Unity2D 青蛙过河FrogJump

编辑器版本2022.1.8f1c1

2d核心模板

场景搭建

导入资源



放置基础资产



设置俯视角渲染方式



设置各个资产的锚点sprite editor







设置各个资产的图层





使用新的输入方式安装input system



创建Input Action



创建用户控制跳长跳获取点击位置



给青蛙添加控制



查找脚本



选择启动Unity事件



新建c#脚本（脚本名和C#中类名相同）



添加到Frog上



打开脚本添加inputsystem



最简单的调试



绑定到frog上



运行执行了相应代码



设置2D碰撞体





添加碰撞箱





跳跃代码

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.InputSystem;

public class PlayerController : MonoBehaviour

{

    //获得自身的组件，一般写在开头

    private Rigidbody2D rb;

    public float jumpDistance;//跳跃的距离

    private float moveDistance;//实际移动的距离

    private Vector2 destination;

    private bool buttonheld;//按键是否被长按

    private bool isjump;//跳跃途中的状态

    private void Update()

    {

        //FIXME：临时操作 跳跃到目标位置后重置跳跃状态

        if(destination.y - transform.position.y <= 0.1 )

        isjump = false;

    }

    private void Awake()

    {

        rb = GetComponent<Rigidbody2D>();

    }

    //FixedUpdate是一个稳定的函数每0.2秒执行一次

    private void FixedUpdate()

    {

        //position 是一个二维向量,希望使用lerp线性差值移动到一个地方

        //必须在跳跃结束后才能跳跃

        if(isjump)

        rb.position = Vector2.Lerp(transform.position,destination,0.134f);

    }

    public void Jump(InputAction.CallbackContext context)

    {

        if(context.phase == InputActionPhase.Performed&&isjump == false)

        {

            moveDistance = jumpDistance;

            //Debug.Log("jump!");

            destination = new Vector2(transform.position.x,transform.position.y+moveDistance);

            isjump = true;

        }

    }

    public void Longjump(InputAction.CallbackContext context)

    {

        if(context.performed&&isjump == false)

        {

            moveDistance = 2\*jumpDistance;

            buttonheld = true;

        }

        if(context.canceled && buttonheld == true && isjump == false)

        {

            //Debug.Log("Longjump!");

            buttonheld = false;

            destination = new Vector2(transform.position.x,transform.position.y+moveDistance);

            isjump = true;

        }

    }

    public void Gettouchposition(InputAction.CallbackContext context)

    {

        if(context.performed)

        {

            Debug.Log("Longjump!");

        }

    }

}

添加呼吸动画







添加跳跃动画



跳跃的时候不循环执行动作



创建过渡



创建一个jump的触发



过渡条件选择Jump



跳跃动画完毕恢复闲置状态







添加两个函数在动画的开头和结尾

    private void TrigerJump()

    {

        canjump = false;

        anim.SetTrigger("jump");

    }

    #region 通过动画控制跳跃

    public void JumpAnimationEvent()

    {

        isjump = true;

    }

    public void FinishJumpAnimationEvent()

    {

        isjump = false;

    }

    #endregion

}

完整代码如下

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.InputSystem;

public class PlayerController : MonoBehaviour

{

    //获得自身的组件，一般写在开头,获得的组件一般在Awake里初始化

    //获取Rigidbody2D组件

    private Rigidbody2D rb;

    //获取animator组件

    private Animator anim;

    public float jumpDistance;//跳跃的距离

    private float moveDistance;//实际移动的距离

    private Vector2 destination;

    private bool buttonheld;//按键是否被长按

    private bool isjump;//跳跃途中的状态

    private bool canjump;//可以跳跃

    private void Update()

    {

        //临时操作 跳跃到目标位置后重置跳跃状态

        //更改为了动画结束重置跳跃状态

        // if(destination.y - transform.position.y <= 0.1 )

        // isjump = false;

        if(canjump)

        {

            TrigerJump();

        }

    }

    private void Awake()

    {

        rb = GetComponent<Rigidbody2D>();

        anim = GetComponent<Animator>();

    }

    //FixedUpdate是一个稳定的函数每0.2秒执行一次

    private void FixedUpdate()

    {

        //position 是一个二维向量,希望使用lerp线性差值移动到一个地方

        //必须在跳跃结束后才能跳跃

        if(isjump)

