



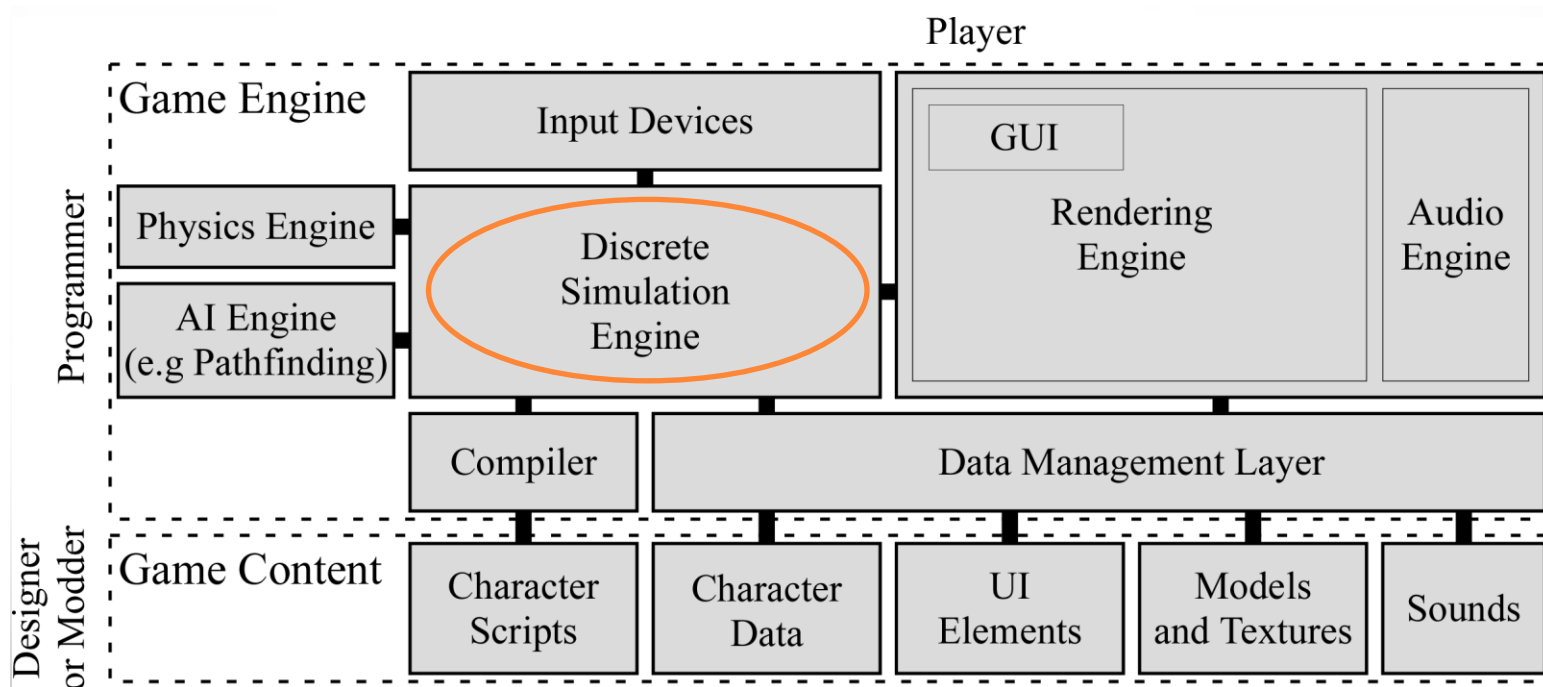
INTRODUCTION TO COMPUTER 3D GAME DEVELOPMENT

Discrete Simulation Engine Basic

潘茂林, panml@mail.sysu.edu.cn

中山大学·软件学院

游戏引擎架构



目录

- 3D 游戏引擎简介
- 离散仿真引擎与程序控制结构
- Unity 离散仿真系统实现
 - 游戏元素及其关系
 - 游戏对象对象及其组合（课堂实验1）
 - 游戏行为对象与绑定
 - 行为对象与游戏循环（课堂实验2）
- 创建、复制对象与预制
- Unity 离散系统 UML 类图

强烈推荐

游戏引擎架构/(美)格雷戈瑞(Gregory,J.)著；叶劲峰译.—北京：电子工业出版社，2014.2



3D 游戏引擎 (1)

