[Very Very Frightening Me] 设计文档

2017013624 王泓鉴

# 游戏介绍

《Very Very Frightening Me》是由王泓鉴开发的一款Windows平台的2D弹幕射击游戏。游戏画面基于OpenGL。得益于开发人员的辛苦优化，游戏具有良好的帧率稳定性，且配置要求较低，能在不同配置下流畅运行。

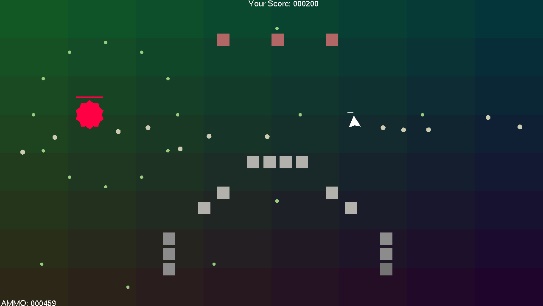
游戏单机剧情基于架空世界观。2007年，世界几乎完全处在意识入侵组织El Dorado的统治之下，未受入侵的幸存者躲在地下，希望渺茫。主角原先是浙江大学数学系的研究生。世界陷入混乱时，他加入地下实验室团队，完成了战斗无人机FutuGear Prototype的设计制作，并成为了他的遥控工程师。现在，他和他的战友要把这只无人机送上危机重重的地面，清理这片区域里El Dorado设立的机械巡逻部队，最后与躲在南极的增援部队取得联系，拯救世界。

游戏自带4个关卡，每个关卡中：在所有敌方小无人机全被击落前，Boss不主动攻击，生命回复极快，无法击毁，且会召唤新的小无人机进入战场；小无人机全部击落后，Boss开始发射三种弹幕，生命回复变慢，且停止召唤小无人机。玩家需要击毁全部小无人机后与Boss作战，从而通过关卡。

该游戏最大的特点就是高度可修改。游戏的预置关卡均写成记事本可打开的ini文件，文件中包含关卡的背景画面、地图砖块、敌军走位、刷兵条件、武器威力、主角自愈速率以及剧情文本。不仅如此，这些文件还支持注释，使得可读性非常高。玩家可以在打不过关卡时，对关卡的一些数据加以修改；也可以自编关卡乃至自编剧情。都非常方便。这给游戏增加了无限的乐趣。

