C#

基础

数组

创建数组

#region arraryCreat

//创建数组并设置大小50

int[] array = new int[50];

//random 随机数类:Next方法计算并返回随机数

Random r = new Random();

for(int i = 0;i < 50;i++)

{

array[i] = r.Next(0, 100);

}

Console.WriteLine("创建一个[50]数组,并随机放入元素");

Console.Write("请输入一个数字,设置每几个数进行一次换行:");

//Convert.ToInt32 数据类型转换:将一个数据类型进行转换

int num = Convert.ToInt32(Console.ReadLine());

int temp = 0;

for (int i = 0; i < 50; i++)

{

temp += 1;

Console.Write(array[i]+" ");

if (temp == num)

{

//\n换行 \r回车 回到旧行的第一个光标

Console.Write("\n\r");

temp = 0;

}

}

Console.ReadLine();

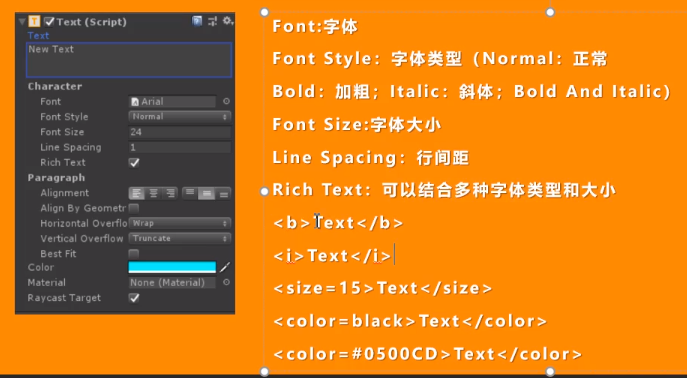
#endregion

求数组最大值

Unity

UGUI+NGUI

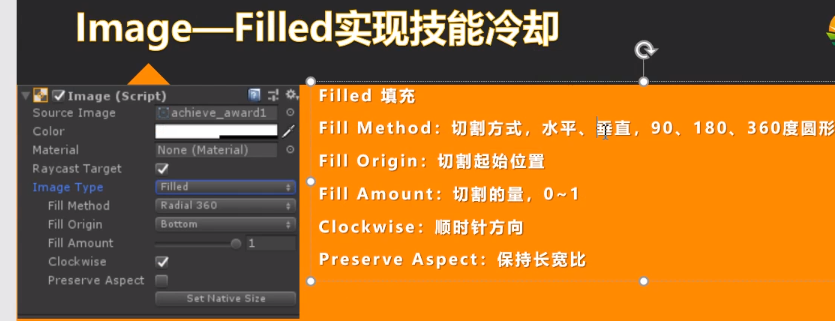
Text



Image



实现技能冷却



public class CoolDown : MonoBehaviour

{

public Image image;

public float restTime;

public float cooldown;

// Start is called before the first frame update

void Start()

{

image = GetComponent < Image>();

}

// Update is called once per frame

void Update()

{

if (restTime > 0)

{

restTime -= Time.deltaTime;

image.fillAmount = restTime / cooldown;

}

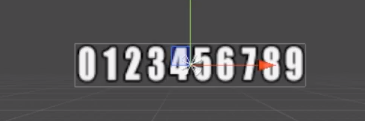
}

}

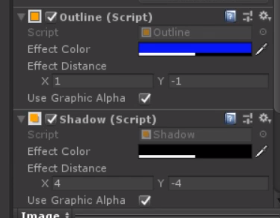
Raw Image



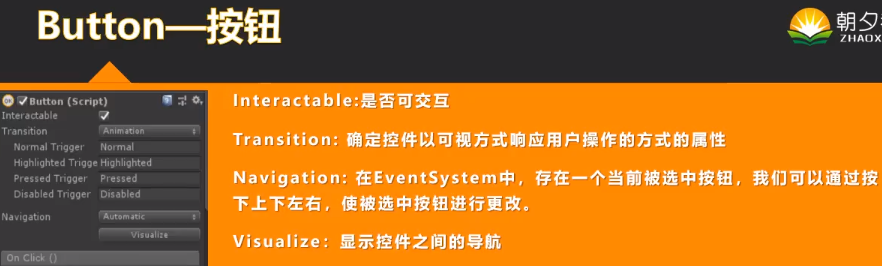
实现图片渐入



Outline描边组件 shadow阴影组件



Button



在脚本中直接订阅事件

public class subscribeBtton : MonoBehaviour

{

// Start is called before the first frame update

void Start()

{

this.gameObject.GetComponent<Button>().onClick.AddListener(OnButtonClick);

String str = "str";

//委托按钮

this.gameObject.GetComponent<Button>().onClick.AddListener(() => OnButtonClickParameter(str));

}

private void OnButtonClick()

{

Debug.Log("Click");

}

private void OnButtonClickParameter(string str)

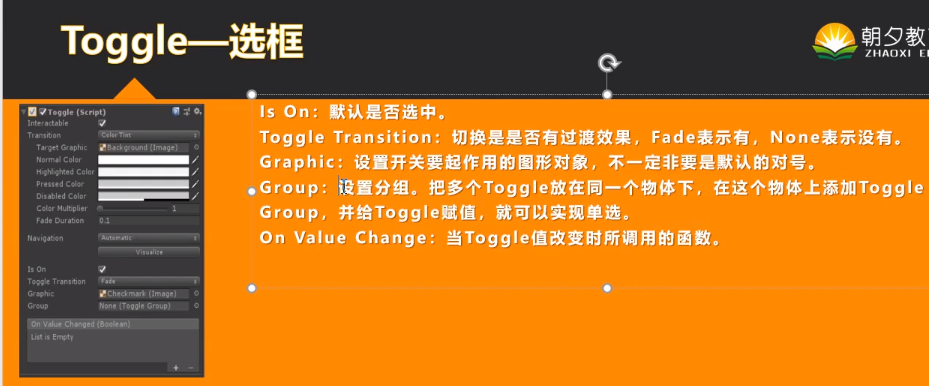
{

Debug.Log(str);

}

}

Toggle



ToggleGroup可以实现多个选框单选

需要在多个toggle的父物体上添加toggleGroup组件

void Start()

{

this.gameObject.GetComponent<Toggle>().onValueChanged.AddListener((bool change) => ChangeValue(change));

}

private void ChangeValue(bool change)

{

Debug.Log("toggle changed");

