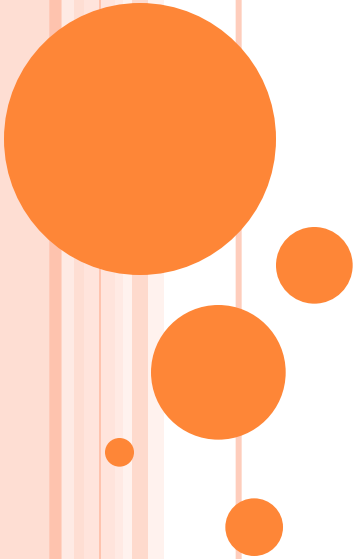
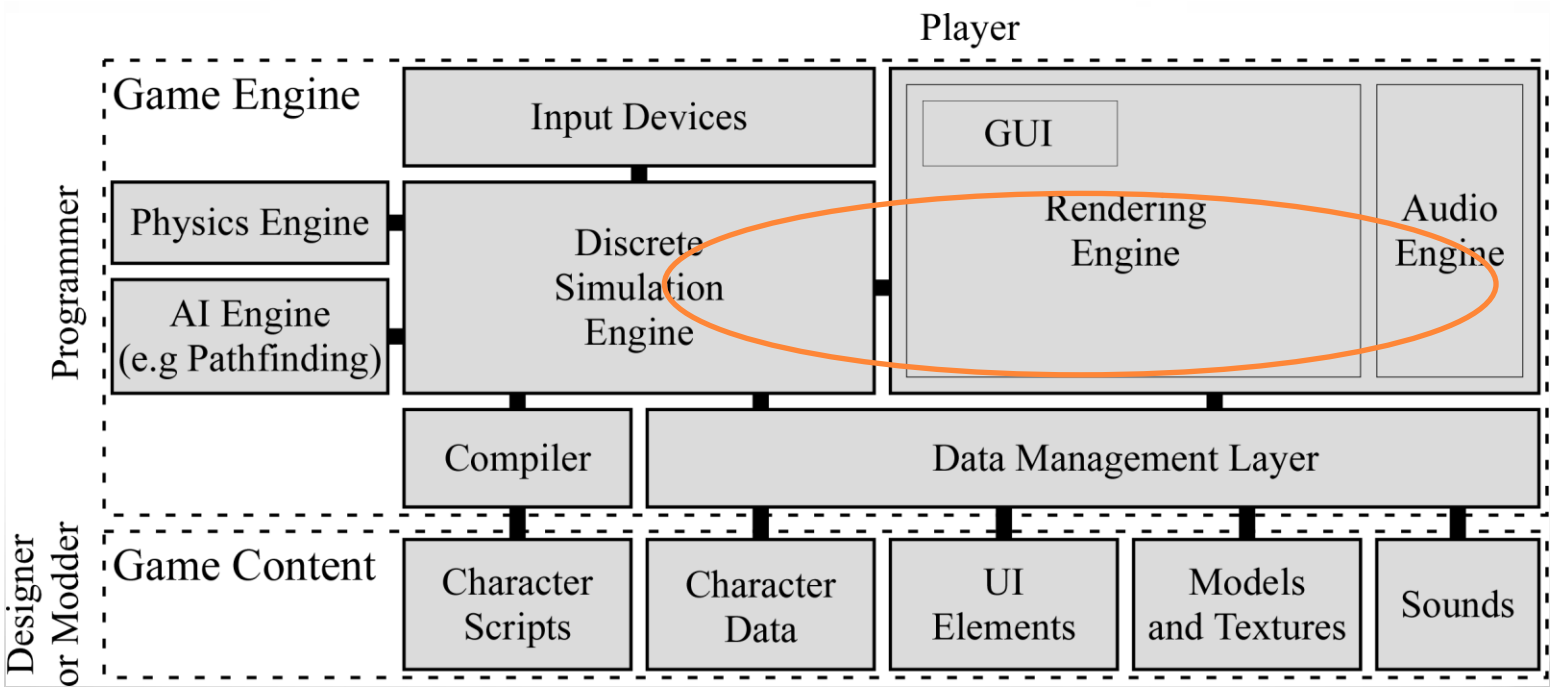


Introduction to Computer 3D Game Development

Game Objects and Graphics Essential

潘茂林，panml@mail.sysu.edu.cn
中山大学·软件学院

游戏引擎架构


Player
Game Engine
Input Devices
GUI
Physics Engine
Discrete
Simulation
Engine
Rendering
Engine
Audio
Engine
AI Engine
(e.g Pathfinding)
Compiler
Data Management Layer
Programmer
Designer
or Modder
Game Content
Character
Scripts
Character
Data
UI
Elements
Models
and Textures
Sounds

目录

- 常见游戏对象与渲染
  - 常见游戏对象
  - Camera 摄像机
  - Skyboxes 天空盒
  - 3D 物体显示
  - 地形构造工具
  - 声音
  - 游戏资源库
- 面向对象设计思考
  - 动作管理器（SSActionManager）
  - 设计常用动作（SSAction）

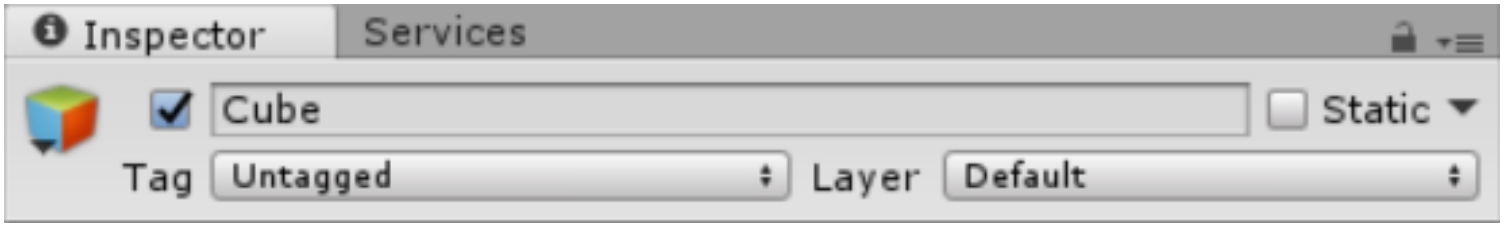
基础游戏对象
（1）GAME OBJECT 菜单

- 菜单仅列出了基础、常用对象
  - Empty （不显示却是最常用对象之一）
  - Camera 摄像机，观察游戏世界的窗口
  - Light 光线，游戏世界的光源
  - 3D 物体

