14计算机 姜峻岩

14307130166

2015年12月

Ballad Museum

《游戏开发基础》课程工程报告

**《游戏开发基础》课程工程报告**

目录

[一、 完成情况 1](#_Toc438223274)

[二、 游戏简介和操作说明 1](#_Toc438223275)

[1. 游戏简介 1](#_Toc438223276)

[2. 操作说明 3](#_Toc438223277)

[三、 工程分析 4](#_Toc438223278)

[1. 工程难点、亮点及其实现 4](#_Toc438223279)

[（1） 光学系统的构建 4](#_Toc438223280)

[（2） 极尽可能友好的人物操作 6](#_Toc438223281)

[（3） 渲染器的实现与优化 9](#_Toc438223282)

[（4） Walking On The Wall：不同重力场下的角色和机关 10](#_Toc438223283)

[（5） 关卡设计与制作 12](#_Toc438223284)

[（6） UI的初步尝试 12](#_Toc438223285)

[2. 工程的性能分析 13](#_Toc438223286)

[四、 创作手记和感想 14](#_Toc438223287)

# 完成情况

在本学期制作了游戏《Ballad Museum》的Demo版本作为游戏开发基础的课程PJ，完成了全部Demo中的游戏机制和基本的UI界面，并且制作了7个可玩关卡，其中2个是早期Indev关卡。5个正式关卡中有3个为介绍游戏机制的教程关卡。

游戏的脚本、设计、模型、UI界面均为个人完成。

模型、功能性脚本均为原创，材质部分使用来源于网络的素材。

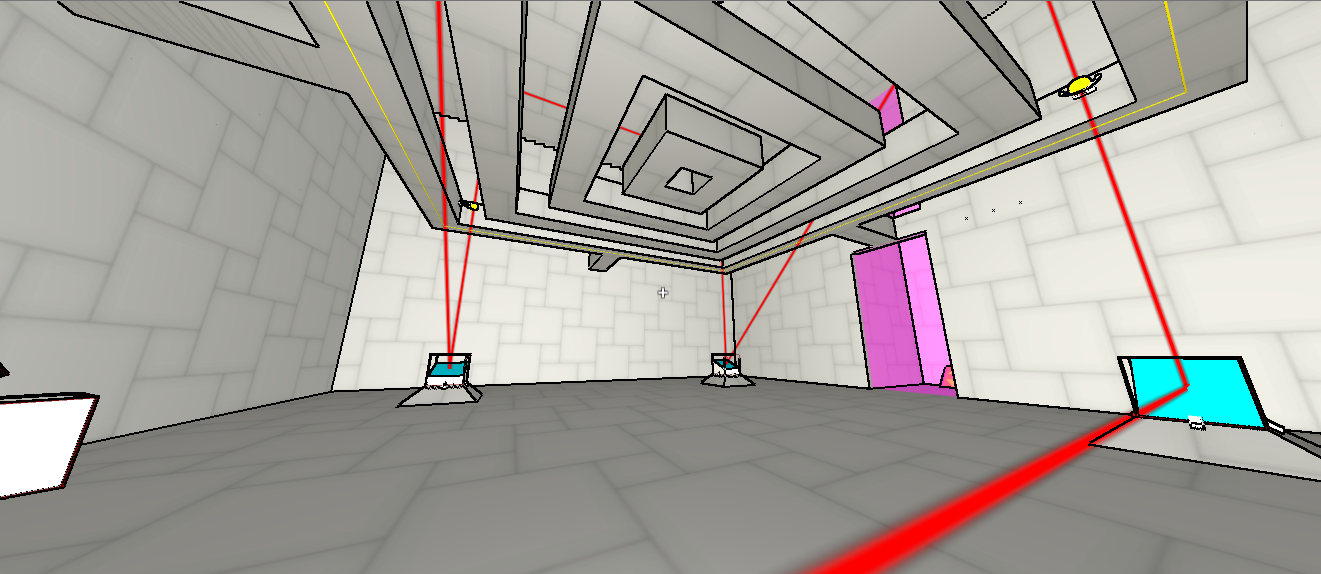
同时完成了游戏结题报告一份，以及学习渲染器时记录的笔记整理成的一篇文章《基于法线和深度信息边缘检测算法的一点研究》。

# 游戏简介和操作说明

## 游戏简介

《Ballad Museum》是一款原创的三维第一人称解谜类游戏。游戏以几何光学为基本元素，玩家可以在游戏里操控光学元件，将发射器发出的光线引导到接收器，从而触发机关并取得游戏目标。

这个游戏的创意是我在高中物理课时构思出来的，目前尚无其他解谜类游戏实现过相同的想法。游戏的原创性体现在游戏概念、交互方式以及关卡设计等多个方面，详细可见“工程分析”栏目。