}

Slider



Scrollbar



ScrollView





DropDown



项目:3DRPG

快捷键

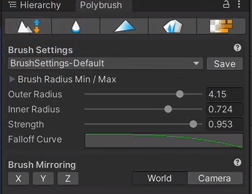
V 顶点吸附

Ctrl+shift 顶点吸附

Ctrl+shift +F 摄像机和预览界面同步

插件

PolyBrush 地形编辑



ProBuilder 更好的多边形预制体

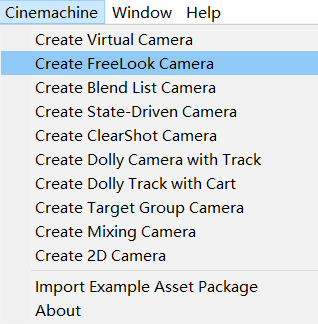


ProGrids 网格标识

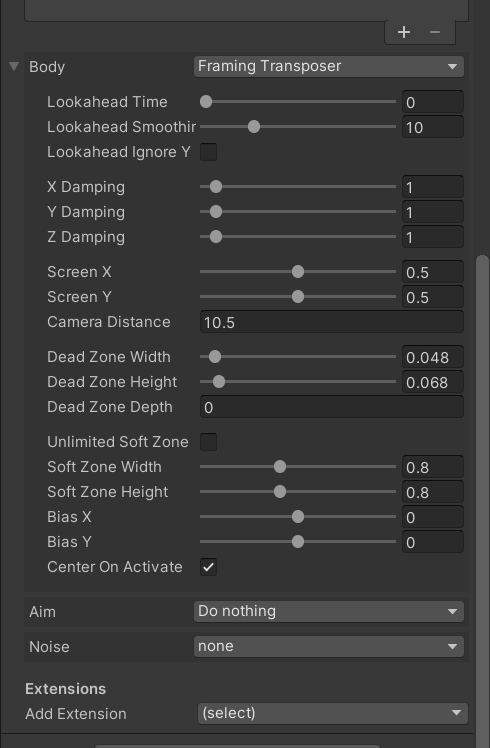
需要显示preview版本才能在package manager显示



Cinemachine 虚拟镜头,镜头跟随



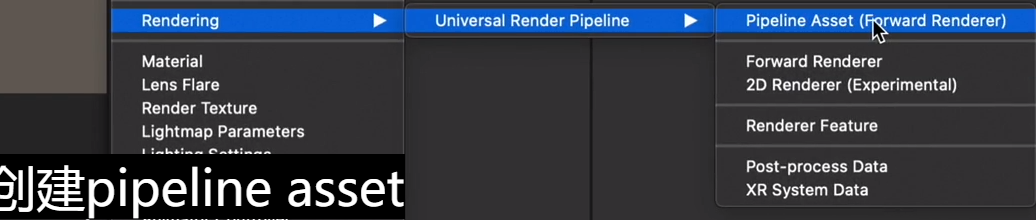
Dead zone 实现镜头缓慢跟随



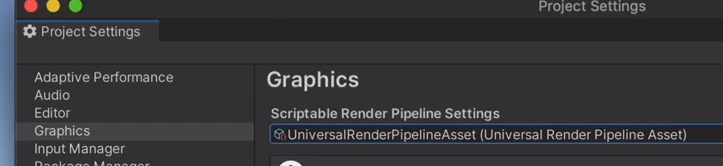
创建渲染管线

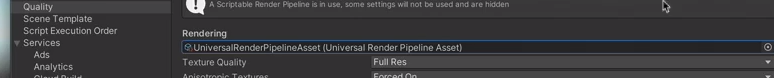
将普通项目升级成UPR渲染管线项目

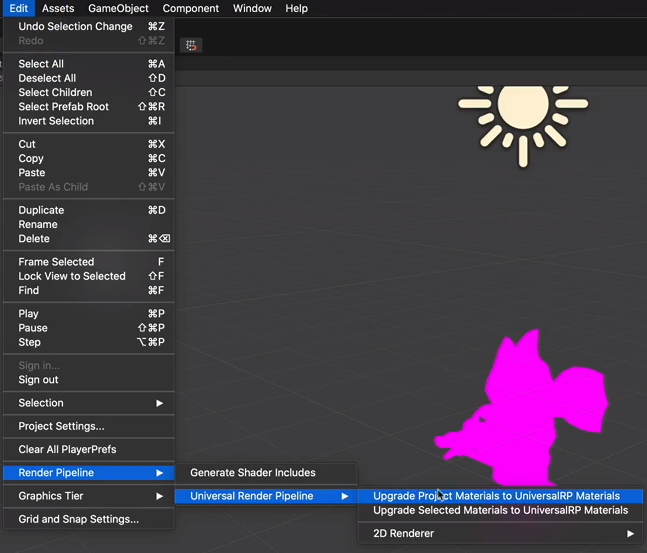
在PackageManager中找到UPR并安装



添加渲染管线



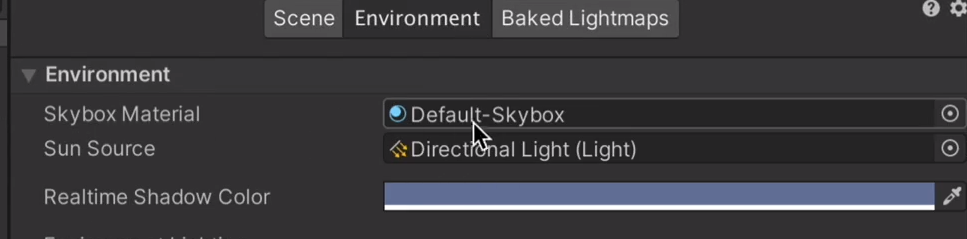




设置渲染管线

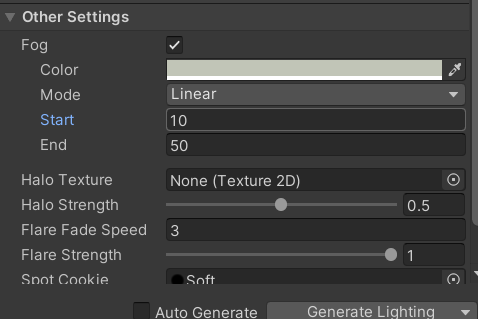
设置天空盒子和环境光



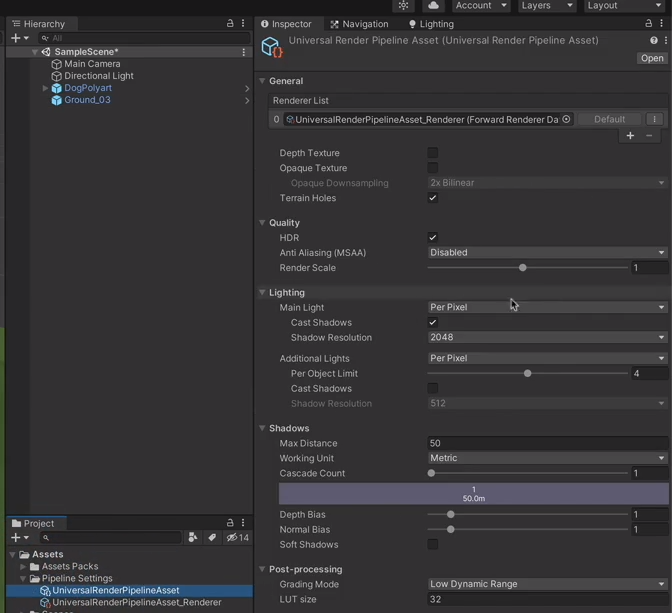


设置迷雾效果





设置阴影



设计模式

单例模式

//单例模式

public static MouseManager Instence;

private void Awake()

{

if (Instence != null)

{

Destroy(gameObject);

}

Instence = this;

}

可以在另一个文件直接调用

MouseManager.Instence.OnMouseClicked += MoveToTarget;

//泛型单例模式

public class SingleTon<T> : MonoBehaviour where T:SingleTon<T>

{

private static T instance;

public static T Instance

{

get

{

return instance;

}

}

protected virtual void Awake()

{

if (instance != null)

{

Destroy(this);

}

else

instance = (T)this;

}

protected static bool IsInstance

{

get{

return instance != null;

}

}

protected virtual void OnDestroy()

