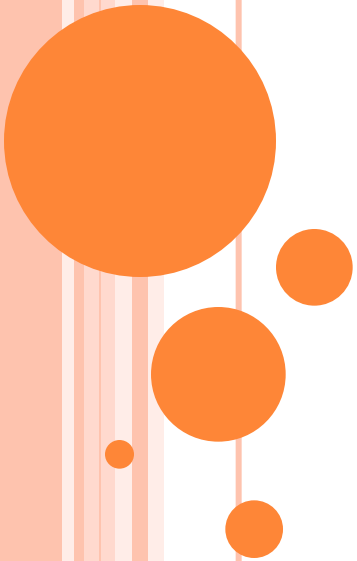
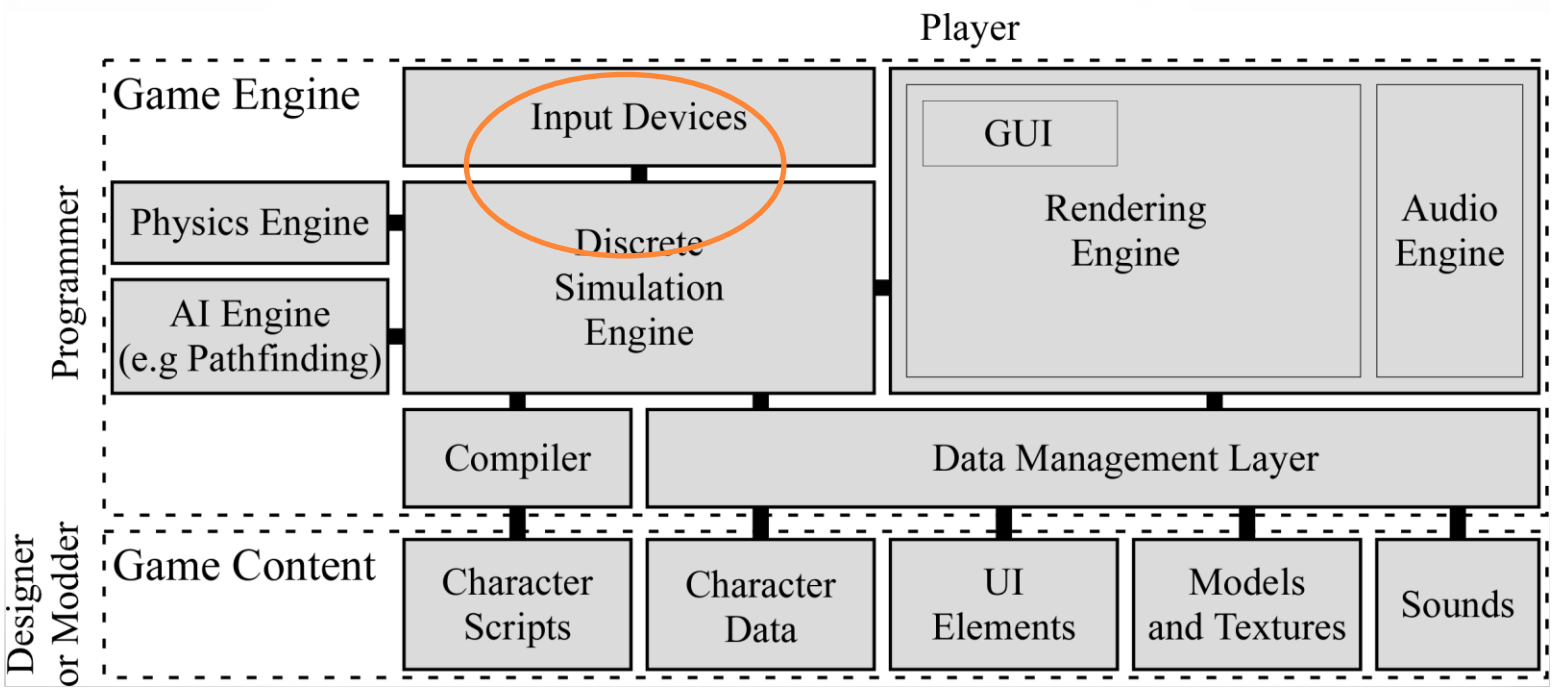


