



同濟大學
TONGJI UNIVERSITY

高级语言程序设计课程大作业报告

题目：贪吃蛇小游戏

班 级：工科试验班（信息类）11 班

学 号：1953610

姓 名：孙久昌

2020 年 5 月 14 日

一. 设计说明与功能描述



Wutahu 是一个基于 Windows 的简单贪吃蛇小游戏。

在主界面上，玩家可以设置自己的名称、查看游戏历史记录、选择三个游戏模式之一开始游戏，或者选择等待加入他人发起的游戏。



可以查看详细的历史记录，查找或删除指定的记录。可以让 AI 或在线玩家参与游戏。玩家若选择等待加入他人的游戏，其他玩家将可从在线玩家列表看到该玩家的名称。



游戏过程中，玩家可以随时退出并观看其他玩家进行游戏，或者离开以开始新游戏。

二. 设计实现与分析

1. Wutahu 的主要架构是基于 C++ 语言和面向对象思想设计的。

可以认为对于贪吃蛇这个游戏场景来说，环境（赛场）和玩家（所控制的蛇）分别是独立的个体，它们在不同时刻表现出特定的状态。本项目对游戏过程的构建基于对这两者的抽象。

本项目部分地基于模型和视图分离的思路。每个视图对应唯一指定的模型。相反，同一模型可以动态地更换视图。这一特性增强了架构的可拓展性。

1) `RecordManager` 类是历史记录模型，同时也实际管理游戏历史记录的查改存取。每次游戏结束后 `ArenaModel` 将记录存入其中。

`RecordView` 类是历史记录的视图。

2) `ArenaModel` 是游戏过程的模型。对应 `setHost(true)` 后其会根据游戏规则决定玩家的状态。对应 `setHost(false)` 后其会根据收到的在线消息决定玩家的状态。`ArenaView` 是游戏过程的视图。

3) `AddView` 是添加玩家的视图。

4) `PlayerController` 是玩家的模型。不同玩家对应三种 `PlayerType` 类型（`local`, `ai`, `remote`）。

5) 在线游戏中的消息收发由 `ArenaSyncProvider` 类提供。每个 `ArenaModel` 保有该类的一个实例。同时 `PlayerController` 也通过对其的调用传输网络信息。







6) AI 玩家的寻路策略由 `NavigatorTools` 提供。

7) 所有的文本消息内容由 `MsgViewProvider` 提供。




8) `GameInstance` 类是游戏实例的模型，同时也是主界面的视图，处理所有主界面上用户操作的处理事务。对应唯一的 `ArenaModel`、`AddView`、`RecordManager` 实例。

2. 游戏地图简单地固定为 16×9 方格的大小，玩家速度简单地固定为 2 格/秒。这样的设定可以提供较好的游戏节奏。

不同位置上的食物被认为是游戏场景（`ArenaModel`）对于玩家（`PlayerController`）表现出的一种反馈。同时，玩家的位置也是场景相关的一种状态。它们不是独立的对象，而属于场景本身。

不同位置上的食物保存在 `int map[][]` 数组中。其中，`0` 代表该位置为空。`3~8` 分别代表      。某个位置被蛇占据也是一种特殊的食物类型。`1` 代表该位置有蛇，`2` 代表该位置有死蛇（仅适用进阶版和高级版模式）。但蛇占据的位置不一定为 `1`。如果蛇身的某个位置被另一条蛇撞击，该位置可以形成一个空洞，允许其他蛇穿过，直到原位置的蛇离开时空洞失效。

玩家位置通过对应于蛇的每一段所占据方格坐标的链表保存。

对于本项目来说，状态更新对应于玩家移动到新位置上的变化。玩家吃到每种食物对应扣分或加分。 对应扣 5 分， 对应加 5 分， 对应加 10 分，以此类推。如果得分扣尽，玩家将不能继续游戏。低于 0 分的得分不会被保存进游戏记录。如果超过 10 秒没有玩家吃到食物，场景会生成新食物替换旧食物。

每个游戏场景对应唯一的本地玩家。为保证游戏性，`AddView` 只允许五名玩家加入同一场游戏，场景本身允许不限量的 AI 或在线玩家加入。AI 的名称在 10 个模板中随机生成。

3. 本项目内部文本均处理为宽字符类型 `wchar_t`，而包括网络连接和历史记录保存等过程向外传递的文本流则全部以 UTF-8 编码，以此保证良好的 Unicode 支持。

游戏历史记录包括日期、玩家名、游戏时长、分数、游戏模式。默认保存在用户文档目录下的 `wutahu.csv` 文件中。游戏开始前玩家可以设定自己的名称，但游戏结束后保存在记录中的名称就会确定下来。可以任意查找某个玩家的记录或删除某条记录。

4. 联机游戏通过状态更新信息的收发进行状态同步。

基于题目提供的网络框架，玩家如果选择等待加入游戏，这时会建立一个名称格式为 `WthW/玩家名` 的房间（等待参与）。查找房间列表中符合该格式的房间名即可得到本游戏在线等待的玩家。

开始游戏时，发起游戏的一方将向其他非本地玩家在对应 `WthW` 房间发送两条分别以 `:` 和 `/` 开头的信息。前者为游戏发起者的名称，后者为在本场游戏中该参与者将使用的与其他玩家不冲突的新名。

