JILIN UNIVERSITY

C语言项目

项目名称：天体运行游戏

项目成员：张益通 张铭贤 于大河

填写日期：2020 年 9 月 7 日

摘要

随着科学技术的发展，人们迎来了宇宙探索的黄金时代。设计这样一个小型的宇宙模拟游戏可以让玩家了解宇宙中各种美丽神奇的天体和物理现象。玩家可以自己创造宇宙中的天体，观察宇宙在万有引力下的演化方式。此游戏旨在提升人们对宇宙的兴趣和认知，同时享受探索宇宙的乐趣。此游戏的创新之处在于使用了 Cocos2d-x 引擎和 Box2D 物理引擎进行开发，在有一定趣味性和可玩性的同时具有强大的真实物理模拟功能。玩家可以调节多种参数，也可以创建多种星体，我们也设计了“黑洞”这一特殊星体来模拟一些较为极端地情况，使得整个游戏的物理背景更加丰满。

1 问题描述

此项目的实现是具有一定社会时代背景的。在社会快速发展的今天，许多人面对快速变化的生活变得功利，浮躁，社会上急需对于科学知识的尊重与重视；同时游戏的审美也不应当仅仅局限于一味地追求刺激和即时的快感，而也需一些宁静平和的趣味。基于这一社会时代背景，我们小组希望能编写一款能融入物理知识的游戏，让人们在体验游戏的乐趣的同时感受到知识的魅力和内心的宁静与平和。

本项目在 Visual Studio 平台上使用 C 与 C++ 进行开发，游戏聚焦于模拟宇宙天体的运动。对于技术实现手段主要使用了 Cocos2d-x 游戏引擎以及 Box2D 物理引擎。本游戏具有以下特点：

科学属性：该天体物理游戏具有浓厚的物理背景。充分借助物理引擎，依靠牛顿动力学规律，最大程度地真实模拟宇观尺度下的物理实际。

多种功能：本游戏中为玩家提供了多种可调节不同参数的宇宙星体，通过适当控制星体运动自由度，保证物理情景模拟的同时具有可玩性。玩家可控制星体的质量，半径，初速度，设置行星黑洞等多种星体，并实现周期运动、碰撞等具体场景。

探索宇宙：通过切实模拟宇宙星体的运动轨迹和碰撞规律，激发玩家对于物理知识的兴趣，顺应新科学技术发展期人类对于宇宙探索的潮流。

2 组内分工

组长张益通：

主要负责技术路线选择、开发环境搭建、场景编写和游戏架构设定等，工作量占比约40%。

组员张铭贤：

主要负责界面设计、特效处理、数据临时存储，工作量占比约30%。

组员于大河：

主要负责物理建模与实现、游戏设计优化、引力算法选择，工作量占比约30%。

3 项目分析

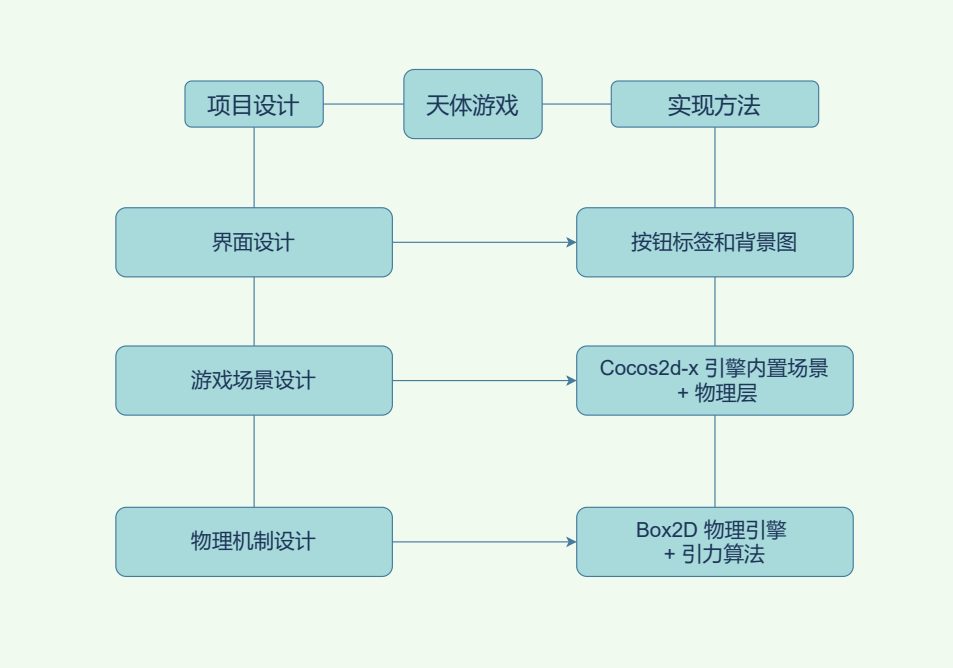
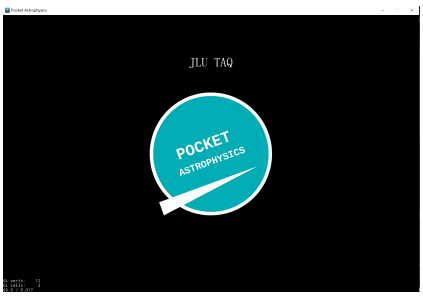


