项宇豪
22岁
湖州学院（本科）

SHOOT IT!

个人作品展示

contents

Part01
项目展示

项目简介：本项目为一款具有Rouge元素的3d地牢闯关类游戏，玩家将通过一把可以“自由装配”的武器，发射子弹弹幕击杀怪物，进行地牢闯关。玩家在游戏过程中，击杀怪物可以掉落随机的子弹组件，通过拾取不同组件和玩家的装配策略，可以组合出不同的子弹弹幕，当玩家组合出相对强力的弹幕时，就能通过关底Boss，获得胜利

https://www.bilibili.com/video/BV1Se4y1t7X2/?spm_id_from=333.999.0.0&vd_source=f6c2c969abfb4e985a8c49044b236ac8

https://www.bilibili.com/video/BV14Y4y1N7z7/?spm_id_from=333.999.0.0&vd_source=f6c2c969abfb4e985a8c49044b236ac8

该项目为协作开发项目，每个模块由组员共同开发完成

Part02
流程概述

项目初始

01

敲定项目玩法
设计游戏流程
角色策划设计

模块划分

02

玩家
怪物
UI
玩法
场景

开始制作

03

场景搭建
角色编写
核心玩法

后期合并

04

Sences合并
保证玩法
修复BUG

Part03
模块实现

韩徐泓

角色设计

01

玩家角色
小怪设计
狼人BOSS

项宇豪

核心玩法

02

玩法设计
子弹编写
武器系统

张巍

角色设计

03

女巫BOSS
小怪设计
场景转换

王梦晗

UI设计

04

UI逻辑
动态背包
UI设计

朱临凤

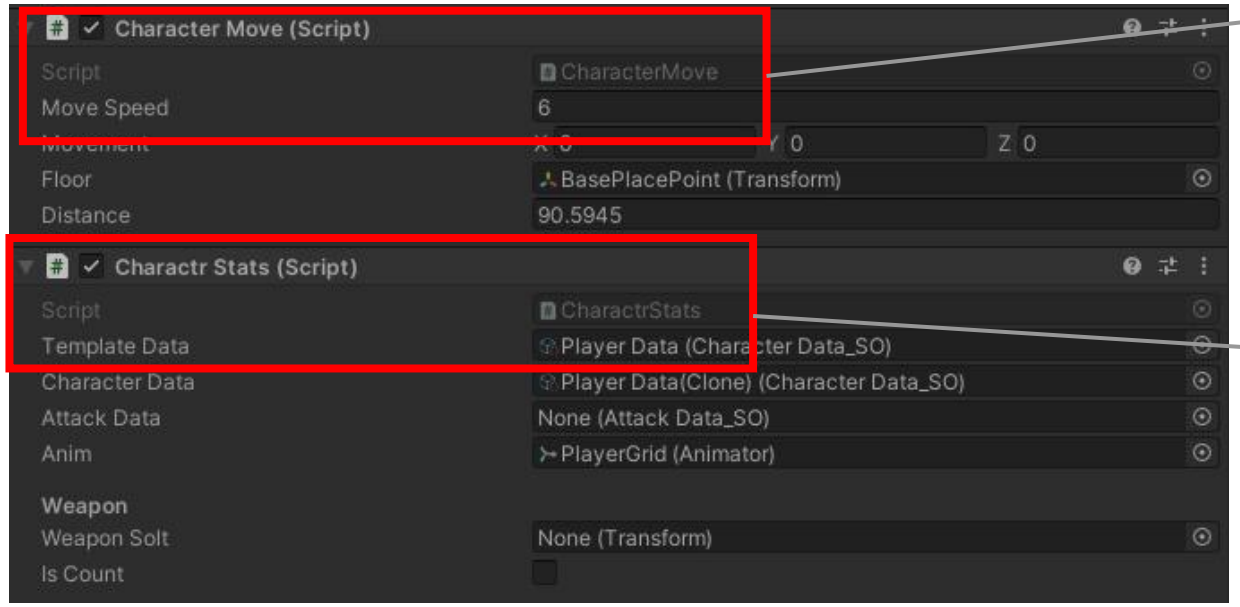
关卡设计

05

关卡搭建
BUFF/陷阱
环境后处理
过场动画

玩家模块

玩家设计



玩家控制脚本

实现代码逻辑和数据的分离

玩家属性脚本

玩家模块

玩家设计
characerMove

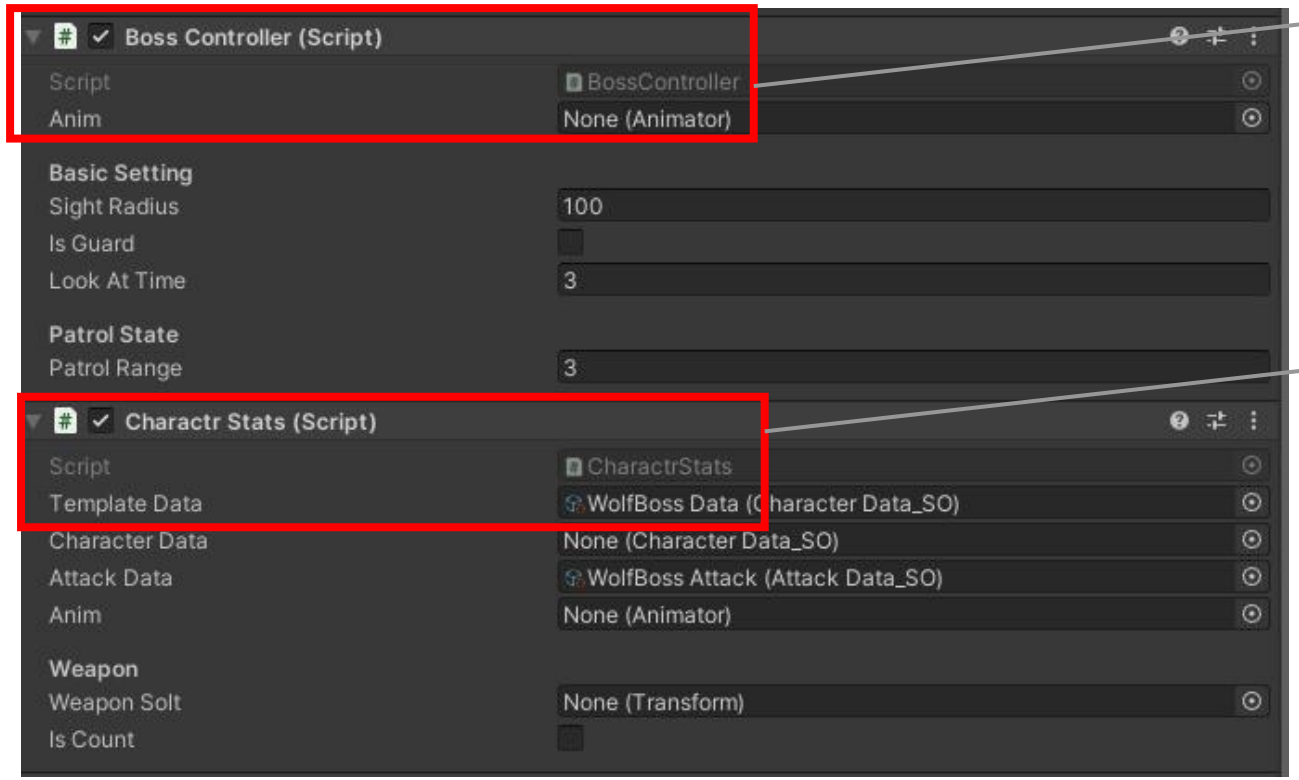
```
private void Move(float h, float v)
{
    movement.Set(h, 0, v);
    movement = movement.normalized * moveSpeed * Time.fixedDeltaTime;
    playerRigidbody.MovePosition(playerRigidbody.position + movement);
}


1 个引用
public void Rotate()
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

    if (plane.Raycast(ray, out distance))
    {
        Vector3 hitPoint = ray.GetPoint(distance);
        Vector3 dir = hitPoint - transform.position;
        dir.y = 0;
        Quaternion targetDir = Quaternion.LookRotation(dir);
        playerRigidbody.MoveRotation(targetDir);
    }

}
```

玩家设计
characerStates

```
public class CharactrStats : MonoBehaviour
{
    public event Action<int, int> UpdateHealthBarOnAttack;

    public CharacterData_SO templateData;
    public CharacterData_SO characterData;


    public AttackData_SO attackData;
    private AttackData_SO baseAttackData;
    private RuntimeAnimatorController baseAnimator;

    public Animator anim;

    [Header("Weapon")]
    public Transform weaponSolt;

    [HideInInspector]
    public bool isCritical;
    public bool isCount;

    //public GameObject UIRoot;
    //public GameObject failed;


    /*private PostProcessVolume volume;
    private Vignette vignette;*/



    //Userlogin userlogin;


    private void Awake()
    {
        if(templateData != null)
        {
            characterData = Instantiate(templateData);
        }
        //baseAttackData = Instantiate(attackData);
        baseAnimator = GetComponent<Animator>().runtimeAnimatorController;
        anim = GetComponent<Animator>();

    }
```

敌人模块

小怪&Boss设计



敌人脚本

敌人状态脚本

敌人模块

小怪&Boss设计
EnemyController

```
void SwitchAnimation()
{
    anim.SetBool("Walk", isWalk);
    anim.SetBool("Chase", isChase);
    anim.SetBool("Follow", isFollow);
    anim.SetBool("Critical", CharactrStats.isCritical);
    anim.SetBool("Death", isDead);
}

void SwitchStates()
{
    if (isDead)
        enemyStates = EnemyStates.DEAD;

    //if finded player, switch to CHASE
    else if (FoundPlayer())
    {
        enemyStates = EnemyStates.CHASE;
        // test Debug.Log("Finded Player"); lession 14
    }
    switch (enemyStates)
    {
        case EnemyStates.GUARD:
            isChase = false;

            if (transform.position != guardPos)
            {
                isWalk = true;
                //agent.isStopped = false;
                agent.destination = guardPos;
                //TODO: Cant doing rotation
                if (Vector3.SqrMagnitude(guardPos - transform.position) <= agent.stoppingDistance)
                {
                    isWalk = false;
                    transform.rotation = Quaternion.Lerp(transform.rotation, guardRotaton, 0.01f);
                }
            }
            break;

        case EnemyStates.PATROL:
```

