（尤其是在速度为 5 格/秒的难度下）。所以这里额外加入了一个随机机制（以启动程序的时间做随机数种子），让小怪不是每次都按正确的方向前进。但总体来说，正确走向的概率也固定不变，比如在小怪移动速度慢和快的时候，应该要在低速度的时候走对的概率大一些。

所以，最终设计了小怪往正确的移动方向移动的概率为 1/(1+speed)，剩余概率下则随机移动。（小怪移动速度为 speed 格/秒）

同时，随机选一个方向也存在一定的问题，随机有可能选到朝反方向移动，这个时候小怪的移动效果则不太美观，可能往复来回移动，所以这里也考虑这一点，设计了在随机选择时不能朝反向移动。

5）UI 设计的一点技巧

程序中，控制台下的 UI 设计无他，就是存粹靠着 X、Y 两个坐标的定位，然后输出。这里在构建布局 UI 时，发现了一些对齐的技巧。很多时候要是界面上的每一个元素的摆放位置都采用硬编码的方式来设定 XY 坐标，则非常不便于修改、调整，所以大部分情况下可以先定义好对齐线，即一个坐标值，相当于是有了一个参考位置。比如，下图定义的一系列值，就是用于实现“历史记录”展示界面中左对齐表格效果。

```
95      //对齐
96      int left_margin_No = WINDOWS_SIZE_X / 2 - 30;
97      int left_margin_name = left_margin_No + 10;
98      int left_margin_score = left_margin_name + 25;
99      int left_margin_time = left_margin_score + 10;
100     int left_margin_speed = left_margin_time + 12;
```

五、遇到的问题及解决方案

起初，识别键盘按键使用了 Windows.h 中提供的 GetKeyState()函数，这个函数识别调用时刻键盘某个键的状态信息（按下/抬起）。可是使用时发现，时钟周期设置极易影响操作体验。比如，周期太短，按一次按键，可能识别到了多次，执行了多次操作；周期太长，也就是识别时间间隔太大，则按一次按键，可能识别不到按下了按键。

最终，参照一个网上已有的控制台小游戏 贪吃蛇 的识别按键方法，最终决定了使用_kbhit()和_getch()的组合，自己测试发现操作效果更加不错。

全文结束，感谢阅读！Thanks For your time...

—— 课程设计一　姓名：唐金麟　学号：171860672