Introduction to Computer 3D Game Development

Interaction with Game World

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)
Programmer
Compiler
Data Management Layer
Designer
or Modder
Game Content
Character
Scripts
Character
Data
UI
Elements
Models
and Textures
Sounds

目录

游戏与创新
（1）了解游戏创新的层次

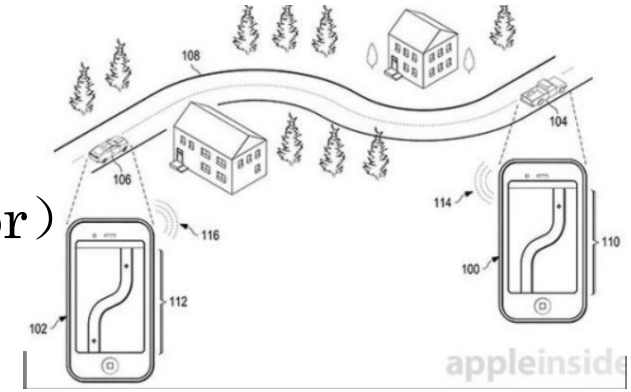
- 输入设备创新
  - 玩家：Apple、Microsoft、Nintendo、Sony
  - 内容：GPS、陀螺仪、…、Kinect、Hololen、AR、VR
- 核心玩法创新（各类竞赛最爱）
  - 新设备在新领域（医疗、公益、电商、社交）的应用
  - 游戏与智能（语音交互、智能学习、各种识别… …）
  - 新颖玩法（Flappy Bird、Temple Run … …）
  - 题材创新（如挑战极限系列，史上最难….）
- 以客户为中心的创新（商业公司最爱）
  - 细腻逼真的 3D 素材（比技术门槛和经费，如暴雪产品）
  - 热门故事（通常与历史、热门电影、政治事件绑定）
  - 满足各种脑残粉（如：开心消消乐、国内所有页游）

游戏与创新
（2）常考虑的输入设备

- 经典“三宝”
  - 键盘（KeyBoard）
  - 鼠标（Mouse）
  - 游戏操纵杆（Joystick）



- 手机游戏输入
  - 触摸屏（TouchPad）
  - 重力/位置传感器（Gravity/Geo Sensor）
  - 麦克风（Audio）
  - 摄像头（AR）
  - 手势/体态（Gesture/Posture）
  - 蓝牙, NFC … … 包括可连接手机的各种智能设备



- 其他设备
  - 条码、体感、激光、Kinect、Hololen、VR … …

游戏与创新
（3）输入设备的重要性与选择

- 体验、体验、新体验！！！
  - 游戏题材：
    - 体育、教育、赛车、社交等题材最敏感
  - 交互创新：
    - 触摸交互，重力交互，语音交互，体感交互
  - AR/VR
    - 沉浸式体验技术，最热

- 经济驱动的游戏
  - 选择手机及其智能设备创新游戏
- 挑战驱动的游戏
  - 选择只有一个：键盘

输入系统
（1）信息处理模型

- 输入处理方式（你喜欢哪个？）
  - 查询（polling），例如：检测键盘 A 的 up 与 down
  - 推送/事件（push / event），例如：OnKeyDown 句柄
- 涉及设计模式
  - GoF 命令模式（Command Pattern，Observer）
  - MVC 设计模式
- 输入组合与映射（以键盘为例）
  - 普通玩家在游戏中控制的按键不超过 6 个
  - 不同玩家生理、习惯等因素操控方式不一样

输入系统
（2）UNITY 原生支持能力

- 传统输入支持，Input 对象
  - 游戏三宝，手机触摸屏等
  - 查询方式
- 与输入设备交互
  - GUI（IMGUI），官方说开发用UI（查询方式）
  - New UI，NGUI 的官方“山寨”版（事件驱动）
- 第三方支持

输入系统
（3）JoyStick 游戏操纵杆

- JoyStick 是标准游戏机装备
  - Axes 轴，某个自由度的速度[-1..1]
  - Button 键，用于开火等
  - 震动反馈（option）
- 游戏有多个这样的设备（一般两个）

