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

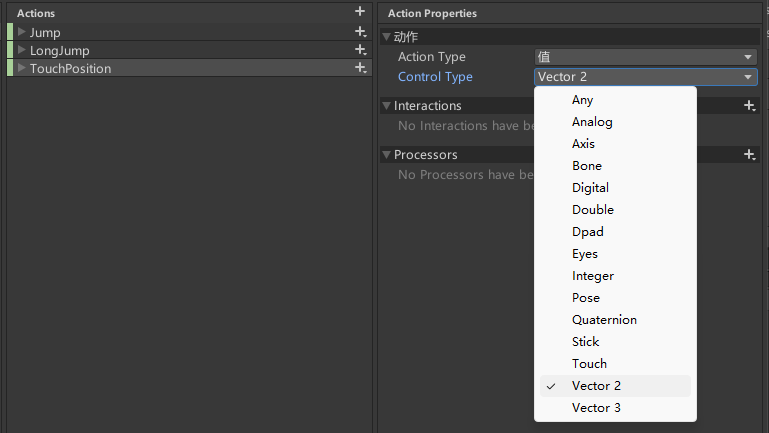


更改EventSystem为新的输入方式





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);

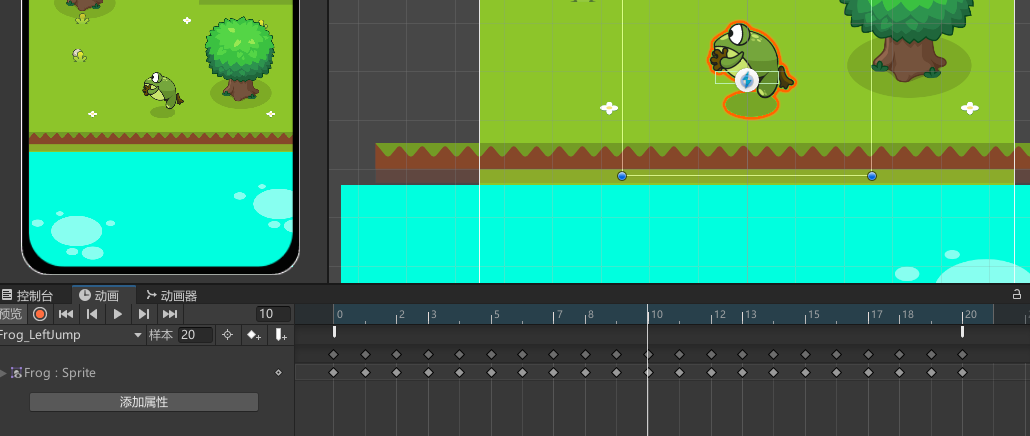
            break;

        }

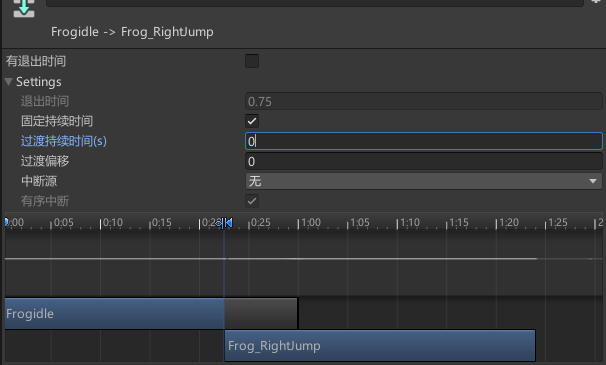
        anim.SetTrigger("jump");

    }

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);

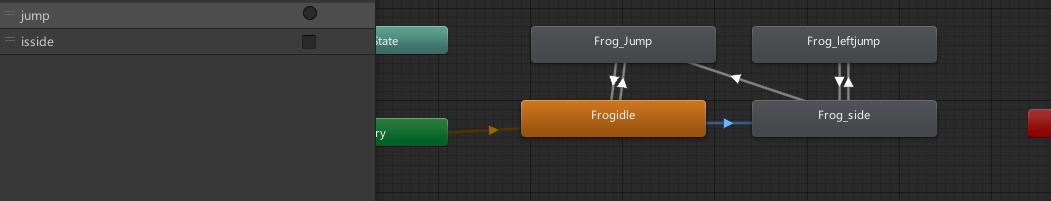
            transform.localScale = new Vector3(-1,1,1);

            break;

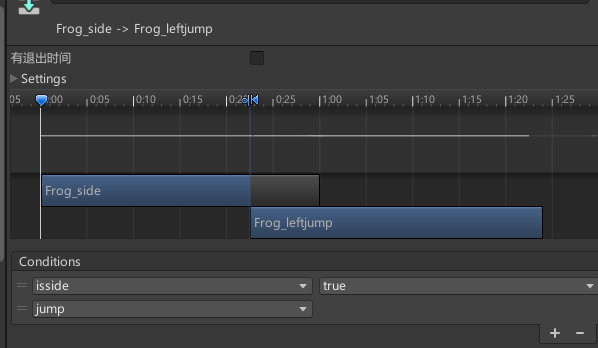
        }

        anim.SetTrigger("jump");

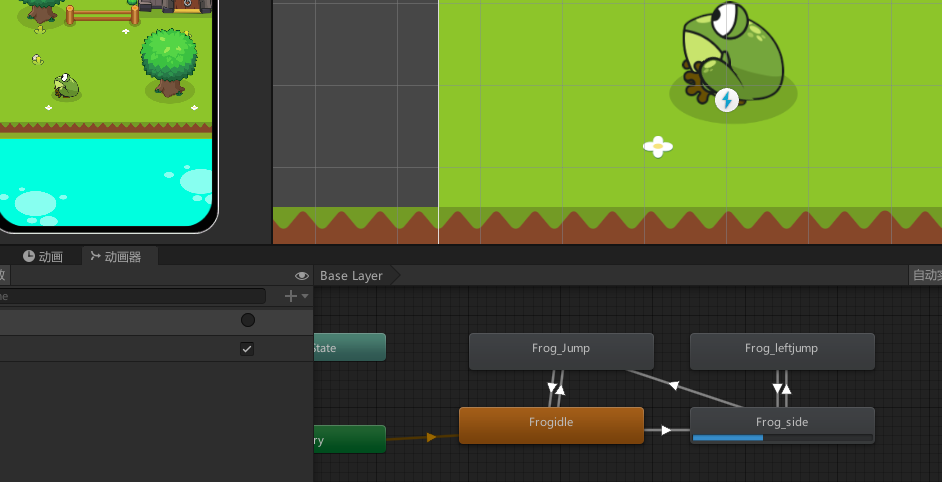
    }



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