图 1: 项目总体架构.

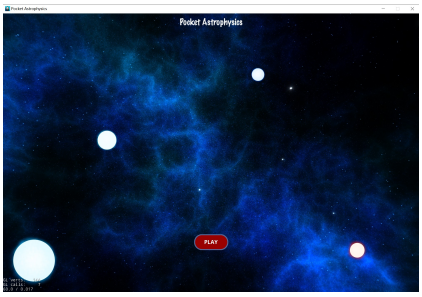
此天体游戏的框架设计主要包括以下三个模块：界面设计，游戏场景设计和物理机制设计。

界面设计部分主要完成的内容是资源文件的选择和插入。按钮，标签和背景图资源均来自网络。运用自动 logo 生成器制作了我们的 logo，按钮的制作过程中使用了 PS 软件。部分按钮采用两张图片，即未选中与选中，这样点击时可以产生明显的效果。背景图采用的是高清宇宙星空，符合游戏主题。背景音乐我们选择了贝多芬的月光奏鸣曲，平静柔和的音乐很好地营造了宇宙天体游戏的氛围。

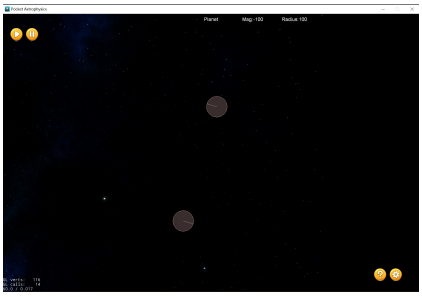
游戏场景设计部分需要设定各个场景的内容以及相互之间的调用关系。我们的游戏分为以下6个场景：Welcome Scene, Hellow World Scene, Main Scene, Setting Scene, Music Setting Scene, Help Scene. Welcome Scene 是第一个场景，包含了我们的 logo，点击即可进入 Hellow World Scene. 这是一个主要用动画实现的场景，展现星体运行，反映游戏主题。点击 Hellow World Scene 中的 Play 按键即可转入游戏的主场景 Main Scene. 主场景用



(a) Welcome Scene



(b) Hello World Scene



(c) Main Scene

(d) Setting Scene

图 2: 游戏主要场景

来实现游戏的主体功能，其中加入了两层物理层 Physics Delegate 和 Physics Layer ，分别用来初始化世界物理属性和获取点击响应事件。主场景中加入了游戏的暂停、继续、设置和帮助按钮。点击帮助按钮(小问号)可以跳转到 Help Scene, 对游戏操作方法进行说明；点击设置按钮可以跳转到 Setting Scene. Setting Scene包含4个标签按钮，分别是 Resume 继续游戏，Restart 重新开始，Setting 设置(点击跳转至 Music Setting Scene 进行音量调节)，以及 Exit to Menu 点击将回到 Hellow World Scene.