游戏开始后，所有参与者将根据得到的游戏发起者的名称，加入名称格式为 `WthG/玩家名` 的房间（进行游戏）。各参与者有新的操作时，这个操作会立刻被发给作为房主的游戏发起者。房主收集各参与者发来的信息，同时游戏逻辑的判断只由房主进行。当游戏状态更新时，其会把有变化的内容发送给各参与者。各参与者接收最新的游戏状态，根据结果将变化呈现出来。

对于 `ArenaModel` 场景类来说，这个过程是通过对 `ArenaSyncProvider` 的函数调用完成的。这些函数调用形如 `tellScore(int score, const wchar_t* name);` 等。

而在 `ArenaSyncProvider` 内部，每个待发送或已接收的操作先会被封装成 `message` 结构体缓存下来，直到 `tellUpdated()` 或 `readUpdated()` 被调用。

```
struct message {
    int order;           // 发送顺序
    uint64_t time = 0;  // 接收时间
    char sign = '\0';   // 表示消息类型的符号
    char name[RecordManager::max_name_bytes]{}; // 和本消息有关的玩家名称
    int typeData = 0;   // 和本消息有关的数值（见表）
    point pos = {};     // 和本消息有关的位置坐标
};
```

意义	sign	typeData 的说明
在线	~	（不适用）
当前方向	:	0 上 1 下 2 左 3 右
新食物	+	食物类型，即场景 <code>map[1][1]</code> 储存的值。
蛇长	&	蛇长。
蛇的位置	\$	蛇节序数。如 0 表示该位置是蛇的第 0 节。
分值	'	分值。
示亡	=	死亡原因。
传输结束	.	已发消息数。

`tellUpdated()` 会将缓存的消息全部发送出去，附带发出的消息数用来核实完整性。

`readUpdated()` 则会试图按照收到的数量检查消息是否接收完整，以备读取。

实际传输的信息是文本串的形式，以此获得更好的容错性。

```
bool _pack(char(*)[msg_length], int& order);
void _unpack(char*, uint64_t time, bool forward);
```

如上的 `_pack` 函数读取缓存的 `message`，按


“(order):(sign),(typeData),(pos.x),(pos.y)/(name)” 的格式编码，并把多条 `message` 以 `\x1e` 作为分隔符组合起来发送。`_unpack` 函数相反从文本串解析出 `message`。

通过这样的处理，消息收发的频率得到降低，并且可保证较大量数据的完整，不同游戏端之间的游戏状态得到了同步。



5. AI 玩家可以安全地吃到食物作为其目标。


```
static bool Stepoff(point& p, char(*map)[16][9],  
                  int(*towardSafetyRank)[4], std::list<point>& steps);
```

`NavigatorTools` 只有如上所示的一个公开函数。给定地图和一个起始点，将四个行进方向按危险程度排序，并尝试找出一条安全通往食物的最近路线，找到则返回 `true`。

`NavigatorTools` 在内部会从给定的点出发，迭代缩小范围搜索四个方向，并根据空位置数量、碰到蛇身的数量、墙的数量和  的数量得到四个方向危险程度的评估值；与此同时通过部分的危险值和移动步数选择一个食物位置，将吃到该食物的路线链表补充完整。

`PlayerController` 得到结果后，判断如果有食物路线就转到该路线的方向，否则按 $\frac{1}{64}$ 概率保持原方向，剩余概率选择最安全的方向。

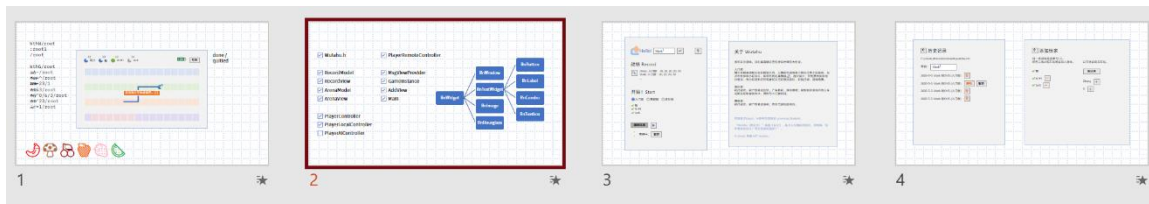
另外，`PlayerController` 会额外判断选择的方向是否即将遇到危险，如撞墙、撞上蛇身等。如果确实会遇到危险，其会按照不接受  → 可以吃到  但不接受死蛇段 → 可以朝向死蛇段的优先级和得到的方向安全顺序更换方向，以此躲避直接的危险。

这个评估过程能较快地得到较好的结果。在入门版一条蛇的测试中，AI 可以放弃危险位置上的食物、盘曲自己的身体获得空间。在进阶版四条蛇的测试中，AI 玩家平均存活了 3 分 28 秒，取得了 105 的分值，即相当于平均每 2.5 秒就能吃到一个 。

6. 为了实现友好的用户界面，

在本项目制作过程中，从头开始设计了完全基于 EasyX 绘图库的简单界面控件包 `Rnvnbtn`，包括按钮、单选框、等待提示、图片标签、文本标签、文本框和窗口组件。这部分与主题关系不大，不加赘述。

三. 讨论和小结



程序设计是综合性的工作。要高质量、高效率地完成一个好的程序设计究竟需要各方面的能力。