Small Fox复盘

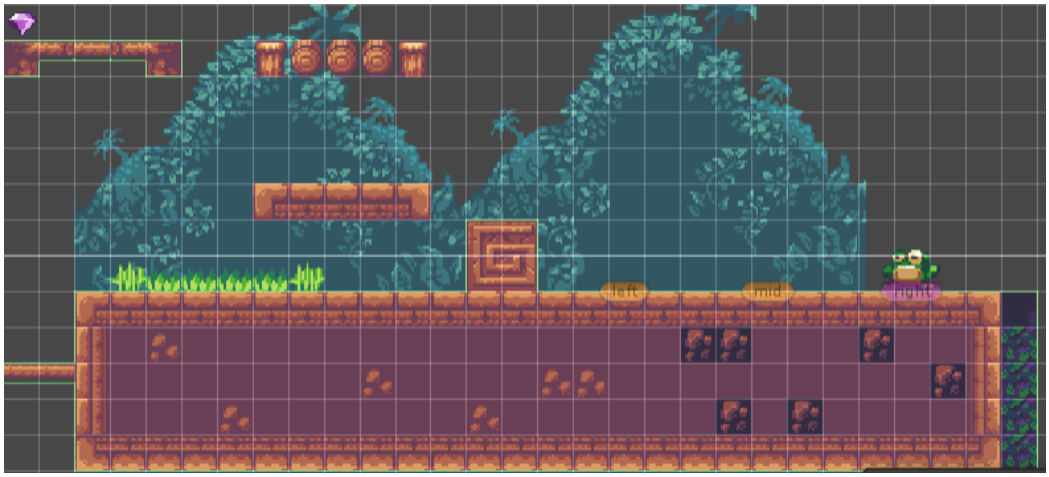
一.unity基本功能方面

1.1 Tilemap

当地图单位为一格一格(像素风格or用小瓦片构建地图时)可使用Tilemap。

创建方式:右键->2D Objects->Tilemap->选择瓦片的类型。

瓦片可添加碰撞器，若是需要所有小瓦片的碰撞器平滑连接，可添加Tilemap Collider 2D和Composite Collider 2D(此时会自动添加刚体，需要的话可调整刚体属性)，然后在Tilemap Collider 2D中将Used By Composite勾选即可实现多个瓦片平滑链接的碰撞体。如下图。



1.2 Cinemachine

需要用到摄影机移动或者更多功能时，可自己编写Camera Controller脚本，也可使用更方便，功能强大的Cinemachine。(目前掌握内容少)

创建方式:在package manager中将packages改成Unity Registry ，搜索Cinemachine->Import之后便可在上方看到Cinemachine(目前在小狐狸中所用到的是2D Camera)

2D Camera:创建完成后，将你的角色拖进Follow，在game视图中摄像机便会跟着角色。如果要设置摄像机能看到的最大范围，可在自带脚本最下面Add Extension中添加Cinemachine Confiner(极限)，于是便新建一个GameObject(也可直接在背景中)添加一个多边形碰撞器(只接受多边形碰撞器)，调整碰撞器的大小拖入Cinemachine Confiner即可。2D Camera的主脚本中有很多提升观感的设置，可在需要时查询相关文档。

1.3 UI

unity自带的UI一般只有最基础的功能及字体，有需要可去Asset Store下载资源。

1.3.1 Text

无需多言。

1.3.2 Panel

Panel，面板，在游戏中为对话框更为合适。可添加相应字体，改变位置，透明度，和背景。

有意思的是，想让对话框实现逐渐浮现的功能，可为Panel添加Animator组件创建动画，通过录制功能改变不同节点的透明的从而实现对话框的逐渐浮现(录制在Animation中讲)。

1.3.3 Button

Button，顾名思义，按钮，相应的属性可搭配文档使用，当然Asset Store也有相应资源啦~(万能的Asset Store)

也可在添加函数达到在点击时执行某函数(如暂停)

1.3.4 Image

Image，作为UI的一种，同样有着GameObject难以替代的功能，可固定在屏幕某一位置，相关设置也无需赘述~

1.4 Animation

1.4.1 添加Animation

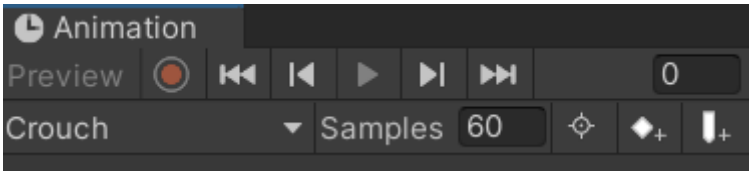
Animation，可以说是游戏中十分重要的功能，在使用时也是稍微繁琐。

首先，为角色添加Animator，再在Assets下创建Animation文件夹，方便管理，在里面创建Animation Controller拖拽至角色的Animator中，接下来创建动画，选中角色，便可为角色添加动画啦。
基本操作，拖入动画。

可通过右下角长的按钮添加一个function(例如播放完该动画后摧毁该物体)，右下角短的按钮则是添加关键帧。

可通过左上角的录取功能录取关键帧，在上文的Panel处即可在录取不同帧时改变图形的透明度不循环，即可实现对话框的浮现。

Samples一般为60就行..

