



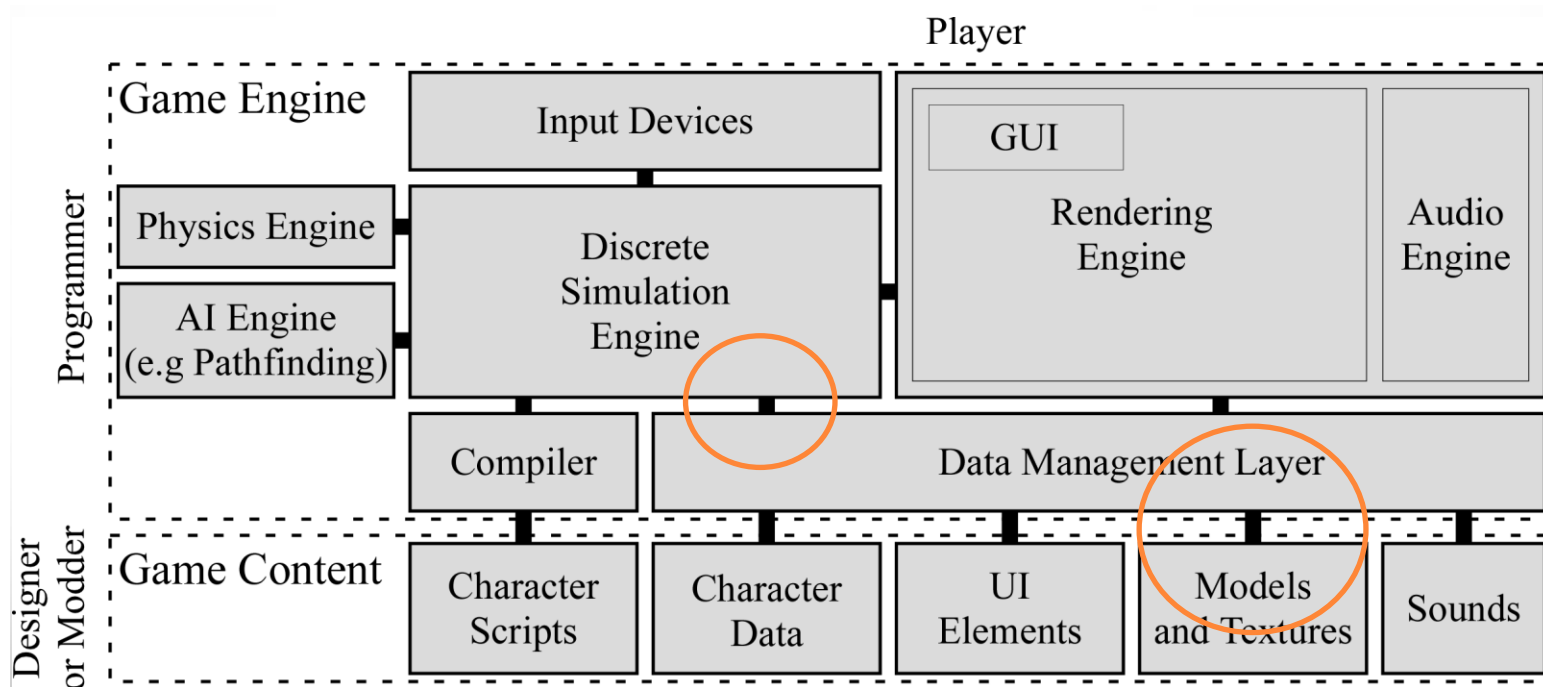
INTRODUCTION TO COMPUTER 3D GAME DEVELOPMENT

Model and Animation (1)

潘茂林, panml@mail.sysu.edu.cn

中山大学·软件学院

游戏引擎架构



游戏引擎架构

- 3D 游戏引擎资源涉及内容
 - 3D 模型与动画
 - Materials and Shaders 材质与着色器（已简单介绍）
 - Particle System 粒子系统
 - UI 系统
 - Texture 2D 二维纹理（自学）
 - Procedural Materials 程序材质（自学）
 - Movie Texture 影片纹理（自学）
 - Audio Files 音频文件（自学）
-
- 驾驭资源也是编程重要内容



目录

- 模型与动画
 - 基本概念
 - 模型的构成
 - 预制与模型
- Mecanim动画系统
 - 动画组件
 - 动画状态机
 - 动画编程
 - 动画事件
- 面向对象的编程思考
 - 控制反转（IOC）技术
 - 发布与订阅（Observer）设计模式



模型

(1) 基本概念

○ 模型 (Model)

- 物体对象的组合, Unity 映射为游戏对象树
- 每个物体包含一个网格 (Mesh) 与蒙皮设置
- 包含使用的纹理、材质以及一个或多个动画剪辑
- 模型由 3D 设计人员使用 3D Max 等创建

○ 动画剪辑 (Animation Clip)