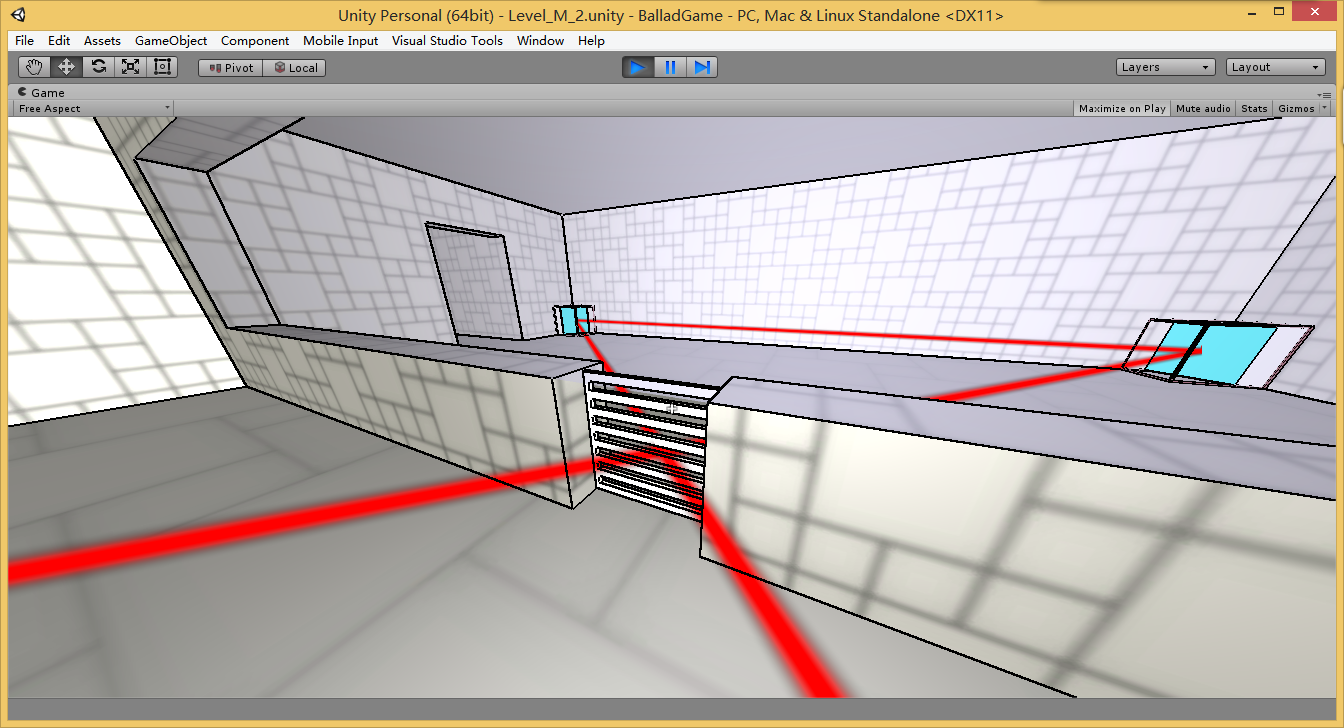
  
图1-1 游戏截图

游戏中的镜子和物理世界中的物体一样，可以在任意位置，以任意角度摆放，光线总是遵循反射定律（入射角=出射角）在镜子间反射。但为了过关，光线往往必须以特定角度出入射（如30度，45度，90度等等）。因此为了过关，镜子不能随意以任意角度放置，镜子必须要对齐放，**镜子必须要对齐放，镜子必须要对齐放**！

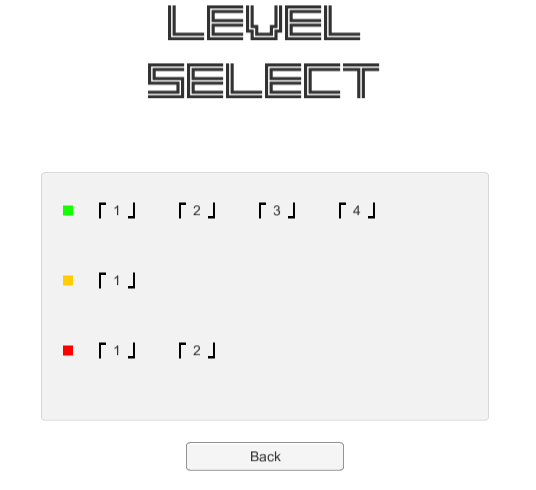
镜子对齐放的方法有两种：（1）把镜子靠墙放；（2）把镜子放在镜架（见机关介绍）上。

游戏中的实现的主要机关如下：

* 发射器：可以发出光线，目前关卡只实现了一种光线；
* 反射镜：一个带有镜面的箱子，可以反射光线；
* 光栅：只能通过与光栅栅面平行的光线；
* 接收器：接受到光线时会激活，触发机关（如开门），作为游戏目标；
* 凹面镜接收器：对光线位置敏感度更低的接收器；
* 镜架：镜子可以被固定在某一位置的镜架上，也可以被拿下来。用于光路对齐；
* 重力转换器：可以改变角色以及角色手持物体重力方向的机关；
* 紫门：只允许玩家通过，不允许玩家携带物体通过的屏障；

  
图1-2 机关介绍关卡

DEMO版本中共有8个关卡分为三个难度（简单，中等，困难），如图1-3。

  
图1-3 DEMO关卡组

|  |  |  |  |
| --- | --- | --- | --- |
| 组别 | # | 关卡名称 | 关卡内容 |
| 简单 | 1 | Tutorial | 教程关卡，介绍游戏基本操作方式以及光学系统 |
| 2 | Light Grating | 教程关卡，介绍光栅机关 |
| 3 | Walking On The Wall | 教程关卡，介绍重力转换器机关 |
| 4 | Room With No Corner | 早期测试关卡，比较简陋，同时较简单的平面地图关卡 |
| 中等 | 1 | 9 Rooms And 8 Mirrors | 具有思考量的中等难度关卡，平面地图 |
| 困难 | 1 | Hypercube | 比较难的关卡，需要重力转换器的非平面地图 |
| 2 | Ascending The Watchtower | 早期测试关卡，思考量非常大，非平面、多层建筑 |

## 操作说明

游戏操作方式会在游戏的教程关卡中介绍，当然最好还是参考如下的说明。

* WSAD（或上下左右）键移动，空格键跳跃，Shift键奔跑，Ctrl键下蹲（默认设置下）。
* 游戏界面中心有个准心。准心变黄时，表示可以捡起面前的物体。按鼠标右键可以捡起物体。再次按鼠标右键可以释放物体。
* 捡起物体的时候，在墙边（或者镜架上）按下鼠标右键可以让物体自动靠墙对齐（或安放在镜架上）放置，详见游戏。
* 捡起物体的时候，按住鼠标左键的时候移动鼠标可以朝上下左右旋转物体；按住鼠标左键的同时按EQ键可以顺时针、逆时针旋转物体。

（当然，旋转物体还有另一种方案，就是用WSAD代替鼠标移动，也就是WSADQE六个旋转控制键，玩家可以自行选择）

  
图1-3 物体旋转操作（按住鼠标左键）

* 鼠标左右键一起按，可以把手中的物体抛射出去（目前此功能用不到）。

# 工程分析

## 工程难点、亮点及其实现

游戏开发基础的课程前前后后制作了大半个学期，为了更好地把脑海中的这份对解谜类游戏特有的想法呈现出来，还是遇到过许多坎坷的。以下内容就是在工程开发中我所遇到的各种难题和想法，以及我是如何解决或实现它们的，算是对工程开发关键部分的一个全面性介绍。