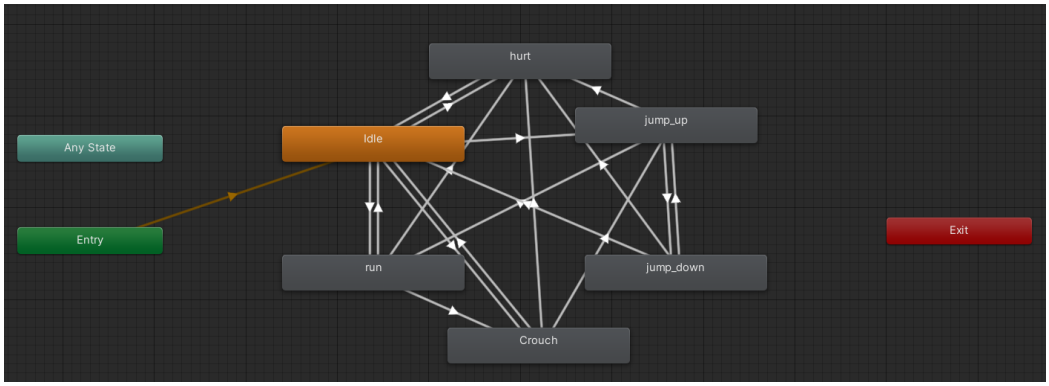


1.4.2 实现动画的切换

打开Animation Controller将创建的动画拖入(如果里面还没动画的话)，然后明确各动画的转化关系。(一定要考虑周全)

图中Entry指向的代表在其他条件未触发时，动画就进行Idle，Any State指向的则是任何情况下都可以转化(即一旦该线的条件满足，就执行该动画)。

用右键make transition的方式来画线。



接下来便是为各种线设置触发条件。

首先在右边方框的Parameters(参数)中添加各自类型的数据(int，float，bool，trigger)来设置相应的条件，通常情况下，bool就可实现。

在设置条件名时最好命名相仿(Idleing,jumping...)以防止在后面拼写错误时不报错，难以发现问题。

条件设置好后在相应的线上添加Inspector面板中设置条件，(如Idle->jump_up 条件为 jumping true)。同时，也需设置相关参数(查)。

然后再脚本中通过函数 `anim.Setbool("Ideling",true)` 来实现条件的成立，动画的转换(anim为该物体的Animator)。通过 `anim.Getbool("Ideling")` 来得到当时动画的在播放。

1.5 Project Setting

可将语言调成中文来调整Project Setting，重要的大致有游戏名，图标，鼠标，是否窗口化，是否可调整窗口大小，分辨率，logo...

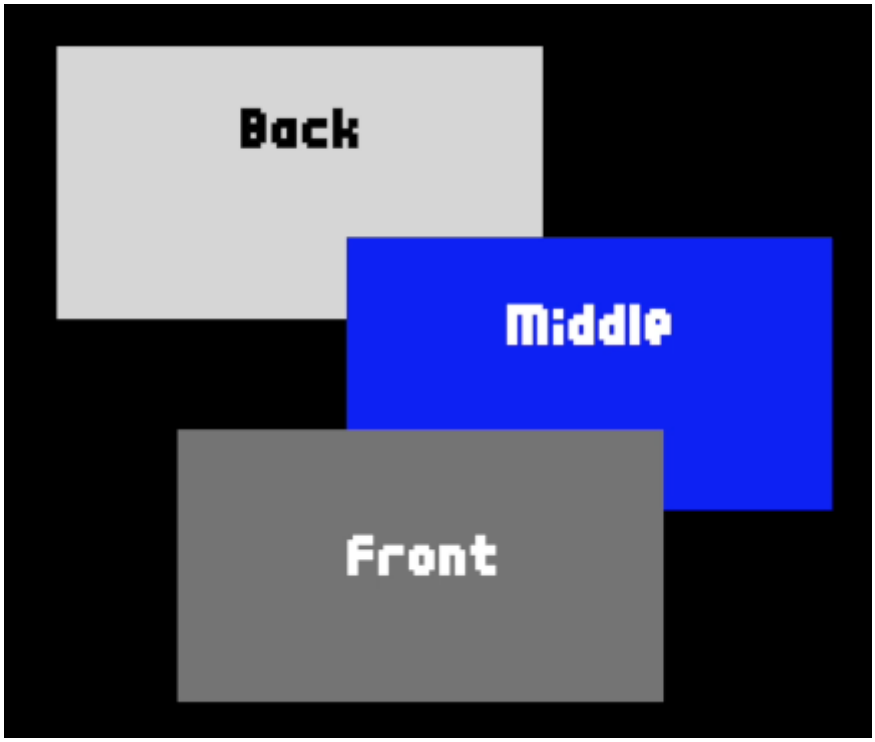
1.6 Light

2D的地图默认是没有光源的，此时如果将物体的SpriteRenderer的材质全设为Diffuse，看到的物体就是没有光照射的样子，然后在地图上添加不同的Light，可实现光源效果。

