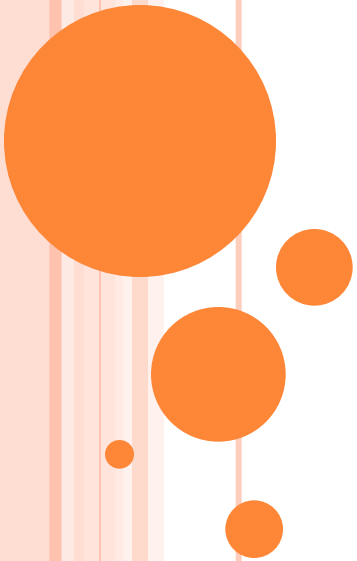
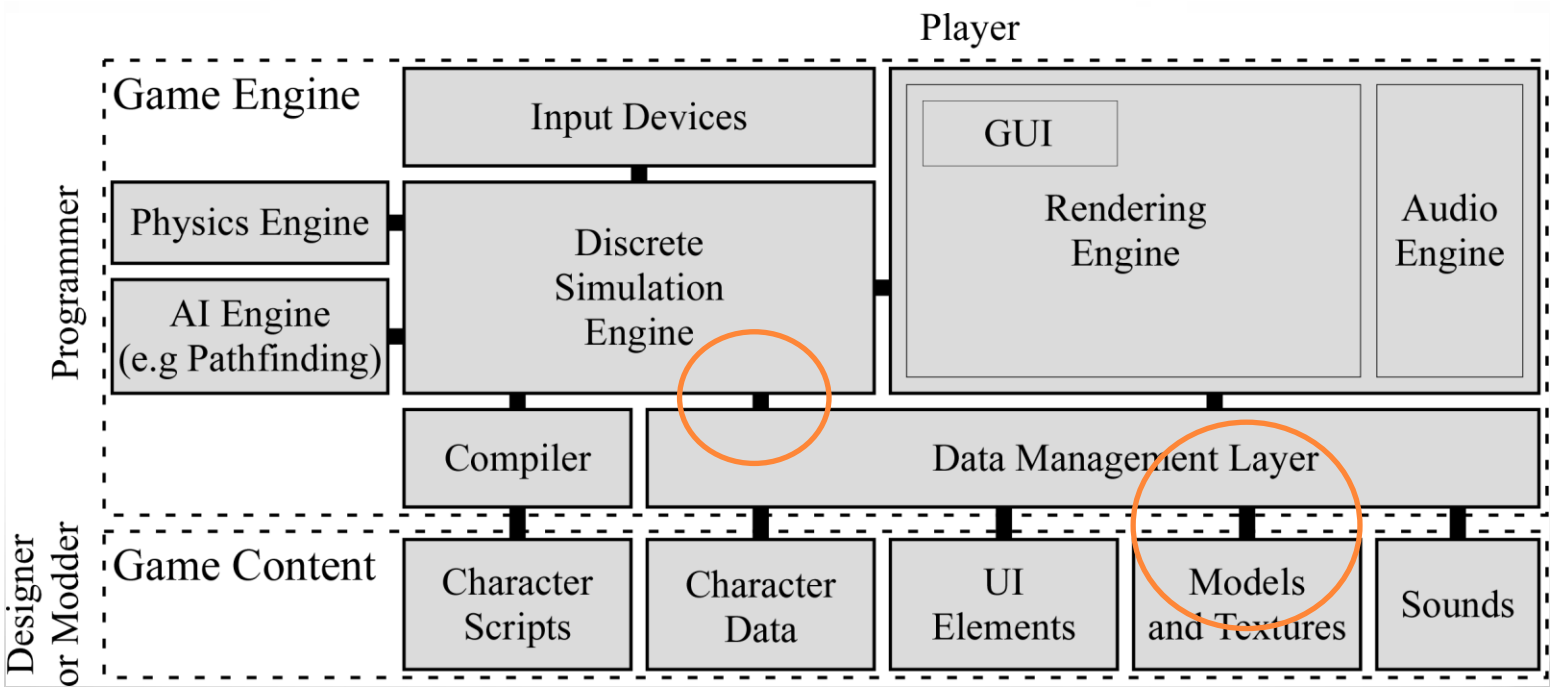