- 物体或骨骼的位置等属性在关键帧的时间序列
- 通过**插值算法**计算每帧的物体的位置或属性

○ 游戏物体显示

- Mesh 表面网格
- MeshRender 网格渲染器



课堂实验（一）

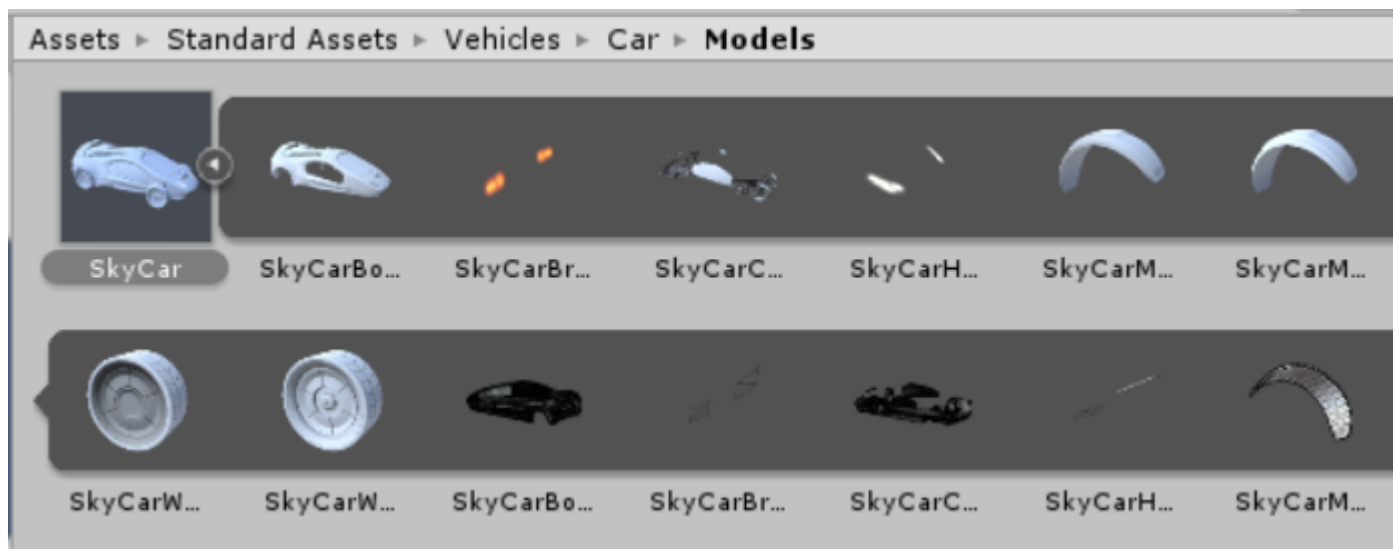
模型构成与直观（1）

○ 创建新项目 Animation

- 准备资源

1. 菜单 assets → Import Package → Vehicles
2. 选择所有内容导入
3. 下载 birds.zip 解压，将 birds 目录拖入项目

- 查看模型的属性（Inspect）



课堂实验（一）

模型构成与直观（2）

- 点击模型与每个部件



- 模型
 - 模型库包括许多内容，模型对象、材料、代码等
 - 物体模型包括：
 - 子物体
 - 网格
 - 动画



模型

(2) 模型使用

○ 模型的格式

- fbx 格式, dae (collada 一种开源的3d格式)
- 3dmax, maya 等格式

○ 使用预制加载模型, 例如:

- 放置 plane 和 car 预制
- 运行
- 按“下箭头”倒车
- 研究 Car 的物体组织



模型

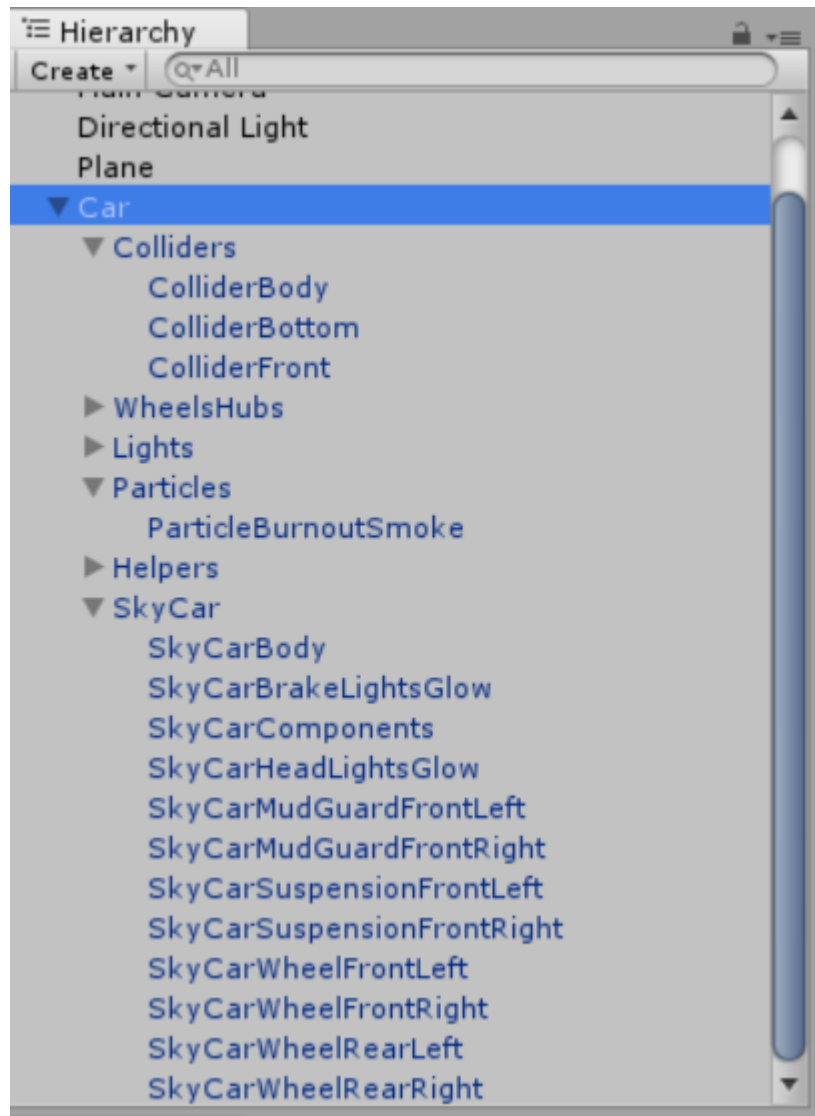
(3) 预制模型

○ Car 预制结构

- 哪些物体有刚体
- 碰撞器设计
- 哪些是显示部件
- 启动的烟雾
- Lights?
- 控制代码

○ 为 crow（乌鸦）建预制

- crow对象结构
- 碰撞器?
- 飞行?



模型

(4) 使用模型注意事项

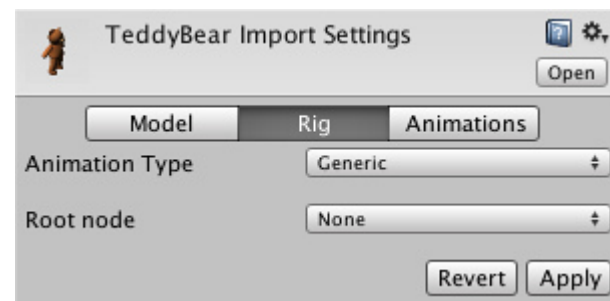
○ Unity 从5.x 开始支持新动画模型

- 模型骨骼（Rig）的动画类型选择

- None (没有动画)。如 skycar
- Legacy (遗留动画)。一般不用，如必须使用请自学
- Generic (普通动画)。如果资源是遗留动画，请将动物们改为普通动画。但带器械的人物设计只能使用普通动画
- Humanoid (人形动画)。可以通过将物体各部件映射的人的身体部件，从而支持类似人的物体（如外星人、机器人、猩猩等）使用同一套动画模型。

