



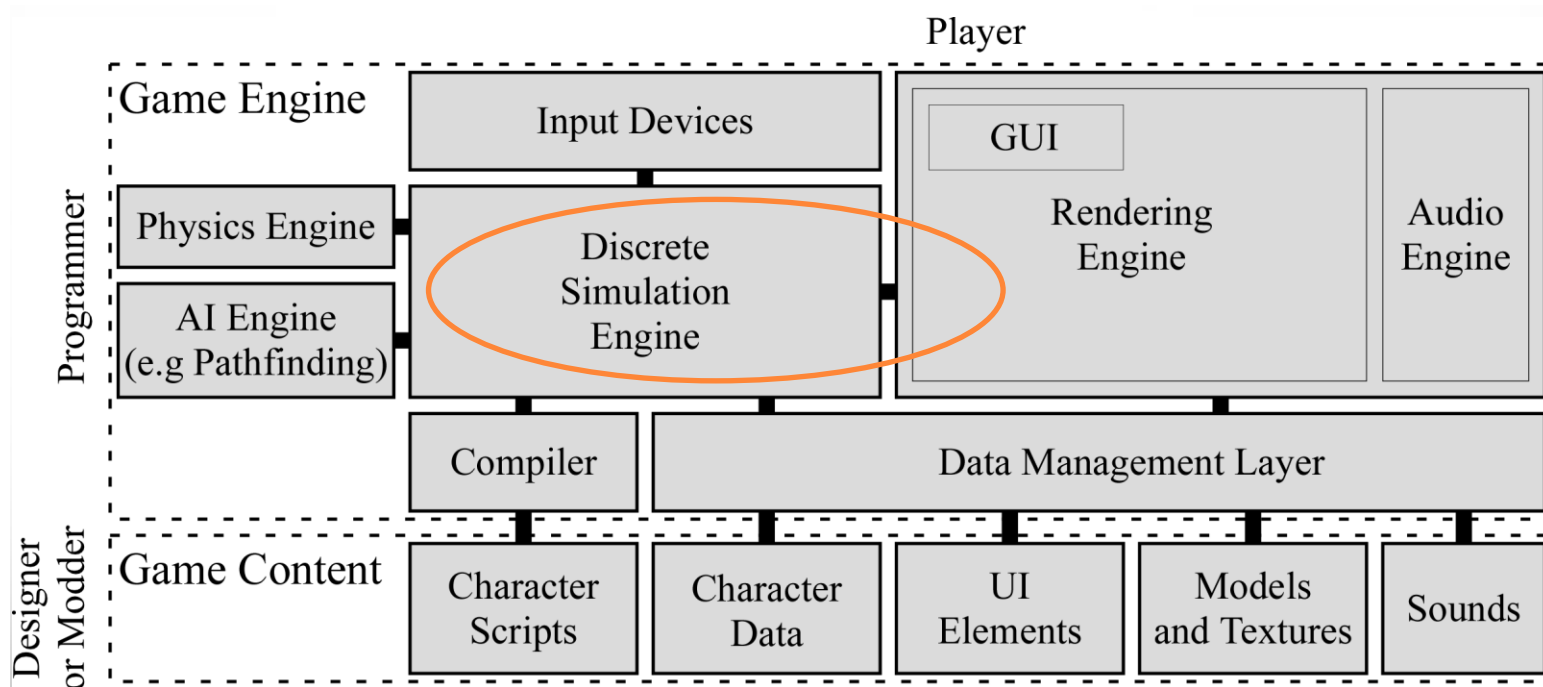
INTRODUCTION TO COMPUTER 3D GAME DEVELOPMENT

Space and Motion of Game Objects

潘茂林, panml@mail.sysu.edu.cn

中山大学·软件学院

游戏引擎架构



目录

- 游戏世界空间模型
- 坐标变换与运动
 - Transform 组件
 - Vector 对象平移
 - Quaternion 对象旋转
- 课堂实验（模拟太阳系）
- 面向对象设计思考
 - 项目的组织
 - 导演类设计
 - 控制器类设计
 - MVC架构小结



游戏世界空间模型

(1) 游戏世界空间关注的问题

- 游戏空间维度 2-D or 3-D?
 - Even if graphics 3-D, may have 2-D gameplay
 - Could you have other dimensions (1-D, 4-D)?
 - 2.5D (卷轴、深度)
- 游戏对象的尺寸?
 - 游戏对象是写实的、抽象的?
 - 它们如何影响玩法?
- What are your boundaries? (玩家可以看、玩的东西)
 - What can the player interact with?