Introduction to Computer 3D Game Development

Model and Animation（2）

潘茂林，panml@mail.sysu.edu.cn
中山大学·软件学院

游戏引擎架构

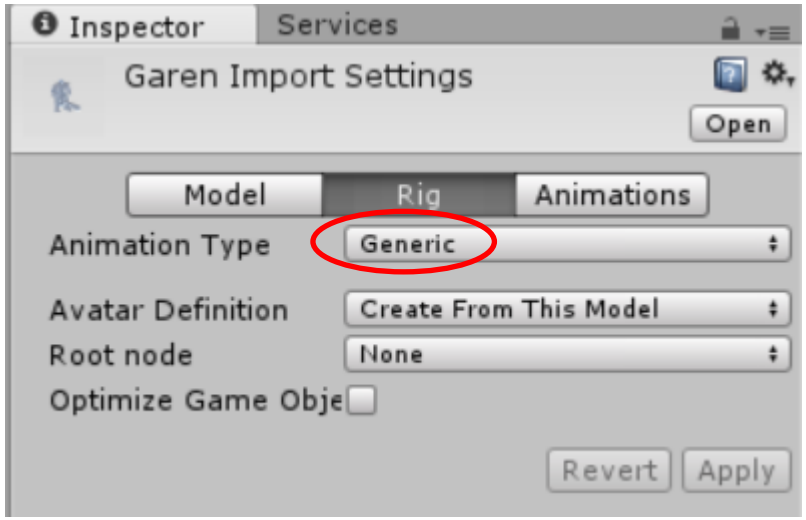

Player
Game Engine
Input Devices
GUI
Rendering Engine
Audio Engine
Physics Engine
Discrete Simulation Engine
AI Engine (e.g Pathfinding)
Programmer
Compiler
Data Management Layer
Designer or Modder
Game Content
Character Scripts
Character Data
UI Elements
Models and Textures
Sounds

目录

- 动画剪辑（Animation Clip）
  - 动画导入
  - 动画设计与编辑
  - 动画剪辑事件
  - 类人动画
- 动画中级技术
  - 直接播放
  - 动画覆盖控制器
  - 混合树（Blend Trees）
- 面向对象的编程思考
  - 设计模式
  - 反射技术

动画剪辑（Animation Clip）
（1）动画导入

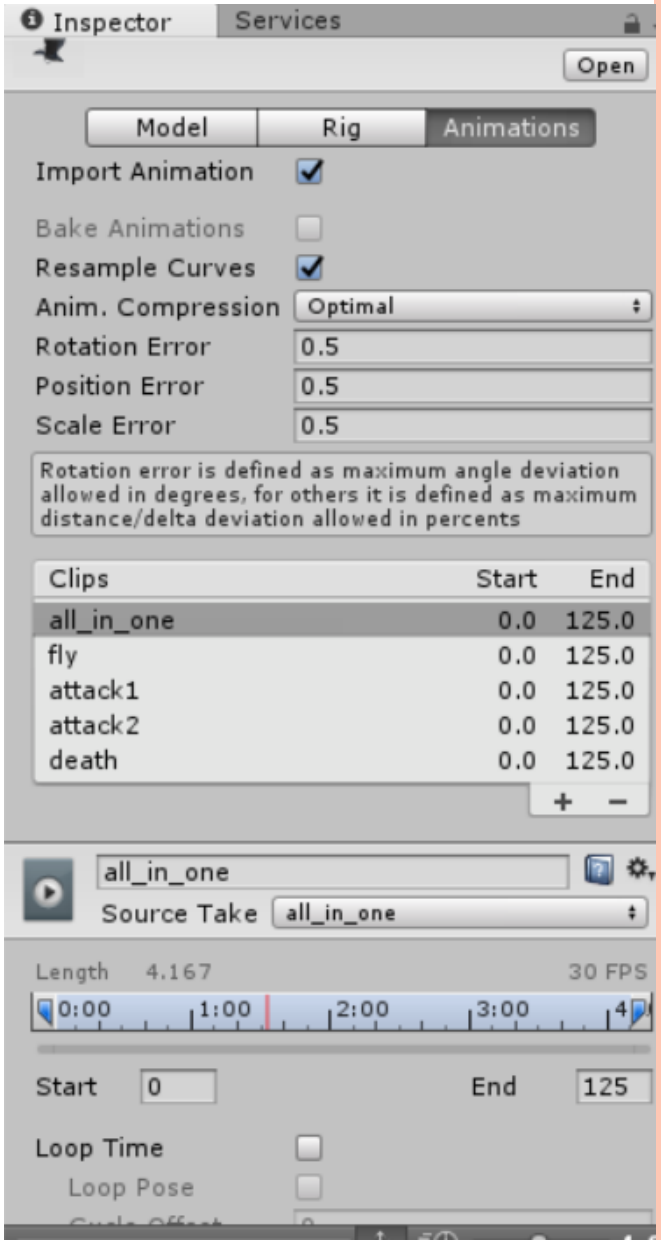
- 导入动画
  - 导入动画，只需拖放模型文件到项目的Assets文件夹。
  - 例如：拖 Garen 目录成为你的项目资源
    - 用 Inspector 观察所有模型和动画
    - 如果是传统动画，则必须改为 Generic
    - Model 是导入参数，一般不需修改
    - Garen是多模型导入，动画命名“模型名@动画名.fbx”



