## 2018.1.17

1. 更改CameraController脚本，使用直接更改localAngle的方法控制CameraPos的转动，因为如果用Rotate会产生Bug(不确定为什么，但是会使得CameraPos的y轴逐渐偏移，可能是Rotate函数执行时屏蔽了其他物理过程。

2. 在大厅里小卡片显示房间信息，我希望对每一个小卡片的按钮动态加载一个加入房间的事件，但是发现这样需要一个参数传递进事件委托表示当前处于哪一个小卡片，然而如果直接用局部变量比如循环变量赋值给委托的参数，则执行委托时依然将其当做变量（比如循环结束后i的值为5，则执行委托时还是按照5去执行），所以最后还是直接在Unity窗口里面改。

3. 一开始就把RoomManager脚本附在Panel\_Right上，但是一开始Panel\_Right是隐藏的。

## 2018.1.20

1. 发现Layout里面的元素一旦SetActive(false)之后再SetActive(true)之后位就会混乱，发现原来如果要自动排列还需要增加LayoutElement Component。

2. GetComponent函数可能带来潜在的性能损耗，所以在StartScene的脚本里面，为了取得房间卡片以及Seat卡片的文本子物体，在Start函数里面用一个循环取出Text。

## 2018.1.21

1. 发现如果在 PhotonNetwork.JoinRoom执行完后面加上RPC调用，RPC不能被执行

如 if(PhotonNetwork.JoinRoom(roomname)

PhotonView.RPC(,,,)

这样子RPC并不能准确执行，经过多次调试确定这是因为JoinRoom实际上还没有完成就已经执行了接下来的RPC调用，所以应该在回调函数 OnLeftRoom里进行。

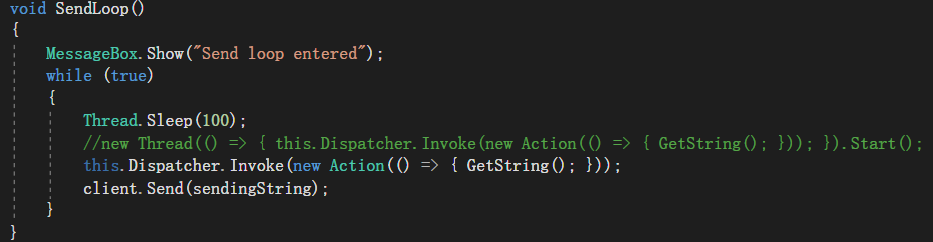
1. GameObject.SetActive(false) 应该等脚本里的代码都执行好再调用。

## 2018.1.22

1. 地图的Texture的大小可以调的比较大，这样不会在地图中出现太多重复的模式，看着更加舒服，猜测也可以提高性能。

2. 考虑在主场景中，需要客户端把自己的指令发送给游戏服务器，但是如果使用PhotonServer PUN进行传输指令，则每一帧指令都会传递给所有的客户端和游戏服务器，虽然可能并不会产生很大的影响，我仍然认为应该先尝试自己采用UDP连接，客户端仅仅把指令传给游戏服务器。于是我首先进行了一些尝试，用WPF窗体程序，上下两个文本框，上面的文本框里的文字实时通过UDP发送给本机上的Server，然后下面的文本框取得Server接受到的文本串。

3. 这样子需要进行多线程，发送和接收都需要单独开一个线程，然后在Send之前/Receive之后使用this.Dispatcher.Invoke(Action) 来在主线程（UI线程）里调用方法，对TextBox进行更新。而且在Send/Receive的死循环里要Sleep几十毫秒以上。否则会卡死。



## 2018.1.23

1. 经过测试，SetActive(false) 并不会让脚本失效。所以即使SetActive(false)之后，脚本里的代码仍然可执行，Update可能也不受影响（未测试）。

2. 对于·UDP监听者，不一定会产生阻塞，可以使用

while(udpListner.Available>0)

{

udpListener.Receive(ref endPoint);

}

的方法，表示在网络缓冲区有内容时进行读取，但是并没有考虑到如果有发送到其他端口的Udp包是否会产生阻塞。（我猜测不会，因为UdpClient已经绑定了一个端口，所以并不会管别的端口, 就是不太明白为什么还有一个ref endPoint，可能表示的是发送方源端口）。

1. 使用UDP协议让客户端给服务器发送，制造两个类 InstructionSender和InstructionReceiver, 进行信息交互，然后服务器控制坦克，然后玩家坦克通过一个中间体和服务器同步，中间体（reference）拥有PhotonView。

## 2018.1.24

1. 成功实现InstructionSender，InstructionReceiver两个类，用Instruction作为接口。InstructionReceiver用一个数组保存各个Client的Instruction。注意在初始化的时候，必须先 Instruction[] ClientInstructions = new Instruction[PlayerNum]