    基础 3D 物体

    构造 3D 物体
  - Audio 声音
  - UI 基于事件的 new UI 系统（专题介绍）
  - Particle System 粒子系统与特效（专题介绍）

基础游戏对象
（2）游戏对象都有的属性



- Active
  - 不活跃则不会执行 update() 和 rendering
- Name
  - 对象的名字，不是 ID。ID 使用 GetInstanceID()
- Tag
  - 字串，有特殊用途。如标识主摄像机
- Layer
  - [0..31]，分组对象。常用于摄像机选择性渲染等

基础游戏对象
（3）CAMERA 摄像机 – 游戏场景渲染

- 添加新的摄像机
  - 菜单 → game object → camera
  - 检查 inspector 应看到：

  Camera 组件（相机基本属性）

  Flare layer （使用炫光镜头，处理物体炫光和雾化纹理）

  GUI layer （渲染 遗留的 GUI 界面）

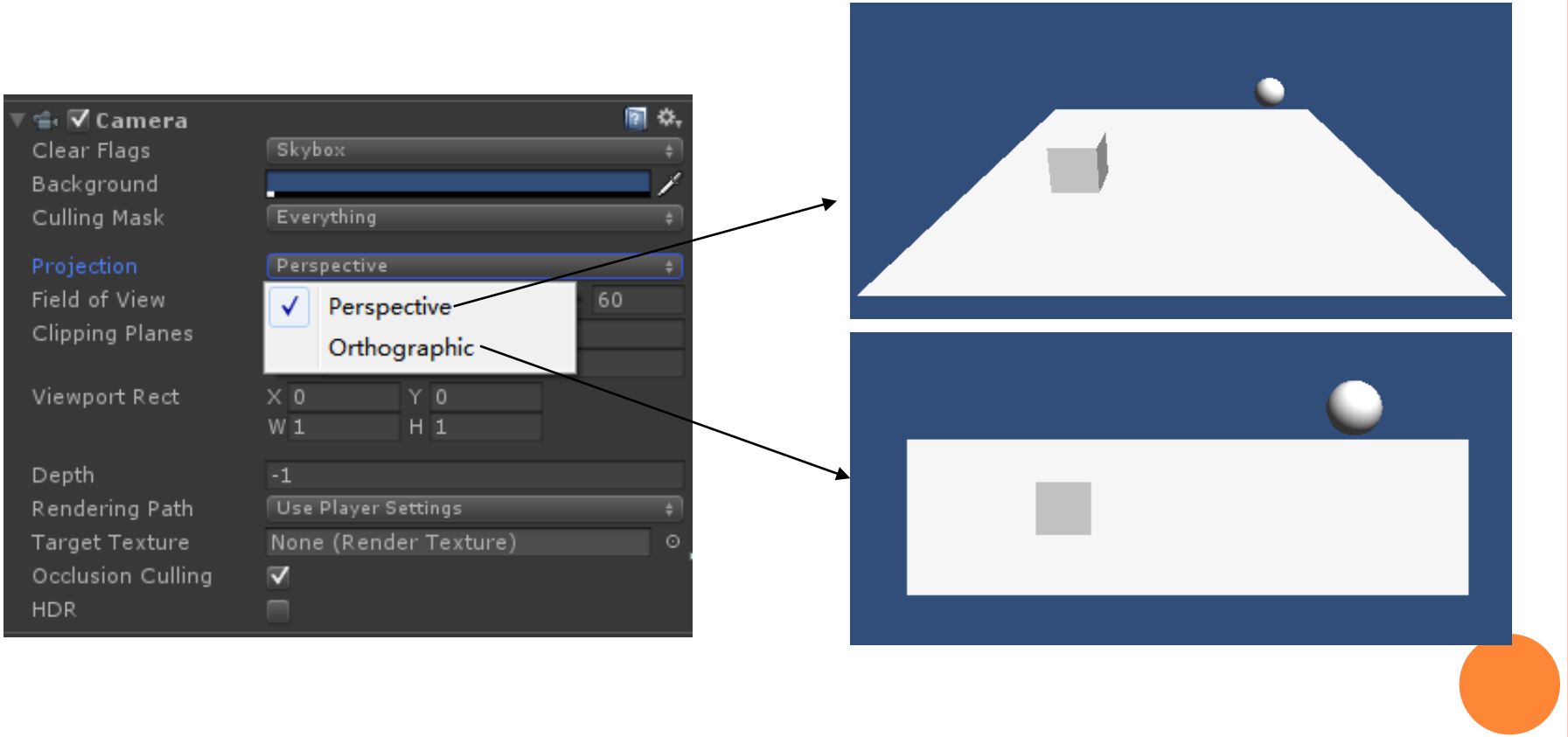
  Audio Listener （挂载拾音器，接收场景中声音）
  - 其他 rendering 组件（见 add component）

- 对比与摄像机 tag 的区别！

镜头光晕：http://www.ceeger.com/Components/class-LensFlare.html
物体 Flare 组件：http://www.ceeger.com/Components/class-Flare.html