- PC, 游戏机平台



画面精美令人难以置信。它们核心都是 c 语言实现



3D 游戏引擎 (2)

○ 一些引擎（传统游戏公司策略“卖完产品卖引擎”）

引擎	代表作	引擎	代表作
虚幻/Unreal	《战争机器》	Cry Engine*	《Crysis》
寒霜/Frostbite	《战地》	Infinity Ward	《使命召唤》
EGO	《尘埃2》	id TECH	《DOOM3》
Source	《半条命2》	X-Ray*	《潜行者》
Havok Vision	《哥特王朝》	Quake/idTECH	《雷神之锤》
Chrome4*	《狂野西部2》	MT framework	《生化危机5》
Gamebryo	《上古卷轴IV》	Jupiter EX	《F.E.A.R》

* 顶级特效引擎

* 都是 C, C++ 核心，都需要顶级 CPU GPU 支持

* 由于传统大型游戏引擎都是自己开发，所以没中国游戏公司什么事情，要收购，要么慢慢积累

3D 游戏引擎 (3)

○ 移动平台

- Unity 3D 借助 mono 平台首次登陆手机平台

Windows + c → mono → Linux

简单, 易用, 成熟, 部分开源

- Unreal
- Frostbite
- Cocos 3d
-

○ 网页平台

- three.js WebGL 官方效果展示项目
- babylon.js 目前发展较好的项目
-



开源的游戏引擎（4）

- 开源引擎
 - OGRE
 - Panda 3D
 - Yake
 -
- 半开源引擎
 - Unity 3d
 - Torque
 -



CORE OF ENGINE

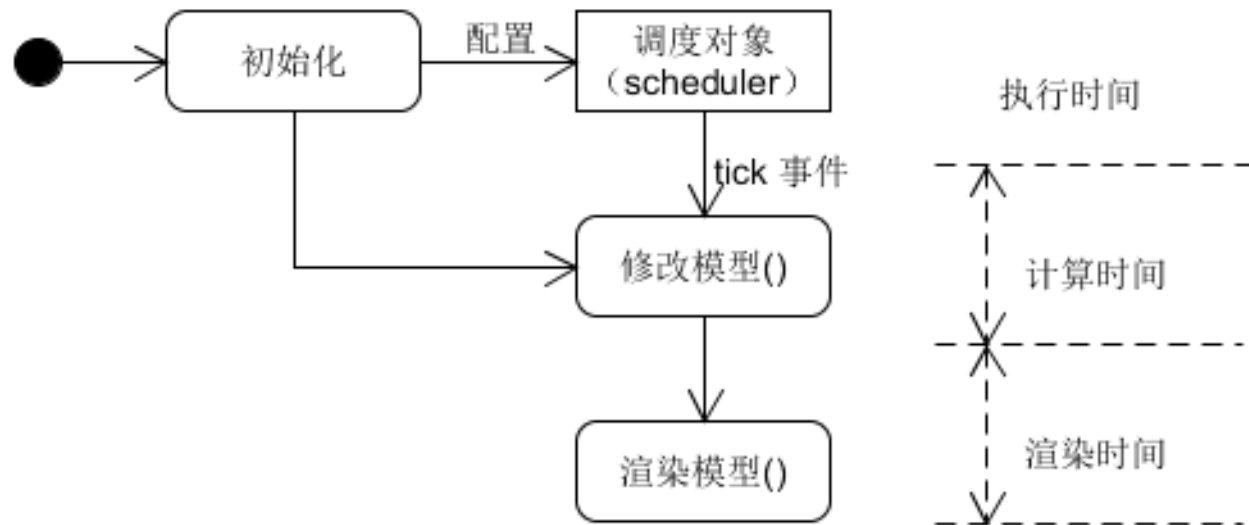
-- DISCRETE SIMULATION ENGINE

- **离散仿真**：时间被分成为若干小的时间片，系统状态被这段时间内发生的系列活动而改变。称为基于活动的仿真（activity-based simulation）
- **离散事件仿真**：一系列事件，在发生的时间改变系统的状态。称为基于事件的仿真（Events-based simulation）
- 对于游戏系统中物体，必须让人感觉到**持续**的变化，则必须使改变物体状态的频率足够快，这个频率就是**FPS（帧数/秒）**，视频要求不能低于 30 fps，通常在 60 fps 以上