{

if (instance == this)

{

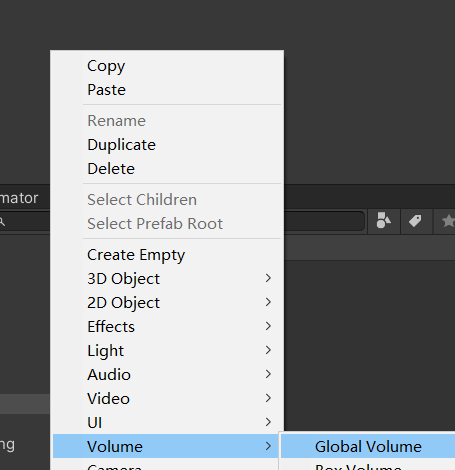
instance = null;

}

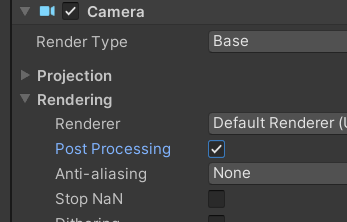
}

}

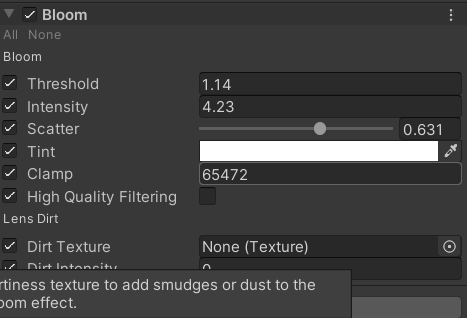
Post Processing 后处理 Global Volume



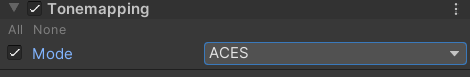
需要在主相机中设置显示



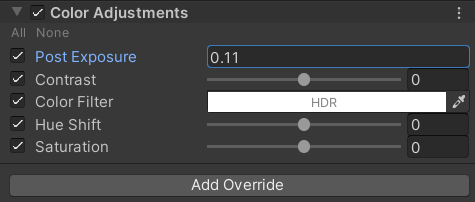
Bloom设置



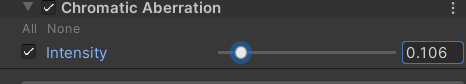
Tonemapping 动画电影设置



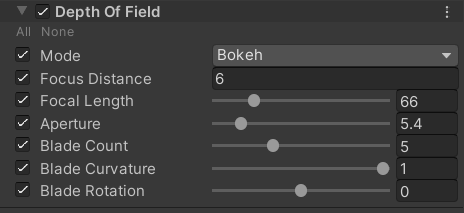
Color Adjustments 颜色调整



Chromatic Aberration 畸变效果



Depth of Field 景深效果

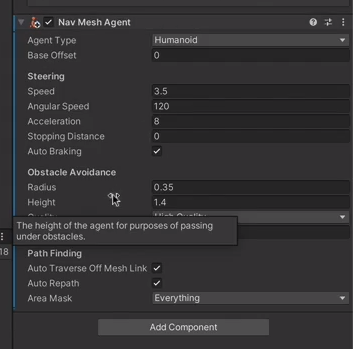


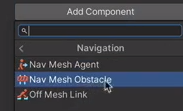
Navigation 智能导航地图烘焙

设置地面可行走区域和不可行走区域



设置人物的Agent 添加人物 Agent组件





Gizmos画出范围

public void OnDrawGizmosSelected()

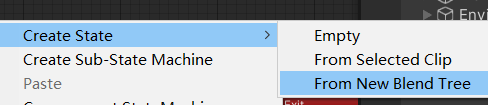
{

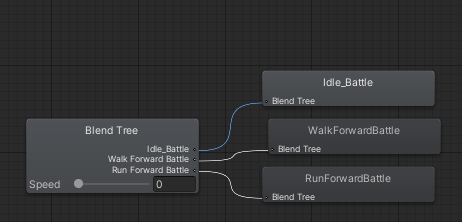
Gizmos.color = Color.yellow;

Gizmos.DrawWireSphere(transform.position, patrolRange);

}

Blend Tree 混合树动画控制器





private Animator anim;

private void Awake()

{

anim = GetComponent<Animator>();

}

private void Update()

{

SwitchAnimation();

}

public void SwitchAnimation()

{

Agent 是Navigation导航数据

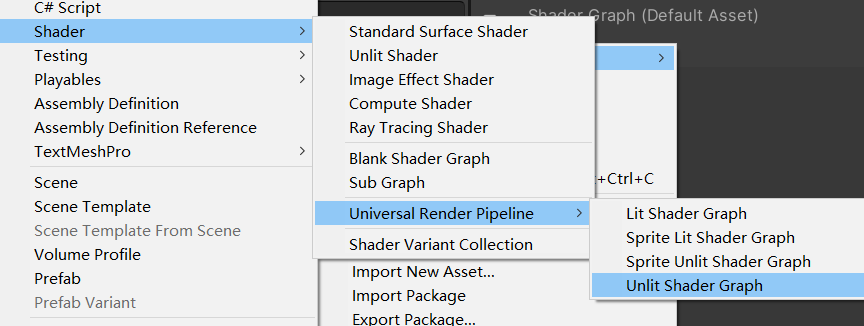
//agent.velocity.sqrMagnitude返回速度的浮点型数据

anim.SetFloat("Speed", agent.velocity.sqrMagnitude);