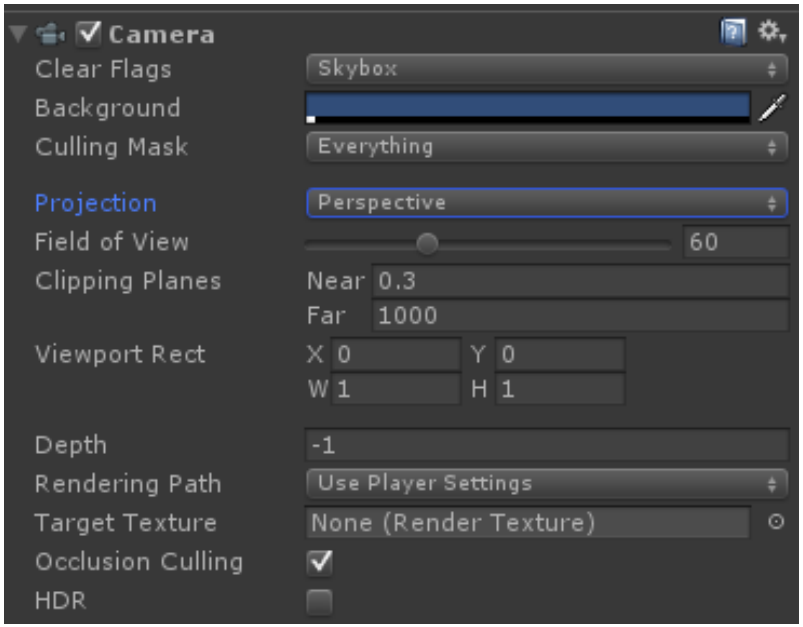
基础游戏对象
（3）CAMERA 摄像机 – 游戏场景视图

- 透视视图与正交视图



基础游戏对象
（3）CAMERA 摄像机 – 常用参数

- 常用参数



- Background: 背景颜色
- Culling Mask: 剔除遮罩。用于指定摄像机所作用的层(Layer)。
- Field of View(FOV): 视野范围。只针对透视镜头，用于控制视角宽度与纵向角度。
- Size: 视口大小。只针对正交镜头，设定为相当于屏幕高度的一半。
- ClippingPlanes: 表示摄像机的作用范围。只显示距离为[Near, Far]之间的物体。
- Viewport Rect: 控制摄像机呈现结果对应到屏幕上的位置以及大小。屏幕坐标系：左下角是(0, 0)，右上角是(1, 1)。
- Depth: 当多个摄像机同时存在时，这个参数决定了呈现的先后顺序，值越大越靠后呈现。

课程实验（一）
鸟瞰图的制作

- 使用双摄像机
  1. 在场景中放置一些 3D 物体
  2. 放置第二摄像机
  - ✓ 设置正交视图；
  - ✓ 位置与旋转： positon(0,3,0), rotation(90,0,0)；
  - ✓ 视口： (x,y,w,h) = (0.9,0,0.1,0.12)
  - ✓ 深度：0 （必须大于主摄像机深度）
  3. 运行结果：右下角出现了一个小窗口
- 使用2D的GUI
  - 鸟瞰图制作：
    http://blog.csdn.net/qq_bingfeng_8/article/details/18561617

基础游戏对象
（3）CAMERA 摄像机 – 多摄像机

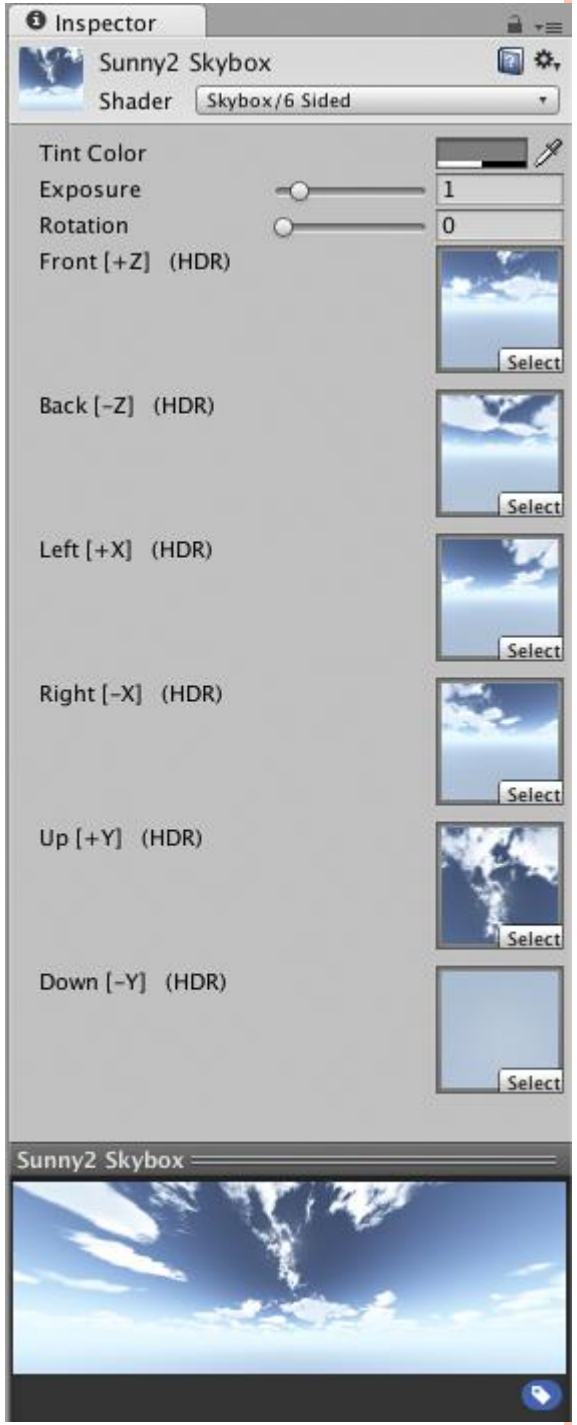
- 绚烂的效果要素之一：多摄像机的使用
- 多摄像机相关属性
  - Clear Flag
  - Culling mask
  - Depth
- 这是一个中级话题

在Unity中使用多个相机 - 及其重要性：
http://www.manew.com/thread-47076-1-1.html

基础游戏对象
（3）天空 – SKYBOXES

- 天空是一个球状贴图，通常用 6 面贴图表示
- 从 Unity 5 开始，官方不再资源天空提供资源，一般从 Asset Store 获取
- 使用 skybox 在 Store 中搜索，下载：
  - Skybox
  - Fantasy Skybox FREE

- 使用
  1. 在摄像机添加天空组件。Component → rendering → skybox
  2. 拖入天空材料（Material）



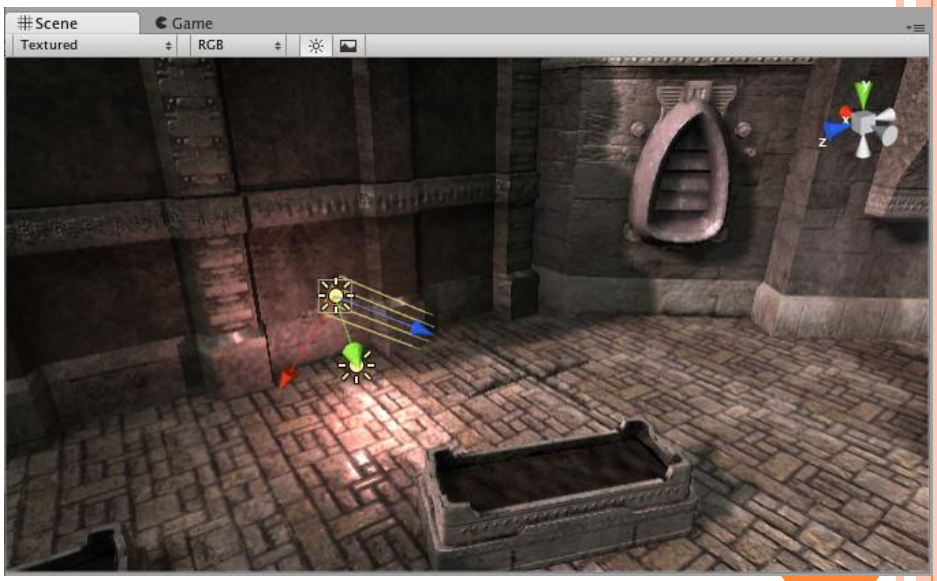
基础游戏对象
（3）天空 – SKYBOX 制作

- 制作一个 6 面体材料
  - 菜单 Assets → create → Material
  - Inspector → shader → skybox → 6 sided
  - 按前后、上下、左右拖入 6 个图片
  - 制作完成，拖入项目 Material 目录