CORE OF ENGINE

-- DISCRETE SIMULATION PROGRAMMING



看似简单的计算模型，离散计算模型导致系列问题：

-- 跳帧：执行时间大于规定调度时间

-- 穿越：高速小物体在相邻帧位置变化大，没有检测到路径上物体发生碰撞

--

CORE OF ENGINE

-- HOW ABOUT EVENT DRIVEN

- 游戏本身是一个连续运动的系统。传统的引擎，包括 XNA，都是离散仿真系统
- 目前，Cocos 2d 等面向对象的游戏系统，看似“完全”事件驱动的游戏系统，其底层核心逻辑都是离散仿真系统。核心代码都是时钟驱动的，如 Action 对象实现。

- 事件驱动的语言特征：

```
XXX Class {  
    onClick(){ ... ... } // events-handler  
    onMouseMoveOver(){ ... ... } // message dispatcher  
}
```

- 为了迎合面向对象的程序员入门，新一代引擎或多或少支持事件驱动。

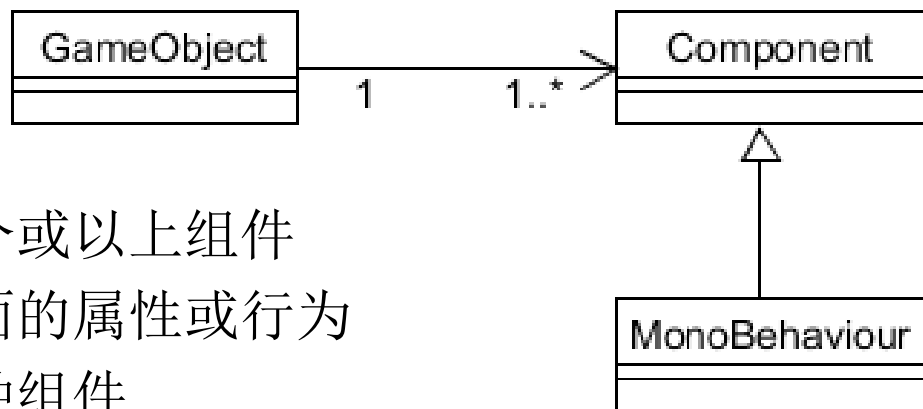


UNITY 离散仿真系统实现

游戏世界抽象元素！！！！

- 离散仿真引擎按定义包括三个对象：
 - 活动/行为 – MonoBehaviour 引擎控制行为对象的基类
 - 系统状态 – GameObject 引擎管理的对象事物基类
 - 调度器 – 定时驱动行为添加、修改、删除事物及其属性

- 行为与状态的关系：



- UML 对象图表示：
- 每个游戏对象包含一个或以上组件
- 这些组件表示某个方面的属性或行为
- 其中，行为对象是一种组件

UNITY 离散仿真系统实现

(1) 游戏世界之游戏物体

- 游戏对象包括（菜单 **GameObjects**）：
 - 空
 - 3D物体（立方体、球体、胶囊、圆柱体、平面和四边形 ...）
 - 2D物体（精灵/图片）
 - 摄像机
 - 灯光（平面，聚光，...）
 - 音频
 - UI 元素
 - 粒子系统
 -

UNITY 离散仿真系统实现

(2) 游戏物体的属性与行为

- 游戏物体的属性和行为在碰撞、渲染等应用中不一样，Unity 使用组件对象，按需组合使用。
- 常用组件（系统菜单 Component）
 - 变换（必须）：物体空间的位置、角度、Scale 属性
 - Mesh/网格：物体的形状与形态
 - 物理：物体碰撞时需要的属性
 - 音频组件
 - 渲染组件
 - 布局组件
 - 动画组件
 -
 - 脚本组件（MonoBehaviour）



UNITY 离散仿真系统实现

(3) 游戏物体之间的空间关系与组织

○ 世界坐标 (world space)

- 对于一个游戏场景(Scene), 它使用的坐标为世界坐标。

○ 相对坐标 (relative space)