# 游戏设计

## 界面设计



如图。画面左下角为剩余子弹数量；画面正上方为得分；主角的上方和Boss的上方显示各自生命值。

## 交互设计

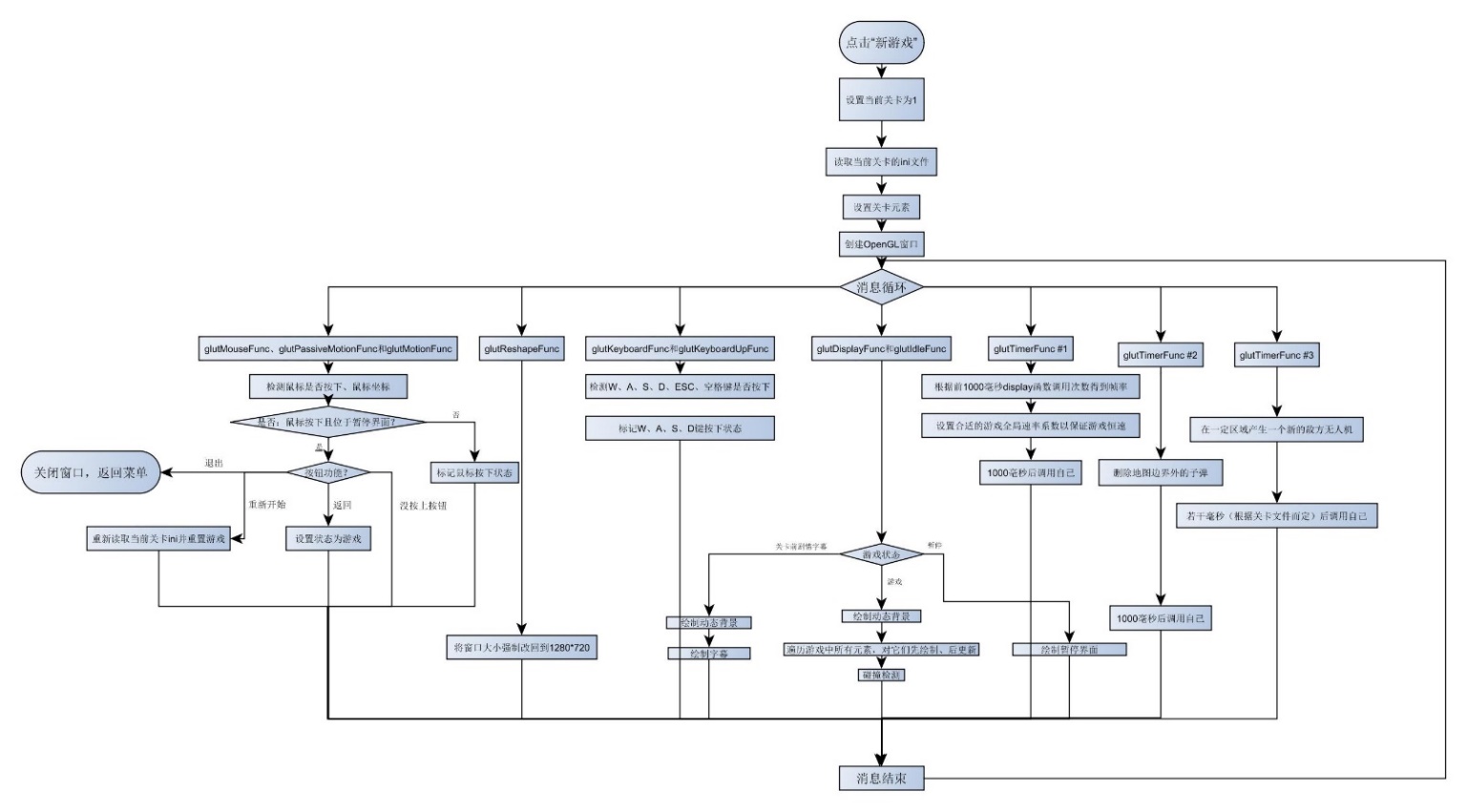
游戏通过鼠标和键盘进行操作。

按WASD键控制主角移动；按下鼠标左键不放连续射出子弹，子弹方向由鼠标位置决定；按ESC键进入暂停界面。

## C:\Users\wangh\AppData\Local\Microsoft\Windows\INetCache\Content.Word\gamer.jpg流程设计

# 游戏实现

## 代码流程

本游戏由两部分组成。一为菜单窗口程序，用Windows API实现；二为游戏窗口程序，用OpenGL Utility Toolkit实现。下面只给出后者的流程以及前者在点击“新游戏”后与后者衔接的流程。

## 核心模块

说几个有趣的模块吧。

首先是reset\_game函数（见source.cpp的1091行）。这个函数在点击开始游戏、点击重新开始、点击进入下一关时被调用。它有一个参数，表示第几关。

这个函数做了两件重要的事情：

1. 读取关卡对应的ini文件；
2. 设置三个计时器函数。

读取ini文件这件事情，是建立一个std::ifstream，然后用输入流读取对应的ini文件。

我还实现了读取文件中很重要的一件事情——跳过注释。

（见source.cpp的1072行）skip\_comment函数能够跳过以#开头的一行，使得ini文件里可以通过#进行注释，这样，可读性就提高很多了。

三个计时器函数分别是游戏速率控制函数、越界子弹删除函数、随机生成敌人函数。请注意一个细节：在reset\_game函数调用的开始（见source.cpp的1096行），我把一个叫call\_time的变量+1。这个变量是为了防止当reset\_game函数多次调用时，计时器函数也被多次调用，它们与自己发生overlap而出错的——这个现象是我偶然发现的，当OpenGL的游戏窗口关闭时，这个窗口创建时用到的计时器函数均会继续下去，所以必须在用一个call\_time变量，使得这次运行游戏的时候让上次运行游戏调用的计时器停下来。具体的实现可以参加任何一个计时器函数的实现。

接下来说说这个游戏速率控制函数fps\_control（见source.cpp的2031行）。这个函数真是经过无数的trials and errors才弄好的，里面的每一个细节都是反复调试的产物。这个事情我在第四部分慢慢讲。