        {

            rb.position = Vector2.Lerp(transform.position,destination,0.134f);

        }

    }

    #region input 输入回调函数

    public void Jump(InputAction.CallbackContext context)

    {

        if(context.phase == InputActionPhase.Performed&&isjump == false)

        {

            moveDistance = jumpDistance;

            //Debug.Log("jump!");

            destination = new Vector2(transform.position.x,transform.position.y+moveDistance);

            canjump = true;

        }

    }

    public void Longjump(InputAction.CallbackContext context)

    {

        if(context.performed&&isjump == false)

        {

            moveDistance = 2\*jumpDistance;

            buttonheld = true;

        }

        if(context.canceled && buttonheld == true && isjump == false)

        {

            //Debug.Log("Longjump!");

            buttonheld = false;

            destination = new Vector2(transform.position.x,transform.position.y+moveDistance);

            canjump = true;

        }

    }

    public void Gettouchposition(InputAction.CallbackContext context)

    {

        if(context.performed)

        {

            Debug.Log("Longjump!");

        }

    }

    #endregion

    private void TrigerJump()

    {

        canjump = false;

        anim.SetTrigger("jump");

    }

    #region 通过动画控制跳跃

    public void JumpAnimationEvent()

    {

        isjump = true;

    }

    public void FinishJumpAnimationEvent()

    {

        isjump = false;

    }

    #endregion

}

自适应摄像机控制

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraControl : MonoBehaviour

{

    public Transform frog;

    public float offsetY;

    private float ratio;

    public float zoombase;

    //不同的摄像机看到的画面应该相同

    private void Start()

    {

    //强制转换FLOAT否则会有问题

        ratio = (float)Screen.height / (float)Screen.width;

        //Debug.Log(ratio);

        Camera.main.orthographicSize = zoombase \* ratio \* 0.5f;

    }

    //摄像机以青蛙为基准跟随移动

    public  void LateUpdate()

    {

        transform.position = new Vector3(transform.position.x,frog.transform.position.y+offsetY\*ratio,transform.position.z);

    }

}



取消背景



添加UI->画布，绑定主摄像机



添加面板，图像选择背景，alpha调为255

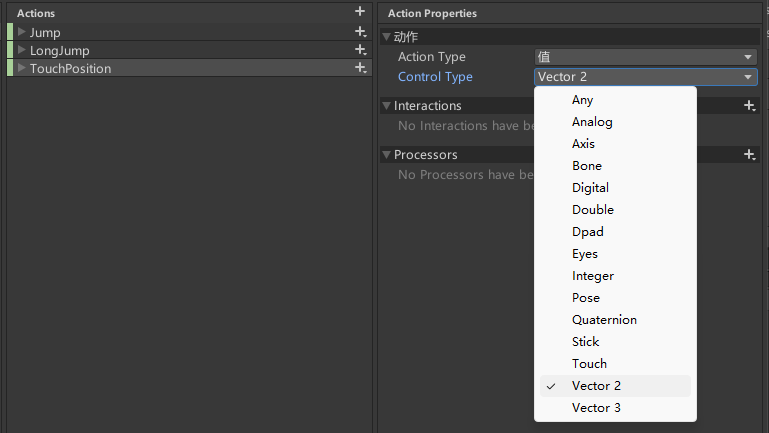


更改EventSystem为新的输入方式





添加左右移动，在Inputcontrols里面修改TouchPosition的控制类型



   //枚举变量 用于控制方向

    private enum Direction

    {

        up,right,left

    }

    private Vector2 touchposition;

    private Direction dir;

private void TrigerJump()

    {

        canjump = false;

        switch (dir)

        {

            case Direction.up:

            destination = new Vector2(transform.position.x,transform.position.y+moveDistance);

            break;

            case Direction.left:

            destination = new Vector2(transform.position.x - moveDistance,transform.position.y);

            break;

            case Direction.right:

            destination = new Vector2(transform.position.x + moveDistance,transform.position.y);

