

**课 程 设 计 报 告**

**课程名称:** C#高级程序设计

**设计题目** 3D坦克大战联网对战版之客户端

**院（系）:** 计算机学院（软件学院）

**专业年级:**软件工程（游戏开发工程师）2017级

**学 号:** 171530445

**姓 名:** 马 昊 瀛

**指导教师:** 彭 伟 国

**2019 年 12 月 16 日**

目 录

[1 系统需求分析 3](#_Toc27865202)

[1.1项目意义 3](#_Toc27865203)

[1.2系统需求分析 3](#_Toc27865204)

[**1.2.1**系统需求调查研究 3](#_Toc27865205)

[**1.2.2**系统需求调查结果 4](#_Toc27865206)

[**1.2.3**系统功能设计要求 5](#_Toc27865207)

[2系统概要设计 6](#_Toc27865208)

[2.1登录注册功能 6](#_Toc27865209)

[3系统详细设计 6](#_Toc27865210)

[3.1坦克 6](#_Toc27865211)

[3.2系统调试及解决方法 7](#_Toc27865212)

[4系统运行结果 8](#_Toc27865213)

[**5**课程设计总结与心得 15](#_Toc27865214)

[参考文献 16](#_Toc27865215)

[附录：源代码 17](#_Toc27865216)

# 1 系统需求分析

## 1.1项目意义

《3D坦克大战联网对战版》是一款多人坦克射击游戏。坦克拥有厚重的装甲和凶猛的火力, 驰骋疆场，越障跨壕，无所阻挡,在游戏中见证坦克工业的历史轮印。

此外，我们真实的模拟操控系统，可以让玩家体验真实的战场混战，感受最特别的游戏乐趣。热血刺激的游戏画面，可以点燃玩家的内心，给玩家最激情的战斗体验。动感的背景音乐，可以让玩家时刻保持着热血，振奋玩家的神经。

## 1.2系统需求分析

**1.2.1**系统需求调查研究

玩到好玩的游戏时，我总希望有朝一日能做出优秀的游戏作品；对生活有感悟时，也总会期待在游戏中表达感想。自Unity引擎流行以来，个人和小团队也能制作精良的游戏，实现梦想不再遥远。

由于我很喜欢“坦克世界”这款游戏，所以就有了开发一款类似坦克世界这款游戏的想法。



图1-1 塔克大战官网和贴吧的百度搜索结果

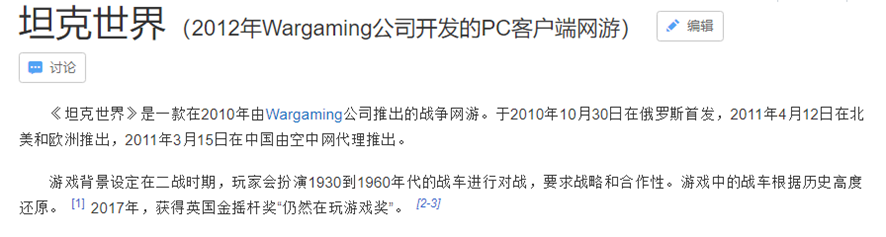


图1-2 塔克大战介绍

**1.2.2**系统需求调查结果

使用Unity引擎 ，游戏开发者再也不用过度关心底层复杂系统的实现，只需关心具体的游戏逻辑。一般来说，游戏引擎都能够很好地处理渲染、物理等通用的底层模块，但对于那些不完全通用的功能，比如本游戏要用到的网络模块，引擎往往没能提供通用的解决方案。这就要求开发者对网络底层有足够深刻的理解，才能做出优质的网络游戏。

如今，游戏联网是一大趋势。几大热门的手机游戏厂商只开发网络游戏，老牌单机游戏也纷纷添加联网功能。作为有志于从事游戏行业、渴望做出顶级产品的我们，更需要深入探讨网络游戏的开发技术。

为完成游戏，第一步工作是完善底层框架。客户端框架使用异步资源加载和事件系统。服务端程序的两大核心是处理客户端的消息和存储玩家数据。图1-3展示的是最基础的单进程服务端结构，客户端与服务端通过TCP连接，使两者可以传递数据；服务端还连接着MySQL数据库，可将玩家数据保存到数据库中。

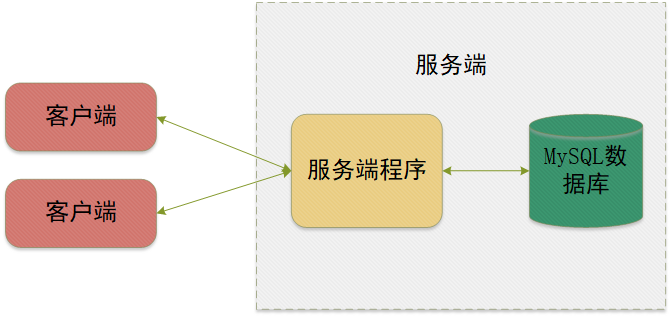


图1-3 单进程服务端程序结构

第二步工作是完成游戏的策划，在我看来好的游戏设计有3个关键因素：

1. 代入感，吸引玩家进来的关键因素
2. 核心玩法，游戏好不好玩是游戏乐趣所在
3. 深度，可以影响玩家能玩多久和游戏的付费潜力

目前的第1期工程是要跑通游戏流程、完成核心战斗体验以及一两个界面系统。为了让游戏有较好的代入感，需要写一段剧情，坦克的资料也需要和历史相符，买了不少坦克类书籍，一边学一边写。另外还做了个很简单的数值，用于计算游戏中各系统的产生和消耗。



图1-4 为了做游戏，购买了些坦克书籍

开发游戏的第3步便是把各个业务系统开发出来，这是较为耗时的一个步骤，因为对每个系统，都要自己完成系统策划、界面设计，然后开发服务端和客户端两部分程序。

**1.2.3**系统功能设计要求

图1-5展示了服务端程序的内部结构。“网络底层”是指处理网络连接的底层模块，它有处理粘包半包、协议解析等功能。消息处理模块属于游戏的逻辑层，比如当收到客户端的MsgMove协议时，服务端会在消息处理模块中记录玩家坐标，然后将MsgMove协议广播给所有客户端。在服务端中，事件处理指的是玩家上线和下线。当玩家上线，可能需要做些初始化的操作；当玩家下线，可能也需要做些数据记录，这些逻辑便在事件处理模块中执行。数据库底层模块提供了保存玩家数据、读取玩家数据、注册、检验用户名及密码是否正确等的功能，是服务端和数据库交互的一层封装。存储结构指定哪些数据需要保存，比如在线记事本中需要保存文本信息，对于大部分游戏需要存储玩家的金币、经验、等级等信息。

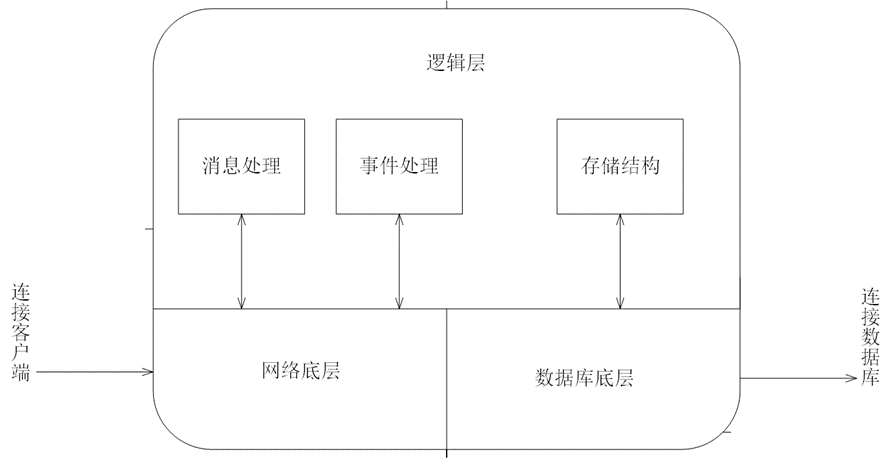


图1-5 服务端程序的内部结构

# 2系统概要设计

## 2.1登录注册功能

从客户端的角度看，在线记事本至少需要4条协议。MsgRegister和MsgLogin 是注册和登录协议。登录后，客户端需要显示已保存的文本信息，它通过MsgGetText获取文本。编辑文本后，玩家点击保存按钮，客户端发送MsgSaveText协议，更新文本信息。协议发送的流程如图7-51所示。

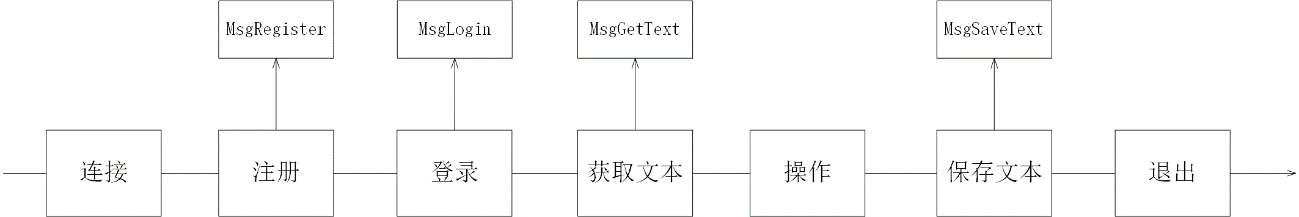
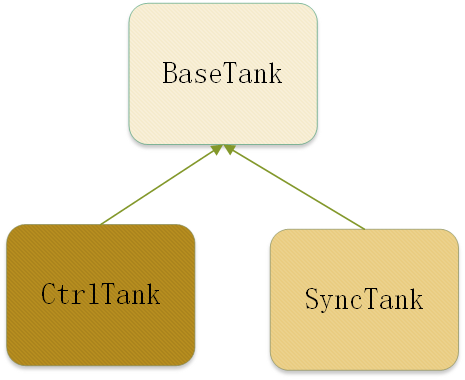


图2-1 客户端需要发送的协议

# 3系统详细设计

## 3.1坦克

图3-1所示的是坦克类的结构。BaseTank是坦克基类，它包含坦克的一些通用功能，比如开炮、皮肤设置等。CtrlTank为玩家控制的坦克，它包含行走控制等功能。SyncTank是同步坦克，它会根据网络数据移动坦克、控制开炮。

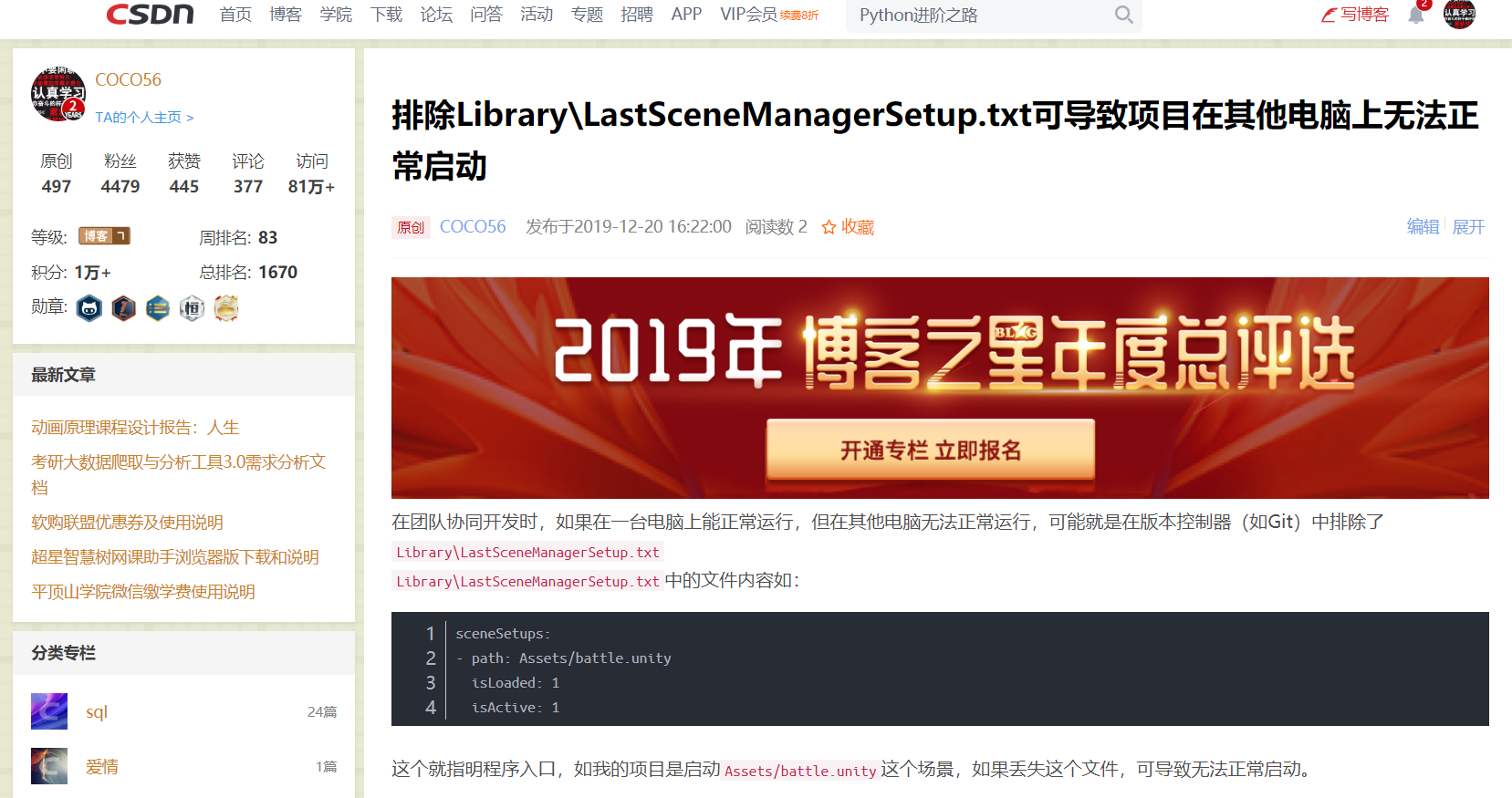


3-1 坦克类的层级结构

## 3.2系统调试及解决方法

实现过程中曾出现以下调试错误信息：

项目在组长电脑上能正常运行，但在我的电脑上无法正常运行，经过搜索发现是在版本控制器中排除了Library\LastSceneManagerSetup.txt这个文件。



