《网络编程设计》大作业技术文档

题目： 网络游戏设计——松鼠大作战

班级： 1503 1504

姓名： 张祺琛 王林超

学号： **1030515332 1030515424**

目 录

[目 录](#_Toc22368)

[第1章 概述](#_Toc20761)

[1.1 类型](#_Toc13493)

[1.2 文档结构](#_Toc18232)

[1.3 使用平台与最低配置](#_Toc2200)

[1.4 游戏规则](#_Toc1772)

[1.5 目标用户](#_Toc32007)

[第2章 游戏的总体设计](#_Toc24100)

[第3章 游戏测试](#_Toc27763)

[第4章 各个技术实现](#_Toc30917)

[4.1.游戏大厅实现以及玩家昵称和颜色同步](#_Toc2430)

[4.1.1技术功能简介:](#_Toc25590)

[4.1.2实现方法及代码:](#_Toc26011)

[4.2.主游戏内玩家和松果的生成以及开启玩家移动](#_Toc18612)

[4.2.1技术功能简介:](#_Toc21951)

[4.2.2实现方法及代码:](#_Toc78)

[4.3.第三人称跟随视角的实现](#_Toc28033)

[4.3.1技术功能简介:](#_Toc31094)

[4.3.2实现方法及代码:](#_Toc19550)

[4.4.玩家信息同步](#_Toc23295)

[4.4.1技术功能简介:](#_Toc5630)

[4.4.2实现方法及代码:](#_Toc31118)

[4.5.客户端同步显示信息](#_Toc30117)

[4.5.1技术功能简介:](#_Toc8128)

[4.5.2实现方法及代码:](#_Toc20027)

[4.6.利用碰撞检测实现捡松果和玩家之间互相伤害](#_Toc25751)

[4.6.1技术功能简介:](#_Toc913)

[4.6.2实现方法及代码:](#_Toc27173)

[4.7.冬天环境系统](#_Toc28366)

[4.7.1技术功能简介:](#_Toc28376)

[4.7.2实现方法及代码:](#_Toc24350)

[致 谢](#_Toc11726)

第1章 概述

1.1 类型

App Store将.io游戏列为2017年度“值得关注的四个趋势”之一，让这个前两年凭借《球球大作战》、《贪吃蛇大作战》风靡一时的游戏类型，再度成为关注焦点。io类游戏发源于国外，以操作规则简单、多人在线对抗、死后即刻复活等为特点，由于在命名时多以“.io”结尾，久而久之也被人称为io类游戏，在国内又被称为休闲竞技类游戏、大作战游戏、大乱斗游戏，并且近年一直在不断的进化和扩展当中。

1.2 文档结构

本文档的内容安排如下：

第一章为概述部分，主要介绍了本游戏相关研究现状和主要研究目的，阐述了其内容以及意义．

第二章为应用的总体游戏设计，进行了本文档研究内容的需求分析和可行性分析，详细设计方面，介绍了应用的框架构成和功能、各模块的详细功能设计，并分析了涉及到的关键技术．

第三章为整个交互游戏中各模块的的功能及实现细节，针对主要功能的实现思路进行了详细说明，对几种计算机动画的编程思路和方法也进行了清晰阐述．

第四章是游戏测试部分，主要阐述了投影、交互、动画等部分的测试工作及具体测试结果．

第五章为总结，分析了游戏的创新点、对当前工作的总结以及对今后工作的展望．

1.3 使用平台与最低配置

运行系统Windows95/98/2k/XP

最低配置：

奔腾166，32MB内存

DierctX 7.0a或更新版本

1.4 游戏规则

每个玩家代表一只松鼠，其目标就是尽可能的收集能支撑自己度过寒冬的松果。

玩家（松鼠）在找到松果后，可以选着直接吃掉松果，也可以将其带回巢穴内。

松鼠在游戏中实时具有新陈代谢属性，即运动会消耗松鼠的能量，吃掉松果会增加松鼠的能量。当补充能量大于消耗能量时，松鼠体重会增加。反之体重会减小。体重小于下限时，玩家被判定死亡（失败）。