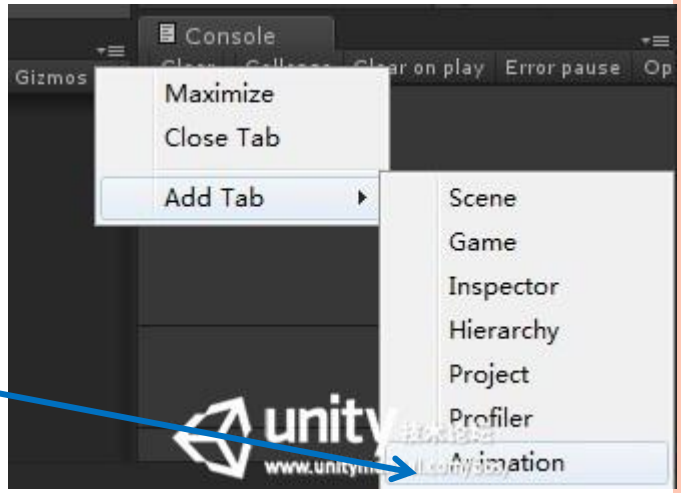
动画剪辑（Animation Clip）

（1）动画导入

- 分割动画
  - 部分动画需要分割



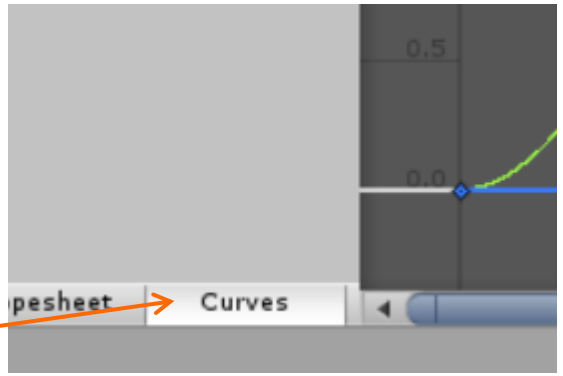
动画剪辑（Animation Clip）
（2）动画编辑 – 创建简单动画



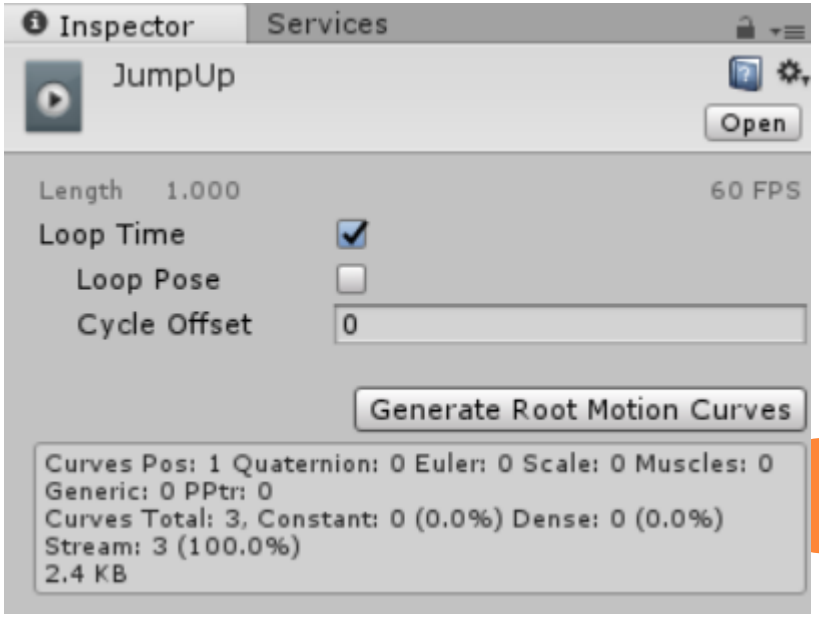
- 准备
  - 创建资源目录 animation
  - 创建游戏对象 cube
  - 添加 animation 编辑窗口
- 操作
  - 选择 cube，postion(0,0,0)
  - 在编辑窗口创建 “JumpUp” 的 Animation Clip
  - 添加一个属性 transtion.postion
  - 拖动编辑红线到中间，输入 postion(0,3,0)
  - 系统在该点生成一个关键帧
  - 拖动编辑红线到尾部，保证位置(0,0,0)
  - 结束编辑（红色录制按钮）
  - 运行！Cube 在跳动

动画剪辑（Animation Clip）
（2）动画编辑 – 动画原理

- 动画原理
  - 属性，控制的属性
  - 关键帧，关键点的值
  - 动画曲线，插值计算每帧值



- AnimationClip属性
  - 选择 JumpUp 动画资源
    - Loop Time
      - 沿时间轴循环
    - Loop pose
      - 动作复位
    - Generate Root …
      - 产生用于 Root 的曲线



动画剪辑（Animation Clip）
（2）动画编辑 – 导出动画

- 模型动画是只读的
  - 使用 “Ctrl-D” 就可以导出动画
  - 导出的动画就是可编辑的
- 实战
  - 选择 Garen@Attack3的动画
  - 按 “Ctrl-D” 并将导出的动画资源放入animation目录
  - 创建一个 Garen 游戏对象
  - 将刚才产生状态机拖入 Garen，运行！
  - 在将 Attack3 加入状态机，并作为下一个动作
  - 运行！
  - 你将看到 Garen 挑起然后转一圈！
  - 你现在可以用编辑器编辑这些 Clip 了

课堂实验（一）
简单动作设计

- Garen大招
  - 给人物穿上衣服
  - 修改 Attack3，效果：跳起的同时旋转一圈

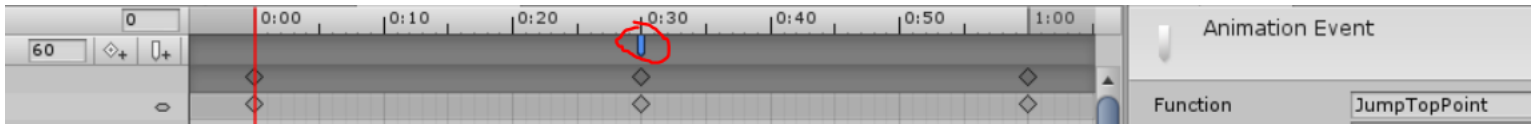
- 设计一个红色按钮，实现动画
  - 开灯：按钮按到底部，回弹到约一半高度
  - 关灯：按钮按到底部，回弹到全部高度

动画剪辑（Animation Clip）
（3）动画剪辑事件

- 添加事件处理代码，并挂载到 garen

```
5 public class GarenEvent : MonoBehaviour {
6
7     public delegate void GarenSubject (GameObject self, string message);
8     public static event GarenSubject OnGarenSubjectNotify;
9
10    void JumpTopPoint() {
11        Debug.Log("Jump to the top point!!!");
12        if (OnGarenSubjectNotify != null)
13            OnGarenSubjectNotify (this.gameObject, "AtTop");
14    }
15 }
16
```