3-3 搜到的CSDN博文

# 4系统运行结果

打开客户端，会看到如图4-1所示的主界面。玩家可以输入用户名和密码登入游戏，或者点击注册按钮注册一个新账号。



图4-1 主界面

图4-2是注册界面，玩家在此界面可以注册账号



图4-2 注册界面

图4-3 是填过注册信息后的界面，玩家需要填写正确的的注册信息，才能成功注册账号。



图4-3 填过注册信息后的界面

图4-4是注册成功界面，若玩家输入的注册信息无误，点击注册按钮后会提示注册成功。



图4-4 注册成功界面

图4-5是填写登录信息界面，玩家可以填写之前注册过的账号信息，以进行登录。



图4-5 填写登录信息界面

图4-6是成功登录后的界面，若玩家填写的登录信息无误，点击登录按钮后会转入这一界面。



图4-6 成功登录后的界面

图4-7是被踢下线界面，若玩家在登录期间有人在其他设备登录了他的账号，则系统会提示他已被踢下线。



图4-7 被踢下线界面

图4-8是创建好房间后的界面，玩家在此界面可以选择开战，或者退出此房间。



图4-8 创建好房间后的界面

图4-9是开战失败的界面，两队至少有一名玩家，并且只有队长才能开战，否则，点击开战按钮后会提示开战失败。



图4-9 开战失败的界面

图4-10是符合开战条件并等待开战的界面，图中双方各有一名玩家，此时符合开战条件，队长玩家（创建房间的人）可以选择开战。



图4-10 符合开战条件并等待开战的界面

图4-11是玩家成功进入游戏的界面，在此界面的下方红色的进度条显示的是玩家的血量，上方显示的是双方当前所存活的玩家。



图4-11 成功进入游戏的界面

图4-12是玩家被击中的界面，下方红色的进度条里的数字会比被击中前小，每被集中一次，血量降低35点。



图4-12 玩家被击中的界面

图4-13是玩家被击翻的界面，玩家被击中时，除了降低血量外，还有一定的几率被击翻。



图4-13 玩家被击翻的界面

图4-14是击杀其他玩家的界面，若玩家将其他玩家击杀，则会提示已击杀，并且对方阵营的当前存货坦克量比之前小1。



图4-14 击杀其他玩家的界面

图4-15是玩家胜利的界面，若敌方阵营的坦克数量为0，则当前阵营胜利，并提示玩家已胜利。



图4-15 玩家胜利的界面

图4-16是玩家失败的界面，若当前玩家阵营的坦克数量为0，则会受到被击败的提示。



图4-16 玩家失败的界面

**5**课程设计总结与心得

接触C#已经有半学期时间，对C#感触颇多。C#作为逻辑性极强，功能强大的程序编程语言，仅仅学习半学期是远远不够的，希望自己能在这条路上不断前行，能够活到老学到老。

跟绝大部分的初学者一样，当看见密密麻麻的代码，就觉得这会是一门枯燥无味的课程。但在平时生活中使用各种功能强大的app，应用程序等时，我又会对程序背后的代码感到好奇。就是在这样矛盾的心情中，我一点点初步掌握了C#。

C#中最难理解也是最有意思的莫过于类的使用了。总的来说类是属性相同的方法或对象的集合。编程语言中的类较为抽象，难以理解。编程语言中类的思想在现实生活有许多体现。比如我们通常会把马路上所有的四轮交通工具归为一类----汽车，在生物学中也使用界、门、纲、目、科、属、种把所有生物分级分类。似乎这样想，类很好理解。但在具体编程中会遇到许多问题，比如类成员中各个成员变量的访问权限问题以及类与类之间的逻辑关系问题，等等。解决这些问题，需要对事物和目标有准确的理解和把握，当然扎实的C#功底基础也是必不可缺的。如果能够充分使用类，那么代码的编写会变得井井有条，至少代码会看上去富有生命力。

由于半学期的学习时间实在太少，想要深刻理解并灵活运用C#是需要多年的实践学习的。光是上课时间认真听讲是远远不够的，还需要课下不断的练习钻研。学习c#语言不能停留在学习它的语法规则，而是利用学到的知识编写c#语言程序，解决实际问题。即把c#语言作为工具，描述解决实际问题的步骤，由计算机帮助我们解题。只有通过上机才能检验自己是否掌握c#语言、自己编写的程序是否能够正确地解题。语法是学习编程语言的关键，光靠背语法，是远远不够的（其他语言学习也是如此）。记笔记是上课时候必须要做的一件事，但不能认为做了笔记就不用上课认真听讲了。C#作为一名理科学科是需要不断深入理解实践才能掌握的。有几点学习心得想分享一下：

1. 不要妄自菲薄 。不能因为遇到困难就不相信自己，每个人都是从零开始。
2. 坚持不懈。学习C#是件漫长的过程，需要不断激励自己鼓励自己。
3. 多和他人交流。一个人的力量是渺小的，多个人相互交流不断切磋才能不断进步。
4. 不要偷懒，经常思考。C#中有许多思想是和现实生活相通的，不断对生活感悟对生活进行反思，从生活中寻找灵感，这样能提高对C#的兴趣以及学习的主动性和趣味性。
5. 经常上机实验，动手操作。实践出真知，在实践中理解那些深奥晦涩难懂的专业知识。
6. 在编程时除了要认真谨慎之外，更重要的是掌握基本的知识，比如if语句和while语句的区别。初学者常常会被这两个语句搞得晕头转向，在编程过程也不会太在意两个语句的使用范围。再比如C#中命名空间，这是常常忽略的方面，初学者一开始编程时会发现定义一个变量或者函数常常会遇到系统提醒错误，开始认真检查下来，却怎么也找不到错在哪，这很有可能就是命名空间错误。

该游戏的实现利用了很多C#本身携带的功能。大部分功能的实现是依靠类和函数的调用的。通过本课程的实验设计，我更清楚的认识了C#。我相信在未来的编程中能更好的利用C#来提高编程的效率。

# 参考文献

[1] 李心蕊.独立游戏的设计艺术和文化特性研究[D].齐齐哈尔：齐齐哈尔大学，2016.

[2] 刘晓扬.基于社交网络游戏的行为干预研究[D].哈尔滨：哈尔滨工业大学,2013.

[3] 王琛瑜.移动游戏界面视觉风格的研究与设计应用[D].上海：东华大学,2017.

[4] 李绍龙.网络游戏用户行为分析和应用[D].北京：北京交通大学, 2009.

[5] 郭振兴.手机“废城赛车”游戏软件的技术研发[D].天津：天津大学,2015.

[6] 杨观.基于Unity的游戏逻辑引擎的设计与实现[D].哈尔滨：哈尔滨工业大学,2015.

[7] 周晓风.基于MAYA和VIRTOOLS的虚拟仓储物流系统的研究与实现[D].上海：上海交通大学,2011.

[8] 贺瞿. 基于动力学原理的Maya粒子特效插件[D].上海：上海交通大学,2011.

[9] 刘佳.关于photoshop应用的几个小技巧[J].信息与电脑(理论版),2013,5(03):187-188.

[10] 赵娟. 地图编辑器在苹果手机游戏设计中的开发与应用[D].北京：北京工业大学,2013.

[11] 张海藩.软件工程导论（第6版）[M].清华大学出版社,2016.

[12] 许鹏森. 游戏软件黑盒测试方法研究与应用[D].武汉：华中科技大学,2008.

附录：源代码

* 3D\_Client\Assets\Autumn\Shaders Scripts\CameraShake.cs

1. using UnityEngine;
2. class Noise
3. {
4. Vector2 seed;
5. float speed;
7. public Noise(float speed)
8. {
9. this.seed = new Vector2(Random.value, Random.value);
10. this.speed = speed;
11. }
13. public float Update()
14. {
15. seed += Vector2.one \* speed \* Time.deltaTime;
16. return Mathf.PerlinNoise(seed.x, seed.y) \* 2 - 1;
17. }
18. }
19. public class CameraShake : MonoBehaviour
20. {
21. Transform myTransform;
22. Vector3 initPos;
24. public float bigRadius = 3.0f;
25. public float bigSpeed = 0.5f;
26. public float smallRadius = 0.5f;
27. public float smallSpeed = 2.0f;
29. Noise noiseBigX;
30. Noise noiseBigY;
31. Noise noiseSmallX;
32. Noise noiseSmallY;
34. void Start()
35. {
36. myTransform = this.transform;
37. initPos = myTransform.position;
38. noiseBigX = new Noise(bigSpeed);
39. noiseBigY = new Noise(bigSpeed);
40. noiseSmallX = new Noise(smallSpeed);
41. noiseSmallY = new Noise(smallSpeed);
42. }
44. void Update()
45. {
46. myTransform.position = initPos + new Vector3(noiseBigX.Update() \* bigRadius + noiseSmallX.Update() \* smallRadius, noiseBigY.Update() \* bigRadius + noiseSmallY.Update() \* smallRadius, 0);
47. }
48. }

* 3D\_Client\Assets\Autumn\Standard Assets\Image Effects (Pro Only)\ ImageEffectBase.cs

1. using UnityEngine;
2. [RequireComponent (typeof(Camera))]
3. [AddComponentMenu("")]
4. public class ImageEffectBase : MonoBehaviour {
5. /// Provides a shader property that is set in the inspector
6. /// and a material instantiated from the shader
7. public Shader   shader;
8. private Material m\_Material;
9. protected virtual void Start ()
10. {
11. // Disable if we don't support image effects
12. if (!SystemInfo.supportsImageEffects) {
13. enabled = false;
14. return;
15. }
17. // Disable the image effect if the shader can't
18. // run on the users graphics card
19. if (!shader || !shader.isSupported)
20. enabled = false;
21. }
22. protected Material material {
23. get {
24. if (m\_Material == null) {
25. m\_Material = new Material (shader);
26. m\_Material.hideFlags = HideFlags.HideAndDontSave;
27. }
28. return m\_Material;
29. }
30. }
32. protected virtual void OnDisable() {
33. if( m\_Material ) {
34. DestroyImmediate( m\_Material );
35. }
36. }
37. }

* 3D\_Client\Assets\Autumn\Standard Assets\Image Effects (Pro Only)\ MotionBlur.cs

1. using UnityEngine;
2. // This class implements simple ghosting type Motion Blur.
3. // If Extra Blur is selected, the scene will allways be a little blurred,
4. // as it is scaled to a smaller resolution.
5. // The effect works by accumulating the previous frames in an accumulation
6. // texture.
7. [ExecuteInEditMode]
8. [AddComponentMenu("Image Effects/Motion Blur (Color Accumulation)")]
9. [RequireComponent(typeof(Camera))]
10. public class MotionBlur : ImageEffectBase
11. {
12. public float blurAmount = 0.8f;
13. public bool extraBlur = false;
15. private RenderTexture accumTexture;
17. override protected void Start()
18. {
19. if(!SystemInfo.supportsRenderTextures)
20. {
21. enabled = false;
22. return;
23. }
24. base.Start();
25. }
27. override protected void OnDisable()
28. {
29. base.OnDisable();
30. DestroyImmediate(accumTexture);
31. }
32. // Called by camera to apply image effect
33. void OnRenderImage (RenderTexture source, RenderTexture destination)
34. {
35. // Create the accumulation texture
36. if (accumTexture == null || accumTexture.width != source.width || accumTexture.height != source.height)
37. {
38. DestroyImmediate(accumTexture);
39. accumTexture = new RenderTexture(source.width, source.height, 0);
40. accumTexture.hideFlags = HideFlags.HideAndDontSave;
41. Graphics.Blit( source, accumTexture );
42. }
44. // If Extra Blur is selected, downscale the texture to 4x4 smaller resolution.
45. if (extraBlur)
46. {
47. RenderTexture blurbuffer = RenderTexture.GetTemporary(source.width/4, source.height/4, 0);
48. Graphics.Blit(accumTexture, blurbuffer);
49. Graphics.Blit(blurbuffer,accumTexture);
50. RenderTexture.ReleaseTemporary(blurbuffer);
51. }
53. // Clamp the motion blur variable, so it can never leave permanent trails in the image
54. blurAmount = Mathf.Clamp( blurAmount, 0.0f, 0.92f );
56. // Setup the texture and floating point values in the shader
57. material.SetTexture("\_MainTex", accumTexture);
58. material.SetFloat("\_AccumOrig", 1.0F-blurAmount);
60. // Render the image using the motion blur shader
61. Graphics.Blit (source, accumTexture, material);
62. Graphics.Blit (accumTexture, destination);
63. }
64. }