松鼠的能量代谢速度与体重成正比。

两个玩家（松鼠）在接近到一定距离后，会发生战斗，战斗失败者被驱离（可直接将失败的玩家放置到一定距离之外）。

玩家战斗获胜的几率，与体重成正比。

松鼠每次携带松果数量不超过4个。

每个玩家（松鼠）有一个巢穴，其内可存放任意多的松果。

没有松鼠看管的巢穴，可以被其他松鼠“洗劫”。（特别的，即使一个松鼠在看管自己的巢穴，也可能被其他松鼠通过战斗而驱离）

游戏中可以设置NPC松鼠，增加游戏可玩性。

游戏在开始一定时间后（比如10分钟），服务器宣布进入冬天，所有玩家不得离开巢穴，只能通过吃掉储存的松果来维持体重。

冬天持续一定时间（比如10分钟）后，存活下来的松鼠（玩家）为赢家，并根据体重排名。

1.5 目标用户

io游戏由于操作门槛低、风格轻松有趣，近年在国外一直属于颇受欢迎的类型；国内市场方面，2015年上线的《球球大作战》、2016年上线的《贪吃蛇大作战》则被看作是io游戏风潮较早的引领者。 《球球大作战》由巨人网络Superpop&Lollipop工作室自主研发，2015年5月上线，游戏规则简单概括起来就是“大球吃小球”：玩家需要操纵自己的球球去吃掉别人的球以不断长大，同时也要避免被别人吃掉。游戏上线至今，在玩法上已经发展出了自由对战、团战、生存、大逃杀、迷雾等多个模式。2017年3月的官方数据显示，该游戏全球用户已超过3亿；此外，其DAU达到2500万，MAU达到1亿。《球球大作战》还是当下最具人气的移动电竞游戏项目之一。2016年共举办了522场官方赛事，全年赛事累计观看人数超过3亿，全年直播时长超过10000分钟。

另一款现象级产品《贪吃蛇大作战》沿用了非智能机时代经典游戏贪吃蛇的基本玩法：玩家需要控制蛇通过不断吃到小豆子来长大，在这个过程中蛇头不能碰到身体，随着蛇身越来越长，游戏难度也不断增加。App Annie统计显示，该游戏自2016年8月在国内上线后，曾有长达2个月的时间持续盘踞在App Store 免费榜第一的位置。

不过数据同样显示，2017年以来，《球球大作战》和《贪吃蛇大作战》的免费榜排名均出现了下降。前者从今年8月开始，长期处在免费榜100-400名之间；后者从今年4月开始，在免费榜上也已掉出100名开外。