物理机制的设计较为复杂，将在模块设计部分重点讨论。

4 模块设计

为了更好地实现预期中的游戏效果，我们组创新性地选择了 Cocos2d-x 引擎结合 Box2D 物理引擎进行项目开发。Cocos2d-x 引擎提供了许多内置的关于节点管理和场景调用及初始化的方法，而 Box2D 物理引擎则内置了强大的物理模拟方法。我们项目的关键就在于将这两个引擎很好地结合，实现模块化编程。以下挑选出三个重点模块进行分析：物理机制实现方法、场景切换与储存和节点的管理。

4.1 物理机制实现方法

为了创建具有物理属性的实体 Planet 和 Black Hole 我们首先设置了一个抽象类 b2Node(物理节点), 它继承自 Cocos2d-x 内自带的 Node 类。为了统一管理这些物理节点，我们又专门新建了 b2NodeManager (物理节点管理类)，然后我们还设置了 b2PhysicsObject 类(继承自 b2Node)，用于设置实时刷新物体位置，最后是 Planet 和 BlackHole 两个实体类，用于创建行星和黑洞的实体以及初始化一些特有的物理属性。

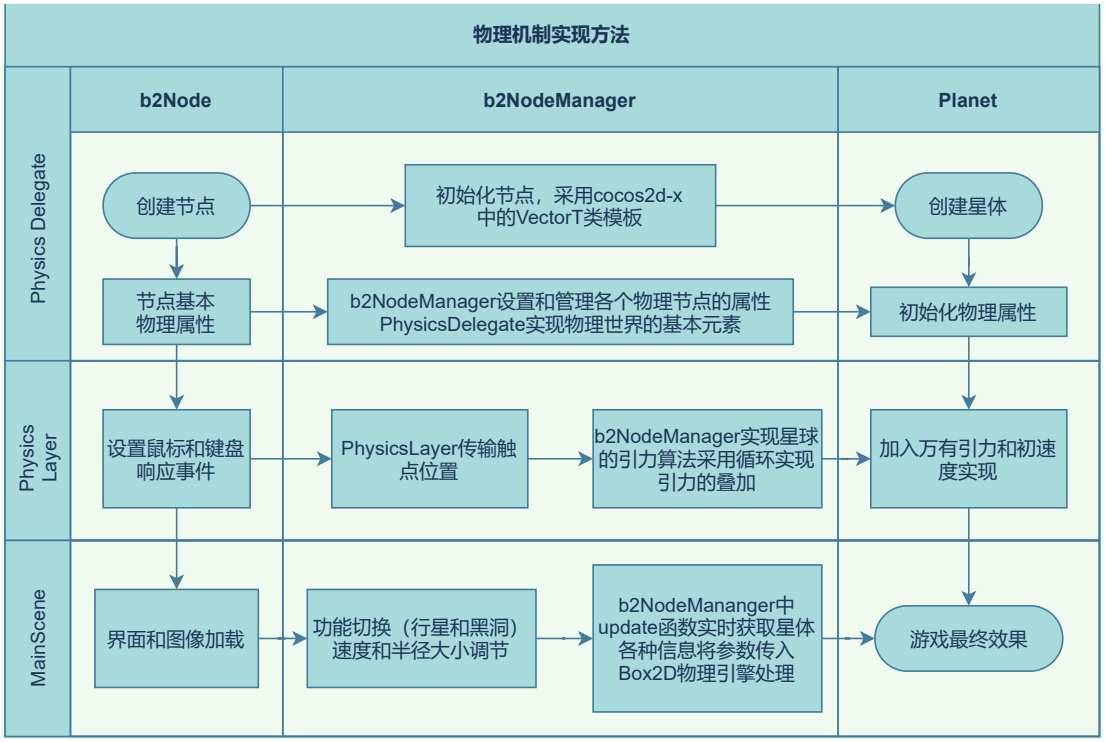


图 3: 物理机制的实现方法.

对于构建的物理世界和设置事件响应以及呈现效果，我们自下而上设置了三个层：最底层 PhysicsDelegate 用于设置物理世界的基本属性，包括重力和物理实体图像的绘制方法(绘图采用了开源的第三方 Cocos2d-x 和 Box2D 绘图库 GLES-Render 采用这种方法可以大大减少在场景中加载图片的内存占用，从而提升游戏的性能，可以同时画几十个星体而不出现卡顿)。PhysicsLayer 层用来设置鼠标和键盘响应事件。最终的界面效果将呈现在 MainScene 中。

下面以在 Planet 模式下点击创建一个星体并受到其它星体的作用为例阐述具体的实现方法。首先，点击事件在 PhysicsLayer 层中被接收，并将星体类型、触点位置、半径大小、初速度等数据传送至 b2NodeManager, b2NodeManager 将判断目前的星体类型模式 (Planet 或 Black Hole) 并利用相应星体中的 create 方法在相应位置创建星体(物理节