* 3D\_Client\Assets\Script\framework\BasePanel.cs

1. using UnityEngine;
2. public class BasePanel : MonoBehaviour {
3. //皮肤路径
4. public string skinPath;
5. //皮肤
6. public GameObject skin;
7. //层级
8. public PanelManager.Layer layer = PanelManager.Layer.Panel;
9. //初始化
10. public void Init(){
11. //皮肤
12. GameObject skinPrefab = ResManager.LoadPrefab(skinPath);
13. skin = (GameObject)Instantiate(skinPrefab);
14. }
15. //关闭
16. public void Close(){
17. string name = this.GetType().ToString();
18. PanelManager.Close(name);
19. }
20. //初始化时
21. public virtual void OnInit(){
22. }
23. //显示时
24. public virtual void OnShow(params object[] para){
25. }
26. //关闭时
27. public virtual void OnClose(){
28. }
29. }

* 3D\_Client\Assets\Script\framework\ByteArray.cs

1. using System;
2. public class ByteArray  {
3. //默认大小
4. const int DEFAULT\_SIZE = 1024;
5. //初始大小
6. int initSize = 0;
7. //缓冲区
8. public byte[] bytes;
9. //读写位置
10. public int readIdx = 0;
11. public int writeIdx = 0;
12. //容量
13. private int capacity = 0;
14. //剩余空间
15. public int remain { get { return capacity-writeIdx; }}
16. //数据长度
17. public int length { get { return writeIdx-readIdx; }}
18. //构造函数
19. public ByteArray(int size = DEFAULT\_SIZE){
20. bytes = new byte[size];
21. capacity = size;
22. initSize = size;
23. readIdx = 0;
24. writeIdx = 0;
25. }
26. //构造函数
27. public ByteArray(byte[] defaultBytes){
28. bytes = defaultBytes;
29. capacity = defaultBytes.Length;
30. initSize = defaultBytes.Length;
31. readIdx = 0;
32. writeIdx = defaultBytes.Length;
33. }
34. //重设尺寸
35. public void ReSize(int size){
36. if(size < length) return;
37. if(size < initSize) return;
38. int n = 1;
39. while(n<size) n\*=2;
40. capacity = n;
41. byte[] newBytes = new byte[capacity];
42. Array.Copy(bytes, readIdx, newBytes, 0, writeIdx-readIdx);
43. bytes = newBytes;
44. writeIdx = length;
45. readIdx = 0;
46. }
47. //写入数据
48. public int Write(byte[] bs, int offset, int count){
49. if(remain < count){
50. ReSize(length + count);
51. }
52. Array.Copy(bs, offset, bytes, writeIdx, count);
53. writeIdx+=count;
54. return count;
55. }
56. //读取数据
57. public int Read(byte[] bs, int offset, int count){
58. count = Math.Min(count, length);
59. Array.Copy(bytes, 0, bs, offset, count);
60. readIdx+=count;
61. CheckAndMoveBytes();
62. return count;
63. }
64. //检查并移动数据
65. public void CheckAndMoveBytes(){
66. if(length < 8){
67. MoveBytes();
68. }
69. }
70. //移动数据
71. public void MoveBytes(){
72. Array.Copy(bytes, readIdx, bytes, 0, length);
73. writeIdx = length;
74. readIdx = 0;
75. }
76. //读取Int16
77. public Int16 ReadInt16(){
78. if(length < 2) return 0;
79. Int16 ret = BitConverter.ToInt16(bytes, readIdx);
80. readIdx += 2;
81. CheckAndMoveBytes();
82. return ret;
83. }
84. //读取Int32
85. public Int32 ReadInt32(){
86. if(length < 4) return 0;
87. Int32 ret = BitConverter.ToInt32(bytes, readIdx);
88. readIdx += 4;
89. CheckAndMoveBytes();
90. return ret;
91. }
93. //打印缓冲区
94. public override string ToString(){
95. return BitConverter.ToString(bytes, readIdx, length);
96. }
97. //打印调试信息
98. public string Debug(){
99. return string.Format("readIdx({0}) writeIdx({1}) bytes({2})",
100. readIdx,
101. writeIdx,
102. BitConverter.ToString(bytes, 0, capacity)
103. );
104. }
105. }

* 3D\_Client\Assets\Script\framework\MsgBase.cs

1. using System;
2. using UnityEngine;
3. public class MsgBase{
4. public string protoName = "null";
5. //编码
6. public static byte[] Encode(MsgBase msgBase){
7. string s = JsonUtility.ToJson(msgBase);
8. return System.Text.Encoding.UTF8.GetBytes(s);
9. }
10. //解码
11. public static MsgBase Decode(string protoName, byte[] bytes, int offset, int count){
12. string s = System.Text.Encoding.UTF8.GetString(bytes, offset, count);
13. Debug.Log("Debug decode:" + s);
14. MsgBase msgBase = (MsgBase)JsonUtility.FromJson(s, Type.GetType(protoName));
15. return msgBase;
16. }
17. //编码协议名（2字节长度+字符串）
18. public static byte[] EncodeName(MsgBase msgBase){
19. //名字bytes和长度
20. byte[] nameBytes = System.Text.Encoding.UTF8.GetBytes(msgBase.protoName);
21. Int16 len = (Int16)nameBytes.Length;
22. //申请bytes数值
23. byte[] bytes = new byte[2+len];
24. //组装2字节的长度信息
25. bytes[0] = (byte)(len%256);
26. bytes[1] = (byte)(len/256);
27. //组装名字bytes
28. Array.Copy(nameBytes, 0, bytes, 2, len);
29. return bytes;
30. }
31. //解码协议名（2字节长度+字符串）
32. public static string DecodeName(byte[] bytes, int offset, out int count){
33. count = 0;
34. //必须大于2字节
35. if(offset + 2 > bytes.Length){
36. return "";
37. }
38. //读取长度
39. Int16 len = (Int16)((bytes[offset+1] << 8 )| bytes[offset] );
40. //长度必须足够
41. if(offset + 2 + len > bytes.Length){
42. return "";
43. }
44. //解析
45. count = 2+len;
46. string name = System.Text.Encoding.UTF8.GetString(bytes, offset+2, len);
47. return name;
48. }
49. }

* 3D\_Client\Assets\Script\framework\NetManager.cs

1. using System.Collections.Generic;
2. using UnityEngine;
3. using System.Net.Sockets;
4. using System;
5. using System.Linq;
6. public static class NetManager {
7. //定义套接字
8. static Socket socket;
9. //接收缓冲区
10. static ByteArray readBuff;
11. //写入队列
12. static Queue<ByteArray> writeQueue;
13. //是否正在连接
14. static bool isConnecting = false;
15. //是否正在关闭
16. static bool isClosing = false;
17. //消息列表
18. static List<MsgBase> msgList = new List<MsgBase>();
19. //消息列表长度
20. static int msgCount = 0;
21. //每一次Update处理的消息量
22. readonly static int MAX\_MESSAGE\_FIRE = 10;
23. //是否启用心跳
24. public static bool isUsePing = true;
25. //心跳间隔时间
26. public static int pingInterval = 30;
27. //上一次发送PING的时间
28. static float lastPingTime = 0;
29. //上一次收到PONG的时间
30. static float lastPongTime = 0;
31. //事件
32. public enum NetEvent
33. {
34. ConnectSucc = 1,
35. ConnectFail = 2,
36. Close = 3,
37. }
38. //事件委托类型
39. public delegate void EventListener(String err);
40. //事件监听列表
41. private static Dictionary<NetEvent, EventListener> eventListeners = new Dictionary<NetEvent, EventListener>();
42. //添加事件监听
43. public static void AddEventListener(NetEvent netEvent, EventListener listener){
44. //添加事件
45. if (eventListeners.ContainsKey(netEvent)){
46. eventListeners[netEvent] += listener;
47. }
48. //新增事件
49. else{
50. eventListeners[netEvent] = listener;
51. }
52. }
53. //删除事件监听
54. public static void RemoveEventListener(NetEvent netEvent, EventListener listener){
55. if (eventListeners.ContainsKey(netEvent)){
56. eventListeners[netEvent] -= listener;
57. }
58. }
59. //分发事件
60. private static void FireEvent(NetEvent netEvent, String err){
61. if(eventListeners.ContainsKey(netEvent)){
62. eventListeners[netEvent](err);
63. }
64. }
65. //消息委托类型
66. public delegate void MsgListener(MsgBase msgBase);
67. //消息监听列表
68. private static Dictionary<string, MsgListener> msgListeners = new Dictionary<string, MsgListener>();
69. //添加消息监听
70. public static void AddMsgListener(string msgName, MsgListener listener){
71. //添加
72. if (msgListeners.ContainsKey(msgName)){
73. msgListeners[msgName] += listener;
74. }
75. //新增
76. else{
77. msgListeners[msgName] = listener;
78. }
79. }
80. //删除消息监听
81. public static void RemoveMsgListener(string msgName, MsgListener listener){
82. if (msgListeners.ContainsKey(msgName)){
83. msgListeners[msgName] -= listener;
84. }
85. }
86. //分发消息
87. private static void FireMsg(string msgName, MsgBase msgBase){
88. if(msgListeners.ContainsKey(msgName)){
89. msgListeners[msgName](msgBase);
90. }
91. }
92. //连接
93. public static void Connect(string ip, int port)
94. {
95. //状态判断
96. if(socket!=null && socket.Connected){
97. Debug.Log("Connect fail, already connected!");
98. return;
99. }
100. if(isConnecting){
101. Debug.Log("Connect fail, isConnecting");
102. return;
103. }
104. //初始化成员
105. InitState();
106. //参数设置
107. socket.NoDelay = true;
108. //Connect
109. isConnecting = true;
110. socket.BeginConnect(ip, port, ConnectCallback, socket);
111. }
112. //初始化状态
113. private static void InitState(){
114. //Socket
115. socket = new Socket(AddressFamily.InterNetwork,
116. SocketType.Stream, ProtocolType.Tcp);
117. //接收缓冲区
118. readBuff = new ByteArray();
119. //写入队列
120. writeQueue = new Queue<ByteArray>();
121. //是否正在连接
122. isConnecting = false;
123. //是否正在关闭
124. isClosing = false;
125. //消息列表
126. msgList = new List<MsgBase>();
127. //消息列表长度
128. msgCount = 0;
129. //上一次发送PING的时间
130. lastPingTime = Time.time;
131. //上一次收到PONG的时间
132. lastPongTime = Time.time;
133. //监听PONG协议
134. if(!msgListeners.ContainsKey("MsgPong")){
135. AddMsgListener("MsgPong", OnMsgPong);
136. }
137. }
138. //Connect回调
139. private static void ConnectCallback(IAsyncResult ar){
140. try{
141. Socket socket = (Socket) ar.AsyncState;
142. socket.EndConnect(ar);
143. Debug.Log("Socket Connect Succ ");
144. FireEvent(NetEvent.ConnectSucc,"");
145. isConnecting = false;
146. //开始接收
147. socket.BeginReceive( readBuff.bytes, readBuff.writeIdx,
148. readBuff.remain, 0, ReceiveCallback, socket);
149. }
150. catch (SocketException ex){
151. Debug.Log("Socket Connect fail " + ex.ToString());
152. FireEvent(NetEvent.ConnectFail, ex.ToString());
153. isConnecting = false;
154. }
155. }
156. //关闭连接
157. public static void Close(){
158. //状态判断
159. if(socket==null || !socket.Connected){
160. return;
161. }
162. if(isConnecting){
163. return;
164. }
165. //还有数据在发送
166. if(writeQueue.Count > 0){
167. isClosing = true;
168. }
169. //没有数据在发送
170. else{
171. socket.Close();
172. FireEvent(NetEvent.Close, "");
173. }
174. }
175. //发送数据
176. public static void Send(MsgBase msg) {
177. //状态判断
178. if(socket==null || !socket.Connected){
179. return;
180. }
181. if(isConnecting){
182. return;
183. }
184. if(isClosing){
185. return;
186. }
187. //数据编码
188. byte[] nameBytes = MsgBase.EncodeName(msg);
189. byte[] bodyBytes = MsgBase.Encode(msg);
190. int len = nameBytes.Length + bodyBytes.Length;
191. byte[] sendBytes = new byte[2+len];
192. //组装长度
193. sendBytes[0] = (byte)(len%256);
194. sendBytes[1] = (byte)(len/256);
195. //组装名字
196. Array.Copy(nameBytes, 0, sendBytes, 2, nameBytes.Length);
197. //组装消息体
198. Array.Copy(bodyBytes, 0, sendBytes, 2+nameBytes.Length, bodyBytes.Length);
199. //写入队列
200. ByteArray ba = new ByteArray(sendBytes);
201. int count = 0;  //writeQueue的长度
202. lock(writeQueue){
203. writeQueue.Enqueue(ba);
204. count = writeQueue.Count;
205. }
206. //send
207. if(count == 1){
208. socket.BeginSend(sendBytes, 0, sendBytes.Length,
209. 0, SendCallback, socket);
210. }
211. }
212. //Send回调
213. public static void SendCallback(IAsyncResult ar){
214. //获取state、EndSend的处理
215. Socket socket = (Socket) ar.AsyncState;
216. //状态判断
217. if(socket == null || !socket.Connected){
218. return;
219. }
220. //EndSend
221. int count = socket.EndSend(ar);
222. //获取写入队列第一条数据
223. ByteArray ba;
224. lock(writeQueue){
225. ba = writeQueue.First();
226. }
227. //完整发送
228. ba.readIdx+=count;
229. if(ba.length == 0){
230. lock(writeQueue){
231. writeQueue.Dequeue();
232. ba = writeQueue.First();
233. }
234. }
235. //继续发送
236. if(ba != null){
237. socket.BeginSend(ba.bytes, ba.readIdx, ba.length,
238. 0, SendCallback, socket);
239. }
240. //正在关闭
241. else if(isClosing) {
242. socket.Close();
243. }
244. }
246. //Receive回调
247. public static void ReceiveCallback(IAsyncResult ar){
248. try {
249. Socket socket = (Socket) ar.AsyncState;
250. //获取接收数据长度
251. int count = socket.EndReceive(ar);
252. readBuff.writeIdx+=count;
253. //处理二进制消息
254. OnReceiveData();
255. //继续接收数据
256. if(readBuff.remain < 8){
257. readBuff.MoveBytes();
258. readBuff.ReSize(readBuff.length\*2);
259. }
260. socket.BeginReceive( readBuff.bytes, readBuff.writeIdx,
261. readBuff.remain, 0, ReceiveCallback, socket);
262. }
263. catch (SocketException ex){
264. Debug.Log("Socket Receive fail" + ex.ToString());
265. }
266. }
267. //数据处理
268. public static void OnReceiveData(){
269. //消息长度
270. if(readBuff.length <= 2) {
271. return;
272. }
273. //获取消息体长度
274. int readIdx = readBuff.readIdx;
275. byte[] bytes =readBuff.bytes;
276. Int16 bodyLength = (Int16)((bytes[readIdx+1] << 8 )| bytes[readIdx]);
277. if(readBuff.length < bodyLength)
278. return;
279. readBuff.readIdx+=2;
280. //解析协议名
281. int nameCount = 0;
282. string protoName = MsgBase.DecodeName(readBuff.bytes, readBuff.readIdx, out nameCount);
283. if(protoName == ""){
284. Debug.Log("OnReceiveData MsgBase.DecodeName fail");
285. return;
286. }
287. readBuff.readIdx += nameCount;
288. //解析协议体
289. int bodyCount = bodyLength - nameCount;
290. MsgBase msgBase = MsgBase.Decode(protoName, readBuff.bytes, readBuff.readIdx, bodyCount);
291. readBuff.readIdx += bodyCount;
292. readBuff.CheckAndMoveBytes();
293. //添加到消息队列
294. lock(msgList){
295. msgList.Add(msgBase);
296. msgCount++;
297. }
298. //继续读取消息
299. if(readBuff.length > 2){
300. OnReceiveData();
301. }
302. }
303. //Update
304. public static void Update(){
305. MsgUpdate();
306. PingUpdate();
307. }
308. //更新消息
309. public static void MsgUpdate(){
310. //初步判断，提升效率
311. if(msgCount == 0){
312. return;
313. }
314. //重复处理消息
315. for(int i = 0; i< MAX\_MESSAGE\_FIRE; i++){
316. //获取第一条消息
317. MsgBase msgBase = null;
318. lock(msgList){
319. if(msgList.Count > 0){
320. msgBase = msgList[0];
321. msgList.RemoveAt(0);
322. msgCount--;
323. }
324. }
325. //分发消息
326. if(msgBase != null){
327. FireMsg(msgBase.protoName, msgBase);
328. }
329. //没有消息了
330. else{
331. break;
332. }
333. }
334. }
335. //发送PING协议
336. private static void PingUpdate(){
337. //是否启用
338. if(!isUsePing){
339. return;
340. }
341. //发送PING
342. if(Time.time - lastPingTime > pingInterval){
343. MsgPing msgPing = new MsgPing();
344. Send(msgPing);
345. lastPingTime = Time.time;
346. }
347. //检测PONG时间
348. if(Time.time - lastPongTime > pingInterval\*4){
349. Close();
350. }
351. }
352. //监听PONG协议
353. private static void OnMsgPong(MsgBase msgBase){
354. lastPongTime = Time.time;
355. }
356. }