- 本课程不再讲解遗留动画机制，如有需要：

- 看以前的课件
- 网上资料很多，特点 **Animation 组件**

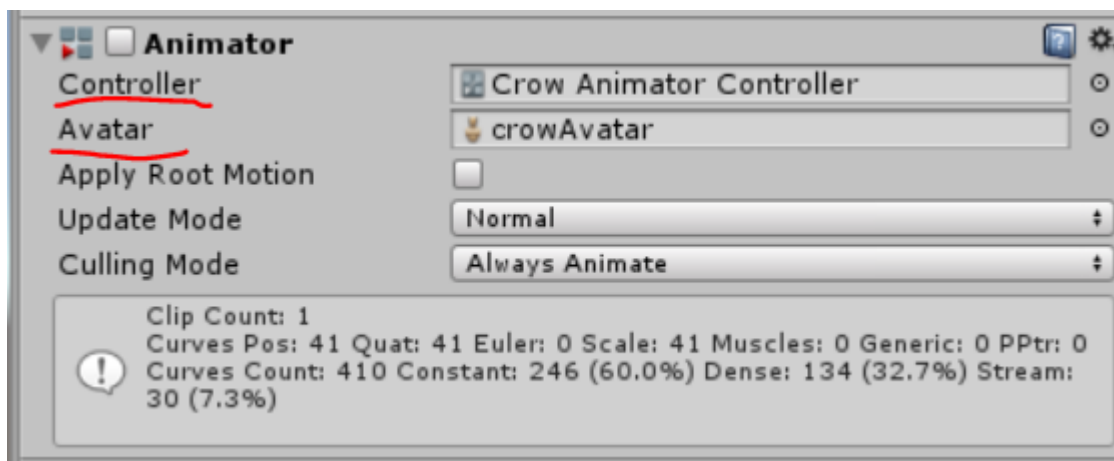


MECANIM动画系统

(1) 动画部件

○ Animator 部件

- 有动画的模型编辑器实例化时会自动添加
- 你也可以在根对象添加该部件



注意：有碰撞器的物体，**运动学与物理不宜混合使用**

○ Animation 部件

- 遗留动画组件



MECANIM动画系统

(2) 动画状态机

○ Mecanim使用状态机管理运动

- 状态机模型:

$$FSM = (\Sigma, S, s_0, \delta, F) \text{ where } \delta: S \times \Sigma \rightarrow S$$

- 状态集 (S) :

- 运动状态、复合运动状态、特殊状态

- 特殊状态:

- Entry (s_0) : 开始状态

- AnyState: 任意状态

- Exit (F) : 终止状态

- 变迁 (δ) :

- 在输入(Σ)/事件驱动下, 转入下一个状态

- 状态机特点

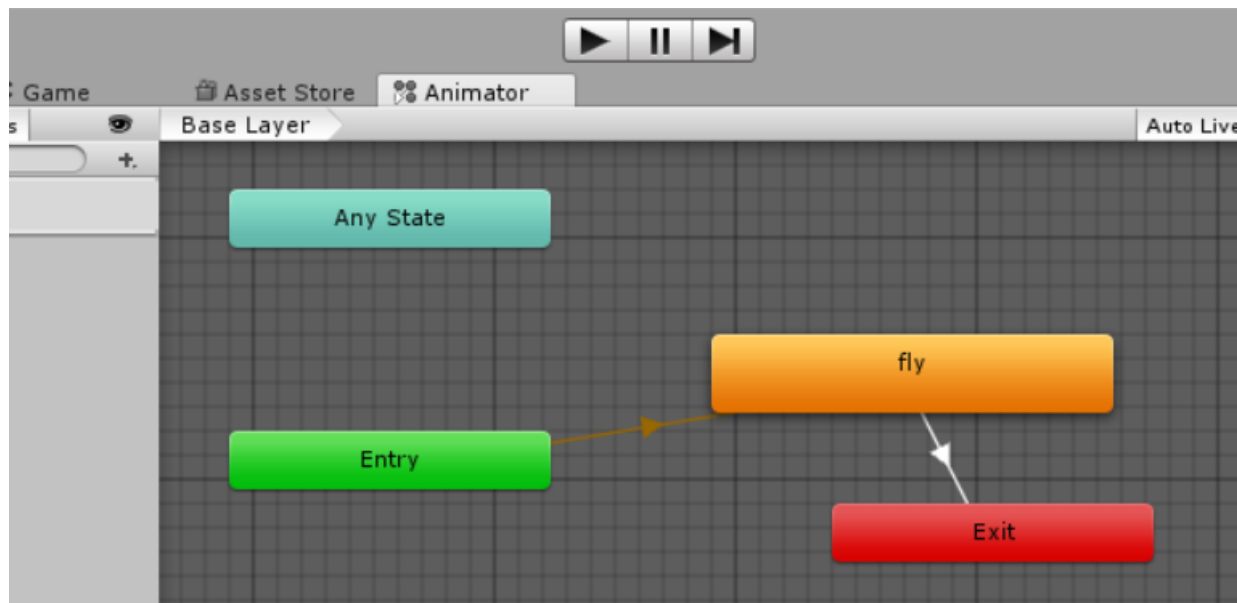
- 任意时刻有**仅有**一个活跃的状态或变迁



MECANIM动画系统

(2) 动画状态机

- 创建动画状态机(CrowAnimatorController)
 - 菜单 Assets → Create → Animator Controller
 - 编辑状态机



