暑培-Unity3D

SAST

2023 年 8 月 13 日

目录

当前进度

Unity 执行顺序

一个完整的游戏，具体经历的过程。
Awake -> OnEnable -> Start +-> OnXXX -> Update +-> OnDisable -> OnDestroy

- Awake: GameObject 被初始化而且启用后立即呼叫，用于一些非常重要的初始化（血量等）。
  (大多数时候用不着，用 Start 足矣。)
  OnDestroy: 物体被销毁时候的行为（扣血到零，加对方金币。。）
- OnEnable/OnDisable: 挂接在 GameObject 上的脚本部件被启用、停用的行为。
- **Start** "Start is called before first frame update."
  请注意：Unity 会执行完所有脚本的 “ 之后再进入更新周期.

Unity 执行顺序

一个完整的游戏，具体经历的过程。
Awake -> OnEnable -> Start +-> OnXXX -> Update +->
OnDisable -> OnDestroy

- Update: 以一定的时间跨度（帧率），反复调用的功能，用以更新游戏元素的物理状态（位移、金币、布局、键盘事件）
- OnXXX：事件触发机制。例如 OnCollisionEnter, OnMouseDown. 可以这样理解：Unity 在反复调用 Update 之间，插入对事件监听的处理。
- Unity 执行顺序

当前进度

多线程

作用体现：多人游戏，监听网络接口。
在游戏内，一边跑 UI，一边做内部运算。
“指导程序一心多用”
警告：场景相关运算，只能在主线程上进行，否则报错。

```
var thread = new Thread(new ThreadStart(Read));
    // 启动线程
    thread.Start();
    // 等待线程结束
    thread.Join();
    void Read() {
    // Do something
    }
```

当前进度

JSON 数据格式

先看例子：

```
[
    {
        "name": "cxk",
        "age" : 25
    },
    {
        "name": "Bob",
        "age" : 30
    }
]
```

- JSON 有层次化结构，用 [] 表示数组，用 表示物件。
- 词典——键值对结构，值可以是数组或其他物体，实现嵌套。
- JSON 可以携带的基本类型：字符串、数字、布尔值。

Newtonsoft.JSON 解析器

JSON 数据的合成、解析。

```
T res = JsonConvert.Deserialize<T>(string);
string json = JsonConvert.Serialize<T>(object);
```

如果要控制上文的数据，可以在 c# 定义这样的类。

```
public class NameCard {
    public string name {get;set;}
    public int age {get;set; }
}
```

当前进度

Unity 文件架构

所有与 Unity 控制逻辑直接关联的部分，都应该放在 Assets 文件夹内。
以下是一个典型的 Unity 项目的目录组织方式

- Materials
- Prefabs
- Scenes
- Scripts
- Resources
- Settings

等

Unity 项目架构

一般而言，玩家下达指令，对游戏提供的资源，加以运用，然后游戏后端按玩家操作和其他环境状态更新当前的游戏状态。

- 游戏主控 (GameController)
- 等级管理（关卡切换转移）
- 资源管理（金币、血量、可用武器）
- 游戏过程监控（记录过程，读取回放）
- 渲染器（把游戏角色生成做一层抽象）
- 用户输入监控（按键组合、JS 前端接口）
- 网络客户端（可选、多人）远端服务器做游戏过程管理

当前进度

Unity 开发实战

先构想游戏故事，设计 GameController，然后思考如何组织外围模块协助主模块。
假设我们要设计一个两人井字游戏。

- 游戏主控：控制其他模块，总指挥。
- 游戏过程监控：双方每一步下子方位。
- 渲染器：在某个方位绘画圆圈、交叉。
- 用户输入监控：按键确定方位，或者如果用 WebGL 技术，就要向 WebGL 的 JS 模块提供接口。
- 网络客户端：和后端交互，交换下子方位信息。

Unity 开发实战

游戏过程监控：
维护一个数组，记录双方下子方位、时间等信息；
用一个规范化的信息表示这些数据，然后提供加载功能，实现回放。
一般而言，提供下列接口：

- 保存当前状态。
- 加载某次对局。
- 跳转到某一次交互。

渲染器

把场景、物件、角色、道具等渲染。
就这个案例而言，如果下子点是抽象坐标，就做一个抽象到物理坐标的转换 (例如 Tilemap)，然后在某处 Instantiate
对于一些比较复杂的游戏而言，角色生成、移动的时候可能有一些特效，有一个调用链，可以将之组合拼装。
例如，放置法师需要有闪烁效果，那就在实例化的同时，播放动画或启动粒子系统。

用户输入监控

如果提供键盘快捷操作方式，可以按用户的按键组合，决定输入类型。
假设，我们用 A,B,C 表示横坐标，1，2，3 表示纵坐标，那么两边组合起来，就能对应一个抽象坐标，交给游戏主控来指挥渲染器生成棋子。
又假设该程序最终在 WebGL 平台上展示，如果对用户暴露的回放功能接口在 JS 部分实现，就要提供接口供 JS 脚本调用，通知过程监控取出某个游戏状态给渲染器。

网络客户端

对于多人游戏，当前进程需要和另外的进程交互，那就涉及到 IPC (Inter-process communication) 的概念。
一方面，我们可以直接用 Socket(套接字)，和游戏后端逻辑交互；另一方面，也可以用对前者加以封装的 WebSocket 和后端通信。
过程如下：

1. 初始化、建立连接。
2. 监听接收窗口。
3. 按需调用 Write 和对方通信。

```
var ws = new ClientWebSocket();
await ws.ConnectAsync(new Uri("ws://" + uri),
Cancellation.None);
```

网络客户端

对于多人游戏，当前进程需要和另外的进程交互，那就涉及到 IPC (Inter-process communication) 的概念。
一方面，我们可以直接用 Socket(套接字)，和游戏后端逻辑交互；
另一方面，也可以用对前者加以封装的 WebSocket 和后端通信。

```
protected override async void Read() {
var buffer = new ArraySegment<byte>(new byte[8192]);
while (ws.State == WebSocketState.Open){
WebSocketReceiveResult result;
using (var ms = new MemoryStream())  {
do  {
    result = await ws.ReceiveAsync(buffer, ...);
    ms.Write(buffer.Array, buffer.Offset, result.Count);
}  while (!result.EndOfMessage); // ...}}}
```

目录

结束