**第2章** **游戏的总体设计**

io 游戏的核心是什么？

多人即时对战，需要跟其他玩家产生竞技关系才有乐趣；

即开即玩，没有永久的成长系统，每局都是重新开始；

越成长越受限，等级越高，则在某一方面越会存在缺陷，比如高等级移动速度变缓慢；

简单的成长系统：1、获取资源；2、杀死对手。

io 游戏的优势

2 ~ 5 分钟之内结束一局游戏，等人等车、坐地铁等间歇杀时间利器；

操作简单，有竞技乐趣，入门易，玩好难，像 LOL、王者荣耀这些游戏一样需要「意识」；

无氪金，纯靠意识、技巧和运气取胜。

io+moba 3V3 团队作战，近身野蛮操作，加入很多 moba 元素：3V3 团战，躲草丛隐身，增加游戏的不可预知性和乐趣；

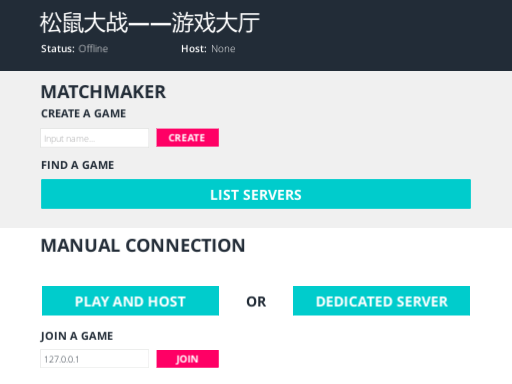
玩法核心：虚拟双摇杆，左侧控制人物移动，右侧控制攻击，人物每升一级可选择技能；

通过吃蘑菇和砍杀敌人获得成长，积分最高的玩家所在队伍获胜；

玩家死亡后可以立即复活，积分不清零，仅降低一部分；

游戏中标注队友以及大魔王（当前分数最高的玩家）的位置情况，因此「怼大魔王」成为抱团玩法的一部分，因此积分第一名玩家总在变动中。

1. **游戏测试**



目前测试了许多次，修复了若干错误后，局域网的游戏能保证顺利进行



玩家接的显示顺利并且能够互相造成伤害且延迟不高



在冬天蓝色的松鼠提前死亡，绿色松鼠收集到的较多所以依然存活，符合游戏规则。

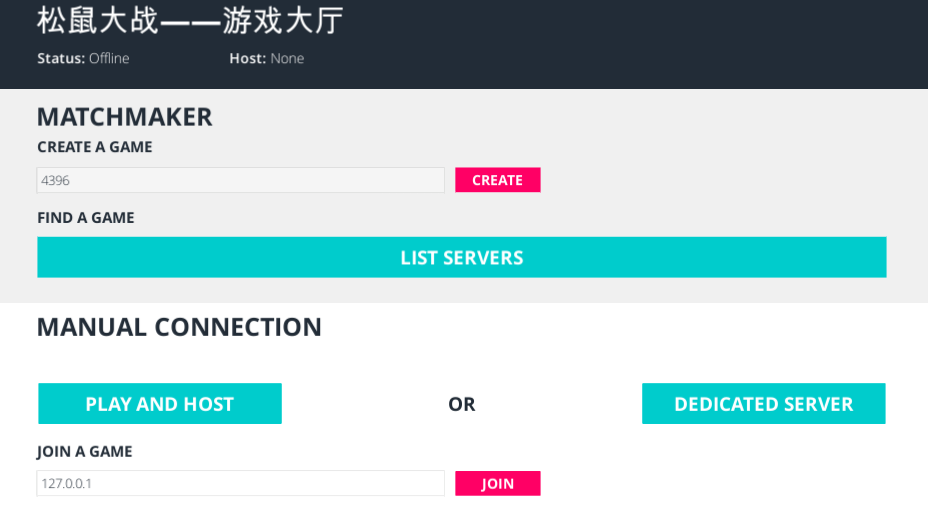
**第4章 各个技术实现**

**4.1.游戏大厅实现以及玩家昵称和颜色同步**

**4.1.1技术功能简介:**

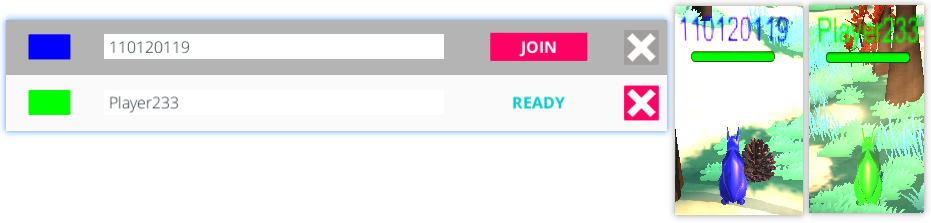
在游戏开始时，生成一个大厅，玩家可以选择两种游戏连接方式，

分别为①服务器玩家利用MATCHMAKER创建带有房间名的房间 ,其他玩家通过房间名列表找到相应房间并加入②服务器玩家利用MANUAL CONNECTION创建房间,其他玩家通过手动输入服务器玩家主机的IP加入



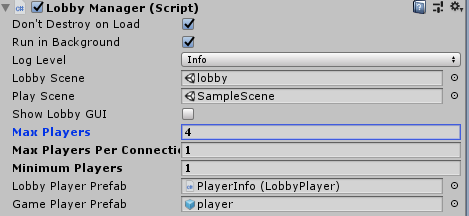
在进入房间后玩家可以更改自己的昵称和旗帜颜色,在所有玩家准备后游戏开始，

玩家的昵称和颜色由和自己所改的昵称及旗帜颜色一致



**4.1.2实现方法及代码:**

通过Unity官方的Network Lobby插件实现，在初始场景中添加插件中的LobbyManager预制件，对其UI进行一定修改，并将初始场景和主游戏场景以及player绑定该组件，在插件中设置游戏最大最小人数并在services里的Multiplayer同步设定,从而实现游戏大厅