点)，并将其节点添加到 Vector<b2Node*>(Vector<T>是 Cocos2d-x 3.x 推出的列表容器，它所能容纳的是 Ref 及子类所创建的对象指针，其中 T 是模板，表示能够放入到容器中的类型。Vector<T>是模仿 C++ 的 std::Vector<T>模板类而设计的。它的内存管理由编译器自动处理，可以不用考虑内存释放问题)中以管理该节点。同时，在 Planet 类中初始化该节点的物理属性，赋予质量、半径等参数。星体的初速度也是在 Planet 类中以冲量的方式实现。最后核心部分是给每个星体加入万有引力，并使其在相互最用下运动。首先由 b2NodeManager 中的引力算法循环实现对每个星体的引力叠加，引力的大小和方向是由 Planet 类中的 AddGravitationp() 根据牛顿万有引力公式 $\boldsymbol{F} = \frac{GMm}{r^2}\frac{\boldsymbol{r}}{r}$ 生成的，需要传入与其他星体的距离和对应质量两个参数。然后由 Planet 类中的 Processp() 函数使用 Box2D 物理引擎的内置方法 ApplyForce() 和 SetTransform() 对星体实现力的作用效果。Planet 类中实现任意初速度的方法：速度方向——将点击开始和结束时产生的向量的方向作为初速度方向；速度大小——实际实现过程是时在一次刷新时间里给星体加上正比于玩家定义的速度和质量的乘积的力，这样就以冲量的方式实现了初速度。最后在 PhysicsLayer 中用 update() 函数调用 b2NodeManager 的 update() 函数，再在主场景 MainScene 中调用 PhysicsLayer 的 update() 函数，就实现了层之间的叠加。

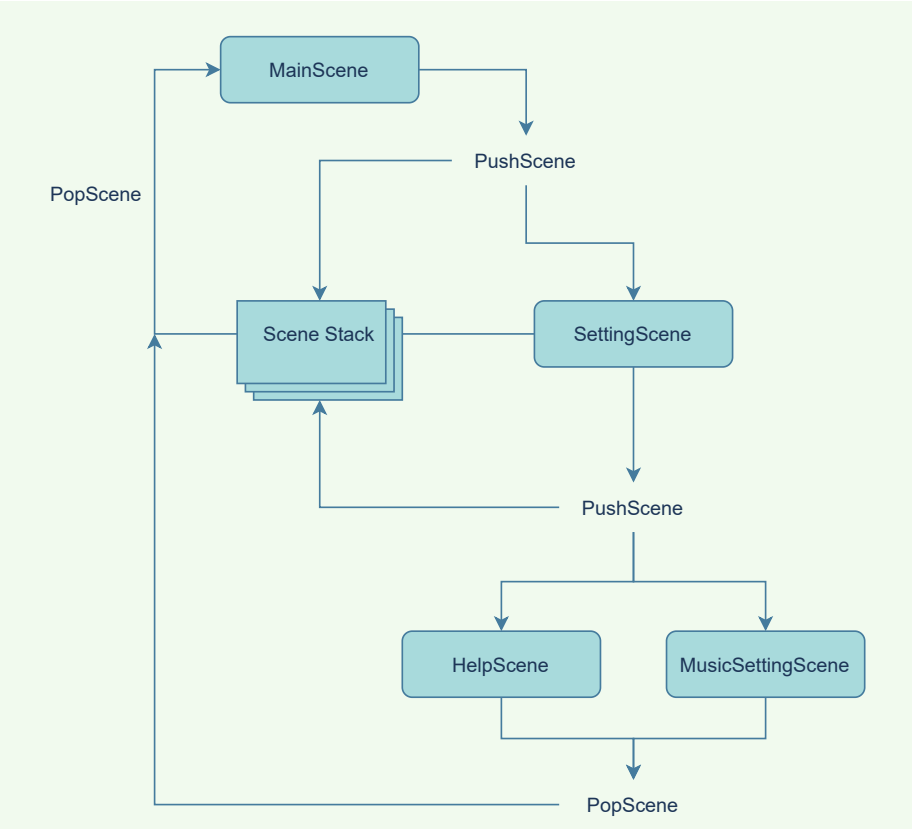


图 4: 场景切换与储存.

