SD

此项目是与广州美术学院进行合作而开发的一款*Unity3D*游戏

技术选型理由

游戏客户端编程选择

> - **Unity3D**
> Unity3D是由Unity Technologies开发的一个让玩家轻松创建诸如三维视频游戏、建筑可视化、实时三维动画等类型互动内容的多平台的综合型游戏开发工具，是一个全面整合的专业游戏引擎。
> 使用Unity3D进行编程，简化了许多编程的流程，也非常适合于对3D模型的处理。

服务器编程选择

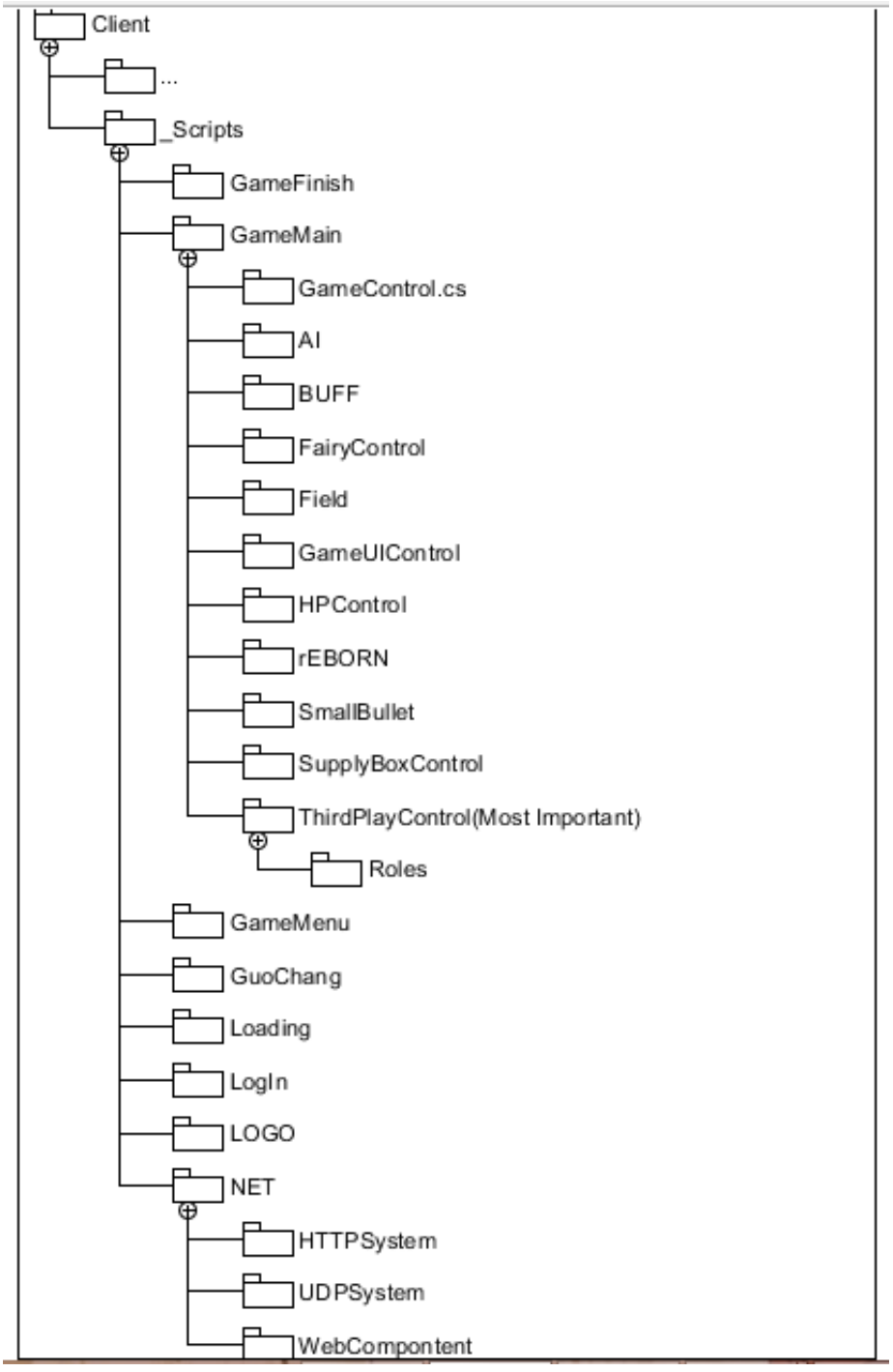
> - **Node.js**
> Node.js是一个基于Chrome JavaScript运行时建立的平台， 用于方便地搭建响应速度快、易于扩展的网络应用。Node.js 使用事件驱动，非阻塞I/O 模型而得以轻量和高效，非常适合在分布式设备上运行数据密集型的实时应用。
> 在项目中用到的Node.js 是为了方便在局域网下进行通信，因为是射击游戏的设定，所以我们用到的通信协议是UDP，也就是说不需要去考虑每一条信息都被接受到，只要维持在每帧的刷新后有信息的更新就可以了。
> Node.js能够迅速的搭建起来一个轻量级的服务器，虽然不能承载大量的需求，但是对于局域网而言已经足够了。

