**实验名称： X86汇编实现双人贪吃蛇游戏**

**学 院：** 睿信书院

**专 业：** 计算机科学与技术

**班 级：** 07111802、07111803

**姓 名：** 田佳析(02)、徐正斐(03)、

宋尚儒(03)、赵旭(03)、陈鸿韬(03)

**指导教师：** 李元章

**目录**

[1 实验目的 3](#_Toc75525700)

[2 游戏功能简介 3](#_Toc75525701)

[2.1 游戏玩家操作 3](#_Toc75525702)

[2.2 输赢规则 3](#_Toc75525703)

[2.3 界面及道具说明 3](#_Toc75525704)

[3 程序架构 5](#_Toc75525705)

[3.1 逻辑层 5](#_Toc75525706)

[3.2 动画层 5](#_Toc75525707)

[3.3 显示层 6](#_Toc75525708)

[4 程序实现 6](#_Toc75525709)

[4.1 数据结构 6](#_Toc75525710)

[4.1.1 物体队列 draw\_struct queue 6](#_Toc75525711)

[4.1.2 状态地图 map 7](#_Toc75525712)

[4.1.3 蛇身数组 point array 7](#_Toc75525713)

[4.2 显示层实现 7](#_Toc75525714)

[4.2.1 定时器线程 8](#_Toc75525715)

[4.2.1 画面显示线程 8](#_Toc75525716)

[4.3 动画层实现 9](#_Toc75525717)

[4.3.1 从逻辑层获取物体队列 9](#_Toc75525718)

[4.3.2 根据物体队列生成动画帧 10](#_Toc75525719)

[4.4 逻辑层实现 12](#_Toc75525720)

[4.4.1 生成物体队列 12](#_Toc75525721)

[4.4.2 判断游戏胜负情况 13](#_Toc75525722)

[4.4.3 道具及地形实现 13](#_Toc75525723)

[5 难点及解决方案 14](#_Toc75525724)

[5.1 定时器问题 14](#_Toc75525725)

# 1 实验目的

采用32位汇编语言编写一个游戏程序，如俄罗斯方块、贪吃蛇、扫雷、简单射击类游戏等，题目自拟，要求具备一定难度。

# 2 游戏功能简介

本组使用X86汇编实现了经典的贪吃蛇游戏，并且在传统贪吃蛇游戏的基础上进行了一些创新，引入双人对战模式以及一些道具，提高了游戏的趣味性也增加了游戏实现逻辑的复杂度。不同于控制台程序，本游戏通过使用WIN32的API和GDI图形框架实现了图形界面，使用双缓冲等技术优化游戏界面显示的流畅度。

## 2.1 游戏玩家操作

本实验程序为一款双人对战型贪吃蛇游戏。游戏包含红蓝两条蛇，分别受两个玩家的控制。当玩家无操作时，蛇会沿着上一次的移动方向进行移动，玩家双方分别通过键盘的“上”、“下”、“左”、“右”方向键和“w”、“a”、“s”、“d”来控制蛇的上下左右方向改变。

## 2.2 输赢规则

当一方蛇头撞到对方蛇身时，该方蛇身会不断减短。若自身蛇身长度变为0，则死亡。游戏的输赢规则即是看哪一方先死亡，对方先死亡，我方就获胜，反之相同。因此，想要获胜，就要尽量让对方蛇头撞到墙或自己的身体，同时不能让自己撞到。

## 2.3 界面及道具说明

界面内容主要包括分数记录、墙体、果实、草地及功能性道具。界面左、右上角分别为双方分数记录，可以使玩家清晰地看到自己当前分数。

墙体：墙体位置固定，蛇头如果撞到墙块，蛇身就会不断减短，直到移动蛇头使之不再撞击墙块。果实位置随机产生且总数保持不变，蛇头吃到果实后，蛇身会增长一格，自身分数也会增长。

