进程和线程

进程和线程是操作系统的基本概念

1 计算机的核心是CPU 承担了计算机的计算任务

这里把计算机看成是一座时刻在运转的工厂

2 假定工厂电力有限 一次只能提供给一个车间使用

当一个车间工作的时候 其他车间都必须停工

意思是 单个CPU一次只能运行一个任务

3 进程就好比工厂的车间 代表CPU能处理的单个任务

任意时刻 CPU总是运行一个进程 其他进程处于非运行状态

4 一个车间里 有需要的工人在工作 他们协同完成同一个任务

线程就好比车间里的工人 人数可多可少

意思是 一个进程可以包含多个线程

5 车间是拥有多空间的集合 而空间是共享的

比如许多房间是每个工人都可以自由进出的(例如厨房 厕所等)

这象征着 一个进程的内存空间是共享的

每一个线程都可以使用这些共享内存

6 房间的大小不同 决定容纳的人数也不同

假设房间很小 例如厕所的隔断 它是共享的 但是做多容纳1个人

有其他人使用 那么其他人就不能进去

这代表着 一个线程使用共享内存时 其他线程必须等他结束后才能用

7 互斥锁

防止他们进入的简单办法 门口加一把锁

用它的人锁上门 看到上面的锁 后面的人就会在门口排队

这就是互斥锁 Mutex

8 信号量

还有些房间 可以容纳不止一个人 比如厨房

假设厨房的容纳人数是N

也就是说如果房间的人数大于N 多出来的人只能等待

这好比某些内存区域 只能提供给固定数目的线程使用

解决办法：门口挂N把钥匙 进去一个人就取一把钥匙

出来时候再把钥匙挂回原来的地方

这就是信号量 作用保证多个线程之间不会相互冲突

互斥锁是信号量为1的一种特殊情况

后者完全可以取代前者

但是互斥锁相对简单 效率更高

所以在必须保证资源独占的情况下 还是采用互斥锁进行设计为主

操作系统的设计：

迅雷软件在运行时 是一个计算机的一个进程在运行

迅雷给用户提供下载的功能

而且下载的条目并不一定只有一个可以多个条目同时下载

说明迅雷是多线程的设计

在网游开发中

理论上来说 后端如果是保持长连接 一定会有一条线程用于处理监听客户端请求
还有一条线程用来处理已连接的客户端的交互

多线程开发的概括

1 在进行程度开发时 可以使用多线程协调开发

2 多条线程和主线程之间 资源共享

3 防止资源冲突 可以使用互斥锁或者信号量来进行控制

4 所有的线程都结束 程序才会结束

在Unity中使用多线程开发的注意事项

在Unity3D编程中 默认总有个主线程执行代码逻辑

也可以创建额外的线程和主线程同时运行 这种线程称为 分线程

在Unity中 仅仅只能在主线程中访问Unity3D的组件 对象和系统调用

这是一个重要的限制 所以 不涉及Unity API才可以放在分线程里 提高多核CPU的使用率

总结：

1 变量都共享的

2 不是UnityEngine的API可以在分线程运行

3 UnityEngine定义的基本结构Vector3 是可以访问的 但是Texure(继承于Object) 不能访问

4 obj.name不能访问 分线程报错：get_name can only be called form the main thread.

大多数的游戏引擎 都是单线程的 因为大多数引擎都是主循环结构

逻辑更新和画面更新的时间点要求有确定性。

如果引入多线程 需要做同步从而加大了开发难度

所以需要使用异步的功能 游戏引擎总是倾向于使用TimeSlicing (时间分割)的策略

Unity中的协程 yield语法的本质就是TimeSlicing

多线程代码

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Threading;

```

```
public class Text : MonoBehaviour
{
    Thread thread1;
    Thread thread2;

    void Start()
    {
        thread1 = new Thread(new ThreadStart(ThreadStart));
        thread1.Start();//无参的线程开启
        thread2 = new Thread(new
ParameterizedThreadStart(Thread2Fun));
        thread2.Start("HelloWorld");//有参数的线程开启
    }

    void ThreadStart()
    {
        Vector3 v = new Vector3();//可以访问

        //Texture2D a = new Texture2D(50,50); 继承于Object 不能访问
        Thread.Sleep(3000);//线程休眠  1000=1s
        Debug.Log(100);
        Debug.Log(100);
        Debug.Log(100);
        Debug.Log(100);
        Debug.Log(100);
        Debug.Log(100);
    }

    void Thread2Fun(object data)
    {
        Debug.Log(200);
        Debug.Log(200);
        Debug.Log(200);
        Debug.Log(200);
        Debug.Log(200);
        Debug.Log(200);
        Debug.Log(200);
    }