* 3D\_Client\Assets\Script\framework\PanelManager.cs

1. using UnityEngine;
2. using System.Collections.Generic;
3. public static class PanelManager{
4. //Layer
5. public enum Layer{
6. Panel,
7. Tip,
8. }
9. //层级列表
10. private static Dictionary<Layer, Transform> layers = new Dictionary<Layer, Transform>();
11. //面板列表
12. public static Dictionary<string, BasePanel> panels = new Dictionary<string, BasePanel>();
13. //结构
14. public static Transform root;
15. public static Transform canvas;
16. //初始化
17. public static void Init(){
18. root = GameObject.Find("Root").transform;
19. canvas = root.Find("Canvas");
20. Transform panel = canvas.Find("Panel");
21. Transform tip = canvas.Find("Tip");
22. layers.Add(Layer.Panel, panel);
23. layers.Add(Layer.Tip, tip);
24. }
25. //打开面板
26. public static void Open<T>(params object[] para) where T:BasePanel{
27. //已经打开
28. string name = typeof(T).ToString();
29. if (panels.ContainsKey(name)){
30. return;
31. }
32. //组件
33. BasePanel panel = root.gameObject.AddComponent<T>();
34. panel.OnInit();
35. panel.Init();
36. //父容器
37. Transform layer = layers[panel.layer];
38. panel.skin.transform.SetParent(layer, false);
39. //列表
40. panels.Add(name, panel);
41. //OnShow
42. panel.OnShow(para);
43. }
44. //关闭面板
45. public static void Close(string name){
46. //没有打开
47. if(!panels.ContainsKey(name)){
48. return;
49. }
50. BasePanel panel = panels[name];
51. //OnClose
52. panel.OnClose();
53. //列表
54. panels.Remove(name);
55. //销毁
56. GameObject.Destroy(panel.skin);
57. Component.Destroy(panel);
58. }
59. }

* 3D\_Client\Assets\Script\framework\ResManager.cs

1. using UnityEngine;
2. public class ResManager : MonoBehaviour {
3. //加载预设
4. public static GameObject LoadPrefab(string path){
5. return Resources.Load<GameObject>(path);
6. }
7. }

* 3D\_Client\Assets\Script\mudule\Battle\AimPanel.cs

1. using UnityEngine;
2. using UnityEngine.UI;
3. public class AimPanel : BasePanel {
4. CtrlTank tank;
5. private Image aimImage;
6. //初始化
7. public override void OnInit() {
8. skinPath = "AimPanel";
9. layer = PanelManager.Layer.Panel;
10. }
11. //显示
12. public override void OnShow(params object[] args) {
13. //寻找组件
14. aimImage = skin.transform.Find("Image").GetComponent<Image>();
15. tank = (CtrlTank)BattleManager.GetCtrlTank();
16. }
18. //关闭
19. public override void OnClose() {
20. }
21. //当按下确定按钮
22. public void Update(){
23. if(tank == null){
24. return;
25. }
26. //3D坐标
27. Vector3 point = tank.ForecastExplodePoint();
28. //屏幕坐标
29. Vector3 screenPoint = Camera.main.WorldToScreenPoint(point);
30. //UI坐标
31. aimImage.transform.position = screenPoint;
32. }
33. }

* 3D\_Client\Assets\Script\mudule\Battle\BaseTank.cs

1. using UnityEngine;
2. public class BaseTank : MonoBehaviour {
3. //坦克模型
4. private GameObject skin;
5. //转向速度
6. public float steer = 30;
7. //移动速度
8. public float speed = 6f;
9. //炮塔旋转速度
10. public float turretSpeed = 30f;
11. //炮塔
12. public Transform turret;
13. //炮管
14. public Transform gun;
15. //发射点
16. public Transform firePoint;
17. //炮弹Cd时间
18. public float fireCd = 0.5f;
19. //上一次发射炮弹的时间
20. public float lastFireTime = 0;
21. //物理
22. protected Rigidbody rigidBody;
23. //生命值
24. public float hp = 100;
25. //属于哪一名玩家
26. public string id = "";
27. //阵营
28. public int camp = 0;
29. //炮管旋转
30. public float minGunAngle = -20;
31. public float maxGunAngle = 20;
32. public float gunSpeed = 4f;
33. //轮子和履带
34. public Transform wheels;
35. public Transform track;
36. // Use this for initialization
37. public void Start () {
38. }
39. //初始化
40. public virtual void Init(string skinPath){
41. //皮肤
42. GameObject skinRes = ResManager.LoadPrefab(skinPath);
43. skin = (GameObject)Instantiate(skinRes);
44. skin.transform.parent = this.transform;
45. skin.transform.localPosition = Vector3.zero;
46. skin.transform.localEulerAngles = Vector3.zero;
47. //物理
48. rigidBody = gameObject.AddComponent<Rigidbody>();
49. BoxCollider boxCollider = gameObject.AddComponent<BoxCollider>();
50. boxCollider.center = new Vector3(0, 2.5f, 1.47f);
51. boxCollider.size = new Vector3(7, 5, 12);
52. //炮塔炮管
53. turret = skin.transform.Find("Turret");
54. gun = turret.transform.Find("Gun");
55. firePoint = gun.transform.Find("FirePoint");
56. //轮子履带
57. wheels = skin.transform.Find("Wheels");
58. track = skin.transform.Find("Track");
59. }
60. //发射炮弹
61. public Bullet Fire(){
62. //已经死亡
63. if(IsDie()){
64. return null;
65. }
66. //产生炮弹
67. GameObject bulletObj = new GameObject("bullet");
68. bulletObj.layer = LayerMask.NameToLayer("Bullet");
69. Bullet bullet = bulletObj.AddComponent<Bullet>();
70. bullet.Init();
71. bullet.tank = this;
72. //位置
73. bullet.transform.position = firePoint.position;
74. bullet.transform.rotation = firePoint.rotation;
75. //更新时间
76. lastFireTime = Time.time;
77. return bullet;
78. }
79. //是否死亡
80. public bool IsDie(){
81. return hp <= 0;
82. }
83. //被攻击
84. public void Attacked(float att){
85. //已经死亡
86. if(IsDie()){
87. return;
88. }
89. //扣血
90. hp -= att;
91. //死亡
92. if(IsDie()){
93. //显示焚烧效果
94. GameObject obj = ResManager.LoadPrefab("explosion");
95. GameObject explosion = Instantiate(obj, transform.position, transform.rotation);
96. explosion.transform.SetParent(transform);
97. }
98. }
99. // Update is called once per frame
100. public void Update () {
101. }
102. //轮子旋转，履带滚动
103. public void WheelUpdate(float axis){
104. //计算速度
105. float v = Time.deltaTime\*speed\*axis\*100;
106. //旋转每个轮子
107. foreach (Transform wheel in wheels){
108. wheel.Rotate(new Vector3(v,0,0),Space.Self);
109. }
110. //滚动履带
111. MeshRenderer mr = track.gameObject.GetComponent<MeshRenderer>();
112. if (mr == null) {
113. return;
114. };
115. Material mtl = mr.material;
116. mtl.mainTextureOffset += new Vector2(0, v/256);
117. }
118. }

