
电子科技大学 计算机 学院

实验报告

课程名称 C++程序设计

实验题目 精灵游戏（大鱼吃小鱼）

难度级别 5

提交时间 2021.12.16

姓 名 覃泽

学 号 2020080908028

1 软件说明

1.1 游戏使用说明

吞食鱼 Feeding Frenzy，国内又称大鱼吃小鱼，是一款动作类小游戏，游戏的目标是通过不断的吞吃比自己小的鱼类快速成长，不断升级进化，最终成为海洋霸主。作为这类游戏的爱好者，我使用 C++语言自主设计了一款基于 ACLLIB 图形库的大鱼吃小鱼精灵游戏，它具有大鱼吃小鱼这类游戏的基本功能：玩家可以控制自己的鱼在“海中”进行游弋，在不断捕食比自己等级低的鱼、增加自己的分以达到升级变大所需条件的同时，还要避免被比自己等级大的 NPC 鱼吃掉，否则游戏结束。运行游行需要在 VS Code 终端输入命令行，进行编译后执行游戏。（详见 1.2 1.2 源码编译说明）

1.2 源码编译说明

游戏所在文件夹 Feeding Frenzy 含有 ACLLIB 库头文件和 C 语言文件：acclib.h, acclib.c;同时含有实现大鱼吃小鱼游戏中基类鱼的头文件和 cpp 文件：FishBase.h, FishBase.cpp；玩家所控制鱼的头文件和 cpp：PlayerFish.h , PlayerFish.cpp; NPC 鱼的头文件和 cpp 文件：Fishlv1shrimp.h , Fishlv1shrimp.cpp, Fishlv2littlefish.h, Fishlv2littlefish.cpp, Fishlv3whale.h, Fishlv3whale；主文件 StartGame。本游戏使用命令行编译器 MinGW 把源代码编译成最终的可执行程序。具体步骤是先在在 Vs Code 打开文件夹 Feeding Frenzy，新建 cmd 终端，输入以下三行编译命令（每输入一行按下回车键）：

```
1. gcc -c *.c
2. g++ -c *.cpp
3. g++ *.o -lgdi32 -lole32 -loleaut32 -luuid -lwinmm -
4. lmsimg32 -o game.exe
```

其中，“gcc -c *.c”命令用以编译 ACLLIB 库文件和各类鱼头文件；“g++ -c *.cpp”用来编译实现各类鱼功能的 cpp 文件；“g++ *.o -lgdi32 -lole32 -loleaut32 -luuid -lwinmm -lmsimg32 -o game.exe”用来调用命令行编译器 MinGW 生成一个名为“game.exe”可执行文件。

然后输入以下命令运行可执行文件，进入游戏初始界面：

```
5. game.exe
```

2 系统设计

（1）基类鱼 Class FishBase:

基类鱼 FishBase 定义了一些具有保护属性的变量如下：

```
1. protected:
2.     int x, y; //鱼的坐标
3.     int dx, dy; //鱼单位时间移动距离
4.     int level; int x, y; //鱼单位时间移动距离
5.     ACL_Image *ring; //右转向图片
6.     ACL_Image *ling; //左转向图片
7.     rect r; //定义矩形
```

这些保护变量定义了一条鱼的基本信息：x,y 用来记录鱼当前的坐标；dx,dy 用来记录鱼单位时间移动距离，也就是鱼的移动速度，它是有正负之分的整型变量，它的正负影响鱼的朝向，若为正朝右，若为负朝左；level 用来记录鱼的等级，它在后续 NPC 鱼的逃避系统和玩家鱼的得分系统中均有用到；*rimg 和*limg 均为 ACLLIB 图形库所定义的图片类型的变量指针，*rimg 为鱼右转向图片，当一条鱼生成时，默认初始朝向向右，鱼生成后，若鱼向右移动，则加载*rimg 所指向的图片，若鱼向左移动，则加载*limg 所指向的图片；r 为 ACLLIB 图形库所定义的矩形类型 rect 变量，其作用相当于一个有大小的矩形边框，*rimg 和*limg 指向的图片填充在里面，它涉及鱼的移动、绘制以及玩家鱼碰撞、得分和 NPC 鱼逃避等。