草地：草地位置固定，蛇身可与草地重叠，当重叠时，草地会遮挡蛇身，从而干扰玩家判断，增加游戏难度。

功能道具：功能性道具有三类：小鸟、蘑菇和时钟，随机出现。蛇吃到小鸟，会使对方出现眩晕效果，眩晕效果为按键后方向反向改变，即左变右、上变下等；吃到蘑菇会使自身变粗；吃到时钟会使双方蛇的速度均增快。蛇处于不同状态时，会产生不同动画。当吃到果实时，会产生红心动画；当死亡时，会产生心碎动画；当拥有眩晕效果时，会产生眩晕动画。

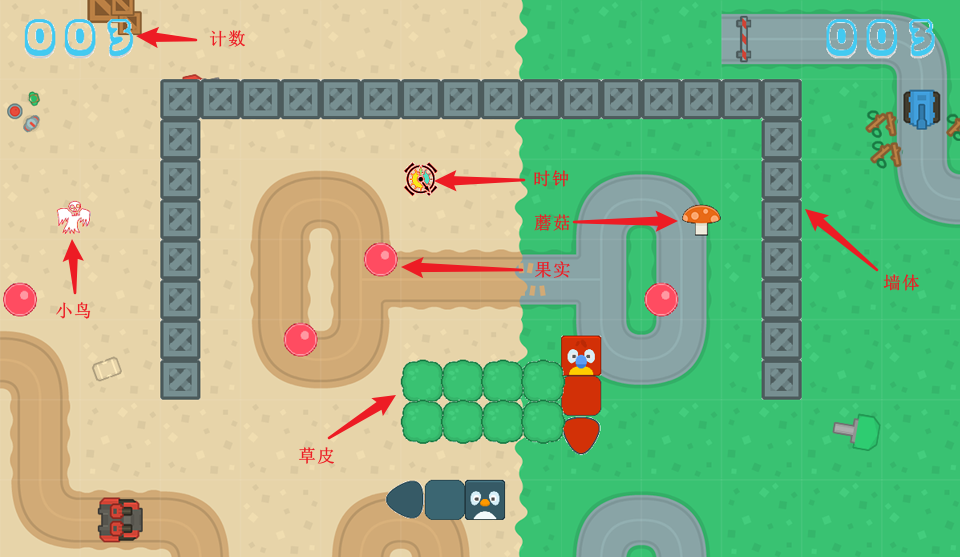
界面如下：

图2-1：游戏界面展示

# 3 程序架构

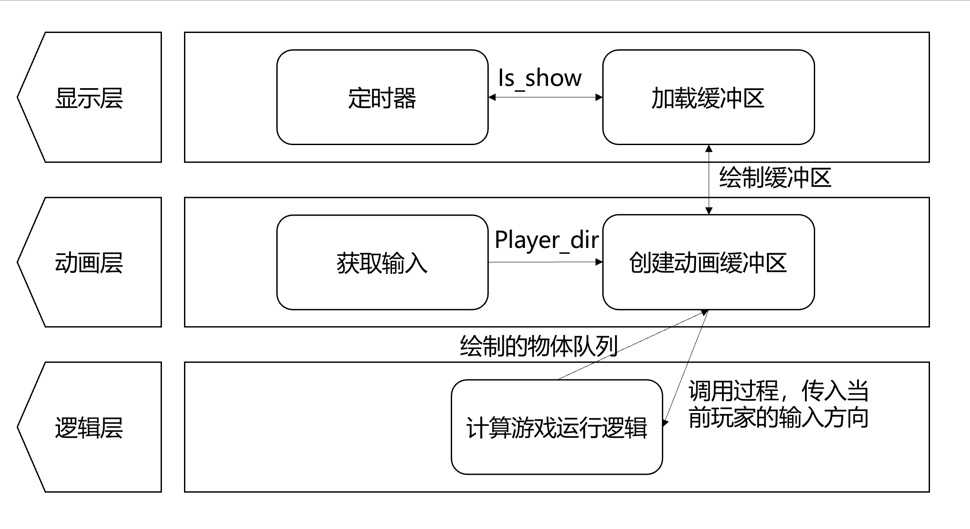


图3-1：程序架构

程序设计采用分层结构，自下而上包括逻辑层、动画层、显示层，如图3-1所示。每个层次的功能分配如下

## 3.1 逻辑层

逻辑层是游戏的后端部分，实现的主要功能包括：碰撞检测、长度计数、道具生成、计算游戏状态。每一次动画显示完毕后，动画缓冲帧的生成函数调用后端逻辑层函数，计算得到新的游戏状态。游戏状态不是保存在数组中而是保存在一个物体队列中，物体队列在后面具体介绍。

## 3.2 动画层

动画层用于生成动画帧，用于显示层刷新界面。从后端获取到一个物体队列后，通过物体队列中每个物体的信息绘制出对应的图像。物体队列中的全部物体绘制完毕后，将整个一帧画面放入画面缓冲区中，显示在窗口上。此外，动画层还负责通过窗口消息接收用户输入的操作。