- 一个游戏物体, 它使用其他物体作为参考坐标原点, 得到的坐标空间, 称为相对坐标

○ 游戏对象空间表示与组合

- 每个游戏对象必须且仅能包含一个表示空间的变换组件 (transform)
- 根据游戏对象之间的空间依赖关系, 我们把游戏对象按树形结构组合起来。其中, 子对象使用父对象空间作为参考坐标系



UNITY 离散仿真系统实现

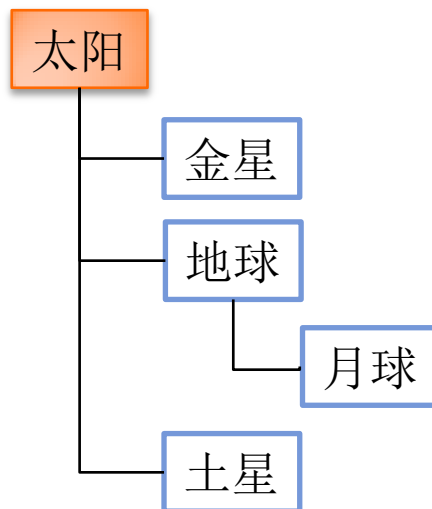
(4) 游戏物体的空间组织与应用

○ 玩家扮演角色与摄像机

- 第一人称游戏：摄像机作为角色子对象，且放置在眼部
- 第三人称游戏：摄像机作为角色子对象，自动跟随
- 普通游戏：摄像机与角色没有空间依赖关系

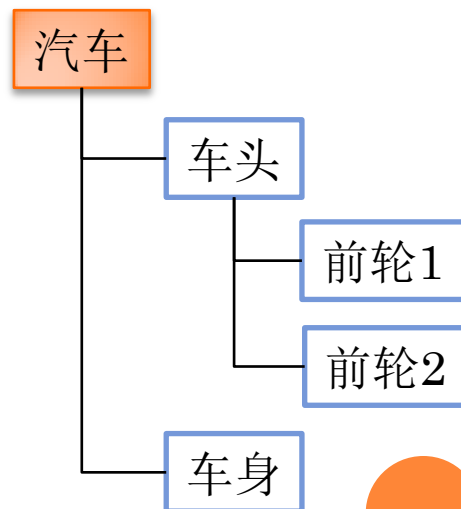
○ 太阳系的表示：

- 太阳在银河（场景空间），
- 行星在太阳系坐标空间
- 卫星在行星坐标空间



○ 汽车的表示：

- 由于汽车是组合体，用空物体表示



课堂实验（一）

（1）创建 HELLO 项目

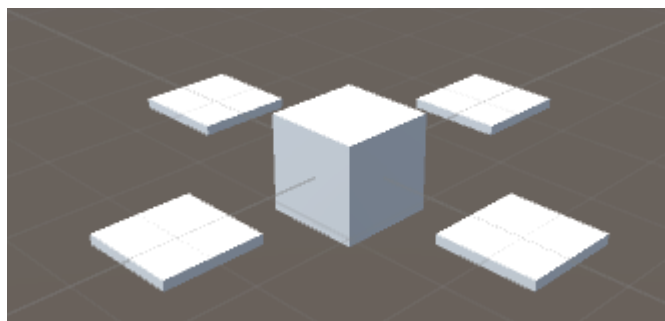
- 创建一个新项目 Hello，识别操作区
 - 编辑器菜单
 - Hierarchy：运行时游戏对象森林
 - Project：项目资源（游戏数据与代码）
 - Inspector：游戏对象绑定组件，以及组件属性与行为
 - Scene：设计用场景
 - Game：游戏输出
 - Console：控制台输出
- Hello项目由哪些对象？
- 观察对象的 Transform 组件的属性