数据库选择

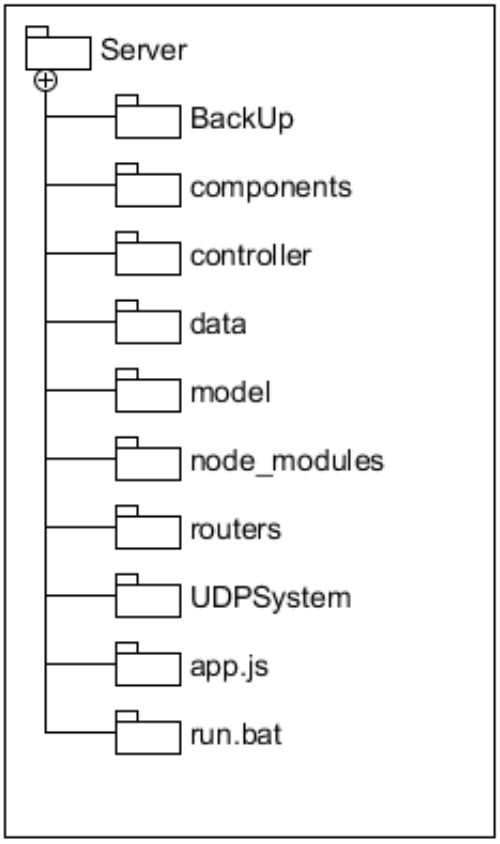
- **MongDB**
  MongDB也是一款轻量的数据库，没错，我们这个项目在后台发面都是轻量级的，他便于操作和存储，而且相对来说更易于读写，对于不需要存储太多数据的后台来说，无疑是比MySql更好的选择。

文件结构

客户端文件结构



后台文件结构



模块划分

客户端

- GameFinish
  结束画面
- GameMenu
  菜单画面
- GuoChang
  过场动画
- Loading
  Loading界面
- LogIn
  注册页面
- LOGO

LOGO标志
- NET
  网络插件
- GameMain
  主场景控制

  - GameControl.cs
    游戏主流程控制
  - AI
    AI控制
  - BUFF
    BUFF计算
  - FairyControl
    特效
  - Field
    碰撞计算
  - GameUIControl
    内部UI
  - HPControl
    血量
  - rEBORN
    玩家重新出生
  - SmallBullet
    子弹
  - SupplyBoxControl
    补给包刷新
  - ThirdPlayControl(Most Important)
    角色控制
  - Roles
    各种不同角色的设定（技能等）

服务器

- BackUp
  备份
- components
  相关插件，包括IP解析，JSON格式数据解析等
- controller
  控制器，服务于HTTP请求
- data
  数据存储
- model
  数据基本模型
- node_modules
  node.js附带modules
- routers
  路由，服务于HTTP
- UDPSystem
  基于UDP的局域网系统
- app.js
  服务器入口
- run.bat
  windows运行脚本

架构设计

客户端架构设计

- GameController
  在游戏的设计中，我们设置了一个GameController来控制整个游戏的过程中的事件（类似于cocos2d的导演类），在该类中，我们直接（在该类的Update函数直接调用）或间接（通过该类来调用其他类的函数）的控制了游戏过程中的AI，BUFF，特效，碰撞领域，游戏主流程内的UI，HP计算，角色复活，子弹，补给包刷新以及最重要的玩家控制。
- OneScene - OneController
  除了玩家进行游戏的主要场景外，我们对于每一个场景，如开头的过场动画，游戏结束提醒，游戏Loading界面都对应了一个的Controller，他们分别用来控制对应的Unity3D场景。
- Less Update Function
  虽然在Unity的设计中，每一个脚本都可以挂载Update或者FixedUpdate函数来使得脚本每一帧都被执行。与此同时，代价就是每帧需要进行的运算变多（起码需要执行更多不同的脚本），所以在架构设计的时候，我们尽量避免了在多个脚本下运行，将所有的Update函数放到了GameController和ThirdPlayerController 下运行，大大的简化了代码。
- Use Memory instead of GameObject find
  在Unity内部，提供了GameObject.find的一系列接口，为的就是方便编程者能够获取到Unity场景中的任何一个对象，当然，这样每一次find需要进行的操作就是对场景中的每一个对象进行遍历，我们用一个Data.cs的全局脚本作为内存来缓存我们需要寻找以及遍历的对象，同样提高了游戏的运行速度。
- Unity Prefab
  一个游戏对象及其组合的集合，或者理解成一个游戏对象的一个类即可，相同的对象可以通过一个Prefab来实现，我们在游戏中大量的使用该种集合来简化设计。