## 3.3 显示层

显示层的功能就是将画面缓冲区的图像显示在窗口上，设置一个定时器线程，每个一段时间将is\_show置1 。显示画面的线程一直处于阻塞状态直到is\_show变为1，在显示完毕后，将is\_show变为0，继续阻塞等待下一次的绘制。

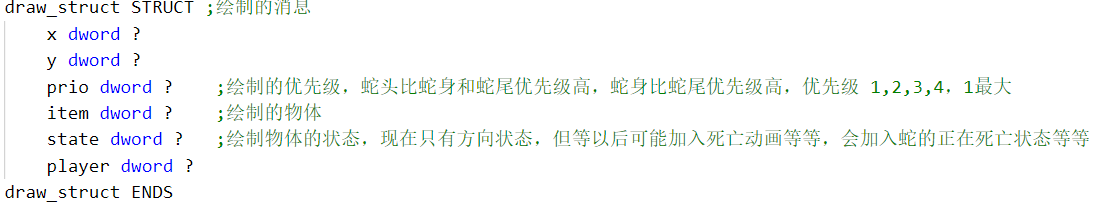
# 4 程序实现

## 4.1 数据结构

在分层的程序架构下，为了保证程序结构的清晰，程序实现的高效以及各层之间的低耦合性，程序设计了一些自定义数据结构，如：物体队列、游戏状态地图map、蛇身数组等，具体介绍如下：

### 4.1.1 物体队列 draw\_struct queue

物体队列是用于逻辑层和动画层通信的数据结构，物体队列中的元素定义如下：



说明数据结构中的变量含义：

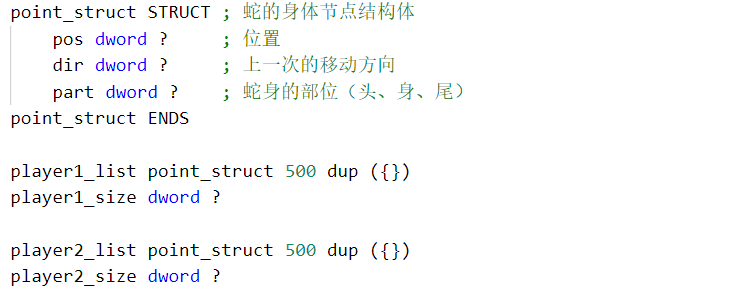
1. x为物体在数组中的x坐标
2. y为物体在数组中的y坐标
3. prio为物体绘制的优先级
4. item为该数据表示的物体id
5. state为绘制物体的状态。不同物体状态值的含义不同，比如，蛇物体的状态表示为蛇的绘制方向，表情物体不同的值会显示不同的表情等等。
6. player为该物体属于哪一个玩家，因为绘制的物体中有玩家1，2的蛇部分还有玩家1，2碰到物体的显示的表情，需要保存蛇和表情是属于玩家1，还是玩家2 。