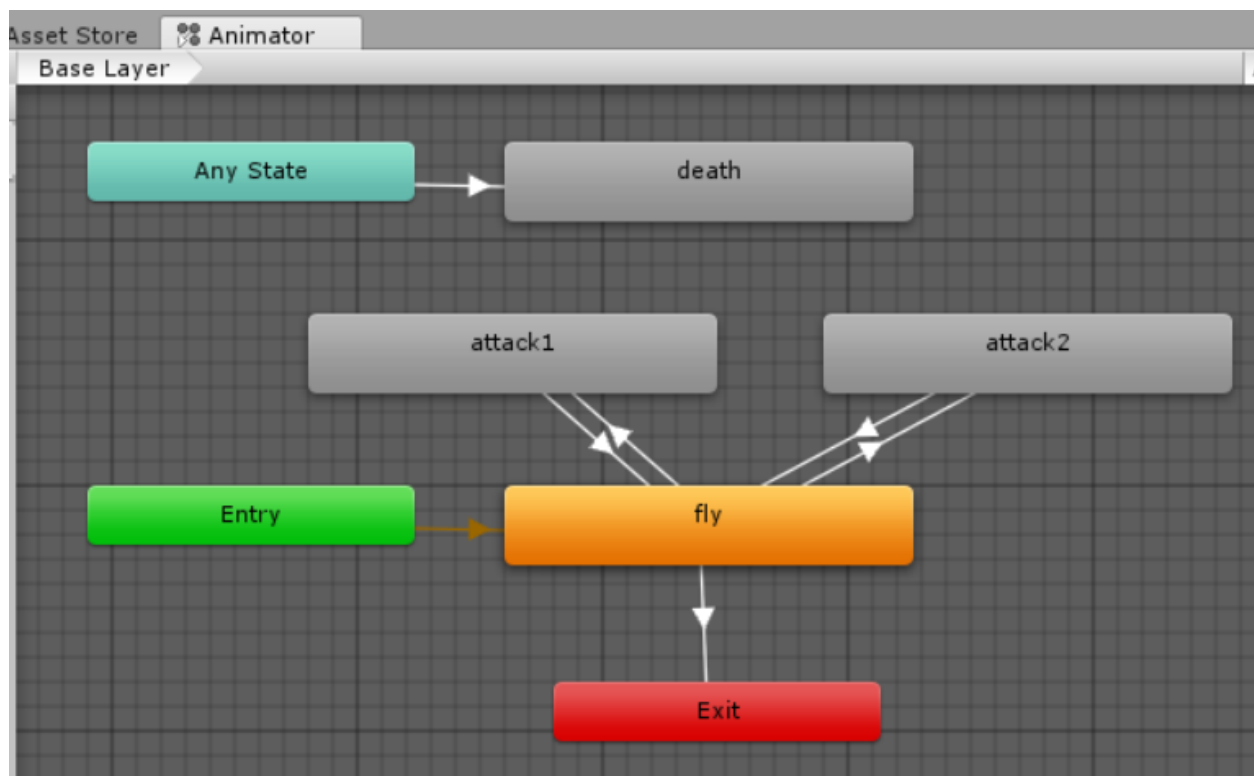
- 在Crow模型中，将fly动画拖入编辑器
- 在fly状态点右键，创建变迁。指向退出。
- 将动画状态机拖入Animator组件，运行！



MECANIM动画系统

(2) 动画状态机

- 状态（State）与变迁（Transistion）
 - 创建如图状态图，控制器如何执行动画呢？



关键要理解：状态、变迁、条件

MECANIM动画系统

(2) 动画状态机

○ State 属性

- Motion 动画剪辑
- Speed 速度（倍率）
- Foot IK?
- ...
- Transtions
 - 按顺序检测生效转移
 - Solo 优先检测转移
 - Mute 禁止转移



MECANIM动画系统

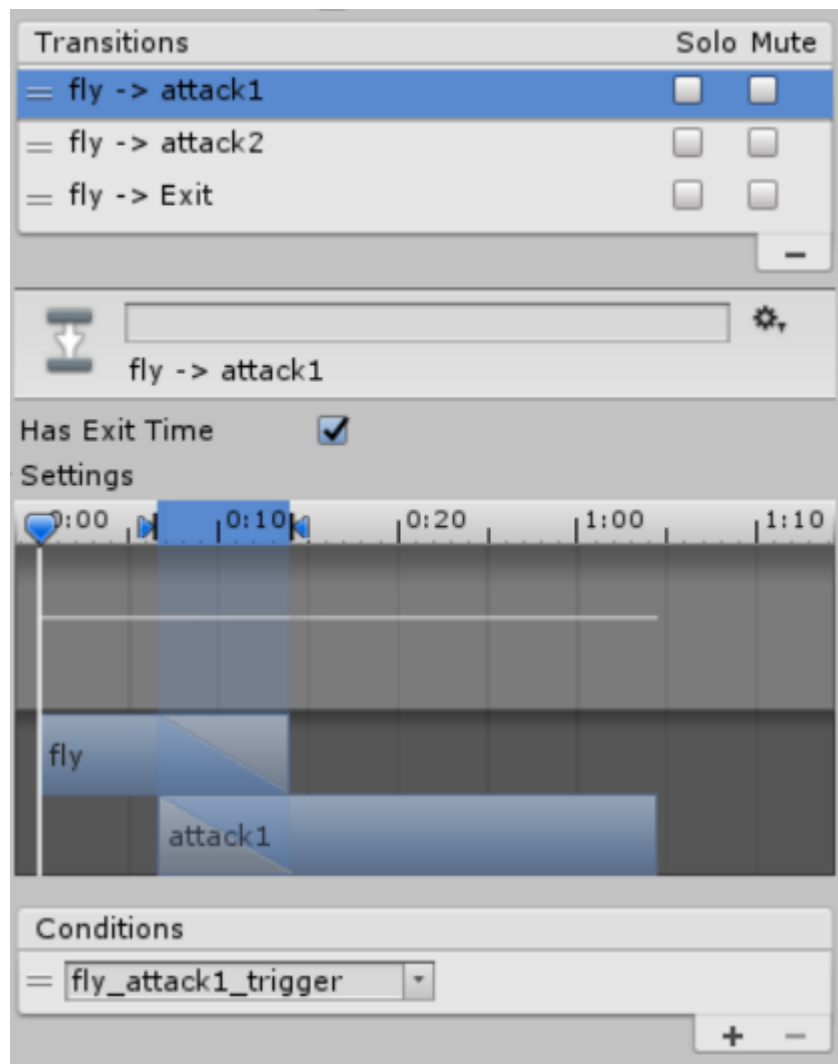
(2) 动画状态机

○ 控制变迁

- 注意变迁的顺序
- 设定 solo 或 mute
- 设定每个变迁
 1. 给变迁命名，便于控制
 2. 是否使用动画结果条件
 3. 变迁动画混合
 4. 变迁条件