FSM系统控制敌人行为模式和状态

```
case EnemyStates.PATROL:
    isChase = false;
    agent.speed = speed * 0.5f;
    //if arrived at target
    if (Vector3.Distance(wayPoint, transform.position) <= agent.stoppingDistance)
    {
        isWalk = false;
        if (remainLookAtTime > 0)
            remainLookAtTime -= Time.deltaTime;
        else
            GetNewWayPoint();
    }
    else
    {
        isWalk = true;
        agent.destination = wayPoint;
    }
    break;
case EnemyStates.CHASE:

    isWalk = false;
    isChase = true;

    if (!FoundPlayer())
    {   //
        isFollow = false;
        if (remainLookAtTime > 0)
        {
            agent.destination = transform.position;
            remainLookAtTime -= Time.deltaTime;// remainLookAtTime = remainLookAtTime -Time.deltaTime;
        }
        else if (isGuard)
            enemyStates = EnemyStates.GUARD;
        else
            enemyStates = EnemyStates.PATROL;

    }
    else
    {
        isFollow = true;
        //agent.isStopped = false;
        agent.destination = attackTarget.transform.position;
    }
    //Attack when player in radius
    if (TargetInAttackRange() || TargetInSkillRange())
    {
        isFollow = false;
        //agent.isStopped = true;


        if (inAttackColider1&& lastAttackTime < 0)
        {
            //agent.isStopped = true;
```

敌人模块

掉宝设计
LootItem

代码实现

```
Unity 脚本 | 8 个引用
public class LootSpawner : MonoBehaviour
{
    [System.Serializable]
    1 个引用
    public class LootItem
    {
        public GameObject item;
        [Range(0, 1)]
        public float weight;
    }

    public LootItem[] lootItem;

    4 个引用
    public void Spawnloot()
    {
        float currentValue = Random.value;
        for (int i = 0; i < lootItem.Length; i++)
        {
            if(currentValue <= lootItem[i].weight)
            {
                GameObject obj = Instantiate(lootItem[i].item);
                obj.transform.position = transform.position + Vector3.up * 2;
                //break;//drop one time then STOP
            }
        }
    }
}
```

```
void OnDisable()
{
    /*if (!GameManager.IsInitialized) return;
    GameManager.Instance.RemoveObserver(this);*/

    if (GetComponent<LootSpawner>() && isDead)
    {
        GetComponent<LootSpawner>().Spawnloot();
        agent.enabled = false;
    }
```

死亡后掉宝

武器模块

武器使用流程

拾取掉落物

↓

背包管理

↓

拖入武器栏

↓

后台解析

↓

点击左键发射弹幕

武器模块

```
public class ShootObject : ScriptableObject
{
    public int spend=1;

    public int recover=0;

    public float fireGap;

    public float energy;

    public Vector3 offset;
    public  enum Objtype
    {
        bullet,
        magic,
        trigger,
        modified
    }
    public Objtype objtype;
```

子弹单位的基类

子弹单位的派生类类型
组合类，组合子弹，形成新的发射子树
修正类，修正子弹行为，为子弹添加命令
物体类，提供子弹预制体，通过Resource加载并通过工厂加载

```
[CreateAssetMenu(fileName = "Modfied", menuName = "ShootObject/Modifile/Modfied")]
public class Modfied : ShootObject
{
    public  CommandObj command;


    public virtual void AddModified(GameObject bullets)
    {

        if (command.eventType == BulletEventType.DataValueCommand)
        {
            if (bullets == null)
                Debug.Log("bullet is null");

            command.Execute(bullets.GetComponent<BulletEventController>(), bullets.GetComponent<BulletData>());
            return;
        }

            bullets.GetComponent<BulletEventController>().AddBulletEvent(command.eventType, command.Execute);


    }
```

修正类

```
public abstract class CommandObj :ScriptableObject
{
    public BulletEventType eventType;
    public abstract void Execute(BulletEventController eventController, BulletData bulletData, Collision collision = null, Collider collider = null);
```

命令基类

子弹的行为控制采用命令模式，将行为抽象为命令，具体从命令基类派生。

武器模块

节点代码，保存数据
发射单位

```
public class ShootGroup
{

    ShootObject trigger;
    List<ShootObject> modified_List;
    List<ShootObject> magic_List;
    List<ShootObject> bullets_List;

    public Dictionary<int, int> triggerChildrenDic;
    public int listIndex;

    public float energy;
    public float fireGap;
    4 个引用
    public List<ShootObject> Modified_List { get => modified_List; set => modified_List = value; }
    4 个引用
    public List<ShootObject> Magic_List { get => magic_List; set => magic_List = value; }
    3 个引用
    public List<ShootObject> Bullets_List { get => bullets_List; set => bullets_List = value; }
    1 个引用
    public ShootObject Trigger { get => trigger; set => trigger = value; }
    1 个引用
    public ShootGroup()
    {
        modified_List = new List<ShootObject>();
        magic_List = new List<ShootObject>();
        bullets_List = new List<ShootObject>();
        triggerChildrenDic = new Dictionary<int, int>();
    }

    0 个引用
    public ShootGroup(List<ShootObject> modified_List, List<ShootObject> magic_List, List<ShootObject> bullets_List, ShootObject trigger)
    {
        Modified_List = modified_List;
        Magic_List = magic_List;
        Bullets_List = bullets_List;
        Trigger = trigger;
    }
}
```

根据双亲孩子表示法将子弹
组合按规则存储在树列表里

```
int NodePaket(int startIndex,List<ShootObject> shootObjects,int recover_Parent,List<ShootGroup> TreeNodes,int listIndex)
{

    int r = recover_Parent;
    ShootGroup group = new ShootGroup();
    group.listIndex = listIndex;

        TreeNodes.Add(group);
    do
    {
        if (startIndex >= shootObjects_UsingList.Count)
            break;
        switch (shootObjects[startIndex].objtype)
        {
            case ShootObject.Objtype.magic:
                group.Magic_List.Add(shootObjects[startIndex]);
                break;
            case ShootObject.Objtype.bullet:
                group.Bullets_List.Add(shootObjects[startIndex]);
                break;
            case ShootObject.Objtype.trigger:
                group.triggerChildrenDic.Add(group.Bullets_List.Count,TreeNodes.Count);
                group.Bullets_List.Add(shootObjects[startIndex]);
                startIndex= NodePaket(startIndex+1,shootObjects, shootObjects[startIndex].recover,TreeNodes,listIndex)-1;
                break;
            case ShootObject.Objtype.modified:
                group.Modified_List.Add(shootObjects[startIndex]);
                break;
        }
        r = r - shootObjects[startIndex].spend + shootObjects[startIndex].recover;
        startIndex++;
    } while (r > 0);
    TreeNodes.Add(group);
    return startIndex;
}
```

武器的需求是能够按规则组合发射子弹，会出现多维的数据结构，所以以按照树的数据结构储存数据，发射器会以树为发射单位发射子弹。多次测试后，优化了代码，去掉了多余的结构。

武器模块

```
public class BulletEventController : MonoBehaviour
{
    protected Dictionary<BulletEventType, Action<BulletEventController,BulletData, Collision,Collider>> eventDic;

//添加子弹修正事件
1 个引用
public virtual void AddBulletEvent(BulletEventType bulletEventType, Action<BulletEventController, BulletData, Collision, Collider> action)
{
    if (bulletEventType == BulletEventType.DataValueCommand)
    {
        action(this,bulletData,null,null);
    }
    if (eventDic.ContainsKey(bulletEventType))
    {
        eventDic[bulletEventType]+=action;
        return;
    }
    eventDic.Add(bulletEventType, action);
}
//删除子弹修正事件
0 个引用
public virtual void RemoveBulletEvent(BulletEventType bulletEventType, Action<BulletEventController, BulletData, Collision, Collider> action)
{
    if (eventDic.ContainsKey(bulletEventType))
    {
        eventDic[bulletEventType] -= action;
        return;
    }
    eventDic.Add(bulletEventType, action);
}

//使用子弹修正事件
12 个引用
protected virtual void Event_Do(BulletEventType bulletEventType, Collision collision = null, Collider collider = null)
{
    if (eventDic.ContainsKey(bulletEventType))
        eventDic[bulletEventType]?.Invoke(this, bulletData, collision,collider);
       // eventDic[bulletEventType](this, bulletData, collision, collider);
}
```

委托字典，存储命令

添加子弹修正事件

命令触发

```
public class BulletData : MonoBehaviour
{
    //通过数值控制子弹的基础行为

    //移动
    [Header("初始速度")]
    public float v_speed;
    [Header("初始力")]
    public float forceValue;

    public float a_speed;//加速度，初始化为零

    public int reflectCount;//反弹次数

    //子弹生命周期
    public float lifeTime;

    //攻击
    [HideInInspector]
    public int Atk;
    [HideInInspector]
    public float AtkGap;


    //基本物理组件的获取
    [HideInInspector]
    public Rigidbody rb;
    public Collider collider;


    //长子弹末端确认；
    [Header("子弹末端")]
    public Transform bulletEndPosition;

    //触发型子弹相关
    [HideInInspector]
    public int tgChildrenIndex;
    [HideInInspector]
    public int listIndex;
```

将逻辑代码与数据分离

```
public abstract class CommandObj :ScriptableObject
{
    public BulletEventType eventType;
    3 个引用
    public abstract void Execute(BulletEventController eventController, BulletData bulletData, Collision collision = null, Collider collider = null);

    0 个引用
```

命令基类，使用命令模式将修正命令抽象出来

武器模块

```
protected virtual void  LifeTimer()
{

    lifeTime_Now += Time.deltaTime;
    if (lifeTime_Now >= bulletData.lifeTime)
    {
        Event_Do(BulletEventType.destroyCommand);

        PoolManager.Instance.ReleasePoolGameObject(bulletData.b_name, gameObject);
        return;//回收;
    }
    TriggerBullt();


}
```

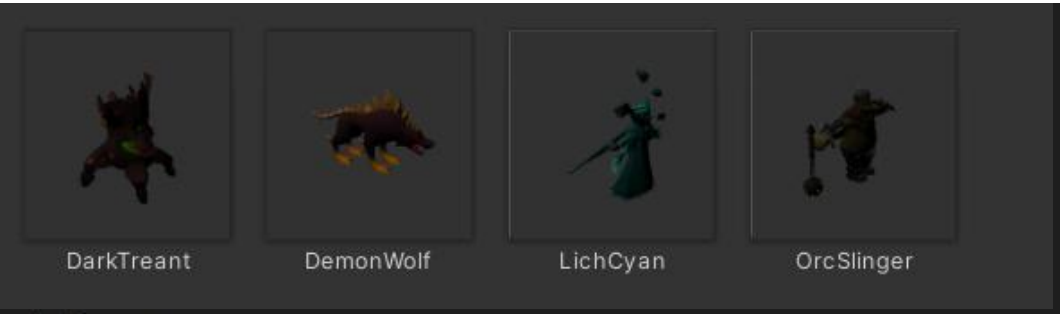
计时器方法，控制子弹的生命周期

使用对象池管理物体，使用工厂生成物体

```
public virtual void Init()
{
    if (eventDic == null)
    {
        eventDic = new Dictionary<BulletEventType, Action<BulletEventController, BulletData, Collision, Collider>>();
    }else
    {
        eventDic.Clear();
    }

    bulletData = GetComponent<BulletData>();

        bulletData.rb = gameObject.GetComponent<Rigidbody>();
        bulletData.collider = gameObject.GetComponent<Collider>();

    lifeTime_Now = 0f;
    bulletData.a_speed = 0f;
    bulletData.listIndex = 0;
    bulletData.tgChildrenIndex = 0;
    bulletData.triggerTime = 0.4f;
}
```

