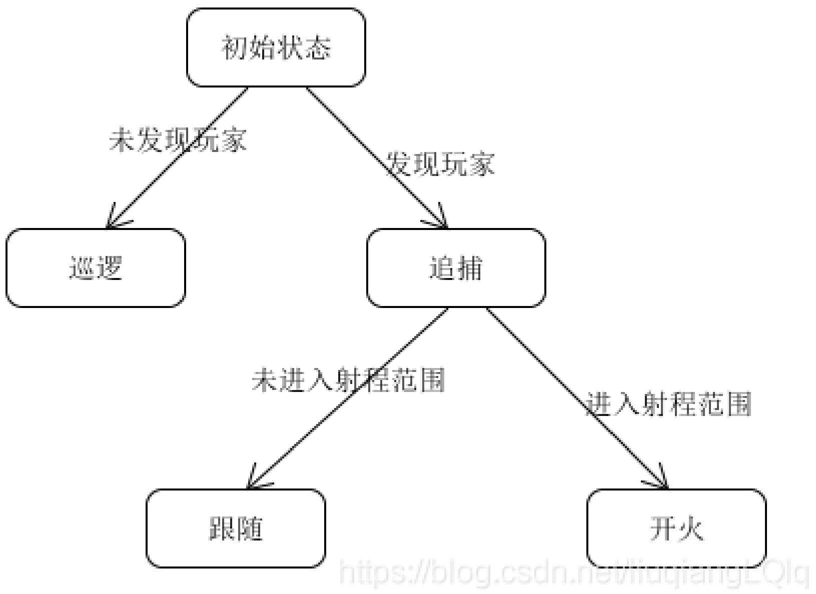
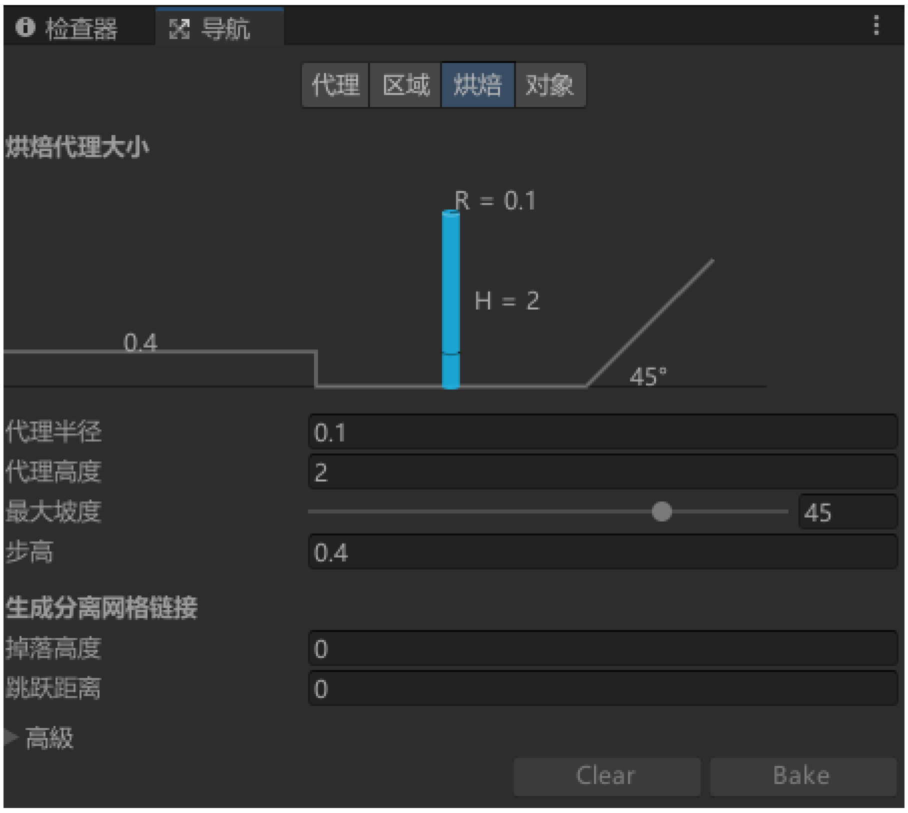
坦克对战游戏 AI设计