- 添加事件

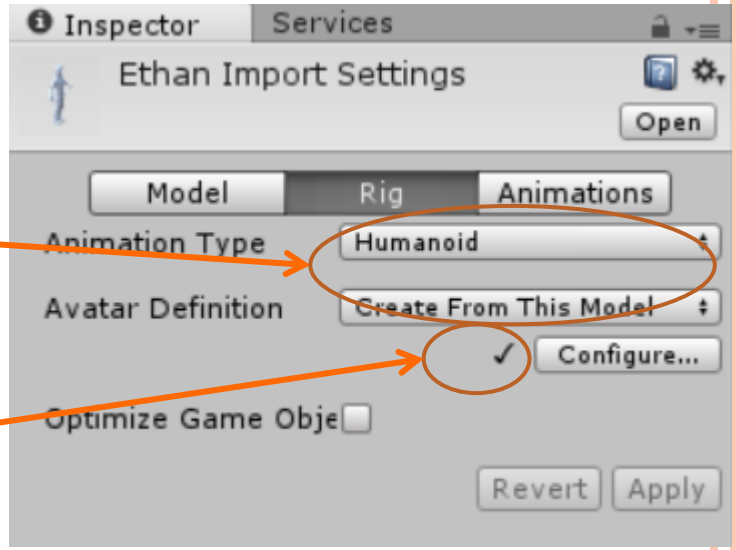


- 这时差不多可以预制该对象了！

动画剪辑（Animation Clip）
（4）类人动画 -- Avatar

- 类人动画
  - 类人骨架是非常常用的特例并且在游戏中广泛使用
  - 骨骼结构的类似性，使得把动画从一个类人骨架映射到另一个成为可能。特别的，即使花草这样的植物，只要有清晰的身体，头，四肢等就可以。
  - 这个映射对象，就是 Avatar！

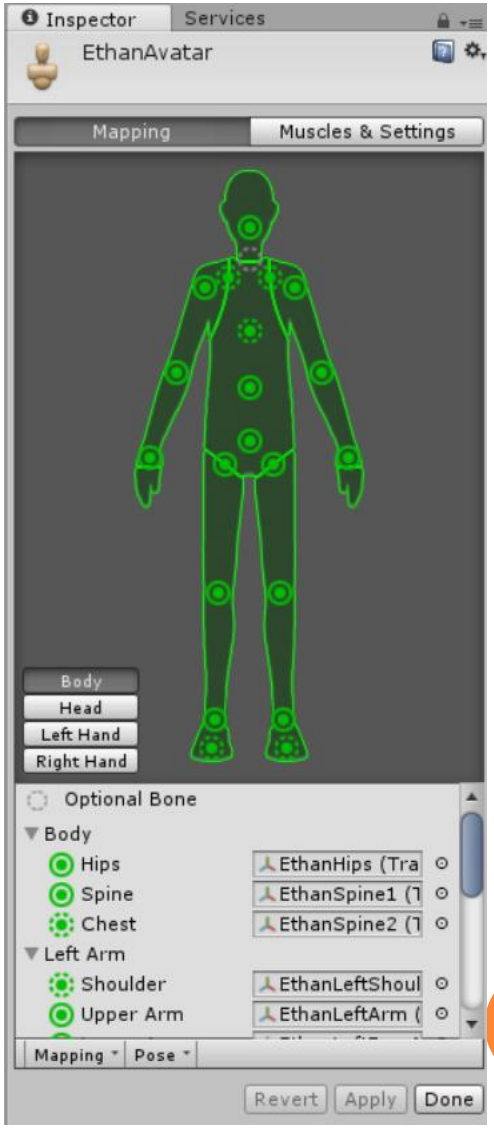
- 导入类人动画模型
  - 模型属性必须是：
  - 导入自带 charaters 模型
  - 观察第三方控制器的模型 Ethan
  - 骨骼匹配成功的模型..