初始化方法

敌人模块

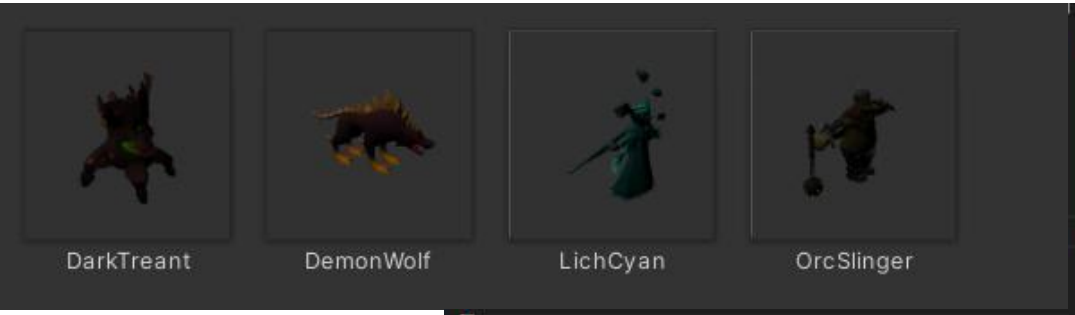


使用工厂模式进行怪物对象创建

```
4 references
public class EnemyFactory
{
    3 references
    private static EnemyFactory instance;

    1 reference
    public static EnemyFactory Instance ···

    1 reference
    private EnemyFactory() ···

    1 reference
    public Enemy CreateEnemy(string id, Vector3 position, Quaternion rotation)
```

```
public Enemy CreateEnemy(string id, Vector3 position, Quaternion rotation)
{
    EnemyData enemyData;

    // ??????
    JsonData jsonData = DataManager.Instance.GetEnemyJsonData(id);    Json读取怪物数据
    if (jsonData == null)
    {
        Debug.LogError("?????????¶??");
        return null;
    }

    enemyData = new EnemyData(jsonData["id"].ToString(), jsonData["name"].ToString(),
        int.Parse(jsonData["dmg"].ToString()), float.Parse(jsonData["atkDis"].ToString()),
        float.Parse(jsonData["atkCD"].ToString()), int.Parse(jsonData["maxHP"].ToString()),
        float.Parse(jsonData["moveSpeed"].ToString()), float.Parse(jsonData["stopDis"].ToString()),
        int.Parse(jsonData["addPlayerHP"].ToString()), int.Parse(jsonData["addPlayerEnergy"].ToString()));
        取出数据创建数据对象

    if (!ObjPools.Instance.HasPool(enemyData.Name))
    {
        // ??Π????
        GameObject prefab = PrefabManager.Instance.GetEnemyPrefab(enemyData.Name);
        if (prefab != null)
        {
            ObjPools.Instance.RegisterPool(enemyData.Name, prefab); // ???????    在此注册对象池
        }
    }

    // ??j?h?????~?GameObject
    Enemy enemy = null;
    ObjPools.Instance.GetObj(enemyData.Name, (obj) =>···    从对象池中获取怪物预制体

    return enemy;
```

敌人模块




```
{
    "1001": {
        "id": 1001,
        "name": "DarkTreant",
        "dmg": 5,
        "atkDis": 1.5,
        "atkCD": 1,
        "maxHP": 100,
        "moveSpeed": 1.5,
        "stopDis": 0,
        "enemyType": "MeleeEnemy",
        "addPlayerHP": 1,
        "addPlayerEnergy": 5
    },
    "1002": {
        "id": 1002,
        "name": "LichCyan",
        "dmg": 10,
        "atkDis": 2,
        "atkCD": 3,
        "maxHP": 200,
        "moveSpeed": 2,
        "stopDis": 0,
        "enemyType": "MeleeEnemy",
        "addPlayerHP": 2,
        "addPlayerEnergy": 6
    },
```

```
// ???÷??????????xű?
Type enemyType = Type.GetType("ZhangWei." + jsonData["enemyType"].ToString());
if (!enemyType.IsSubclassOf(Type.GetType("ZhangWei.Enemy")))
    Debug.LogError($"??????????{enemyType.FullName}?ű?ť??????????????Enemy??");
enemy = obj.AddComponent(enemyType) as Enemy;
enemy.tag = Tags.Enemy;
```

读取配置表中的怪物类型，利用反射将脚本挂在预制体上，实现代码复用

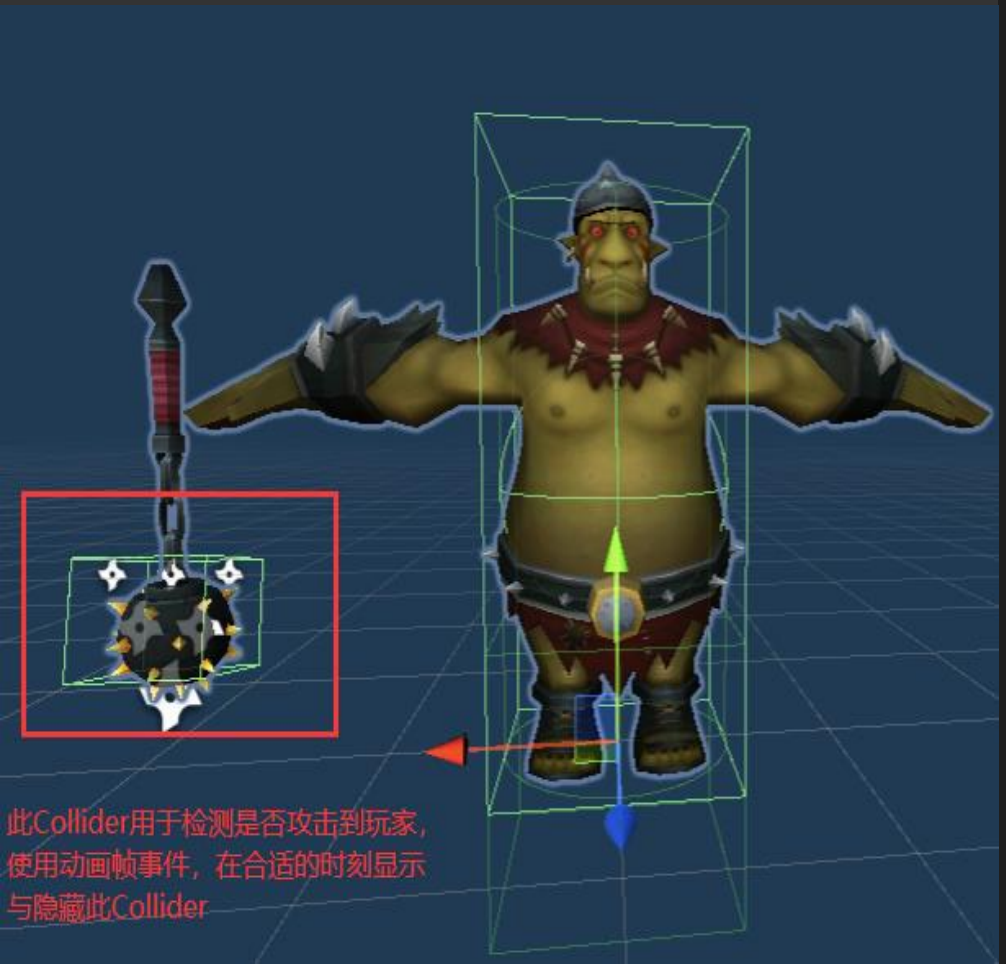
```
SphereCollider atkDisCollider = obj.GetComponent<SphereCollider>();
if (atkDisCollider == null)
{
    Debug.LogError("????ġ??δ???SphereCollider");
    atkDisCollider = obj.AddComponent<SphereCollider>();
}

NavMeshAgent agent = obj.GetComponent<NavMeshAgent>();
if (agent == null)
{
    Debug.LogError("????ġ??δ???NavMeshAgent");
    agent = obj.AddComponent<NavMeshAgent>();
}

NavMeshObstacle meshObstacle = obj.GetComponent<NavMeshObstacle>();
if (meshObstacle == null)
{
    Debug.LogError("????ġ??δ???NavMeshObstacle");
    meshObstacle = obj.AddComponent<NavMeshObstacle>();
}

HitCube hitCube = obj.GetComponentInChildren<HitCube>();
if (hitCube == null)
    Debug.LogError("????ġ??δ???HitCube");

Animator animator = obj.GetComponent<Animator>();
if (animator == null)
    Debug.LogError("????ġ??δ???Animator");
```

添加其它组件

```
enemy.Init(enemyData, atkDisCollider, agent, meshObstacle, hitCube, animator);
```

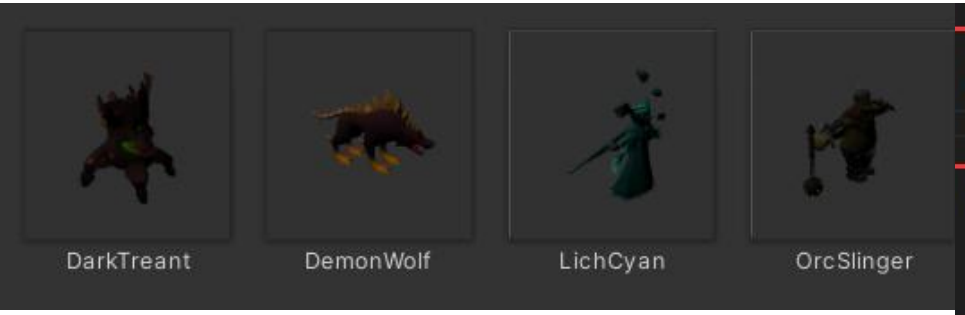
对怪物脚本进行初始化

敌人模块




```
public void Init(EnemyData data, SphereCollider atkDisCollider, NavMeshAgent agent,
    NavMeshObstacle meshObstacle, HitCube hitCube, Animator animator)
{
    this.data = data;
    this.atkDisCollider = atkDisCollider;
    this.agent = agent;
    this.meshObstacle = meshObstacle;
    this.hitCube = hitCube;
    this.animator = animator;

    spawner = GetComponent<LootSpawner>();

    damageCollider = transform.Find("DamageBox").GetComponent<BoxCollider>();

    recoveryEnemyTime = GameManager.Instance.recoveryEnemyTime;

    playerObjs = GameObject.FindGameObjectsWithTag(Tags.Player); // 锟斤拷取锟斤拷戏锟叫硼拷锟斤拷叶锟斤拷锟斤拷锟斤拷
    agent.enabled = false;
    agent.angularSpeed = 360;
    meshObstacle.enabled = false; // 锟截憋拷NavMeshObstacle
    meshObstacle.carving = false;

    hitCube.OnHitPlayer += Hit; // 注锟结攻锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷 使用委托注册击中玩家事件

    atkID = Animator.StringToHash("Atk");
    moveID = Animator.StringToHash("Move");
    dieID = Animator.StringToHash("Die");

    // 锟斤拷锟斤拷锟揭恍┼拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
    atkDisCollider.radius = data.AtkDis; // 锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
    atkDisCollider.isTrigger = true; // 锟斤拷锟斤拷为锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
    agent.speed = data.MoveSpeed; // 锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
    agent.stoppingDistance = data.StopDis; // NavMeshAgent锟斤拷锟斤拷停止锟斤拷锟斤拷锟斤拷锟斤拷
    animator.Play("Idle");
    dieAnimationTime = animator.GetClipLength("Die"); // 锟斤拷锟斤拷前锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
```