### 光学系统的构建

由于本游戏的主题是光学，因此最初碰到的难点就是光学系统的构建。但是其实Unity里刚好有一个自带的系统，非常方便处理光学问题，那就是射线（Raycast）系统。

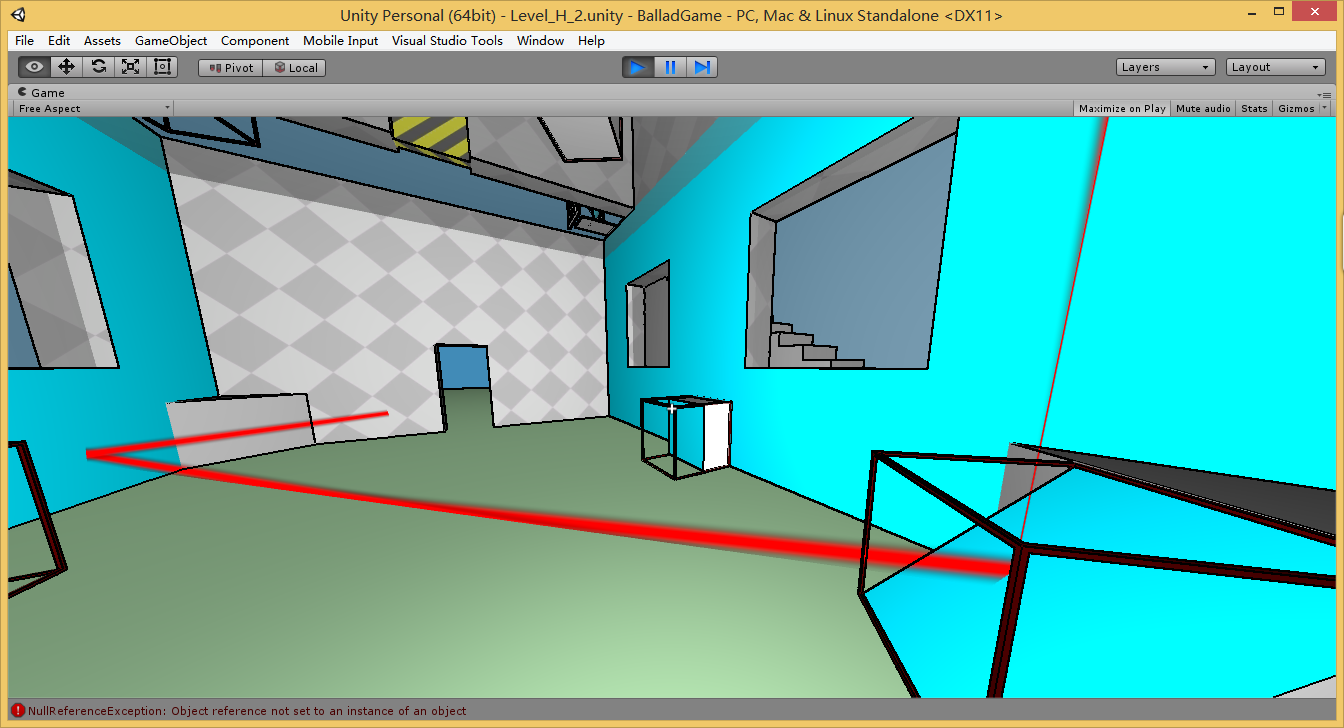
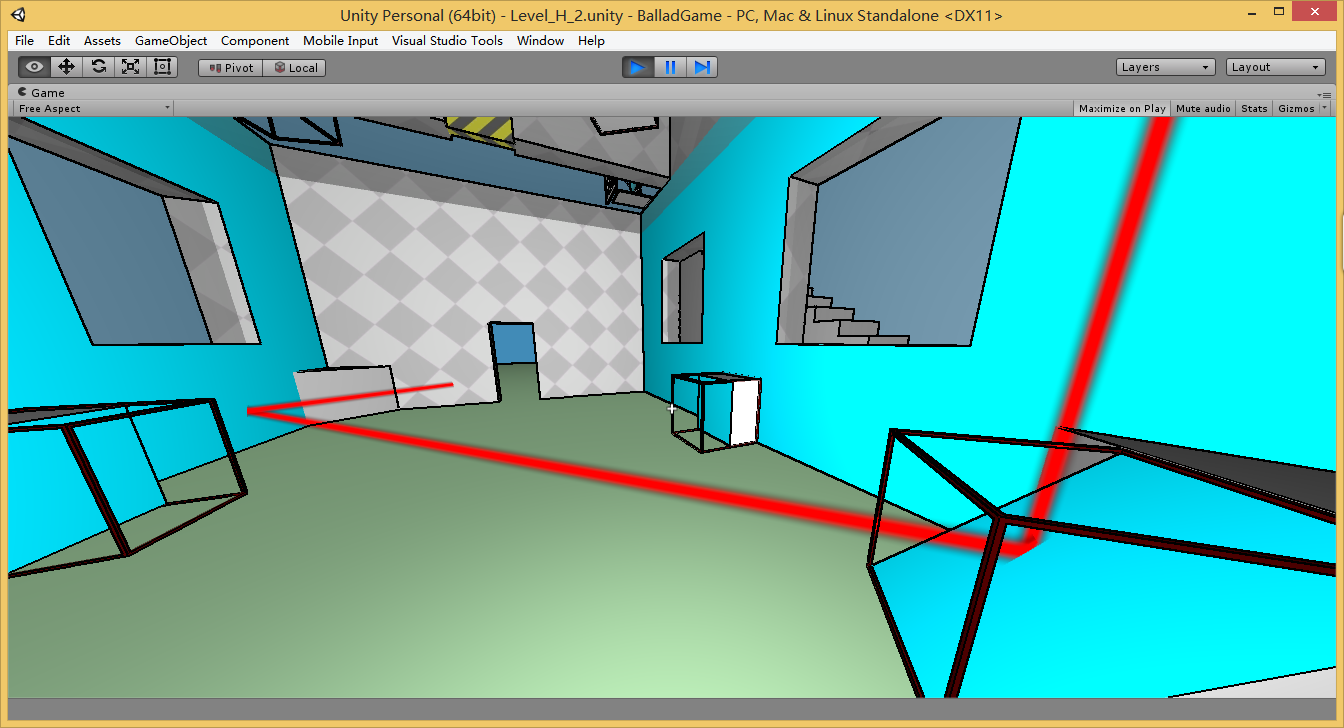
射线系统在3D游戏中用途广泛，在一般的游戏中通常用作鼠标物体拾取检测，或在FPS游戏中用作子弹碰撞检测，这个游戏也广泛使用了射线系统。但对于光学来说，它可以用来模拟光线的行为。如以下语句判断某一光线是否与障碍物体相交：



游戏涉及的光学机关很多，总的来说可以分成镜面、非镜面和透明物体。如果相交面是镜面，则按向量反射的法则，通过入射向量和碰撞面的法向量来判断出射向量的方向，继续迭代处理。如果是非镜面则光线在交点处结束，跳出迭代。如果是透明物体则直接穿过或按折射定律折射。迭代过程中获得的交点连成的就是光路。

渲染光线使用的是Line Renderer。但由于Line Renderer本身的优化实现，Line Renderer显示的曲折线条其实是一个个连续的“表面”，这些面总是很努力地把法线朝向摄像机，让其显示的表面积尽可能的大。但由于其机制原因，也存在着“露馅”的问题，如图3-1右侧渲染的光线发生了形变。

被这个问题困扰良久了之后，我发现了一个完美的解决方法。在每个光线的转折点处，在入射和出射光线靠近碰撞点的地方分别额外插入一个新的采样点，这样能使摄像机保证光线的主要部分的“表面”永远面朝摄像机，如图3-2。

  
图3-1 LineRenderer渲染问题  
   
图3-2 LineRenderer渲染问题的解决

另外由于本游戏涉及到一些各向异性的机关，在和光线交互的时候还需要对物体的方向进行判断，比如图3-3 a中是光栅机关，光需要和光栅的局部yz（蓝绿）轴相平行的光才可以通过，不然会被滤光；而图3-3 b中是发射器，光总是沿着发射器的局部z（蓝）轴方向发射。