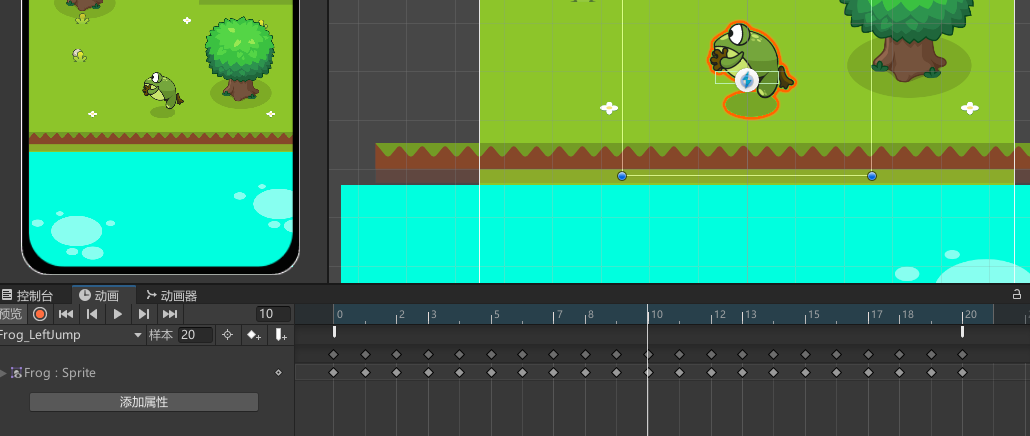
            break;

        }

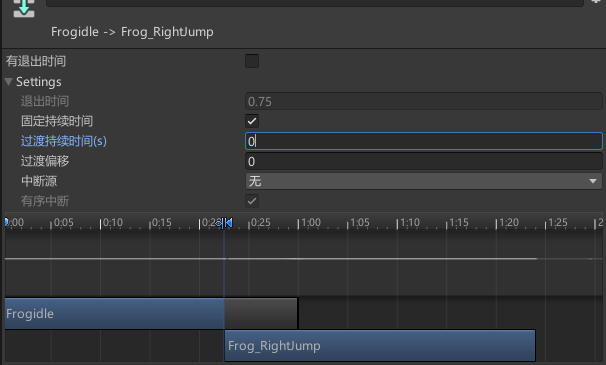
        anim.SetTrigger("jump");

    }

添加向左向右的动画



创建过渡



private void TrigerJump()

    {

        canjump = false;

        switch (dir)

        {

            case Direction.up:

            anim.SetBool("isside",false);

            destination = new Vector2(transform.position.x,transform.position.y+moveDistance);

            transform.localScale = Vector3.one;

            break;

            case Direction.left:

            anim.SetBool("isside",true);

            destination = new Vector2(transform.position.x - moveDistance,transform.position.y);

            transform.localScale = Vector3.one;

            break;

            case Direction.right:

            anim.SetBool("isside",true);

            destination = new Vector2(transform.position.x + moveDistance,transform.position.y);

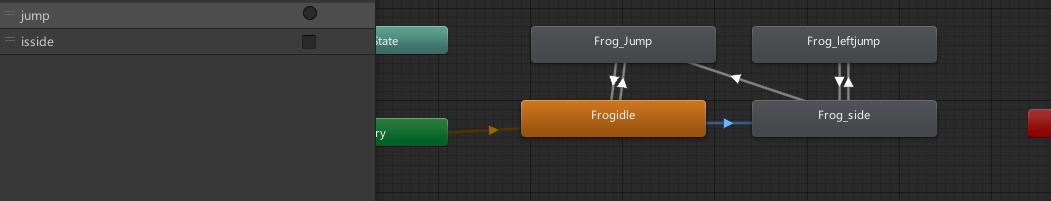
            transform.localScale = new Vector3(-1,1,1);

            break;

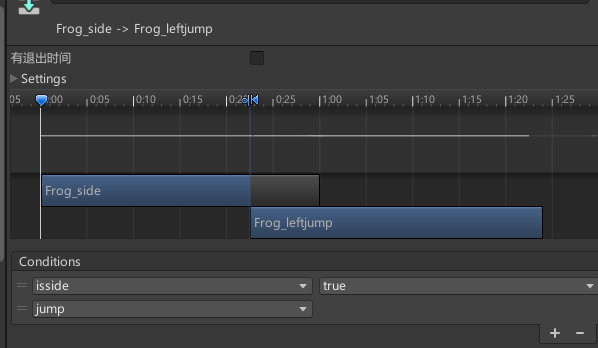
        }

        anim.SetTrigger("jump");