从商店下载游戏：“ Kawaii” Tank 或 其他坦克模型，构建 AI 对战坦克。具
体要求：
✓ 使用“感知-思考-行为”模型，建模 AI 坦克
✓ 场景中要放置一些障碍阻挡对手视线
✓ 坦克要放置一个矩阵包围盒触发器，保证 AI 坦克能使用射线探测对手方位
✓ AI 坦克必须在有目标条件下使用导航，并能绕过障碍。
✓ 实现人机对战

AI坦克状态图如下：



因为Unity3d的最终作品是供受众对3D场景进行实时操作的，就像其他3D软件场景编辑状态的操作，而一般的3D软件最终作品是将场景渲染成图片或图片序列呈现给受众的，两者的最终作品有本质的区别，简单地说，前者呈现给受众的是3D场景，后者呈现给受众的是图片或图片序列(动画)。尽管如此，两者都必须有较强的立体感和较好的光影视觉效果，否则是不被受众所接受的。下图中第一张是没经烘焙的场景，看上去苍白突兀，没有立体感和美感，第二张是经烘焙之后的的场景，立体感很强，视觉效果比第一张好得多。接着进行Bake，以便AI寻路：



烘焙代理大小

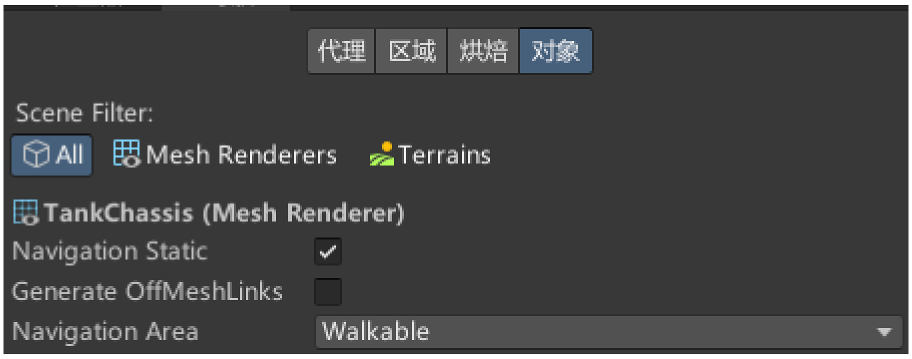
代理半径 0.1
代理高度 2
最大坡度 45
步高 0.4

生成分离网格链接