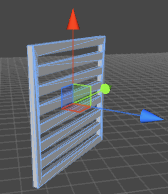


图3-3 a 光栅机关的轴与光线

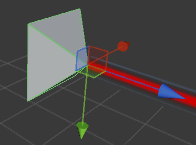


图3-3 b 发射器的轴与光线

### 极尽可能友好的人物操作

作为一个资深解谜类游戏的玩家，我个人认为，很大部分三维解谜类游戏的通病就是操作不友好。例如，物理连续世界和离散坐标世界的冲突，物理物体操作变扭，是很多解谜类独立游戏经常出现的问题。

在这个游戏的初期设计中我同样遇到这个严肃的问题。物理世界的物体都受物理定则影响，那么作为玩家，应该如何捡起一个箱子，并且把它搬到某处再放下呢？Unity自带的角色控制器并不能很好地处理这些问题，不管是FPS角色控制器还是物理角色控制器，它们和物理物体的交互太生硬了。

于是一怒之下，我自己实现了一个角色控制器。首先，它是一个刚体；其次，它可以通过鼠标移动视角；再次，它可以使用键盘移动，包含Unity自带角色控制器的所有内容。最后，它可以用鼠标左右键和物理世界进行交互。

#### **物体拖拽**

起初第一版的角色控制器，我写的用来操控物体的方式是这样的（假设面前有个箱子）：

* 左键点击箱子的时候，和箱子间创建一个Spring，当你后退的时候箱子也会后退，从而实现“拉”的功能，左键释放时Spring自动销毁；
* 当你向前靠近箱子的时候，由于你本身是刚体，会对箱子产生力的作用，所以箱子会被你推动，这就是“推”的功能。

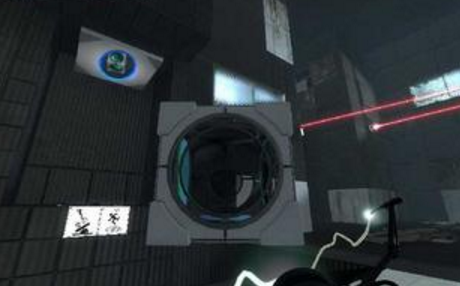
这个实现起来很简单，而且能比较灵活地移动箱子，但是有一个问题就是不能**转动手中的物体**。而这个游戏最重要的事情就是转动镜子再放在地上，不然就不能愉快玩耍了。

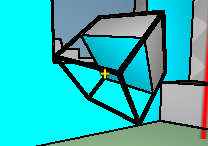
#### **物体操纵与旋转**

于是第二版角色控制器做了一个大工程，我参考了传送门（Portal）系列游戏中传送枪的行为（如图3-4 a），实现了一个非常炫酷的操作方式（还是假设面前有个箱子）：

* 右键点击箱子的时候，箱子会被“吸引”到你面前正中心，并且你走动时它会迟缓（而不是生硬）地跟踪你，如图3-4 b。
* 再点击右键的时候，箱子会脱离你的控制，并且落到地上。
* 在你控制箱子的时候，按住鼠标左键移动鼠标（或用WASDQE键）可以上下左右旋转这个箱子（详见“操作说明”部分）。

具体实现时，角色右键点击物体时将它附加到一个空物体上，采用施力的地方让它迟缓地跟踪角色，捕获旋转按键的时候执行RotateAround即可。

  
图3-4 a 传送门中的物体拾取效果

  
图3-4 b 本游戏中箱子的右键点选

第二版的操作方式相对于第一版真是极大的进步，操作友好了很多。

#### **墙角检测与自动对齐**

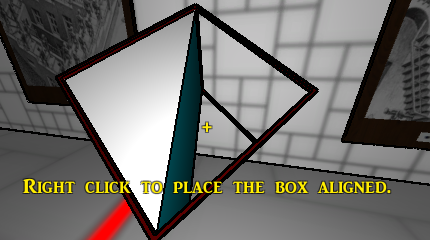
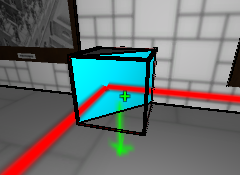
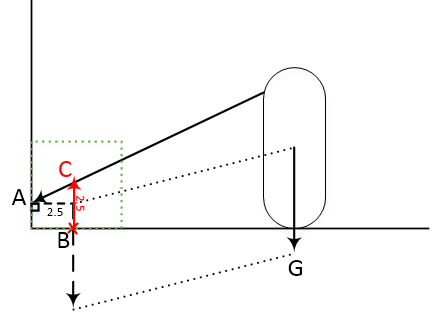
后来受另一个大型解谜类游戏的影响，让我希望在操作方面作更进一步的优化，让玩家对游戏有更好的体验。那个游戏是The Talos Principle，它在玩家拾取物体并指向特定物理机关时会给出放置提示（如图3-5 a），而不需要玩家扔下箱子后手动调整微调箱子的具体位置。而在此前，在我的游戏中，把镜子靠墙这个功能需要玩家做如下动作：（1）放下箱子；（2）用身体把箱子推到墙边。

因此在游戏制作的中期，我在第二版操作方式上加了放置物体的优化。通过一系列算法判断玩家所指向的地方是否是墙角，如果是的话，玩家可以通过右键直接把箱子靠在“墙角”放置，并且对齐放置，省去了“用身体把箱子推到墙边”这个操作，如图3-5 b，实测非常方便。

检测墙角的具体算法步骤如下（参考图3-6）：

1. 首先从摄像机中心发出一条平行于视角的射线R1，和非刚体墙面发生碰撞，取碰撞点为A，碰撞法线为N；
2. 判断R1的碰撞法线与角色重力场所成夹角是否在90°±5°范围内，若不是则表明所指处不是合法的竖直墙面，返回否；
3. 不然，从A处以N为方向移动2.5单位（箱子边长为5单位）得到点P，以点P为起点，重力场方向为方向作射线R2与地面相交于交点B；
4. 若B点不存在，或者BP距离大于2.5，或者R2的碰撞法线与重力场方向不平行，则返回否，表明所指处下方没有合法的地面；
5. 不然，从B处以R2的碰撞法线为方向移动2.5单位得到点C，则点C则是物体靠墙角时的中心点，且返回是。
6. 当角色放下物体的时候，如果所指处是合法的墙角，则取角色手中物体的Eular角度为θ1。选取24种箱子的不同靠墙摆放角度中，与角色当前持箱子的方式最接近的角度θ2，即使得|θ1.x-θ2.x|+|θ1.y-θ2.y|+|θ1.z-θ2.z|最小的摆放角度θ2作为物体的新Eular角度，以C点的坐标作为物体的新坐标，并且将物体的刚体运动速度归零。
7. 如果放下物体的位置左右有近邻的障碍物，在放置前通过射线判断再调整中心点C左右的位置，防止放置物体时卡墙等bug。