动画剪辑（Animation Clip）
（4）类人动画 -- Avatar

- 配置Avatar
  - 点击模型属性 configuration
  - 或模型 Avatar 的属性

  - 配置场景中模型为 “T-pose” 状态
  - 完成骨骼映射后保存



http://www.ceeger.com/Manual/ConfiguringtheAvatar.html

动画剪辑（Animation Clip）
（4）类人动画 – 使用

- 实验
  - 了解第三人称控制器的模型与动画
  - 按文档提供的步骤，找几个类人动画完成动画复用

http://www.ceeger.com/Manual/Retargeting.html

动画中级技术
（1）直接播放

- Animator 的两个直播方法：
  - Play(…) 直接转到一个状态
  - CrossFade(…) 带过度转到一个状态

- 为什么它是中级技术？

动画中级技术
（2）动画覆盖状态机控制器

- 问题场景
  - 游戏有N个角色，这N个角色的状态机一样，仅动画内容不一样。
- 解决方案
  - 为一个角色设计状态机，其他的角色就设计 AnimatorOverrideController 的资源，在用工厂生产这个角色时，使用这个资源赋予 该角色 Animator.runtimeAnimatorController 属性
- 实际代码

```
5  public class MyFactory : MonoBehaviour {
6
7      public AnimatorOverrideController overAnim;
8      public Transform prefabs;
9
10     public GameObject GetYourObject(){
11         //...
12         return null;
13     }
```