- 如何恢复系统默认天空？
  - 删除摄像机自定义天空
  - 菜单 → window → lighting
  - Scene → skybox → Default-skybox

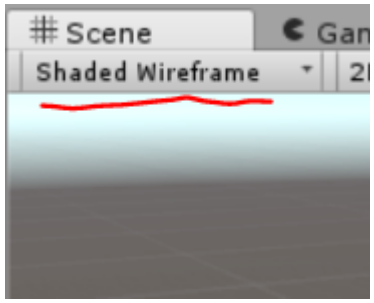
基础游戏对象
（4）光源 – LIGHT

- 光与影是让游戏世界富有魅力。
- 创建一个灯光（两种方法一样）
  - 菜单 GameObject → Light
  - 创建一个空对象，添加灯光组件
- 灯光组件属性
  - 灯光类型（type）
    - ✓ 平行光（类似太阳光）
    - ✓ 聚光灯（spot）
    - ✓ 点光源（point）
    - ✓ 区域光（area，仅烘培用）
  - 阴影（shallow）
  - 剪影（cookies）



灯光（光源）：http://www.ceeger.com/Manual/Lights.html

基础游戏对象
（5）3D 物体 – 3D OBJECTS

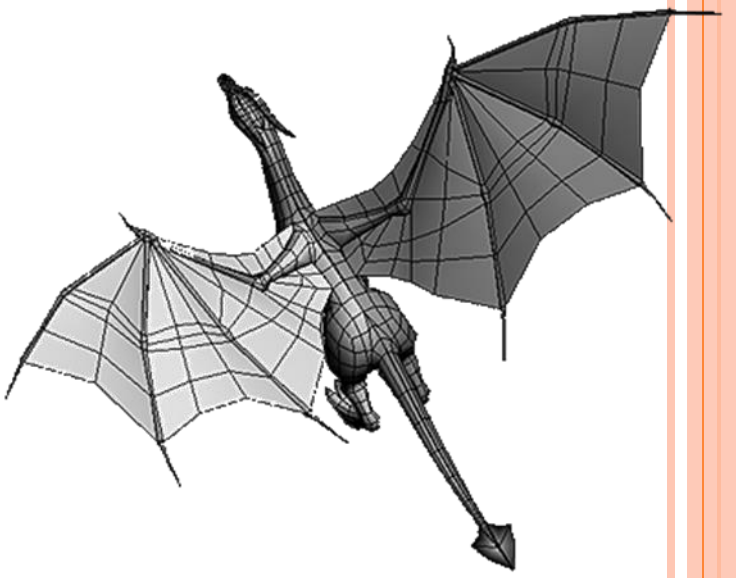


- 网格与物体
  - 三角网格是游戏物体表面的唯一形式
  - 将场景视图从 Shade → Wirframe 就看到
  - 所有物体包括立方体，圆都是网格
- 3D 物体显示组件
  - Mesh 组件：