* 3D\_Client\Assets\Script\mudule\Battle\BattleManager.cs

1. using System.Collections.Generic;
2. using UnityEngine;
3. public class BattleManager {
4. //战场中的坦克
5. public static Dictionary<string, BaseTank> tanks = new Dictionary<string, BaseTank>();
6. //初始化
7. public static void Init() {
8. //添加监听
9. NetManager.AddMsgListener("MsgEnterBattle", OnMsgEnterBattle);
10. NetManager.AddMsgListener("MsgBattleResult", OnMsgBattleResult);
11. NetManager.AddMsgListener("MsgLeaveBattle", OnMsgLeaveBattle);
12. NetManager.AddMsgListener("MsgSyncTank", OnMsgSyncTank);
13. NetManager.AddMsgListener("MsgFire", OnMsgFire);
14. NetManager.AddMsgListener("MsgHit", OnMsgHit);
15. }
16. //添加坦克
17. public static void AddTank(string id, BaseTank tank){
18. tanks[id] = tank;
19. }
20. //删除坦克
21. public static void RemoveTank(string id){
22. tanks.Remove(id);
23. }
24. //获取坦克
25. public static BaseTank GetTank(string id) {
26. if(tanks.ContainsKey(id)){
27. return tanks[id];
28. }
29. return null;
30. }
31. //获取玩家控制的坦克
32. public static BaseTank GetCtrlTank() {
33. return GetTank(GameMain.id);
34. }
35. //重置战场
36. public static void Reset() {
37. //场景
38. foreach(BaseTank tank in tanks.Values){
39. MonoBehaviour.Destroy(tank.gameObject);
40. }
41. //列表
42. tanks.Clear();
43. }
44. //开始战斗
45. public static void EnterBattle(MsgEnterBattle msg) {
46. //重置
47. BattleManager.Reset();
48. //关闭界面
49. PanelManager.Close("RoomPanel");//可以放到房间系统的监听中
50. PanelManager.Close("ResultPanel");
51. PanelManager.Close("KillPanel");
52. PanelManager.Close("BattlePanel");
53. PanelManager.Close("AimPanel");
54. //产生坦克
55. for(int i=0; i<msg.tanks.Length; i++){
56. GenerateTank(msg.tanks[i]);
57. }
58. //打开界面
59. PanelManager.Open<BattlePanel>();
60. PanelManager.Open<AimPanel>();
61. }
62. //产生坦克
63. public static void GenerateTank(TankInfo tankInfo){
64. //GameObject
65. string objName = "Tank\_" + tankInfo.id;
66. GameObject tankObj = new GameObject(objName);
67. //AddComponent
68. BaseTank tank = null;
69. if(tankInfo.id == GameMain.id) {
70. tank = tankObj.AddComponent<CtrlTank>();
71. }
72. else {
73. tank = tankObj.AddComponent<SyncTank>();
74. }
75. //camera
76. if(tankInfo.id == GameMain.id) {
77. CameraFollow cf = tankObj.AddComponent<CameraFollow>();
78. }
79. //属性
80. tank.camp = tankInfo.camp;
81. tank.id = tankInfo.id;
82. tank.hp = tankInfo.hp;
83. //pos rotation
84. Vector3 pos = new Vector3(tankInfo.x, tankInfo.y, tankInfo.z);
85. Vector3 rot = new Vector3(tankInfo.ex, tankInfo.ey, tankInfo.ez);
86. tank.transform.position = pos;
87. tank.transform.eulerAngles = rot;
88. //init
89. if(tankInfo.camp == 1){
90. tank.Init("tankPrefab");
91. }
92. else{
93. tank.Init("tankPrefab2");
94. }
95. //列表
96. AddTank(tankInfo.id, tank);
97. }
98. //收到进入战斗协议
99. public static void OnMsgEnterBattle(MsgBase msgBase){
100. MsgEnterBattle msg = (MsgEnterBattle)msgBase;
101. EnterBattle(msg);
102. }
103. //收到战斗结束协议
104. public static void OnMsgBattleResult(MsgBase msgBase){
105. MsgBattleResult msg = (MsgBattleResult)msgBase;
106. //判断显示胜利还是失败
107. bool isWin = false;
108. BaseTank tank = GetCtrlTank();
109. if(tank!= null && tank.camp == msg.winCamp){
110. isWin = true;
111. }
112. //显示界面
113. PanelManager.Open<ResultPanel>(isWin);
114. //关闭界面
115. PanelManager.Close("AimPanel");
116. }
117. //收到玩家退出协议
118. public static void OnMsgLeaveBattle(MsgBase msgBase){
119. MsgLeaveBattle msg = (MsgLeaveBattle)msgBase;
120. //查找坦克
121. BaseTank tank = GetTank(msg.id);
122. if(tank == null){
123. return;
124. }
125. //删除坦克
126. RemoveTank(msg.id);
127. MonoBehaviour.Destroy(tank.gameObject);
128. }
129. //收到同步协议
130. public static void OnMsgSyncTank(MsgBase msgBase){
131. MsgSyncTank msg = (MsgSyncTank)msgBase;
132. //不同步自己
133. if(msg.id == GameMain.id){
134. return;
135. }
136. //查找坦克
137. SyncTank tank = (SyncTank)GetTank(msg.id);
138. if(tank == null){
139. return;
140. }
141. //移动同步
142. tank.SyncPos(msg);
143. }
144. //收到开火协议
145. public static void OnMsgFire(MsgBase msgBase){
146. MsgFire msg = (MsgFire)msgBase;
147. //不同步自己
148. if(msg.id == GameMain.id){
149. return;
150. }
151. //查找坦克
152. SyncTank tank = (SyncTank)GetTank(msg.id);
153. if(tank == null){
154. return;
155. }
156. //开火
157. tank.SyncFire(msg);
158. }
159. //收到击中协议
160. public static void OnMsgHit(MsgBase msgBase){
161. MsgHit msg = (MsgHit)msgBase;
162. //查找坦克
163. BaseTank tank = GetTank(msg.targetId);
164. if(tank == null){
165. return;
166. }
167. bool isDie = tank.IsDie();
168. //被击中
169. tank.Attacked(msg.damage);
170. //击杀提示
171. if(!isDie && tank.IsDie() && msg.id == GameMain.id){
172. PanelManager.Open<KillPanel>();
173. }
174. }
175. }

* 3D\_Client\Assets\Script\mudule\Battle\BattlePanel.cs

1. using UnityEngine;
2. using UnityEngine.UI;
3. public class BattlePanel : BasePanel {
4. //hp
5. private Image hpFill;
6. private Text hpText;
7. //info
8. private Text camp1Text;
9. private Text camp2Text;
10. //初始化
11. public override void OnInit() {
12. skinPath = "BattlePanel";
13. layer = PanelManager.Layer.Panel;
14. }
15. //显示
16. public override void OnShow(params object[] args) {
17. //寻找组件
18. hpFill = skin.transform.Find("HpBar/Fill").GetComponent<Image>();
19. hpText = skin.transform.Find("HpBar/HpText").GetComponent<Text>();
20. camp1Text = skin.transform.Find("CampInfo/Camp1Text").GetComponent<Text>();
21. camp2Text = skin.transform.Find("CampInfo/Camp2Text").GetComponent<Text>();
22. ReflashCampInfo();
23. NetManager.AddMsgListener("MsgLeaveBattle", OnMsgLeaveBattle);
24. NetManager.AddMsgListener("MsgHit", OnMsgHit);
25. BaseTank tank = BattleManager.GetCtrlTank();
26. if(tank != null){
27. ReflashHp(Mathf.CeilToInt(tank.hp));
28. }
29. }
30. //更新信息
31. private void ReflashCampInfo(){
32. int count1 = 0;
33. int count2 = 0;
34. foreach(BaseTank tank in BattleManager.tanks.Values){
35. if(tank.IsDie()){
36. continue;
37. }
38. if(tank.camp == 1){count1++;};
39. if(tank.camp == 2){count2++;};
40. }
41. camp1Text.text = "红:" + count1.ToString();
42. camp2Text.text = count2.ToString()+":蓝";
43. }
44. //更新hp
45. private void ReflashHp(int hp){
46. if(hp < 0){hp=0;}
47. hpFill.fillAmount = hp/100f;
48. hpText.text = "hp:" + hp;
49. }
50. //关闭
51. public override void OnClose() {
52. NetManager.RemoveMsgListener("MsgLeaveBattle", OnMsgLeaveBattle);
53. NetManager.RemoveMsgListener("MsgHit", OnMsgHit);
54. }
55. //收到玩家退出协议
56. public void OnMsgLeaveBattle(MsgBase msgBase){
57. ReflashCampInfo();
58. }
59. //收到击中协议
60. public void OnMsgHit(MsgBase msgBase){
61. MsgHit msg = (MsgHit)msgBase;
62. if(msg.targetId == GameMain.id){
63. BaseTank tank = BattleManager.GetCtrlTank();
64. if(tank != null){
65. ReflashHp(Mathf.CeilToInt(tank.hp));
66. }
67. }
68. ReflashCampInfo();
69. }
70. }

* 3D\_Client\Assets\Script\mudule\Battle\Bullet.cs

1. using UnityEngine;
2. public class Bullet : MonoBehaviour {
3. //移动速度
4. public float speed = 220f;
5. //发射者
6. public BaseTank tank;
7. //炮弹模型
8. private GameObject skin;
9. //物理
10. Rigidbody rigidBody;
11. //初始化
12. public void Init(){
13. //皮肤
14. GameObject skinRes = ResManager.LoadPrefab("bulletPrefab");
15. skin = (GameObject)Instantiate(skinRes);
16. skin.transform.parent = this.transform;
17. skin.transform.localPosition = Vector3.zero;
18. skin.transform.localEulerAngles = Vector3.zero;
19. //物理
20. rigidBody = gameObject.AddComponent<Rigidbody>();
21. rigidBody.useGravity = false;
22. }
23. // Update is called once per frame
24. void Update () {
25. //向前移动
26. transform.position += transform.forward \* speed \* Time.deltaTime;
27. }
28. //碰撞
29. void OnCollisionEnter(Collision collisionInfo) {
30. //打到的坦克
31. GameObject collObj = collisionInfo.gameObject;
32. BaseTank hitTank = collObj.GetComponent<BaseTank>();
33. //不能打自己
34. if(hitTank == tank){
35. return;
36. }
37. //打到其他坦克
38. if(hitTank != null){
39. SendMsgHit(tank, hitTank);
40. }
41. //显示爆炸效果
42. GameObject explode = ResManager.LoadPrefab("fire");
43. Instantiate(explode, transform.position, transform.rotation);
44. //摧毁自身
45. Destroy(gameObject);
46. }
47. //发送伤害协议
48. void SendMsgHit(BaseTank tank, BaseTank hitTank){
49. if(hitTank == null || tank == null){
50. return;
51. }
52. //不是自己发出的炮弹
53. if(tank.id != GameMain.id){
54. return;
55. }
56. MsgHit msg = new MsgHit();
57. msg.targetId = hitTank.id;
58. msg.id = tank.id;
59. msg.x = transform.position.x;
60. msg.y = transform.position.y;
61. msg.z = transform.position.z;
62. NetManager.Send(msg);
63. }
64. }

* 3D\_Client\Assets\Script\mudule\Battle\CameraFollow.cs

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. public class CameraFollow : MonoBehaviour {
5. //距离矢量
6. public Vector3 distance = new Vector3(0, 15, -22);//x代表角度
7. //相机
8. public Camera camera;
9. //偏移值
10. public Vector3 offset = new Vector3(0, 8f, 0);
11. //相机移动速度
12. public float speed = 6f;
13. //最大和最小距离
14. public float minDistanceZ = -35f;
15. public float maxDistanceZ = -10f;
16. //距离变化速度
17. public float zoomSpeed = 2f;
18. // Use this for initialization
19. void Start () {
20. //默认为主相机
21. camera = Camera.main;
22. //相机初始位置
23. Vector3 pos = transform.position;
24. Vector3 forward = transform.forward;
25. Vector3 initPos = pos - 30\*forward + Vector3.up\*10;
26. camera.transform.position = initPos;
27. }
28. //调整距离
29. void Zoom(){
30. float axis = Input.GetAxis("Mouse ScrollWheel");
31. distance.z += axis\*zoomSpeed;
32. distance.z = Mathf.Clamp(distance.z, minDistanceZ, maxDistanceZ);
33. }
34. //调整角度
35. void Rotate(){
36. if(!Input.GetMouseButton(1)){//右键
37. return;
38. }
39. float axis = Input.GetAxis("Mouse X");
40. distance.x += 2\*axis;
41. distance.x = Mathf.Clamp(distance.x, -20, 20);
42. }
43. //所有组件update之后发生
44. void LateUpdate () {
45. //坦克位置
46. Vector3 pos = transform.position;
47. //坦克方向
48. Vector3 forward = transform.forward;
49. Vector3 rigiht = transform.right;
50. //相机目标位置
51. Vector3 targetPos = pos;
52. targetPos = pos + forward\*distance.z + rigiht\*distance.x;
53. targetPos.y += distance.y;
54. //相机位置
55. Vector3 cameraPos = camera.transform.position;
56. cameraPos = Vector3.MoveTowards(cameraPos, targetPos,Time.deltaTime\*speed);
57. camera.transform.position = cameraPos;
58. //对准坦克
59. Camera.main.transform.LookAt(pos + offset);
60. //调整距离
61. Zoom();
62. //调整角度
63. Rotate();
64. }
65. }