敌人模块




```
yield return null;

agent.enabled = true;
stopPathfind = false;
agent.isStopped = false;
```

1、寻路开始时，开启寻路，关闭NavMeshObstacle。
2、寻路结束时，关闭寻路，开启NavMeshObstacle。
3、这样可避免多个怪物寻找同一目标时相互推挤的问题，当前一个怪物到达目标停下来后，则则其它怪物均会重新寻路，更加智能

```
while (!stopPathfind)
{
    NavMeshPath path = new NavMeshPath();
    Vector3[] corners;
    Transform minDisPlayer = null; // 寻路锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
    float minSqrDis = float.MaxValue;

    // 循锟斤拷锟揭筹拷锟斤拷锟斤拷前寻路锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
    foreach (GameObject playerObj in playerObjs)
    {
        if (agent.CalculatePath(playerObj.transform.position, path)) // 模锟斤拷锟斤拷锟斤拷锟斤拷寻路锟斤拷锟斤拷每锟斤拷锟斤拷锟斤拷业锟铰较凤拷锟斤拷
        {
            corners = path.corners;
            // 锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟铰较凤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷杰撅拷锟斤拷锟斤拷
            float sqrDis = 0;
            if (corners.Length > 0)
            {
                sqrDis = Vector3.SqrMagnitude(transform.position - corners[0]);
                for (int i = 1; i < corners.Length; i++)
                    sqrDis += Vector3.SqrMagnitude(corners[i - 1] - corners[i]);
            }

            if (minSqrDis > sqrDis)
            {
                minSqrDis = sqrDis;
                minDisPlayer = playerObj.transform;
            }
        }
    }

    if (minDisPlayer != null)
    {
        agent.SetDestination(minDisPlayer.position); // 锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷叹锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
```

敌人模块


```
0 references
public static class AnimatorExt
{
    1 reference
    public static float GetClipLength(this Animator animator, string clip)
    {
        if (null == animator || string.IsNullOrEmpty(clip) || null == animator.runtimeAnimatorController)
            return 0;
        RuntimeAnimatorController ac = animator.runtimeAnimatorController;
        AnimationClip[] tAnimationClips = ac.animationClips;
        if (null == tAnimationClips || tAnimationClips.Length <= 0) return 0;
        AnimationClip tAnimationClip;
        for (int tCounter = 0, tLen = tAnimationClips.Length; tCounter < tLen; tCounter++)
        {
            tAnimationClip = ac.animationClips[tCounter];
            if (null != tAnimationClip && tAnimationClip.name == clip)
                return tAnimationClip.length;
        }

        return 0f;
    }
}
```

对Animator组件进行扩展，利用RuntimeAnimatorController可获取所有帧动画的详细信息，以此获得帧动画的播放时长

敌人模块

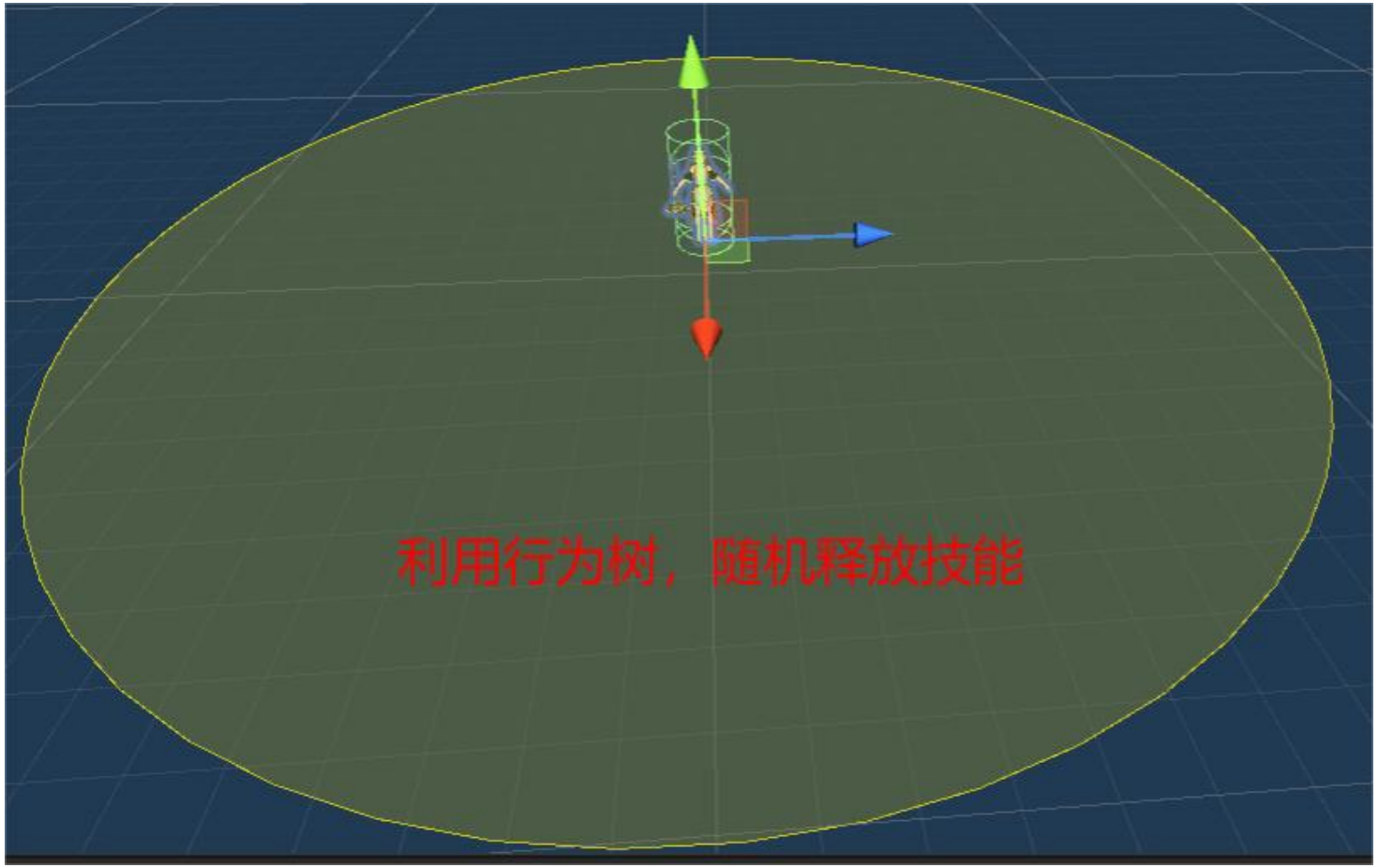
```
4 references
public class DataManager : MonoBehaviour
{
    4 references
    public static DataManager Instance;

    1 reference
    private string enemyDataPath = "Data/EnemyData";
    4 references
    private Dictionary<string, JsonData> enemyJsonData;
    5 references
    private List<string> allEnemyID;

    0 references
    private void Awake()
    {
        Instance = this;

        enemyJsonData = new Dictionary<string, JsonData>();
        Parse();

        // 锟斤拷锟斤拷锟斤拷锟叫碉拷锟斤拷锟斤拷ID锟斤拷锟斤拷小->锟斤拷锟斤拷锟剿筹拷锟芥到锟叫憋拷锟斤拷锟斤拷
        allEnemyID = new List<string>();
        foreach (string id in enemyJsonData.Keys)
        {
            allEnemyID.Add(id);
        }

        allEnemyID.Sort((a, b) => int.Parse(a) - int.Parse(b));
    }

    /// <summary>
    /// 锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷
    /// </summary>
    1 reference
    private void Parse()
    {
        TextAsset textAsset = Resources.Load<TextAsset>(enemyDataPath);
        if (textAsset == null)
            Debug.LogError("锟斤拷锟斤拷锟斤拷锟斤拷锟斤拷源失锟斤拷锟斤拷");
        else
            enemyJsonData = JsonMapper.ToObject<Dictionary<string, JsonData>>(textAsset.text);
    }
```