掉落高度 0
跳跃距离 0

高級

Window -> Navigation，设置游戏对象的Navigation，如果是障碍物则设置Navigation Area为not walkable：

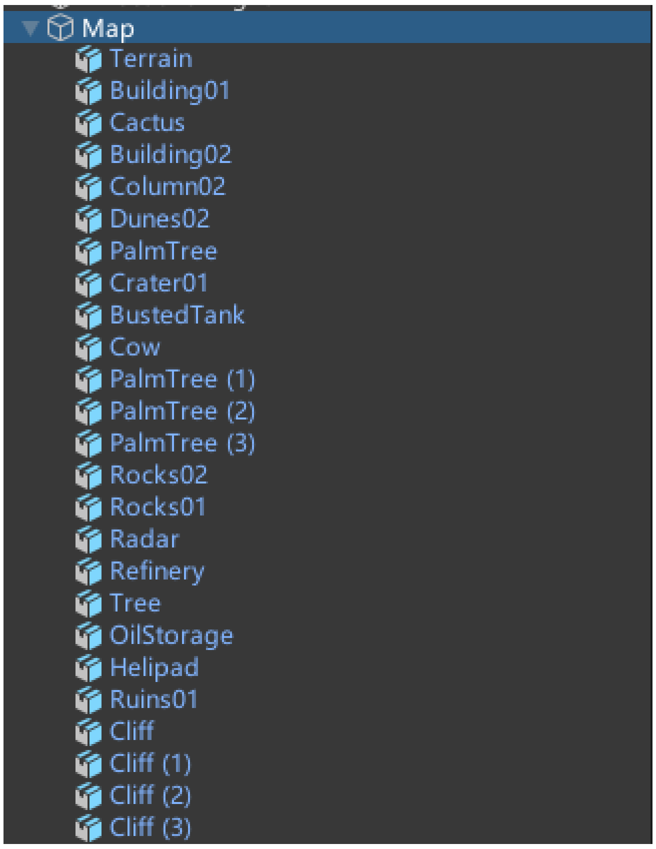


Scene Filter:

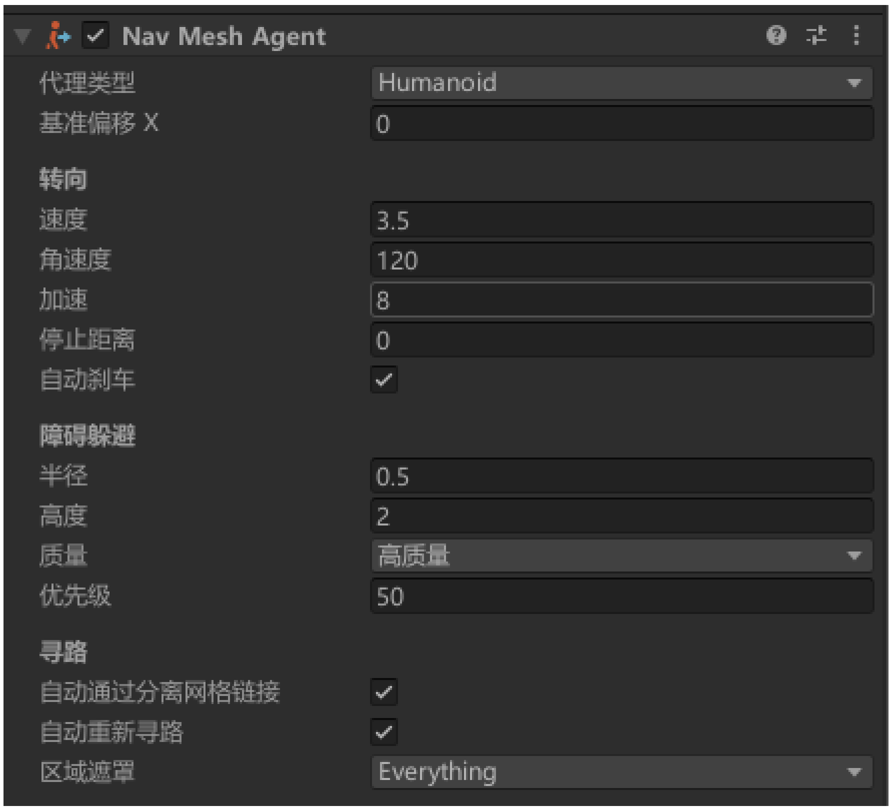
TankChassis (Mesh Renderer)

Navigation Static ✓
Generate OffMeshLinks
Navigation Area Walkable

地图构成元素：


Map
Terrain
Building01
Cactus
Building02
Column02
Dunes02
PalmTree
Crater01
BustedTank
Cow
PalmTree (1)
PalmTree (2)
PalmTree (3)
Rocks02
Rocks01
Radar
Refinery
Tree
OilStorage
Helipad
Ruins01
Cliff
Cliff (1)
Cliff (2)
Cliff (3)

设置坦克属性:


Nav Mesh Agent

代理类型	Humanoid
基准偏移 X	0
转向	
速度	3.5
角速度	120
加速	8
停止距离	0
自动刹车	✓
障碍躲避	
半径	0.5
高度	2
质量	高质量
优先级	50
寻路	
自动通过分离网格链接	✓
自动重新寻路	✓
区域遮罩	Everything

一开始AI坦克如果在自己附近没有发现玩家，则会进入巡逻状态。这里预先设置了几个点，AI坦克会随机选取一个作为目的点，并自动寻路移动到目的点，并继续下次巡逻；在这个过程中如果AI坦克发现了附近的玩家，则会进行追捕，把玩家的位置设置为AI坦克的目的点，从而使AI坦克自动向玩家方向移动；当距离进入了AI坦克的射程范围，则AI坦克会通过协程每隔一秒发射一颗子弹。

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

public class AITank : Tank {

    public delegate void recycle(GameObject tank);
    public static event recycle recycleEvent;

    private Vector3 target;
    private bool gameover;

    // 巡逻点
    private static Vector3[] points = { new Vector3(37.6f,0,0), new
Vector3(40.9f,0,39), new Vector3(13.4f, 0, 39),
        new Vector3(13.4f, 0, 21), new Vector3(0,0,0), new
Vector3(-20,0,0.3f), new Vector3(-20, 0, 32.9f),
        new Vector3(-37.5f, 0, 40.3f), new Vector3(-37.5f,0,10.4f), new
Vector3(-40.9f, 0, -25.7f), new Vector3(-15.2f, 0, -37.6f),
        new Vector3(18.8f, 0, -37.6f), new Vector3(39.1f, 0, -18.1f)
    };
    private int destPoint = 0;
    private NavMeshAgent agent;
    private bool isPatrol = false;

    private void Awake()
    {
        destPoint = UnityEngine.Random.Range(0, 13);
    }

    // Use this for initialization
    void Start () {
        setHp(100f);
        StartCoroutine(shoot());
        agent = GetComponent<NavMeshAgent>();
    }

    private IEnumerator shoot()
    {
        while (!gameover)
        {
            for(float i = 1; i > 0; i -= Time.deltaTime)
            {
                yield return 0;
            }
            // 当敌军坦克距离玩家坦克不到20时开始射击
            if(Vector3.Distance(transform.position, target) < 20)
            {
                GameObjectFactory mf =
Singleton<GameObjectFactory>.Instance;
```

```csharp
                GameObject bullet = mf.getBullet(tankType.Enemy);
                bullet.transform.position = new Vector3(transform.position.x, 1.5f, transform.position.z) + transform.forward * 1.5f;
                bullet.transform.forward = transform.forward;

                // 发射子弹
                Rigidbody rb = bullet.GetComponent<Rigidbody>();
                rb.AddForce(bullet.transform.forward * 20, ForceMode.Impulse);
            }
        }
    }

    // Update is called once per frame
    void Update () {
        gameover = GameDirector.getInstance().currentSceneController.isGameOver();
        if (!gameover)
        {
            target = GameDirector.getInstance().currentSceneController.getPlayerPos();
            if (getHp() <= 0 && recycleEvent != null)
            {//如果npc坦克被摧毁，则回收它
                recycleEvent(this.gameObject);
            }
            else
            {
                if(Vector3.Distance(transform.position, target) <= 30)
                {
                    isPatrol = false;
                    //否则向玩家坦克移动
                    agent.autoBraking = true;
                    agent.SetDestination(target);
                }
                else
                {
                    patrol();
                }
            }
        }
        else
        {
            NavMeshAgent agent = GetComponent<NavMeshAgent>();
            agent.velocity = Vector3.zero;
            agent.ResetPath();
        }
    }

    private void patrol()
    {
        if(isPatrol)
        {
            if(!agent.pathPending && agent.remainingDistance < 0.5f)
                GotoNextPoint();
        }
        else
        {
            agent.autoBraking = false;
            GotoNextPoint();
        }
        isPatrol = true;
    }