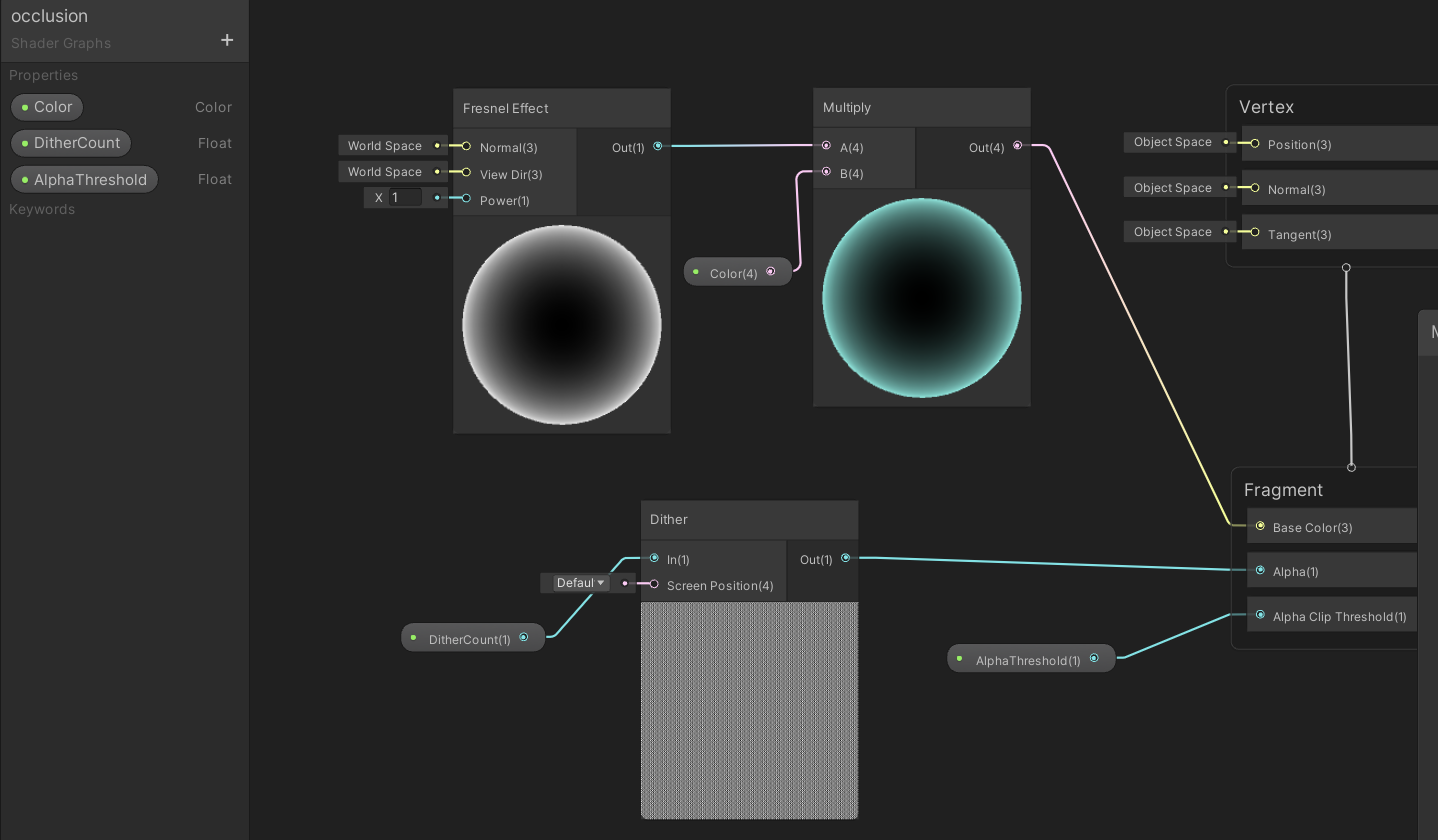
}

Shader Graph 渲染管线实现 遮挡剔除

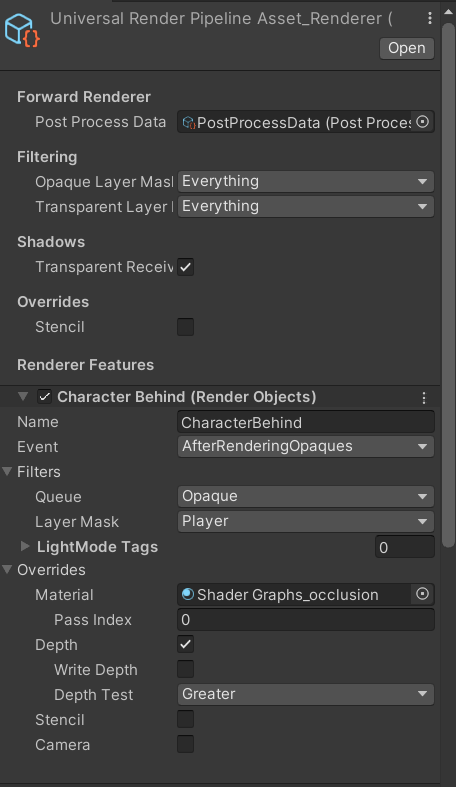
设置shader

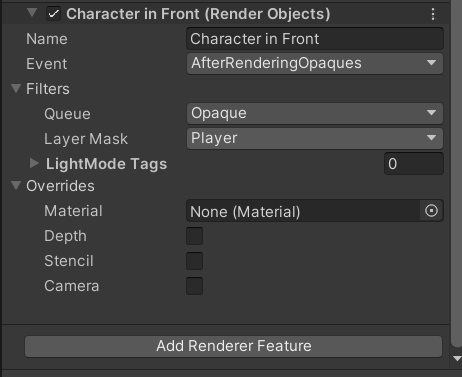


实现菲尼尔效果和栅格点效果材质



设置渲染管线实现遮挡剔除





避免射线检测,关闭mesh Collider



或者设置图层



从脚本注册组件

[RequireComponent(typeof(NavMeshAgent))]

玩家控制

鼠标控制人物移动

传入人物控制

将鼠标点击获取的值传入 destination



using UnityEngine.Events;

[System.Serializable]

//EventVector3因为不是继承MonoBehaviour 无法在引擎里显示,需要被系统序列化

public class EventVector3 : UnityEvent<Vector3> { }

public class MouseManager : MonoBehaviour

{

RaycastHit hitInfo;

public EventVector3 onMouseClicked;

// Update is called once per frame

void Update()

{

SetCursorTexture();

MouseControl();

}

void SetCursorTexture()

{

//鼠标点击的点返回射线的值

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

Debug.DrawRay(ray.origin, ray.direction \* 10, Color.yellow);

//射线碰撞到物体后返回物体信息 RaycastHit hit

if (Physics.Raycast(ray,out hitInfo))

{

//设置鼠标贴图

}

}

//鼠标控制

void MouseControl()

{

if (Input.GetMouseButtonDown(0)&& hitInfo.collider != null)

{

if (hitInfo.collider.CompareTag("Ground"))

{

//触发事件

//?.Invoke()若事件不为空就调用()中

onMouseClicked?.Invoke(hitInfo.point);

}

}

}

}

不用传入人物注册事件

using System;

public event Action<Vector3> OnMouseClicked;

**在另一个脚本中注册函数**

using UnityEngine.AI;

public class PlayerController : MonoBehaviour

{

private NavMeshAgent agent;

private void Awake()

{

agent = GetComponent<NavMeshAgent>();

}

private void Start()

{

MouseManager.Instence.OnMouseClicked += MoveToTarget;

}

public void MoveToTarget(Vector3 target)

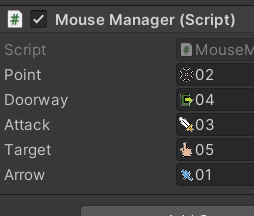
{

agent.destination = target;

}

}

更改鼠标贴图



//移动鼠标时的鼠标贴图

public Texture2D point, doorway, attack, target, arrow;

void SetCursorTexture()

{

//鼠标点击的点返回射线的值

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

Debug.DrawRay(ray.origin, ray.direction \* 10, Color.yellow,1f, false);

//射线碰撞到物体后返回物体信息 RaycastHit hit

if (Physics.Raycast(ray,out hitInfo))

{

//设置鼠标贴图

switch (hitInfo.collider.gameObject.tag)

{

case "Ground":

//设置鼠标的贴图,第二个为偏移量

Cursor.SetCursor(arrow,new Vector2(0,0), CursorMode.Auto);

break;

default:

break;

}

}

}

ScriptableObject 设置基本的人物属性

创建脚本

[CreateAssetMenu(fileName = "New Date",menuName = "Character State/Date")]

public class CharacterDate\_SO : ScriptableObject

{

[Header("State Info")]

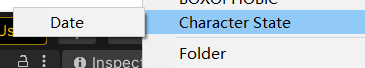
public int maxHealth;

public int currentHealth;

public int baseDefence;

public int currentDefence;

}





Properties 属性的用法 get set 实现读取数据

在CharacterState中调用攻击属性和人物的属性

public class CharacterState : MonoBehaviour

{

public CharacterDate\_SO characterDate;

public AttackDate\_SO attackDate\_SO;