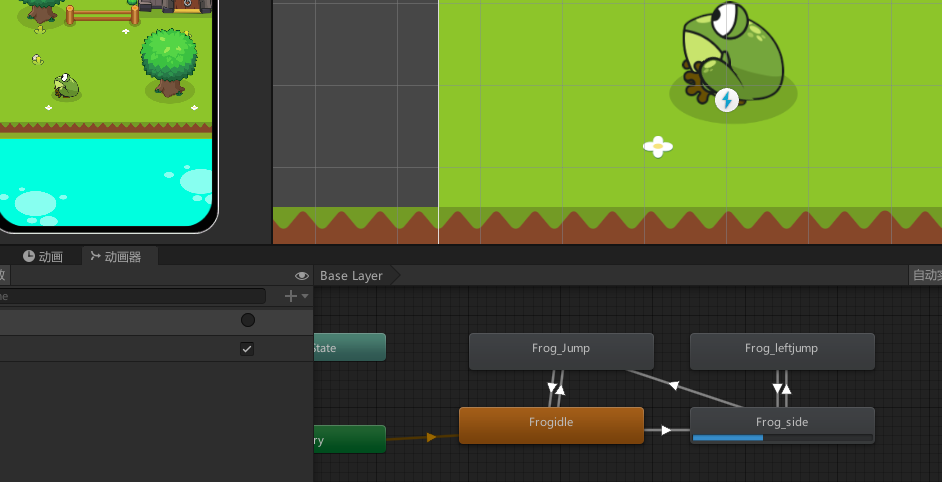
    }



创建布尔值isside用来判断是否前往侧面的状态



满足状态后朝向侧方



通过修改图层实现飞跃的效果

    public void JumpAnimationEvent()

    {

        //改变状态

        isjump = true;

        // Debug.Log(dir);

        //修改图层顺序

        sr.sortingLayerName = "front";

    }

    public void FinishJumpAnimationEvent()

    {

        isjump = false;

        //修改图层顺序

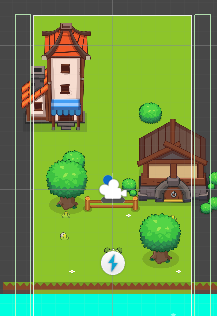
        sr.sortingLayerName = "middle";

    }

创建场景草坪

编辑碰撞器





    //Collider2D other是除了青蛙之外，对方的触发器

    public void  OnTriggerStay2D(Collider2D other)

    {

        if(other.CompareTag("Obs")&&!isjump)

        {

            Debug.Log("Gameover!");

        }

        if(other.CompareTag("Border"))

        {

            Debug.Log("gameover!");

        }

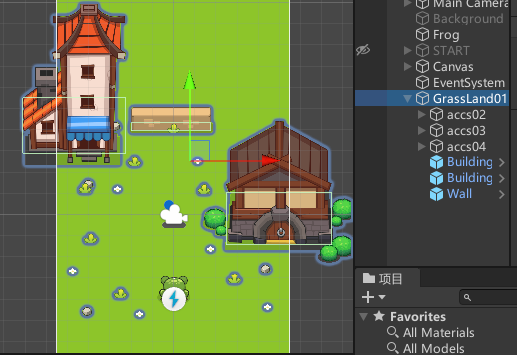
    }

预制体

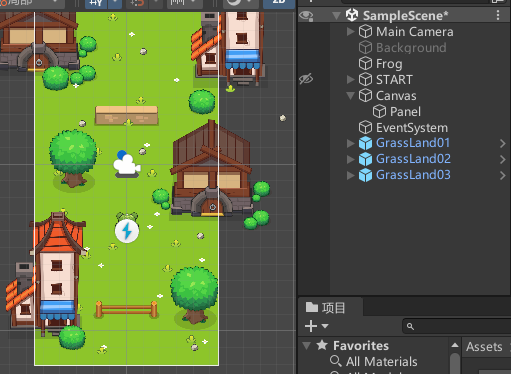
如果预制体属性发生变化，那么场景中所有与预制体相关联的物体都会发生变化，实现模板的效果