    private void GotoNextPoint()
    {
        agent.SetDestination(points[destPoint]);
        destPoint = (destPoint + 1) % points.Length;
    }
}
```

子弹类的要点在于通过OnCollisionEnter事件判断在子弹碰撞到其他物体时，爆炸范围内的所有碰撞体对象，如果子弹是AI坦克发射的并且碰撞体为玩家，则玩家坦克会扣血，子弹失活回收；如果子弹是玩家发射并且碰撞体是AI坦克，则AI坦克扣血。还要注意当子弹落地时（通过transform.position.y < 0 判断）应该把子弹回收。

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

public class Bullet : MonoBehaviour {
    // 子弹伤害半径
    public float explosionRadius = 3f;
    private tankType type;

    public void setTankType(tankType type)
    {
        this.type = type;
    }

    private void Update()
    {
        if(this.transform.position.y < 0 && this.gameObject.activeSelf)
        {
            GameObjectFactory mf = Singleton<GameObjectFactory>.Instance;
            // 落地爆炸
            ParticleSystem explosion = mf.getPs();
            explosion.transform.position = transform.position;
            explosion.Play();
            mf.recycleBullet(this.gameObject);
        }
    }

    void OnCollisionEnter(Collision other)
    {
        // 获得单实例工厂
        GameObjectFactory mf = Singleton<GameObjectFactory>.Instance;
        ParticleSystem explosion = mf.getPs();
        explosion.transform.position = transform.position;

        // 获取爆炸范围内的所有碰撞体
        Collider[] colliders = Physics.OverlapSphere(transform.position, explosionRadius);
        for(int i = 0; i < colliders.Length; i++)
        {
            if(colliders[i].tag == "tankPlayer" && this.type == tankType.Enemy || colliders[i].tag == "tankEnemy" && this.type == tankType.Player)
            {
                // 根据击中坦克与爆炸中心的距离计算伤害值
                float distance = Vector3.Distance(colliders[i].transform.position, transform.position);//被击中坦克与爆炸中心的距离
                float hurt = 100f / distance;
                float current = colliders[i].GetComponent<Tank>().getHp();
                colliders[i].GetComponent<Tank>().setHp(current - hurt);
            }
        }

        explosion.Play();
        if (this.gameObject.activeSelf)
        {
            mf.recycleBullet(this.gameObject);
        }
    }
}
```