对该组件中函数进行重写，实现颜色及昵称一致，代码如下

public class ShitLobbyHook : LobbyHook

{

public override void OnLobbyServerSceneLoadedForPlayer(NetworkManager manager, GameObject lobbyPlayer, GameObject gamePlayer)

{ LobbyPlayer lobby = lobbyPlayer.GetComponent<LobbyPlayer>();

SetupLocalPlayer localPlayer = gamePlayer.GetComponent<SetupLocalPlayer>();

localPlayer.pname = lobby.playerName;

localPlayer.playerColor = lobby.playerColor;

}

}

**4.2.主游戏内玩家和松果的生成以及开启玩家移动**

**4.2.1技术功能简介:**

各个玩家的player在四个特定位置生成。

在玩家生成后开启玩家移动功能

松果在地图内随机生成。

**4.2.2实现方法及代码:**

在主游戏场景内添加四个空物体，拖到地图合适位置，并都加上Network Strat Position组件,作为player的四个生成位置。

并在LobbyManager将PlayerSpawnMethod设为Round Robin即轮流生成，确保不会出现因为随机生成而使两玩家在同一位置生成的情况。



在玩家上挂载如下代码，控制drive函数开启， void Start ()

{

if (isLocalPlayer)

{

GetComponent<drive>().enabled = true;

}

}

用drive函数控制玩家的移动，在player上得把其设为关闭状态，即默认不打钩

public class drive : MonoBehaviour {

float speed = 10.0F;

float rotationSpeed = 100.0F;

void Update () {

float translation = Input.GetAxis("Vertical") \* speed \* Time.deltaTime;

float rotation = Input.GetAxis("Horizontal") \* rotationSpeed \* Time.deltaTime;

transform.Translate(0, 0, translation);

transform.Rotate(0, rotation, 0);

}

}

添加一个空物体，用于生成松果，加上Network Identity,再加上如下代码

实现松果在地图上按给定数目numberOfCones随机生成。

public class ConeSpawner : NetworkBehaviour {

public GameObject ConePrefab;

public int numberOfCones;

public override void OnStartServer()

{

for (int i = 0; i < numberOfCones; i++)

{

var spawnPosition = new Vector3(

Random.Range(-32.0f, 42.0f),

1.0f,

Random.Range(-31f, 31f));

var spawnRotation = Quaternion.Euler(

0.0f,

Random.Range(0, 180),

0.0f);

var cone = (GameObject)Instantiate(ConePrefab, spawnPosition, spawnRotation);

NetworkServer.Spawn(cone);

}

}

**4.3.第三人称跟随视角的实现**

**4.3.1技术功能简介:**

将主相机的位置设在player的后方，并实现跟随效果。

**4.3.2实现方法及代码:**

这个功能是最难实现的，卡了大半天才实现，期间尝试使用两个相机，使相机通过enable来实现对相机跟随，也尝试使用对tag为player的物体进行跟随，可是在联网后都失败了，因为消息机制都过于繁琐，常常所有的相机对准的都是一个player，最后用绑定在player下的这个代码实现，简洁高效。

步骤分别为①在本地player的情况下找到主相机②将主相机的位置设在player后方③将主相机对准player ④将主相机的父对象设为player从而实现绑定跟随。

void Start ()

{

if (isLocalPlayer)

{

Camera.main.transform.position = this.transform.position - this.transform.forward \* forward + this.transform.up \* upward;

Camera.main.transform.LookAt(this.transform.position);

Camera.main.transform.parent = this.transform;

}

}

**4.4.玩家信息同步**

**4.4.1技术功能简介:**

实现client和sever之间玩家昵称、玩家生命值、玩家松果数目、玩家是否死亡这几个信息的同步。

**4.4.2实现方法及代码:**

将这几个值都都设为同步变量,当数值改变后能自动从服务器同步到客户端上，

加上(hook = "OnChangeHealth")可以在改变值后顺便执行OnChangeHealth函数

[SyncVar(hook = "OnChangeHealth")]

public int currentHealth = 100;