基类鱼所实现的功能可以用图 1 表示：

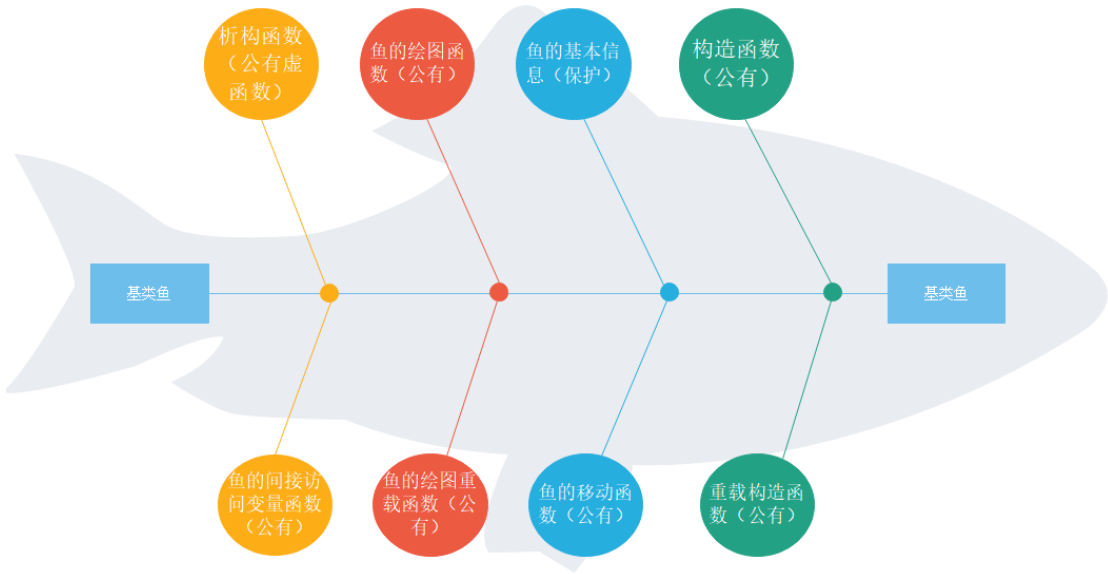


图 1

如图 1，基类鱼定义了公有函数：

构造函数和同名重载构造函数，用以初始化鱼的基本信息；

基类鱼的移动函数为虚函数，这是因为玩家鱼和 NPC 鱼有不同的移动方式，继承基类鱼时，派生出的移动函数也不同；

绘图函数和同名重载绘图函数用来加载鱼的右转向和左转向图片，实现函数时，需要先判断整型变量 dx 的正负来确定鱼的朝向，然后加载通过调用 ACLLIB 库的函数加载图片；

基类鱼的间接访问变量函数用来在类外实现对类内保护变量进行访问。由于 level 和 r 时基类鱼在类外访问最频繁的变量，因此定义访问这两个变量的函数。

几种鱼的关系如图 2：

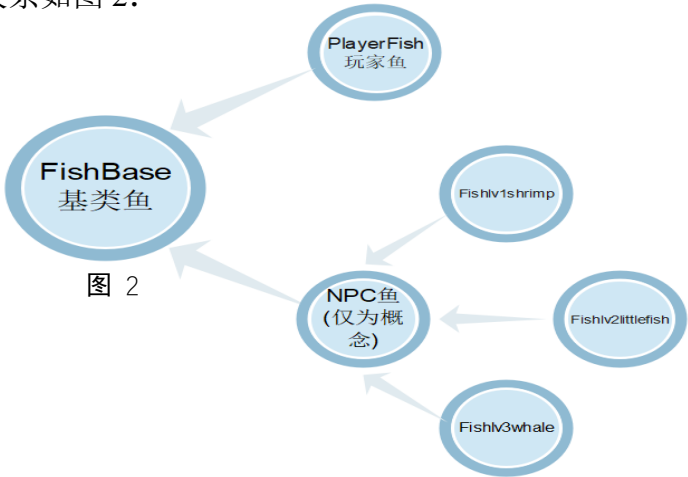


图 2

（2）玩家鱼 Class PlayerFish

玩家鱼 PlayerFish 可实现的功能如图 3 所示。首先是公有继承基类鱼，通过构造函数获取基本信息，其次定义了私有变量 `score` 用来记录玩家得分，与得分系统相挂钩，玩家吃的小鱼越多，得分也越高，当得分达到某些值时，会触发玩家鱼的升级系统。

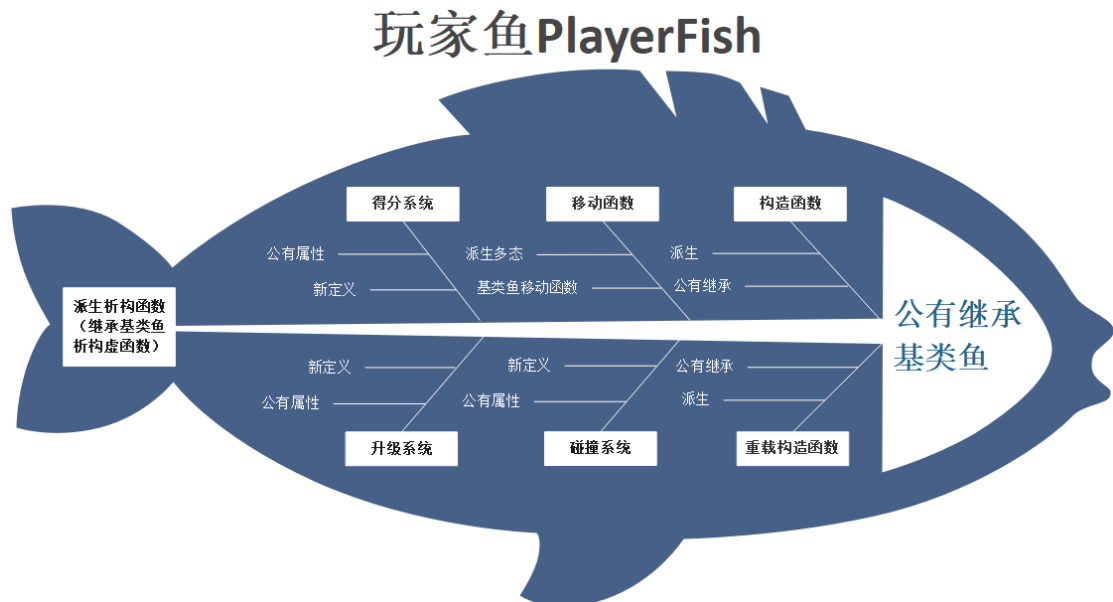


图 3

现在简要分析玩家鱼的各项功能：

玩家鱼的移动函数和同名移动函数由基类鱼派生而来，需要接收用户的键盘键入从而与用户进行交互，实现上、下、左、右、左上、右上、左下、右下八个方向的移动，玩家鱼在此过程中坐标 `x,y` 发生改变，`r` 进行移动，移动速度和等级有关，等级越高，移动速度当然越快。

碰撞系统为新定义的公有函数，它需要接收另一个“矩形边框”的值，通过两个矩形的顶角角坐标关系来判断是否发生碰撞，如果发生碰撞，返回一个布尔类型变量的值：

得分系统在类内实现时较为简单，仅仅是对玩家鱼的 `score` 进行访问或者修改（增加），该系统主要在类外，在后续的主文件 `StartGame` 文件中完善。

升级系统可类比为含有部分变量的构造函数，当玩家升级时更新 `width`, `height`, `level`, `limg`, `rimg`，使得玩家鱼的形态发生变化，该系统也在类外，在后续的主文件 `StartGame` 文件中完善。