动画中级技术
（2）动画覆盖控制器

- 添加一个Animator Override Controller 的资源

- 代码控制动态加载动画

  http://www.cnblogs.com/hnlyfy/p/5846689.html

- 动态生成 AnimationClip

  http://www.ceeger.com/Script/AnimationClip/AnimationClip.html

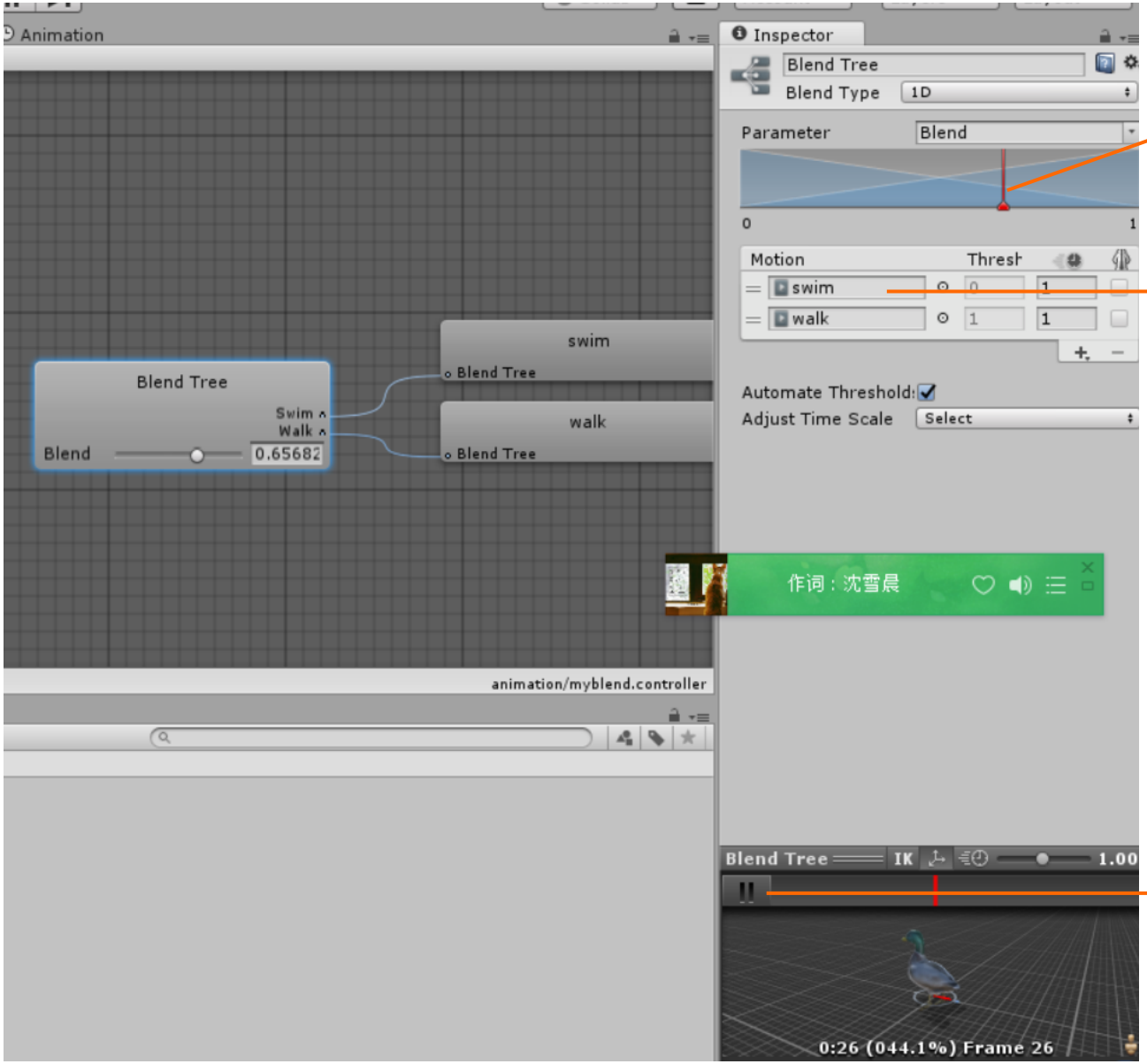
- 添加 AnimationCurve

  http://www.ceeger.com/Script/AnimationClip/AnimationClip.SetCurve.html

- 添加 AnimationEvent

动画中级技术
（3）动画混合树（Blend Trees）

- 游戏场景：
  - 一只鸭子走在沼泽地上，遇到水塘就游，遇到陆地就走。输入参数是地面与水面的距离 h，当 h > 0 走，当 h < 2 游，在之间则处于游走状态。
- 动画混合：
  - 创建一个鸭子对象 duck
  - 在状态机编辑窗口，点右键出现 Create State -> Form new Blend Tree
  - 修改状态名 swim_walk。 双击 swim_walk 状态，进入 Blend Tree 编辑界面。
    http://www.ceeger.com/Manual/1DBlending.html
  - 添加两个动作，拖入 swim 和 walk 。 系统会生成一个混合控制变量，移动红线，并看播放效果



参数值位置

编辑动作

动画播放预览

动画中级技术
（3）动画混合树（Blend Trees）

- 2D融合
  - 阅读文档 http://www.ceeger.com/Manual/2DBlending.html
  - 研究 Ethan 的状态机

动画中级技术
（4）其他

- 换武器与装备
  - 动画中有许多子物体，选择 enable，disable 就可以了
- 子状态机
- 动画分层
- 根动作

http://www.ceeger.com/Manual/MecanimAnimationSystem.html

面向对象设计思考（1） 设计模式