  
图3-5 a 游戏The Talos Principle截图（图中文字大意：点击鼠标右键将箱子放在这）

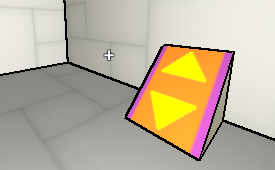
  
图3-5 b 本游戏中右键将箱子靠墙放置的提示  
  
图3-5 c 图3-5 b中按下右键后的现象，箱子自动靠墙且以合适的角度对齐  
  
图3-6 墙角判定算法原理图

在角色控制器上大做文章，事实上就是想尽力规避解谜类游戏操作不友好的诟病。解谜游戏中玩家关注的应该是谜题元素，而如果一直让玩家卡顿于生涩的操作方式上，那游戏一定有待改进。

**一款好的解谜类游戏，总应该尽力让它不成为一个“动作类”游戏。**

### 渲染器的实现与优化

此游戏里让我花费最大功夫的部分之一，就是游戏渲染器的实现。本游戏使用了一种名为法线边缘检测（Edge Detect Normals）的算法，对几何物体进行描边操作，如图3-6所示，来增加场景的纯几何性，并且弥补材质不足产生的空洞性。

  
图3-6 边缘检测Shader实现前后效果对比图

虽然Unity自带了边缘检测算法，但总体来说，Unity采用的所有边缘检测算法都不能直接使用，因为写得不好。想来想去，我还是自学了Unity的一套渲染器机制，搞明白了Unity如何创建作用于整个摄像机（Camera）的场景渲染器，自学了一些CG语言（渲染器语言果然是名副其实的最难学的高级语言之一），研究了边缘检测算法的原理，前前后后更改了多次算法实现，最终得到了比较满意的版本。

这个学习和研究的过程以及我参考的资料我放在了一篇自己撰写的文章里，它在报告的同一目录下，名称为“基于法线和深度信息边缘检测算法的一点研究.pdf”，具体内容就详见这篇文章，这里就不赘述了。

### Walking On The Wall：不同重力场下的角色和机关

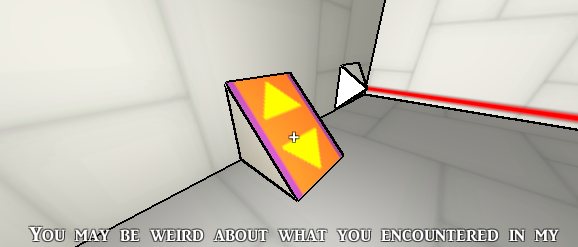
由于这个游戏在最初开题的时候我提出了一个“多重力场”的脑洞，意思就是说，我可以通过某些机关转换自身的重力方向，从而可以在竖直的墙面行走，甚至走到天花板上等等。在这个游戏的中期，我还是想尽办法实现了这么一套机制，算是一个尝试吧。

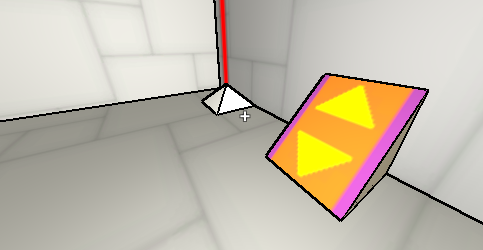
其实这个脑洞也不是第一次有，在游戏中改变重力场的启发来源于一部叫做《盗梦空间》的电影，如图3-8所示。

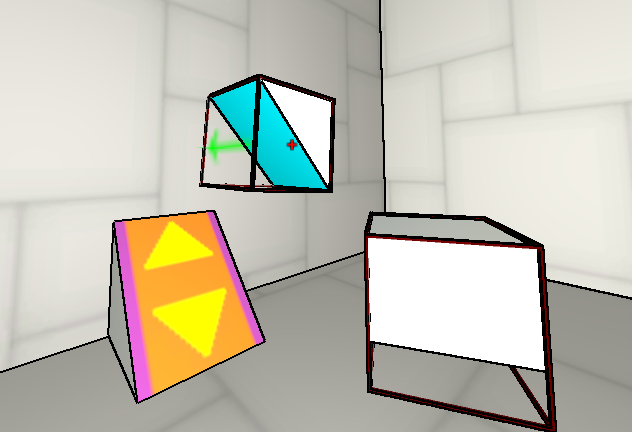
其实Unity提供了实现多重力场的基本功能。为了给每个物体不同的重力场，不能再使用刚体模式下的Use Gravity（恒定重力场）功能，而应该使用Constant Force（恒力）组件。Constant Force可以给每个刚体一个独一无二的恒力，在这里力的大小等于mg，其中m是物体的质量，g是重力加速度，力的方向等于物体的重力方向。在游戏中每个角色和机关都只有6种有效的重力方向，分别是上、下、左、右、顶、底，即六个基本的方向元向量，暂不支持任意角度的重力改变。

同时游戏里有重力转换的机关，如图3-9 a，图3-9b所示。角色踩在机关上会触发重力场变更，在完全离开机关后完成重力场变更的过程，同时角色手中的物体的重力场也会随之改变。这个机关用了一些特殊的判定方法使得它的鲁棒性很高，可以随便调戏都不会出奇怪的bug（比如丢失重力场导致人浮空），具体不展开了。

  
图3-8 《盗梦空间》电影宾馆场景中，重力场一直在绕着狭长的走廊旋转

  
图3-9 a 游戏中的重力翻转机关

  
图3-9 b 图a处改变重力后

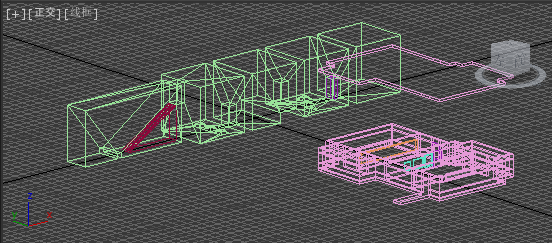
  
图3-9 c 不同重力场下的箱子，绿色的箭头表示重力场方向

### 关卡设计与制作

此游戏的关卡在后期设计的比较多，超出了课程的要求（2个左右），是因为我想通过尽量少的关卡完整传达游戏的基本机制以及游戏目标时，发现仅仅用两三个关卡并不可能完成。因此前前后后总共积累了8个可玩关卡。