使用LitJson对Json配置表反序列化

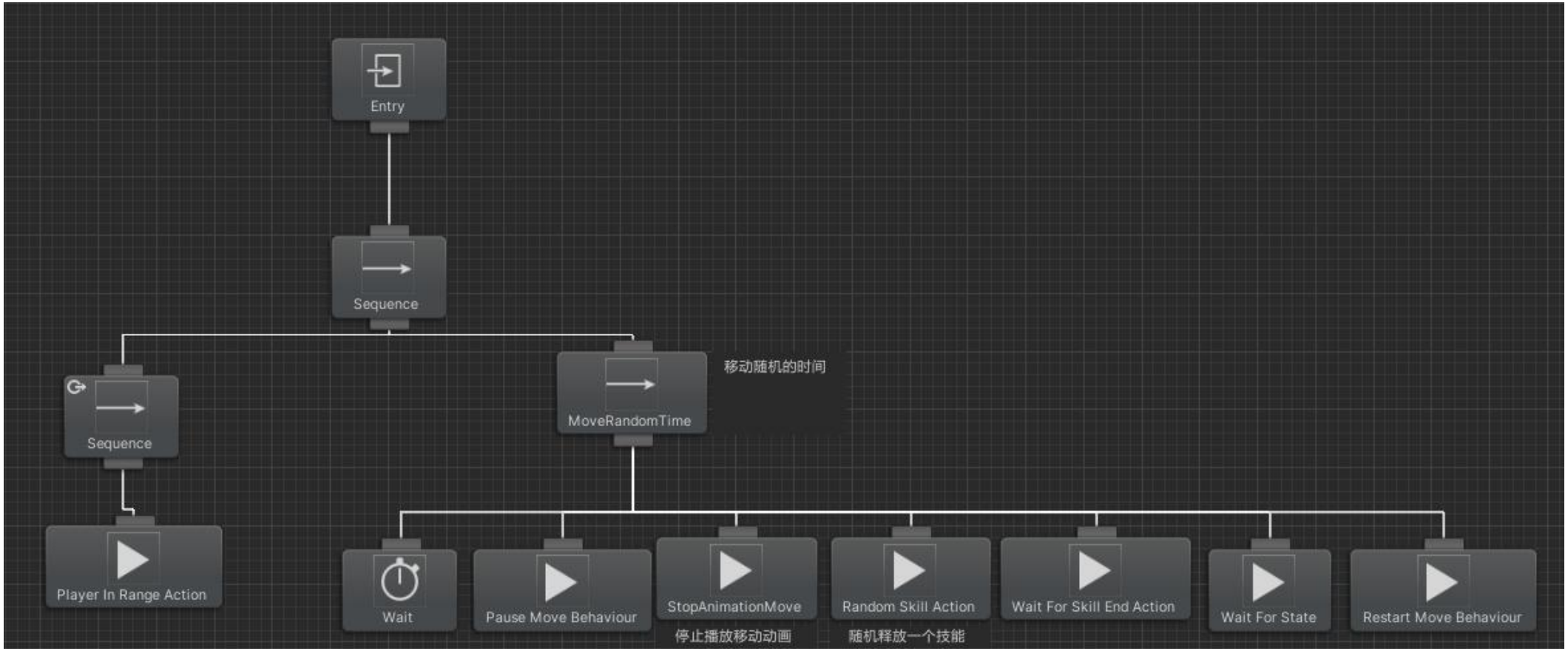
```
2 references
public class PrefabManager : MonoBehaviour
{
    2 references
    public static PrefabManager Instance;

    1 reference
    private string enemyPath = "Enemy";
    4 references
    private Dictionary<string, GameObject> enemyPrefabDic;

    0 references
    private void Awake()
    {
        Instance = this;

        enemyPrefabDic = new Dictionary<string, GameObject>();

        GameObject[] enemyPrefabs = Resources.LoadAll<GameObject>(enemyPath);
        foreach (GameObject obj in enemyPrefabs)
        {
            if (enemyPrefabDic.ContainsKey(obj.name))
            {
                Debug.LogError($"加载敌人预制体失败，名字相同：{obj.name}");
            }
            else
            {
                enemyPrefabDic.Add(obj.name, obj);
            }

            // .Net2.1及以上才有用
            // if (!enemyPrefabDic.TryAdd(obj.name, obj))
            //     Debug.LogError($"加载敌人预制体失败，名字相同：{obj.name}");
        }
    }
```

提前将所有敌人预制体都加载进内存，便于在第一次注册对象池时使用

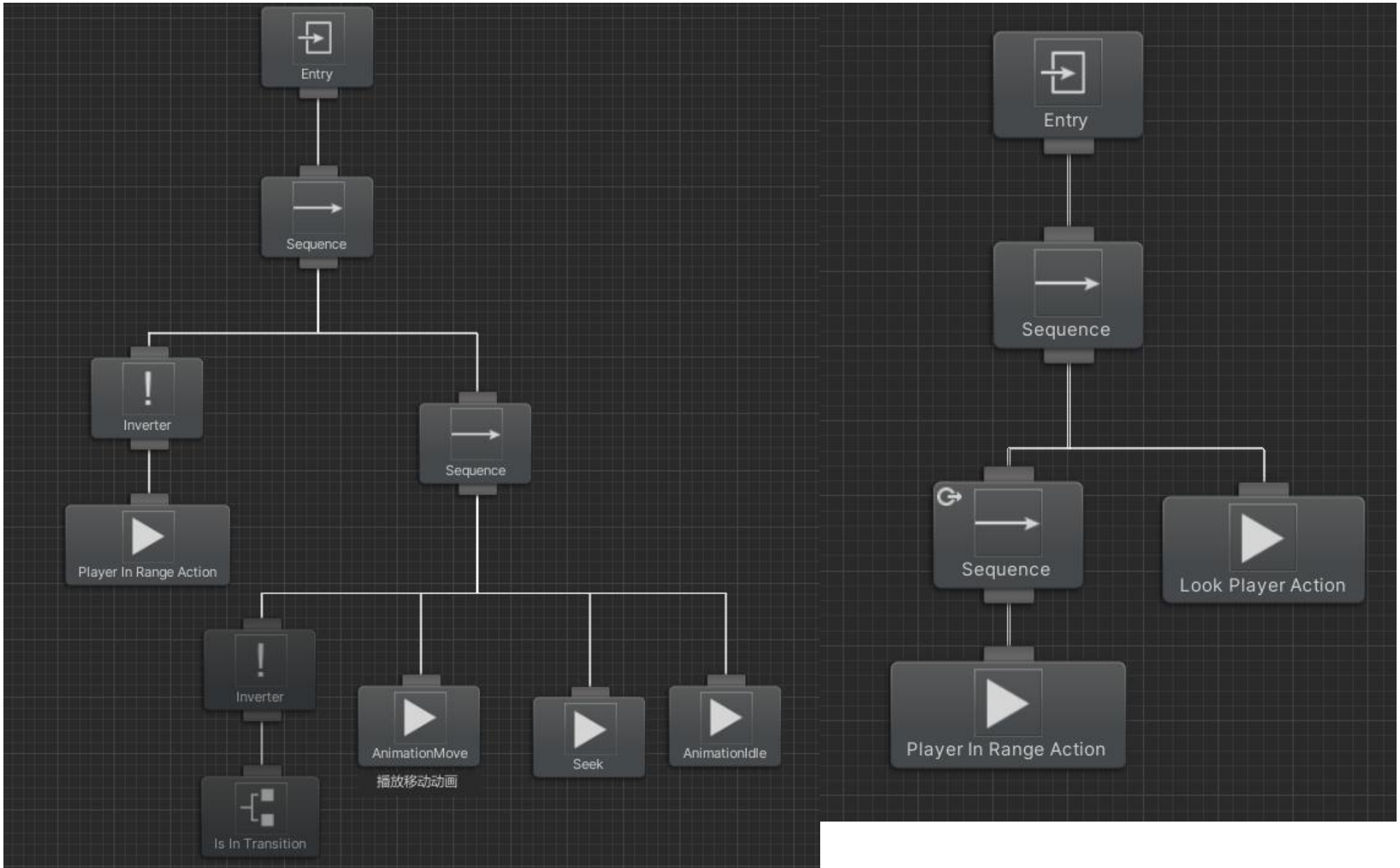
敌人模块



敌人模块


Entry
Sequence
Sequence
MoveRandomTime
移动随机的时间
Player In Range Action
Wait
Pause Move Behaviour
StopAnimationMove
停止播放移动动画
Random Skill Action
随机释放一个技能
Wait For Skill End Action
Wait For State
Restart Move Behaviour

敌人模块


Entry
Sequence
Inverter
Player In Range Action
Sequence
Inverter
Is In Transition
AnimationMove
播放移动动画
Seek
AnimationIdle
Entry
Sequence
Sequence
Look Player Action
Player In Range Action

UI模块

动态加载界面：将每个View放在Resources里，第一次加载放入字典，之后打开从字典得到View。

```
8 个引用
public enum UIType
{
    MainView, LoginView, SettingView, CharacterView
}
0 个引用
public class UIManager
{
    private static UIManager _instance;
    3 个引用
    public static UIManager Instance
    {
        get
        {
            if (_instance == null)
            {
                _instance = new UIManager();
                UIViewInfoList InfoList = Resources.Load<UIViewInfoList>("UIViewInfo_Config");
                foreach(UIViewInfo item in InfoList.List)
                {
                    _instance.allPaths.Add(item.uiView, item.uiPath);
                }
            }
            return _instance;
        }
    }

    public Dictionary<UIType, string> allPaths = new Dictionary<UIType, string>();//键是枚举UI类型，值是路径
    public Dictionary<UIType, GameObject> allViews = new Dictionary<UIType, GameObject>();//键是枚举UI类型，值是对象Canvas。

    0 个引用
    public void OpenView(UIType uiType)
    {
        string path = _instance.allPaths[uiType];
        if (_instance.allViews.ContainsKey(uiType))
        {
            allViews[uiType].SetActive(true);
            allViews[uiType].GetComponent<Canvas>().sortingOrder = GetMaxSortingOrder() + 1;
        }
        else
        {
            GameObject prefab = Resources.Load<GameObject>(path);//获取预制体
            GameObject clonePop = GameObject.Instantiate(prefab);//克隆对象
            clonePop.transform.SetParent(GameObject.Find("UICanvasParent").transform);//设置父物体
            clonePop.GetComponent<RectTransform>().anchorMin = Vector2.zero;//动态更新需要设置锚点。
            clonePop.GetComponent<RectTransform>().anchorMax = Vector2.one;
            clonePop.GetComponent<RectTransform>().sizeDelta = Vector2.zero;
            clonePop.transform.localPosition = Vector3.zero;
            clonePop.transform.localScale = Vector3.one;
            clonePop.GetComponent<Canvas>().sortingOrder = GetMaxSortingOrder() + 1;
            allViews.Add(uiType, clonePop);
        }
    }
```

使用ScriptableObject储存数据：一个是储存image和text的资源文件；另外一个是储存这个资源文件的列表的资源文件。

```
[CreateAssetMenu(fileName = "MyNew Item", menuName = "MyInventory/MyItem Data")]
public class MyItemData_SO : ScriptableObject
{
    public string itemName;
    public Sprite itemIcon;
    public int itemAmount=1;
    public ShootObject shootObject;
    [TextArea]
    public string description = "";

    1 个引用
    public string Setdescription()
    {
        string str=description;
        Debug.Log("Dic");
        switch (shootObject.objtype)
        {
            case ShootObject.Objtype.bullet:
                str = "Spend: " + shootObject.spend + "\n" + "Recover: " + shootObject.recover + "\n" + "FireGap: " + shootObject.fireGap + "\n" + "Energy: " + shootObject.energy + "\n";
                break;
            case ShootObject.Objtype.magic:
                str = "Spend: " + shootObject.spend + "\n" + "Recover: " + shootObject.recover + "\n" + "FireGap: " + shootObject.fireGap + "\n" + "Energy: " + shootObject.energy + "\n";
                break;
            case ShootObject.Objtype.trigger:
                str = "Spend: " + shootObject.spend + "\n" + "Recover: " + shootObject.recover + "\n" + "FireGap: " + shootObject.fireGap + "\n" + "Energy: " + shootObject.energy + "\n";
                break;
            case ShootObject.Objtype.modified:
                str = "Spend: " + shootObject.spend + "\n" + "Recover: " + shootObject.recover + "\n" + "FireGap: " + shootObject.fireGap + "\n" + "Energy: " + shootObject.energy + "\n";
                break;
        }
        str = str + "\n" + description;
        return str;
    }
```

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

[CreateAssetMenu(fileName = "MyNew Inventory", menuName = "MyInventory/MyInventory Data")]
public class MyInventoryData_SO :ScriptableObject
{
    [SerializeField]
    public List<MyInventoryItem> myitems = new List<MyInventoryItem>();