为了实现一个比较好的游戏体验，实现一个MainCameraControl来控制主摄像机的移动跟随效果，并且能够通过游戏场景中所有坦克的距离大小来设置摄像机的Size。

```
public class MainCameraControl : MonoBehaviour {

    public float m_DampTime = 0.2f;                  // 相机refocus的时间
    public float m_ScreenEdgeBuffer = 4f;            // 最靠近边界的坦克与边界之间的缓冲大小
    public float m_MinSize = 6.5f;                   // 相机Size最小值
    [HideInInspector] public List<Transform> m_Targets; // 保存所有坦克的transform


    private Camera m_Camera;
    private float m_ZoomSpeed;
    private Vector3 m_MoveVelocity;
    private Vector3 m_DesiredPosition;


    private void Awake()
    {
        m_Camera = Camera.main;
    }

    public void setTarget(Transform transform)
    {
        m_Targets.Add(transform);
    }


    private void FixedUpdate()
    {
        // 把相机移动到希望的位置
        Move();
        // 改变相机size
        Zoom();
    }


    private void Move()
    {
        FindAveragePosition();
        transform.position = Vector3.SmoothDamp(transform.position, m_DesiredPosition, ref m_MoveVelocity, m_DampTime);
    }

    // 计算平均位置
    private void FindAveragePosition()
    {
        Vector3 averagePos = new Vector3();
        int numTargets = 0;

        for (int i = 0; i < m_Targets.Count; i++)
        {
            if (!m_Targets[i].gameObject.activeSelf)
                continue;
            averagePos += m_Targets[i].position;
            numTargets++;
        }

        if (numTargets > 0)
            averagePos /= numTargets;

        averagePos.y = transform.position.y;
        m_DesiredPosition = averagePos;
    }


    private void Zoom()
    {
        float requiredSize = FindRequiredSize();
        m_Camera.orthographicSize = Mathf.SmoothDamp(m_Camera.orthographicSize, requiredSize, ref m_ZoomSpeed, m_DampTime);
    }

    // 计算合适的Size
    private float FindRequiredSize()
    {
        Vector3 desiredLocalPos = transform.InverseTransformPoint(m_DesiredPosition);
        float size = 0f;

        for (int i = 0; i < m_Targets.Count; i++)
        {
            if (!m_Targets[i].gameObject.activeSelf)
                continue;

            Vector3 targetLocalPos = transform.InverseTransformPoint(m_Targets[i].position);
            Vector3 desiredPosToTarget = targetLocalPos - desiredLocalPos;

            size = Mathf.Max(size, Mathf.Abs(desiredPosToTarget.y));
```

```
                size = Mathf.Max(size, Mathf.Abs(desiredPosToTarget.x) /
m_Camera.aspect);
            }

            size += m_ScreenEdgeBuffer;
            size = Mathf.Max(size, m_MinSize);

            return size;
        }

        // 初始化相机
        public void SetStartPositionAndSize()
        {
            FindAveragePosition();
            transform.position = m_DesiredPosition;
            m_Camera.orthographicSize = FindRequiredSize();
        }
}
```

场记SceneController内容也比较简单，主要负责通知工厂初始化各种游戏对象，如player、AI坦克等，并初始化主摄像机，然后实现IUserAction接口中声明的函数即可。