* 3D\_Client\Assets\Script\mudule\Battle\CtrlTank.cs

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. public class CtrlTank : BaseTank {
5. //上一次发送同步信息的时间
6. private float lastSendSyncTime = 0;
7. //同步帧率
8. public static float syncInterval = 0.05f;
9. //自动瞄准的坦克
10. public BaseTank aimTank;
11. private float lastSearchTime;           //上一次搜索时间
12. private float searchInterval = 1f;      //搜索时间间隔
13. private float maxSearchDistance = 150f; //最大搜索距离
14. private float preventSearchTime = 0;    //禁止自动瞄准到某个时间
15. private float preventInterval = 2f;     //手动操作炮塔后的保护时间
16. new void Update(){
17. base.Update();
18. //移动控制
19. MoveUpdate();
20. //炮塔控制
21. TurretUpdate();
22. //开炮
23. FireUpdate();
24. //发送同步信息
25. SyncUpdate();
26. //自动搜寻目标
27. SearchUpdate();
28. //自动瞄准
29. AutoAimUpdate();
30. }
31. //移动控制
32. public void MoveUpdate(){
33. //已经死亡
34. if(IsDie()){
35. return;
36. }
37. //旋转
38. float x = Input.GetAxis("Horizontal");
39. transform.Rotate(0, x \* steer \* Time.deltaTime, 0);
40. //前进后退
41. float y = Input.GetAxis("Vertical");
42. Vector3 s = y\*transform.forward \* speed \* Time.deltaTime;
43. transform.transform.position += s;
44. //轮子旋转，履带滚动
45. WheelUpdate(y);
46. }
47. //炮塔控制
48. public void TurretUpdate(){
49. //已经死亡
50. if(IsDie()){
51. return;
52. }
53. //或者轴向
54. int axis = 0;
55. if(Input.GetKey(KeyCode.Q)){
56. axis = -1;
57. }
58. else if(Input.GetKey(KeyCode.E)){
59. axis = 1;
60. }
61. if(axis == 0){
62. return;
63. }
64. //旋转角度
65. Vector3 le = turret.localEulerAngles;
66. le.y += axis\*Time.deltaTime\*turretSpeed;
67. turret.localEulerAngles = le;
68. //保护时间
69. preventSearchTime = Time.time + preventInterval;
70. aimTank = null;
71. }
72. //开炮
73. public void FireUpdate(){
74. //已经死亡
75. if(IsDie()){
76. return;
77. }
78. //按键判断
79. if(!Input.GetKey(KeyCode.Space) && !Input.GetMouseButton(0)){
80. return;
81. }
82. //cd是否判断
83. if(Time.time - lastFireTime < fireCd){
84. return;
85. }
86. //发射
87. Bullet bullet = Fire();
88. //发送同步协议
89. MsgFire msg = new MsgFire();
90. msg.x = bullet.transform.position.x;
91. msg.y = bullet.transform.position.y;
92. msg.z = bullet.transform.position.z;
93. msg.ex = bullet.transform.eulerAngles.x;
94. msg.ey = bullet.transform.eulerAngles.y;
95. msg.ez = bullet.transform.eulerAngles.z;
96. NetManager.Send(msg);
97. }
98. //发送同步信息
99. public void SyncUpdate(){
100. //时间间隔判断
101. if(Time.time - lastSendSyncTime < syncInterval){
102. return;
103. }
104. lastSendSyncTime = Time.time;
105. //发送同步协议
106. MsgSyncTank msg = new MsgSyncTank();
107. msg.x = transform.position.x;
108. msg.y = transform.position.y;
109. msg.z = transform.position.z;
110. msg.ex = transform.eulerAngles.x;
111. msg.ey = transform.eulerAngles.y;
112. msg.ez = transform.eulerAngles.z;
113. msg.turretY = turret.localEulerAngles.y;
114. msg.gunX = gun.localEulerAngles.x;
115. NetManager.Send(msg);
116. }
117. //计算爆炸位置
118. public Vector3 ForecastExplodePoint(){
119. //碰撞信息和碰撞点
120. Vector3 hitPoint = Vector3.zero;
121. RaycastHit hit;
122. //沿着炮管方向的射线
123. Vector3 pos = firePoint.position;
124. Ray ray = new Ray(pos, firePoint.forward);
125. //射线检测
126. int layerMask = ~(1 << LayerMask.NameToLayer("Bullet"));
127. if (Physics.Raycast(ray, out hit, 200.0f,  layerMask)){
128. hitPoint = hit.point;
129. }
130. else {
131. hitPoint = ray.GetPoint(200);
132. }
133. return hitPoint;
134. }
135. //搜寻自动瞄准目标
136. public void SearchUpdate(){
137. //时间间隔判断
138. if(Time.time - lastSearchTime < searchInterval){
139. return;
140. }
141. lastSearchTime = Time.time;
142. //搜索
143. aimTank = null;
144. foreach(BaseTank tank in BattleManager.tanks.Values){
145. //同个阵营
146. if(tank.camp == camp){
147. continue;
148. }
149. //自己
150. if(tank == this){
151. continue;
152. }
153. //已经死亡
154. if(tank.IsDie()){
155. continue;
156. }
157. //相对位置（z）
158. Vector3 p = firePoint.InverseTransformPoint(tank.transform.position);
159. if(p.z <=0 || p.z > maxSearchDistance){
160. continue;
161. }
162. //相对位置，45°角度限制
163. if(Mathf.Abs(p.x) > p.z){
164. continue;
165. }
166. //是否切换目标
167. if(aimTank != null){
168. float d1 = Vector3.Distance(tank.transform.position,    transform.position);
169. float d2 = Vector3.Distance(aimTank.transform.position, transform.position);
170. if(d1 > d2){
171. continue;
172. }
173. }
174. aimTank = tank;
175. }
176. }
177. //自动旋转炮管炮塔
178. public void AutoAimUpdate(){
179. //保护时间
180. if(Time.time < preventSearchTime){
181. return;
182. }
183. Vector3 p;
184. if(aimTank == null){
185. //回正
186. p = firePoint.InverseTransformPoint(transform.position + transform.forward\*100 + transform.up\*5);
187. }
188. else{
189. //相对位置
190. p = firePoint.InverseTransformPoint(aimTank.transform.position + new Vector3(0,5f,0));
191. }
192. //旋转炮塔
193. float axis = Mathf.Clamp(p.x, -1, 1);
194. Vector3 le = turret.localEulerAngles;
195. le.y += axis\*Time.deltaTime\*turretSpeed;
196. turret.localEulerAngles = le;
197. //旋转炮管
198. axis = Mathf.Clamp(p.y, -1, 1);
199. le = gun.localEulerAngles;
200. le.x -= axis\*Time.deltaTime\*gunSpeed;
201. if(le.x > 180) {le.x=360-le.x;}
202. le.x = Mathf.Clamp(le.x, minGunAngle, maxGunAngle);
203. gun.localEulerAngles = le;
204. }
205. }

* 3D\_Client\Assets\Script\mudule\Battle\KillPanel.cs

1. using UnityEngine;
2. public class KillPanel : BasePanel {
3. //界面开始显示的时间
4. private float startTime = 0;
5. //初始化
6. public override void OnInit() {
7. skinPath = "KillPanel";
8. layer = PanelManager.Layer.Tip;
9. }
10. //显示
11. public override void OnShow(params object[] args) {
12. startTime = Time.time;
13. }
15. //关闭
16. public override void OnClose() {
17. }
18. //当按下确定按钮
19. public void Update(){
20. if(Time.time - startTime > 2f){
21. Close();
22. }
23. }
24. }

* 3D\_Client\Assets\Script\mudule\Battle\ResultPanel.cs

1. using UnityEngine.UI;
2. public class ResultPanel : BasePanel {
3. //胜利提示图片
4. private Image winImage;
5. //失败提示图片
6. private Image lostImage;
7. //确定按钮
8. private Button okBtn;
9. //初始化
10. public override void OnInit() {
11. skinPath = "ResultPanel";
12. layer = PanelManager.Layer.Tip;
13. }
14. //显示
15. public override void OnShow(params object[] args) {
16. //寻找组件
17. winImage = skin.transform.Find("WinImage").GetComponent<Image>();
18. lostImage = skin.transform.Find("LostImage").GetComponent<Image>();
19. okBtn = skin.transform.Find("OkBtn").GetComponent<Button>();
20. //监听
21. okBtn.onClick.AddListener(OnOkClick);
22. //显示哪个图片
23. if(args.Length == 1){
24. bool isWIn = (bool)args[0];
25. if(isWIn){
26. winImage.gameObject.SetActive(true);
27. lostImage.gameObject.SetActive(false);
28. }else{
29. winImage.gameObject.SetActive(false);
30. lostImage.gameObject.SetActive(true);
31. }
32. }
33. }
35. //关闭
36. public override void OnClose() {
37. }
38. //当按下确定按钮
39. public void OnOkClick(){
40. PanelManager.Open<RoomPanel>();
41. Close();
42. }
43. }

* 3D\_Client\Assets\Script\mudule\Battle\SyncTank.cs

1. using UnityEngine;
2. public class SyncTank : BaseTank {
3. //预测信息，哪个时间到达哪个位置
4. private Vector3 lastPos;
5. private Vector3 lastRot;
6. private Vector3 forecastPos;
7. private Vector3 forecastRot;
8. private float lastTurretY;
9. private float forecastTurretY;
10. private float lastGunX;
11. private float forecastGunX;
12. private float forecastTime;
13. //重写Init
14. public override void Init(string skinPath){
15. base.Init(skinPath);
16. //不受物理运动影响
17. rigidBody.constraints = RigidbodyConstraints.FreezeAll;
18. rigidBody.useGravity = false;
19. //初始化预测信息
20. lastPos = transform.position;
21. lastRot = transform.eulerAngles;
22. forecastPos = transform.position;
23. forecastRot = transform.eulerAngles;
24. lastTurretY = turret.localEulerAngles.y;
25. forecastTurretY = turret.localEulerAngles.y;
26. lastGunX = gun.localEulerAngles.x;
27. forecastGunX = gun.localEulerAngles.x;
28. forecastTime = Time.time;
29. }
30. new void Update(){
31. base.Update();
32. //更新位置
33. ForecastUpdate();
34. }
35. //移动同步
36. public void SyncPos(MsgSyncTank msg){
37. //预测位置
38. Vector3 pos = new Vector3(msg.x, msg.y, msg.z);
39. Vector3 rot = new Vector3(msg.ex, msg.ey, msg.ez);
40. //forecastPos = pos + 2\*(pos - lastPos);
41. //forecastRot = rot + 2\*(rot - lastRot);
42. forecastPos = pos;  //跟随不预测
43. forecastRot = rot;
44. forecastTurretY = msg.turretY;
45. forecastGunX = msg.gunX;
46. //更新
47. lastPos = pos;
48. lastRot = rot;
49. lastTurretY = turret.localEulerAngles.y;
50. lastGunX = turret.localEulerAngles.x;
51. forecastTime = Time.time;
52. }
53. //更新位置
54. public void ForecastUpdate(){
55. //时间
56. float t =  (Time.time - forecastTime)/CtrlTank.syncInterval;
57. t = Mathf.Clamp(t, 0f, 1f);
58. //位置
59. Vector3 pos = transform.position;
60. pos = Vector3.Lerp(pos, forecastPos, t);
61. transform.position = pos;
62. //旋转
63. Quaternion quat = transform.rotation;
64. Quaternion forcastQuat = Quaternion.Euler(forecastRot);
65. quat = Quaternion.Lerp(quat, forcastQuat, t) ;
66. transform.rotation = quat;
67. //轮子旋转，履带滚动
68. float axis = transform.InverseTransformPoint(forecastPos).z;
69. axis = Mathf.Clamp(axis\*1024, -1f, 1f);
70. WheelUpdate(axis);
71. //炮管
72. Vector3 le = turret.localEulerAngles;
73. le.y = Mathf.LerpAngle(le.y, forecastTurretY, t);
74. turret.localEulerAngles = le;
75. //炮塔
76. le = gun.localEulerAngles;
77. le.x = Mathf.LerpAngle(le.x, forecastGunX, t);
78. gun.localEulerAngles = le;
79. }
80. //开火
81. public void SyncFire(MsgFire msg){
82. Bullet bullet = Fire();
83. //更新坐标
84. Vector3 pos = new Vector3(msg.x, msg.y, msg.z);
85. Vector3 rot = new Vector3(msg.ex, msg.ey, msg.ez);
86. bullet.transform.position = pos;
87. bullet.transform.eulerAngles = rot;
88. }
89. }

* 3D\_Client\Assets\Script\mudule\Common\TipPanel.cs

1. using UnityEngine.UI;
2. public class TipPanel : BasePanel {
3. //提示文本
4. private Text text;
5. //确定按钮
6. private Button okBtn;
7. //初始化
8. public override void OnInit() {
9. skinPath = "TipPanel";
10. layer = PanelManager.Layer.Tip;
11. }
12. //显示
13. public override void OnShow(params object[] args) {
14. //寻找组件
15. text = skin.transform.Find("Text").GetComponent<Text>();
16. okBtn = skin.transform.Find("OkBtn").GetComponent<Button>();
17. //监听
18. okBtn.onClick.AddListener(OnOkClick);
19. //提示语
20. if(args.Length == 1){
21. text.text = (string)args[0];
22. }
23. }
25. //关闭
26. public override void OnClose() {
27. }
28. //当按下确定按钮
29. public void OnOkClick(){
30. Close();
31. }
32. }