（3）NPC 鱼 Class Fishlv1shrimp, Fishlv2littlefish, Fishlv3whale

三种 NPC 鱼均由基类鱼公有继承而来，实现功能都相同（如图 4 所示），它们的不同仅仅是基本信息不同（等级、移动速度、形态等）。

NPC 鱼首先是公有继承基类鱼，通过构造函数获取鱼的基本信息，其次定义了私有变量 `score` 用来记录当玩家吃掉 NPC 鱼时可获得的分数。

现在简要说明 NPC 鱼所实现的功能：

NPC 鱼的危险判断系统是新定义的公有函数，它的实现需要玩家鱼的位置信息，通过两个矩形中心的距离来判断 NPC 是否存在危险，如距离小于特定值并且 NPC 鱼的等级小于或等于玩家鱼，则存在危险，返回布尔类型变量的值为真；

动函数和同名移动函数由基类鱼派生而来，它包含了 NPC 鱼的逃避系统和正常移动系统，因此需要传入危险判断系统的返回值，如果存在危险，则 NPC 鱼将朝远离玩家鱼的方向逃避；如果不存在危险，则 NPC 鱼正常游弋。

得分系统在类内实现时较为简单，仅仅是对玩家鱼的 score 进行访问或者修改（增加），该系统主要在类外，在后续的主文件 StartGame 文件中完善。

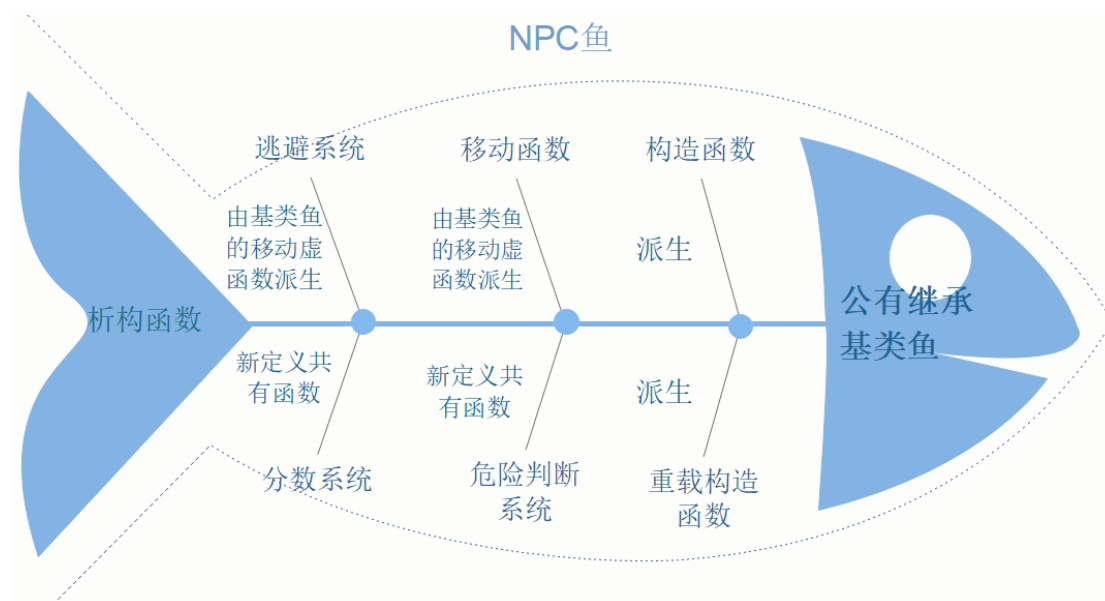


图 4

3 程序实现

所有的鱼类功能均先在头文件中声明，然后在各自对应的 cpp 文件中引用所需的头文件并实现类。实现类后，在 StartGame.cpp 中将类功能完善，并进行实例化，生成对象，并搭建游戏环境。

(1) 定义全局变量

定义每一种 NPC 鱼的最大数量为常量整型 `const int maxNum`，再用一个整型变量 `nowNum` 记录处在游戏界面的鱼数量；定义 `windows` 窗口界面宽和高为常量整型，`const int winWidth` 和 `const int winHeight`；定义一个整型变量 `playerlv`，用于记录玩家鱼等级并在其升级时，通过调用玩家鱼类内函数将 `playerlv` 赋给 `level`；定义两个用于初始界面、游戏界面、游戏结束界面转换的整型变量 `startgame` 和 `ifgameover`，两者初始值均为零；定义玩家鱼类指针并初始化 `PlayerFish *playerfish=NULL`；定义三种 NPC 鱼的类指针数组并初始化，如 `Fishlv1shrimp *fishlv1shrimp[maxNum_shrimp]={0}`；定义指向每一种鱼左右转向图片和初始界面背景图片的指针，如 `ACL_Image playerfish_left, playerfish_right`。

（2）setup 主函数

setup 主函数相当于 ACLLIB 库里的 main 函数。

它先调用 initWindow 函数，根据之前定义 winWidth 和 winHeight 的初始化对话框窗口大小，并为其添加标题 Feeding Frenzy。

其次调用 loadImage 函数将定义的指针变量指向每一种鱼左右转向图片和初始界面背景图片。

然后通过 createData 函数首次生成玩家鱼和 NPC 鱼。玩家鱼仅在 setup 中生成一次。调用 registerTimerEvent 函数注册名称为 timerEvent 的定时器函数。两次调用 startTimer 函数启动定时器，第一次调用将鱼的移动定时器 id 设为 0，每隔 40ms 调用一次定时器函数，对每一条鱼的状态进行更新；第二次调用将 NPC 鱼的生成定时器 id 设为 1，每隔 1000ms 调用三种 NPC 鱼的 createData 生成三种 NPC 鱼。

再调用 registerKeyboardEvent 函数注册名称为 keyEvent 的键盘处理函数函数，用来接收用户的键盘键入。调用 registerMouseEvent 函数注册名称为 mouseEvent 的键盘处理函数函数，用来接收用户鼠标键入。