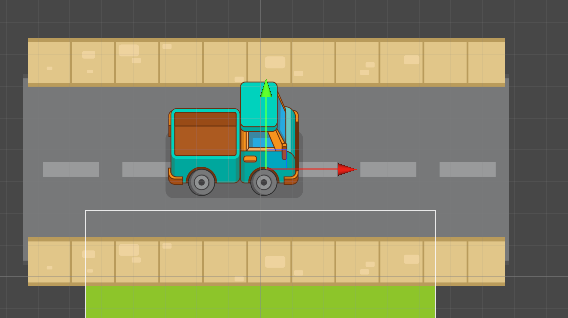
创建一个草地场景



做三个草地



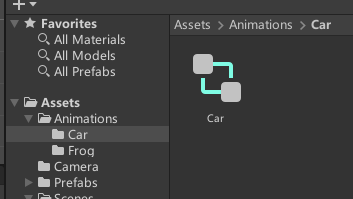
做马路和车



给小汽车添加动画



创建小汽车的动画控制器



小汽车的自动销毁和移动

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MoveForward : MonoBehaviour

{

    private Vector2 startPos;

    public float speed;

    private void Start() {

        startPos = transform.position;

    }

    // Update is called once per frame

    void Update()

    {

        if(Mathf.Abs(transform.position.x - startPos.x)>15)

        {

            Destroy(this.gameObject);

        }

        Move();

    }

    private void Move()

    {

        transform.position += transform.right \*speed\*Time.deltaTime;

    }

}

添加出生点



using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Spawner : MonoBehaviour

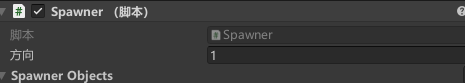
{

    //创建列表来存储三种小汽车

    public List<GameObject> SpawnerObjects;

}

向左标记为-1，向右标记为1



    private void Move()

    {

        transform.position += transform.right \* dir \*speed\*Time.deltaTime;

    }

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Spawner : MonoBehaviour

{

    public int direction;

    //创建列表来存储三种小汽车

    public List<GameObject> SpawnerObjects;

    private void Spawn()

    {

        var index =  Random.Range(0,SpawnerObjects.Count);

        var car = Instantiate(SpawnerObjects[index],transform.position,Quaternion.identity,transform);

        car.GetComponent<MoveForward>().dir = direction;

    }

    private void Start()

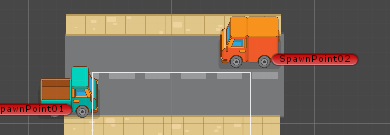
    {

        InvokeRepeating(nameof(Spawn),0.2f,Random.Range(6f,7f));

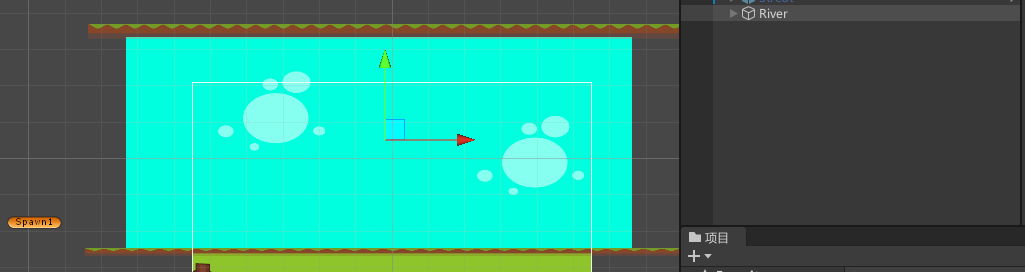
    }

}

便会有随机生成的小车左右出现了



同理，制作小河



会有木板从两边经过

青蛙跳上木板时跟随木板移动

青蛙掉到水里游戏结束

private RaycastHit2D[] result = new RaycastHit2D[2];//不分配内存的物理射线检测

public void  OnTriggerStay2D(Collider2D other)

    {

        if(other.CompareTag("Obs")&&!isjump)

        {

            Debug.Log("Gameover!");

        }

        if(other.CompareTag("Border") || other.CompareTag("Car"))

        {

            Debug.Log("gameover!");

        }

        if(other.CompareTag("Water")&&!isjump)

        {

            //通过青蛙发射向下的射线判断是水还是木板

            Physics2D.RaycastNonAlloc(transform.position + Vector3.up \* 0.1f,Vector2.zero,result);

            //检测碰撞 在result的两个值中循环

            //result可以返回碰撞体

            foreach (var hit in result)

            {

                if(hit.collider == null) continue;

                if(hit.collider.CompareTag("Wood"))

                {

                    Debug.Log("inwood");