- PC 对游戏的支持
  - 模拟 JoyStick 装备
  - 通常把方向键、asdw 模拟成轴
  - Ctrl，command，Alt模拟成按键

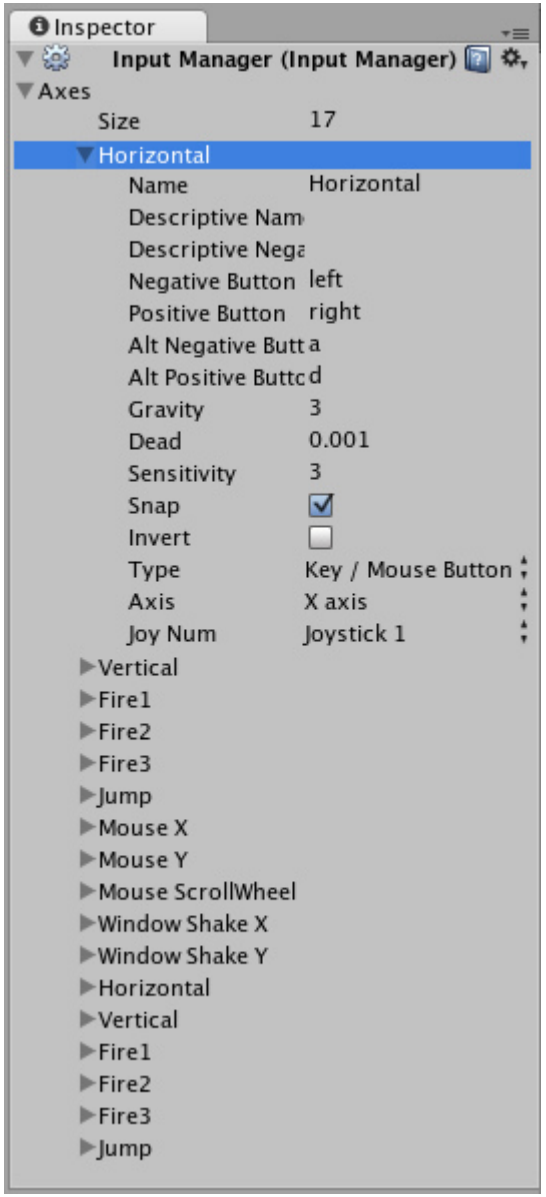
输入系统
（3）JoyStick – 虚拟轴与按键

- Unity用虚拟轴和按键
  - 通过名称访问轴和按键
  - 用户自定义按键

- 定义虚拟轴
  - 菜单 Edit → Project setting → Input
  - 用户与程序可以添加、修改按键映射

- 使用虚拟轴和按键

```
Input.GetAxis("Vertical") * speed;
If (Input.GetButton(“Fire1”)) { }
```



输入：http://www.ceeger.com/Manual/Input.html

课堂实验（一）
验证虚拟轴与按键

- 操作步骤：
  - 新建项目 Input，使用unity虚拟轴默认配置
  - 设置主摄像机为正交视图
  - 在（0，0，0）位置一个 cube，在其上挂代码

```
5 public class joystick : MonoBehaviour {
6
7     public float speedX = 10.0F;
8     public float speedY = 10.0F;
9
10    void Update () {
11        float translationY = Input.GetAxis("Vertical") * speedY;
12        float translationX = Input.GetAxis("Horizontal") * speedX;
13        translationY *= Time.deltaTime;
14        translationX *= Time.deltaTime;
15        transform.Translate(0, translationY, 0);
16        transform.Translate(translationX, 0, 0);
17        if (Input.GetButtonDown("Fire1")) {
18            Debug.Log ("Fired Pressed");
19        }
20    }
```

输入系统
（4）KEYBOARD – 键盘

- Input 静态变量（只读）
  - anyKey　　判断是否有键按着
  - anyKeyDown　　判断是否有键按下
  - inputString　　输入缓冲区
- Input 静态方法
  - bool GetKey(string name) 检测指定的按键按着
  - bool GetKeyDown(string name) 指定的键按下
- 枚举
  - KeyCode