[HideInInspector]

public bool isCritical;

#region Read From Date\_SO;

public int MaxHealth

{

get{ if (characterDate != null) return characterDate.maxHealth; else return 0; }

set { characterDate.maxHealth = value; }

}

public int CurrentHealth

{

get { if (characterDate != null) return characterDate. currentHealth; else return 0; }

set { characterDate. currentHealth = value; }

}

public int BaseDefence

{

get { if (characterDate != null) return characterDate.baseDefence; else return 0; }

set { characterDate.baseDefence = value; }

}

public int CurrentDefence

{

get { if (characterDate != null) return characterDate.currentDefence; else return 0; }

set { characterDate.currentDefence = value; }

}

#endregion

}**在另一个脚本中使用**

private CharacterState characterState;

private void Awake()

{

characterState = GetComponent<CharacterState>();

}

private void Start()

{

characterState.MaxHealth = 2;

}

设置攻击属性

[CreateAssetMenu(fileName = "New Date",menuName = "Character State/Date")]

public class CharacterDate\_SO : ScriptableObject

{

[Header("State Info")]

public int maxHealth;

public int currentHealth;

public int baseDefence;

public int currentDefence;

}

//玩家攻击帧伤害判定 Animation Event

public void Hit()

{

if (attackTarget != null)

{

var targetState = attackTarget.GetComponent<CharacterState>();

targetState.TakeDamage(characterState, targetState);

}

}

通过协程实现移动到敌人面前并采取攻击

建立鼠标点击事件

public event Action<GameObject> OnMouseAttackClicked;

void Update()

{

MouseControl();

}

//点击到敌人时触发事件

void MouseControl()

{

if (Input.GetMouseButtonDown(0)&& hitInfo.collider != null)

{

if (hitInfo.collider.CompareTag("Enemy"))

{

//触发事件

OnMouseAttackClicked?.Invoke(hitInfo.collider.gameObject);

}

}

}

**在另一个脚本中注册函数**

private void Start()

{

MouseManager.Instence.OnMouseAttackClicked += EventAttack;

}

//更新冷却时间

private void Update()

{

nextAttack -= Time.deltaTime;

}

//鼠标点击移动

public void MoveToTarget(Vector3 target)

{

//移动时打断攻击 停止协程

StopAllCoroutines();

//移动时保证导航不是停止状态

agent.isStopped = false;

agent.destination = target;

}

private void EventAttack(GameObject target)

{

if (target != null)

{

attackTarget = target;

//暴击判断

characterState.isCritical = UnityEngine.Random.value < characterState.attackDate.criticalChance;

//执行攻击协程

StartCoroutine(MoveToAttackTarget());

}

}

//协程判断攻击移动

IEnumerator MoveToAttackTarget()

{

//保证一定能移动

agent.isStopped = false;

//转头看向目标

transform.LookAt(attackTarget.transform);

//持续判断是否到达攻击目标点

while (Vector3.Distance(attackTarget.transform.position, transform.position) > characterState.attackDate\_SO.attackRange )

{

//移动到目标点

agent.destination = attackTarget.transform.position;

//使它持续判断

yield return null;

}

//到达目标点停止行动

agent.isStopped = true;

//进行攻击

if (nextAttack <= 0)

{

//显示动画

anim.SetTrigger("Attack");

//显示暴击的动画

anim.SetBool("Critical", characterState.isCritical);

//攻击冷却

nextAttack = characterState.attackDate.coolDown; }

}

}

敌人状态机

public enum EnemyStates { GUARD, PATROL,CHASE,DEAD}

public class EnemiesControl : MonoBehaviour

{

public EnemyStates enemyStates;

private NavMeshAgent agent;

private Animator anim;

[Header("Besic Setting")]

public float chaseRadius;

public GameObject attackTarget;

public bool isGuard;

private float speed;

private bool isWalk,isChase,isFollow;

[Header("Patrol State")]

public float patrolRange;

public Vector3 wayPoint;

private Vector3 guardPos;

private NavMeshAgent agent;

private void Awake()

{

agent = GetComponent<NavMeshAgent>();

agent = GetComponent<NavMeshAgent>();

speed = agent.speed;

guardPos = transform.position;

}

//切换敌人动画

private void SwitchAnimationState()

{

anim.SetBool("Walk",isWalk);

anim.SetBool("Chase",isChase);

anim.SetBool("Follow", isFollow);

}

//切换状态

void Start()

{

if (isGuard)

{

enemyStates = EnemyStates.GUARD;

}

else

{

enemyStates = EnemyStates.PATROL;

GetNewWayPoint();

}

}

private void Update()

{

SwitchState();

SwitchAnimationState();

}

public void SwitchState()

{

//是否找到玩家

if (FoundPlayer())

{

enemyStates = EnemyStates.CHASE;

Debug.Log("F");

}

switch (enemyStates)

{

case EnemyStates.GUARD:

break;

case EnemyStates.PATROL:

PatrolState()

break;

case EnemyStates.CHASE:

ChaseState()

break;

case EnemyStates.DEAD:

break;

default:

break;

}

}

}

//追击模式

//追击模式

public void ChaseState()

{

agent.isStopped = false;

isWalk = false;

isChase = true;

if (!FoundPlayer())

{

//回到上个状态

isFollow = false;

agent.destination = transform.position;

//脱战回到上个状态

if (remainLookAtTime > 0)

{

remainLookAtTime -= Time.deltaTime;

}

else if (isGuard)

{

enemyStates = EnemyStates.GUARD;

}

else

{

enemyStates = EnemyStates.PATROL;

}

}

else

{

//追击玩家

isFollow = true;

agent.speed = speed;

//追击

agent.destination = attackTarget.transform.position;

//重置脱战后原地等待时间

remainLookAtTime = lookAtTime;

}

//TODO:在攻击范围内攻击

if (TargetInAttackRange()||TargetInSkillRange())

{

isFollow = false;

agent.isStopped = true;

if (lastAttackTime < 0)

{

lastAttackTime = characterState.attackDate\_SO.coolDown;

//暴击判断

随机获得一个0-1的值 isCritical bool值是否暴击

characterState.isCritical = Random.value < characterState.attackDate\_SO.criticalChance;

//执行攻击

Attack();

}

}

}

//巡逻模式

public void PatrolState()

{

agent.isStopped = false;

isChase = false;

agent.speed = speed \* 0.5f;

if (Vector3.Distance(wayPoint, transform.position) <= agent.stoppingDistance)

{

isWalk = false;

GetNewWayPoint();

}

else

{

isWalk = true;

agent.destination = wayPoint;

}

}

//获取随机巡逻点

public void GetNewWayPoint()

{

//Random.Range获取两个范围内随机的值

float randomX = Random.Range(-patrolRange, patrolRange);

float randomZ = Random.Range(-patrolRange, patrolRange);

//保持y轴防止地面有起伏

Vector3 randomPoint = new Vector3(guardPos.x + randomX, transform.position.y,guardPos.z + randomZ);

//NavMesh.SamplePosition 获取导航范围可移动的点,防止随机点放到不可以移动的地方

NavMeshHit hit;

wayPoint = NavMesh.SamplePosition(randomPoint, out hit, patrolRange, 1) ? hit.position : transform.position;

}

//是否找到玩家

private bool FoundPlayer()