### 4.1.2 状态地图 map



一维数组map保存地图，记录地图上每个块的类型，块的类型包括不同的道具、墙、草丛等，可以通过二重循环的方式进行遍历获取地图上道具的信息，如果有新生成的道具也会在该数据结构中做出相应修改。

### 4.1.3 蛇身数组 point array



蛇身节点（point）：包含三个属性的结构体，三个属性分别代表该节点的位置、方向以及类别。

1. 位置：节点在map中对应的一维索引值，因为蛇的节点存在与道具地点重合的可能性，所以单独在该数据结构中对蛇的节点位置加以保存；
2. 方向：指该节点在下一计算中前进的方向，蛇的每个节点均可能存在不同的方向，玩家只会控制蛇头节点的方向，蛇身节点的方向会自动计算得出；
3. 类型：标记该蛇节点是蛇头、蛇身还是蛇尾。

两个point结构的数组保存两个玩家的蛇的完整结构。

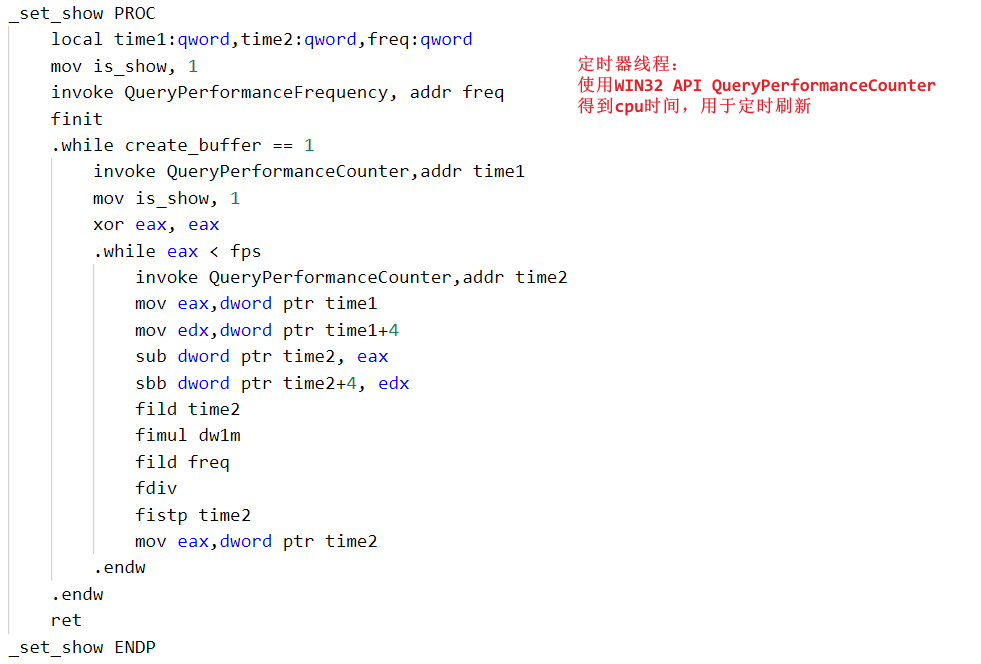
## 4.2 显示层实现

显示层包含两个线程，一个定时器线程，一个显示画面的线程。

### 4.2.1 定时器线程

由于要达到流畅的动画效果需要较高的帧率。我们设置的绘制帧率为5ms刷新一次，这就需要定时器具有较高的精度。所以使用win32函数QueryPerformanceCounter得到cpu内部时钟的tick次数，获取两个时刻的tick数差，除以cpu的时钟频率就得到了两个时刻的时间差，由于QueryPerformanceCounter函数得到tick数为64位，无法使用常规的寄存器计算乘除，所以将其塞入浮点寄存器进行计算。

在线程循环开始的时候将is\_show置为1，之后阻塞线程进行计时。通过循环不断获取当前时刻的tick数减去开始的tick数，直到时间差大于刷新时间，此时退出阻塞进行下一个循环。