输入系统
（4）KEYBOARD – 离散系统状态与事件

- 假设离散系统每秒采集 60 次状态
  - 玩家按了 “z” 键
  - 在 Update 中 Input.GetKey(“z”) 就会检出几十次
  - 在 Update 中 Input.GetKeyDown(“z”) 只会出现一次，但如果你这帧没有检查，则所有状态自动复位
- 对于事件，例如：KeyDown
  - 可能同时发生多个
  - 所以必须使用缓冲区缓存多个事件，如 inputString
- 编程要点
  - Input.GetKey 检测用户按键 “力度”，请使用虚拟轴
  - Input.GetKeyDown 检测用户按键次数

输入系统

（4）KEYBOARD – 字串的输入

```
public class ExampleClass : MonoBehaviour {
    public GUIText gt;
    void Start() {
        gt = GetComponent<GUIText>();
    }
    void Update() {
        foreach (char c in Input.inputString) {
            if (c == "\b"[0])
                if (gt.text.Length != 0)
                    gt.text = gt.text.Substring(0, gt.text.Length - 1);

            else
                if (c == "\n"[0] || c == "\r"[0])
                    print("User entered their name: " + gt.text);
                else
                    gt.text += c;
        }
    }
}
```

输入系统
（5）Mouse – 鼠标

- Input 静态变量（只读）
  - mousePosition　　鼠标位置（Vector3）
  - mousePresent　　是否有鼠标
- Input 静态方法
  - bool GetKey(string name) 检测指定的按键按着
  - bool GetKeyDown(string name) 指定的键按下
- 枚举
  - KeyCode.Mouse0 .. KeyCode.Mouse6

- 编程注意
  - 其实程序 onMouseMove，onMouseMoveOver 也难写
  - 每帧检测 MouseMove 可能导致计算过载，FPS！！

课堂实验（一）
实现新事件检测

- 要求：
  - 实现 DoubleClicked 检测

- 伪代码

```
Update() {
    IF “fire1” keydown THEN
      IF dc_count <= CONSTANT THEN
         DebugOut “Clicked!!!”
      ELSE
          Reset dc_count
      ENDIF
    ENDIF
    dc_count += delta-time
}
```

有bug吗？

输入系统
（6）手机传感器支持

- 多点触摸

multiTouchEnabled	Property indicating whether the system handles multiple touches.
simulateMouseWithTouches	Enables/Disables mouse simulation with touches. By default this option is enab
stylusTouchSupported	Returns true when Stylus Touch is supported by a device or platform.
touchCount	Number of touches. Guaranteed not to change throughout the frame. (Read Or
touches	Returns list of objects representing status of all touches during last frame. (Rea
touchPressureSupported	Bool value which let's users check if touch pressure is supported.
touchSupported	Returns whether the device on which application is currently running supports

