一

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;//引入命名空间using UnityEngine.AI; 导航网格系统命名空间

public class EnemyControl : MonoBehaviour

{

private NavMeshAgent enemyAgent;//声明寻路组件，Unity 内置的寻路组件，自动计算并遵循导航网格路径

private GameObject player;//设置追踪对象——玩家

public GameObject wayPointParent;//所有巡逻点的父对象

public Transform[] wayPoints;//获取各个巡逻点的位置

public int nextIndex = 0;//巡逻数组下标索引

public float minDistance = 10;//追击范围，敌人开始追击玩家的距离阈值

void Start()

{

enemyAgent = this.GetComponent<NavMeshAgent>();//获取寻路组件

player = GameObject.FindGameObjectWithTag("Player");//通过标签查找场景中标签为"Player"的玩家对象

if (wayPointParent)//检查wayPointParent是否被赋值（不为null)

{

wayPoints = wayPointParent.GetComponentsInChildren<Transform>();

SetNextDestination();//获取父物体及其所有子物体上的Transform组件,返回结果包含父物体自身的Transform（索引 0）,子物体层级不限（递归查找所有后代）

Debug.Log(wayPoints.Length);//测试

SetNextDestination();

}

}

void Update()

{

if (Vector3.Distance(enemyAgent.transform.position, player.transform.position) < minDistance)

{

enemyAgent.destination = player.transform.position; //追踪玩家，当设置新的destination时，设置 AI 的目标位置，NavMeshAgent会计算路径后，自动沿路径移动。

}

else if (!enemyAgent.pathPending && enemyAgent.remainingDistance < 0.2f)//路径计算已完成，且距离目标点足够近（阈值 2 米）,则切换到下一个巡逻点

{

SetNextDestination();

}

}

private void SetNextDestination()//设置下一个巡逻点

{

if (wayPoints.Length <= 1) return;//如果巡逻点小于2，则不进行巡逻

enemyAgent.destination = wayPoints[nextIndex].position;//

nextIndex = (nextIndex + 1) % wayPoints.Length; //循环索引，使用取模运算（%）实现循环巡逻，索引范围为 0~3，例如，当nextIndex=3时，(3+1)%4=0回到第一个点

}

private void OnCollisionEnter(Collision collision)

{

if (collision.collider.tag == "Bullet")//敌人与带有 "Bullet" 标签的物体发生碰撞

{

Destroy(collision.collider.gameObject); //销毁子弹对象

Destroy(gameObject); //销毁自身

ScoreManager.score++;//通过ScoreManager类的静态变量score增加分数

}

}

}

二

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour

{

public GameObject enemy;// 敌人预制体

public GameObject wayPointParent;// 巡逻点父物体

public Transform[] wayPoints;// 巡逻点数组

public int bornCount = 0;//已生成敌人数量

public int limitCount = 10;//场景敌人限制数量， 敌人数量上限

void Start()

{

wayPoints = wayPointParent.GetComponentsInChildren<Transform>();//获取父对象wayPointParent本身及所有子对象的Transfrom（作为敌人生成点）

StartCoroutine("BornEnemy");//开启BornEnemy协程

}

void Update()

{

}

IEnumerator BornEnemy()//BornEnemy协程

{

while (bornCount < limitCount)//当已生成敌人数量小于场景敌人数量上限

{

int index = Random.Range(0, wayPoints.Length);//敌人生出点随机，index取值范围：0，1，2，3

GameObject newEnemy = Instantiate(enemy, wayPoints[index].position, enemy.transform.rotation);//实例化敌人及地点

newEnemy.GetComponent<EnemyControl>().nextIndex = index;

bornCount += 1;

// yield return new WaitForSeconds(1);//等1秒

}

}

public void RetryBtn()

{

//SceneManager.GetActiveScene()：获取当前正在运行的场景,返回值为Scene 对象（包含场景的各种属性）

// scene.buildIndex:获取场景在 Build Settings 中的索引值,索引规则：在 File → Build Settings 中按顺序排列第一个场景索引为 0，依次递增

//SceneManager.LoadScene(index)：加载指定索引的场景

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);//通过场景索引获取当前场景并重新加载,场景必须已添加到 Build Settings 中才能被加载

}

public void GameOverBtn()

{

Application.Quit();//退出应用，仅在 Play Mode 下模拟退出，不会真正关闭 Unity 编辑器

}

}

三

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class HealthController : MonoBehaviour

{

public float PH = 100;//当前血量

private float maxPH = 100;//血条最大值

public Slider PH\_Slider;//滑动条， 血条UI组件

public GameObject FailUI;//FailUI是空物体父对象

void Start()

{

PH = maxPH;// 初始化生命值为最大值

if (PH\_Slider)//如果PH\_Slider不为空

{

// 下方设置血条UI值（按百分比计算）\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PH\_Slider.value = PH / maxPH \* PH\_Slider.maxValue;

}

FailUI.SetActive(false);// 隐藏失败界面

}

void Update()

{

}

private void OnCollisionEnter(Collision collision)

{

//当角色与标签为 "Enemy" 的物体发生碰撞时， 受到10点伤害

if (collision.collider.tag == "Enemy") // 如果碰撞到的游戏对象的标签为 Enemy

{

Damage(10);

}

}

public void Damage(float damage)//伤害方法

{

if (PH > 0)//如果血量大于0

{

PH -= damage;//减血

PH\_Slider.value = PH / maxPH \* PH\_Slider.maxValue;// 更新血条显示

}

else

{

//下方显示失败界面

FailUI.SetActive(true);

Time.timeScale = 0;

this.GetComponent<playerController>().enabled = false;// 禁用玩家控制

}

}

}

四

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class playerController : MonoBehaviour