  物体表面三角网格，形成物体形状
  - Mesh Renderer组件：

  表面渲染器，显示物体色彩
  - 其中：Material 和 Shader 对象

  （材料与着色器）则是绘制物体的工具



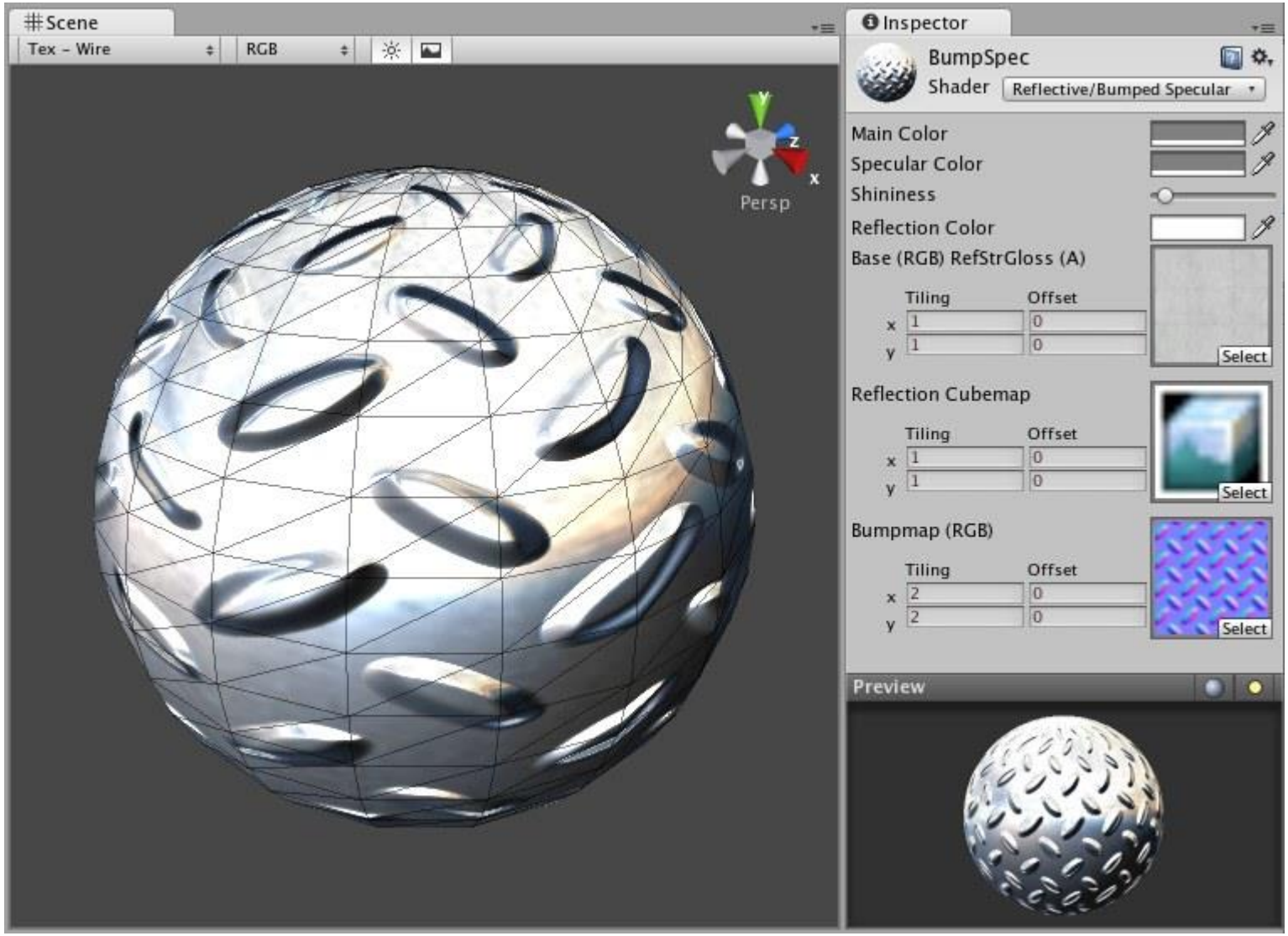
基础游戏对象
（5）3D 物体 – 渲染

- 材质与着色器（Materials and Shaders）
  - 纹理（Texture）：位图，表示物体本身的色彩。
  - 材质（Material）：包含一个或一组 Texture，以及元数据（meta-data）属性，着色程序（Shader）。
  - 其中 meta-data 定义了 Texture 与 mesh 的映射关系，材料的光线吸收、透明度、反射与漫反射、折射、自发光、眩光等特性
  - Shader 是着色程序。它能利用显卡硬件渲染特性，按 meta-data 将材质按物体纹理、光线特性，结合游戏场景中的光线生成用户感知的位图（像素点）。
  - Shader 的编程超出了本课程的范围，涉及大量图形学与显卡计算的知识。一句话，物体发光、质感、层次、镜像等效果取决于Shader。

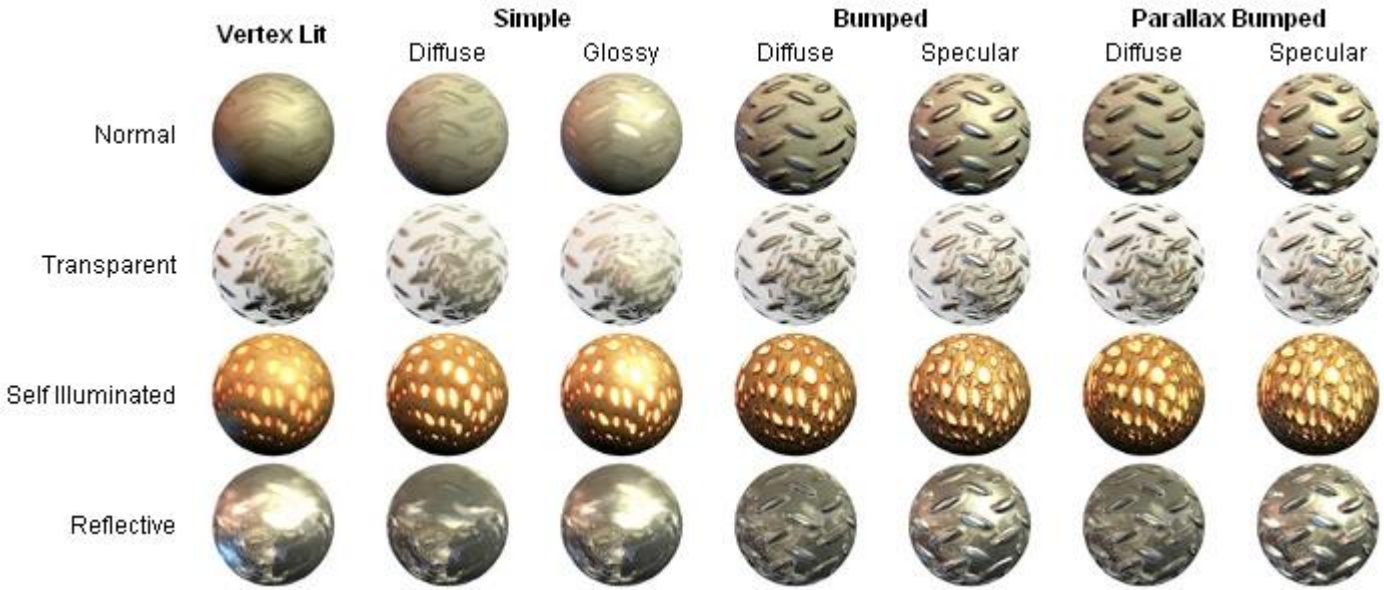
材质与着色器：http://www.ceeger.com/Manual/Materials.html