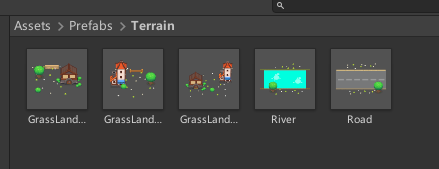
                }

            }

        }

    }

相应场景的预制体



制作自动循环生成这些地图

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Terrn : MonoBehaviour

{

    private int lastnum = 4;

    public float offsety;

    private GameObject spanobject;//生成的物品

    public List<GameObject> SpawnerObjects; //创建列表来存储物品

    private void Spawn()

    {

        var index =  Random.Range(0,SpawnerObjects.Count);//生成一个随机数字0到序号

        while (lastnum == index)

        {

            index =  Random.Range(0,SpawnerObjects.Count);//生成一个随机数字0到序号

        }

        lastnum = index;

        spanobject = SpawnerObjects[index];

        Instantiate(spanobject,transform.position, Quaternion.identity);

    }

    public void Checkposition()

    {

        if(transform.position.y - Camera.main.transform.position.y < offsety / 2 )

        {

            transform.position = new Vector3(0,Camera.main.transform.position.y+offsety,0);

            Debug.Log(transform.position);

            Spawn();

        }

    }

    // private void Start()

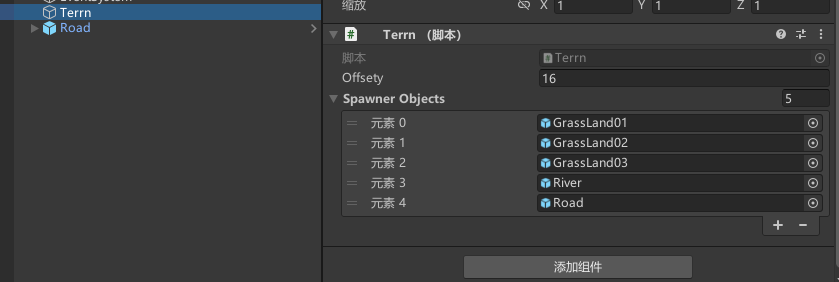
    // {

    //     Checkposition();

    // }

}

挂载到Terrn上



销毁之前生成的地图

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Block : MonoBehaviour

{

    // Start is called before the first frame update

    void Start()

    {

    }

    // Update is called once per frame

    void Update()

    {

        //FIXME

        Checkposition();

    }

    private void Checkposition()

    {

        if(Camera.main.transform.position.y - transform.position.y>25)

        {

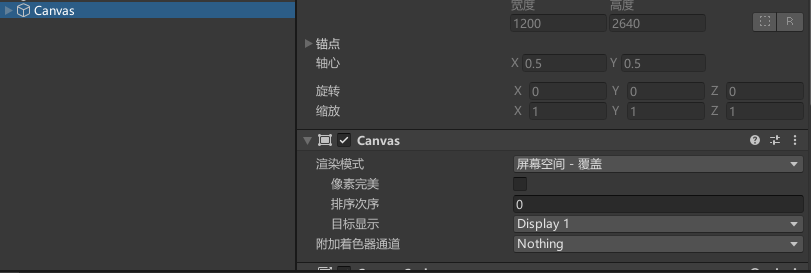
            Destroy(this.gameObject);