1.6 Parallax

Parallax，视觉差，在2D游戏中可通过此方法让游戏更有层次感，立体感。

简单的Parallax就是通过前景，中景，后景移动速度的不同来实现

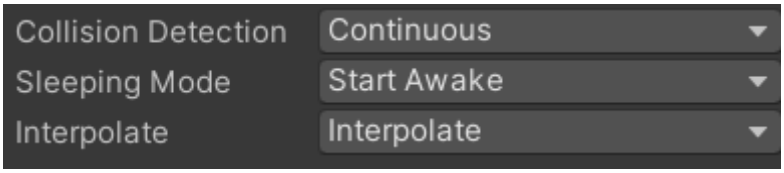


第一种方法 可通过代码实现。

第二种方法 通过Z轴实现

首先将main camera的projection属性改成perspective(透视)，再将前景，中景，后景设置不同的Z轴高度运行时就可感觉到背景在晃动，有明显的层次感！

1.7 Rigidbody2D



1.7.1 Collision Detection:

Discrete：离散碰撞检测，该模式与场景中其他的所有碰撞体进行碰撞检测，该项为默认值。

Continuous：连续碰撞检测，该项用于检测与动态碰撞体（带有的rigidboby）的碰撞，其他的刚体会采用离散体碰撞模式，从模式使用于那些需要与采用连续动态碰撞检测的对象相碰撞的对象。这对物理性能有很大的影响，如果不需要对快速运动的对象进行碰撞检测，就使用离散碰撞检测模式。

1.7.2 Interpolate

物体在发生微小碰撞时可能会凹陷然后在弹出，选择Interpolate可改变这种情况。

Interpolate：一个以固定的帧率平滑物理运行的插值（选择**Interpolate**时物理移动更平滑）。

Interpolate表示根据前几帧的位置来做平滑插值，Extrapolate表示根据预测的下一帧的位置来做平滑插值。

二.C#代码方面

1.1 代码习惯

多封装函数，主函数中能少写就少写。

该物体需要设置预制体时少用public，用private然后在start中得到。

与物理有关的函数都放在FixedUpdate。

1.2 零散的代码知识

```
GameObject.FindWithTag("Player");//通过标签寻找带有Player标签的游戏物体，并返回。
GameObject.FindGameObjectsWithTag("Player");//通过标签寻找带有Player标签的所有游戏物体，并返回。
Debug.DrawRay(transform.position + new Vector3(-0.6f,-0.3f, 0), Vector2.up, Color.blue);//打印光线，只能在Scene界面看到。
[Header("Dash参数")]//可实现在unity界面中添加该部分变量的标题。
Cherry_score.text = Cherryscore.ToString();//转换类型，将文字转换为数字形，也可在后面加上""什么都没有的字符..
Time.timeScale//时间流速，可通过改变事件流速从而达到暂停或者缓慢行动，一旦更改，全局都会受到影响。
SceneManager.LoadScene(SceneManager.GetActiveScene().name);//重新加载当前场景。
SceneManager.LoadScene(数字);//加载场景编号为该数字的场景。
[SerializeField]private//定义一个可以在unity界面看到并且更改的变量，但不能被其他脚本引用。
```

*1.3 2D*角色控制器的手感问题

Rigidbody2D中改变角色的Gravity Scale改变重力加速度让角色有着更好的下落感。

将移动放在FixedUpdate中从而让相同时间内移动的距离相等。

关于**getkeydown**失灵的问题 (button也适用)

```
void Update(){
    if(Input.GetKeyDown(KeyCode.Space)){
        A=true;
    }
}
void FixedUpdate(){
    if(A){
        Action();
    }
}
```

将按下放在Update中，执行放在FixedUpdate中可完美解决该问题！

通过射线检测，检测角色是否可站起时出现的**bug**

如果用`if(Physics2D.Raycast(transform.position + new Vector3(0.6f, -0.3f, 0), Vector2.up, 1f, ground) && Physics2D.Raycast(transform.position + new Vector3(-0.6f, -0.3f, 0), Vector2.up, 1f, ground))`