基础游戏对象

（5）3D 物体渲染 – 材料示例



基础游戏对象
（5）3D 物体渲染 – SHADER与显示效果



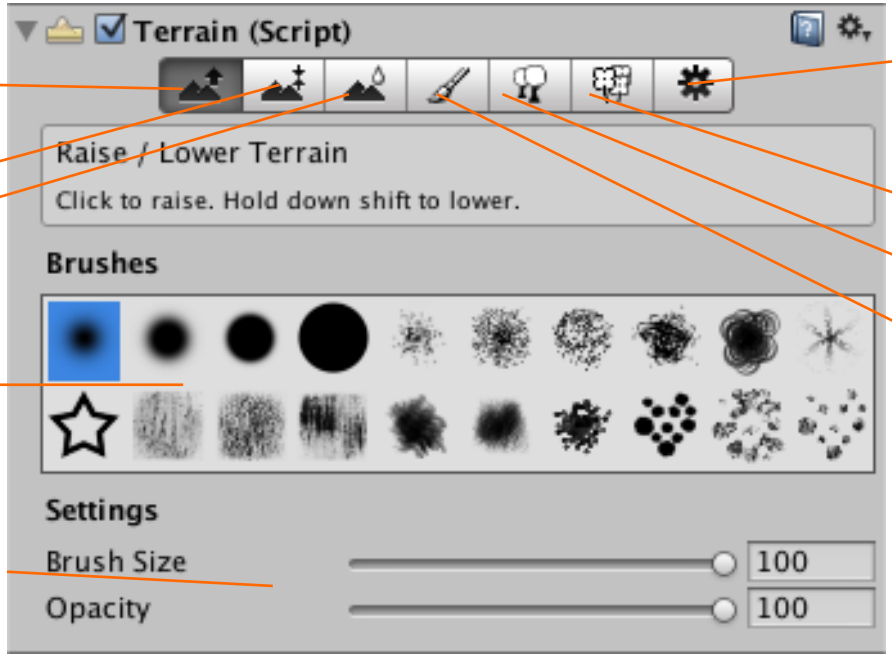
游戏编程人员须知：

- 同样物体纹理有不同的效果
- 效果取决于材料的元数据以及shader的设置
- 材质和shader资源获取是3D游戏成功要素之一

折射、镜面反射shader：http://www.ceeger.com/forum/read.php?tid=3162

基础游戏对象
（6）地形系统 – 创建与编辑

- Unity 提供了简单的地形设计工具
  - 菜单 → Game object → 3d object → Terrain
- 地形设计工具箱



造山

等高地形

平缓

刷子形态

刷子参数：
大小、高度、密度

设置

种草

种树

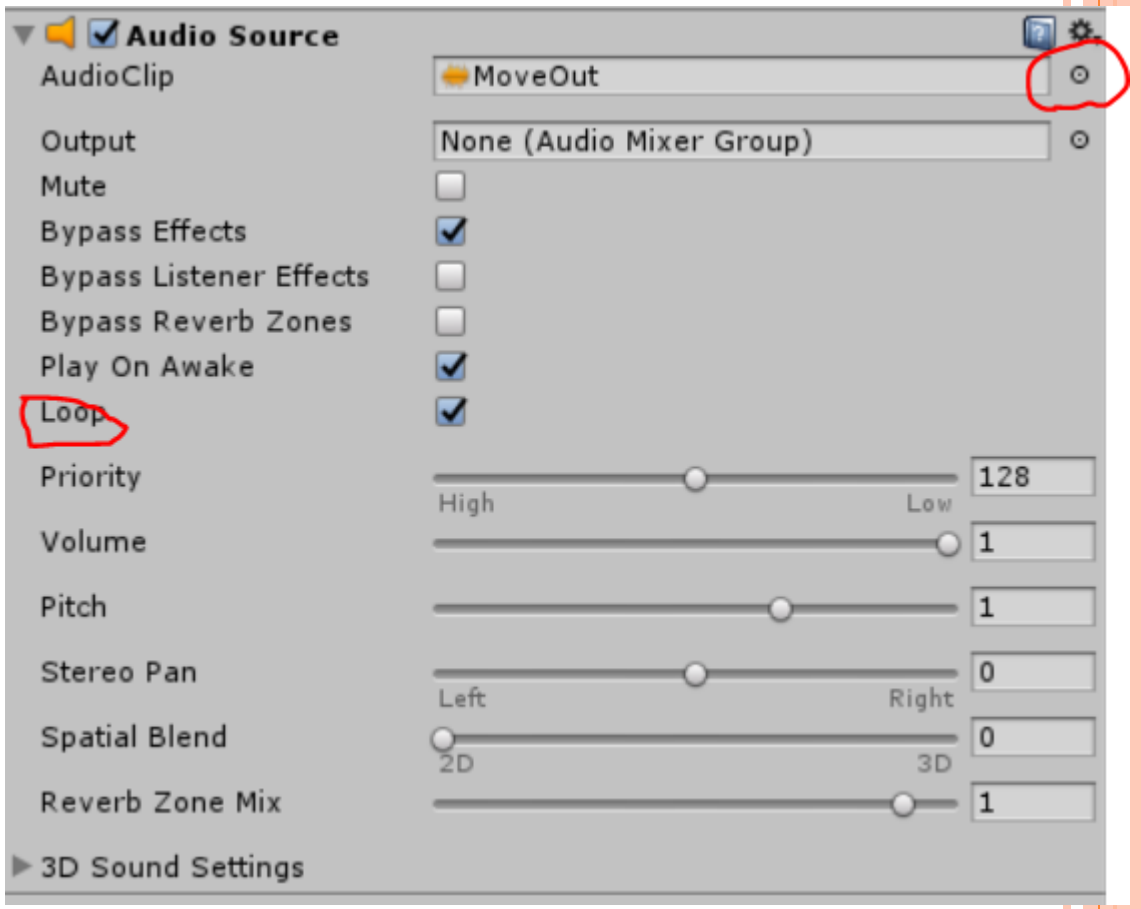
刷地

基础游戏对象
（6）地形系统－实战

1. 设置地图大小与位置，例如：
   - Resolution: Length, width, heigh = 100,100,20
   - Tranform：position = -50，-3，0
2. 使用造山工具
   - 点击地图网格，上升地形；按 shift 键，下降地形
3. 使用高度工具
   - 画出一些山中路径
4. 使用平缓工具
   - 画出与平地的连接
5. 使用地面、种草、种树工具
   - 下载 Fantasy Skybox FREE 在demo 目录下有资源
   - 然后选择合适材料铺地、种草、种树

基础游戏对象
（7）音频源

- 创建一个音源
  - 菜单 game object → audio → audio source
- 设置组件属性
  - 选择声音素材
  - Play 时机
  - Loop
- 检查摄像机
  - Audio listener



基础游戏对象
（8）使用游戏资源库

- 导入角色资源
  - 菜单 assets → import packages → characters
- 添加第一人称角色
  - 项目，查找所有 prefabs
  - 添加 FirstPersonCharacterController
  - 禁止主摄像机
  - 运行！
  - 移动鼠标 ……