关卡的设计都是我无聊的时候在纸上乱涂乱画想出来的，有可能不是很完善，但已经有一定的内容在里面。并且我也希望从关卡设计中，把整个游戏的解谜理念表现出来。

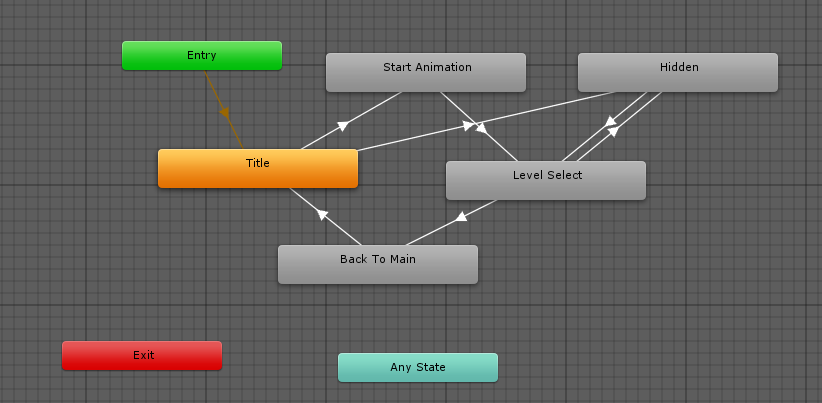
关卡制作采用的是3dsmax建模。由于没有足够精美的贴图和贴图技巧，3dsmax建完立体模型后一律直接使用真实世界坐标下的uvw贴图立方体贴图方式。建模的过程中使用较多的技术是布尔操作，以及可编辑多边形对点、面的修改。总体来说建模花掉了很大的时间和精力，虽然自学得力，但是由于缺少一些实践经验以及美术能力的支撑，做起来还是碰到了不少麻烦的，但最后在努力下还是完成了不少能玩的关卡，感觉有些许欣慰。

  
图3-10 3DSMAX中制作关卡

### UI的初步尝试

这个游戏前期和中期是没有UI的，后期我感觉这样不太好，一定要搭一个UI，即使难看到爆也要搭，不然游戏根本不像一个游戏。于是回想起Unity课初学的Animation和Animator两大系统，我使用自动机模型制作了一整个UI系统，把每个界面（如主界面、关卡选择界面、黑屏界面）分离开来处理，如图3-11。

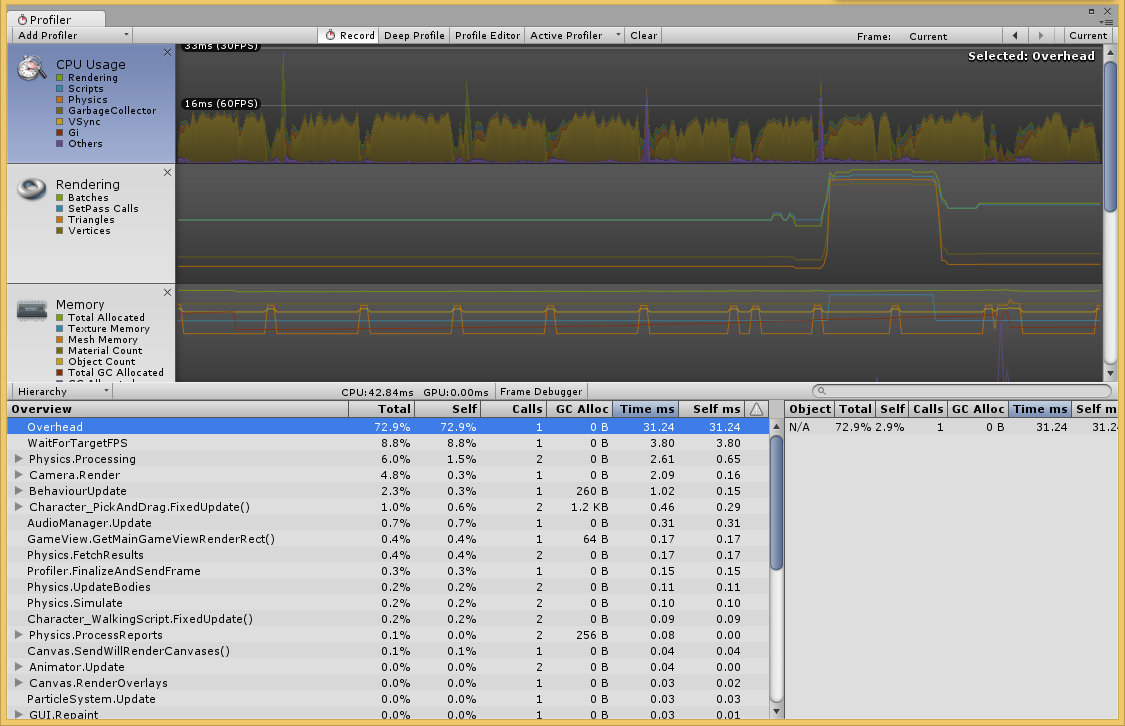
其中每一个状态对应于一个Animation，即使像Level Select这样静止的界面也使用Animation。当按下按钮或者发生事件（比如游戏里按Esc）会触发自动机的转移。Animator应用的最大好处，就是可以减少花在UI上的代码量，把不那么显然的代码转为更加直观的可视化编辑模式。

  
图3-11 UI总系统的Animator

## 工程的性能分析

在工程制作的过程中，我作了一些简单的性能分析，主要是为了提高游戏在低配机上的效率，优化程序的瓶颈。在使用了Unity自带的性能分析工具（如图3-12）之后，发现除了物理系统以外，游戏最主要的CPU消耗是放在了渲染上。因为边缘检测算法不仅是GPU的事儿，CPU也要摊上活，要把摄像机捕捉到的深度和法线信息全部传给Shader，每帧都要这样做一次。其次比较耗资源的就是光学模块，模拟光线反射需要每帧执行很多次Raycast，但也很难进一步优化了。

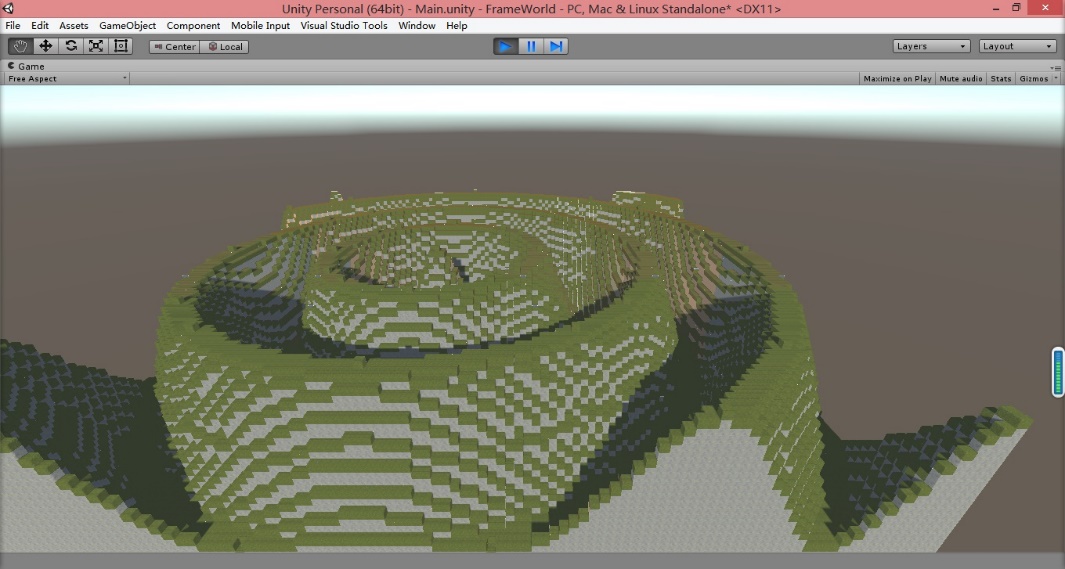
至于游戏在配置多少的电脑上可以流畅运行，我还没有测试过。但在本机的测试都非常流畅。