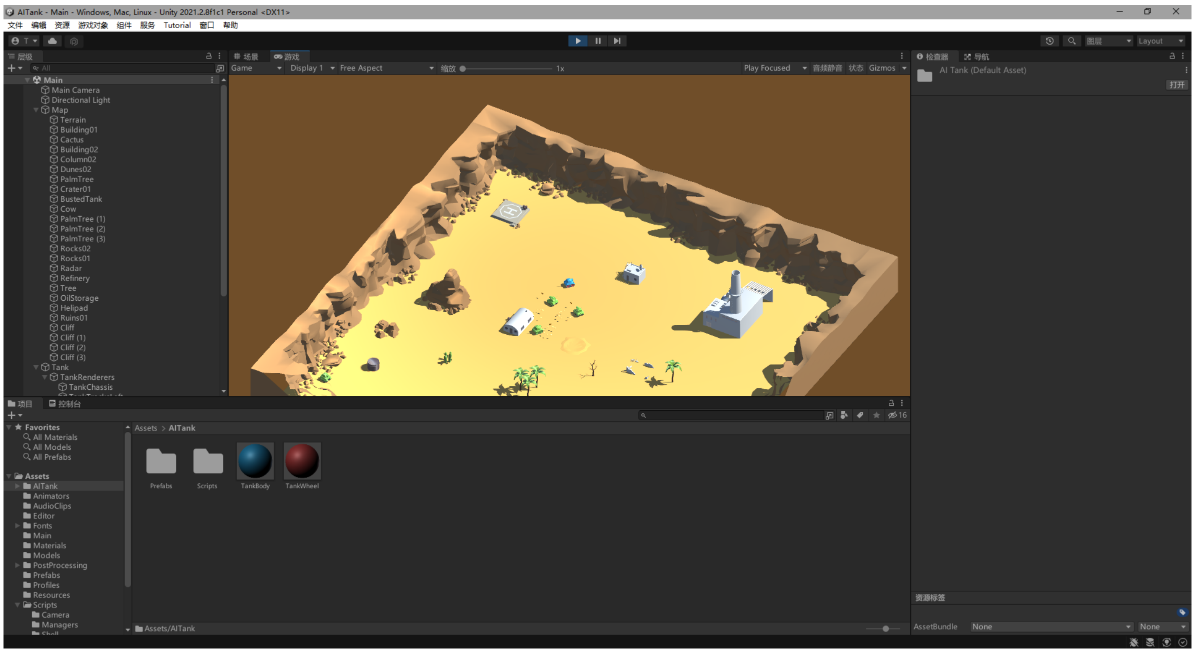
```
public class SceneController : MonoBehaviour, IUserAction {

    public GameObject player;
    private bool gameOver = false;
    private int enemyCount = 6;
    private GameObjectFactory mf;
    private MainCameraControl cameraControl;


    private void Awake()
    {
        GameDirector director = GameDirector.getInstance();
        director.currentSceneController = this;
        mf = Singleton<GameObjectFactory>.Instance;
        player = mf.getPlayer();
        cameraControl = GetComponent<MainCameraControl>();
        cameraControl.setTarget(player.transform);
    }

    // Use this for initialization
    void Start () {
        for(int i = 0; i < enemyCount; i++)
        {
            GameObject gb = mf.getTank();
            cameraControl.setTarget(gb.transform);
        }
        Player.destroyEvent += setGameOver;

        // 初始化相机位置
        cameraControl.SetStartPositionAndSize();
    }

    // 更新相机位置
    void Update () {
        Camera.main.transform.position = new
Vector3(player.transform.position.x, 15, player.transform.position.z);
    }

    public Vector3 getPlayerPos()
    {
        return player.transform.position;
    }

    public bool isGameOver()
    {
        return gameOver;
    }
    public void setGameOver()
    {
        gameOver = true;
    }

    public void moveForward()
    {
        player.GetComponent<Rigidbody>().velocity = player.transform.forward
* 20;
    }
    public void moveBackward()
    {
        player.GetComponent<Rigidbody>().velocity = player.transform.forward
* -20;
    }

    public void turn(float offsetX)
    {
        float y = player.transform.localEulerAngles.y + offsetX * 5;
        float x = player.transform.localEulerAngles.x;
        player.transform.localEulerAngles = new Vector3(x, y, 0);
    }

    public void shoot()
    {
        GameObject bullet = mf.getBullet(tankType.Player);
        // 设置子弹位置
        bullet.transform.position = new Vector3(player.transform.position.x,
1.5f, player.transform.position.z) + player.transform.forward * 1.5f;
        // 设置子弹方向
        bullet.transform.forward = player.transform.forward;

        // 发射子弹
        Rigidbody rb = bullet.GetComponent<Rigidbody>();
        rb.AddForce(bullet.transform.forward * 20, ForceMode.Impulse);
    }
}
```