4.2 场景切换与储存

场景的编写主要参考了 Cocos2d-x 引擎中的 HelloWorld 场景，一个场景通常包含三种方法，分别是 create 创建场景，init 初始化场景，callback 回调函数。init 用来实现场景中的各种内容。回调函数可以用来设置点击响应事件。场景之间的互相调用是通过点击按钮触发回调函数实现的。

Cocos2d-x 提供了场景之间互相调用的方法，使用 replaceScene() 可以退出当前场景并创建新的场景，还可以套用 Transitionfade() 等方法添加转场的特效。但不足之处在于使用 replaceScene 会首先释放当前场景，无法保存场景。于是，我们换用了 pushScene() 方法，其原理是利用一个栈将场景储存起来，调用 pushScene() 时，首先将当前场景存入栈中，再创建另一个场景。相应地，利用 popScene() 方法可以首先释放当前运行的场景，再弹出位于栈顶的场景并运行，这样就实现了场景的临时存储，实现了游戏的 Resume 功能。

4.3 节点管理

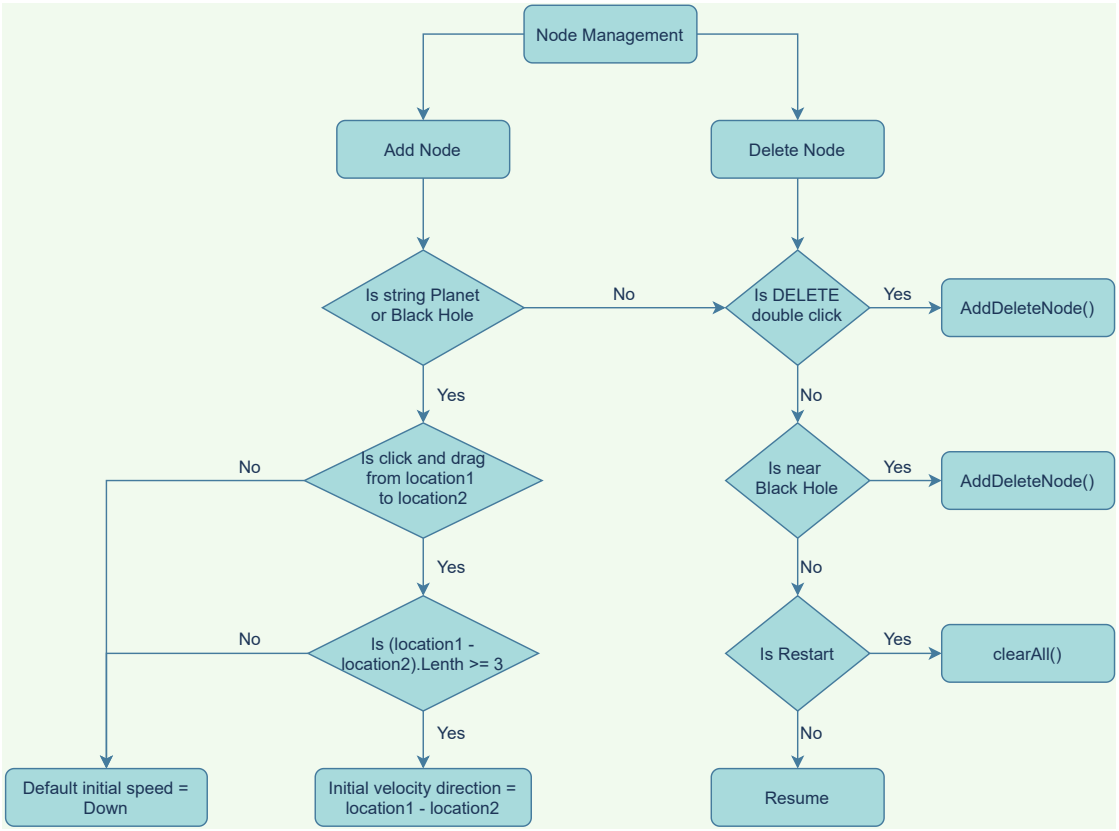


图 5: 节点管理.