游戏世界空间模型

(1) 游戏世界空间关注的问题



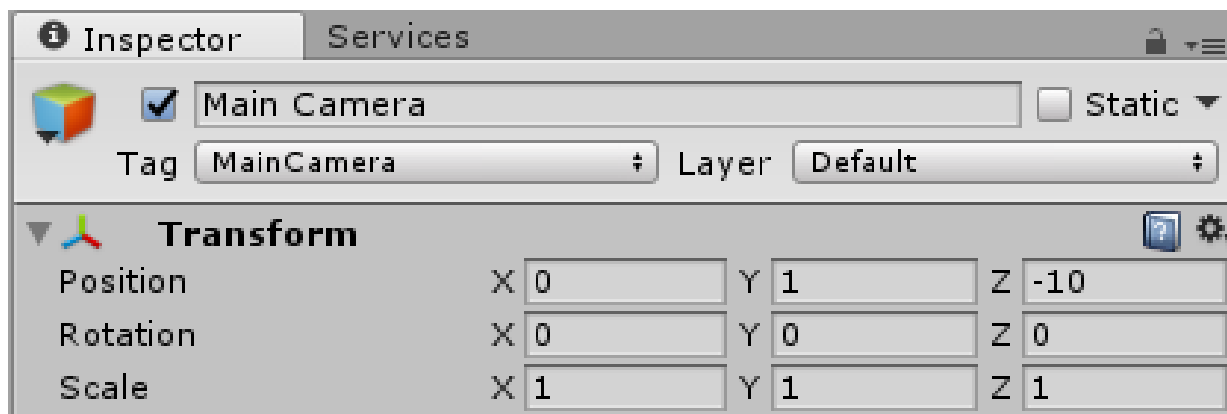
第一人称、第三人称.....看到的玩家的视图。例如，建造的物体，大小，地图啊等等



游戏空间模型

(2) 坐标系统与实现

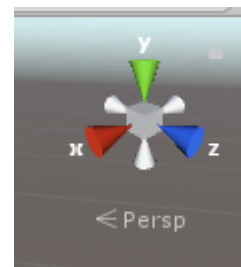
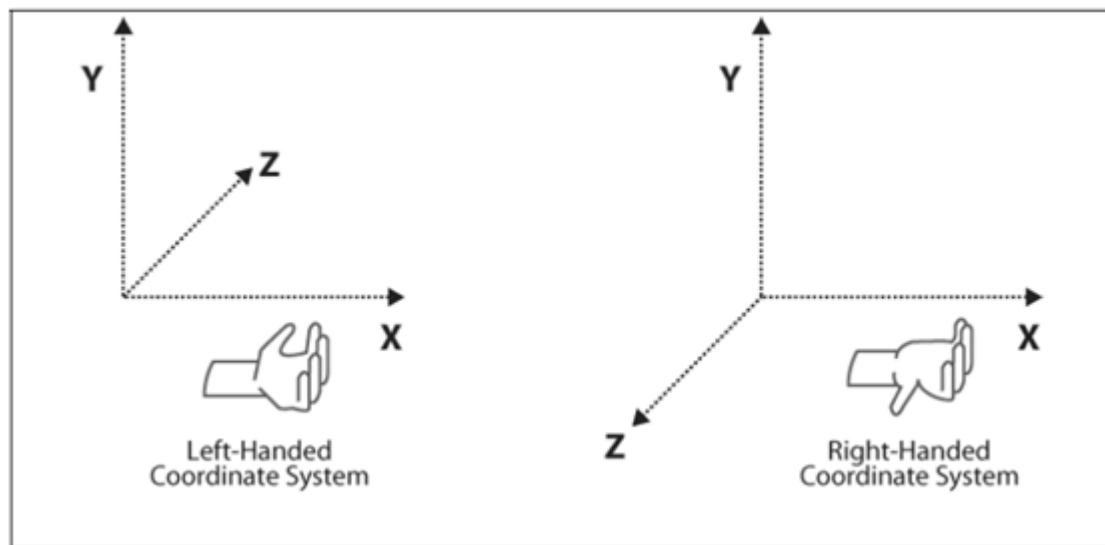
- 世界坐标（绝对坐标）
 - 游戏对象在地图上的绝对位置、角度、比例
- 对象坐标（相对坐标）
 - 游戏对象相对父游戏对象的位置、角度、比例
- Transform 组件（每个游戏对象必须的组件）