○ 例如：

- Fly->exit 条件是 live
- Not live 从任意状态转死亡

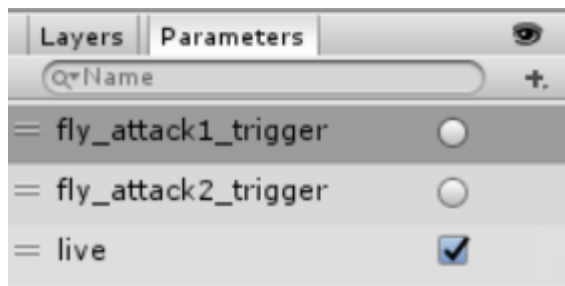


MECANIM动画系统

(2) 动画状态机

○ 使用转移控制变量

- Float, Int, Bool 类型
- Trigger 类型



○ 规划转移变量与发生条件

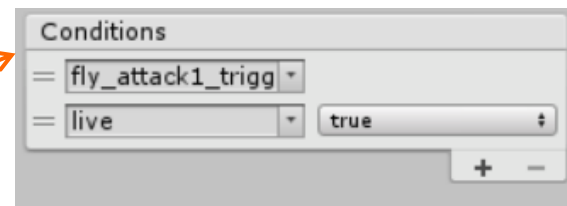
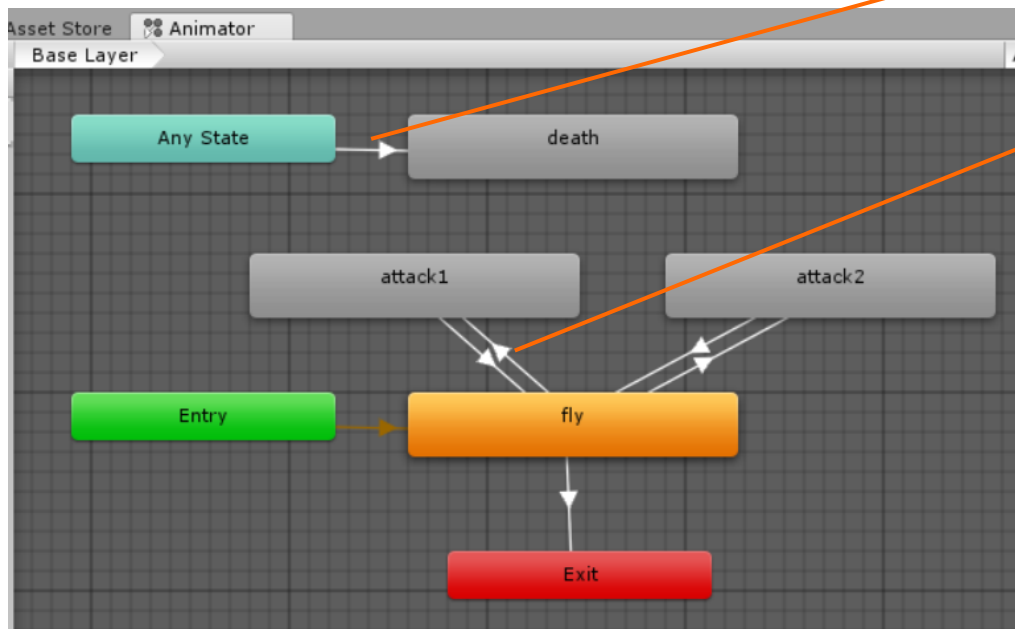
- 转移变量设计
 - 建议多使用 **trigger** 类型变量
 - 确保转移条件唯一，避免使用顺序决定转移（位操作通常OK）
 - 使用 **mute** 关闭不用的转移
- 条件设计
 - **live = false** 转入死亡
 - **fly_attack_trigger** 确定转入攻击



MECANIM动画系统

(2) 动画状态机

○ 变迁设置

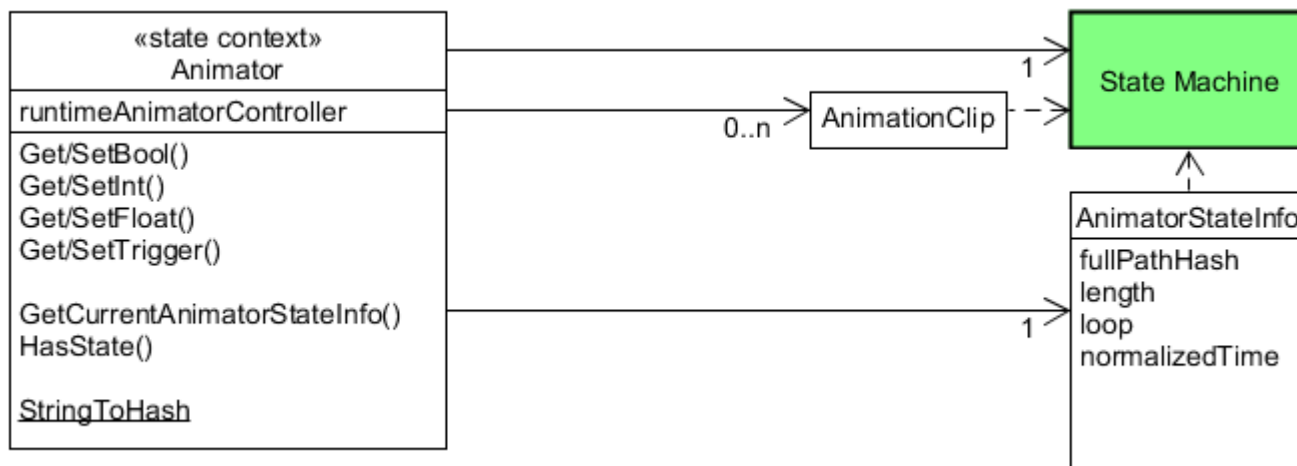


MECANIM动画系统

(3) 动画基本编程

Animator 运行时属性与方法

- 设计时，功能强大
- 运行时，你甚至无法查询设计了哪些状态



- 通过参数控制状态机才是正解！！！！

课堂实验（二）

控制小鸟飞行游戏

```
5 public class CrowController : MonoBehaviour {
6
7     public float speed = 4.0f;
8     private Animator ani;
9     private Rigidbody rig;
10
11     // Use this for initialization
12     void Start () {
13         ani = GetComponent<Animator> ();
14         rig = GetComponent<Rigidbody> ();
15     }
16
17     // Update is called once per frame
18     void FixedUpdate () {
19         float high = Input.GetAxis("Vertical") * speed;
20         rig.AddForce (Vector3.up * (high + 7), ForceMode.Force);
21         float right = Input.GetAxis("Horizontal") * speed;
22         rig.AddForce (Vector3.right * right, ForceMode.Force);
23
24         if (Input.GetButtonDown("Fire1")) {
25             ani.SetTrigger ("fly_attack1_trigger");
26         }
27     }
28
29     void OnCollisionEnter() {
30         ani.SetBool("live", false);
31     }
```

1. 请自己思考刚体、碰撞体设定
2. 加空气阻力有利控制



MECANIM动画系统

(4) 动画事件

○ 状态行为与事件

- StateMachineBehaviour 对象
 - 对象继承 ScriptableObject
 - 绑定特定状态，执行特定行为
 - 工作原理类似 SSAction
- StateMachineBehaviour 对象事件处理器
- 常用案例（大招）
 1. Enter 状态，放一个烟雾效果
 2. Exit 状态，结束效果
- 常用案例（**攻击事物**）！！！！
 1. 动作 Update 到指定时间（位置）
 2. 开始执行爆破、计分等任务

Messages

[OnStateEnter](#)

[OnStateExit](#)

[OnStateIK](#)

[OnStateMove](#)

[OnStateUpdate](#)



MECANIM动画系统

(4) 动画事件

○ MonoBehaviour 事件

OnAnimatorIK

Callback for setting up animation IK (inverse kinematics).