{

public CharacterController player;//Unity 内置的角色移动组件，比 Rigidbody 更适合玩家控制（无需处理复杂物理）

public float moveSpeed = 10;//移动速度

public float rotateSpeed = 180;//旋转速度

[Range(1,2)] public float rotateRatio = 1;//旋转敏捷度（在Inspector 中显示为滑动条，1到2之间）

public Transform player\_Trans;//Player位置

public Transform MainCamera\_Trans;//相机位置

private float x\_rotateOffset;//沿着x轴旋转偏移量，相机俯仰角度（仰头，低头）

void Start()

{

player = GetComponent<CharacterController>();//获取游戏对象上的CharacterController组件

}

void Update()

{

PlayerMovement();//每帧调用移动控制方法

PlayerRotateControl();//每帧调用旋转控制方法

}

private void PlayerMovement()//移动控制方法

{

Vector3 motionValue = Vector3.zero;// 将motionValue初始化为三维零向量new Vector3(0, 0, 0)

float h = Input.GetAxis("Horizontal");

float v = Input.GetAxis("Vertical");

motionValue += this.transform.forward \* moveSpeed \* v \* Time.deltaTime;//transform.forward基于角色自身的坐标系（不是世界坐标系）Z轴正向，长度为1

motionValue += this.transform.right \* moveSpeed \* h \* Time.deltaTime;//transform.right基于角色自身的坐标系（不是世界坐标系）X轴正向，长度为1

player.Move(motionValue);//调用Move（）函数，将motionValue参数传递给Move，motionValue`是一个相对于当前位置的偏移量（位移向量），而不是目标位置。

}

private void PlayerRotateControl()//旋转控制方法 注意：旋转分离（玩家物体控制左右旋转（Y轴）;相机作为子物体控制俯仰（X轴））

{

float mouseX = Input.GetAxis("Mouse X");// 鼠标左右移动（-1到1）

float mouseY = Input.GetAxis("Mouse Y");// 鼠标上下移动（-1到1）

player\_Trans.Rotate(Vector3.up \* mouseX \* rotateSpeed \* rotateRatio \* Time.deltaTime);//玩家左右方向的旋转（绕Y轴）

//鼠标向上移动（正`mouseY`）对应抬头，在Unity中，抬头是绕X轴的负旋转（角度减小）。而低头是绕X轴的正旋转（角度增大）。所以用`-=`将鼠标移动与角度变化关联起来。

x\_rotateOffset -= mouseY \* rotateSpeed \* rotateRatio \* Time.deltaTime;// 相机俯仰旋转（绕X轴）

x\_rotateOffset = Mathf.Clamp(x\_rotateOffset, -60, 60);//限制角度

//设置相机的局部旋转，局部旋转（localRotation）是指物体相对于其父物体的坐标系进行的旋转

//假设有一个玩家（Player），其旋转为 (0, 90, 0)（绕世界Y轴旋转90度）, 相机（Camera）是Player的子物体，其局部旋转为 (30, 0, 0)（绕Player的局部X轴旋转30度）,

//那么相机的世界旋转将是：先跟随Player旋转90度（Y轴），然后再绕Player的X轴旋转30度。

MainCamera\_Trans.localRotation = Quaternion.Euler(new Vector3(x\_rotateOffset, MainCamera\_Trans.localEulerAngles.y, MainCamera\_Trans.localEulerAngles.z));

}

}

五

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class ScoreManager : MonoBehaviour

{

public static int score;// 全局分数（静态变量）

private int displayScore; // UI显示的分数

public Text scoreUI; // 分数显示UI组件

private int enemyNum; // 敌人总数

public GameObject winUI; // winUI是空物体父对象，胜利界面

bool isWin = false; // 胜利状态标志

void Start()

{

score = 0; // 重置分数

displayScore = 0; // 重置显示分数

enemyNum = GetComponent<GameManager>().limitCount;// 获取敌人总数

winUI.SetActive(false);// 隐藏胜利界面

}

void Update()

{

// 更新分数显示

// if (score != displayScore)

{

// displayScore = score;

// scoreUI.text = displayScore.ToString();

}

// 检查胜利条件

if (score == enemyNum)

{

winUI.SetActive(true);// 显示胜利界面

isWin = true;// 标记为胜利状态

Win();// 执行胜利逻辑

}

}

void Win()

{

if (isWin == true)

{

GameObject player = GameObject.Find("Player");

player.GetComponent<playerController>().enabled = false;//禁用玩家控制器，防止胜利后继续移动

}

}

}

六

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class WeaponController : MonoBehaviour

{

public Transform bulletStartPoint;//开火位置

public GameObject bullet;//子弹物体

public float bulletStartSpeed;//子弹发射速度

void Start()

{

}

void Update()

{

// OpenFire();//开火

}

private void OpenFire()//开火方法

{

if (Input.GetMouseButtonDown(0))//当鼠标左键被点击时

{

//克隆化子弹预制体，将克隆出的游戏对象进行位置设定及旋转信息设定，然后赋给newBullet

GameObject newBullet = Instantiate(bullet, bulletStartPoint.position, bulletStartPoint.rotation);

//newBullet.transform.forward表示子弹当前的前向方向，这是一个单位向量（模长为 1）,仅表示方向，不包含距离信息，是局部 Z 轴向前在世界坐标系中的方向向量，如果需要向后方向，可使用 -transform.forward

newBullet.GetComponent<Rigidbody>().velocity = newBullet.transform.forward \* bulletStartSpeed;

Destroy(newBullet, 1);

//子弹在生成 1 秒后会被自动销毁

}

}

}