**高级语言程序设计**

**课设报告**

**题 目\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**学 号\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**姓 名\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**指导教师\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**提交日期\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**成绩评价表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **报告内容** | **报告结构** | **报告图表** | **界面** | **最终成绩** |
| **□丰富正确**  **□正确**  **□基本符合要求**  **□过于简单** | **□完全符合要求**  **□基本符合要求**  **□有欠缺** | **□符合规范**  **□基本符合规范**  **□有一些错误** | **□丰富、炫酷**  **□有背景图片**  **□有背景色**  **□简单**  **□开始界面 □游戏界面** |  |
| **程序功能实现** | **程序执行情况** | **问题回答情况** | **总体评价** | |
| **□多个扩展功能**  **□少量扩展功能**  **□完成基本功能**  **□未完成基本功能** | **□流畅**  **□界面有闪动**  **□操作不灵活**  **□有错误** | **□清楚、正确**  **□基本正确**  **□回答有部分错误**  **□不能回答问题** |  | |

**教师签字:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

目录

[1 需求分析 2](#_Toc479255613)

[1.1 功能 2](#_Toc479255614)

[1.1.1 基本功能 3](#_Toc479255615)

[1.1.2 扩展功能 3](#_Toc479255616)

[1.2数据 3](#_Toc479255617)

[1.3界面 4](#_Toc479255618)

[1.4 开发与运行环境需求 5](#_Toc479255619)

[2 总体设计 5](#_Toc479255620)

[2.1 主要数据结构 5](#_Toc479255621)

[2.2 程序模块设计 7](#_Toc479255622)

[2.2.1 子模块设计 7](#_Toc479255623)

[3 关键技术的实现 12](#_Toc479255624)

[3.1 动画的实现 12](#_Toc479255625)

[3.2 鱼的运动方式 13](#_Toc479255626)

[3.3 鱼的生成方式 13](#_Toc479255627)

[3.4 游戏的状态存取 14](#_Toc479255628)

[3.5 鱼的追踪算法 14](#_Toc479255629)

[4 测试 14](#_Toc479255630)

[5 总结提高 15](#_Toc479255631)

[5.1 课程设计总结 15](#_Toc479255632)

[5.2 对本课程意见与建议 15](#_Toc479255633)

[5.3 附件：程序源代码 15](#_Toc479255634)

# 1 需求分析

这次课设的目标是制作一个大鱼吃小鱼的游戏。主要要实现的功能有要实现小鱼在碰到高级的大鱼的时候要扣分，而在碰到比自己级数略低的小鱼时，要做出相应的奖励。在这个过程中，要对玩家的游戏成绩进行记录。这个程序应该有自己的菜单，以及游戏内容等等。