{

//圆形范围检测

var colliders = Physics.OverlapSphere(transform.position, chaseRadius);

foreach (var target in colliders)

{

attackTarget = target.gameObject;

if (target.CompareTag("Player"))

{

return true;

}

}

return false;

}

//执行攻击

public void Attack()

{

transform.LookAt(attackTarget.transform);

if (TargetInAttackRange())

{

//近攻击

anim.SetTrigger("Attack");

}

else if (TargetInSkillRange())

{

//远程攻击

anim.SetTrigger("Skill");

}

}

//是否进入攻击距离

public bool TargetInAttackRange()

{

if (attackTarget != null)

{

return Vector3.Distance(attackTarget.transform.position, transform.position) <= characterState.attackDate\_SO.attackRange;

}

else

return false;

}

//是否进入技能攻击距离

public bool TargetInSkillRange()

{

if (attackTarget != null)

{

return Vector3.Distance(attackTarget.transform.position, transform.position) <= characterState.attackDate\_SO.skillRange;

}

else

return false;

}

//守卫模式

public void GuardState()

{

agent.speed = speed \* 0.5f;

//isChase = false;

//返回原本位置 guardPos

agent.destination = guardPos;

//SqrMagnitude 目标点减去目前的距离 小于停止距离代表归位

if (Vector3.SqrMagnitude(guardPos - transform.position) <= agent.stoppingDistance)

{

//到达目标点停止移动动画

isWalk = false;

//转身到原本的朝向位置

transform.rotation = Quaternion.Lerp(transform.rotation, guardRotation, 0.05f);

}

}

//死亡模式

private void Update()

{

if (characterState.CurrentHealth == 0)

{

isDead = true;

}

}

//死亡模式

private void DeadState()

{

//关闭碰撞检测

coll.enabled = false;

//不继续追击 agent.radius设置为0就不会追击??

agent.radius = 0;

//一段时间后销毁

Destroy(gameObject, 60f);

}

设置GameManager 和广播接口

public class GameManager : SingleTon<GameManager>

{

public CharacterState playerState;

//创建列表加载所有敌人

List<InEndObserve> endObserves = new List<InEndObserve>();

//实现观察者模式

public void RigisterPlayer(CharacterState player)

{

playerState = player;

}

//敌人加载加入列表

public void AddObserves(InEndObserve observe)

{

endObserves.Add(observe);

}

//敌人死亡移除列表

public void RemoveObserve(InEndObserve observe)

{

endObserves.Remove(observe);

}

public void NotifyObserves()

{

foreach (var Observe in endObserves)

{

Observe.EndNotify();

}

}

}

在敌人加载时调用

//加载时

private void OnEnable()

{

GameManager.Instance.AddObserves(this);

}

//销毁后

private void OnDisable()

{

if (!GameManager.IsInstanced) return;

GameManager.Instance.RemoveObserve(this);

}

实现人物死亡,对全部敌人进行广播

public interface InEndObserve

{

void EndNotify();

}

敌人脚本

private void Update()

{

if (characterState.CurrentHealth == 0)

{

isDead = true;

}

if(!playerDead)

{

SwitchState();

SwitchAnimationState();

lastAttackTime -= Time.deltaTime;

}

}

实现接口

public void EndNotify()

{

anim.SetBool("Win", playerDead);

//停止所有行动

isChase = false;

isWalk = true;

playerDead = true;

attackTarget = null;

}

实现角色属性文件的复制

public CharacterDate\_SO templateCharacterDate;

public CharacterDate\_SO characterDate;

public AttackDate\_SO attackDate;

[HideInInspector]

public bool isCritical;

private void Awake()

{

if (templateCharacterDate != null)

{

characterDate = Instantiate(templateCharacterDate);

}

}

设置动画行为脚本

public class StopAgentt : StateMachineBehaviour

{

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

override public void OnStateEnter(Animator animator, AnimatorStateInfo stateinfo, int layerindex)

{

animator.GetComponent<NavMeshAgent>().isStopped = true;

}

// OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks

override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.GetComponent<NavMeshAgent>().isStopped = true;

}

// OnStateExit is called when a transition ends and the state machine finishes evaluating this state

override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

animator.GetComponent<NavMeshAgent>().isStopped = false;

}

// OnStateMove is called right after Animator.OnAnimatorMove()

//override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that processes and affects root motion

//}

// OnStateIK is called right after Animator.OnAnimatorIK()

//override public void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

//{

// // Implement code that sets up animation IK (inverse kinematics)

//}

}

设置击飞和眩晕效果

public int kickForce;

public void Kick()

{

if (attackTarget != null)

{

//看向目标

transform.LookAt(attackTarget.transform.position);

//计算击飞方向

Vector3 direction = attackTarget.transform.position - transform.position;

//Normalize()使值为 1 或 -1

direction.Normalize();

//保持不移动

attackTarget.GetComponent<NavMeshAgent>().isStopped = true;

//添加击飞力

attackTarget.GetComponent<NavMeshAgent>().velocity = direction \* kickForce;

//启动眩晕动画

attackTarget.GetComponent<Animator>().SetTrigger("Dizzy");

}

}

扩展方法 潜入开发 使用点积判断目标是否在面前

public static class ExtensionMethod

{

private static float dotThreshold = 0.5f;

public static bool IsFacingTarget(this Transform transform, Transform target)

{

var vectorToTarget = target.position - transform.position;

vectorToTarget.Normalize();

float dot = Vector3.Dot(transform.position, vectorToTarget);

return dot >= dotThreshold;

}

}

#region Combat Set

//受伤伤害

public void TakeDamage(CharacterState attacker,CharacterState defener)

{

//Mathf.Max 取最大值,保证伤害不为负数

int damage = Mathf.Max(attacker.CurrentDamage() - defener.CurrentDefence,0);

//更新生命值

CurrentHealth = Math.Max(CurrentHealth - damage, 0);

//受伤动画显示

if (attacker.isCritical)

{

defener.GetComponent<Animator>().SetTrigger("Hit");

}

//TODO:更新血量的UI显示

//todo:敌人死亡后更新经验值

}

//石头攻击

public void TakeDamage(int damage,CharacterState defener)

{

int damageRock = Mathf.Max(damage - defener.CurrentDefence, 0);

CurrentHealth = Mathf.Max(CurrentHealth - damageRock, 0);

}

//暴击伤害计算

private int CurrentDamage()

{

float coreDamage = UnityEngine.Random.Range(attackDate.maxDamage, attackDate.minDamage);

if (isCritical)

{

coreDamage \*= attackDate.criticalMultipler;

}

return (int)coreDamage;

}

#endregion

}

//攻击石头,击飞

if (attackTarget.CompareTag("Attackable"))

{

if (attackTarget.GetComponent<Rock>() && attackTarget.GetComponent<Rock>().rockStates == Rock.RockStates.HitNothing)

{

attackTarget.GetComponent<Rock>().rockStates = Rock.RockStates.HitEnemy;

attackTarget.GetComponent<Rock>().rb.velocity = Vector3.one;

//向前施加力

attackTarget.GetComponent<Rigidbody>().AddForce(transform.forward \* 20, ForceMode.Impulse);

}

}

//石头攻击

public void TakeDamage(int damage,CharacterState defener)

{

int damageRock = Mathf.Max(damage - defener.CurrentDefence, 0);

CurrentHealth = Mathf.Max(CurrentHealth - damageRock, 0);