课堂实验（一）

（2）游戏物体与组织

- 用五个立方体组成如图桌椅套件。
 - 使用Transform组件，修改Position，Scale属性
 - 请不要使用鼠标修改物体位置，或 观察视角



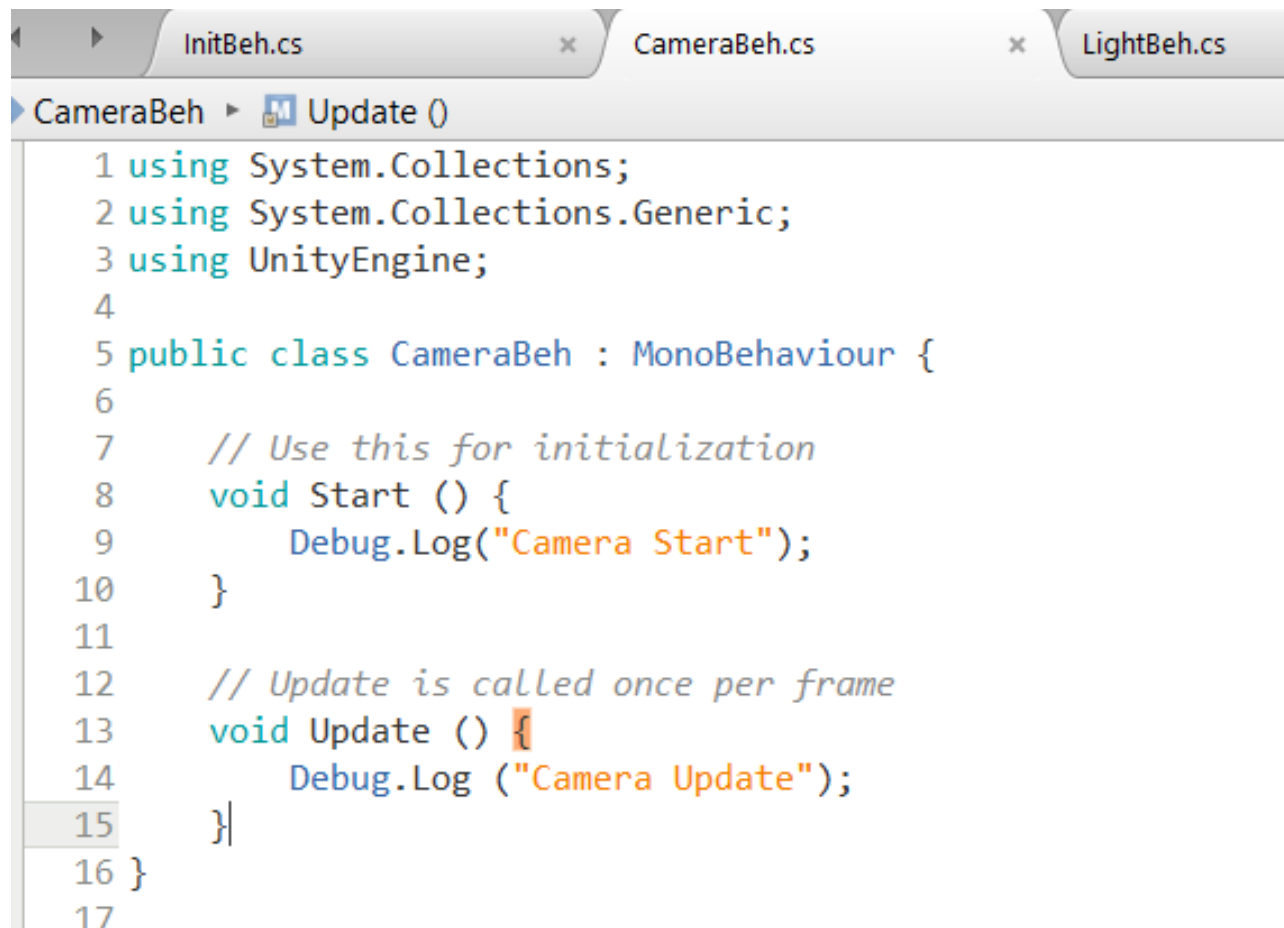
- 最后，修改 table 位置，观察效果
 - 运行该程序！
- 用菜单 File → Save Scence 起名 table