由于老师有严格限制语言和使用的函数库。所以我尽量使用最基础的函数完成整个游戏。在这一点上，可以从我的代码中完整的体现出来。

## 1.1 功能

游戏功能主要分为以下内容：

1. **游戏主菜单：**
   1. **开始游戏**

这里是玩家开始游戏的地方。

* 1. **继续游戏**

这里玩家可以恢复保存的游戏

* 1. **游戏选项**

这里玩家可以设定游戏的设定

* 1. **关于**

这里玩家可以看到关于作者与开发过程的内容

* 1. **退出**

这里玩家可以退出游戏

1. **游戏内容**
   1. **中断游戏**

玩家中断游戏，返回至主菜单

* 1. **渲染**

渲染游戏内容，我们采取了帧为单位的渲染方式

* 1. **判定**

判定玩家位置信息，并作出逻辑判断

* 1. **存储操作**

存储游戏相关数据结构（例如鱼的链表，以及鱼素材的纹理）

1. **游戏资料存储**
   1. **游戏内容存储**

将中断游戏时的游戏内容存储至文件，并可以从文件中恢复游戏内容。方便玩家进行游戏。

* 1. **游戏设置存储**

将游戏设置存至文件，包括分辨率设置（需重启游戏）以及难度设置。

### 1.1.1 基本功能

1、可以以链表形式存储小鱼

2、可以判定玩家鱼的位置与每只在链表中的小鱼的关系

3、可以记录玩家的得分情况

4、小鱼可以以多种运动方式在游戏中移动

### 1.1.2 扩展功能

1、加入了更加美观的主菜单以及主菜单背景动画

2、加入了游戏设置与游戏内容的存储功能

3、加入了继续游戏的功能

5、加入了小鱼游动会避开玩家，大鱼游动会朝向玩家的设定

## 1.2数据

1. 影响游戏难度的参数

1）游戏模式 （整型）

2）速度比率 （单精度浮点）

1. 需要保存的内容
2. 玩家名字 （字符串）
3. 起始时间戳，结束时间戳 （TIME）
4. 游戏得分 （整型）
5. 无敌模式 （布尔值）
6. 游戏级别 （浮点）
7. 电脑NPC链表头 （FISH结构体指针）
8. 游戏环境相关