基础游戏对象
（9）综合效果

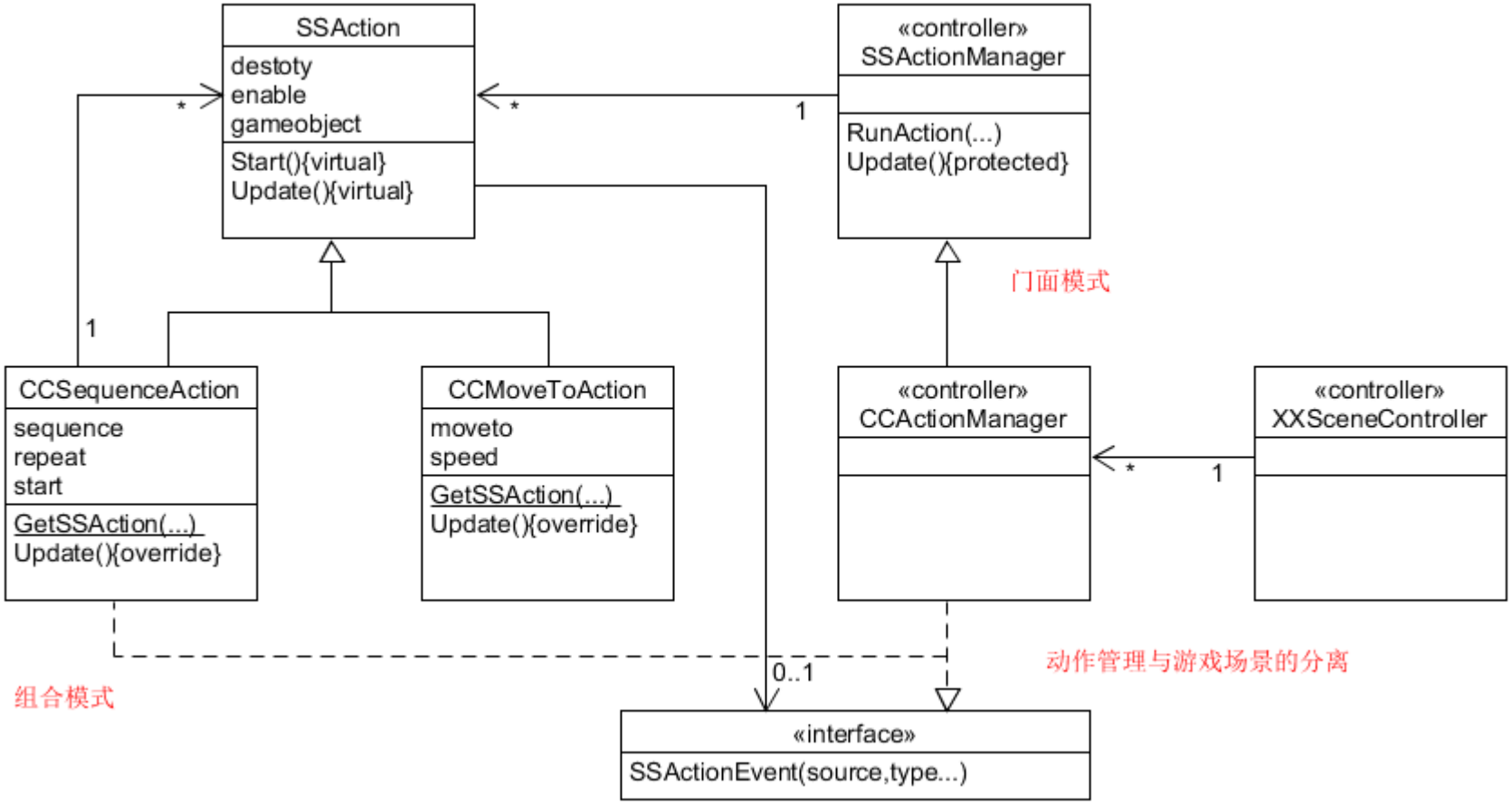
- 保存你的工作
- 打开 Fantasy Skybox FREE 在 demo 下场景
  - 运行！

- 你会够造这样的游戏场景吗？

面向对象设计思考
（1）建立动作管理器

- 动作管理是游戏内容重要内容
  - 电影也有专人负责
  - Unity自身有强大支持（动画系统，协程）
- 本课程的动作管理
  - 决解简单动作组合问题，类似 cocos2d 的 CCAction
    - 实现动作管理器
    - 实现基础动作类
    - 回调实现动作完成通知（经典方法）
  - 训练面向对象的程序设计能力
    - 程序更能适应需求变化
    - 更易于维护

面向对象设计思考
（2）规划与设计



面向对象设计思考
（3）动作基类（SSAction）

设计要点：

1. ScriptableObject 是不需要绑定 GameObject 对象的可编程基类。这些对象受 Unity 引擎场景管理
2. 防止用户自己 new 对象
3. 使用 virtual 申明虚方法，通过重写实现多态。这样继承者就明确使用 Start 和 Update 编程游戏对象行为
4. 利用接口实现消息通知，避免与动作管理者直接依赖。

```
5  public class SSAction : ScriptableObject {
6
7      public bool enable = true;
8      public bool destory = false;
9
10     public GameObject gameobject { get; set; }
11     public Transform transform { get; set; }
12     public ISSActionCallback callback { get; set; }
13
14     protected SSAction () {}
15
16     // Use this for initialization
17     public virtual void Start () {
18         throw new System.NotImplementedException ();
19     }
20
21     // Update is called once per frame
22     public virtual void Update () {
23         throw new System.NotImplementedException ();
24     }
25
```

面向对象设计思考
（4）简单动作实现

```
5 public class CCMoveToAction : SSAction
6 {
7     public Vector3 target;
8     public float speed;
9
10    public static CCMoveToAction GetSSAction(Vector3 target, float speed){
11        CCMoveToAction action = ScriptableObject.CreateInstance<CCMoveToAction> ()
12        action.target = target;
13        action.speed = speed;
14        return action;
15    }
16
17    public override void Update ()
18    {
19        this.transform.position = Vector3.MoveTowards (this.transform.position, ta
20        if (this.transform.position == target) {
21            //waiting for destroy
22            this.destory = true;
23            this.callback.SSActionEvent (this);
24        }
25    }
26
27    public override void Start () {
```

1. 让 Unity 创建动作类，确保内存正确回收。别指望手机开发者是 c 语言高手。

2. 多态。C++ 语言必申明重写，Java则默认重写。

3. 似曾相识的运动代码。动作完成，则期望管理程序自动回收运行对象，并发出事件通知管理者。

面向对象设计思考
（5）组合动作实现

```
5  public class CCSequenceAction : SSAction, ISSActionCallback
6  {
7      public List<SSAction> sequence;
8      public int repeat = -1; //repeat forever
9      public int start = 0;
10
11     public static CCSequenceAction GetSSAction(int repeat, int start , List<SSAction> sequence){
12         CCSequenceAction action = ScriptableObject.CreateInstance<CCSequenceAction> ();
13         action.repeat = repeat;
14         action.sequence= sequence;
15         action.start = start;
16         return action;
17     }
18
19     // Update is called once per frame
20     public override void Update ()
21     {
22         if (sequence.Count == 0) return;
23         if (start < sequence.Count) {
24             sequence [start].Update ();
25         }
26     }
27
28     public void SSActionEvent (SSAction source, SSActionEventType events = SSActionEventType.Competeted,
29     {
30         source.destory = false;
31         this.start++;
32         if (this.start >= sequence.Count) {
33             this.start = 0;
34             if (repeat > 0) repeat--;
35             if (repeat == 0) { this.destory = true; this.callback.SSActionEvent (this); }
36         }
37     }
```