服务器架构设计

- Two Kinds Of Requirement
  对于后台的实现，其实分为2种需求，一种是游戏中玩家的登陆，获得该局域网下的userID（唯一），以及玩家的名字等类似于用户登陆注册的需求，这一类通信需求是要求通信稳定，并且服务器能给出正确与否的应答，所以考虑使用TCP来进行通信，最终我们确定了使用HTTP的POST来进行该类型的通信。
  另一方面则是在玩家的实战游戏中，通过通信来完成客户端的同步，此时的通信要求快，最好是能够有每秒60帧的同步通信，以至于游戏中不会出现卡顿，瞬移，向后移位的现象，所以我们决定使用UDP来完成这种类型的通信，并为了防止数据包发生阻塞（排队队列过长），在客户端和服务器都设计了能够主动Abort或者丢弃数据包的功能，来保证信息的及时。
- Small Data With Small DataBase
  游戏需要进行数据库存储的东西非常少，因为游戏的主要通信是玩家之间进行的操作而产生的传递需求，所以使用MongoDB去记录玩家的基本信息，用户名，id，游戏时间，游戏中的积分即可。
- Distribute By Server
  游戏的信息机制为，玩家A将消息传入服务器，服务器再将该消息记录，并同时发给应该接受到该信息的玩家，在这种模式下，需要对用户进行分组，即能够互相接受到消息或者能够互相传递的消息的用户在一组，那么他们就在一个房间里面，在客户端的表示就是他们在同一张地图下进行游戏。

软件设计技术

帧动画

开发游戏避免不了动画，而我们采用的是3DM下的帧动画，同时将位移取消（具体的位移应该由玩家的操作决定），而动画的效果是由美术组提供，包括特效动画，由3DM模型预设设定，在控制函数中以Animator的形式调用。
\ClientCode_Scripts\GameMain\ThirdPlayerControl\ThirdPlayerControl.cs

状态机

从游戏人物禁止，行动，释放技能，蓄力，开枪等动作都可以视为一个状态，要实现角色的动作转变只需要完成状态之间的转换即可。
\ClientCode_Scripts\GameMain\ThirdPlayerControl\ThirdPlayerControl.cs

订阅发布

在TPC中（一个用于控制玩家行动的controller），会在指定的时刻发出事件通知（用户输入，伤害判断等），通过参数传递，来通知各个不同的脚本接受事件并触发各自的回调函数，从而实现角色的控制。
\ClientCode_Scripts\GameMain\ThirdPlayerControl\InputController.cs

协程延时

Unity的协程系统是基于C#的一个简单而强大的接口 ，IEnumerator，它允许为自己的集合类型编写枚举器。我们通过协程来实现计时，角色延时刷新等延后计算。
\ClientCode_Scripts\GameMain\GameControl.cs

消息队列

为了更加方便的进行通信，在客户端内的用户的任何操作为Message，而进行通信的就只有Message了，此时再使用一个数组将他们由时间戳的顺序进行存储，便形成了消息队列。在有了消息队列之后，不仅通信变得更加方便，而且添加消息和放弃不重要的消息也可以通过很简单的方法实现。为进一步的优化设计提供了可能。
\ClientCode_Scripts\NET\UDPSystem\UDPBack.cs