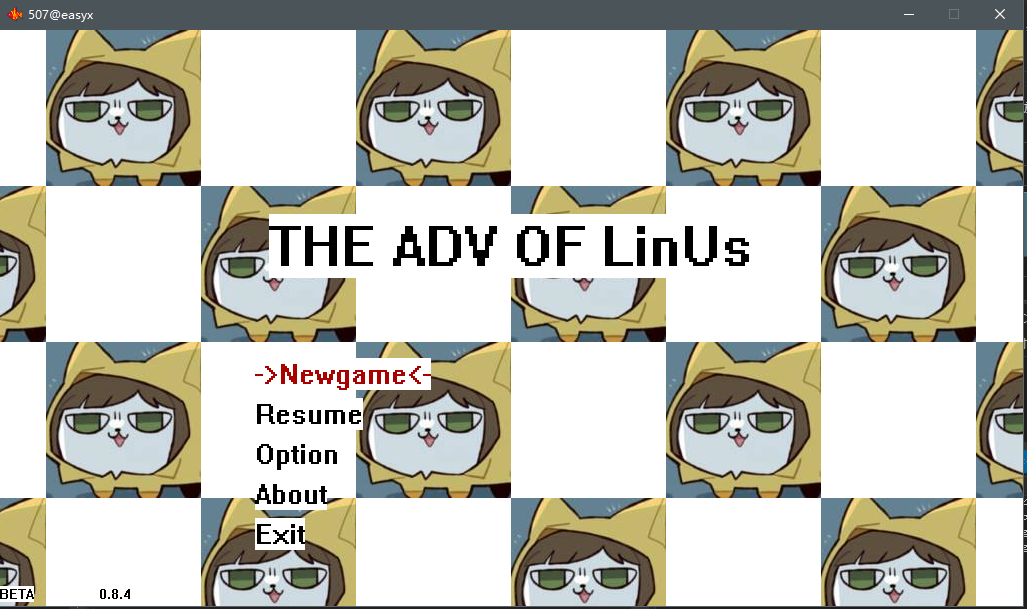
1）鼠标消息 （MOUSEMSG）

2）NPC图片素材表 （IMAGE[]）

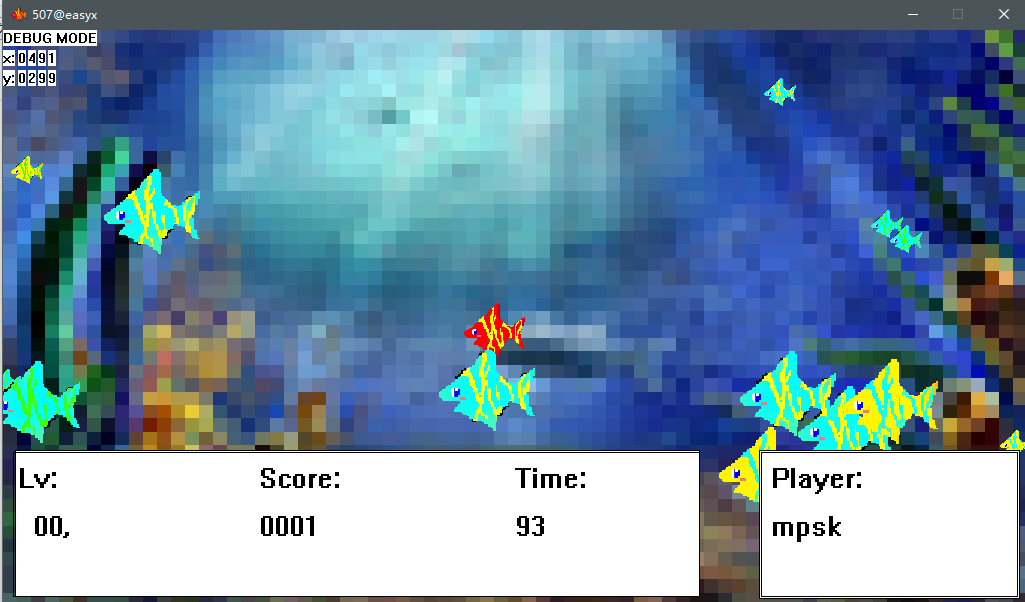
3）玩家图片素材 （IMAGE）

4）背景图片素材 （IMAGE）

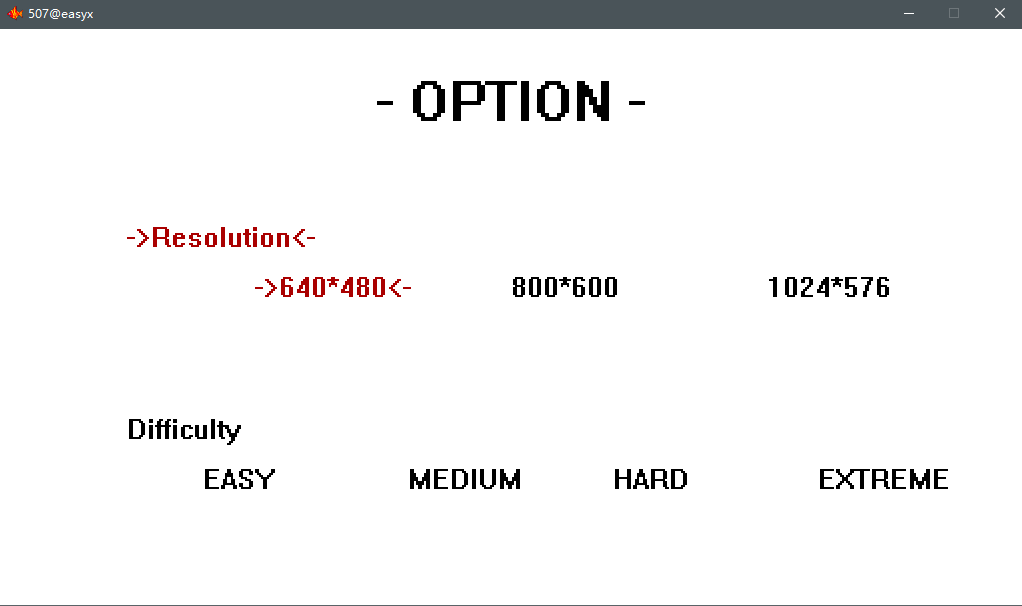
## 1.3界面



游戏主菜单



游戏内容



游戏设置界面

## 1.4 开发与运行环境需求

Visual Studio 2015 with easyX

CPU: 1.6GHz Pentuim 以上 推荐 2.4GHz 赛扬处理器

Memory: 1GB以上 推荐 2GB

运行系统: Windows XP SP1以上 推荐 Windows 7 SP1

软件为32位

# 2 总体设计

在需求分析的基础上，采用自顶向下的方法进行模块化的程序设计，合理划分模块。在此部分应给出程序总体的设计方案，划分了几个模块。

## 2.1 主要数据结构

游戏结构体GAME，包含游戏内的所有参数，以及进行内容操作的资源。

typedef struct game

{

    //  mode 0 for rand position based

    //  mode 1 for rand speed based

    //  other is for rand speed without y

    int         mode = 2;

    int         speed\_ratio = 1;

    wchar\_t     \*player;                    //  Player's name

    int         time\_begin, time\_sec;       //  Timer for count Game time

    int         score = 0;                  //  Score

    bool        god;                        //  GOD Flag

    float       level;                      //  Player's level

    MOUSEMSG    mouse;                      //  Mouse Message

    IMAGE       npc\_fishes[RES\_FISHES];     //  NPC Texture Array

    IMAGE       player\_fish;                //  Player's fish texture

    IMAGE       background;                 //  Background Texture

    FISH        \*fish = NULL;               //  Head of NPC chain list

}GAME;