- 什么是设计模式
  - 设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。
  - 设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。
  - 设计模式使代码编工程化，是软件工程的基石，如同大厦的内部部件结构之间固定的搭配一般。
- 设计模式的应用领域
  - 面面对象程序设计：GOF 23 种对象之间的基础结构。如：工厂、单实例、适配器、组合、订阅发布、装饰 … …
  - 面向企业应用架构：如MVC、生产消费模型、C/S
  - 面向界面设计：

http://baike.baidu.com/item/%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F/1212549

面向对象设计思考（1） 设计模式

- 设计模式的表示
  - 模式名称：一个或几个易于记忆的日常名词或现象
  - 问题与需求：应用中典型的问题或故事。例如：对象的生产，需要：
    - 包装复杂的初始化业务规则
    - 对象的回收、缓存与复用
  - 解决方案：对问题进行抽象，给出软件的部件极其关系
    - 适配器部件：用户接口，适配器、适配对象
    - UML模型：给出部件之间关系与约束
    - 典型代码：使用单实例、工厂生产适配器
  - 效果与分析
    - 应用效果，如：订阅发布的结果解耦
    - 适用范围与注意事项，如：适配器合适做插件

面向对象设计思考
（1） 设计模式

- GOF 23 种设计模式
  - 创建型模式：单例模式、抽象工厂模式、建造者模式、工厂模式、原型模式。
  - 结构型模式：适配器模式、桥接模式、装饰模式、组合模式、外观模式、享元模式、代理模式。
  - 行为型模式：模版方法模式、命令模式、迭代器模式、观察者模式、中介者模式、备忘录模式、解释器模式（Interpreter模式）、状态模式、策略模式、职责链模式(责任链模式)、访问者模式。
- 设计模式应用
  - 在具体问题中，模式是以综合的方式出现的
  - 例如：游戏中 IUserAction 既是用户命令的接口（命令模式）也是SceneController对用户的外观（外观模式）。它使得用户界面类不关注命令的具体实现