        }

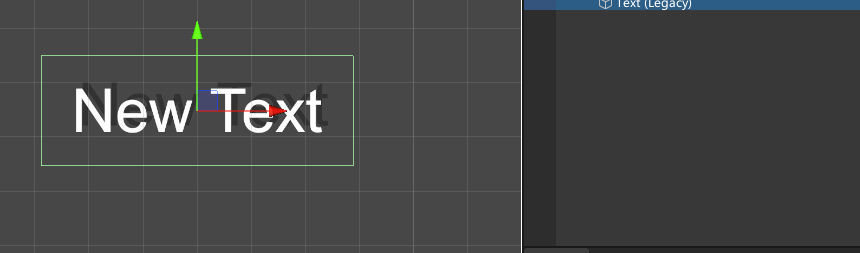
    }

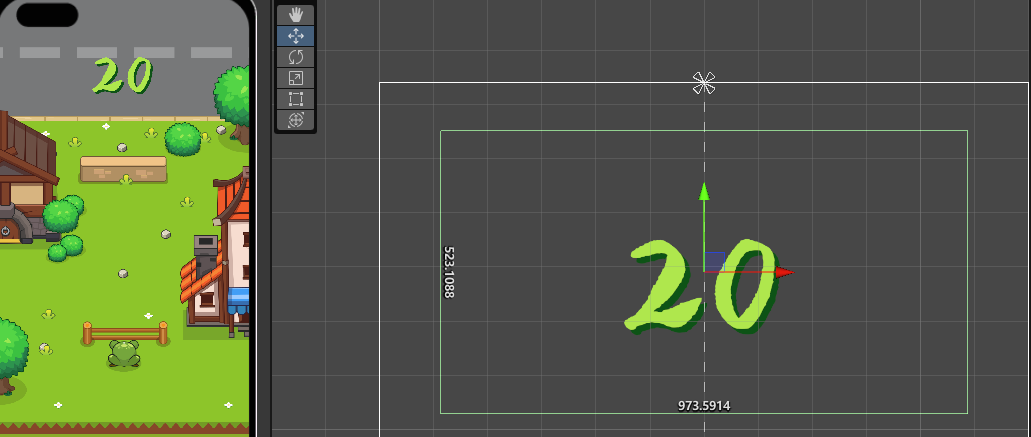
}

创建UI

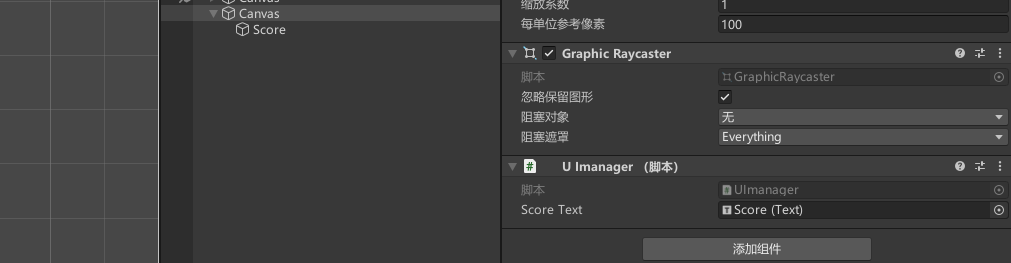


得分框





挂载text



解决耦合性

首先是调度中心

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System;

//不用挂载到任何物体上，就不会有任何父类函数

public class EventHandler

{

    public static event Action<int> GetPointEvent;//去定义通知其它函数和类要调用的数值类型

    public static void CallGetPointEvent(int point)//来具体呼叫的函数

    {

        //  if(GetPointEvent != null)

        //  {

        //     //启动这个

        //     GetPointEvent.Invoke(point);

        //  }

        //问号等同于 不为空的情况下再执行

        GetPointEvent?.Invoke(point);

    }

}

然后是目的脚本中需要有注册和注销以及具体的方法

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class UImanager : MonoBehaviour

{

    public Text scoreText;

    //注册

    private void OnEnable()

    {

        //使用+=来注册

        EventHandler.GetPointEvent+= OnGetPointEvent;//注册函数的参数必须和方法相同

    }

    private void OnDisable()

    {

        //使用-=来注销

        EventHandler.GetPointEvent-= OnGetPointEvent;//注销函数的参数必须和方法相同

    }

    //注销

    private void Start()

    {

        scoreText.text = "00";

    }

    private void OnGetPointEvent(int point)

    {

        //转换字符串

        scoreText.text = point.ToString();

    }

}

发起脚本中通知调度中心

    public void FinishJumpAnimationEvent()

    {

        isjump = false;

        //修改图层顺序

        sr.sortingLayerName = "middle";

        if(!isdead && dir == Direction.up)

        {

            //TODO得分触发地图检测

            //FiXME

            terrn.Checkposition();

            EventHandler.CallGetPointEvent((int)resultpoint);

            //Debug.Log("总得分"+resultpoint);

        }

    }

游戏结束UI



   if(isdead)

    {

       //广播通知游戏结束

       EventHandler.CallGameOverEvent();

    }

    private void OnGameOverEvent()

    {

        //启用一个物体

        gameOverPanel.SetActive(true);