鱼节点结构：用来存储鱼类特征的结构体，鱼的存储方式是线性链表

typedef struct fish

{

    float x;                //  Node fish's position in X

    float y;                //  Node fish's position in Y

    float s\_x;              //  Node fish's speed in X

    float s\_y;              //  Node fish's speed in Y

    float level;            //  Node fish's level(judge score or hurt)

    int res\_num;            //  Alternate texture amount

    int flag;               //  From Left or Right

    fish \*next;             //  Pointer to next

}FISH;

这里我们抽象了每个NPC的位置与速度。我们不直接操控NPC的位置（不直接让位置参与递增或递减运算），而是通过控制抽象速度来控制鱼的运动。鱼的位置是通过将鱼的速度计算得来的。

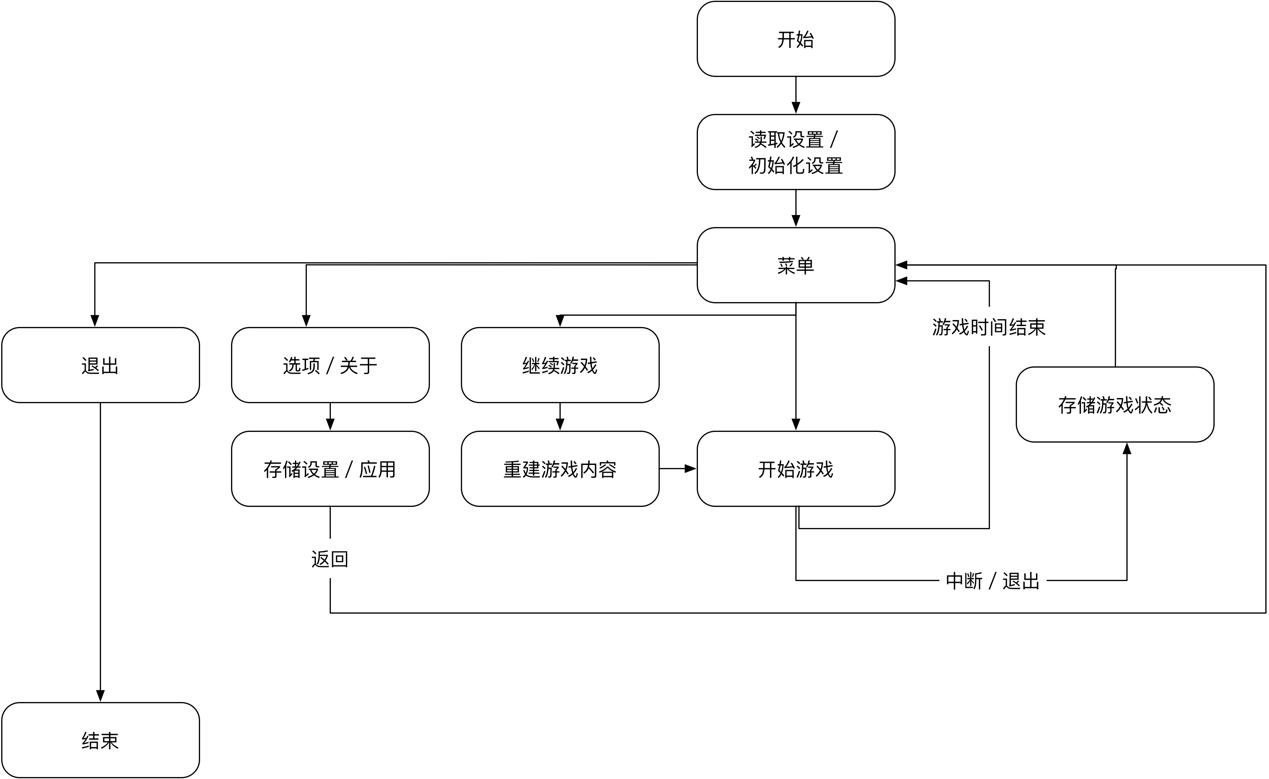
鱼的速度是通过随机方法得来的。它在y方向的速度是完全的随机，但是x方向的速度是有偏向性的，这使得不同朝向的NPC可以朝向对应的方向运动。

我在游戏世界中抽象了阻力系数。这样鱼类在运动的时候速度就会慢慢减小直至接近停止（并不是绝对的等于停止，而是存在阈值控制）。在近似停止的情况下，程序会随机为NPC生成一个速度，让NPC继续运动下去。这部分我们会在下面继续详细解释。