游戏空间模型

(3) 左手、右手坐标系

- 典型 3D 正交坐标系（绝对坐标）
 - Z 轴：深度维度，前后方向。Z 越小越靠前
 - Y 轴：高度维度，上下方向。Y 越大越高
 - X 轴：水平维度，左右方向。
- 左手坐标，X轴向左；右手坐标，X轴向右。



坐标变换与运动

(1) 简单运动

- 先看一段小程序

```
public class MoveLeft : MonoBehaviour {  
  
    // Use this for initialization  
    void Start () {  
  
    }  
  
    // Update is called once per frame  
    void Update () {  
        this.transform.position += Vector3.left * Time.deltaTime;  
    }  
}
```



课堂实验（一）

简单运动

○ 创建新游戏 motion

○ 测试1:

- 添加一个立方体
- 为立方体添加行为 MoveLeft
- 运行

○ 测试2:

- 从 hello 项目目录assets下拖入table.prefab 到 assets
- 将 table 预制拖入 Hierarchy。
- 将 MoveLeft 拖入 table 的 Inspector
- 运行
- 在 table 上添加 MoveUp 行为（代码如何写？）
- 运行



坐标变换与运动

(2) 深入 TRANSFORM 组件 - 属性

- 位置、欧拉角、比例、旋转
 - 世界坐标: position, eulerAngles, scale, rotation
 - 相对坐标: localposition, local...
- 相对位置
 - up, right, forward
- 空间依赖
 - parent, childCount
- 欧拉角: 以某个轴为法向量旋转, 范围 [0..360)
 - 例如: (0, 45, 0) 按左手法则沿 y 轴转 45
- Rotation: Quaternion, 四元素(x, y, z, w)。内部旋转变换表示, Unity建议不要直接修改它们。

坐标变换与运动

(2) 深入 TRANSFORM 组件 – 方法

- 平移 Translate

`translation : Vector3, relativeTo : Space = Space.Self`

- 旋转 Rotate

`eulerAngles : Vector3, relativeTo : Space = Space.Self`

- 绕转 RotateAround （世界坐标）

`point : Vector3, axis : Vector3, angle : float`

- 指向 LookAt

`target : Transform, worldUp : Vector3 = Vector3.up`



坐标变换与运动

(3) 向量 - VECTOR3

○ 计算

- normalized 归一化
- magnitude 计算长度
- 向量各种计算 dot, project, angle, distance

○ 追踪 MoveTowards

current : Vector3, target : Vector3, maxDistance : float

○ 转向 RotateTowards

-

○ 插值 Lerp

from : Vector3, to : Vector3, t : float [0..1]

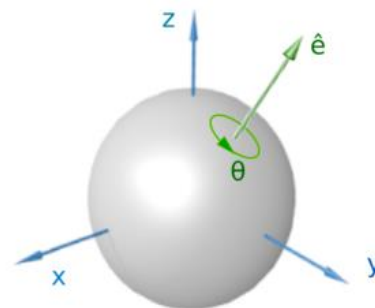
Vector3 三维向量: <http://www.ceeger.com/Script/Vector3/Vector3.html>