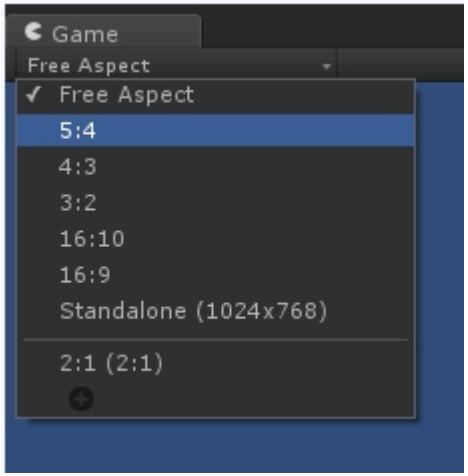
- 3D加速传感器
- 陀螺仪
- 国际化输入法（汉字）

输入系统
（6）其他值得探究的输入

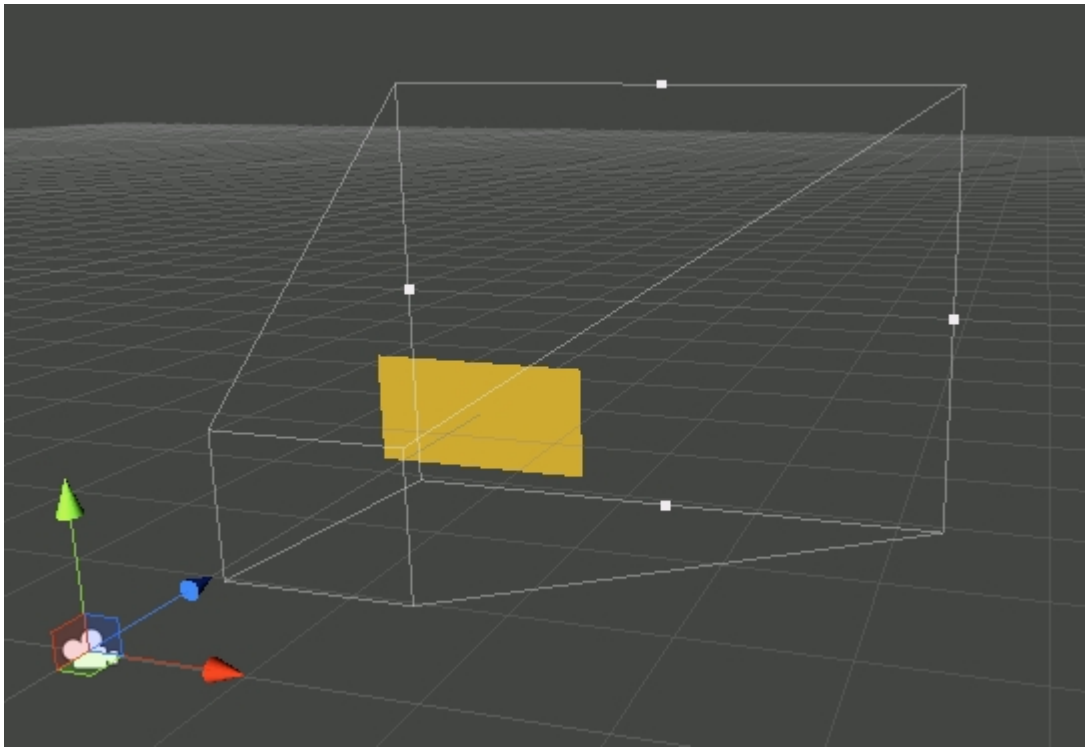
- 手势输入（Gesture/Posture）
  - 到商店搜索一下… …
- 语音输入
  - 也可以在网上搜索（unity 语音）
- 图像输入与AR
- GPS 位置输入
  - MR 越野游戏：定向追踪（定向寻宝）

人机交互
（1）Screen Space – 屏幕坐标系统

- 屏幕坐标（ Screen Space ）
  - 像素为单位，屏幕的左下角为（0，0）点，右上角为（Screen.width，Screen.height）
- 视口坐标（ViewPort）
  - 视口坐标是标准的和相对于相机的。相机的左下角为（0，0）点，右上角为（1，1）点
- Aspect（观察面）
  - 适应不同播放制式的观察面



人机交互
（2）Screen Space – World Space



从camera角度：

1. 物体透视按摄像机 z 轴深度投影到一个观察面。这个面是 z = z0 的一个正方型 (-1,-1)~(1,1)，称为标准面。正交投影就按 size 标准化；

2. 用 aspect 将正方形裁剪成一个矩形，这个矩形就是视口；

3. 将矩形映射的像素空间，就是 Screen Space

相机控制：
透视方式，就是视角参数， Field of View。
正交方式，就是 Size 参数。

人机交互
（3）鼠标与 3D 空间物体交互

- 射线碰撞与屏幕坐标
  - 屏幕每个点 p 都对应标准面上一个点 p’
  - 摄像机坐标系中 op’ 向量上点都投影到 p （透视图）
  - 因此，op’ 向量（射线）碰撞到的第一个三角面所属对象就是该坐标显示的物体

- Unity 的支持
  - Ray 射线结构，表示从某个点（original）发出的射线
  - RaycastHit 射线与物体碰撞的数据结构
  - Physics 物理世界管理对象，包括重力常数设置；射线、线段、盒子、球体碰撞的常用静态函数
  - Camera 部件，根据透视形式，返回正确的射线结构