同时，为了防止鱼在同一个地方卡住，我们还规定了flag，来区分不同来源的NPC，让他们能够大致朝向同一方向运动。

为了增加NPC的多样性，我们使用了NPC图片素材表的下标值来标记NPC不同的纹理。而且我们还可以通过随机数来让NPC使用不同的纹理。

## 2.2 程序模块设计



### 2.2.1 子模块设计

我接下来的所有程序设计都是围绕着操作游戏结构体来展开的。实际上这些模块都是逻辑上的，仅仅体现在函数命名上。

1. **开始游戏**

这个模块主要是创建一个新的游戏结构体并维护。使其完成对游戏内容的更新与渲染。此模块中主要的内容就是维护一个游戏结构体。

在加入存储功能以后，这个模块被拆分成了建立游戏对象和维护游戏对象两个部分。

首先我们在进入游戏的时候，会询问玩家的名字，这个时候程序会组建游戏结构体的内容，初始化所有的游戏变量，包括NPC链表、玩家级别、素材的载入。

之后程序会提示准备好，玩家就将进入游戏界面开始游戏。

开始游戏后，玩家碰到级别更小的鱼会分数加一，而碰到级别更大的鱼则会扣掉5分，同时玩家会获得大概1秒左右的无敌。在无敌状态下，玩家不会受到伤害。（分数不会减少）

游戏如果在游戏时间结束前按下了ESC，那么就认为玩家中途退出，程序会自动将当前游戏状态（包括游戏级别，游戏得分，游戏NPC的所有信息）存储至文件，使之可以在下次运行（可以关闭游戏）时，恢复游戏的内容。

当然在游戏结束的时候（不管是中途退出还是游戏时间结束），玩家的得分都会显示在屏幕上。当然在游戏时间结束后，玩家是没有办法再次存档的。

1. **继续游戏**

这个模块主要是从文件中恢复之前存储的游戏内容，相当于从文件读取游戏内容并重新组织一个游戏结构体。

当玩家选择了这个选项时，程序会从唯一的存档文件中格式化读取之前程序存储的内容。

存储文件的起始是玩家的名字，玩家的得分，玩家的级别，以及玩家的游戏时间。这些相对静态的游戏内容在存档文件中很容易恢复。但是对于NPC链式表，就需要格式化读入后，动态再次组建。由于链表中元素都相对平等，于是他们之间的物理存储位置并不影响游戏内容的再现。因此，我们就可以根据影响NPC行为与表现的信息重新组织结构体，并将他们再次相连。在组装好链表后，程序会将组织好的链表头返回给游戏结构体。这样一个游戏结构体就重新组装好了。

重组好了游戏结构体后，我们就需要将游戏结构体重新送回维护函数，进行正常的游戏内容了。

当然如果玩家从恢复的游戏过程中中途退出，其进度一样会被保存。

1. **选项**

这个模块用于维护从文件中读取或向文件中写入设置，并将设置传入主程序。

玩家可以选择不同的分辨率，但是这个操作需要重新启动。因为我们在创建绘图区域的时候，就需要一个固定的值。我们在选项中提供了三个常用的分辨率，当然玩家也可以通过自己修改Setting.bin文件来自己设定分辨率参数，来获得更好的体验。但是这种操作可能会带来任何不可意料地后果。例如画布创建失败等等。一旦出现这种情况，仅仅需要删除Setting.bin，清空已经保存的设置文件，程序就可以重新生成正确参数的设置文件。

关于难度，目前的版本我抽象了两个参数，一个是mode，另一个是speed\_ratio。这两个变量分别控制鱼类的运动模式，和运动的修正因数。我认为通过控制这两个变量可以有效控制游戏的难度。游戏内置了四种预设难度，当然玩家也可以通过手工修改这两个参数，来达到生成自定义难度的游戏。但是同时和分辨率一样，手工修改可能会导致不可意料的后果，如果出现游戏崩溃的情况，只需要删除相应的设置文件并重启游戏就可以了。

1. **关于**

关于页面介绍了我的个人信息，以及我想对玩家说的话。我的整个程序文件和开发进度，以及一些相关的信息可以在github上我的主页找到。我的github账号是mpskex@163.com。欢迎各位老师关注。