UNITY 离散仿真系统实现

(1) 游戏世界之行为对象

- 为游戏对象添加一个（New Script）组件，例如：



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class CameraBeh : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9         Debug.Log("Camera Start");
10    }
11
12    // Update is called once per frame
13    void Update () {
14        Debug.Log ("Camera Update");
15    }
16 }
17
```

UNITY 离散仿真系统实现

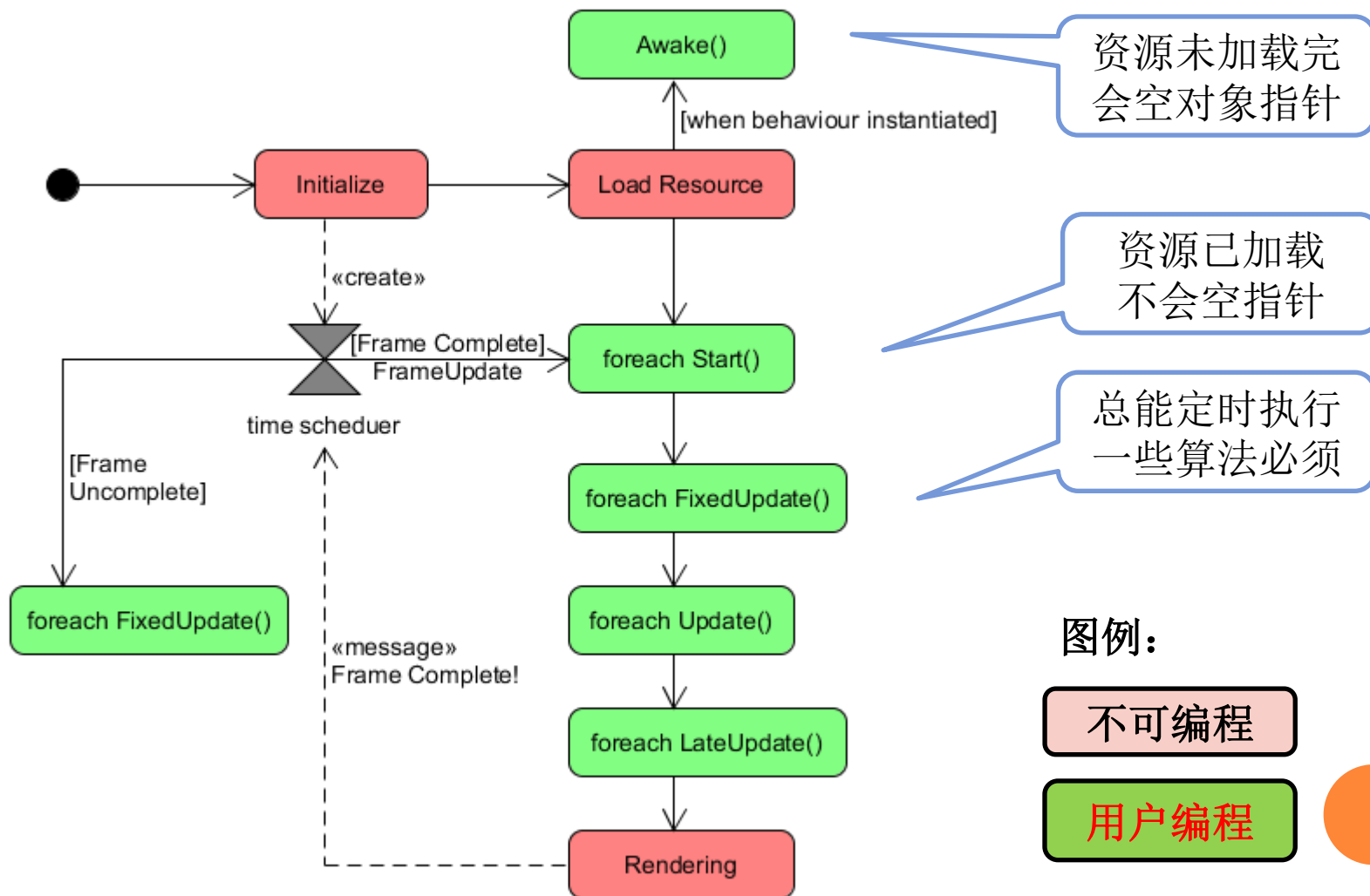
(2) 行为对象与调度对象相关的方法

- MonoBehaviour 对象
 - 所有行为对象的基类
 - 没有 public 构造函数，只能游戏对象创建
- 调度相关方法

方法	说明
Awake	当一个脚本实例被载入时Awake被调用。
Start	Start在所有Update函数之前被调用一次。
Update	当行为启用时，其Update在每一帧被调用。
FixedUpdate	当行为启用时，其 FixedUpdate 在每一时间片被调用。
LateUpdate	当行为启用时，在所有Update函数之后执行。例如：摄像机跟随。

UNITY 离散仿真系统实现