[SyncVar(hook = "OnChangeCount")]

public int count = 0;

[SyncVar]

public bool dead = false;

[SyncVar]

public string pname = "player";

void OnChangeHealth(int currentHealth)

{

healthBar.sizeDelta = new Vector2(currentHealth, healthBar.sizeDelta.y);

if (isLocalPlayer)

{

this.currentHealth = currentHealth;

}

}

void OnChangeCount(int count)

{

if (isLocalPlayer)

{

this.count= count;

}

}

因此这些值的变化都在服务器端实现,并同步到客户端，所以对这些值做出改变的函数分成两种情况: ①如果是在服务器端掉用得有if (!isServer){return;}确保唯一性

public void TakeDamage(int amount) //受到冬天伤害,血量越高,伤害越高

{

if (!isServer)

{

return;

}

currentHealth -= (int)(((amount \* currentHealth) / 200) + 5);

if (currentHealth <= 0)

{

if (destroyOnDeath)

{

Destroy(gameObject);

}

else

{

dead = true;

}

}

}

public void TakeplayerDamage(int amount) //受到玩家伤害,

{

if (!isServer)

{

return;

}

currentHealth -= amount;

if (currentHealth <= 0)

{

if (destroyOnDeath)

{

Destroy(gameObject);

}

else

{

dead = true;

}

}

}

②如果是在客户端触发的函数要在服务器端调用得有[Command]并在函数前有Cmd,像是玩家主动吃松果，玩家主动改名这些必定在客户端调用的函数就得将其中数值变化的部分用[Command]实现在服务器端实现。

[Command]

public void CmdEat()

{

if (count > 0)

{

count--;

currentHealth += 20; //吃一个松果回+20血

}

}

[Command]

public void CmdChangeName(string newname)

{

pname = newname;

}

对于第二种情况还有一点要注意的就是，包含[Command]函数的函数,在Cmd函数前得加上if (isLocalPlayer)的限定条件,来确保唯一性，杜绝一个玩家同时操作其他玩家的事件发生。

void Update()

{

if (isLocalPlayer)

{

if (Input.GetKeyDown(KeyCode.E)) //按E键吃松果

{

var heal = GetComponent<health>();

heal.CmdEat();

}

}

}

void OnGUI()

{ if (isLocalPlayer)

{

if (GUI.Button(new Rect(130,Screen.height - 40, 80, 30), "变更昵称"))

{

pname = GUIname;

CmdChangeName(pname);

}

}

}

确保了以上的规范之后,数据出现不同步的问题就只可能是客户端在显示数据时出错，下面就介绍玩家在客户端如何显示信息。

**4.5.客户端同步显示信息**

**4.5.1技术功能简介:**

在GUI上显示玩家的松果数目，显示玩家的生命值，并提供改名选项，

并在player上方用血条和text同步显示玩家的生命值和名称信息。







**4.5.2实现方法及代码:**

1. 昵称显示及更改

首先昵称由大厅内的玩家昵称得到，之前介绍过

public class ShitLobbyHook : LobbyHook

{

public override void OnLobbyServerSceneLoadedForPlayer(NetworkManager manager, GameObject lobbyPlayer, GameObject gamePlayer)

{

LobbyPlayer lobby = lobbyPlayer.GetComponent<LobbyPlayer>();

SetupLocalPlayer localPlayer = gamePlayer.GetComponent<SetupLocalPlayer>();

localPlayer.pname = lobby.playerName;

localPlayer.playerColor = lobby.playerColor;

}

}

用TextMesh显示玩家昵称。

void Update()

{

this.GetComponentInChildren<TextMesh>().text = pname;

}

在TextMesh上加入如下代码使之一直对准主相机。

public class facecamera : MonoBehaviour {

void Update () {

this.transform.LookAt(Camera.main.transform.position);

this.transform.Rotate(new Vector3(0,180,0));

}

}

在GUI中挂载如下代码用于改变玩家昵称。

void OnGUI()

{

if (isLocalPlayer)