这个游戏内容是完全由我个人开发的。并且这个程序遵循MIT协议。代码完全开源，并且在非商业用途情况下可以被转载。

关于接下来的开发计划，我自己在完善一个基于SDL2（OpenGL）的游戏引擎。目前这个引擎简单支持lua，并且拥有自己的粒子系统和抽象2d物理引擎。整个项目基于c++，未来会加入更多的元素，同样在我的github上公开。欢迎老师来关注。

1. **退出**

这个部分就是使游戏退出的。

1、鱼的管理

本模块用于和鱼相关的操作。

相关函数：

int fish\_add(GAME &game, int num, int SCREEN\_W, int SCREEN\_H);

功能：向鱼链表中添加一条鱼

参数：游戏结构体，添加数量，屏幕宽，屏幕高

返回值 整型，表示添加状态（成功/失败）

int fish\_clear(GAME &game)

功能：清空鱼链表

参数：游戏结构体

返回值：整型，表示删除状态

int fish\_init(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：初始鱼链表，创建链表的头指针

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示删除状态

int fish\_judge(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：判定鱼的位置是否与玩家相交

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示判断状态（成功/失败）

FISH \*fish\_rm(GAME &game, FISH \*fish)

功能：删除特定的鱼

参数：游戏结构体，屏幕宽，屏幕高

返回值：指向鱼类型的指针（被删除的鱼的下一条鱼）

int fish\_single(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：鱼类显示的单周期函数，包括鱼类运动方式

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

2、游戏内容

int game\_background\_single(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：显示游戏世界背景

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

int game\_loop(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：游戏世界渲染循环

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

int game\_main(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：游戏内容主函数

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

返回值决定了游戏走向（返回值代表退出游戏类型）

int game\_player\_single(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：显示玩家对象

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

int game\_score(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：显示最终游戏结果

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

int game\_status\_single(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：游戏状态栏显示

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

3、接待画面

int name\_loop(GAME &game, int SCREEN\_W, int SCREEN\_H)

功能：接待画面渲染循环

参数：游戏结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

4、选项画面

int option\_loop(OPTION option, int SCREEN\_W, int SCREEN\_H)

功能：选项画面渲染循环

参数：选项结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

5、菜单画面

int menu\_loop(MENU &menu, int SCREEN\_W, int SCREEN\_H)

功能：菜单画面渲染循环

参数：菜单结构体，屏幕宽，屏幕高

返回值：整型，表示单步执行状态（成功\失败）

void menu\_show\_version(int SCREEN\_W, int SCREEN\_H)

功能：菜单页面显示版本

参数：屏幕宽，屏幕高

返回值：无

6、文件操作

int load\_setting(SETTING &setting)

功能：从文件载入设置

参数：设置结构体

返回值：整型，表示单步执行状态（成功\失败）

int write\_setting(SETTING setting)

功能：向文件写入设置

参数：设置结构体

返回值：整型，表示单步执行状态（成功\失败）

int write\_setting(SETTING setting)

功能：向文件写入设置

参数：设置结构体

返回值：整型，表示单步执行状态（成功\失败）

int write\_save(SAVE \*save);

功能：向文件写入内容

参数：游戏状态结构体

返回值：整型，表示单步执行状态（成功\失败）

int load\_save(SAVE &save);

功能：从文件读取内容

参数：游戏状态结构体

返回值：整型，表示单步执行状态（成功\失败）

7、关于画面

void about\_loop(ABOUT about, int SCREEN\_W, int SCREEN\_H);