(3) 调度器与行为对象执行顺序



UNITY 离散仿真系统实现

(4) 项目执行过程与编程要点

○ Initialize

- 菜单 Edit → Project setting → time, script order ...

○ Main

- 使用行为的对象的 awake()
- 自己保证首轮加载的行为对象，仅有一个有 awake()

○ Awake

- 不要访问除了自己以外的对象，不可猜测系统加载顺序

○ Start

- 只会执行一次

○ FixedUpdate

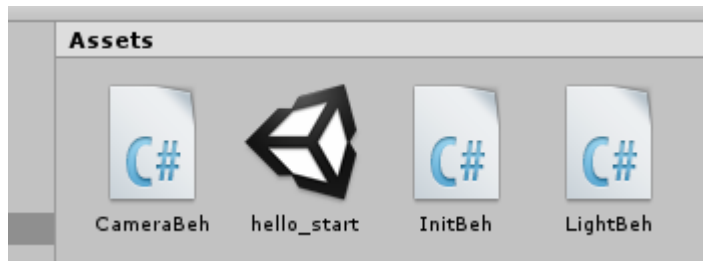
- 不能认为它在 Update 之前执行
- 如果必须在 Update 之前做准备，LastUpdate很有用



课堂实验（二）

（1）挂载行为对象

- 添加一个新的 Empty 对象 init
- 将项目组织成如图结构
- 创建并挂载行为对象
 1. 选择 Main Camera, 在 Inspect 添加前述代码
 2. 添加 Debug.log 输出当前行为执行函数
 3. 分别选择 Light 和 init 添加行为对象，如图



4. 执行程序，在 console 观察执行顺序



课堂实验（二）

（2）行为对象添加属性与方法

- 修改 InitBeh 代码如图：
 - 保存代码
 - 观察该对象组件是否多了属性
 - 将 table 对象拖入该属性
 - 在哪个方法中设置 table 起始位置？
 - 为什么添加 MoveTable 方法？
 - 为什么不在 Update 中完成？
- 总结了一下，离散仿真引擎的工作程序