    void OnApplicationQuit()
    {
        thread1.Abort();//杀死线程
```

```
        thread2.Abort();
    }

}


```

互斥锁

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Threading;

public class Text : MonoBehaviour
{
    Thread thread1;
    Thread thread2;

    int x = 100;

    void Start()
    {
        thread1 = new Thread(new ThreadStart(ThreadStart));
        thread1.Start();//无参的线程开启
        thread2 = new Thread(new
ParameterizedThreadStart(Thread2Fun));
        thread2.Start("HelloWorld");//有参数的线程开启
    }

    void ThreadStart()
    {
        lock(this){
            x = 300;
            Debug.Log("Thread1:" + x);
            Debug.Log("Thread1:" + x);
            Debug.Log("Thread1:" + x);
        }
    }
```

```
31
32      void Thread2Fun(object data)
33      {
34          lock (this)
35          {
36              x = 800;
37              Debug.Log("Thread2:" + x);
38              Debug.Log("Thread2:" + x);
39              Debug.Log("Thread2:" + x);
40          }
41      }
42
43      void OnApplicationQuit()
44      {
45          thread1.Abort();//杀死线程
46          thread2.Abort();
47      }
48  }
```

Byte数组和Int类型的互转

```
1       //把int转化为byte[]
2       byte[] GetBytes(int s,bool asc)
3       {
4           byte[] buf = new byte[4];
5           if (asc)
6           {
7               for (int i = buf.Length - 1; i >= 0; i--)
8               {
9                   buf[i] = (byte)(s & 0x000000ff);
10                  s >>= 8;
11              }
12          }
13          else
14          {
15              for (int i = 0; i <buf.Length; i++)
16              {
17                  buf[i] = (byte)(s & 0x000000ff);
18                  s >>= 8;
```

```
            }
        }
        return buf;

    }


  int GetInt(byte[] buf,bool asc)
    {
        int r = 0;

        if (!asc)
        {
            for (int i = buf.Length - 1; i >= 0; i--)
            {
                r <<= 8;
                r |= (buf[i] & 0x000000ff);
            }
        }
        else
        {
            for (int i = 0; i <buf.Length; i++)
            {
                r <<= 8;
                r |= (buf[i] & 0x000000ff);
            }
        }
        return r;
    }
```

服务器代码：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;
```

```
8
9  namespace ServerSocket
10 {
11     class ServerSocket
12     {
13         static void Main(string[] args)
14         {
15             new ServerSocket().Start();
16         }
17
18         void Start()
19         {
20             IPAddress ip = IPAddress.Parse("127.0.0.1");
21             IPEndPoint point = new IPEndPoint(ip, 1234);
22             Socket serverSocket = new
   Socket(AddressFamily.InterNetwork, SocketType.Stream,
   ProtocolType.Tcp);
23             serverSocket.Bind(point);
24             serverSocket.Listen(20);
25             Console.WriteLine("服务器开启");
26
27             while (true)
28             {
29                 Socket clientSocket= serverSocket.Accept();
30                 Console.WriteLine("有客户端连接进来了");
31             }
32         }
33     }
34
35
36 }
37
```

客户端代码：

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System.Net;
```

```
using System.Net.Sockets;

public class Client : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }


    void OnGUI () {
        if (GUILayout.Button("连接服务器"))
        {
            Socket mySocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            mySocket.Connect("127.0.0.1", 1234);
        }
    }
}