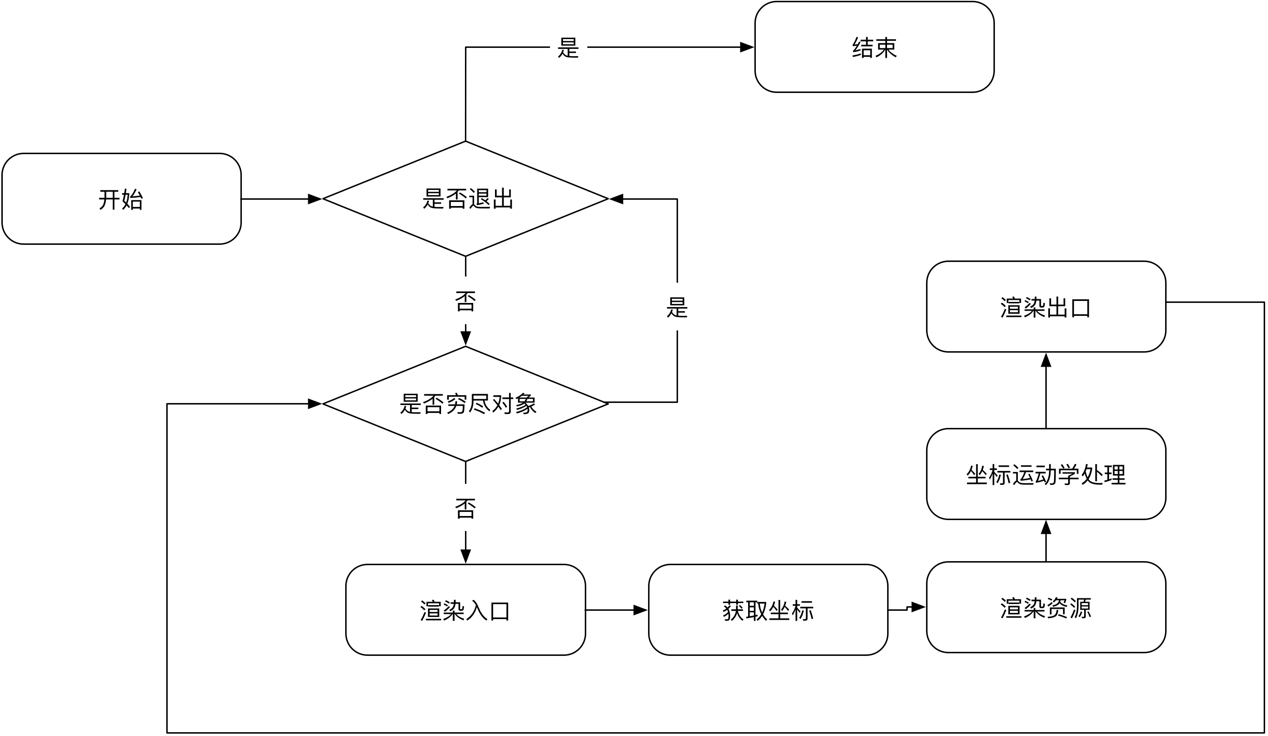
功能：关于画面渲染循环

参数：关于结构体

返回值：无

# 3 关键技术的实现

## 3.1 动画的实现



历遍NPC链表的过程是整个游戏渲染Loop中的重中之重，因为这段代码是最容易出现指针越界，非法指针的。因此对于涉及到链表以及指针的代码我都是相当谨慎的。

在完成这部分代码的时候，其实比较顺利，主要就是注重起始条件，迭代过程，和终止条件。如果对于整个要操作的边界情况我们都考虑清楚的话，链表就会变得非常灵活而且看起来更加聪明伶俐。

我们在迭代渲染这部分内容的时候，会对于每一个NPC节点的位置信息进行计算并且将其位置用对应的素材画出来。当我们对于每一个节点都历遍的时候（判定条件指向指针等于空），我们就将之前绘制的内容呈现在屏幕上。由于计算机处理速度相当快，所以历遍链表并且绘制图像的速度非常短，可以说是一瞬，那么通过循环地迭代这个过程，再通过一定手段控制NPC的坐标，我们就可以绘制出流畅的动画了。

## 3.2 鱼的运动方式

我们采用了抽象鱼的速度与鱼的位置，方便我们的坐标的运算。

根据公式△*r* =*r*（*t*+△t）－*r*（*t*），我们就可以用一个常量来表示一个固定的抽象加速度，然后根据随机生成的x，y速度，计算出对应的鱼的位置。

根据公式v = ∆S／∆t，我们可以将∆t简化为1（tik tok），那么得到的了公式∆S = v，这样我们简易的运动学模拟就完成了。

鱼的运动系数主要是通过速度描述的，而这个速度还会乘上一个时间系数SPEED\_RATIO（宏）（在早期版本是宏形式出现的），相当于∆S = v \*（∆t \* speed\_ratio）。因为不直接操作位置，我们就能获得更佳平滑的NPC运动效果，也会显得更佳自然。