课堂实验（二）
鼠标拾取物体（1）

```
18        if (Input.GetButtonDown("Fire1")) {
19            Debug.Log ("Fired Pressed");
20            Debug.Log (Input.mousePosition);
21
22            Vector3 mp = Input.mousePosition; //get Screen Position
23
24            //create ray, origin is camera, and direction to mousepoint
25            Camera ca = cam.GetComponent<Camera> ();
26            Ray ray = ca.ScreenPointToRay(Input.mousePosition);
27
28            //Return the ray's hit
29            RaycastHit hit;
30            if (Physics.Raycast(ray, out hit)) {
31                print (hit.transform.gameObject.name);
32                if (hit.collider.gameObject.tag.Contains("Finish")) { //plane tag
33                    Debug.Log ("hit " + hit.collider.gameObject.name +"!" );
34                }
35            }
```

1. 使用 **Camera.main** 获取主摄像机的 **Camera** 部件
2. mousePosition 是 Vector3，请不要修改 z 坐标
3. Camera 部件支持正确生成世界坐标的射线
4. Raycast 函数使用了变参（值参与变参），为什么 hit 必须用变参？
5. 为了优化性能，Raycast 支持在特定层扫描对象
6. 如何实现移到某个物体上就变色？有潜在的麻烦吗？

课堂实验（二）
鼠标拾取多个物体（2）

```
5  public class RayRender : MonoBehaviour {
6
7      public GameObject cam;
8
9      void Update() {
10         if (Input.GetButtonDown ("Fire1")) {
11
12             Vector3 mp = Input.mousePosition;
13             Camera ca = cam.GetComponent<Camera> ();
14             Ray ray = ca.ScreenPointToRay(Input.mousePosition);
15
16             RaycastHit[] hits;
17             hits = Physics.RaycastAll (ray);
18
19             for (int i = 0; i < hits.Length; i++) {
20                 RaycastHit hit = hits [i];
21                 Renderer rend = hit.transform.GetComponent<Renderer> ();
22
23                 if (rend) {
24                     // Change the material of all hit colliders
25                     // to use a transparent shader.
26                     rend.material.shader = Shader.Find ("Transparent/Diffuse");
27                     Color tempColor = rend.material.color;
28                     tempColor.a = 0.3F;
29                     rend.material.color = tempColor;
30                 }
31             }
```

面向对象设计思考
（1）游戏对象的创建

- 创建空对象并添加组件
  - new GameObject();
  - new GameObject(string name);
  - new GameObject(string name, params Type[] components);
- 创建基础类型游戏对象
  - GameObject CreatePrimitive(PrimitiveType type);
- 从已知对象或预制克隆（主要方法）
  - Instantiate<Transform> (brick, new Vector3(x, y, 0), Quaterni on.identity);

面向对象设计思考
（1）游戏对象的创建

```
5 public class CreateWall : MonoBehaviour {
6
7     public Transform brick;
8
9     void Start () {
10        BuildWall ();
11    }
12
13    public void BuildWall() {
14        GameObject wall = new GameObject ("A Wall");
15        for (int y = 0; y < 5; y++) {
16            for (int x = 0; x < 5; x++) {
17                Transform br = Instantiate<Transform> (brick, new Vector3(x, y, 0), Quaternion.identity);
18                br.name = "brick_" + x.ToString () + "_" + y.ToString ();
19                br.parent = wall.transform;
20            }
21        }
```

Transform brick 通常是一个预制，不是一个游戏对象！！！

面向对象设计思考
（2）游戏对象的销毁

- 对象销毁
  - Object.Destroy(Object obj, float t = 0.0F);
  - 如果是组件对象，则从游戏对象中立即摘除；
  - 如果是游戏对象，则不会在update期间立即销毁，通常在render 前销毁它的部件以及所有子对象；

- 立即销毁
  - Object. DestroyImmediate
  - Unity 建议不要立即销毁，这可能导致离散引擎并发的行为之间依赖关系产生不可预知错误；