坐标变换与运动

(4) 四元素 QUATERNION

- Quaternion 和 eulerAngles 都是等价的 3D 空间旋转矩阵表示（见维基百科）
 - Conversion between quaternions and Euler angles
- Quaternion 有很好的计算特性，例如：
 - AngleAxis（如右图），eular 轴旋转矩阵
float angle, Vector3 axis
 - FromToRotation 向量旋转
Vector3 fromDirection, Vector3 toDirection
 - LookRotation 指向
Vector3 forward, Vector3 upwards = Vector3.up
 - Slerp 插值
- 更多请参考官方原文



坐标变换与运动

(5) 阅读并解释代码行为

```
// Use this for initialization
void Start () {
    // ???
    this.transform.rotation = Quaternion.AngleAxis(30, Vector3.up);
}

// Update is called once per frame
void Update () {
    // ???|
    this.transform.rotation *= Quaternion.AngleAxis(30 * Time.deltaTime, Vector3.up);
}
```

矩阵变换的作用通常不是 Transform 常用函数能做到的！！



坐标变换与运动

(5) 阅读并解释代码行为


```
// Update is called once per frame  
void Update () {  
    // ???  
    this.transform.Rotate (Vector3.up * 30 * Time.deltaTime);  
    // ???  
    this.transform.RotateAround ( Vector3.zero, Vector3.up, 30 * Time.deltaTime);  
    // ???  
    this.transform.RotateAround (- Vector3.left * 2.5f, Vector3.up, 30 * Time.deltaTime);  
}
```



坐标变换与运动

(5) 阅读并解释代码行为

```
public class MoveTo : MonoBehaviour {  
  
    // The target marker.  
    Vector3 target = Vector3.right * 5;  
  
    // Speed in units per sec.  
    float speed = 5;  
  
    // Use this for initialization  
    void Start () {  
  
    }  
  
    // Update is called once per frame  
    void Update () {  
        float step = speed * Time.deltaTime;  
        transform.position = Vector3.MoveTowards(transform.position, target, step);  
    }  
}
```



课堂实验（二）

旋转练习（太阳系的世界）

- 下载太阳系贴图，拖入 assets （只要地球）
- 创建 3 个球分别命名 sun, earth, moon
 - 按您理解，设置位置、大小（不用鼠标是正解）
 - 用菜单 assets → create → material 创建太阳金和月球白两个资源
 - 把图片、资源拖放到对应球体之上，OK！
- 编写行为 RoundSun 挂在任意对象上，如 sun
 - 建立 public Transform 保存 sun, earth, moon
 - 初始设置 的位置 $x=0,6,8$; $y=z=0$
 - 让地球按 Y 轴方向围绕太阳公转
 - 运行！

课堂实验（二）

旋转练习（太阳系的世界）

```
5 public class RoundSun : MonoBehaviour {
6
7     public Transform sun;
8     public Transform earth;
9     public Transform moon;
10
11     // Use this for initialization
12     void Start () {
13         sun.position = Vector3.zero;
14         earth.position = new Vector3 (6, 0, 0);
15         moon.position = new Vector3 (8, 0, 0);
16
17     }
18
19     // Update is called once per frame
20     void Update () {
21         earth.RotateAround(sun.position, Vector3.up, 10 * Time.deltaTime);
22         earth.Rotate (Vector3.up * 30 * Time.deltaTime);
23         moon.transform.RotateAround(earth.position, Vector3.up, 359 * Time.deltaTime);
24     }
25 }
26
```