最后调用 PlaySound 函数播放游戏 BGM。

（3）定时器处理函数 timeEvent

在该函数中利用 switch+break 组合语句，根据 id 的值来执行事件。

当 id 为 0 时，用三个 for 循环语句分别遍历三种 NPC 鱼类指针数组的每一个元素，首先判断该元素是否为 NULL，如果为 NULL 说明 NPC 鱼被吃掉或者未生成，如果不为 NULL 则说明鱼存在在游戏界面上，那么就要调用 playefish（前边定义的类指针变量）中的 getRect 函数获取玩家鱼信息，调用数组中类指针元素的 ifdanger 和 move 函数来判断鱼的移动情况，如果存在危险，则 NPC 鱼逃逸，如果不存在危险，则 NPC 鱼正常游弋。NPC 每 40ms 就通过 dx 和 dy 改变 x, y 的值实现一次移动。上述过程就可以造成 NPC 鱼自动移动的效果。

当 id 为 1 时，用三个 if 语句判断三种 NPC 鱼已生成的数量（包括被玩家吃掉的 NPC 鱼）是否小于之前定义的最大数量，如果小于，则调用 createData 函数生成实例化的 NPC 鱼类对象。

在 switch 语句执行完之后，用 paint 函数对界面进行更新。

（4）对象实例化函数 createData

createData 函数有四个，命名规则是“createData”+“_”+“playerfish / shrimp / littlefish / whale”。

对象化实例化函数里，先调用 rand 函数用来生成窗口范围内的随机数，作为鱼显示的初始位置，由于鱼的 x, y 坐标为大于零的数，所有需要 if 语句判断生成的随机数是否小于 0，如果小于 0，则令坐标为 0；然后根据不同的鱼定义整型变量 dx, dy，用来初始化鱼类单位时间内的移动距离。

玩家鱼的实例化函数仅在 setup 主函数中调用一次，它还通过全局变量 playerlv（值为 1）初始化一级玩家鱼，调用运算符 new 生成玩家鱼，玩家与生成之后，通过玩家鱼类中的 setscore 函数将玩家得分初始化为 0；

NPC 鱼的实例化函数中, dx , dy 是随机的, 代表了在海洋中不同的鱼有不同的运动状态, 有的移动得快, 有的移动得慢。但仅从纵向对比但, 等级越高的鱼移动得越慢, 这通过调整随机数生成范围可以确定, 如 `shrimp` 的 $dx, dy = \text{rand}() \% 5 + 1$; 而 `whale` 的 $dx, dy = \text{rand}() \% 2 + 1$ 。此外还定义了一个整型变量 t , t 也是随机数, 只有当 t 满足一定条件时, 才会使用 `new` 函数实例化 NPC 鱼类指针数组中的元素, 因此 t 相当于生成 NPC 鱼的概率, 等级越高的鱼单位时间内生成概率越低, 符合这类游戏的一个特点。我还根据不同的鱼定义整型变量 lv , 用以初始化鱼的等级 `level`; 定义 `score` 用来初始化鱼的分数, 等级越高的鱼, 玩家吃掉它所获得的分数也越高。

(5) 键盘处理函数 `keyEvent`

该函数需要两个参数, 其中 `key` 是用户使用键盘键入的按键名称, `event` 是键盘是按下还是弹起的事件。玩家鱼的升级、得分系统和游戏结束条件判断也在该函数中完善。

函数先用 `if` 语句判断是否是键盘按下事件, 如果不是, 即 `event != KEY_DOWN`, 则退出该函数。

如果是键盘按下事件:

①执行 `playefish` 中的 `move` 函数。在玩家鱼类中, 我用 `GetAsyncKeyState` 函数接收键盘键入, 结合 `if` 语句, 在 `if` 语句中用 dx, dy 改变 x, y 的值, 这样就实现了玩家鱼的八个方向的移动。

②完善玩家鱼得分功能。用三个 `for` 循环语句分别遍历三种 NPC 鱼类指针数组的每一个元素, 首先判断该元素是否为 `NULL`, 如果为 `NULL` 说明 NPC 鱼被吃掉或者未生成, 如果不为 `BULL` 则说明鱼存在在游戏界面上。然后判断玩家鱼是否与 NPC 鱼碰撞, 如果两者相撞且 NPC 鱼等级小于玩家鱼, 则调用 NPC 鱼 `getscore` 函数访问 NPC 鱼的分数, 然后调用玩家鱼的 `addscore` 函数将返回值加到玩家鱼的 `score` 上, 即为玩家当前得分, 最后使用 `delete` 运算符将 NPC 鱼删除, 类指针变量指向 `NULL`; 如果两者碰撞且 NPC 鱼等级大于玩家鱼, 则调用 `Playsound` 函数播放游戏结束背景音乐, 令游戏结束判断变量 `ifgameover = 1`。

③完善玩家鱼的升级功能。用 `if` 语句判断当前得分, 如果得分到达特定范围, 改变全局变量 `playerlv` 的值, 调用玩家鱼类中的 `upgrade` 函数, 改变玩家鱼大小 (`Width` 和 `Height`)、形态 (`rimg` 和 `limg`)、等级 (`level`)。

最后调用 `paint` 函数更新界面。

(6) 鼠标处理函数 `mouseEvent`

该函数需要四个参数, 其中 mx, my 是鼠标位置坐标, `button` 是用户使用鼠标键入的按键名称, `event` 是鼠标是按下还是弹起的事件。

函数先用 `if` 语句判断是否是鼠标左键按下事件, 如果不是, 即 `event != KEY_DOWN`, 则退出该函数。

如果是, 即 `button == LEFT_BUTTON && event == BUTTON_DOWN`, 则当鼠标的坐标在一个区域内时(即初始界面让玩家开始游戏的按钮), 令 `startgame=1`, 最后在 `paint` 函数中实现从初始界面到游戏界面的转换