    }

死亡时禁用游戏控制

    private void DisableInput()//死亡时禁用游戏控制

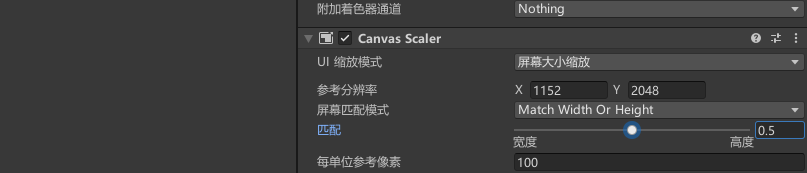
    {

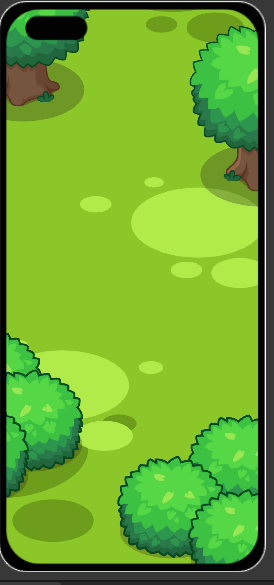
        playerInput.enabled = false ;

    }

创建游戏主界面

根据图片尺寸设置





添加素材，添加动画效果



为开始游戏添加脚本按钮监听的方法

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class PlayButton : MonoBehaviour

{

private Button button;

private void Awake()

{

    button = GetComponent<Button>();

    button.onClick.AddListener(StartGame);

}

private void StartGame()

{

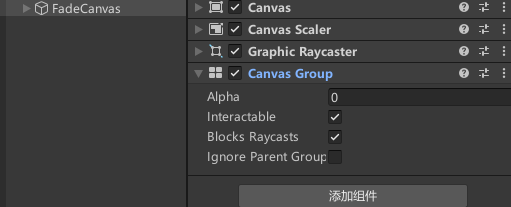
    //启动游戏加载场景

    SceneManager.LoadScene("Gameplay");

}

}

过渡组件，调节透明度与禁用点击



UI过渡

    public void Transition(string sceneName)

    {

        StartCoroutine(TransitionToScene(sceneName));

    }

    //协程函数迭代器 可以按照想要的顺序执行

    private IEnumerator TransitionToScene(string sceneName)

    {

        yield return Fade(1);

        yield return SceneManager.LoadSceneAsync(sceneName);

        yield return Fade(0);

    }

同时静态化实例，以便在其他脚本中使用场景过渡例如从主界面到游戏界面

    //创建一个静态实例，使其他脚本可以调用

    public static TransitionManager instance;//一般都写为instance;

    public void BackToMeau()

    {

        TransitionManager.instance.Transition("Tiltle");

    }

在弹出的UI里面引用

    public void RestartGame()

    {

        // SceneManager.LoadScene(SceneManager.GetActiveScene().name);

        gameOverPanel.SetActive(false);

        TransitionManager.instance.Transition("Gameplay");

    }

    public void BackToMeau()

    {

        gameOverPanel.SetActive(false);

        TransitionManager.instance.Transition("Tiltle");

    }

排行榜

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System.IO;

using Newtonsoft.Json;

public class GameManager : MonoBehaviour

{

    public List<int> scoreList;

    private int score;

    private string dataPath;

    private void Awake() {

        dataPath = Application.persistentDataPath+"/leaderboard.json";

        scoreList = GetScoreListData();

        DontDestroyOnLoad(this);

    }

    private void OnEnable()

    {

        EventHandler.GameOverEvent += OnGameOverEvent;

        EventHandler.GetPointEvent += OnGetPointEvent;//注册函数的参数必须和方法相同

    }

    private void OnDisable() {

        EventHandler.GameOverEvent -= OnGameOverEvent;

        EventHandler.GetPointEvent -= OnGetPointEvent;

    }

    private void OnGetPointEvent(int point)

    {

        score = point;

    }

    private void OnGameOverEvent()

    {

        //TODO在List中添加新的分数，排序

        //判断是否包含相同分数

        if(!scoreList.Contains(score))

        {

            scoreList.Add(score);

        }

        scoreList.Sort();

        scoreList.Reverse();

        //序列化

        File.WriteAllText(dataPath,JsonConvert.SerializeObject(scoreList));

    }

    private List<int> GetScoreListData()

    {

        if(File.Exists(dataPath))

        {

            string jsonData = File.ReadAllText(dataPath);

            //反序列化

            return JsonConvert.DeserializeObject<List<int>>(jsonData);

        }

        return new List<int>();

    }

}