### 4.2.1 画面显示线程

线程不断循环从画面缓冲区取出画面显示。使用死循环 buffer\_cnt==0 || (is\_show) 阻塞等待画面缓冲区中存在画面和 is\_show为1 。之后获取设备环境也就是窗口，使用 BitBit 函数将缓冲区的画面拷贝到窗口上，缓冲区计数减一，将is\_show置为0，进行下一次循环。

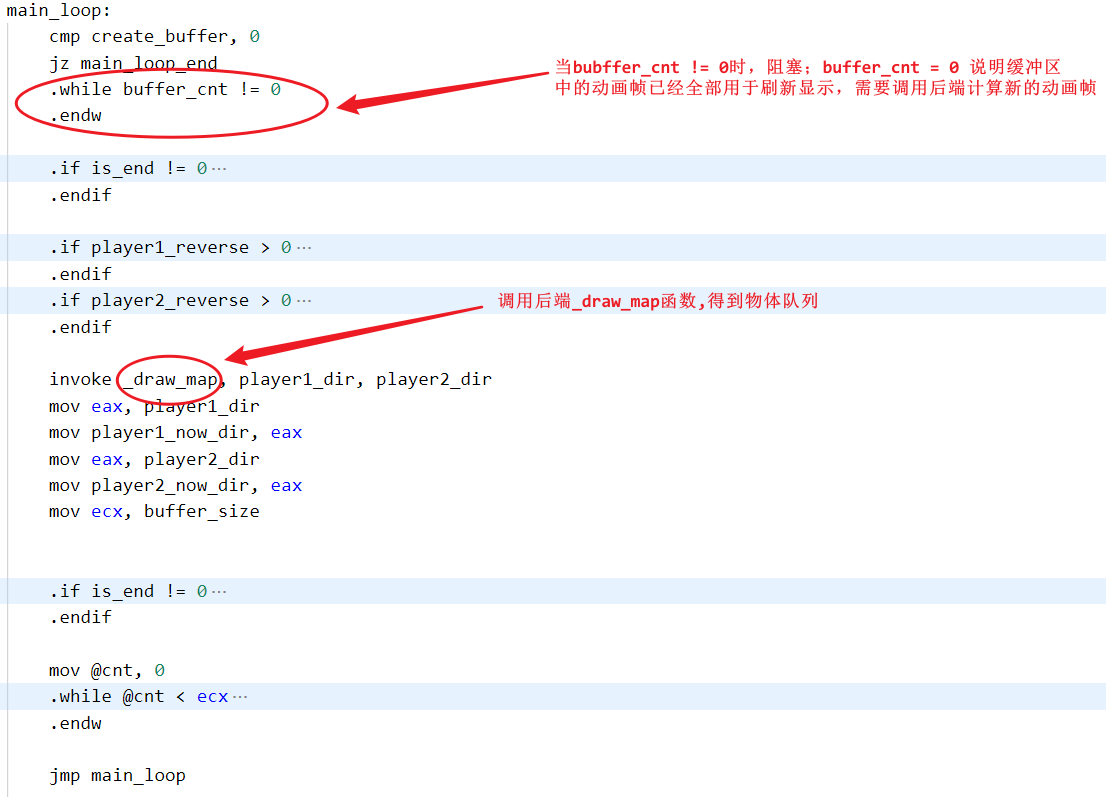
这里采用了双缓冲技术，每次从后端获取的新的游戏状态中蛇都已经向前移动一个格子，如果按照每次移动一格的帧率进行界面刷新，则蛇的移动会变得不连续不平滑，通过多缓冲技术，在每获取到一次蛇和其他道具的位置（物体队列）后，计算出从上一位置到当前位置的多帧连续画面，将其缓存下来，每次刷新一帧。这样就能提高界面显示的帧率，使得动画变得平滑。