}private void OnCollisionEnter(Collision other)

{

switch (rockStates)

{

case RockStates.HitPlayer:

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

{

//击退

other.gameObject.GetComponent<NavMeshAgent>().isStopped = true;

other.gameObject.GetComponent<NavMeshAgent>().velocity = direction \* force;

//眩晕

other.gameObject.GetComponent<Animator>().SetTrigger("Dizzy");

//伤害

other.gameObject.GetComponent<CharacterState>().TakeDamage(damage, other.gameObject.GetComponent<CharacterState>());

//更改状态

rockStates = RockStates.HitNothing;

}

break;

case RockStates.HitEnemy:

if (other.gameObject.GetComponent<Golem>())

{

other.gameObject.GetComponent<CharacterState>().TakeDamage(damage, other.gameObject.GetComponent<CharacterState>());

Destroy(gameObject);

}

break;

case RockStates.HitNothing:

break;

default:

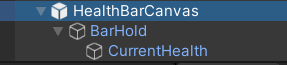
break;

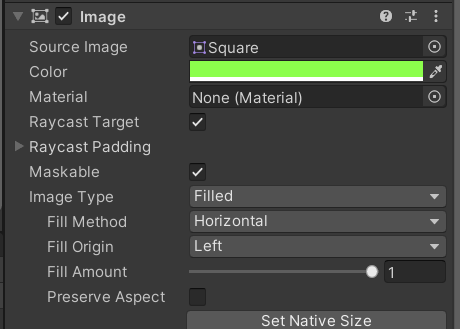
}

}

}

设置血条UI显示





定义血条事件

public event Action<int, int> UpdateHealthBarOnAttack;

//受伤伤害

public void TakeDamage(CharacterState attacker,CharacterState defener)

{

//Mathf.Max 取最大值,保证伤害不为负数

int damage = Mathf.Max(attacker.CurrentDamage() - defener.CurrentDefence,0);

//更新生命值

CurrentHealth = Math.Max(CurrentHealth - damage, 0);

//受伤动画显示

if (attacker.isCritical)

{

defener.GetComponent<Animator>().SetTrigger("Hit");

}

//TODO:更新血量的UI显示传入角色的生命数据

UpdateHealthBarOnAttack?.Invoke(CurrentHealth, MaxHealth);

//todo:敌人死亡后更新经验值

}

using UnityEngine;

using UnityEngine.UI;

public class HealthBarUI : MonoBehaviour

{

public GameObject healthBarUI;

public Transform barPoint;

//血条可视时间

public bool alwaysVisible;

public float visibleTime;

public float timeleft;

Image healthSlider;

Transform barUI;

Transform cam;

CharacterState currentState;

private void Awake()

{

currentState = GetComponent<CharacterState>();

//血条显示事件

currentState.UpdateHealthBarOnAttack += UpdateHealthBar;

}

private void OnEnable()

{

cam = Camera.main.transform;

foreach (Canvas canvas in FindObjectsOfType<Canvas>())

{

if (canvas.renderMode == RenderMode.WorldSpace)

{

barUI = Instantiate(healthBarUI, canvas.transform).transform;

healthSlider = barUI.GetChild(0).GetComponent<Image>();

barUI.gameObject.SetActive(alwaysVisible);

}

}

}

private void UpdateHealthBar(int currentHealth, int maxHealth)

{

if ( currentHealth <= 0)

{

Destroy(barUI.gameObject);

}

barUI.gameObject.SetActive(alwaysVisible);

timeleft = visibleTime;

float sliderPercent = (float)currentHealth / maxHealth;

healthSlider.fillAmount = sliderPercent;

}

private void LateUpdate()

{

if (barUI != null)

{

barUI.transform.position = barPoint.position;

barUI.forward = -cam.forward;

if (alwaysVisible == true && timeleft <= 0)

{

barUI.gameObject.SetActive(false);

}

else

{

timeleft -= Time.deltaTime;

}

}

}

}

实现升级系统

using UnityEngine;

[CreateAssetMenu(fileName = "New Date",menuName = "Character State/Date")]

public class CharacterDate\_SO : ScriptableObject

{

//[Header("State Info")]

//public int maxHealth;

//public int currentHealth;

//public int baseDefence;

//public int currentDefence;

[Header("LevelSeting")]

public int killPoint;

public int currentLevel;

public int maxLevel;

public int baseExp;

public int currentExp;

public float levelBuff;

public float levelMultiplier

{

get { return 1 + (currentLevel - 1) \* levelBuff; }

}

public void UpdateExp(int point)

{

currentExp += point;

if (currentExp >= baseExp)

{

levelUp();

}

}

private void levelUp()

{

currentLevel = Mathf.Clamp(currentLevel + 1, 0, maxLevel);

baseExp += (int)(baseExp \* levelMultiplier);

maxHealth = (int)(maxHealth \* levelMultiplier);

currentHealth = maxHealth;

Debug.Log(currentLevel + ",," + maxHealth);

}

}

在攻击时调用

public void TakeDamage(CharacterState attacker,CharacterState defener)

{

//Mathf.Max 取最大值,保证伤害不为负数

int damage = Mathf.Max(attacker.CurrentDamage() - defener.CurrentDefence,0);

//更新生命值

CurrentHealth = Math.Max(CurrentHealth - damage, 0);

//受伤动画显示

if (attacker.isCritical)

{

defener.GetComponent<Animator>().SetTrigger("Hit");

}

//TODO:更新血量的UI显示

UpdateHealthBarOnAttack?.Invoke(CurrentHealth, MaxHealth);

//todo:敌人死亡后更新经验值

if (CurrentHealth <= 0)

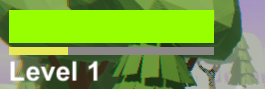
{

attacker.characterDate.UpdateExp(characterDate.killPoint);

}

}

实现玩家的生命值,经验等级的UI更新



using UnityEngine.UI;

public class PlayerHealthBarUI : MonoBehaviour

{

private Text levelText;

private Image healthSlider;

private Image expSlider;

private void Awake()

{

levelText = transform.GetChild(2).GetComponent<Text>();

healthSlider = transform.GetChild(0).GetChild(0).GetComponent<Image>();

expSlider = transform.GetChild(1).GetChild(0).GetComponent<Image>();

}

private void Update()

{

levelText.text= "Level " + GameManager.Instance.playerState.characterDate.currentLevel.ToString("00");

UpdateHealth();

UpdateExp();

}

private void UpdateExp()

{

float sliderPercent = (float)GameManager.Instance.playerState.characterDate.currentExp / GameManager.Instance.playerState.characterDate.baseExp;

expSlider.fillAmount = sliderPercent;

Debug.Log("exp"+sliderPercent);

}

private void UpdateHealth()

{

float sliderPercent = (float)GameManager.Instance.playerState.CurrentHealth / GameManager.Instance.playerState.MaxHealth;

healthSlider.fillAmount = sliderPercent;

Debug.Log(sliderPercent);