{

GUIname = GUI.TextField(new Rect(25, Screen.height - 40, 100, 30), GUIname);

if (GUI.Button(new Rect(130,Screen.height - 40, 80, 30), "变更昵称"))

{

pname = GUIname;

CmdChangeName(pname);

}

[Command]

public void CmdChangeName(string newname)

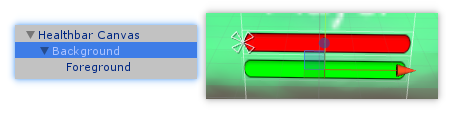
{

pname = newname;

}

1. 血条显示

创建血条组建，红色条为底绿色条在上，将其挂在player上



player下加载如下代码实现控制血条绿条y方向长度从而显示血量。

void OnChangeHealth(int currentHealth)

{

healthBar.sizeDelta = new Vector2(currentHealth, healthBar.sizeDelta.y);

if (isLocalPlayer)

{

this.currentHealth = currentHealth;

}

}

并在player下的血条组件内加入如下代码，使血条永远面向主相机。

public class Billboard : MonoBehaviour {

void Update () {

transform.LookAt(Camera.main.transform);

}

}

1. GUI显示玩家的血量和携带松果数目以及对收到死亡信息就销毁player。

if (isLocalPlayer)

{

display\_health = GetComponent<health>().currentHealth;

display\_count = GetComponent<health>().count;

display\_dead = GetComponent<health>().dead;

GUI.Label(new Rect(220, Screen.height - 40, 150, 30), "携带松果数目："+ display\_count,blackStyle);

GUI.Label(new Rect(410, Screen.height - 40, 100, 30), "当前生命值：" + display\_health, blackStyle);

GUI.Label(new Rect(610, Screen.height - 40, 100, 30), "WASD移动,按E吃下松果" , blackStyle);

if (display\_dead)

{

GUI.Label(new Rect(Screen.width \* 0.5f-200, 80, 180, 400), "游戏结束,你存活了" + (int)(60.0f - timer) + "s", whiteStyle);

Destroy(gameObject);

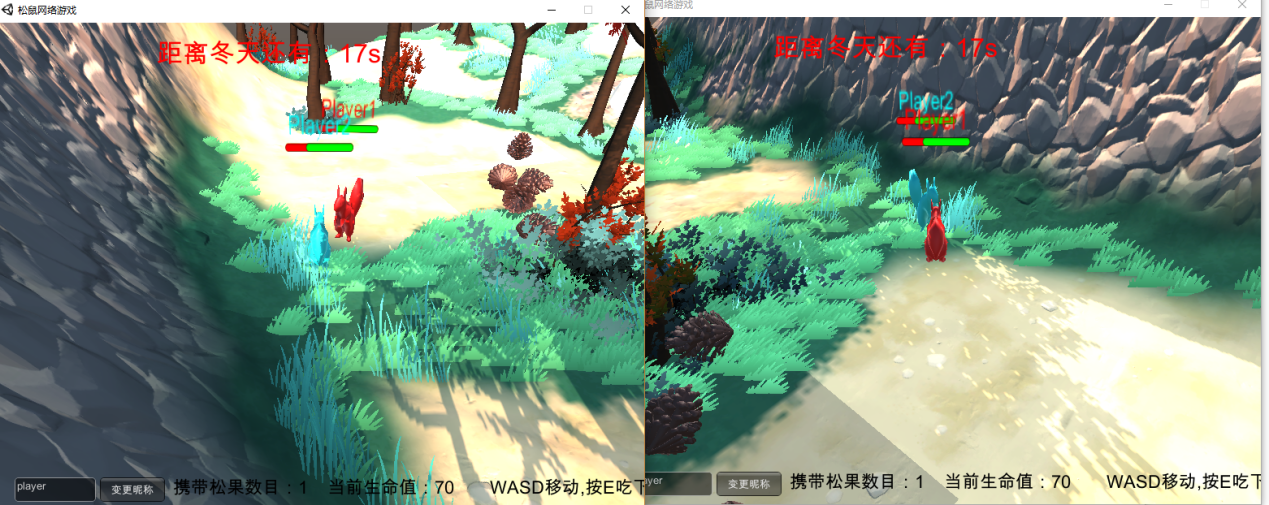
}

**4.6.利用碰撞检测实现捡松果和玩家之间互相伤害**

**4.6.1技术功能简介:**

玩家在地图中游走时，碰到松果，松果数目+1，且不能超过4

玩家与玩家之前碰撞会是使双方玩家都受到伤害。



**4.6.2实现方法及代码:**

在松果上挂载如下代码，实现被玩家碰撞捡起。

public class pine : MonoBehaviour {

void OnCollisionEnter(Collision collision)