```

NetWork

Unity自带的一套网络框架 目的实现状态同步

一般可以用于开发一些简单的网络项目或者Demo 局域网游戏等

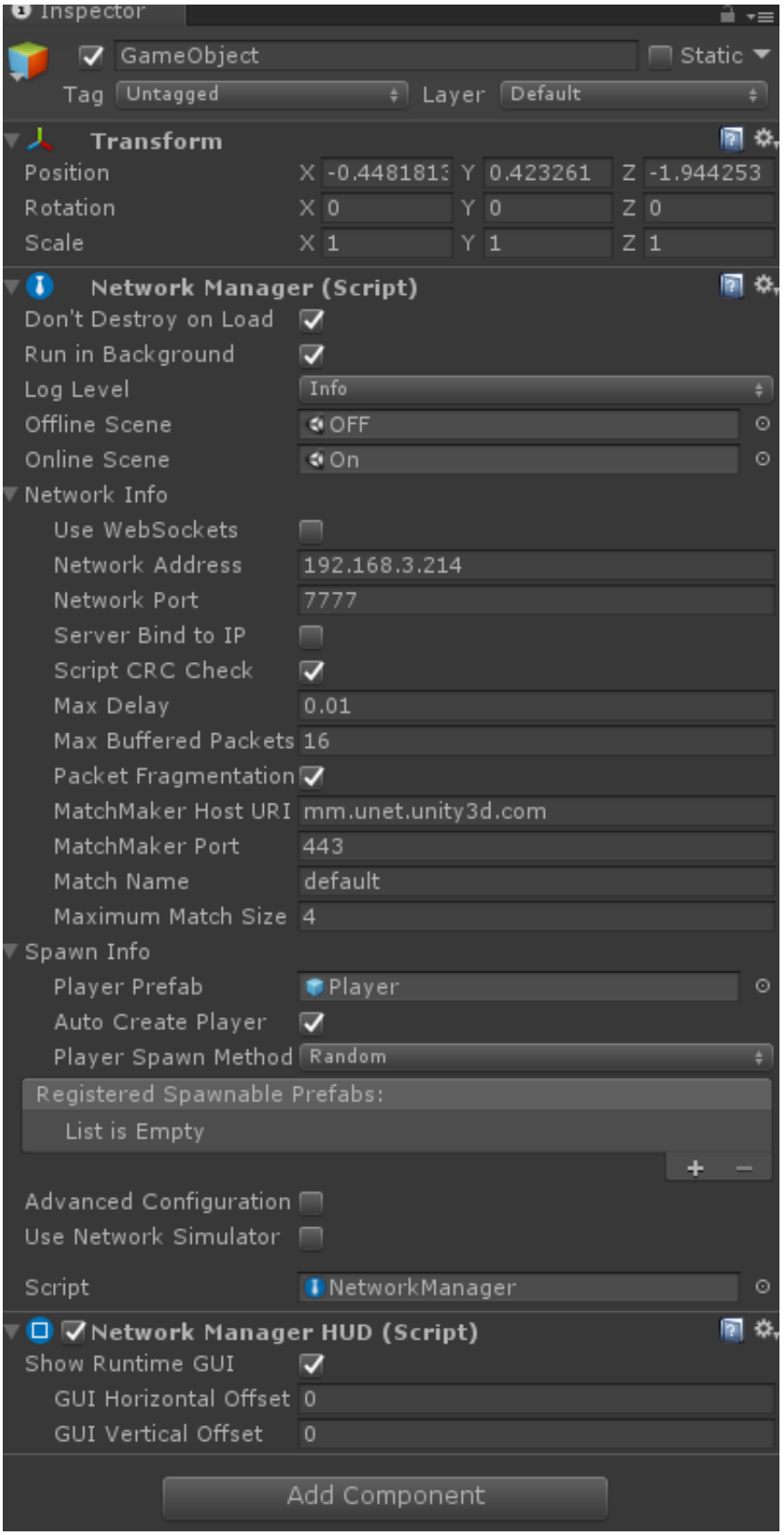
网游的运行机制

1台服务器->N台客户端(1 server->n Client)