  
图3-12 Unity的Profiler界面

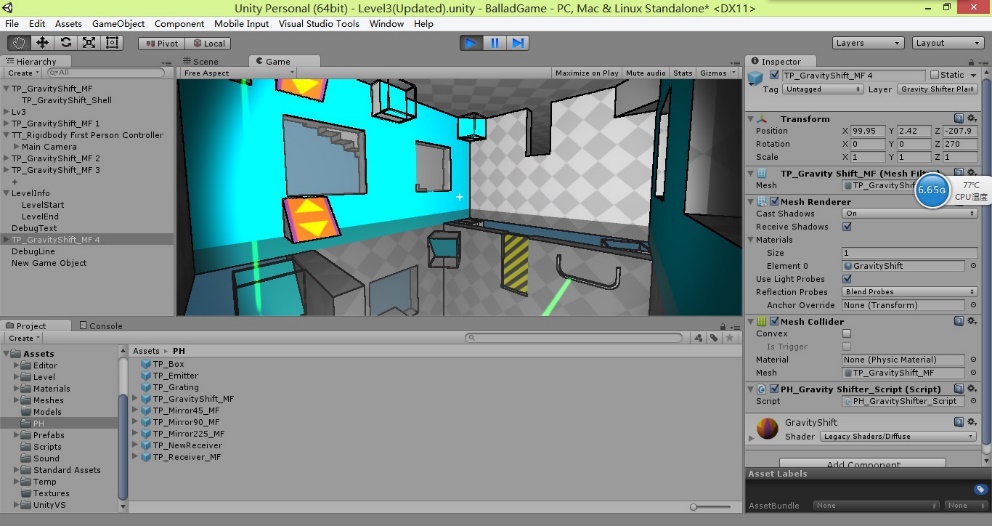
# 创作手记和感想

游戏开发课程的PJ接近尾声之际，回想从构思、尝试到制作、发布游戏的过程里，还是走了不少路来的。代码写了一千多行，模型建了一堆。虽说是花了很大的功夫和时间，但是应该说收获和所花时间是成正比的，这些都值了。

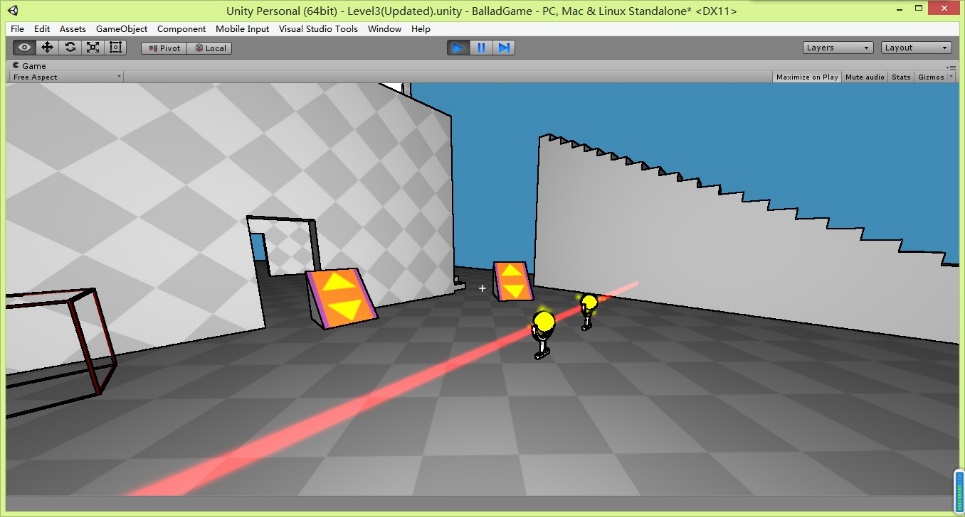
选定题目之前我还曾经想过制作一个更加困难的PJ：在Unity里制作一个Minecraft（我的世界）的克隆版，再加冒险元素进去。但众所周知，Minecraft这样的游戏涉及到很多高级的编程技巧和优化，如多线程、区块缓存等等。所以做了三天，实现到一个简单的测试（如图4-1）之后就没有继续做下去，并且选择了现在做的这个解谜游戏作为PJ选题。

  
图4-1 某不现实的PJ选题

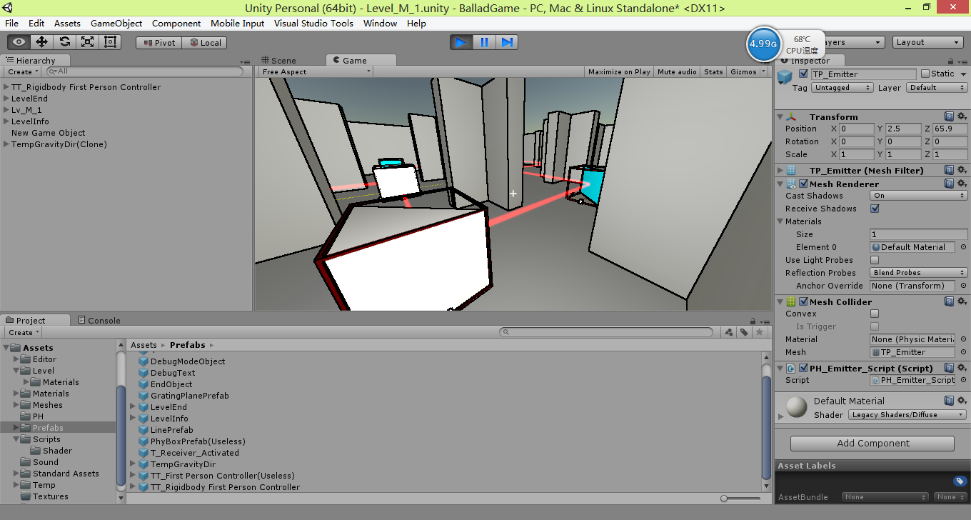
在制作的初期，由于对C#语言的不熟悉，把C#当C++来写，写了不少难以维护的代码。后来去借了本C#的书观摩了一下，学习了一些面向对象的姿势，把整个代码重构了一遍。比如说，在游戏里通用的一些静态变量用静态类Game的单一实例来维护，一些变量设置访问器，使用接口，等等。包括后来学习Shader，总的来说都是在一边学一边做，对Unity也由不熟悉到熟悉。