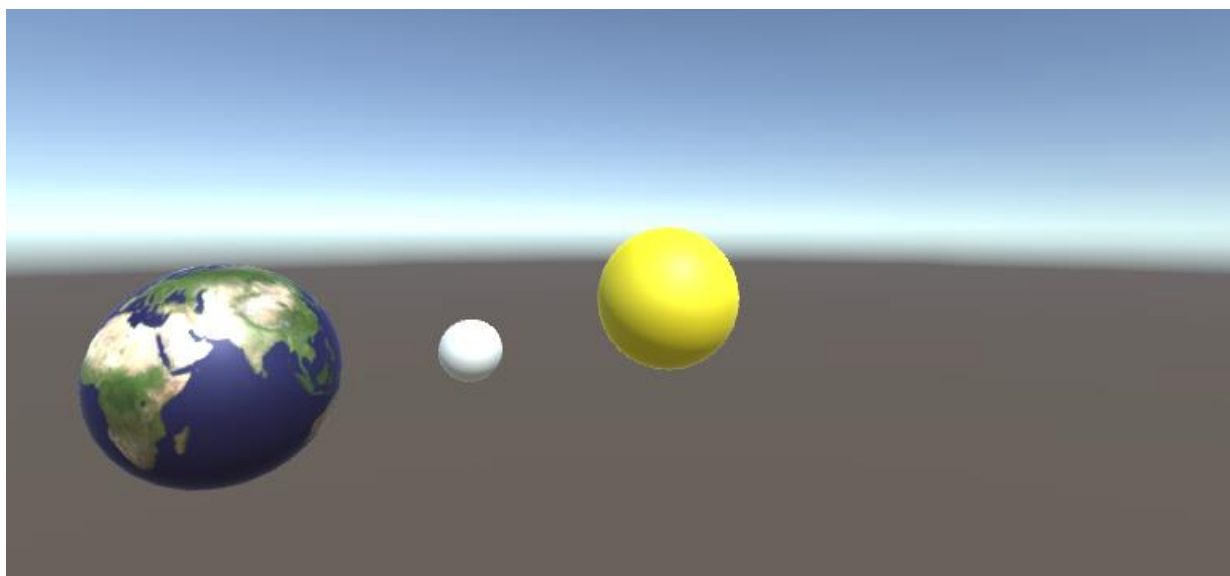
对吗？



课堂实验（二）

旋转练习（太阳系的世界）

○ 实验结果



○ 如果把

Main Camera
Directional light
sun
earth
moon

改成

Main Camera
Directional light
▼ sun
▼ earth
moon

结果呢？



坐标变换与运动

(6) 空对象的作用

○ 月球轨迹的问题

- 地球公转，月球轨道是按地球描述的，所以月球应该设计为地球子对象
- 地球自转，月球运动与自转无关，所以月球设计不该是地球子对象

○ 建议解决方法

- 使用一个空对象作为地球的影子
- 将月球挂在这个空对象上
- 空对象与地球位置保持一致
- 用 Quaternion 旋转这个空对象，月球会跟转
- 调整空对象与月球之间距离，就是椭圆轨道了