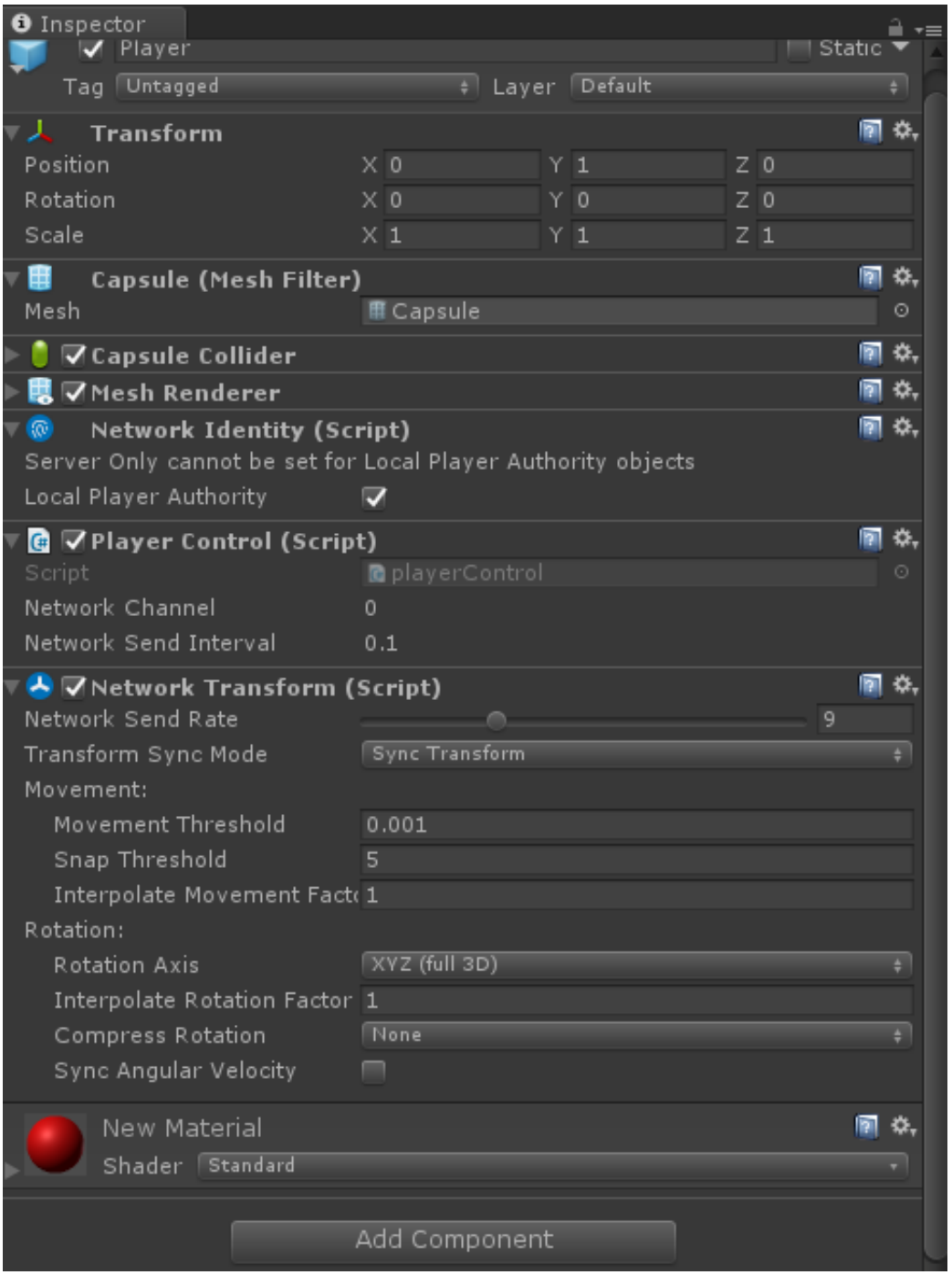
Client1->发送消息给Server->Server处理消息并发送给Client ->其他Client实现数据更新

模式类似于手机短消息的群发功能

创建空物体 添加NetWorkManager组件和NetWorkManagerHUD组件


Inspector
GameObject
Static
Tag Untagged
Layer Default
Transform
Position X -0.4481813 Y 0.423261 Z -1.944253
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Network Manager (Script)
Don't Destroy on Load
Run in Background
Log Level Info
Offline Scene OFF
Online Scene On
Network Info
Use WebSockets
Network Address 192.168.3.214
Network Port 7777
Server Bind to IP
Script CRC Check
Max Delay 0.01
Max Buffered Packets 16
Packet Fragmentation
MatchMaker Host URI mm.unet.unity3d.com
MatchMaker Port 443
Match Name default
Maximum Match Size 4
Spawn Info
Player Prefab Player
Auto Create Player
Player Spawn Method Random
Registered Spawnable Prefabs:
List is Empty
Advanced Configuration
Use Network Simulator
Script NetworkManager
Network Manager HUD (Script)
Show Runtime GUI
GUI Horizontal Offset 0
GUI Vertical Offset 0
Add Component

增加玩家预设



人物移动代码

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
```

```
using UnityEngine.Networking;

//同步NetWork组件时 必须继承的父类 NetWorkBehaviour
//IsLocalPlayer 是不是本地玩家
//IsClient 是不是客户端
//IsServer 是不是服务器端
public class playerControl : NetworkBehaviour {

    // Use this for initialization
    void Start () {

    }


    void Update () {
        //是否是当前可控制的玩家对象
        if (!isLocalPlayer)
            return;
        Move();
    }

    void Move()
    {
        float x = Input.GetAxis("Horizontal");
        float y = Input.GetAxis("Vertical");

        if (x != 0 || y != 0)
        {
            Vector3 dir = Camera.main.transform.TransformDirection(new
Vector3(x, 0, y));
            dir.y = 0f;
            transform.position += dir * 0.1f;
            transform.rotation = Quaternion.LookRotation(dir);
        }
    }

    public override void OnStartLocalPlayer()
    {
        GetComponentInChildren<Renderer>().material.color =
Color.blue;
    }

```

```
44
45 }
46
```