鱼的速度是通过随机方法得来的。它在y方向的速度是完全的随机，但是x方向的速度是有偏向性的，这使得不同朝向的NPC可以朝向对应的方向运动。

我在游戏世界中抽象了阻力系数。这样鱼类在运动的时候速度就会慢慢减小直至接近停止（并不是绝对的等于停止，而是存在阈值控制）。在近似停止的情况下，程序会随机为NPC生成一个速度，让NPC继续运动下去。

## 鱼的生成方式

如果NPC生成的大小都是固定的，那么游戏就没有办法推进。但如果鱼类生成的大小过于随机，那么也许会生成极小的鱼，影响游戏的观感与体验。那么如何生成大小合适的鱼类就成了一个非常需要解决的问题。

\_fish->level = rand() % 10 / 7 + game.level - (0.5 / game.level);

这就是控制生成鱼类大小的公式。首先这个随机数除以7，是在控制生成数的散度，如果这个散度过大，就也许会出现生成过小或者过大的鱼类。之后与游戏级别加和的部分是要将现在游戏级别作为生成鱼的基准。最后减去的部分是作为修正因数出现的。如果游戏级别过高，那么生成小鱼的几率就会变小，使得游戏的难度虽玩家级别的上升而增高，从而达到压制最高分的功能。

NPC出现的位置也是需要考虑的问题。鱼类应该是在屏幕两端生成。并且由于前面运动系统设计导致我们需要将两侧的鱼类分开，那么我们就是用了flag来区分。flag为1的鱼类沿x轴速度只会递增，flag为2的鱼类沿x轴的速度只会递减。但是在NPC生成的同时，我们需要给他一个初始速度，那么这个初始速度也是根据他们的flag值确定的。根据不两种flag，我们对应生成正与负的沿x轴的初始速度。这样的话他们就会规矩地向游戏场景中央移动了。

如果NPC的外观只有一种，那么就会显得太过于单调了。因此我加入了不同的NPC外观素材。在生成的过程中，我们使用了随机方法，为他们分配素材表中的对应下标，并在渲染函数过程中，提取对应下表的素材，完成不同外观的NPC的绘制。

## 游戏的状态存取

存储文件的起始是玩家的名字，玩家的得分，玩家的级别，以及玩家的游戏时间。这些相对静态的游戏内容在存档文件中很容易恢复。但是对于NPC链式表，就需要格式化读入后，动态再次组建。由于链表中元素都相对平等，于是他们之间的物理存储位置并不影响游戏内容的再现。因此，我们就可以根据影响NPC行为与表现的信息重新组织结构体，并将他们再次相连。在组装好链表后，程序会将组织好的链表头返回给游戏结构体。这样一个游戏结构体就重新组装好了。

重组好了游戏结构体后，我们就需要将游戏结构体重新送回维护函数，进行正常的游戏内容了。

我们没有选择直接存储结构体的二进制内容。二进制存储很难解析，通俗讲，就是我都不知道存的是什么。于是我们使用将结构体的数值型内容（除了与结构体的头指针）转存入一个存档容器中（SAVE结构）然后我们分别将这个结构中的变量一一写入文件当中。这种存储方式可以以文本形式打开存档文件，而且内容还是可读的格式。（使用fprintf）

读取的时候我们需要重新根据文本文件中的内容重新组织链表。对应每一条记录，我们需要新建一个节点，并将它的上一个节点的子节点指针赋值。这样就非常的简单地完成了重建链表的功能。

## 鱼的追踪算法

# 4 测试

在此部分报告应给出你所设计的用于整个程序的测试用例和测试结果。测试用例的设计应包括：正确运行程序的用例、导致程序运行错误的用例、边界数据的用例等。

如果需要可以再细分小节。

# 5 总结提高

## 5.1 课程设计总结

同学依据自己的经历写出个性化的总结。内容可以包括：程序开发中的体会与收获，开发中遇到的问题与解决情况，自己对自己完成课设情况的评价等等。

## 5.2 对本课程意见与建议

我们共同完成了一个学期的学习，其中有辛酸，也有收获，有感动，也有遗憾，请同学们针对《高级语言程序设计Ⅲ》与《高级语言程序设计课程设计》这两门课程谈谈你的体会，总结优点，给出不足，为我们提高教学质量做出你的贡献。谢谢同学们！

## 5.3 附件：程序源代码