面向对象设计思考

(1) 类似 COCOS2D 编程的挑战

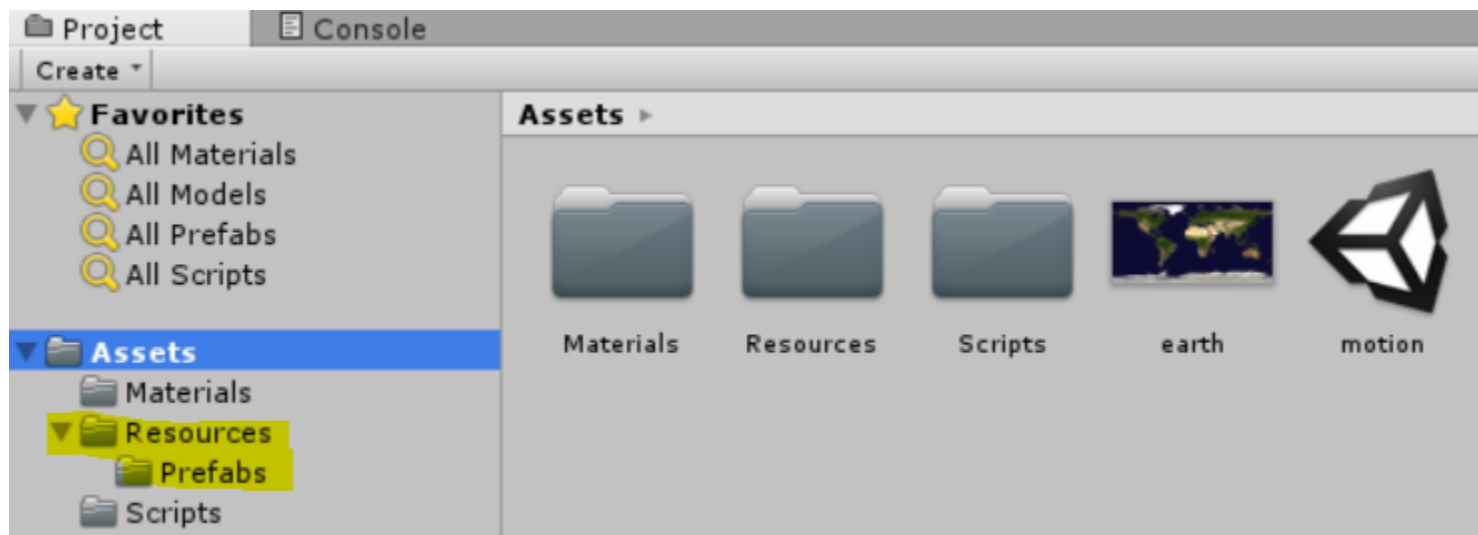
- 现代游戏系统（如 cocos2d）非常易用
 - 面向对象包装非常好
- 游戏离散事件系统组成
 - 对象（+组件）
 - 行为
- 3D 游戏离散系统编程挑战
 - 行为的并行和无序（如何调试？）
 - 复杂的空间组合与效果设计
- MVC分离的设计？
 - 模型（Model）-- GameObject 及其关系
 - 控制（Controller）-- MonoBehavior
 - 视图（View）-- Camera



面向对象设计思考

(2) 游戏资源的组织

- Unity 项目设计器已经给出了推荐组织方案



- 材料
- 脚本
- 模型
- 预制（太阳系变成预制件，必须放在 Resources 目录下）

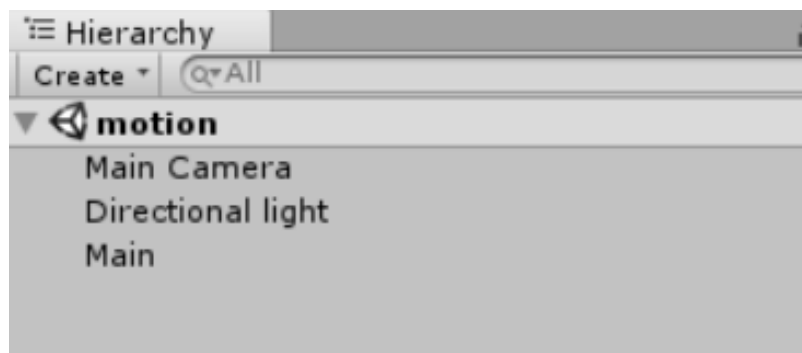


面向对象设计思考

(3) 创建场景启动对象和控制器

- 构造 Main 使得行为有限有序!!!

1. 使得场景有仅有一个对象，如 Main(空对象)



2. 挂载该场景第一个执行的脚本
场景控制器，如 FirstController

- 为什么要这么做？

- 回顾 Unity 离散仿真引擎执行过程



面向对象设计思考

(4) DIRECTOR (导演) 对象与单实例模式

- 创建 SSDirector 对象，其职责大致如下：
 - 获取当前游戏的场景
 - 控制场景运行、切换、入栈与出栈
 - 暂停、恢复、退出
 - 管理游戏全局状态
 - 设定游戏的配置
 - 设定游戏全局视图
- 面向对象设计就是基于职责的设计。
 - 每个对象有仅可能单一的职责（与现实保持一致）
 - 尽可能少暴露内部细节

面向对象设计思考

(4) DIRECTOR (导演) 对象与单实例模式

```
5 public class SSDirector : System.Object {  
5     // singleton instance  
7     private static SSDirector _instance;  
3  
9     public ISceneController currentSceneController { get; set; }  
9     public bool running { get; set; }  
1  
2     // get instance anytime anywhere!  
3     public static SSDirector getInstance() {  
4         if (_instance == null) {  
5             _instance = new SSDirector ();  
5         }  
7         return _instance;  
3     }  
9  
9     public int getFPS() {  
1         return Application.targetFrameRate;  
2     }  
3  
4     public void setFPS(int fps) {  
5         Application.targetFrameRate = fps;  
5     }
```