（7）界面绘制函数 paint

界面绘制函数负责绘制初始界面，游戏界面和游戏结束界面，是本项目最重要的函数之一。它实现绘制的动作代码在“beginpaint (); clearDevice ();”和“endpaint ()”之间，通过两个全局整型变量 startgame 和 ifgameover 的值来判断需要绘制哪个游戏界面。如果 startgame==0，则说明还未开始游戏，界面停留在初始界面等待玩家鼠标响应。如果 startgame==1 且 ifgameover==0，则说明用户鼠标键入想开始游戏，则界面由初始界面转化成游戏界面。如果 ifgameover==1，则说明玩家鱼被比自己等级高的大鱼吃掉，结束游戏界面，加载游戏结束界面。

①绘制初始界面：

如果 startgame==0，则绘制初始界面。需要调用 ACLLIB 库中的 putImageScale 函数将背景图片加载到窗口中，利用 ACLLIB 库中 setText 系列函数和 paintText 函数生成艺术字“Click Here to Start Game!”引导用户用鼠标点击文字处开始游戏。

②绘制游戏界面：

当用户通过鼠标处理函数 mouseEvent 使得 startgame==1，此时 ifgameover 也为 0，则绘制游戏界面。



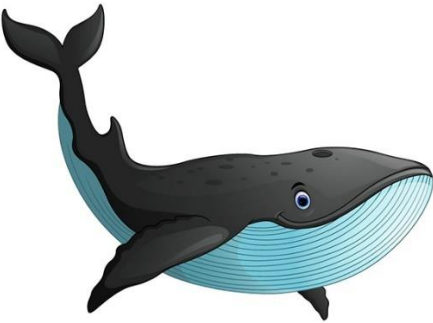


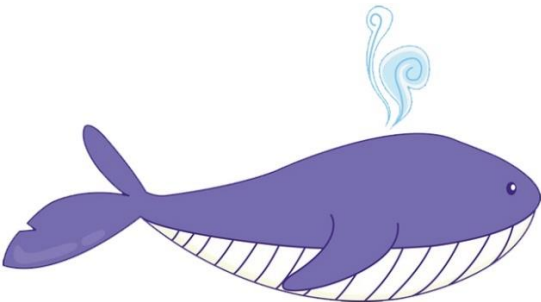

用三个 for 循环语句分别遍历三种 NPC 鱼类指针数组的每一个元素，首先判断该元素是否为 NULL，如果为 NULL 说明 NPC 鱼被吃掉或者未生成，如果不为 NULL 则说明鱼存在在游戏界面上，那么就调用类中鱼的绘图函数和同名重载绘图函数 paintfish 来加载鱼的右转向和左转向图片，实现函数时，首先用 if 语句判断整型变量 dx 的正负，若 dx 为正，则说明鱼正向右运动，此时调用 ACLLIB 库中的 putImageScale 函数加载*ring 所指向的右转向图片；若 dx 为负，则说明鱼正向左运动，此时调用 ACLLIB 库中的 putImageScale 函数加载*ling 所指向的左转向图片。对于 NPC 鱼，生成时默认从屏幕左侧朝右侧移动，“危险情况”出现时，NPC 鱼类中 move 函数会利用 math 库中的 abs 函数改变 dx 正负，即 dx=-abs (dx)，利用 abs 函数也可以保证由左向右的转向；如果没有“危险情况出现”，则 dx=abs (dx)，实现 NPC 鱼的转向。对于玩家鱼，若接收键盘输入向左移动 dx=-abs (dx)，向右则 dx=abs (dx)，这样就可以保证两个转向能够互相转换。