则会出现前面一个条件满足便进入if内，所以改成两个if即可~

给角色的Collider添加一个摩擦力为0的材质可防止贴墙

再在脚底添加一个有摩擦力的Collider可避免角色滑动不停

脚底下的Collider可作为地面检测，检测是否在地面上从而防止无限跳

这样，新的问题出现了，双碰撞体导致在几帧内两碰撞体同时碰到樱桃导致得分+2

解决办法

```
if (collision.gameObject.tag == "Cherry")
{
    item.Play();
    Destroy(collision.gameObject);
    Instantiate(Item_Eaten,
collision.gameObject.transform.position, Quaternion.identity);
    Cherryscore++;
    ollision.gameObject.tag = "Untagged";
}
```

在碰撞后将它的标签改为"Untagged"，即可完美解决问题啦~

1.4 对象池实现冲刺残影

1.4.1 对象池介绍

先说说为什么要用对象池，如果不断Instantiate和Destory几种相同的物体，会不断消耗内存从而导致游戏崩溃，于是对象池便用来 减少对象频繁创建所占用的内存空间和初始化时间 。预先创建几个对象放入对象池中，后续需使用时直接查询对象池内的对象，不需使用时放回，便可实现对象池。

在Small Fox中，对象池用的是 `private Queue<GameObject> availableObjects = new Queue<GameObject>();` 来实现的，当然其他情况下也可用List，因为冲刺残影满足先进先出，故直接用Queue更方便。

1.4.2 对象池实现

先贴出创建对象池的脚本

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

public class ShadowPool : MonoBehaviour
{
    public static ShadowPool instance;//定义一个单例，外部可以方便地访问到这个定义的instance对象

    public GameObject shadowPrefab;

    public int shadowCount;//对象的数量,需保证对象的数量能满足实例的残影

    private Queue<GameObject> availableObjects = new Queue<GameObject>();
```

```
14
15      void Awake()
16      {
17          instance = this;//初始化单例
18
19          //初始化对象池
20          FillPool();
21      }
22
23      public void FillPool()
24      {
25          for (int i = 0; i < shadowCount; i++)
26          {
27              var newShadow = Instantiate(shadowPrefab);//实例化
    对象
28              newShadow.transform.SetParent(transform);//将对象池
    的物体放在对象池的子物体下
29
30              //取消启用,返回对象池
31              ReturnPool(newShadow);
32          }
33      }
34
35      public void ReturnPool(GameObject gameObject)
36      {
37          gameObject.SetActive(false);//默认对象为关闭状态
38
39          availableObjects.Enqueue(gameObject);//将该对象添加到
    Queue中
40      }
41
42      public GameObject GetFormPool()
43      {
44          if (availableObjects.Count == 0)//对象取完了，再次填冲对象
    池
45          {
46              FillPool();
47          }
48          var outShadow = availableObjects.Dequeue();//取出对象
49
50          outShadow.SetActive(true);//取出的对象设为启用状态，此处即
    残影的产生
51
52          return outShadow;
53      }
54  }
```

产生对象池中的对象的脚本

```
1  using UnityEngine;
```

```
2
3  public class ShadowSprite : MonoBehaviour
4  {
5      private GameObject Player;
6      private Transform player;
7      private SpriteRenderer thisSprite;
8      private SpriteRenderer playerSprite;
9
10     private Color color;
11
12     [Header("时间控制参数")]
13     public float activeTime;//显示时间
14     public float activeStart;//开始显示的时间点
15
16     [Header("不透明度控制")]
17     private float alpha;//不透明度
18     public float alphaSet;//初始值
19     public float alphaMultiplier;//小于1的值
20
21     private void OnEnable()
22     {
23         Player = GameObject.FindGameObjectWithTag("Player");
24         player =
   GameObject.FindGameObjectWithTag("Player").transform;
25         thisSprite =GetComponent<SpriteRenderer>();
26         playerSprite = Player.GetComponent<SpriteRenderer>
   ();//角色当前的Sprite
27
28         alpha = alphaSet;
29
30         thisSprite.sprite = playerSprite.sprite;
31
32         transform.position = player.position;
33         transform.localScale = player.localScale;
34         transform.rotation = player.rotation;
35
36         activeStart = Time.time;
37     }
38
39     void FixedUpdate()
40     {
41         alpha *= alphaMultiplier;//透明度逐渐降低
42
43         color = new Color(0.5f, 0.5f, 1,
   alpha);//Color(1,1,1,1)代表100%显示各通道颜色，请查看Api手册
44
45         thisSprite.color = color;
46
47         if (Time.time >= activeStart + activeTime)
48         {
49             //返回对象池
```

```
            ShadowPool.instance.ReturnPool(this.gameObject);//因为instance了对象池脚本，故可直接调用
        }
    }
}

```