不被Unity
内存管理管理



面向对象设计思考

(5) SCENECONTROLLER (场记)

- XXXSceneController 类
 - 也称 XXX 场景控制器
- 场景管理器的职责
 - 管理本次场景所有的游戏对象
 - 协调游戏对象（预制件级别）之间的通讯
 - 响应外部输入事件
 - 管理本场次的规则（裁判）
 - 杂务



面向对象设计思考

(5) SCENECONTROLLER (场记)

```
5 public class FirstController : MonoBehaviour, ISceneController {
6
7     // the first scripts
8     void Awake () {
9         SSDirector director = SSDirector.GetInstance ();
10        director.setFPS (60);
11        director.currentSceneController = this;
12        director.currentSceneController.LoadResources ();
13    }
14
15    // loading resources for first scene
16    public void LoadResources () {
17        GameObject sunset = Instantiate<GameObject> (
18            Resources.Load <GameObject> ("prefabs/sun"),
19            Vector3.zero, Quaternion.identity);
20        sunset.name = "sunset";
21        Debug.Log ("load sunset ...\n");
22    }
23
24    // Use this for initialization
25    void Start () {
26        //give advice first
27    }
```

至此，初始化，资源加载都在您代码控制之下！！！！



面向对象设计思考

(5) 接口 (INTERFACE)

○ 接口的定义

- 一种数据类型，表示对象的对外行为类型

○ 案例：

- 每个场景都有自己的场记，导演需要与不同场景打交道
- 导演只知道场记的加载资源、暂停、恢复等行为，但并不知道实现细节，如：暂停前要保存哪些状态等
- 导演也不希望知道细节，而是规定场记们必须会做什么，怎么做自由发挥。这个规定就是接口

```
public interface ISceneController
{
    void LoadResources();
    void Pause();
    void Resume();
}
```



面向对象设计思考

(5) 接口与门面 (FASÀDE) 模式

○ 门面 (Fasàde) 模式的概念概念

- 外部与一个子系统的通信必须通过一个统一的门面 (Facade)对象进行。

○ 案例研究

- 按游戏的定义，游戏规则是一个条件、动作、结果列表
- 用户的行为将改变游戏状态。
- 最直观的做法就是定义一个用户动作接口，这样就实现了用户行为与游戏系统规则计算的分离。优势是显而易见的，例如：
- 用户行为编程人员可以选择菜单、键盘、或组合实现用户交互行为，而模型处理人员可以自由定义游戏规则。



面向对象设计思考

(5) 接口与门面 (FASÀDE) 模式

- 定义用户界面与游戏模型的交互接口

```
4 public interface IUserAction
5 {
6     void GameOver();
7 }
```

- 实现用户界面程序

```
public class UserGUI : MonoBehaviour {

    private IUserAction action;

    void Start () {
        action = SSDirector.GetInstance ().currentSceneController as IUserAction;
    }

    void OnGUI() {
        float width = Screen.width / 6;
        float height = Screen.height / 12;

        if (GUI.Button(new Rect(0, 0, width, height), "Game Over!")) {
            action.GameOver();
        }
    }
}
```