因为OpenGL“在空闲时段自动调用display函数”的奇葩策略，不同的配置下帧率不同，会导致游戏速率显著不同。所以，根据帧率动态改变游戏全局速度系数就有必要了。

这个函数是一个每1000毫秒调用一次的计时器函数。它配合一个叫frame的变量，这个变量在display函数每调用一次是+1，在fps\_control函数调用时清零，这样，fps\_control就得到了这一秒的帧率。

这时候，调整一个全局变量universal\_speed\_multiplier，满足这个变量与帧率乘积近似8000。而这个universal\_speed\_multiplier则与游戏中所有跟时间有关的事情挂钩——对象的更新，射击间隔的控制……于是，这个游戏在配置迥异的机子上帧率显著不同，但是游戏还是以同样的速率跑着。

接下来说说关卡前打字幕的方法吧。OpenGL自己有一个把一个字符绘制到给定坐标的函数，基于此，自己写一个把一个std::string绘制到指定坐标的函数也就不难了。进一步，我们还可以写出把一个std::string的前k个字符绘制到指定坐标的函数（见source.cpp的1556行）。而在display函数显示字幕的这一部分，只要让这个k过一会儿增加1，就可以实现把一串字符一个个打出来的特效了（见source.cpp的1308行）！

最后说一下动态背景的实现。由于OpenGL里绘制位图极为困难，我就改为用参数控制的渐变来当背景——然后我就发现一个好处，这样一来就可以实现动态背景了！

（见source.cpp的1387行）background函数就是绘制背景的。它实质是把背景划分成8\*8个矩形，然后在每个矩形的顶点染一个色，OpenGL会自动根据顶点的颜色给这个矩形画渐变（OpenGL的这个功能太棒了!）顶点的颜色由三个东西确定：

1. 关卡ini里的数据决定的初始颜色值。
2. 关卡ini里的数据决定的RGB增量关于这个矩形坐标(i, j)的函数。
3. 鼠标连续按下或连续松开的时间。

这样，鼠标按下，画面变亮；松开，变暗。就是背景与游戏氛围动态变化的效果！

# 总结与感悟

感悟太多了！

这个游戏的框架完全是我自己写的。当然，稍微研究了一下助教给的东西，毕竟菜单界面还是win32的……

2.1k行代码，是从MSDN上一个100行不到的win32示例程序改起的。其中OpenGL相关内容是我零基础自学，然后慢慢搭建出来的。累。

最早选择用OpenGL，只是为了实现可以360°旋转的主角而已。不料在这条路上越走越远——OpenGL不支持位图绘制，于是，主角、Boss、敌军、子弹、背景……全是多边形！不过这似乎更符合我喜欢的几何极简风格，游戏看起来也酷酷的。

暑假没事翻过C++的书，学了点OOP，于是特别想在这上面试试看。所以，这个程序里我用了好多C++的东西，最主要的就是各种构造函数的重载和namespace，真的非常方便。

通过完全自己写程序，我还满足了自己的强迫症——我所有代码全部是GNU代码标准的风格。舒坦。

下面说说最想说的两个东西吧。

## 漫漫优化路

优化！优化！

OpenGL里面，实现动画显示的方式有两种——一种是写个计时器，每若干毫秒repost一下显示。我一开始想这么去做，结果发现，哪怕我把repost间隔改到1毫秒，Fraps测出来的游戏帧率也只有100左右——100fps下，游戏画面比较卡顿（太玄学了，100fps为什么会卡？我也很无奈啊）。

于是，我不得不改用另一种方式——只要系统空闲，就调用display函数，即把idle函数也设为display。

听起来是个不错的想法，确实，游戏在我的本子上跑了1600fps左右（那是刚开始做游戏不久的时候）。然而，当我试图把游戏放到机房电脑运行的时候，我发现它卡得令人怀疑人生；一位用Surface 3运行游戏的高中同学也向我反映，游戏比较慢，Fraps测出来500fps左右。

这可怎么办？我灵机一动，决定对游戏加一个全局变量universal speed multiplier，并且把[键和]键设为调速键，让玩家可以手动调速。这个全局变量与对象的移动速度成正比。