此游戏中的各种星体本质上都是 Cocos2d-x 中的 Node (节点)。玩家在游戏中可能需要

创造出相当多的节点，并可以选择任意删除。这些节点根据物理定律相互作用，如果星体掉进黑洞还会消失。同时，Restart 和 Exit to Menu 的功能也需要清除所有物理节点，否则再次进入游戏将产生奇怪的错误。可见，在本游戏中统一地管理所有物理节点是非常重要的。b2NodeManager 就是用来专门管理这些节点的。下图反映了添加和删除节点的流程。

添加节点时，首先判断字符串是否是星体类型，如果否，就将添加删除节点；如果是，再判断是否点击并拖动，如果只是单击或者拖动距离小于三个像素，都默认星体初速度方向向下，否则将点击开始和结束时产生的向量的方向作为初速度方向。在删除节点时，同样需要先判断字符串内容是否是 DELETE 并鼠标双击了节点所在框区域，如果是，就将该节点添加入删除节点，否则继续判断星体是否接近黑洞区域，如果是，就将该节点添加入删除节点。最后一个判断是如果重新开始则会清除所有物理节点，防止对重新创建的游戏场景造成影响。

5 代码实现

由于本游戏采用的是 Cocos2d-x 引擎和 Box2D 物理引擎进行开发，许多代码的架构都建立在引擎的源码之上。总代码行数较多，但涉及底层代码不多，且如场景的代码实则为一些模板，都基本上大同小异，所以只挑选一些重要函数模块进行分析展示。

```
36      auto closeItem = MenuItemImage::create(
37          "button_play.png",
38          "button_play _s.png",
39          CC_CALLBACK_1(HelloWorldScene::menuCloseCallback, this));
40
41      if (closeItem == nullptr ||
42          closeItem->getContentSize().width <= 0 ||
43          closeItem->getContentSize().height <= 0)
44      {
45          problemLoading("'button_play.png' and 'button_play _s.png'");
46      }
47      else
48      {
49          float x = origin.x + visibleSize.width / 2;
50          float y = origin.y + closeItem->getContentSize().height / 2 + 200;
51          closeItem->setPosition(Vec2(x, y));
52      }
53
54      // create menu, it's an autorelease object
55      auto menu = Menu::create(closeItem, NULL);
56      menu->setPosition(Vec2::ZERO);
57      this->addChild(menu, 1);
```

图 6: 场景中添加图片按钮.

Cocos2d-x 提供了许多在场景中加入按钮的方法，以上是以图片的形式加入按钮。分为以下几个步骤：以图片创建按钮(MenuItemImage::create)，设置按钮位置(setPosition())，

```
60      b2Vec2 force = b2Vec2 (0, 0);
61      float fx, fy, sum;
62      fx = Planet::location2.x - Planet::location1.x;
63      fy = Planet::location2.y - Planet::location1.y;
64      sum = sqrt(fx * fx + fy * fy);
65      if (sum >= 3)//防止sum过小产生数值爆炸
66      {
67          fx = fx / sum;
68          fy = fy / sum;
69          force = b2Vec2 (-fx, -fy);
70      }
71      else
72      {
73          force = b2Vec2 (0, -1);
74      }
75      force.Normalize();
76      float mag = b2NodeManager::getInstance()->mag;
77      force = mag * _body->GetMass() * force;//以冲量方式实现初速度
78      _body->ApplyForceToCenter(force, true);
79  }
```

图 7: 初速度的实现方法.

以及添加回调函数(menuCloseCallback())。

在 Planet 类中，实现任意初速度的方法：速度方向——将点击开始和结束时产生的向量的方向作为初速度方向；速度大小——实际实现过程是时在一次刷新时间里给星体加上正比于玩家定义的速度和质量的乘积的力，以冲量的方式实现了初速度。值得注意的是，在计算单位向量时，一开始我们并没有 if (sum >= 3) 的判断，在运行时就出现了一些奇怪的现象，后来经过分析发现是作为分母的数值如果过小则会数值爆炸，产生错误。这样就增强了程序的健壮性。