```
5 public class InitBeh : MonoBehaviour {
6
7     public Transform table;
8
9     // First run in project
10    void Awake () {
11        Debug.Log("Init Awake");
12    }
13
14    // Use this for initialization
15    void Start () {
16        Debug.Log("Init Start");
17    }
18
19    // Update is called once per frame
20    void Update () {
21        Debug.Log ("Init Update");
22        MoveTable (table);
23    }
24
25    void MoveTable(Transform moveObj) {
26        ;
27    }
28 }
```

创建与复制对象

(1) 创建 PRIMITIVE 对象

○ 添加一个新行为对象

```
5 public class InitBeh_another : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9         print ("hello!");
10        // create a game object
11        GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
12        cube.name = "a cube";
13        // setting position
14        cube.transform.position=new Vector3(0,Random.Range(0,5),0);
15        // play as this gameobject sub-object
16        cube.transform.parent=this.transform;
17    }
18
```

○ 运行，并在对象层次树找到该对象



创建与复制对象

(2) 克隆游戏对象

- 游戏中大量对象都是一样的，如组成墙的砖头，大量的机器人等，克隆就是最方便的！

```
// Use this for initialization
void Start () {
    Debug.Log("Init Start");
    //copy table, use static method (why? not OO)
    GameObject anotherTable = (GameObject)Instantiate(table.gameObject);
    anotherTable.transform.position=new Vector3(0,Random.Range(5,7),0);
    anotherTable.transform.parent=this.transform;
}
```

- 运行代码，发现桌子和4个椅子都克隆了！
- 请问：
 - Deep clone
 - Shadow clone



创建与复制对象

(3) 使用预制技术

- 预制，预制好的部件。操作：
 - 将 table 对象拖放到项目视图，即生成一个预制
 - 菜单 Assets → create → Prefabs，将对象拖入预制。
 - 预制完成后，就可以删除游戏对象
- 使用预制，假设有预制 table
 - 创建一个行为对象，Tranform 或 GameObject 类型的 table 作为 public 变量。直接从预制拖入该变量使用。
 - 利用 Resources.Load(“table”) 加载。
 - 编辑器中将预制拖入对象层次标签
- 思考题：
 - 假设 table 预制中有行为部件TableBeh，预制在start()中实例化，请问 table 中行为 update 在此帧执行，还是在下一帧。在awake（）中有区别吗？



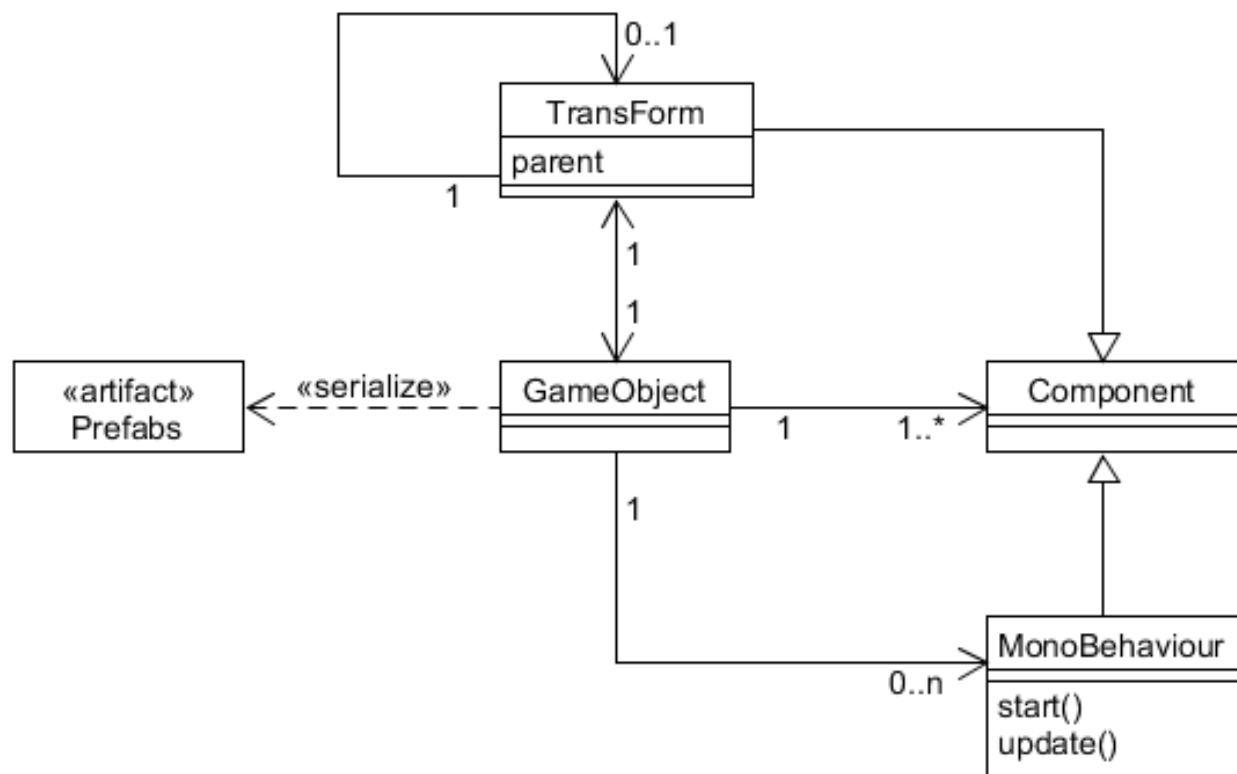
课堂练习（三）

预制与对象实例化

- 创建对象砖块（**Brick**）作为预制
- 创建一个行为（**BuildWall**），它使用砖块切一面 5*10 的墙



UNITY 离散系统 UML 类图



组合模式的使用，使得 Unity 的离散引擎灵活、易于扩展；
Component 强组合于 GameObject，使得内存空间管理富有效率，提高了性能

自学内容： GUI

- 为了方便编程，Unity 也提供了原始的GUI辅助完成许多任务。
- 自学 4399 提供的操作入门 PPT



课程小结

- 离散仿真系统
- Unity 的实现
 - 对象、行为、调度器
 - 对象及其组织
 - 行为与对象的关系
 - 系统的执行过程
- 对象的创建与预制
 - 创建基本对象
 - 预制
- 简单 UI