用户分组

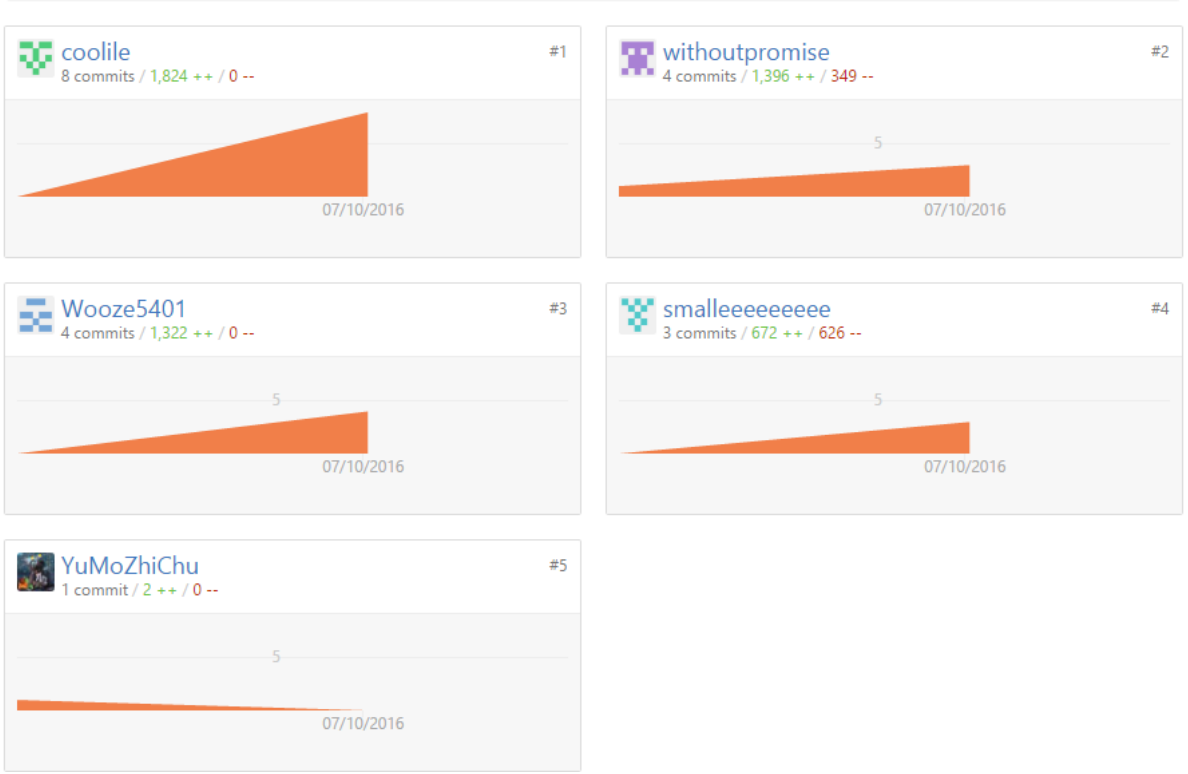
简单来说，就是将用户分别在一个一个的房间里面，其实这是一件理所当然的事情，因为作为一个匹配机制性的，以一局游戏为计算的FPS，理应将一个玩家的玩家分配到同一个子网中，再将子网的网关和服务器相连接来实现数据传输即可。
在进一步的设计中，可以去考虑模仿MineCraft的个人服务器机制，在加强玩家互动的同时可以减少服务器的压力。
\ServerCode\lib\components\roomListOp.js

主动丢包

对于FPS游戏的同步，有主动的丢包反而不会影响游戏的正常运行，因为FPS只需要同步玩家位置，血量，角度等信息，而不是太需要保证每一个数据包都能够发挥作用，在我们的游戏中，丢包率大概有10%左右。
而丢包的机制在客户端的实现方式即为判断消息队列，越往后的消息则越大的概率进行丢弃，具体算法参考了计网课程的丢包机制（但实验发现，直接将消息队列控制在一个上限，多余的统统丢弃的效果反而更好）。
丢包在服务器的实现便是查阅当前请求数（使用全局变量，每建立一条请求+1，结束则-1），当请求数多于上限时，同样可以使用丢包机制，丢弃的概率变得越来越大，当然，直接将请求数维持于某一上限对于这种局部的网络效果更好。
\ClientCode_Scripts\NET\UDPSystem\UDPBack.cs
\ServerCode\lib\components\UDPSystem\UDP.js

GitHub及分工

GITHUD地址：
https://github.com/YuMoZhiChu/Popper
截止至7月14号的github-contribute截图



小组分工与贡献率

			贡献

学号	姓名	分工	率
13331170	刘健坤（组长）	后台代码架构设计，通信系统编写，答辩	23%
13331027	陈奕龙	需求分析，通信架构设计，客户端架构设计与编写	23%
13331009	陈德森	项目管理，文档整理，后台路由设计与整合	17%
13331266	魏婉婷	UI设计，交互设计，用户体验设计，后台组件编写	19%
13331080	胡志亮	需求分析，客户端游戏要素编写	18%

制品与贡献率

制品	刘健坤	陈奕龙	陈德森	魏婉婷	胡志亮
客户端游戏要素	-	10%	-	-	90%
后台HTTP及路由管理	10%	-	90%	-	-
后台基础架构	90%	-	-	10%	-
后台额外组件	-	-	-	100%	-
客户端基础架构	-	100%	-	-	-
客户端通信	80%	20%	-	-	-
用户手册	-	-	-	100%	-
SD	60%	40%	-	-	-
SRS	-	-	40%	-	60%