1. 创建一个动作顺序执行序列，-1 表示无限循环，start 开始动作。

2. 执行当前动作。

3. 收到当前动作执行完成，推下一个动作，如果完成一次循环，减次数。如完成，通知该动作的管理者。

面向对象设计思考
（5）组合动作实现

```
37      }
38
39      // Use this for initialization
40      public override void Start () {
41          foreach (SSAction action in sequence) {
42              action.gameobject = this.gameobject;
43              action.transform = this.transform;
44              action.callback = this;
45              action.Start ();
46          }
47      }
48
49      void OnDestory() {
50          //TODO: something
51      }
52  }
```

4. 执行动作前，为每个动作注入当前动作游戏对象，并将自己作为动作事件的接收者。

5. 如果自己被注销，应该释放自己管理的动作。

6. 这是标准的组合设计模式。被组合的对象和组合对象属于同一种类型。通过组合模式，我们能实现几乎满足所有越位需要、非常复杂的动作管理。
7. 建议大家了解 Cocos 2d CCAction 类 或 Unity 动画系统，本课程方法是这些系统的实现的关键技术。

面向对象设计思考
（6）动作事件接口定义

```
5 public enum SSActionEventType:int { Started, Competeted }
6
7 public interface ISSActionCallback
8 {
9     void SSActionEvent(SSAction source,
10         SSActionEventType events = SSActionEventType.Competeted,
11         int intParam = 0 ,
12         string strParam = null,
13         Object objectParam = null);
14 }
15
```

1. 事件类型定义，使用了枚举变量
2. 定义了事件处理接口，所有事件管理者都必须实现这个接口，来实现事件调度。所以，组合事件需要实现它，事件管理器也必须实现它。
3. 这里展示了语言函数默认参数的写法。

面向对象设计思考
（7）动作管理基类 – SSActionManager

```
5  public class SSActionManager : MonoBehaviour {
6
7      private Dictionary <int, SSAction> actions = new Dictionary <int, SSAction> ();
8      private List <SSAction> waitingAdd = new List<SSAction> ();
9      private List<int> waitingDelete = new List<int> ();
10
11     // Update is called once per frame
12     protected void Update () {
13         foreach (SSAction ac in waitingAdd) actions [ac.GetInstanceID ()] = ac;
14         waitingAdd.Clear ();
15
16         foreach (KeyValuePair <int, SSAction> kv in actions) {
17             SSAction ac = kv.Value;
18             if (ac.destory) {
19                 waitingDelete.Add(ac.GetInstanceID()); // release action
20             } else if (ac.enable) {
21                 ac.Update (); // update action
22             }
23         }
24
25         foreach (int key in waitingDelete) {
26             SSAction ac = actions[key]; actions.Remove(key); DestroyObject (ac);
27         }
28         waitingDelete.Clear ();
29     }
```

1. 创建 MonoBehaiviour 管理一个动作集合，动作做完自动回收动作。

2. 该类演示了复杂集合对象的使用。

面向对象设计思考
（7）动作管理基类 – SSActionManager

```
30
31      public void RunAction(GameObject gameobject, SSAction action, ISSActionCallback manager) {
32          action.gameobject = gameobject;
33          action.transform = gameobject.transform;
34          action.callback = manager;
35          waitingAdd.Add (action);
36          action.Start ();
37      }
38
39
40      // Use this for initialization
41      protected void Start () {
42      }
```

3. 提供了运行一个新动作的方法。该方法把游戏对象与动作绑定，并绑定该动作事件的消息接收者。
4. 执行改动作的 Start 方法

面向对象设计思考
（8）实战动作管理

- 该类的职责
  - 接收场景控制的命令
  - 管理动作的自动执行
- 场景控制器与动作管理器的关系
  - 建议场景控制器在 start 中用 GetComponent<T> () 将它作为场景管理的一员
  - 后面的代码主要是演示动作管理。加入场景过程比较奇葩！
  - 动作管理器不应该有模型的知识，游戏对象信息必须由场景控制器提供
- 功能：创建了4 各 CCMoveToAction,
  - 其中两个动作由管理器亲自管理
  - 另两个组成一个顺序执行组合，效果一样

面向对象设计思考
（8）实战动作管理

```
5  public class CCActionManager : SSActionManager, ISSActionCallback {
6
7      public FirstController sceneController;
8      public CCMoveToAction moveToA , moveToB, moveToC, moveToD;
9
10     protected new void Start() {
11         sceneController = (FirstController)SSDirector.getInstance ().currentSceneController;
12         sceneController.actionManager = this;
13         moveToA = CCMoveToAction.GetSSAction (new Vector3 (5, 0, 0), 1);
14         this.RunAction (sceneController.move1, moveToA, this);
15         moveToC = CCMoveToAction.GetSSAction (new Vector3 (-2, -2, -2), 1);
16         moveToD = CCMoveToAction.GetSSAction (new Vector3 (3, 3, 3), 1);
17         CCSequenceAction ccs = CCSequenceAction.GetSSAction (3, 0, new List<SSAction> {moveToC, moveToD});
18         this.RunAction (sceneController.move2, ccs, this);
19     }
20
21     // Update is called once per frame
22     protected new void Update ()
23     {
24         base.Update ();
25     }
26
27     #region ISSActionCallback implementation
28     public void SSActionEvent (SSAction source, SSActionEventType events = SSActionEventType.Competeted, ir
29     {
30         if (source == moveToA) {
31             moveToB = CCMoveToAction.GetSSAction (new Vector3 (-5, 0, 0), 1);
32             this.RunAction (sceneController.move1, moveToB, this);
33         } else if (source == moveToB) {
34             moveToA = CCMoveToAction.GetSSAction (new Vector3 (5, 0, 0), 1);
35             this.RunAction (sceneController.move1, moveToA, this);
36         }
37     }
38     #endregion
```

1. 这是实战的管理器。所以它需要与场景控制器配合。

2. Update 是方法覆盖，提醒编程人员重新该方法不会多态，且要用 base 调用原方法。

自学内容:
C# 常用集合类型

- ArrayList
- HashSet
- HashTable
- List
- Dictionary

课程小结

- 游戏对象
  - 游戏对象由它拥有的组件决定功能
  - 摄像机相机与组件
  - 光线组件
  - 3D 对象组件
  - 纹理、材料、着色器
  - 游戏环境设计与地形编辑
- 面向对象设计技巧
  - 虚方法、改写方法、多态、覆盖方法
  - 组合模式
  - 接口与事件通知