    1 个引用
    public void MyAddItem(MyItemData_SO newItemData, int amount)
    {
        for (int i = 0; i < myitems.Count; i++)
        {
            if (myitems[i].myitemData == null)
            {
                myitems[i].myitemData = newItemData;
                myitems[i].myamount = amount;
                break;
            }
        }
    }
    [System.Serializable]
    2 个引用
    public class MyInventoryItem
    {
        public MyItemData_SO myitemData;
        public int myamount;
    }
}
```

数据的交换

格子列表的每一个格子对应每一个资源文件列表里的资源文件。
背包列表对应背包资源文件列表，装备列表对应装备资源文件列表。

遍历每一个格子的脚本，得到对应的资源文件，将资源文件的数据传入item脚本，item脚本得到image和text组件，将数据给image和text组件。

```
public class MyBagUI : MonoBehaviour
{
    public List<GeziUI> MyBaglist;//MyBaglist存放每个格子对象的geziui脚本对象。
    public void MyRefreshUI()//
    {
        for(int i = 0; i < MyBaglist.Count; i++)
        {
            MyBaglist[i].myitemUI.Myindex = i;//将geziui的脚本对象的序号对应每一个item资源文件的序号
            MyBaglist[i].MyUpdateItem();
        }
    }
}
```

```
public class MyItemUI : MonoBehaviour
{
    public Image icon = null;
    public Text amount = null;

    public MyInventoryData_SO MyBag { get; set; }
    public int Myindex { get; set; } = -1;
    public void MySetUpItemSlotUI(MyItemData_SO item, int itemAmount)
    {
        if (item != null)
        {
            icon.sprite = item.itemIcon;
            amount.text = itemAmount.ToString();
            icon.gameObject.SetActive(true);
        }
        else
        {
            icon.gameObject.SetActive(false);
        }
    }

    public MyItemData_SO MyGetItem()
    {
        return MyBag.myitems[Myindex].myitemData;
    }
}
```

```
public enum SoltType { BAG, EQUIPMENT}
public class GeziUI : MonoBehaviour,IPointerEnterHandler,IPointerExitHandler
{
    public SoltType soltType;
    public MyItemUI myitemUI;
    public void MyUpdateItem()
    {
        switch (soltType)
        {
            case SoltType.BAG:
                myitemUI.MyBag = MyInventoryManager.Instance.myInventoryData;//将唯一的那个背
                break;
            case SoltType.EQUIPMENT:
                myitemUI.MyBag = MyInventoryManager.Instance.myEquipmentData;
                break;
        }
        var myitem = myitemUI.MyBag.myitems[myitemUI.Myindex];
        myitemUI.MySetUpItemSlotUI(myitem.myitemData, myitem.myamount);
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        if (myitemUI.MyGetItem())
        {
            MyInventoryManager.Instance.MyItemToolTip.SetUpToolTip(myitemUI.MyGetItem());
            MyInventoryManager.Instance.MyItemToolTip.gameObject.SetActive(true);

        }
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        MyInventoryManager.Instance.MyItemToolTip.gameObject.SetActive(false);
    }
```

不管是拾取物品；背包、装备物品的拖拽和交换都是先改变资源文件列表的顺序，然后格子脚本得到最新的对应的资源文件，通过资源文件的数据更新UI。

拾取世界道具到背包

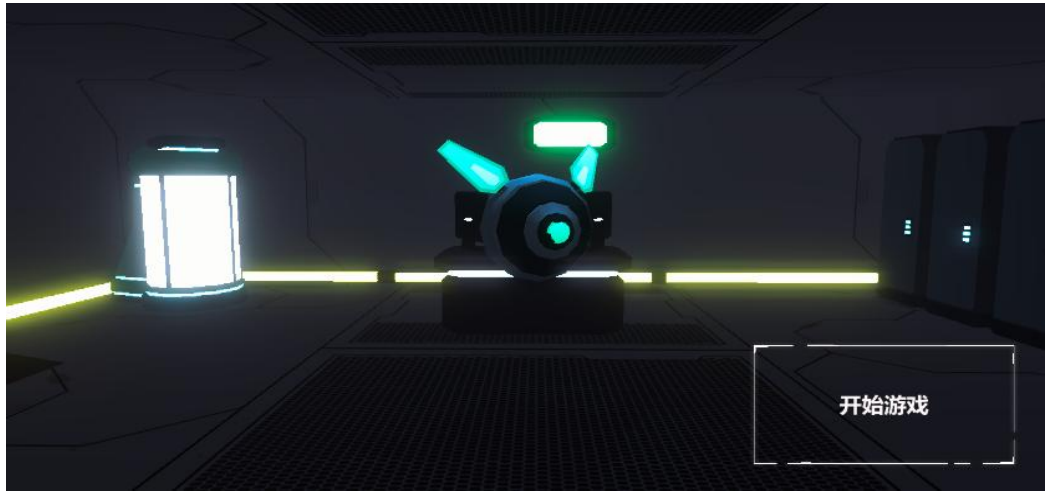
```
public class MyItemPickUp : MonoBehaviour
{
    public MyItemData_SO myitemData;
    public MagicList_SO magicListSO;
    private List<MyItemData_SO> magicList;

    //private List<ShootObject> shootObjectsList;

    Unity 消息|0 个引用
    private void Start()
    {
        magicList = magicListSO.Magiclist;
    }

    Unity 消息|0 个引用
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {

            //Debug.Log("picked up");


            if (magicList.Count != 0)
            {
                int i = Random.Range(0, magicList.Count);
                //Debug.Log(i);
                myitemData = magicList[i];

            }

            MyInventoryManager.Instance.myInventoryData.MyAddItem(myitemData, myitemData.itemAmount);
            MyInventoryManager.Instance.myInventoryUI.MyRefreshUI();
            Destroy(gameObject);
        }
    }
}
```

背包、装备物品的拖拽与交换

```
public class MyDragItem : MonoBehaviour, IBeginDragHandler, IDragHandler, IEndDragHandler
{
    MyItemUI myCurrentItemUI;
    GeziUI myCurrentGeziUI;
    GeziUI myTargetGeziUI;
    Unity 消息|0 个引用
    private void Awake()
    {
        myCurrentItemUI = GetComponent<MyItemUI>();
        myCurrentGeziUI = GetComponentInParent<GeziUI>();
    }
    1 个引用
    public void OnBeginDrag(PointerEventData eventData)
    {
        //记录原始信息
        MyInventoryManager.Instance.currentDrag = new MyInventoryManager.DragData();
        MyInventoryManager.Instance.currentDrag.originalGeziUI = GetComponentInParent<GeziUI>();
        MyInventoryManager.Instance.currentDrag.originalParent =(RectTransform)transform.parent;
        transform.SetParent(MyInventoryManager.Instance.dragCanvas.transform,true);
    }

    2 个引用
    public void OnDrag(PointerEventData eventData)
    {
        //跟随物品移动
        transform.position = eventData.position;
    }