可是我很快发现，这样能调整对象的移动速度，主角的加速度、射击频率却没有变。于是，我不得不重写主角加速和主角射击的帧计数器——每帧加1不行，得每帧加universal speed multiplier。

但是我又觉得，这样让玩家每次开游戏手动调帧率的想法非常愚蠢——在机房的电脑上，我需要按将近20下加速键才能得到勉强的速度。自动检测帧率并匹配游戏速率的想法就在那时侯诞生了。

很快，我写了一个计时器，每秒收集帧率（前文已介绍），然后将universal speed multiplier改为8000/fps。效果很不错。

可是，我很快发现了bug——当在不关闭主窗口的情况下多次启用游戏时，常常会发生游戏速率变得奇快无比的情况。为了找出bug，我赶紧把储存帧数的变量打到屏幕上看它的变化——一看，才发现它经常没到一秒就被清零。我马上意识到了原因——第二次创建OpenGL窗口时，第一次的计时器并没有消失，而是和第二次产生的计时器共存，自管自调用着——它们共同清零变量frame，并共同根据frame值设置帧率。于是，bug就发生了！（这个bug很像并行编程里面的date race问题，别问我怎么知道的，我的信概大作业不是白写的2333333）

我于是对此又想了个解决方法——让每次reset游戏时使用一个不同的数值（即call\_time变量），让计时器函数按这个变量彼此区别，在后一次运行游戏时，不要调用前一次运行游戏留下的计时器函数。这样，我终于完成了一个没有bug的游戏速率优化模块。

至于内存泄漏什么的问题我似乎没遇到。我让主角无敌，然后挂机了几分钟。帧率一点没掉，内存占用也没有暴增。优化这件事，我算是放心了。

## 游戏引擎与数据分离

把游戏关卡数据分离出来，然后利用文件输入读取关卡的怪想法，最早只是为了自己更方便写关卡、调试关卡吧（为了无需重新编译就能调试关卡）。

后来，我突然意识到，允许玩家修改关卡、自编关卡，是增加一个游戏可玩性的重要方式！于是，“高度可编辑”成了我这个游戏最大的广告语。

在这一模式上给我启发的，应该是命令与征服：红色警戒2这个游戏吧。这个游戏把全局的单位数据写在了一些ini里面，把关卡、地图之流也写得非常容易编辑（很多时候大佬编地图都不用专用的编辑器）——现在想来，对modding的友好性也是这个游戏到现在仍有余热的重要原因吧——毕竟这么多人还在做mod。

印象很深的一件事是，有一次打红色警戒2里面一个在月球的关卡，困难难度，过不去。我于是学了一点游戏ini的编辑，把那关的关卡文件找了出来，把一个兵种的主武器改得强了一点——然后我就纯靠爆那个兵推图过关了。挺爽的。我也希望我写的游戏的玩家也能经历这种“爽”吧。虽然我自己实现的可编辑功能，相当有限。

一个关卡ini文件里不能只有数据，还要能注释——没注释我或许看得懂，别人来看就是天书了。注释功能的实现也费了不少劲——最后阻挠我的不是跳过用#打头的行，而是跳过空行的空格。不过，最后问题顺利解决了。我很快用写字板写了第二、第三、第四关。

我觉得分离引擎和数据是一个很聪明的想法——不管玩家对数据修改感不感兴趣，这至少让开发高效，也让代码不至于过于臃肿杂糅。

好像没什么别的想说的了。那么就这样吧。

总之，这次作业从一开始我就在尝试。我尝试OpenGL，尝试各种功能。很多东西能在网上找到参考，很多东西只能自己摸索。过程很累，尽管实现的东西跟别人比起来确实不多——连BGM都没空做。

当然最后我活着把游戏写完了。年轻人第一次完整地做一个工程，还是很开心的。~~特别是把trailer发到空间后，很多小姐姐点赞的时候。~~

# 5. 参考资料

CSDN上撰写与OpenGL编程相关内容的诸多博主，在此列举几个：

<http://blog.csdn.net/wangzhaolong87>

<http://blog.csdn.net/sxc1989>

<http://blog.csdn.net/xie_zi>

GLUT官方指南：

<https://www.opengl.org/resources/libraries/glut/spec3/node1.html>

清华大学软件学院

王泓鉴 2017013624

■