面向对象设计思考
（2）自定义的游戏对象属性

- 创新游戏对象“子类”
  - Unity 不建议用户继承 GameObject
  - 通过改游戏对象添加行为，作为内部数据存储
- 给游戏对象附加数据属性的方法
  - 创建一个脚本 DiskData:MonoBehaiver
  - 添加公共属性与方法，删除 start 和 update 方法
  - 挂载到一个 disk 的游戏对象（圆柱）
  - 制作成预制
- 检测一个对象是否拥有数据属性
  - GetComponent<T>()

面向对象设计思考
（4）游戏对象工厂对象

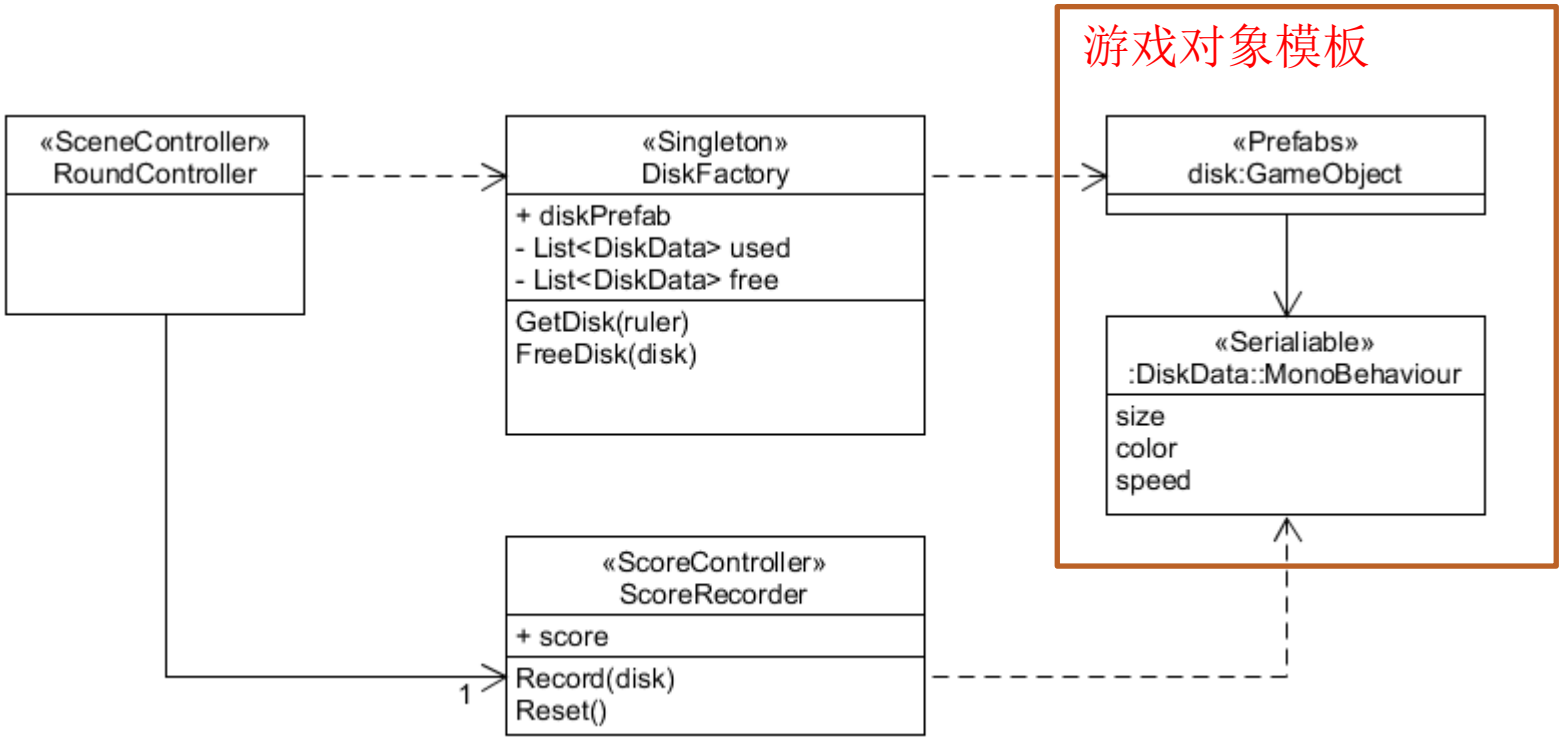
- 为什么需要工厂对象
  - 游戏对象的创建与销毁高成本，必须减少销毁次数。如：游戏中子弹
  - 屏蔽创建与销毁的业务逻辑，使程序易于扩展