    1 个引用
    public void OnEndDrag(PointerEventData eventData)
    {
        //放下物品，交换数据
        if (EventSystem.current.IsPointerOverGameObject())
        {
            if (MyInventoryManager.Instance.CheckInventoryUI(eventData.position)||MyInventoryManager.Instance.CheckEquipmentUI(eventData.position))
            {
                if (eventData.pointerEnter.gameObject.GetComponent<GeziUI>())
                    myTargetGeziUI = eventData.pointerEnter.gameObject.GetComponent<GeziUI>();
                else
                    myTargetGeziUI = eventData.pointerEnter.gameObject.GetComponentInParent<GeziUI>();
            }
            switch (myTargetGeziUI.soltType)
            {
                case SoltType.BAG:
                    SwapItem();
                    break;
                case SoltType.EQUIPMENT:
                    SwapItem();
                    break;
            }
            myCurrentGeziUI.MyUpdateItem();
            myTargetGeziUI.MyUpdateItem();
```

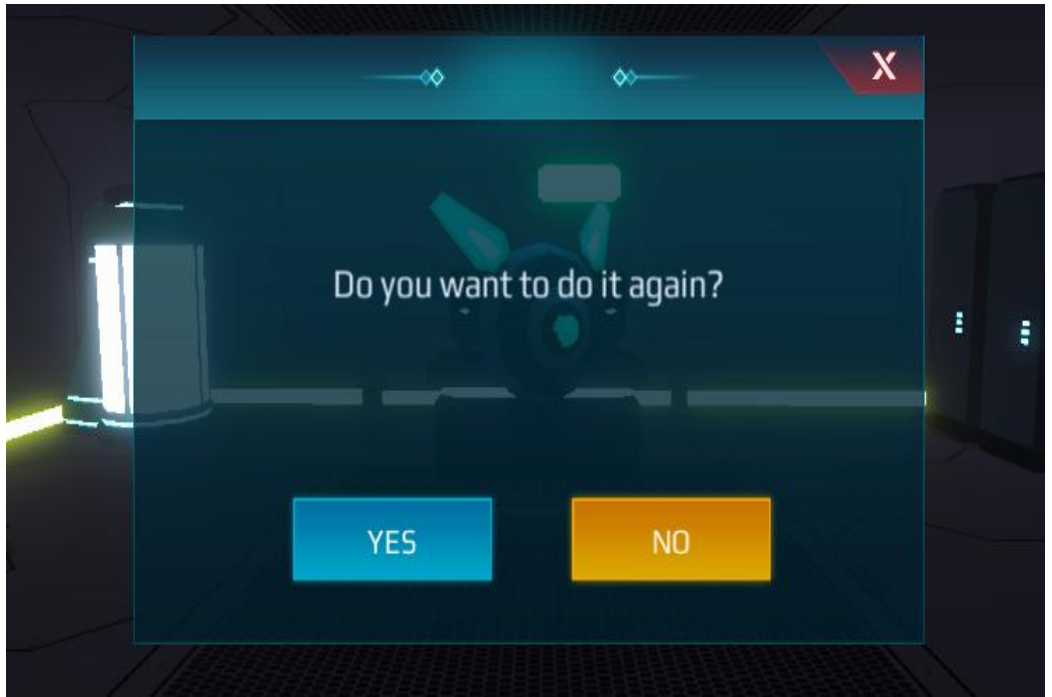
1.物品信息的属性描述

2.切换场景进入游戏画面



游戏胜利和失败

可以退出和再来一局



1.悬浮到物品可以显示详细属性。

2.点击开始游戏，使用Onclick事件监听切换到游戏场景，UI动态加载不销毁。

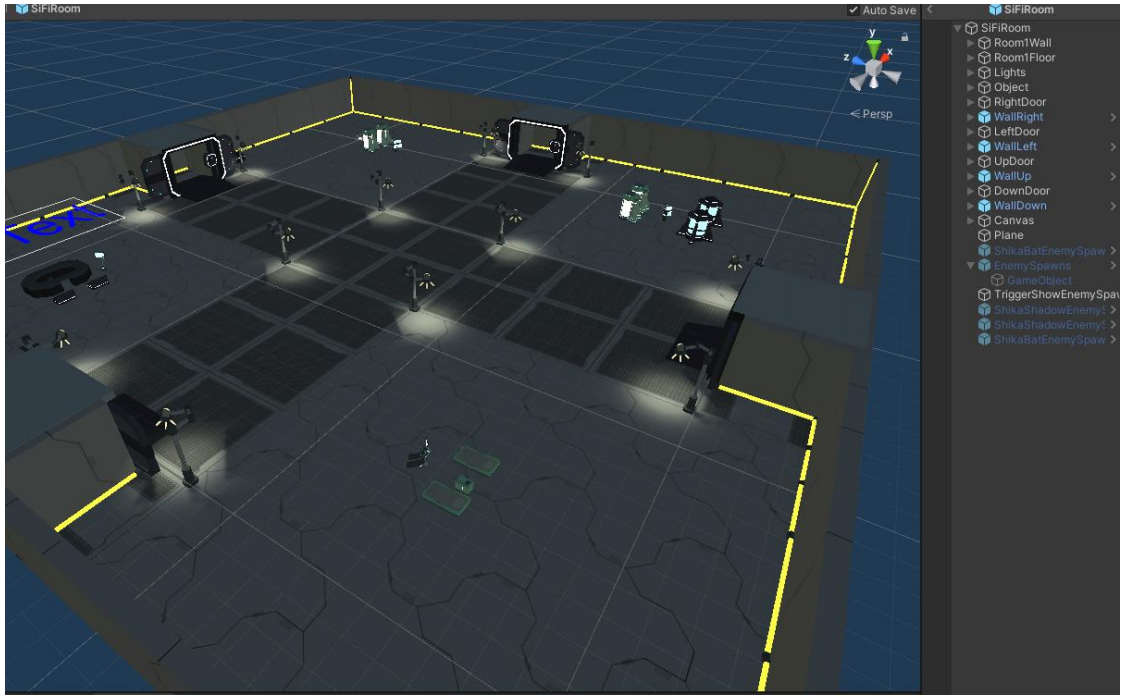
3.血量为零激活失败图标。
打败boss激活胜利图标。
可以选择再来一局，进入开始场景。

使用MVC思想，将代码拆分

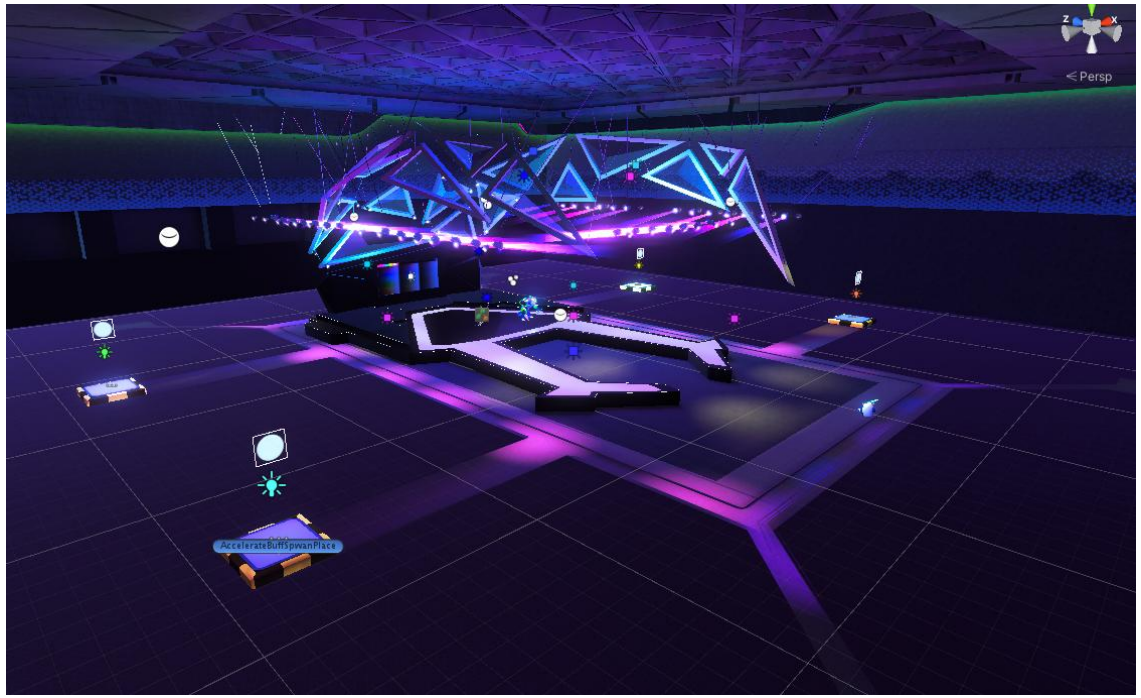
场景模块

场景搭建

- 普通关卡



- BOSS关卡



场景模块

代码展示 - 普通关卡

```
public class Room : MonoBehaviour
{
    public GameObject doorLeft, doorRight, doorUp, doorDown;
    public bool roomLeft, roomRight, roomUp, roomDown;
    public GameObject wallLeft, wallRight, wallUp, wallDown;

    public Transform[] doors;

    public Animation dooranim;
    public int stepToStart;
    public Text text;
    public int doorNumber;
    // Start is called before the first frame update
    void Start()
    {
        //布尔门是否生成
        doorLeft.SetActive(roomLeft);
        doorRight.SetActive(roomRight);
        doorUp.SetActive(roomUp);
        doorDown.SetActive(roomDown);

        wallLeft.SetActive(!roomLeft);
        wallRight.SetActive(!roomRight);
        wallUp.SetActive(!roomUp);
        wallDown.SetActive(!roomDown);
    }

    public void UpdateRoom()
    {
        stepToStart = (int)(Mathf.Abs(transform.position.x / 20.8f)+Mathf.Abs(transform.position.z/20.4f)/2);

        text.text = stepToStart.ToString();

        if (roomUp)
            doorNumber++;
        if (roomDown)
            doorNumber++;
        if (doorLeft)
            doorNumber++;
        if (doorRight)
            doorNumber++;

    }
```

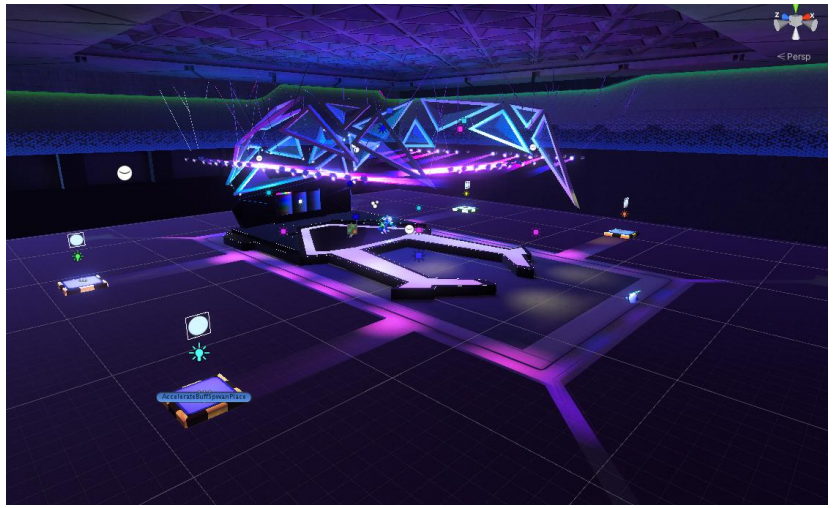
- 房间判定：生成门，方向，墙的阻断

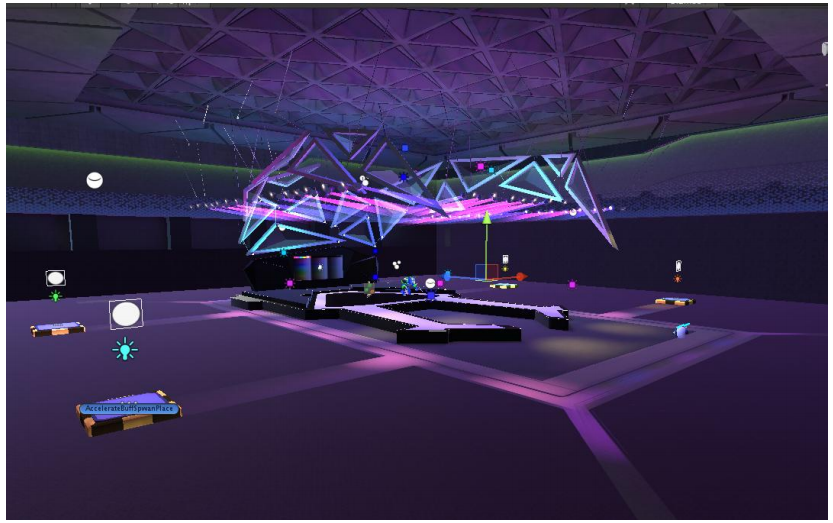
```
void Start()
{
    bakeTool.SetActive(false);
    for (int i = 0; i < roomNumber; i++)
    {
        rooms.Add(Instantiate(roomPrefab, generatorPoint.position, Quaternion.identity).GetComponent<Room>());
        ChangePointPosition();
    }
    endRoom = rooms[0].gameObject;
    foreach (var room in rooms)
    {
        SetupRoom(room, room.transform.position);
    }
    FindEndRoom();
    NavMeshBake();
}
/// <summary> 房间生成
void Update()...