面向对象设计思考
（2）反射技术

- 面向对象语言面试的必考三大技术
  - 垃圾回收机制与优化
  - 反射技术
  - 进程、线程与协程
- 什么是反射技术
  - 反射是面向对象语言中的重要机制。通过反射，
    - 可以在运行时获得程序集（dll,exe）或对象的类型（包括类、结构、委托、接口和枚举等）的成员和成员的信息。
    - 通过反射支持工具，动态调用函数与接口
- 为什么要学习反射技术
  - 动态决定是否调用对象的方法
  - 动态创建对象，调用对象
  - 实现插件自动加载与调用

面向对象设计思考
（2）反射技术

- C# 反射机制 API 的常用命名空间
  - System.Reflection
  - System.Type
  - System.Reflection.Assembly
  - System.Reflection.Emit（Unity5.0 IOS支持不详 ）

http://baike.baidu.com/item/%E5%8F%8D%E5%B0%84%E6%8A%80%E6%9C%AF

面向对象设计思考
（3）C# 基于代理的反射技术

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 using System;
6 //using System.Reflection;
7 //using System.Security.Permissions;
8
9 public delegate void D_Up(int u); //代理方法
10
11 public class mydelegate : MonoBehaviour {
12
13     // Use this for initialization
14     void Start () {
15         up (1);
16         try { //suppose that you do not know this object has method up?
17             D_Up test = (D_Up) Delegate.CreateDelegate(typeof(D_Up),this,"up");
18             test (2);
19         } catch (Exception e) {
20             Debug.Log("Why?"+ e.Message);
21         }
22     }
23
24     void up(int u){
25         this.transform.position += new Vector3(0,u,0);
26     }
27 }
```

1. 将代码挂载cube上
2. 运行！
3. 修改方法名？
4. …

https://msdn.microsoft.com/en-us/library/s3860fy3(v=vs.110).aspx

面向对象设计思考
（3）C# 基于代理的反射技术

- 你知道 Unity 实现私有函数调用的原理了吗？
  - 大致过程：
  - GameObject 创建一个部件
  - 获取该部件的特定方法的代理，例如 update
  - 如果成功，则将 update 的代理变量放置到一个 Event 类型的主题
  - 在游戏循环中触发该主题事件
  - 这些部件的 update 方法被逐一调用
- C# 反射与委托
  - 更多参考（晦涩难懂）：

http://www.cnblogs.com/cyjb/archive/2013/03/21/DelegateBuilder.html

http://tec.5lulu.com/detail/111ppn2e7zn788323.html

面向对象设计思考
（3）思考与练习

- 增强作业：实现Unity的以下方法的功能【options】
  - 用反射技术编写类似的函数
    1. GameObject.SendMessage
    2. GameObject.SendMessageUpwards
    3. GameObject.BroadcastMessage
  - 分析这些方法的计算成本
  - OnCollisionEntry() 的调用是这样实现的吗？给出理由
  - 使用 c# 反射技术列出一个对象的所有方法名称

面向对象设计思考
（3）在设计模式中使用反射技术

- 反射与委托技术大大简化了设计模式的实现
  - 游戏离散引擎为了与游戏部件对象交互，如果每个部件或行为元素都实现一个接口，这会导致每个引擎事件都需要遍历所有部件。
  - 通过反射，引擎只需要选择那些有需要的部件
  - 以现实中主播与粉丝的关系将，粉丝关注主播的主题仅满足了部分需求。如果主播发现粉丝有送花等能力，通过感知（发射），直接建立私下联系！
  - 反射技术是实现插件的重要技术内容！！！

课程小结

- 动画剪辑（Animation Clip）
  - 动画原理
  - 动画设计与编辑
  - 动画剪辑事件
  - 类人动画的使用
- 动画中级技术
  - 直接播放
  - 动画覆盖
  - 1D，2D 动画混合
- 面向对象的编程思考
  - GOF 设计模式概述
  - 基于反射的消息机制的实现

作业（Lab 8）

- 无

- 技术研究
  - 在 Github 搜索 unity animator
  - 例如：这个一个有趣的代码 https://github.com/keijiro/KvantWall
  - 搜索并研究一些你喜欢的 unity 动画技术或有趣的动画案例，写成博客
  - 如有可能，解释一些你认为有价值的设计模式