还得在for循环里

for(int i = 0; i< ClientInstructioins.Length; i++)

{

ClientInstructions[i] = new Instruction();

}

否则取数组元素后还是会产生NullPointerException。

2. 服务器在所有客户端的场景加载完毕后(MainGameLoader的Start函数结尾会进行RPC调用)，会用RPC调用把 player序号（座位号）- playerID的对应表发送给客户端，客户端根据该表进行Tank和Reference的绑定。tank在一开始就根据玩家数目生成在出生点。

## 2018.1.25

1. 依然遇到了卡顿问题，我把PUN升级了，发现要注意autojoinLobby’的设置在PhotonServerSettings界面上设置好，否则不会自动加入大厅。

2. 我发现PhotonNetwork.SendRate 和PhotonNetwork.SendRateOnSerialize是可以改的。。分别改到90和30卡顿大幅缓解已经可以接受。

3. 如果要进一步优化流畅度，接下来的工作应该是修改PUN源码里面的Plugin里面的TransformView中的内容，包含一些插值算法。

## 2018.1.26

1. 服务器端搞了自由漫步的Camera，可以进行上帝视角。

## 2018.1.27

1. 通过Transform.LookAt ，再加上Quaternion.Towards方法，计算出炮管旋转的方向。得出的EularAngle的x分量就是炮管向上/下的角度，而y分量就是在水平面里炮塔的转动角度。

2. 通过把炮塔的localEularAngle.y和枪管的x赋给syncer的y,x，然后再反着赋值回来，就可以实现炮塔、炮管的同步。

3. 通过Raycast计算出屏幕中心对应的坐标，炮塔转向该坐标。坐标已经封装在Instruction里面可以通过UDP传输。

## 2018.1.28

1. 准心、实际准心绘制。使用Raycast获取实际炮塔指向的位置。在联机模式下流畅度满意。

## 2018.1.29，2018.1.30

1. 坦克武器系统作为TankWeapon类绑定在tank上和TankMotion同一个位置，接收指令，利用Instruction，由MainGameLoader以及ServerMainController进行设置。

2. 用Ammo类指定弹药，以及相关特性。

3. 炮弹的prefab放在Resources文件夹下，注意PhotonView和RigidBody需要在同一个GameObject下才可以对rigidbody施加力后同步，否则rigidbody如果在子物体上，那么Owner可以看到子物体移动，而客户端无法看到变化因为PhotonView同步的是母物体。

4. Shell类里面执行死亡的代码还得分 在线模式或离线模式，所以得在一开始设定好，所以在TankWeapon的Fire函数执行时就得设定。

## 2018.1.31

1. 使用TankArmor类表示坦克装甲的碰撞器，在TankArmor类里面再触发Shell类的方法。

2. 注意OnCollisionEnter只在有刚体的GameObject上触发，带碰撞器的子物体并不会触发该函数，因为所有的碰撞器都属于最近的一个RigidBody

3. OnCollisionEnter参数Collision的Contacts包含很多个点，所以还要具体考虑到底是哪一个点先被击中。

## 2018.2.1

1. 碰撞检测的最终解决方案：在OnTriggerEnter函数里（把炮弹设为Trigger），让炮弹回退一段距离，进行Raycast检测。

2. 注意客户端必须保留地形和装甲的Collider，否则无法获取正确的炮塔指向位置。

## 2018.2.2

1. 发现客户端有时候炮弹击中效果错误的原因是因为把击中效果设置成了炮弹的子物体，客户端和服务器进行同步会使得实际的炮弹位置发生位移，也有可能是在生成爆炸效果之后子弹还进行了一次FixedUpdate，导致位置偏移。把击中效果设置为WorldSpace里产生就不会有问题。

2. TankBody类进行伤害统计。

3. 在全局静态变量中设计一个统计类进行统计击中信息。

## 2018.2.3

1. 实现了统计信息的序列化以及网络传输。

2. 在游戏开始、炮弹发射、炮弹击中时进行传输。

## 2018.2.4

1. 实现了在服务器以及离线游戏中按Tab键进入菜单可以选择结束当前游戏，并且在联机模式下可以保留房间的各个信息，也就是游戏开始前的房间界面。

2. 为此需要DontDestroyOnload StartScene里面的一些GameObject，按照广泛采用的做法，把这些GameObject放在最初的一个场景里，这样子就不会在返回之前场景时产生冗余的GameObject.

3. WholeGameController控制，在Awake函数里控制GameObject和Script是否激活

4.结束场景时关闭UdpListener，防止端口被占用。