## 4.3 动画层实现

### 4.3.1 从逻辑层获取物体队列

由上面的介绍，可以知道，游戏的运行就是一个循环的过程，前端调用后端，画面显示完后继续调用后端。具体实现如下，在main\_loop 循环中，除非游戏结束，create\_buffer为0，否则一直循环进行游戏。使用死循环buffer\_cnt != 0阻塞线程直到所有的画面被取出。之后调用\_draw\_map函数，该函数就是后端过程，得到物体队列。之后使用一个循环绘制动画帧。绘制完毕后进入下一个循环等待所有画面被取出。



### 4.3.2 根据物体队列生成动画帧

循环绘制动画帧的程序逻辑如下：

首先拷贝背景到缓冲帧上，然后进行进行一个循环，原因是物体优先级的存在，低优先级的物体需要先绘制，然后高优先级的物体再绘制覆盖低优先级的物体。每一次循环时，绘制对应循环值的优先级的物体，这样就实现了不同优先级物体的绘制。绘制每个物体时会调用 \_draw\_item函数，这个函数的输入是要绘制的物体的数据和当前帧画面的序号。函数判断要绘制的物体是什么，根据物体数据中的item字段判断要绘制的物体，根据不同的绘制物体调用不同的绘制函数。

由于绘制函数较多，下面就介绍绘制蛇头、蛇吃到不同道具显示的表情的实现方法：

**绘制表情：**

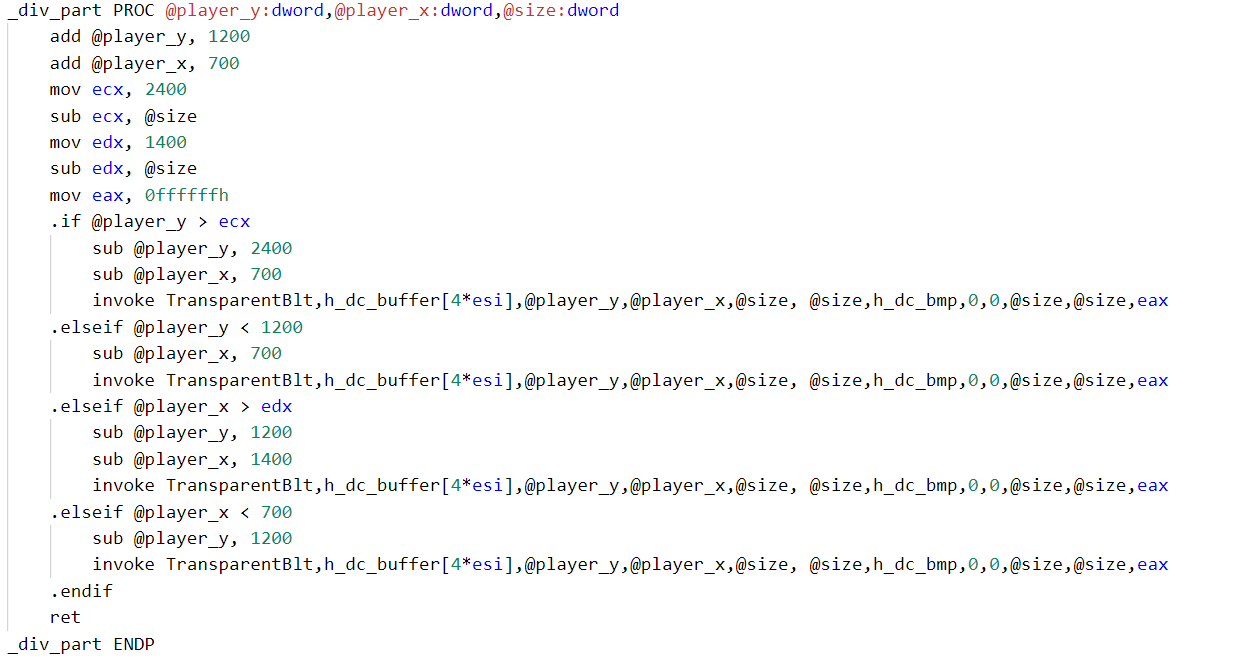
首先判断这个表情是属于玩家1还是玩家2，然后初始化显示在窗口中的大小和位置，根据输入的物体状态判断要显示哪个表情。使用StertchBlt函数将对应部分缩放到一个临时的图片区，然后使用 TransparentBlt 函数将白色背景部分变透明绘制在画面帧中。