****  
图4-2 开发中截图：*站在天花板上！（11月14日）*

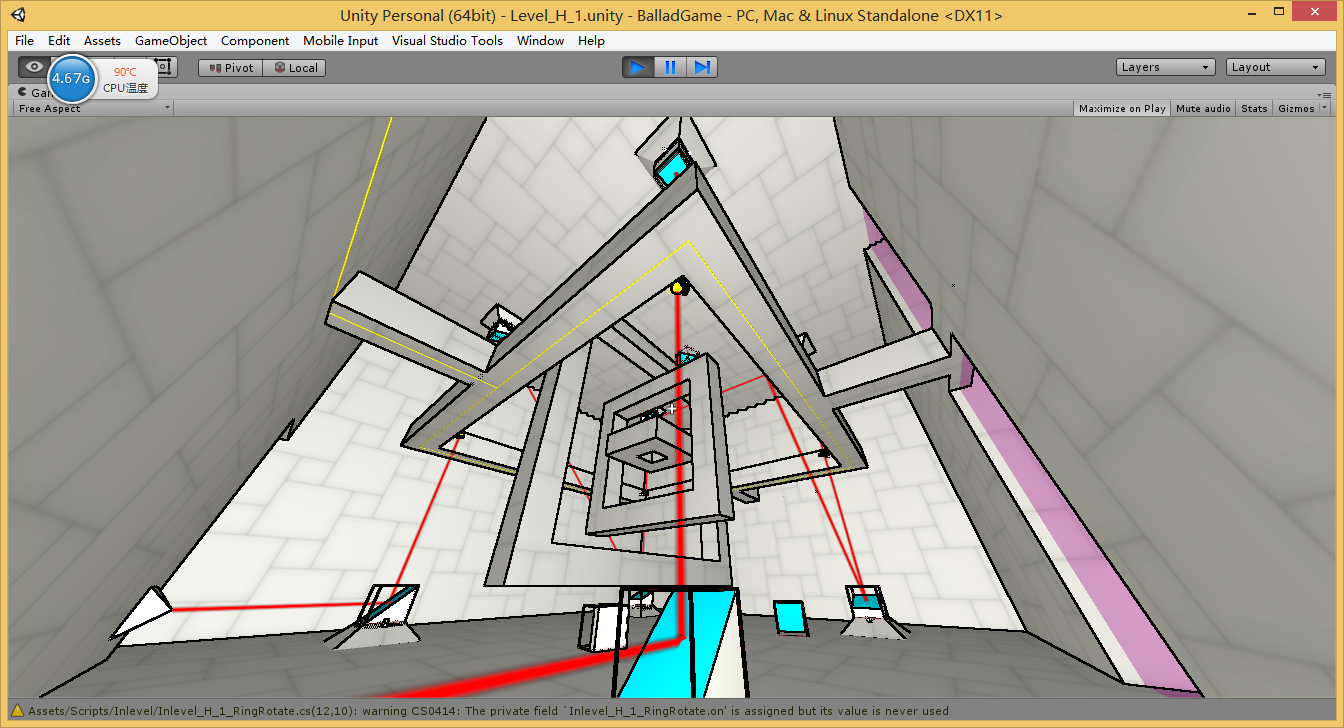
在开发的后期遇到过一些很头疼的问题，都是因为不熟悉UI系统的编写导致的，最后还是勉强制作出了一个可以用并且尚可以维护的UI系统。总体来说，这个PJ的开发历程还算顺利，没有碰到一个bug调一天之类的情况。

  
图4-3 开发中截图：*移动的路面！（11月19日）*

本来希望给这个游戏加入剧情的。具体的剧情在游戏进行中逐渐清晰，主角作为一个训练中的强人工智能，游戏中的“上帝”允诺给它将人类知识的博物馆带到他的面前，前提是主角通过了一道道谜题机关所组成的测试。事实上在某些关卡里，我已经尝试将“上帝”对主角说的话作为字幕显示出来了。可惜因为时间紧迫，不能把一个完整的剧情加入游戏，所以暂时就只有解谜部分了。

  
图4-4 开发中截图：*创造光路（11月24日）*

游戏名字起成《Ballad Museum》，也并不是一个深思熟虑的名称，至少第一个单词不是，不用在意标题本身的中文译文或者内涵。其实想通过艺术字化的标题来反映游戏内容：Ballad Museum两个单词的艺术字都是轴对称图形（详见开题报告第一张PPT），可以反映“反射”这个概念。然后Museum跟剧情相关，当然现在游戏不存在剧情。当然啦，很多游戏在demo之后都会改名字。

  
图4-5 开发中截图：*一个漂亮的关卡（12月8日）*

虽然说从开学开始到现在，做了那么长时间，但对游戏的最终效果而言，还没有达到我所想象的程度，它暂时只能作为一个阶段性的demo先提交。

我说“暂时”这个词，是因为我不会因为课程结束就停止对游戏的开发。我很早就接触了游戏设计，并且在业余时间，当周围人都在玩大型射击类、策略类或是动作类游戏的时候，我却经常会一个人被一些益智类或解谜类游戏的创意所吸引——或许这就是后来我走上信息竞赛道路，参加OI和ACM的原因。而由于创意好的解谜类游戏在当时并不多见，当时也不了解“独立游戏”这样一种存在形式，玩到的好游戏不多，反而自己有时会萌生出一些游戏的点子。

到大学里接触了Unity之后，便很迫不及待地希望能够自己做点什么出来。但其实在这个过程中我意识到，游戏的开发不像竞赛题一样可以在短期内解决，而是像一个大型工程，需要一段时间从地基开始建起。**并且，独自一人开发游戏的效率是非常低的。一个好游戏背后总有一个由志同道合的开发者们组成的团队。**

而我在这方面也有一个愿望，如同在ACM比赛中获奖的愿望一样，我希望能在大学生涯里制作一个真正意义上的完整游戏作品，能够去申请中国独立游戏节，甚至世界独立游戏节学生作品的提名。



当然，走到这种高度，必不可少的是足够的脑洞、足够的编程经验以及从音乐到美工各方面的完美配合。同时，也需要一种持之以恒的精神。

回首过去，感谢这门课程，让我有了这样一次静下心来学习制作游戏的机会。

展望未来，希望在学业之余，能够寻找志同道合之人，一同把心目中那份唯美的想法展现出来。

# 使用库及参考资料

1. First Person Controller参考了Standard Assets中Character Controllers资源包（Unity 4.x版本），使用了MouseLook脚本；
2. Edge Detect Shader编写时的参考资料见《基于法线和深度信息边缘检测算法的一点研究》。

部分参考资料

1. Unity3D