/// <summary> 设置随机生成房间
public void ChangePointPosition()
{
    Vector3 temp = generatorPoint.position;
    do
    {
        generatorPoint.position = temp;
        direction = (Direction)Random.Range(0, 4);
        switch (direction)...
    } while (Physics.OverlapBox(generatorPoint.position, new Vector3(1f,1f,1f), Quaternion.identity).Length>0);

}

/// <summary> 设置房间之间的门的连接
public void SetupRoom(Room newRoom, Vector3 roomPosition)
{
    newRoom.roomUp = Physics.OverlapBox(roomPosition + new Vector3(0, 0, zOffset * 2), new Vector3(1f, 1f, 1f), Quaternion.identity).Length > 0;
    newRoom.roomDown = Physics.OverlapBox(roomPosition + new Vector3(0, 0, -zOffset * 2), new Vector3(1f, 1f, 1f), Quaternion.identity).Length > 0;
    newRoom.roomLeft = Physics.OverlapBox(roomPosition + new Vector3(-xOffset * 2, 0, 0), new Vector3(1f, 1f, 1f), Quaternion.identity).Length > 0;
    newRoom.roomRight = Physics.OverlapBox(roomPosition + new Vector3(xOffset * 2, 0, 0), new Vector3(1f, 1f, 1f), Quaternion.identity).Length > 0;

    newRoom.UpdateRoom();
}
```

- 房间生成：随机

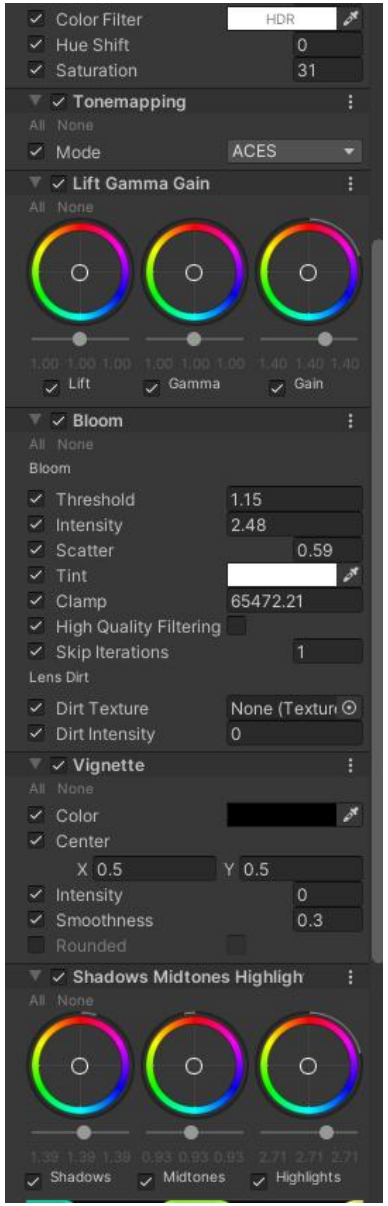
场景模块

BOSS关卡





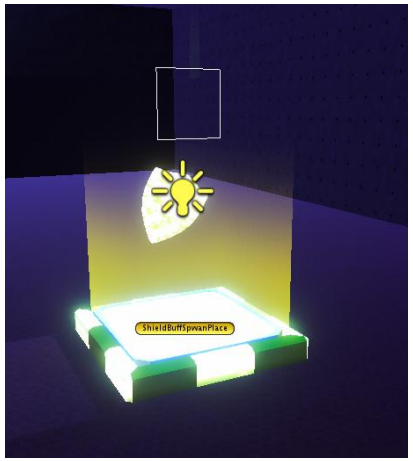
- 后处理效果添加

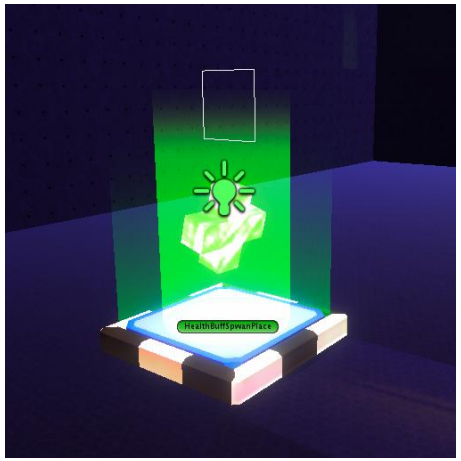


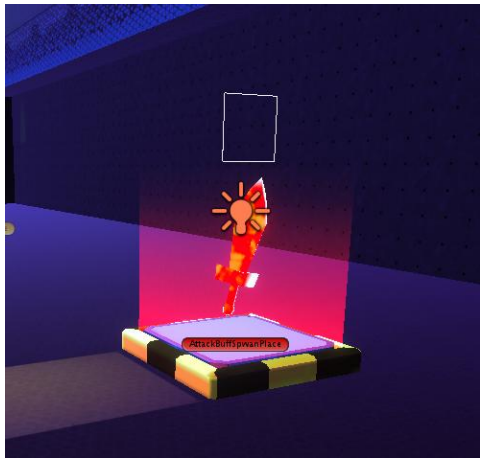
- 后处理组件:
  - Bloom
  - Vignette
  - Shadows
  - White Balance
  - Depth of Field
  - Motion Blur

场景模块

BOSS关卡 - 增益BUFF







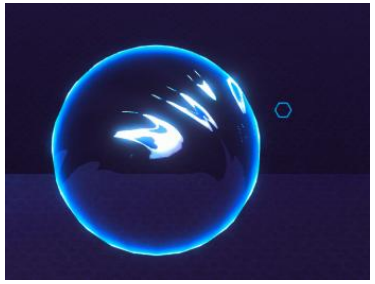
- 四种正面效应BUFF：护盾（无敌），加速，加血，伤害
  - BUFF拾取后进入倒计时，上方显示倒计时图标
  - 延迟后生成BUFF
  - 护盾：生成一个扭曲护盾在玩家处，一定时间后消失
  - 加血：直接加血
  - 加速：一定时间内玩家加速
  - 伤害：直接对BOSS造成伤害

场景模块

BOSS关卡 - 增益BUFF - 护盾代码

```
void Start()
{
    playerLayer = LayerMask.NameToLayer("Player");
    player = GetComponent<Transform>();
    shieldBuff = Instantiate(shieldVFX, transform.position, transform.rotation);
}

void Update()
{
    if (coolDown >= 0)
    {
        coolDown -= Time.deltaTime;
    }
    coolDownSlider.fillAmount = coolDown / coolDownMax;
}

private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.layer == playerLayer)
    {
        isTaken = true;
        Destroy(shieldBuff);

        if (coolDown <= 0 && isTaken)
        {
            ShieldAdd(other);
            coolDown = coolDownMax;
            isTaken = false;
            StartCoroutine(BuffReSpwan(coolDownMax));
        }
    }
}
```

```
void ShieldAdd(Collider player)
{if (shieldSpwan != null)
    {
        shieldSpwan.SetActive(true);
        return;
    }
    Vector3 pos = player.transform.position;
    pos.y += 1f;
    shieldSpwan = Instantiate(shield, pos, Quaternion.identity, player.transform);
    player.enabled = false;
    StartCoroutine(PlayerColliderActive(player, 10));
    AudioManagement.BuffGainingAudio();
}

IEnumerator BuffReSpwan(float time)
{
    yield return new WaitForSeconds(time);
    shieldBuff = Instantiate(shieldVFX, transform.position, transform.rotation);
    AudioManagement.BuffSpwaningAudio();
}

IEnumerator PlayerColliderActive(Collider obj,float time)
{
    yield return new WaitForSeconds(time);
    obj.enabled = true;
}
```

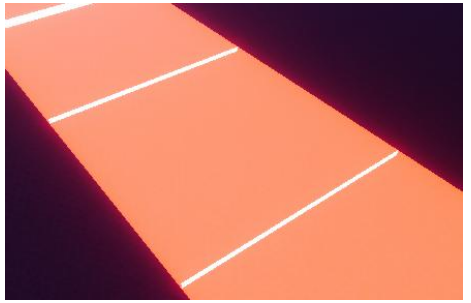


场景模块

BOSS关卡 - 陷阱 - 代码

```
void Awake()
{
    playerLayer = LayerMask.NameToLayer("Player");
    explosion = GetComponent<GameObject>();
    mat = gameObject.GetComponent<Renderer>();
}

private void Start()
{
    gameObject.GetComponent<BoxCollider>().enabled = false;//没爆炸不启动collider
}
void Update()...

public void PlayExplosion()...

private void OnTriggerEnter(Collider player)
{
    if (player.gameObject.layer == playerLayer)
    {
        player.GetComponent<CharactrStats>().characterData.currentHealth -= 10;
    }
    gameObject.GetComponent<BoxCollider>().enabled = false;//扣完血关闭collider
}

IEnumerator TrapPadColorChange(float target)
{
    float current = mat.material.color.a;
    float lerp = 0;
    while (lerp < 1)
    {
        color.a = Mathf.Lerp(current, target, lerp);
        lerp += Time.deltaTime * padColorChangeTime;
        mat.material.SetColor("_BaseColor", color);
        yield return null;
    }
    explosion = Instantiate(explosionVFX, gameObject.transform.position, gameObject.transform.rotation);
    gameObject.GetComponent<BoxCollider>().enabled = true;//爆炸时启动collider
    Destroy(explosion, 3f);
    color.a = 0;
    mat.material.SetColor("_BaseColor", color);
}
```

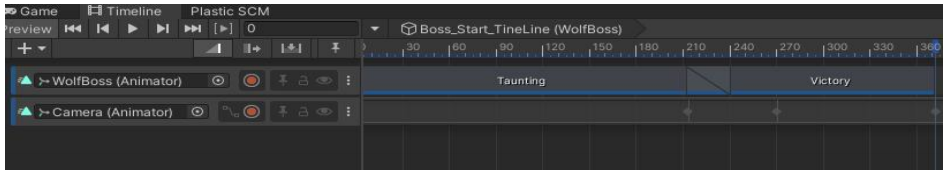
```
public Transform[] rows;
public Transform[][] trap;
public float traptime = 1;

private void Awake()
{
    TimeLinePlayControl.Instance.TimeLinePlayControlAction += Open;
}

void Start()
{

    trap = new Transform[13][];
    for (int i = 0; i < 13; i++)
    {
        trap[i] = new Transform[rows[i].childCount];
        for (int j = 0; j < trap[i].Length; j++)
        {
            trap[i][j] = rows[i].GetChild(j);
        }
    }

    StartCoroutine(TrapControlExplosion(traptime));
}



IEnumerator TrapControlExplosion(float time)
{
    for (int i = 0; i < 13; i++)
    {
        for (int j = 0; j < trap[i].Length; j++)
        {
            trap[i][j].GetComponent<ExplosionTrap>().PlayExplosion();
        }
        yield return new WaitForSeconds(time);

    }
}

public void Open()
{
    this.enabled = true;
}
```



- 场地陷阱制作
  - 透明 - 红色的颜色改变
  - 生成爆照特效
  - 设为二维数组组件，进行调用方法

场景模块

BOSS关卡 - 过场动画 - 代码




```
Unity 脚本 | 6 个引用
public class TimeLinePlayControl : MonoBehaviour
{
    public static TimeLinePlayControl Instance;
    private PlayableDirector pd;
    public Action TimeLinePlayControlAction;

    Unity 消息 | 0 个引用
    private void Awake()
    {
        Instance = this;
        TimeLinePlayControlAction = new Action(() => { });
    }

    Unity 消息 | 0 个引用
    void Start()
    {
        pd = GetComponent<PlayableDirector>();
        pd.stopped += OnStopped;
        Debug.Log(TimeLinePlayControlAction.Method.Name);
    }

    1 个引用
    private void OnStopped(PlayableDirector obj)
    {
        TimeLinePlayControlAction.Invoke();
        TimeLinePlayControlAction -= TimeLinePlayControlAction;
    }

    Unity 消息 | 0 个引用
    void Update()...
}
```

```
Unity 消息 | 0 个引用
private void Awake()
{
    TimeLinePlayControl.Instance.TimeLinePlayControlAction += Open;
}
```

```
1 个引用
public void Open()
{
    this.enabled = true;
}
```

- 过场动画制作
  - 添加时间轴
  - 添加模型，导入动画组件
  - 设置脚本控制玩家，怪物，陷阱的开关

SHOOT IT!

谢谢