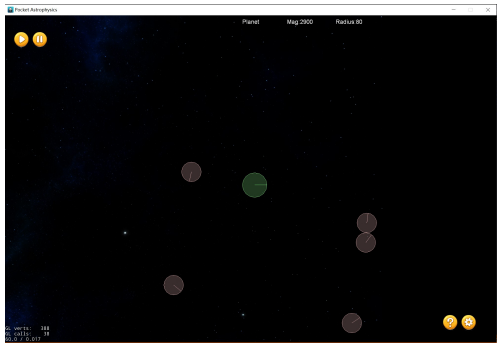
```
92  Planet * planet;
93  Planet* planetj;
94  BlackHole * blackhole;
95
96  //研究第i个星球
97  for (int i = 0; i < _planets->size(); i++)
98  {
99      planet = (Planet*) _planets->at(i);
100     b2Vec2 positionp = planet->getPhysicPosition();
101     planet->ClearGravitationp();
102     for (int j = 0; j < _planets->size(); j++)
103     {
104         if (j != i)
105         {
106             planetj = (Planet*) _planets->at(j);
107             b2Vec2 distancepj = planetj->getPhysicPosition() - positionp;
108             //CCLOG("%d", distancepj);
109             planet->AddGravitationp(distancepj, planetj->getMass());//施加其他星球产生的引力
110         }
111     }
112     for (int k = 0; k < _blackholes->size(); k++)
113     {
114         blackhole = (BlackHole*) _blackholes->at(k);
115         b2Vec2 distanceb = blackhole->getPhysicPosition() - positionp;
116         planet->AddGravitationp(distanceb, blackhole->getMass());//施加黑洞产生的引力
117         //CCLOG("%f   %f\n", distanceb.LengthSquared(), (planet->getRadius() + blackhole->getR
118         if (distanceb.LengthSquared() <= 1.1 * (planet->getRadius() + blackhole->getRadius())
119             * (planet->getRadius() + blackhole->getRadius()))
120         {
121             this->addDeleteNode(planet);
122         }//星球掉入黑洞被删除
123     }
124     planet->Processp();
125 }
```

图 8: 添加引力算法.

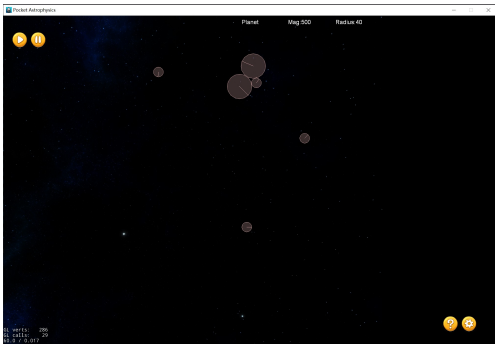
b2NodeManager.cpp 中，星球的引力算法采用循环实现引力的叠加，同时考虑了其他所有星球对某个星球的引力和所有黑洞对其的引力，从而产生互相吸引的效果，同时通过判断与黑洞的距离决定是否掉入黑洞并删除行星节点。

6 测试效果

在整个项目过程中，我们不断对游戏进行各种调试，出现过许许多多的问题，主要是一些内存访问冲突和节点管理的问题，基本上都顺利地通过各种途径解决了。在答辩的过程中，我们也根据助教提出的建议进行了改进，加入了游戏帮助界面(Help Scene)，方便玩家熟悉游戏操作，增强了游戏的可玩性。最终的项目经过多次测试，运行较为稳定，可以同时模拟几十个星球的相互引力作用而不发生卡顿，基本上达到了预期的效果。下面是一些项目运行截图。

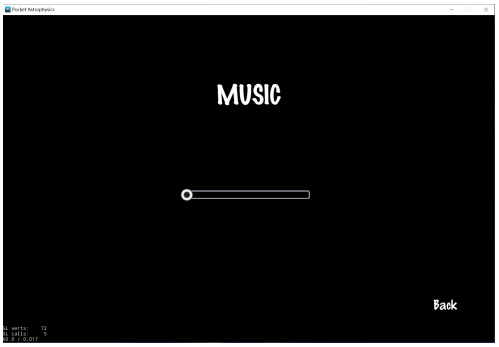


(a) Black Hole



(b) Many Planets

(c) Help Scene



(d) Music Setting Scene

图 9: 游戏截图

在我们项目进行到中期的时候，曾出现过不少 bug，在向优秀的其他同学提问的同时，我们也在积极地解决问题。最终，凭借其他同学的指导与我们组的不断尝试，修复了一个又一个漏洞，在这里我们要对他们表示感谢！在游戏开发的过程中我们也对最初的设想进行了不断地删减与补充，例如我们删除了对于后期星球发展出生命可能性的估算，我们也添加了例如拖动创建星体等新的功能，使得游戏的整体框架变得更加合理完善。