面向对象设计思考

(5) 接口与门面 (FASÀDE) 模式

- 前面代码看出，人机交互接口委托给了当前场景控制器

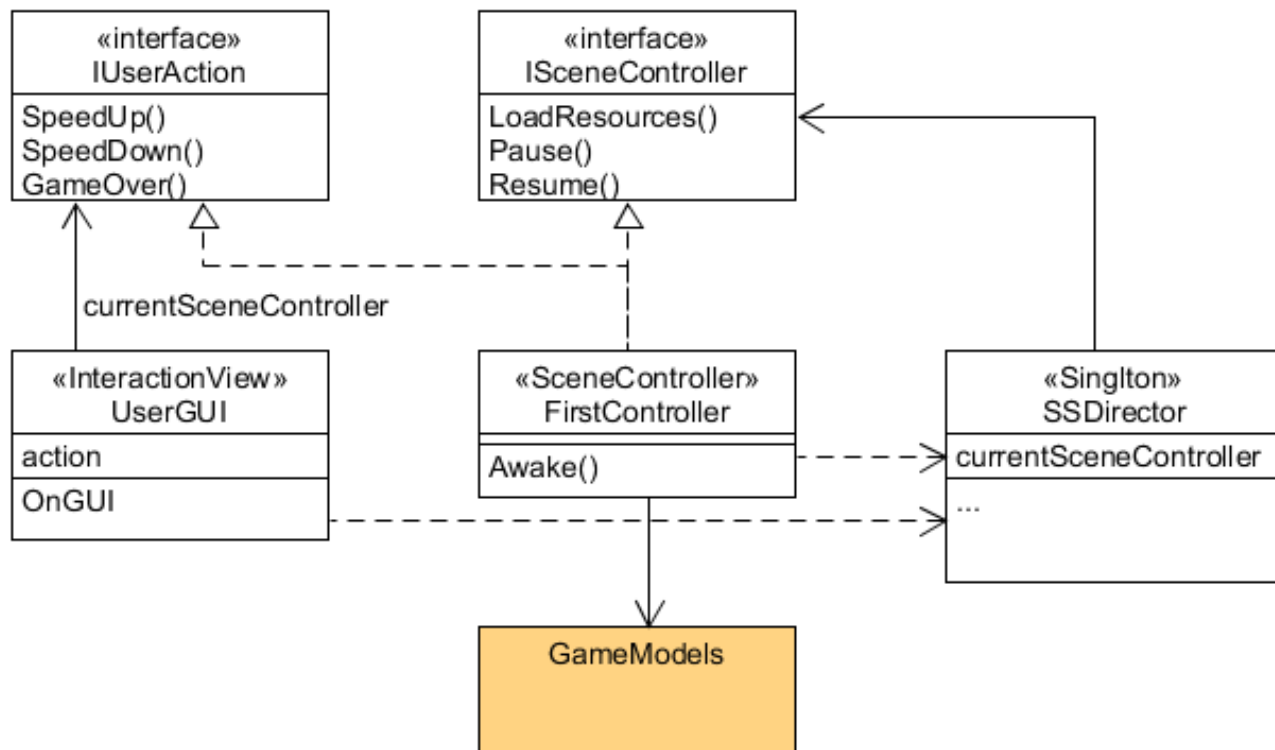
```
5 public class FirstController : MonoBehaviour, ISceneController, IUserAction {
6
7     // the first scripts
8     void Awake () {
9
10    ...
11
12    ...
13
14    #region IUserAction implementation
15    public void GameOver ()
16    {
17        SSDirector.getInstance ().NextScene ();
18    }
19    #endregion
20
21    ...
22
23    ...
24
25    ...
26
27    ...
28
29    ...
30
31    ...
32
33    ...
34
35    ...
36
37    ...
38
39    ...
40
```

结果，控制器说，我也不知道怎么结束游戏，请导演决定！



面向对象设计思考

(6) 小游戏对象图



标准 MVC 结构的 game。游戏虽小，稳固的框架经得起任意扩展！！！！



自学内容： 常用运动计算相关API

- Time
- Random
- Mathf



课堂作业

- 建立一个行为，完成以下任务：
 - 在一个 p 点半径 5 unit 圆形内，
 - 随机部署 3 个物体（怪物） $z = 0$ 。
 - 行为名称： ScatterObject
 - 参数：
 - Vector3 p ,
 - float radius,
 - int number,
 - GameObject obj
 - 注意，必须克隆或预制哦！



课程小结

○ 空间与运动

- 绝对空间与相对空间
- 坐标变换与运动
- Transform对象与简单运动
- Vector3对象与平移
- Quaternion对象与旋转

○ 面向对象设计技巧

- 单实例模式
- 门面模式
- MVC架构