**绘制蛇头：**

和表情绘制步骤一样，初始化蛇头在窗口中的位置和大小，但这里存在一个额外的步骤，由于要绘制蛇头移动的过程，不同的动画帧中蛇头的位置是不同的，所以需要判断蛇头的方向，并且将蛇头从初始位置移动帧序号乘以移动速度的距离。在得到蛇头的大小和位置后，就可以和绘制表情的步骤一样将蛇头绘制在动画帧中了。

但是还需要调用\_div\_part函数，这个函数的作用为实现蛇在进入边界的时候的绘制。由于我们的贪吃蛇在碰到边界时并不会死亡，而是从边界的另一边出现。这就出现了一个问题，进入边界时在边界外的部分实际上得绘制在另一边边界，显然需要对蛇进行两次绘制。所以对于蛇进入边界的绘制，我们放入了一个函数进行处理。

\_div\_part函数如下：



首先获取当前蛇头的位置，根据位置判断蛇头是否绘制出了边界，如果出了边界，就需要在另一边绘制超出的部分。蛇头位置有可能超出左或者上边界，这样x或y就为负数，由于if不能判断负数，所以对蛇进入边界的判断采用加上一个大数进行处理。当蛇头超出右边界时，蛇头的x，y值减去大数，然后y值减去窗口的长度，让蛇头绘制在左边界。下面几个分支步骤类似。分支结束后，就完成了超出边界的绘制步骤。

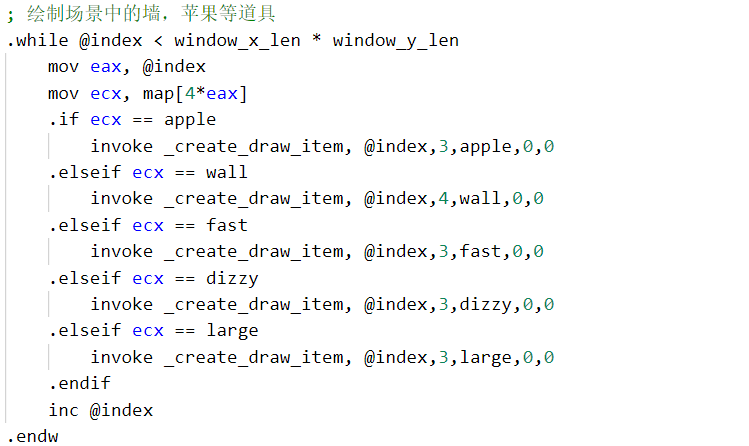
## 4.4 逻辑层实现

### 4.4.1 生成物体队列

该部分主要由函数\_draw\_map完成

1. **绘制蛇身**

调用\_draw\_snake函数完成，因为蛇是运动的，对蛇身需要计算其运动轨迹，该函数主要主要思路为：

1. 获取蛇头地址，由于蛇头地址设置为point数组的第一位，只需要获取该元素即可获取蛇头节点的全部信息，同时获取此时玩家控制的移动方向（该变量作为参数传入），然后调用\_get\_nxt\_pos函数使得蛇头移动。
2. 判断蛇头是否碰到道具，采用不同特效
   1. 墙：碰到墙蛇头位置不会变化，视为蛇此时长度减一，在蛇节点列表中删去原蛇尾节点即可，同时如果蛇的长度降低到0，需要对胜负进行判断。
   2. 苹果：蛇吃苹果后长度加一，具体表现point数组节点数加一。记录原蛇尾节点位置，在完成蛇身移动的工作后，在此处添加蛇尾。
   3. 加速道具、眩晕道具等，具体道具的实现在4.4.3节说明
3. 移动蛇身，完成绘制工作，具体操作是所有蛇节点沿着自身方向移动一位，方向变为前一节点的方向，这工作可以由\_get\_nxt\_pos完成，调用\_create\_draw\_item创建绘画对象。
4. **绘制道具**