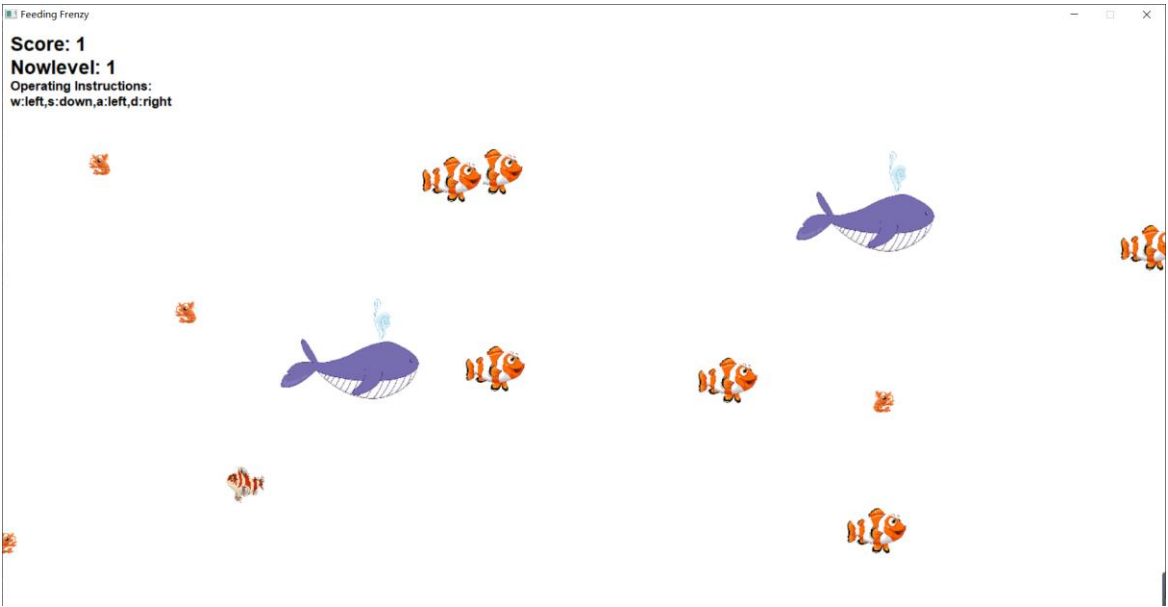
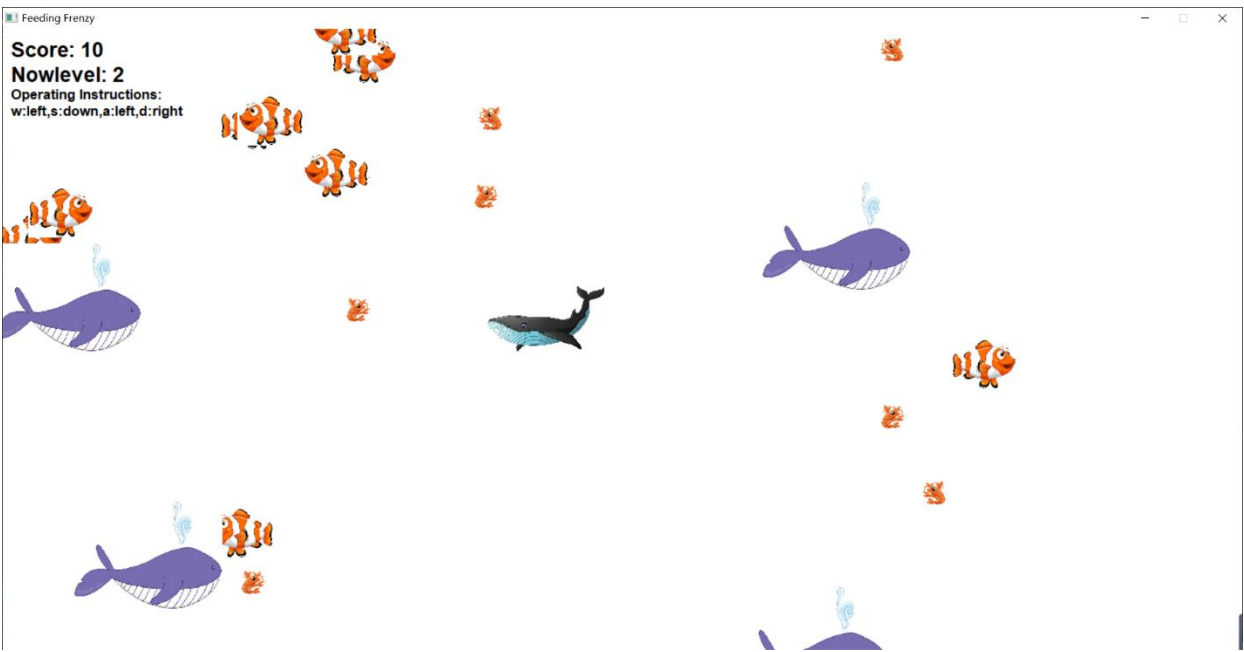
此外还利用 ACLLIB 库中 setText 系列函数和 paintText 函数打印玩家当前得分、当前等级以及游戏操作说明在窗口左上角。

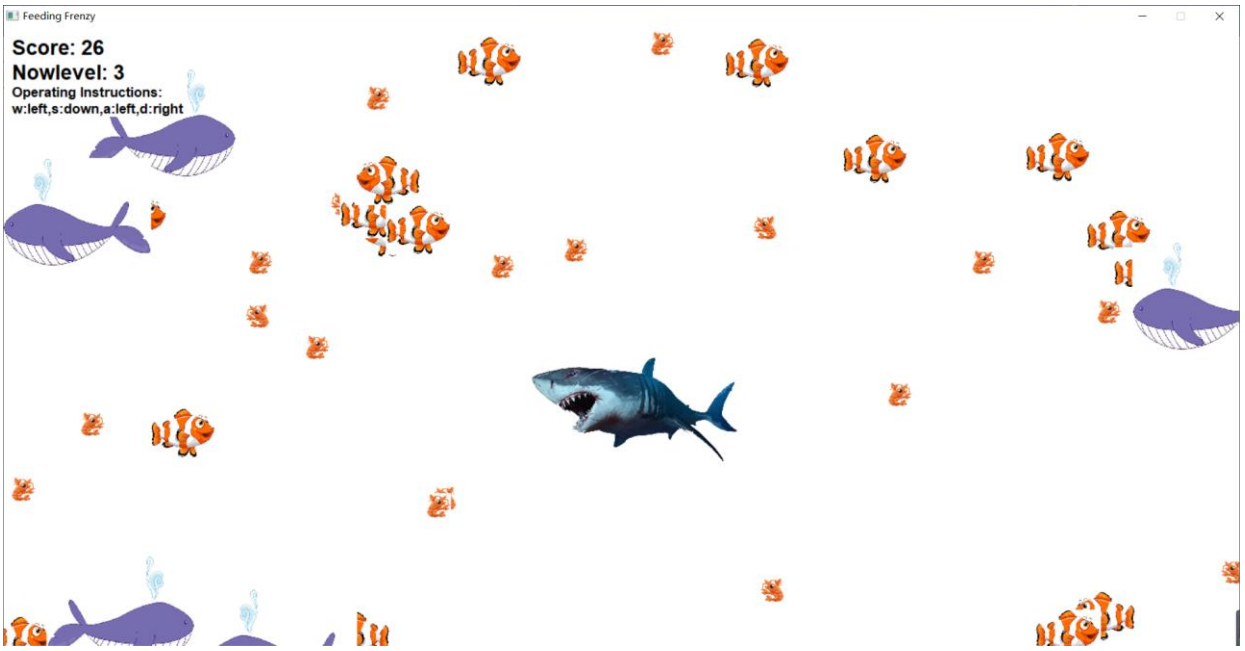
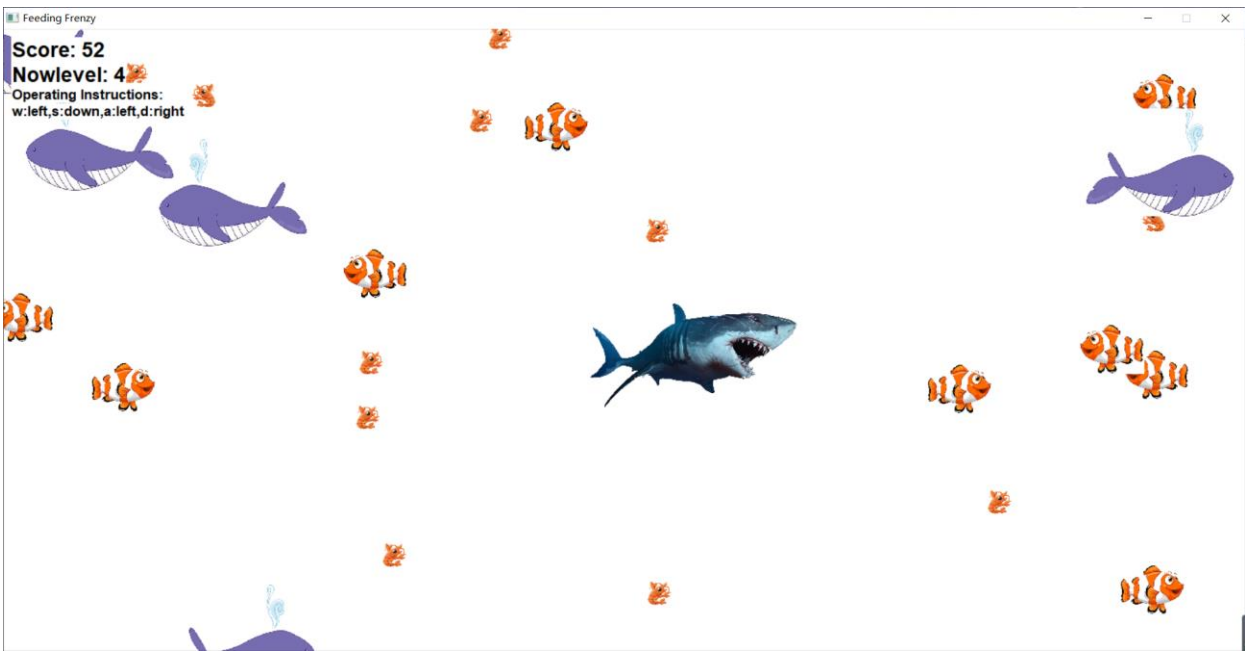
③绘制游戏结束界面：