通过单实例工厂GameObjectFactory来统一管理玩家player、 AI坦克、子弹、爆炸粒子系统等游戏对象，通过Dictionary来维护。

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

```csharp

public enum tankType : int { Player, Enemy }

public class GameObjectFactory : MonoBehaviour {
    // 玩家
    public GameObject player;
    // npc
    public GameObject tank;
    // 子弹
    public GameObject bullet;
    // 爆炸粒子系统
    public ParticleSystem ps;

    private Dictionary<int, GameObject> usingTanks;
    private Dictionary<int, GameObject> freeTanks;

    private Dictionary<int, GameObject> usingBullets;
    private Dictionary<int, GameObject> freeBullets;

    private List<ParticleSystem> psContainer;

    private void Awake()
    {
        usingTanks = new Dictionary<int, GameObject>();
        freeTanks = new Dictionary<int, GameObject>();
        usingBullets = new Dictionary<int, GameObject>();
        freeBullets = new Dictionary<int, GameObject>();
        psContainer = new List<ParticleSystem>();
    }

    // Use this for initialization
    void Start () {
        //回收坦克的委托事件
        AITank.recycleEvent += recycleTank;
    }

    public GameObject getPlayer()
    {
        return player;
    }

    public GameObject getTank()
    {
        if(freeTanks.Count == 0)
        {
            GameObject newTank = Instantiate<GameObject>(tank);
            usingTanks.Add(newTank.GetInstanceID(), newTank);
            //在一个随机范围内设置坦克位置
            newTank.transform.position = new Vector3(Random.Range(-100, 
100), 0, Random.Range(-100, 100));
            return newTank;
        }
        foreach (KeyValuePair<int, GameObject> pair in freeTanks)
        {
            pair.Value.SetActive(true);
            freeTanks.Remove(pair.Key);
            usingTanks.Add(pair.Key, pair.Value);
            pair.Value.transform.position = new Vector3(Random.Range(-100, 100), 0, Random.Range(-100, 100));
            return pair.Value;
        }
        return null;
    }

    public GameObject getBullet(tankType type)
    {
        if (freeBullets.Count == 0)
        {
            GameObject newBullet = Instantiate(bullet);
            newBullet.GetComponent<Bullet>().setTankType(type);
            usingBullets.Add(newBullet.GetInstanceID(), newBullet);
            return newBullet;
        }
        foreach (KeyValuePair<int, GameObject> pair in freeBullets)
        {
            pair.Value.SetActive(true);
            pair.Value.GetComponent<Bullet>().setTankType(type);
            freeBullets.Remove(pair.Key);
            usingBullets.Add(pair.Key, pair.Value);
            return pair.Value;
        }
        return null;
    }

    public ParticleSystem getPs()
    {
        for(int i = 0; i < psContainer.Count; i++)
        {
            if (!psContainer[i].isPlaying) return psContainer[i];
        }
        ParticleSystem newPs = Instantiate<ParticleSystem>(ps);
        psContainer.Add(newPs);
        return newPs;
    }

    public void recycleTank(GameObject tank)
    {
        usingTanks.Remove(tank.GetInstanceID());
        freeTanks.Add(tank.GetInstanceID(), tank);
        tank.GetComponent<Rigidbody>().velocity = new Vector3(0, 0, 0);
        tank.SetActive(false);
    }

    public void recycleBullet(GameObject bullet)
    {
        usingBullets.Remove(bullet.GetInstanceID());
        freeBullets.Add(bullet.GetInstanceID(), bullet);
        bullet.GetComponent<Rigidbody>().velocity = new Vector3(0, 0, 0);
        bullet.SetActive(false);
    }
}

```

最终呈现效果：


AITank - Main - Windows, Mac, Linux - Unity 2021.2.8f1c1 Personal <DX11>
文件 编辑 资源 游戏对象 组件 服务 Tutorial 窗口 帮助
图层
Layout
层级
场景
游戏
Game
Display 1
Free Aspect
缩放
1x
Play Focused
音频静音
状态
Gizmos
检查器
导航
AI Tank (Default Asset)
打开
Main
Main Camera
Directional Light
Map
Terrain
Building01
Cactus
Building02
Column02
Dunes02
PalmTree
Crater01
BustedTank
Cow
PalmTree (1)
PalmTree (2)
PalmTree (3)
Rocks02
Rocks01
Radar
Refinery
Tree
OilStorage
Helipad
Ruins01
Cliff
Cliff (1)
Cliff (2)
Cliff (3)
Tank
TankRenderers
TankChassis
项目
控制台
Favorites
All Materials
All Models
All Prefabs
Assets
AITank
Animators
AudioClips
Editor
Fonts
Main
Materials
Models
PostProcessing
Prefabs
Profiles
Resources
Scripts
Camera
Managers
Assets > AITank
Prefabs
Scripts
TankBody
TankWheel
16
Assets/AITank
资源标签
AssetBundle
None
None