OnAnimatorMove

Callback for processing animation movements for modifying root motion.

- Animator 会通知游戏对象的行为，当
- 反向运动求解器启动时（主要是脚部平滑）
 - 参见官方手册
- 物体动作基准坐标（中心）移动时
 - 尽管 Animator 有 apply root motion 属性
 - 人跑动时，动画速度与跑动速度一致
 - 自定义任务处理



MECANIM动画系统

(4) 动画事件

○ 处理自定义事件(思考)

```
5 public class AttackStateBehaviour : StateMachineBehaviour {
6
7     // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
8     override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {
9         //The integer part is the number of time a state has been looped.
10        //The fractional part is the % (0-1) of progress in the current loop.
11        if (stateInfo.normalizedTime > 0.7f) {
12            // This is EventSource
13            // How to trigger the event to Subscribers ???
14        }
15    }
16 }
```

- 需求1: 每个动作在特定位置能仅能触发一次事件
- 需求2: 如何与对该动画事件感兴趣的对象交互
- 这里不合适处理业务逻辑, 任务就是检测事件
- 面向对象设计就是**基于职责的设计**



面向对象设计思考

(1) 反转控制 (IOC) – 依赖

○ 什么是依赖？

- 对象 (A) 的改变会导致对象 (B) 的行为发生变化，则 B 依赖 A：

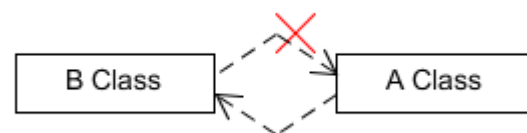
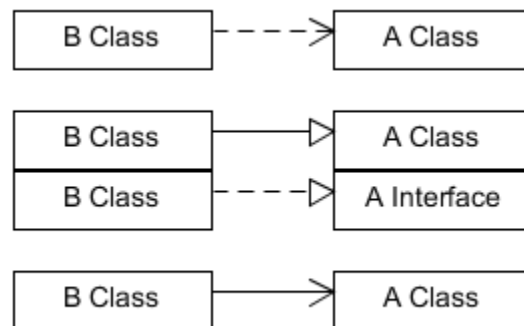
- B 继承或实现 A
- B 使用 A 作为对象实例
- B 使用 A 作为函数参数或局部变量（调用）

- 循环依赖（直接与间接都算）

- B 和 A 是不可分割的
- 例如：gameobject – component
- 优点：便于集中管理，如果提供稳定的核心代码，易于使用
- 缺点：无法解耦，只能按业务功能模块共享；难以扩展

- 不合理依赖问题

1. 循环依赖的一组对象，需要划分
2. B 依赖具体功能对象，导致难以扩展

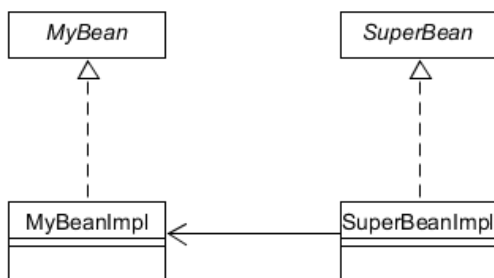


面向对象设计思考

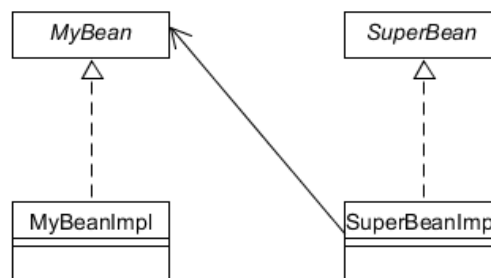
(1) 反转控制-IOC

○ 反转控制

- B 依赖 A，变为 B 依赖 A 的抽象，或 A 的接口。且实现抽象/超类或行为/接口的对象由第三方注入 B
- 第三方可能是：
 - 工厂对象
 - 单实例对象
- 或者，由 A 或 A 的管理者主动注入对象到 B，而 B 并不了解 A 的具体实现，这一过程为控制反转。