较为简单，只需要遍历map数组即可完成，针对不同的道具类型，调用\_create\_draw\_item创建不同的绘画对象

### 4.4.2 判断游戏胜负情况

调用\_judge\_win函数完成，会根据蛇的碰撞情况判断游戏结束与胜利情况，修改全局变量is\_end，变量值分别对应

表4-1：游戏状态

|  |  |  |
| --- | --- | --- |
| 状态 | 是否结束 | 胜负情况 |
| 0 | 未结束 | 待定 |
| 1 | 结束 | 蛇头蛇身相撞，1号胜 |
| 2 | 结束 | 蛇头蛇身相撞，2号胜 |
| 3 | 结束 | 蛇头蛇头相撞，1号胜 |
| 4 | 结束 | 蛇头蛇头相撞，2号胜 |
| 5 | 结束 | 平局 |

首先需要获取两个蛇头的地址，获取方法与之前相同，如果蛇头位置相同说明蛇头发生碰撞，根据两者的长度判断较长者获胜，修改is\_end为3或4或5；如果位置不同，分别遍历两个point数组，判断其中节点有无与另一个蛇头位置重合，如果重合，说明发生碰撞，这时需要做进一步的判断，如果两者都发生碰撞，根据两者的长度判断较长者获胜，修改is\_end为1或2或5，如果只有甲方蛇头碰撞乙方蛇身，则乙方获胜，修改is\_end为1或2。如果以上情况均为发生，则没发生任何碰撞，游戏继续。

### 4.4.3 道具及地形实现

1. **道具实现**

加速道具：该道具的功能是使整个游戏的速率加快。snake.asm文件中有一个控制界面刷新速度的全局变量fps，当一方碰到加速道具，便修改此全局变量从而加快游戏速度。

变大道具：该道具的功能是使自己在接下来一段时间内体型变大。当一方碰到变大道具，向绘图队列中加入large\_eat消息，绘图时改变图块的大小。此道具不会使碰撞体积变大，所以其余逻辑功能的实现不变。

眩晕道具：该道具的功能是使对方眩晕20步。在眩晕状态下，系统识别到的操作和玩家实际的操作上下相反、左右相反。snake.asm文件中有表示剩余眩晕步数的全局变量player1\_reverse和player2\_reverse。当一方碰到眩晕道具，系统将对方的眩晕步数增加20，并且每次移动之后步数减一。每当接收到键盘输入，若对应玩家正在眩晕状态，直接对输入进行转换，其余移动操作不变。

以上道具的产生方式为随机出现在某空地，游戏开始时三种道具各一个，若有道具被吃，则随机再产生一个相同道具，但为了避免游戏速度过快，加速道具最多出现两次。

1. **地形实现**

草：该地形可以和其他物体共存在同一位置，并且绘图的优先级是最大的，从而起到隐蔽其他物体的效果。由于草的位置是在设计地图时确定好并且不会改变的，所以草不需要在绘图队列中出现，也不需要在map中记录，只需要在每次绘图结束前向固定几个位置添加草的图块即可。

墙：该地形用于阻挡蛇的前进，增加游戏难度。墙的位置也是在设计地图时确定好并且不会改变的，但是因为墙不能和其他物体共存，所以墙在map中有特定的标记。当蛇前进时，若蛇头接下来会撞上墙，则蛇头不再移动，蛇尾按照原本的规则继续前进，因此蛇的长度会缩短。若玩家没有及时调整方向，长度缩短到0，则直接输掉游戏。

# 5 难点及解决方案

## 5.1 定时器问题

开始的时候使用的定期器为timeSetEvent函数，但程序运行几分钟后就会崩溃，显示栈溢出。当不使用timeSetEvent限制刷新速度的时候，程序不会崩溃，所以问题在于timeSetEvent函数，查询资料也没找到问题。所以后面只能自己实现定时器实现刷新速度的限制。