- 案例研究：“鼠标打飞碟”游戏设计

  游戏需求：

  1. 分多个 round ， 每个 round 都是 n 个 trail；
  2. 每个 trail 的飞碟的色彩，大小；发射位置，速度，角度，每次发射飞碟数量不一；。
  3. 鼠标击中得分，得分按色彩、大小、速度不同计算，计分规则自由定

面向对象设计思考
（4）DiskFactory 的设计



设计读图：这是一个带游戏对象缓存的工厂类

1. 前面游戏，由导演、场记、运动管理师、演员构成。
2. 新游戏中，场记请了记分员、飞碟管理员
3. 其中记分员按飞碟的数据计分，记分员拥有计分规则
4. 场记只需要管理出飞碟规则与管理碰撞就可以了

面向对象设计思考
（5）UNITY 场景单实例的实现

```
5 public class Singleton<T> : MonoBehaviour where T : MonoBehaviour
6 {
7
8     protected static T instance;
9
10    public static T Instance {
11        get {
12            if (instance == null) {
13                instance = (T)FindObjectOfType (typeof(T));
14                if (instance == null) {
15                    Debug.LogError ("An instance of " + typeof(T) +
16                    " is needed in the scene, but there is none.");
17                }
18            }
19            return instance;
20        }
21    }
22 }
```

场景单实例要求场景中自少有一个 T 类型的 Mono 子类，FindObjectOfType会导致漫长的检索，除非单实例模式。使用非常简单（虚线表示引用）：

```
DiskFactory df = Singleton <DiskFactory>.Instance;
```

面向对象设计思考
（4）道具工厂和记分员设计意义

- 面向对象设计的核心
  - 抽象、包装、隐藏
- 道具工厂
  - 通过场景单实例，构建了方便可取获取DISK的类；
  - 包装了复杂的Disk生产与回收逻辑，易于使用；
  - 它包含Disk产生规则（控制每个round的难度），可以积极应对未来游戏规则的变化，减少维护成本
- 记分员
  - 包装了计分规则（控制业务均衡）
  - 提供了简单的对外业务接口

- 应对规则、地图等变化，是游戏设计的要点之一！

面向对象设计思考
（5）带缓存工厂模式的实现

- getDisk(ruler) 伪代码

```
IF (free list has disk) THEN
   a_disk = remove one from list
ELSE
   a_disk = clone from Prefabs
ENDIF
Set DiskData of a_disk with the ruler
Add a_disk to used list
Return a_disk
```

- FreeDisk(disk)

```
Find disk in used list
IF (not found) THEN THROW exception
Move disk from used to free list
```

自学内容:
C# 枚举类型

- 枚举支持哪些基础类型

课程小结

- 游戏输入
  - 游戏输入与应用创新的关系
  - 查询模型与事件模型
  - 虚拟轴与虚拟按键（虚拟的优点）
  - 键盘查询
  - 用鼠标拾取3D对象
- 面向对象设计技巧
  - 对象创建与回收
  - 扩展游戏对象
  - 工厂模式
  - 场景单实例模式

作业：

- 飞碟游戏
  - **游戏内容：**
  1. 游戏有 n 个 round，每个 round 都包括10 次 trial；
  2. 每个 trial 的飞碟的色彩，大小；发射位置，速度，角度，受该 round 的 ruler 控制；
  3. 每个 trial 的飞碟有随机性，总体难度随 round 上升；
  4. 鼠标点中得分，得分规则按色彩、大小、速度不同计算，规则可自由设定。
  - **游戏涉及的知识点：**
  1. 世界与屏幕坐标、射线与碰撞、动态修改shader
  2. 支持mono的单实例、游戏对象工厂、扩展游戏对象属性与行为的方法