当 keyEvent 中 ifgameover 的值被修改为 1，则说明玩家被比自己大的鱼吃掉，绘制游戏结束界面。此时需要调用 ACLLIB 库中的 putImageScale 函数将背景图片加载到窗口中，利用 ACLLIB 库中 setText 系列函数和 paintText 函数生成艺术字“Game Over!”。

4 测试报告

鱼类介绍			
playerfish level=1		shrimp	
playerfish level=2		littlefish	
playerfish level=3,4		whale	
运行测试			
初始界面			

	测试结果分析
	输出初始界面，并播放背景音乐 bgm. wav，提示用户用鼠标点击进击“Click Here to StartGame”开始游戏”。经测试，背景音乐可以正常播放。
Nowlevel=1	游戏过程
	
	测试结果分析
	初始状态加载正常。左上角显示玩家当前得分，当前等级和游戏操作说明。鱼都可以实现正常移动和转向。玩家鱼按 w 键向上移动，s 键向下,a 键向左,d 键向右。当按下 a 键和 d 键，玩家可以正常转向。可以通过比自己等级低的小鱼（shrimp）得分。图中存在向左移动的 shrimp，它们等级等于玩家鱼，正在逃避玩家鱼。测试结果符合预期。
Nowlevel=2	游戏过程
	

	测试结果分析
	可以看到，当玩家得分为 10 时，玩家升级为等级 2，形态改变且可以加载正常，图中存在向左移动的 littlefish 和 shrimp，它们等级小于或等于玩家鱼，正在逃避玩家鱼。测试结果符合预期。
Nowlevel=3	游戏过程
	
	测试结果分析
	可以看到，当玩家得分大于 25 时，玩家升级为等级 3，形态改变且可以加载正常，图中存在向左移动的三种 NPC 鱼，它们等级小于或等于玩家鱼，正在逃避玩家鱼。测试结果符合预期。
Nowlevel=4	游戏过程
	
	测试结果分析

	可以看到，当玩家得分大于 50 时，玩家升级为等级 4，形态改变且可以加载正常，图中存在向左移动的三种 NPC 鱼，它们等级小于玩家鱼，正在逃避玩家鱼。此时跟等级 3 的唯一区别是玩家鱼移动速度更快，成为了横扫海底的海洋霸主。测试结果符合预期。
游戏结束	
测试结果分析	
当玩家碰撞到比自己等级高的鱼时，游戏结束。画面出现文字“Game Over!”并正常播放游戏结束音乐 gameover.wav。测试结果达到预期效果。	

5 实验心得

大鱼吃小鱼，是一款动作类小游戏，在海洋区域内，要求通过不断的吞吃比自己小的鱼类快速成长，不断升级进化，最终成为海洋霸主，然而在游戏过程中，也要注意利用走位操作技巧躲避比自己等级高的鱼，否则稍不小心就会被大鱼吃掉，游戏结束。本次实验是基于 C++自主开发的精灵，要求对该游戏进行分析、设计及编程实现。期间最难的就是编辑程序代码，在反复修改代码的时候，会发现总体设计和一些函数的实现出现问题，通过不断的改进，不仅可以让程序达到预期效果，而且还能在修改的过程中去学习。

我遇到的第一个问题是在某个 NPC 鱼类头文件中重复定义了基类鱼的非虚函数，creat_Data 函数名写错，导致在编译时无法通过。通过反复检查，终于将代码修正，顺利通过编译。