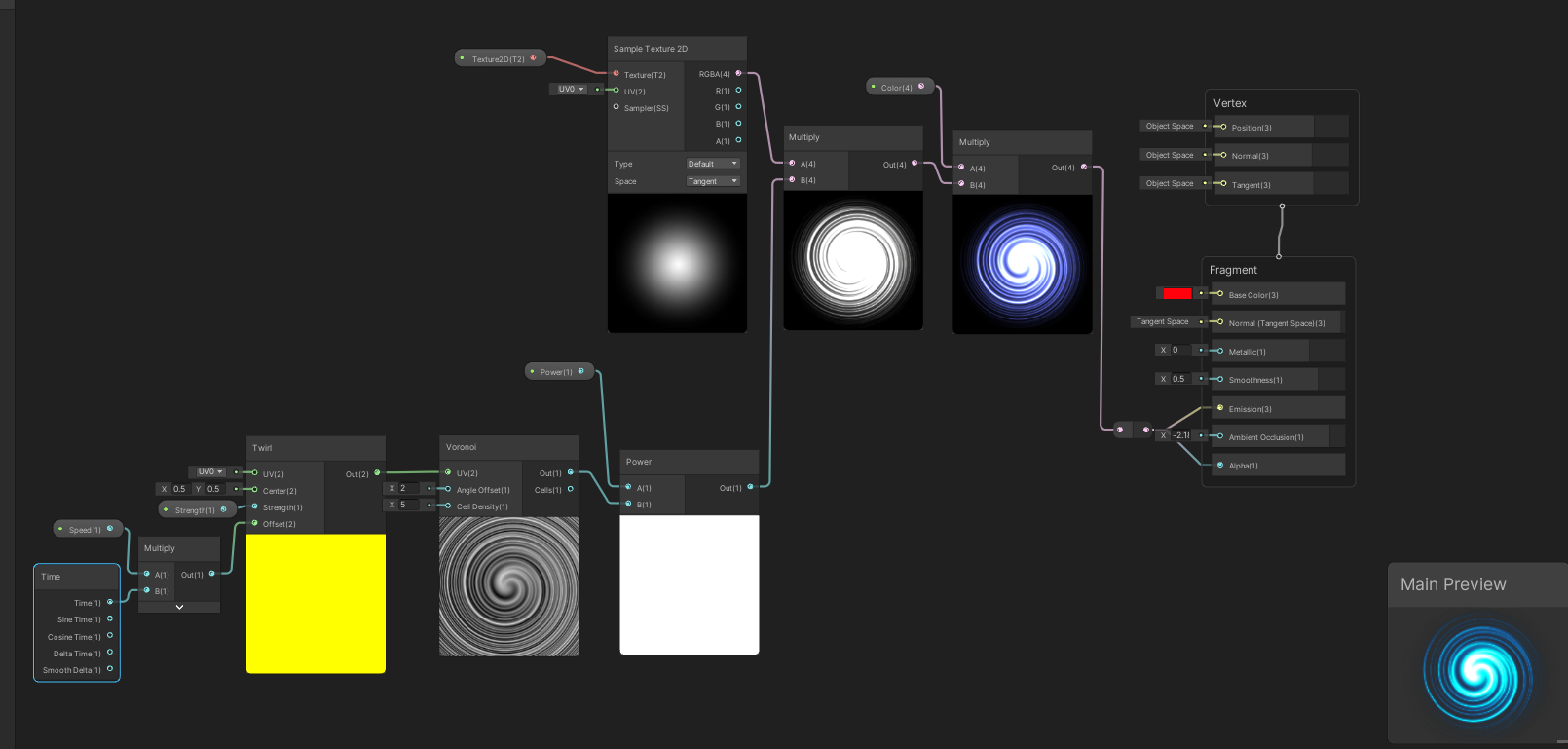
Debug.Log(healthSlider.fillAmount);

}

}

设置传送门

shader



传送门的入口

public class PatrolPoint : MonoBehaviour

{

public enum TransitionType {SameScenc,DifferentScenc}

[Header("Transition Info")]

public string scencName;

public TransitionType transitionType;

public PatrolDestination.DestinationType destinationType;

public bool canEntry;

private void Update()

{

if (Input.GetKeyDown(KeyCode.E) && canEntry == true)

{

ScencControl.Instance.TransitionToDestination(this);

}

}

private void OnTriggerStay(Collider other)

{

if (other.CompareTag("Player"))

{

canEntry = true;

}

}

private void OnTriggerExit(Collider other)

{

if (other.CompareTag("Player"))

{

canEntry = false;

}

}

}

传送门的出口

public class PatrolDestination : MonoBehaviour

{

public enum DestinationType { Entry,A,B,C}

public DestinationType destinationType;

}

传送门控制 协程实现

using UnityEngine.SceneManagement;

using UnityEngine.AI;

public class ScencControl : SingleTon<ScencControl>

{

GameObject player;

NavMeshAgent playerAgent;

public GameObject playerPrefab;

protected override void Awake()

{

base.Awake();

DontDestroyOnLoad(this);

}

public void TransitionToDestination(PatrolPoint patrolPoint)

{

switch (patrolPoint.transitionType)

{

case PatrolPoint.TransitionType.SameScenc:

StartCoroutine(Transition(SceneManager.GetActiveScene().name, patrolPoint.destinationType));

break;

case PatrolPoint.TransitionType.DifferentScenc:

StartCoroutine(Transition(patrolPoint.scencName, patrolPoint.destinationType));

break;

default:

break;

}

}

//获取传送出口的点

public PatrolDestination GetDestination(PatrolDestination.DestinationType destinationType)

{

var entrances = FindObjectsOfType<PatrolDestination>();

for (int i = 0; i < entrances.Length; i++)

{

if (entrances[i].destinationType == destinationType)

{

return entrances[i];

}

}

return null;

}

//设置玩家传送的位置

IEnumerator Transition(string scencName,PatrolDestination.DestinationType destination)

{

if (SceneManager.GetActiveScene().name == scencName)

{

player = GameManager.Instance.playerState.gameObject;

playerAgent = player.GetComponent<NavMeshAgent>();

playerAgent.enabled = false;

player.transform.SetPositionAndRotation(GetDestination(destination).transform.position, GetDestination(destination).transform.rotation);

playerAgent.enabled = true;

yield return null;

}else

{

//异步加载场景

yield return SceneManager.LoadSceneAsync(scencName);

yield return Instantiate(playerPrefab, GetDestination(destination).transform.position, GetDestination(destination).transform.rotation);

yield break;

}

}

}

保存玩家数据 JsonUtility 的用法PlayerPrefs 的用法

public class SaveMannager : SingleTon<SaveMannager>

{

protected override void Awake()

{

base.Awake();

DontDestroyOnLoad(this);

}

//测试功能

private void Update()

{

if (Input.GetKeyDown(KeyCode.S))

{

SavePlayerData();

}

if (Input.GetKeyDown(KeyCode.R))

{

LoadPlayerData();

}

}

public void SavePlayerData()

{

Save(GameManager.Instance.playerState.characterDate, GameManager.Instance.playerState.characterDate.name);

}

public void LoadPlayerData()

{

Load(GameManager.Instance.playerState.characterDate, GameManager.Instance.playerState.characterDate.name);

}

public void Save(object data,string key)

{

//json将数据存入

var jsonData = JsonUtility.ToJson(data,true);//使数据变好看

PlayerPrefs.SetString(key, jsonData);

//将数据写入磁盘

PlayerPrefs.Save();

}

public void Load(object data, string key)

{

if (PlayerPrefs.HasKey(key))

{

//读入数据

JsonUtility.FromJsonOverwrite(PlayerPrefs.GetString(key), data);

}

}

}