* 3D\_Client\Assets\Script\mudule\Login\LoginPanel.cs

1. using UnityEngine;
2. using UnityEngine.UI;
3. public class LoginPanel : BasePanel {
4. //账号输入框
5. private InputField idInput;
6. //密码输入框
7. private InputField pwInput;
8. //登陆按钮
9. private Button loginBtn;
10. //注册按钮
11. private Button regBtn;
12. //背景图
13. private Image bgImage;
14. //开始显示的时间
15. private float startTime = float.MaxValue;
16. //显示连接失败
17. private bool showConnFail = false;
18. //ip和地址
19. private string ip = "coco56.top";
20. private int port = 82;
21. //初始化
22. public override void OnInit() {
23. skinPath = "LoginPanel";
24. layer = PanelManager.Layer.Panel;
25. }
26. //显示
27. public override void OnShow(params object[] args) {
28. //寻找组件
29. idInput = skin.transform.Find("IdInput").GetComponent<InputField>();
30. pwInput = skin.transform.Find("PwInput").GetComponent<InputField>();
31. loginBtn = skin.transform.Find("LoginBtn").GetComponent<Button>();
32. regBtn = skin.transform.Find("RegisterBtn").GetComponent<Button>();
33. bgImage = skin.transform.Find("BgImage").GetComponent<Image>();
34. //监听
35. loginBtn.onClick.AddListener(OnLoginClick);
36. regBtn.onClick.AddListener(OnRegClick);
37. //网络协议监听
38. NetManager.AddMsgListener("MsgLogin", OnMsgLogin);
39. //网络事件监听
40. NetManager.AddEventListener(NetManager.NetEvent.ConnectSucc, OnConnectSucc);
41. NetManager.AddEventListener(NetManager.NetEvent.ConnectFail, OnConnectFail);
42. //连接服务器
43. NetManager.Connect(ip, port);
44. //记录时间
45. startTime = Time.time;
46. }
47. //关闭
48. public override void OnClose() {
49. //网络协议监听
50. NetManager.RemoveMsgListener("MsgLogin", OnMsgLogin);
51. //网络事件监听
52. NetManager.RemoveEventListener(NetManager.NetEvent.ConnectSucc, OnConnectSucc);
53. NetManager.RemoveEventListener(NetManager.NetEvent.ConnectFail, OnConnectFail);
54. }
55. //连接成功回调
56. void OnConnectSucc(string err){
57. Debug.Log("OnConnectSucc");
58. }
59. //连接失败回调
60. void OnConnectFail(string err){
61. showConnFail = true;
62. //PanelManager.Open<TipPanel>(err);
63. }
64. //当按下注册按钮
65. public void OnRegClick() {
66. PanelManager.Open<RegisterPanel>();
67. }
69. //当按下登陆按钮
70. public void OnLoginClick() {
71. //用户名密码为空
72. if (idInput.text == "" || pwInput.text == "") {
73. PanelManager.Open<TipPanel>("用户名和密码不能为空");
74. return;
75. }
76. //发送
77. MsgLogin msgLogin = new MsgLogin();
78. msgLogin.id = idInput.text;
79. msgLogin.pw = pwInput.text;
80. NetManager.Send(msgLogin);
81. }
82. //收到登陆协议
83. public void OnMsgLogin (MsgBase msgBase) {
84. MsgLogin msg = (MsgLogin)msgBase;
85. if(msg.result == 0){
86. Debug.Log("登陆成功");
87. //设置id
88. GameMain.id = msg.id;
89. //打开房间列表界面
90. PanelManager.Open<RoomListPanel>();
91. //关闭界面
92. Close();
93. }
94. else{
95. PanelManager.Open<TipPanel>("登陆失败");
96. }
97. }
98. //update
99. public void Update(){
100. //背景图动画效果
101. float w = Mathf.Ceil(Time.time\*2) % 10 == 0 ?500f:0.1f;//频率
102. float a = 1+ 0.1f - 0.1f\*Mathf.Sin(w\*Time.time);    //振幅
103. bgImage.transform.localScale = new Vector3(a,a,1);
104. //连接失败
105. if(showConnFail){
106. showConnFail = false;
107. PanelManager.Open<TipPanel>("网络连接失败，请重新打开游戏");
108. }
109. }
110. }

* 3D\_Client\Assets\Script\mudule\Login\RegisterPanel.cs

1. using UnityEngine;
2. using UnityEngine.UI;
3. public class RegisterPanel : BasePanel {
4. //账号输入框
5. private InputField idInput;
6. //密码输入框
7. private InputField pwInput;
8. //重复输入框
9. private InputField repInput;
10. //注册按钮
11. private Button regBtn;
12. //关闭按钮
13. private Button closeBtn;
14. //初始化
15. public override void OnInit() {
16. skinPath = "RegisterPanel";
17. layer = PanelManager.Layer.Panel;
18. }
19. //显示
20. public override void OnShow(params object[] args) {
21. //寻找组件
22. idInput = skin.transform.Find("IdInput").GetComponent<InputField>();
23. pwInput = skin.transform.Find("PwInput").GetComponent<InputField>();
24. repInput = skin.transform.Find("RepInput").GetComponent<InputField>();
25. regBtn = skin.transform.Find("RegisterBtn").GetComponent<Button>();
26. closeBtn = skin.transform.Find("CloseBtn").GetComponent<Button>();
27. //监听
28. regBtn.onClick.AddListener(OnRegClick);
29. closeBtn.onClick.AddListener(OnCloseClick);
30. //网络协议监听
31. NetManager.AddMsgListener("MsgRegister", OnMsgRegister);
32. }
33. //关闭
34. public override void OnClose() {
35. //网络协议监听
36. NetManager.RemoveMsgListener("MsgRegister", OnMsgRegister);
37. }
38. //当按下注册按钮
39. public void OnRegClick() {
40. //用户名密码为空
41. if (idInput.text == "" || pwInput.text == "") {
42. PanelManager.Open<TipPanel>("用户名和密码不能为空");
43. return;
44. }
45. //两次密码不同
46. if (repInput.text != pwInput.text) {
47. PanelManager.Open<TipPanel>("两次输入的密码不同");
48. return;
49. }
50. //发送
51. MsgRegister msgReg = new MsgRegister();
52. msgReg.id = idInput.text;
53. msgReg.pw = pwInput.text;
54. NetManager.Send(msgReg);
55. }
56. //收到注册协议
57. public void OnMsgRegister (MsgBase msgBase) {
58. MsgRegister msg = (MsgRegister)msgBase;
59. if(msg.result == 0){
60. Debug.Log("注册成功");
61. //提示
62. PanelManager.Open<TipPanel>("注册成功");
63. //关闭界面
64. Close();
65. }
66. else{
67. PanelManager.Open<TipPanel>("注册失败");
68. }
69. }
70. //当按下关闭按钮
71. public void OnCloseClick() {
72. Close();
73. }
74. }

* 3D\_Client\Assets\Script\mudule\Room\RoomListPanel.cs

1. using UnityEngine;
2. using UnityEngine;
3. using UnityEngine.UI;
4. public class RoomListPanel : BasePanel {
5. //账号文本
6. private Text idText;
7. //战绩文本
8. private Text scoreText;
9. //创建房间按钮
10. private Button createButton;
11. //刷新列表按钮
12. private Button reflashButton;
13. //列表容器
14. private Transform content;
15. //房间物体
16. private GameObject roomObj;
17. //坦克视图
18. private GameObject tankCamera;
19. private GameObject tankObj;
20. //初始化
21. public override void OnInit() {
22. skinPath = "RoomListPanel";
23. layer = PanelManager.Layer.Panel;
24. }
25. //显示
26. public override void OnShow(params object[] args) {
27. //寻找组件
28. idText = skin.transform.Find("InfoPanel/IdText").GetComponent<Text>();
29. scoreText = skin.transform.Find("InfoPanel/ScoreText").GetComponent<Text>();
30. createButton = skin.transform.Find("CtrlPanel/CreateButton").GetComponent<Button>();
31. reflashButton = skin.transform.Find("CtrlPanel/ReflashButton").GetComponent<Button>();
32. content = skin.transform.Find("ListPanel/Scroll View/Viewport/Content");
33. roomObj = skin.transform.Find("Room").gameObject;
34. tankCamera = skin.transform.Find("InfoPanel/TankCamera").gameObject;
35. //不激活房间
36. roomObj.SetActive(false);
37. //显示id
38. idText.text = GameMain.id;
39. //按钮事件
40. createButton.onClick.AddListener(OnCreateClick);
41. reflashButton.onClick.AddListener(OnReflashClick);
42. //协议监听
43. NetManager.AddMsgListener("MsgGetAchieve", OnMsgGetAchieve);
44. NetManager.AddMsgListener("MsgGetRoomList", OnMsgGetRoomList);
45. NetManager.AddMsgListener("MsgCreateRoom", OnMsgCreateRoom);
46. NetManager.AddMsgListener("MsgEnterRoom", OnMsgEnterRoom);
47. //发送查询
48. MsgGetAchieve msgGetAchieve = new MsgGetAchieve();
49. NetManager.Send(msgGetAchieve);
50. MsgGetRoomList msgGetRoomList = new MsgGetRoomList();
51. NetManager.Send(msgGetRoomList);
52. //创建坦克模型
53. GameObject tankSkin = ResManager.LoadPrefab("tankPrefab");
54. tankObj = (GameObject)Instantiate(tankSkin, tankCamera.transform);
55. tankObj.transform.localPosition = new Vector3(0, -2, 25);
56. tankObj.transform.Rotate(0,90,-30);
57. }
58. //关闭
59. public override void OnClose() {
60. //协议监听
61. NetManager.RemoveMsgListener("MsgGetAchieve", OnMsgGetAchieve);
62. NetManager.RemoveMsgListener("MsgGetRoomList", OnMsgGetRoomList);
63. NetManager.RemoveMsgListener("MsgCreateRoom", OnMsgCreateRoom);
64. NetManager.RemoveMsgListener("MsgEnterRoom", OnMsgEnterRoom);
65. }
66. //收到成绩查询协议
67. public void OnMsgGetAchieve (MsgBase msgBase) {
68. MsgGetAchieve msg = (MsgGetAchieve)msgBase;
69. scoreText.text = msg.win + "胜 " + msg.lost + "负";
70. }
71. //收到房间列表协议
72. public void OnMsgGetRoomList (MsgBase msgBase) {
73. MsgGetRoomList msg = (MsgGetRoomList)msgBase;
74. //清除房间列表
75. for(int i = content.childCount-1; i >= 0 ; i--){
76. GameObject o = content.GetChild(i).gameObject;
77. Destroy(o);
78. }
79. //重新生成列表
80. if(msg.rooms == null){
81. return;
82. }
83. for(int i = 0; i < msg.rooms.Length; i++){
84. GenerateRoom(msg.rooms[i]);
85. }
86. }
87. //创建一个房间单元
88. public void GenerateRoom(RoomInfo roomInfo){
89. //创建物体
90. GameObject o = Instantiate(roomObj);
91. o.transform.SetParent(content);
92. o.SetActive(true);
93. o.transform.localScale = Vector3.one;
94. //获取组件
95. Transform trans = o.transform;
96. Text idText = trans.Find("IdText").GetComponent<Text>();
97. Text countText = trans.Find("CountText").GetComponent<Text>();
98. Text statusText = trans.Find("StatusText").GetComponent<Text>();
99. Button btn = trans.Find("JoinButton").GetComponent<Button>();
100. //填充信息
101. idText.text = roomInfo.id.ToString();
102. countText.text = roomInfo.count.ToString();
103. if(roomInfo.status == 0){
104. statusText.text = "准备中";
105. }
106. else{
107. statusText.text = "战斗中";
108. }
109. //按钮事件
110. btn.name = idText.text;
111. btn.onClick.AddListener(delegate(){
112. OnJoinClick(btn.name);
113. });
114. }
115. //点击刷新按钮
116. public void OnReflashClick(){
117. MsgGetRoomList msg = new MsgGetRoomList();
118. NetManager.Send(msg);
119. }
120. //点击加入房间按钮
121. public void OnJoinClick(string idString) {
122. MsgEnterRoom msg = new MsgEnterRoom();
123. msg.id = int.Parse(idString);
124. NetManager.Send(msg);
125. }
126. //收到进入房间协议
127. public void OnMsgEnterRoom (MsgBase msgBase) {
128. MsgEnterRoom msg = (MsgEnterRoom)msgBase;
129. //成功进入房间
130. if(msg.result == 0){
131. PanelManager.Open<RoomPanel>();
132. Close();
133. }
134. //进入房间失败
135. else{
136. PanelManager.Open<TipPanel>("进入房间失败");
137. }
138. }
139. //点击新建房间按钮
140. public void OnCreateClick() {
141. MsgCreateRoom msg = new MsgCreateRoom();
142. NetManager.Send(msg);
143. }
144. //收到新建房间协议
145. public void OnMsgCreateRoom (MsgBase msgBase) {
146. MsgCreateRoom msg = (MsgCreateRoom)msgBase;
147. //成功创建房间
148. if(msg.result == 0){
149. //PanelManager.Open<TipPanel>("创建成功");
150. PanelManager.Open<RoomPanel>();
151. Close();
152. }
153. //创建房间失败
154. else{
155. PanelManager.Open<TipPanel>("创建房间失败");
156. }
157. }
158. public void Update(){
159. //更新坦克视图
160. tankObj.transform.Rotate(0, Time.deltaTime\*2f, 0);
161. }
162. }