原来两个对象的依赖关系

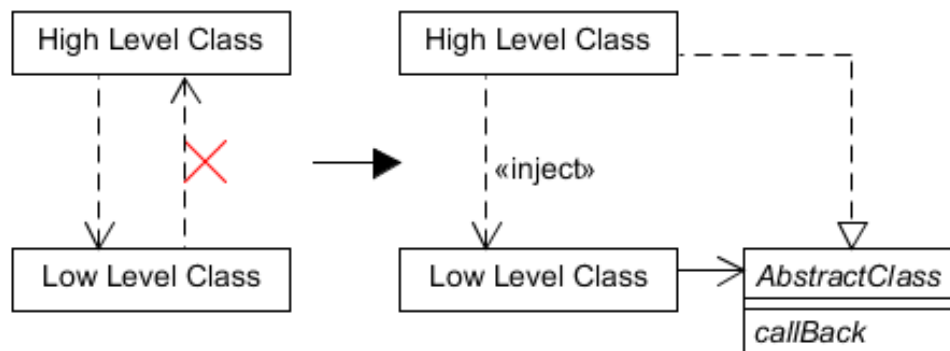


使用Spring后两个对象的依赖关系
Spring Context 完成MyBean的实例化

面向对象设计思考

(1) 反转控制—IOC

- 在层次模型中，这种依赖反转就特别明显



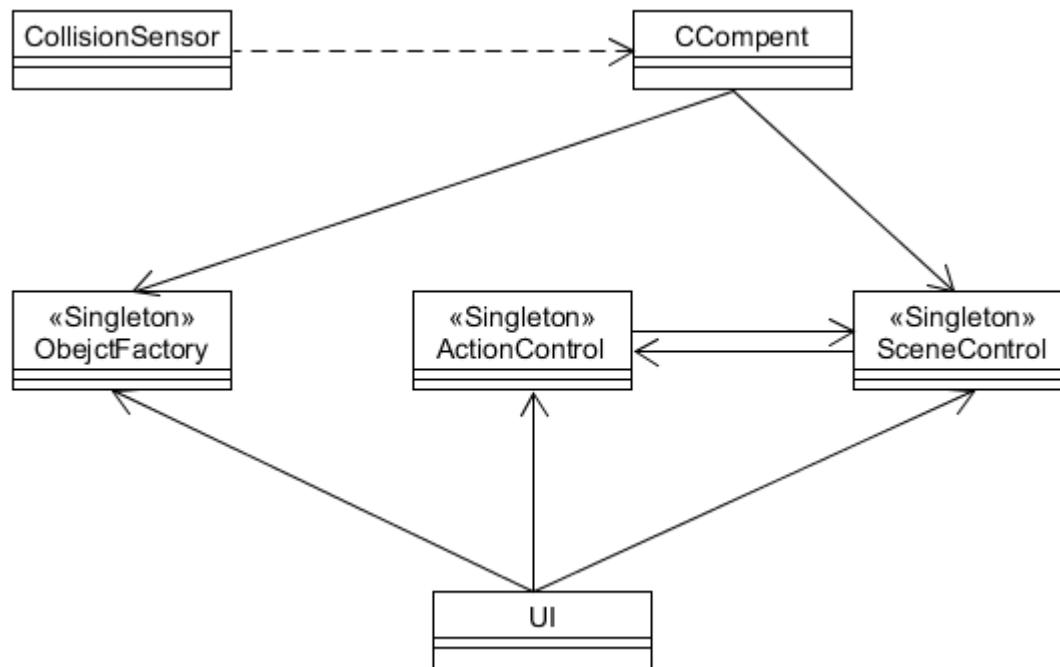
IoC 解决了核心类、底层类不知道使用者行为的基本问题。例如，**car**类代码，并不知道发生碰撞后，游戏处理的逻辑。

它是构造分层架构、模块结构软件的基础方法。实现了高层控制底层软件行为。

面向对象设计思考

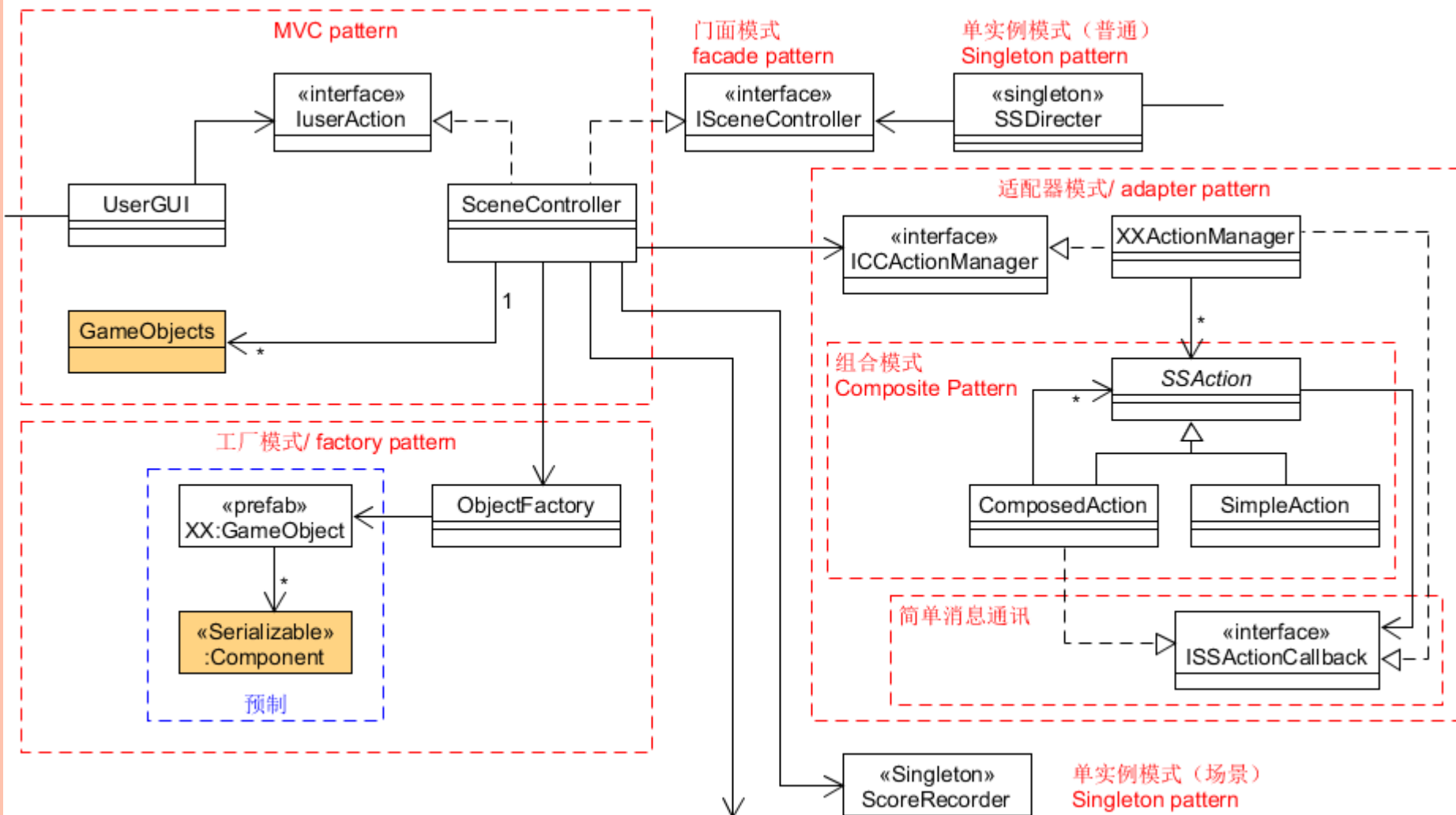
(1) 依赖实战案例分析

- 某同学设计的对象依赖，好吗？



面向对象设计思考

(1) 依赖实战案例分析（课程推荐）

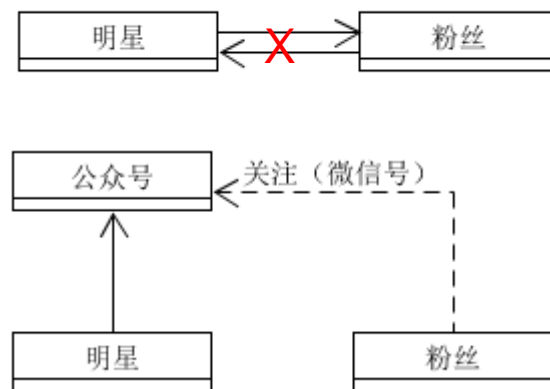


面向对象设计思考

(2) 粉丝与明星 - 发布与订阅模型

○ 粉丝与明星互动的启示

- 明星与粉丝成为朋友密切互动?
- 明星关注粉丝的行为与反馈?
- 明星通过媒体发布各种事件 (狗粮)
- 粉丝关注相关媒体
- 媒体向粉丝推送事件与数据
- 粉丝们收到事件后, 或欢笑或悲伤