第二个问题是如何解决鱼的转向。为解决这个问题，我综合利用了 ACLLIB 库和 math 库中的函数。通过鱼的绘图函数和同名重载绘图函数 paintfish 来加载鱼的右转向和左转向图片，实现函数时，首先用 if 语句判断整型变量 dx 的正负，若 dx 为正，则说明鱼正向右运动，此时调用 ACLLIB 库中的 putImageScale 函数加载*rimg 所指向的右转向图片；若 dx 为负，则说明鱼正向左运动，此时调用 ACLLIB 库中的 putImageScale 函数加载*limg 所指向的左转向图片。对于 NPC

鱼，生成时默认从屏幕左侧朝右侧移动，“危险情况”出现时，NPC 鱼类中 `move` 函数会利用 `math` 库中的 `abs` 函数改变 `dx` 正负，即 `dx=-abs(dx)`，利用 `abs` 函数也可以保证由左向右的转向；如果没有“危险情况出现”，则 `dx=abs(dx)`，实现 NPC 鱼的转向。对于玩家鱼，若接收键盘输入向左移动 `dx=-abs(dx)`，向右则 `dx=abs(dx)`，这样就可以保证两个转向能够互相转换。

第三个问题是玩家鱼形态升级问题，一开始将升级和形态改变分成两个部分，发现代码较为繁琐冗长，要写很多个 `if` 语句，在改进时我直接将两者合成玩家鱼类中的 `upgrade` 函数，它类似与构造函数，只是传入的参数仅有更新的等级、鱼尺寸、形态。

第四个问题就是对初始界面、游戏界面、游戏结束条件界面的转换一时难以入手。一开始实现游戏界面和游戏结束界面转换的想法是在玩家鱼碰撞到比自己等级高的鱼后，先直接用运算符 `delete` 玩家鱼指针 `playefish`，然后再加载游戏结束界面，结果都是未等到游戏结束页面加载直接闪退。后来通过在界面绘制函数 `paint` 中引入两个全局整型变量 `startgame` 和 `ifgameover` 的值并结合 `if` 语句来实现转换。如果 `startgame==0`，则说明还未开始游戏，界面停留在初始界面等待玩家鼠标响应。如果 `startgame==1` 且 `ifgameover==0`，则说明用户鼠标键入想开始游戏，则界面由初始界面转化成游戏界面。如果 `ifgameover==1`，则说明玩家鱼被比自己等级高的大鱼吃掉，结束游戏界面，加载游戏结束界面。

当然我设计的游戏也依然存在一些悬而未解的问题：

第一个问题就是由于 NPC 鱼生成的位置具有随机性，在调试过程中，会出现比玩家鱼等级高的鱼生成在周围，很可能会导致玩家鱼来不及躲避直接游戏结束，这符合真实世界的海底存在不确定性和风险性，但从游戏体验的角度上来说，这点需要改进。

第二个问题如图 5 所示，若在初始界面停留较长时间才开始游戏，会发现开始游戏后界面上生成的鱼数量极多，推测原因应该是在用户点击按钮开始游戏前，npc 鱼和玩家鱼就已经开始生成，导致当进入游戏界面时，生成的 npc 鱼和玩家鱼同时出现在初始时刻，然而自己因时间仓促、精力和能力有限没办法改进。

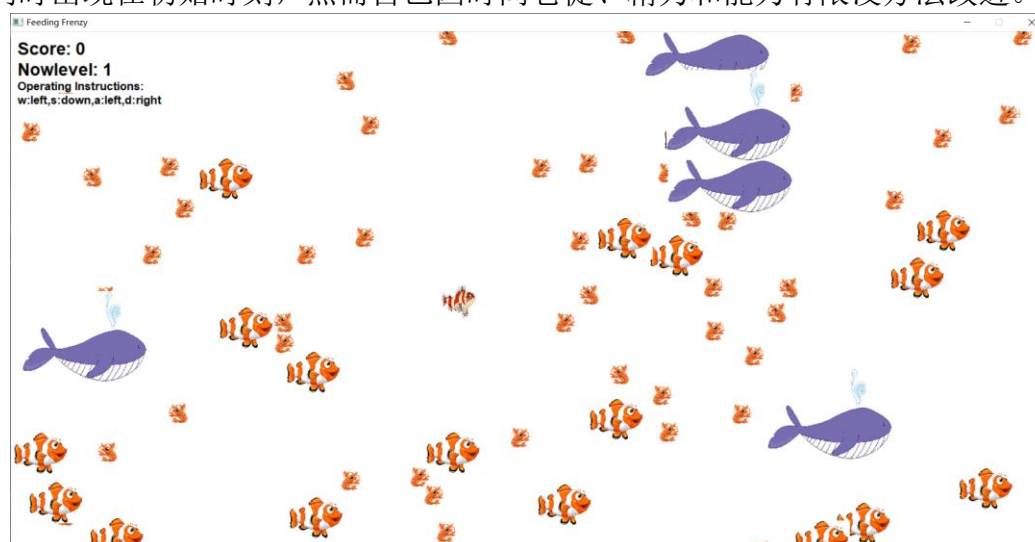


图 5

在后续还可以增加的功能还有：设置关卡，有通关机制，如分数到达某一个值或者吃完地图里所有的 NPC 鱼，玩家升级所需得分变多；丰富鱼的种类，增

加 NPC 鱼的功能，如比自己等级高的 NPC 鱼会来追逐自己；增加游戏功能，重玩，关卡选择，结束游戏后不是只能关闭程序，而是可以重新回到初始界面重新开始游戏；添加挑战模式，在规定时间内分数必须到达某一个值或者吃完地图里所有的 NPC 鱼。

无论如何，这次制作游戏过程中我收获颇丰，自己设计的游戏可以细分成很多小的“系统”，通过不同的鼠标操作按键操作去实现不同的“系统”功能达到不同的效果。利用这学期所学的 C++ 语言知识来编写程序，让我对 C++ 中较为抽象和难以理解的知识了解更清楚（例如运用纯 C 语言 ACLLIB 图形库实现 C++ 的封装、继承和多态）。通过本次实验，我了解了软件开发基本技术和工作过程以及图形界面设计基本技术，同时我的编程能力也得到了综合训练。