* 3D\_Client\Assets\Script\mudule\Room\RoomPanel.cs

1. using UnityEngine;
2. using UnityEngine.UI;
3. public class RoomPanel : BasePanel {
4. //开战按钮
5. private Button startButton;
6. //退出按钮
7. private Button closeButton;
8. //列表容器
9. private Transform content1;//蓝色阵营
10. private Transform content2;//红色阵营
11. //玩家信息物体
12. private GameObject playerObj;
13. //初始化
14. public override void OnInit() {
15. skinPath = "RoomPanel";
16. layer = PanelManager.Layer.Panel;
17. }
18. //显示
19. public override void OnShow(params object[] args) {
20. //寻找组件
21. startButton = skin.transform.Find("CtrlPanel/StartButton").GetComponent<Button>();
22. closeButton = skin.transform.Find("CtrlPanel/CloseButton").GetComponent<Button>();
23. content1 = skin.transform.Find("ListPanel/Scroll View1/Viewport/Content");
24. content2 = skin.transform.Find("ListPanel/Scroll View2/Viewport/Content");
25. playerObj =  skin.transform.Find("Player").gameObject;
26. //不激活玩家信息
27. playerObj.SetActive(false);
28. //按钮事件
29. startButton.onClick.AddListener(OnStartClick);
30. closeButton.onClick.AddListener(OnCloseClick);
31. //协议监听
32. NetManager.AddMsgListener("MsgGetRoomInfo", OnMsgGetRoomInfo);
33. NetManager.AddMsgListener("MsgLeaveRoom", OnMsgLeaveRoom);
34. NetManager.AddMsgListener("MsgStartBattle", OnMsgStartBattle);
35. //发送查询
36. MsgGetRoomInfo msg = new MsgGetRoomInfo();
37. NetManager.Send(msg);
38. }
39. //关闭
40. public override void OnClose() {
41. //协议监听
42. NetManager.RemoveMsgListener("MsgGetRoomInfo", OnMsgGetRoomInfo);
43. NetManager.RemoveMsgListener("MsgLeaveRoom", OnMsgLeaveRoom);
44. NetManager.RemoveMsgListener("MsgStartBattle", OnMsgStartBattle);
45. }
46. //收到玩家列表协议
47. public void OnMsgGetRoomInfo (MsgBase msgBase) {
48. MsgGetRoomInfo msg = (MsgGetRoomInfo)msgBase;
49. //清除玩家列表
50. for(int i = content1.childCount-1; i >= 0 ; i--){
51. GameObject o = content1.GetChild(i).gameObject;
52. Destroy(o);
53. }
54. for(int i = content2.childCount-1; i >= 0 ; i--){
55. GameObject o = content2.GetChild(i).gameObject;
56. Destroy(o);
57. }
58. //重新生成列表
59. if(msg.players == null){
60. return;
61. }
62. for(int i = 0; i < msg.players.Length; i++){
63. GeneratePlayerInfo(msg.players[i]);
64. }
65. }
66. //创建一个玩家信息单元
67. public void GeneratePlayerInfo(PlayerInfo playerInfo){
68. //创建物体
69. GameObject o = Instantiate(playerObj);
70. o.SetActive(true);
71. o.transform.localScale = Vector3.one;
72. //设置阵营 1-红 2-蓝
73. if(playerInfo.camp == 1){
74. o.transform.SetParent(content2);
75. }
76. else {
77. o.transform.SetParent(content1);
78. }
79. //获取组件
80. Transform trans = o.transform;
81. Text idText = trans.Find("IdText").GetComponent<Text>();
82. Image ownerImage = trans.Find("OwnerImage").GetComponent<Image>();
83. Text scoreText = trans.Find("ScoreText").GetComponent<Text>();
84. //填充信息
85. idText.text = playerInfo.id;
86. if(playerInfo.isOwner == 1){
87. ownerImage.gameObject.SetActive(true);
88. }
89. else{
90. ownerImage.gameObject.SetActive(false);
91. }
92. scoreText.text = playerInfo.win + "胜 " + playerInfo.lost + "负";
93. }
94. //点击退出按钮
95. public void OnCloseClick(){
96. MsgLeaveRoom msg = new MsgLeaveRoom();
97. NetManager.Send(msg);
98. }
99. //收到退出房间协议
100. public void OnMsgLeaveRoom (MsgBase msgBase) {
101. MsgLeaveRoom msg = (MsgLeaveRoom)msgBase;
102. //成功退出房间
103. if(msg.result == 0){
104. //PanelManager.Open<TipPanel>("退出房间");
105. PanelManager.Open<RoomListPanel>();
106. Close();
107. }
108. //退出房间失败
109. else{
110. PanelManager.Open<TipPanel>("退出房间失败");
111. }
112. }
113. //点击开战按钮
114. public void OnStartClick(){
115. MsgStartBattle msg = new MsgStartBattle();
116. NetManager.Send(msg);
117. }
118. //收到开战返回
119. public void OnMsgStartBattle (MsgBase msgBase) {
120. MsgStartBattle msg = (MsgStartBattle)msgBase;
121. //开战
122. if(msg.result == 0){
123. //关闭界面
124. Close();
125. }
126. //开战失败
127. else{
128. PanelManager.Open<TipPanel>("开战失败！两队至少都需要一名玩家，只有队长可以开始战斗！");
129. }
130. }
131. }

* 3D\_Client\Assets\Script\mudule\proto\BattleMsg.cs

1. //坦克信息
2. [System.Serializable]
3. public class TankInfo{
4. public string id = "";  //玩家id
5. public int camp = 0;    //阵营
6. public int hp = 0;      //生命值
7. public float x = 0;     //位置
8. public float y = 0;
9. public float z = 0;
10. public float ex = 0;    //旋转
11. public float ey = 0;
12. public float ez = 0;
13. }
14. //进入战场（服务端推送）
15. public class MsgEnterBattle:MsgBase {
16. public MsgEnterBattle() {protoName = "MsgEnterBattle";}
17. //服务端回
18. public TankInfo[] tanks;
19. public int mapId = 1;   //地图，只有一张
20. }
21. //战斗结果（服务端推送）
22. public class MsgBattleResult:MsgBase {
23. public MsgBattleResult() {protoName = "MsgBattleResult";}
24. //服务端回
25. public int winCamp = 0;  //获胜的阵营
26. }
27. //玩家退出（服务端推送）
28. public class MsgLeaveBattle:MsgBase {
29. public MsgLeaveBattle() {protoName = "MsgLeaveBattle";}
30. //服务端回
31. public string id = "";  //玩家id
32. }

* 3D\_Client\Assets\Script\mudule\proto\LoginMsg.cs

1. //注册
2. public class MsgRegister:MsgBase {
3. public MsgRegister() {protoName = "MsgRegister";}
4. //客户端发
5. public string id = "";
6. public string pw = "";
7. //服务端回（0-成功，1-失败）
8. public int result = 0;
9. }
10. //登陆
11. public class MsgLogin:MsgBase {
12. public MsgLogin() {protoName = "MsgLogin";}
13. //客户端发
14. public string id = "";
15. public string pw = "";
16. //服务端回（0-成功，1-失败）
17. public int result = 0;
18. }
19. //踢下线（服务端推送）
20. public class MsgKick:MsgBase {
21. public MsgKick() {protoName = "MsgKick";}
22. //原因（0-其他人登陆同一账号）
23. public int reason = 0;
24. }

* 3D\_Client\Assets\Script\mudule\proto\NotepadMsg.cs

1. //获取记事本内容
2. public class MsgGetText:MsgBase {
3. public MsgGetText() {protoName = "MsgGetText";}
4. //服务端回
5. public string text = "";
6. }
7. //保存记事本内容
8. public class MsgSaveText:MsgBase {
9. public MsgSaveText() {protoName = "MsgSaveText";}
10. //客户端发
11. public string text = "";
12. //服务端回（0-成功 1-文字太长）
13. public int result = 0;
14. }

* 3D\_Client\Assets\Script\mudule\proto\RoomMsg.cs

1. //查询成绩
2. public class MsgGetAchieve:MsgBase {
3. public MsgGetAchieve() {protoName = "MsgGetAchieve";}
4. //服务端回
5. public int win = 0;
6. public int lost = 0;
7. }
8. //房间信息
9. [System.Serializable]
10. public class RoomInfo{
11. public int id = 0;      //房间id
12. public int count = 0;   //人数
13. public int status = 0;  //状态0-准备中 1-战斗中
14. }
15. //请求房间列表
16. public class MsgGetRoomList:MsgBase {
17. public MsgGetRoomList() {protoName = "MsgGetRoomList";}
18. //服务端回
19. public RoomInfo[] rooms;
20. }
21. //创建房间
22. public class MsgCreateRoom:MsgBase {
23. public MsgCreateRoom() {protoName = "MsgCreateRoom";}
24. //服务端回
25. public int result = 0;
26. }
27. //进入房间
28. public class MsgEnterRoom:MsgBase {
29. public MsgEnterRoom() {protoName = "MsgEnterRoom";}
30. //客户端发
31. public int id = 0;
32. //服务端回
33. public int result = 0;
34. }
35. //玩家信息
36. [System.Serializable]
37. public class PlayerInfo{
38. public string id = "lpy";   //账号
39. public int camp = 0;        //阵营
40. public int win = 0;         //胜利数
41. public int lost = 0;        //失败数
42. public int isOwner = 0;     //是否是房主
43. }
44. //获取房间信息
45. public class MsgGetRoomInfo:MsgBase {
46. public MsgGetRoomInfo() {protoName = "MsgGetRoomInfo";}
47. //服务端回
48. public PlayerInfo[] players;
49. }
50. //离开房间
51. public class MsgLeaveRoom:MsgBase {
52. public MsgLeaveRoom() {protoName = "MsgLeaveRoom";}
53. //服务端回
54. public int result = 0;
55. }
56. //开战
57. public class MsgStartBattle:MsgBase {
58. public MsgStartBattle() {protoName = "MsgStartBattle";}
59. //服务端回
60. public int result = 0;
61. }

* 3D\_Client\Assets\Script\mudule\proto\SyncMsg.cs

1. //同步坦克信息
2. public class MsgSyncTank:MsgBase {
3. public MsgSyncTank() {protoName = "MsgSyncTank";}
4. //位置、旋转、炮塔旋转
5. public float x = 0f;
6. public float y = 0f;
7. public float z = 0f;
8. public float ex = 0f;
9. public float ey = 0f;
10. public float ez = 0f;
11. public float turretY = 0f;
12. public float gunX = 0f;
13. //服务端补充
14. public string id = "";      //哪个坦克
15. }
16. //开火
17. public class MsgFire:MsgBase {
18. public MsgFire() {protoName = "MsgFire";}
19. //炮弹初始位置、旋转
20. public float x = 0f;
21. public float y = 0f;
22. public float z = 0f;
23. public float ex = 0f;
24. public float ey = 0f;
25. public float ez = 0f;
26. //服务端补充
27. public string id = "";      //哪个坦克
28. }
29. //击中
30. public class MsgHit:MsgBase {
31. public MsgHit() {protoName = "MsgHit";}
32. //击中谁
33. public string targetId = "";
34. //击中点
35. public float x = 0f;
36. public float y = 0f;
37. public float z = 0f;
38. //服务端补充
39. public string id = "";      //哪个坦克
40. public int hp = 0;          //被击中坦克血量
41. public int damage = 0;      //受到的伤害
42. }

* 3D\_Client\Assets\Script\mudule\proto\SysMsg.cs

1. public class MsgPing:MsgBase {
2. public MsgPing() {protoName = "MsgPing";}
3. }
4. public class MsgPong:MsgBase {
5. public MsgPong() {protoName = "MsgPong";}
6. }

* 3D\_Client\Assets\Script\GameMain.cs

1. using UnityEngine;
2. public class GameMain : MonoBehaviour {
3. public static string id = "";
4. // Use this for initialization
5. void Start () {
6. //网络监听
7. NetManager.AddEventListener(NetManager.NetEvent.Close, OnConnectClose);
8. NetManager.AddMsgListener("MsgKick", OnMsgKick);
9. //初始化
10. PanelManager.Init();
11. BattleManager.Init();
12. //打开登陆面板
13. PanelManager.Open<LoginPanel>();
14. /\*用于单机测试
15. GameMain.id = "cat";
16. TankInfo tankInfo = new TankInfo();
17. tankInfo.camp = 1;
18. tankInfo.id = GameMain.id;
19. tankInfo.hp = 30;
20. tankInfo.x = 262;
21. tankInfo.y = -8;
22. tankInfo.z = 342;
23. BattleManager.GenerateTank(tankInfo);
24. PanelManager.Open<BattlePanel>();
25. PanelManager.Open<AimPanel>();
26. TankInfo tankInfo2 = new TankInfo();
27. tankInfo2.camp = 2;
28. tankInfo2.id = "dog";
29. tankInfo2.hp = 100;
30. tankInfo2.z = 30;
31. tankInfo2.y = 5;
32. tankInfo2.ey = 130;
33. BattleManager.GenerateTank(tankInfo2);
34. \*/
35. }
36. // Update is called once per frame
37. void Update () {
38. NetManager.Update();
39. }
40. //关闭连接
41. void OnConnectClose(string err){
42. Debug.Log("断开连接");
43. }
44. //被踢下线
45. void OnMsgKick(MsgBase msgBase){
46. PanelManager.Open<TipPanel>("被踢下线");
47. }
48. }

课程设计评分表

课题名称： C#高级程序设计

|  |  |
| --- | --- |
| **项 目** | **评 价** |
| 系统功能及代码编写（权重40%） |  |
| 信息检索工具（权重30%） |  |
| 项目的测试（权重10%） |  |
| 项目的评价（权重10%） |  |
| 课程设计报告质量（权重10%） |  |
| 综合成绩 |  |

教师签名：

日 期：