{

if (collision.gameObject.CompareTag("Player"))

{

var hit = collision.gameObject;

var counttt = hit.GetComponent<health>();

if (counttt.count < 4) //限制捡4个松果

{

if (counttt != null)

{

counttt.Take();

}

Destroy(gameObject);

}

}

}

}

在玩家上挂上如下代码，实现互相伤害

void OnCollisionEnter(Collision collision) //玩家互相伤害，每下扣10点

{

if (collision.gameObject.CompareTag("Player"))

{

var hit = collision.gameObject;

var counttt = hit.GetComponent<health>();

counttt.TakeplayerDamage(10); //玩家互相伤害，每下扣10点

}

}public void TakeplayerDamage(int amount) //受到玩家伤害,

{

if (!isServer)

{

return;

}

currentHealth -= amount;

if (currentHealth <= 0)

{

if (destroyOnDeath)

{

Destroy(gameObject);

}

else

{

dead = true;

}

}

}

**4.7.冬天环境系统**

**4.7.1技术功能简介:**

利用GUI通知玩家冬天倒计时

并使玩家在冬天及以后每半秒受到持续的环境伤害。









**4.7.2实现方法及代码:**

timer用于时间控制

private float timer = 60.0f;

在GUI上加入如下代码

void OnGUI()

{

GUIStyle blackStyle = new GUIStyle();

blackStyle.fontSize = 22;

blackStyle.normal.textColor = Color.black;

GUIStyle whiteStyle = new GUIStyle();

whiteStyle.fontSize = 32;

whiteStyle.normal.textColor = Color.red;

if (isLocalPlayer)

{

if(timer>20.0f)

GUI.Label(new Rect(Screen.width\*0.5f-200, 20, 180, 200), "距离冬天还有："+(int)(timer-20.0f)+"s", whiteStyle);

else if (timer > 0.0f)

GUI.Label(new Rect(Screen.width \* 0.5f-200, 20, 180, 200), "距离冬天结束还有：" + (int)(timer) + "s", whiteStyle);

else if(!display\_dead)

GUI.Label(new Rect(Screen.width \* 0.5f-200, 20, 180, 200),

"成功度过了冬天，目前存活" + (int)(60.0f - timer) + "s", whiteStyle);

if (display\_dead)

{

GUI.Label(new Rect(Screen.width \* 0.5f-200, 80, 180, 400), "游戏结束,你存活了" + (int)(60.0f - timer) + "s", whiteStyle);

Destroy(gameObject);

}

}

}

在update中加入代码实现在冬天及以后每半秒对所有玩家进行一次环境伤害

void Update()

{

if (display\_dead!=true) //没有dead

{

timer -= Time.deltaTime;

}

if (i\_time\_count >= 0.5) //冬天以后，，，实现每半秒触发一次环境伤害

{

i\_time\_count = 0; //实现每半秒触发一次

if (timer < 20)

{

var heal = GetComponent<health>();

heal.TakeDamage(5);

}

}

i\_time\_count += Time.deltaTime; //实现每半秒触发一次

}

}

在环境伤害函数,因为体重越大，代谢越多，扣得血也多些，

这里是线性关系,基础伤害5，每多20点血，伤害+1;

public void TakeDamage(int amount) //受到冬天伤害,血量越高,伤害

{

if (!isServer)

{

return;

}

currentHealth -= (int)(((amount \* currentHealth) / 200) + 5); //血量越高,伤害越高,这里是线性关系,设amount=10,,基础伤害5，每多20点血，伤害+1;

if (currentHealth <= 0)

{

if (destroyOnDeath)

{

Destroy(gameObject);

}

else

{

dead = true;

}

}

}