○ 模型化与概念

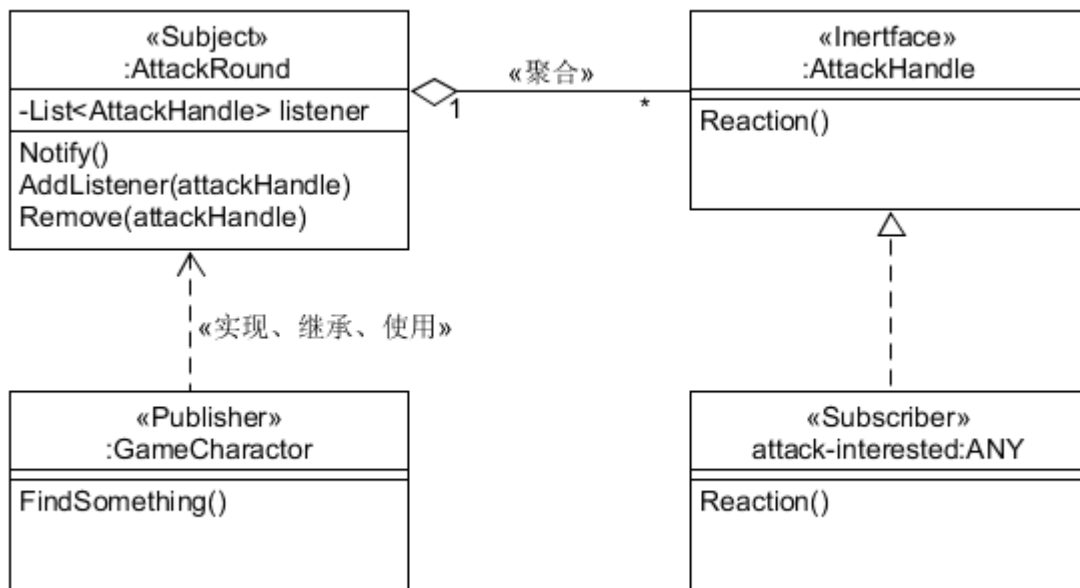
- 发布者 (事件源) /Publisher: 事件或消息的拥有者
- 主题 (渠道) /Subject: 消息发布媒体
- 接收器 (地址) /Handle: 处理消息的方法
- 订阅者 (接收者) /Subscriber: 对主题感兴趣的人



面向对象设计思考

(2) 发布与订阅模式 (PUB/SUB PATTERN)

○ Publisher/Subscriber Pattern



- 也称为观察者模式 (Observer Pattern)

- 发布者与订阅者没有直接的耦合
- 是实现模型与视图分离的重要手段
- 例如：数据DataSource对象，就是Subject。任何使用该数据源的显示控件，如Grid都会及时更新。

面向对象设计思考

(2) 发布与订阅模式 (PUB/SUB PATTERN)

○ 设计模式与游戏的持续改进

- 当一个游戏对象实现“击打”行为，可能的处理是：
 - 按规则计分
 - 按规则计算对周边物体的伤害
 - 显示各种效果
 -
- 如果你在事件写了以上代码
 - 你想修改游戏规则，你将无法保证修改正确，因为很多行为都有类似的代码
 - 添加一些新行为，你不仅无法复用代码，而且产生逻辑冲突
- 如果使用设计模式
 - 计分员对象感兴趣该事件，会计分（编程时哪有合理规则！）
 - 控制器感兴趣这个事件，会按规则做响应
 - 添加新需求，如生成奖励对象，则添加一个奖励管理者



面向对象设计思考

(3) 发布与订阅模式实现

- 在 C# 中十分简单，定义事件源

```
5 public class XXEventManager : MonoBehaviour
6 {
7     public delegate void AttackAction(Object sender, string info);
8     public static event AttackAction OnAttackAction;
9
10    void Update() {
11        //... ..
12        FindSomething();
13        //... ..
14    }
15
16    void FindSomething()
17    {
18        if (GUI.Button(new Rect(Screen.width / 2 - 50, 5, 100, 30), "Click"))
19        {
20            if(OnAttackAction != null)
21                OnAttackAction(this, "attack1");
22        }
23    }
24 }
```

1. 定义回调函数类型
2. 定义subject
3. 发出通知，事件由谁处理，如何处理都不需要知道!!!

面向对象设计思考

(3) 发布与订阅模式实现

○ 假设 SceneController 处理该事件

```
void OnEnable()  
{  
    XXEventManager.AttackAction += Teleport; //regist  
}
```

```
void OnDisable()  
{  
    XXEventManager.AttackAction -= Teleport; //remove  
}
```

```
void Teleport(Object sender, string info)  
{  
    Vector3 pos = transform.position;  
    pos.y = Random.Range(1.0f, 3.0f);  
    transform.position = pos;  
}
```

当你的程序象积木一样，用时加上，不用时撤去。而程序都是可以按设计逻辑运行，面向对象设计就懂了。



课堂实验（三）

修改小鸟飞行游戏

○ 实验目的：

- 实现事件源与业务处理代码分离

○ 实验要求：

- 主程序控制器加载“小鸟”后，等待用户按开始按钮
- 计分程序关注小鸟存活时间，每10秒计？ 分



自学内容： C# 与发布订阅模式

- 课程内容对于中小游戏已足够
- 自学内容（一）
 - Delegate 委托
 - Event 事件

Events:

<https://unity3d.com/cn/learn/tutorials/topics/scripting/events?playlist=17117>



课程小结

○ 模型

- 模型是由网格、材料与骨骼动画构成
- 模型预制包括定义刚体、碰撞、脚本等

○ Mecanim动画系统

- Animator组件与属性
- 状态机原理与动画设计
- 动画控制编程
- 动画事件与自定义事件实现

○ 面向对象设计思考

- IOC在程序设计中作用
- 观察者模式在游戏中的应用



作业 (LAB 7) :

○ 无

○ 智能巡逻兵

- 提交要求：仅博客
- 游戏设计要求：
 - 创建一个地图和若干巡逻兵；
 - 每个巡逻兵走一个3~5个边的凸多边形，位置数据是相对地址。即每次确定下一个目标位置，用自己当前位置为原点计算；
 - 巡逻兵碰撞到障碍物如树，则会自动选下一个点为目标；
 - 巡逻兵在设定范围内感知到玩家，会自动追击玩家；
 - 失去玩家目标后，继续巡逻；
 - 计分：每次甩掉一个巡逻兵计一分，与巡逻兵碰撞游戏结束；
- 程序设计要求：
 - 必须使用订阅与发布模式传消息、工厂模式生产巡逻兵

