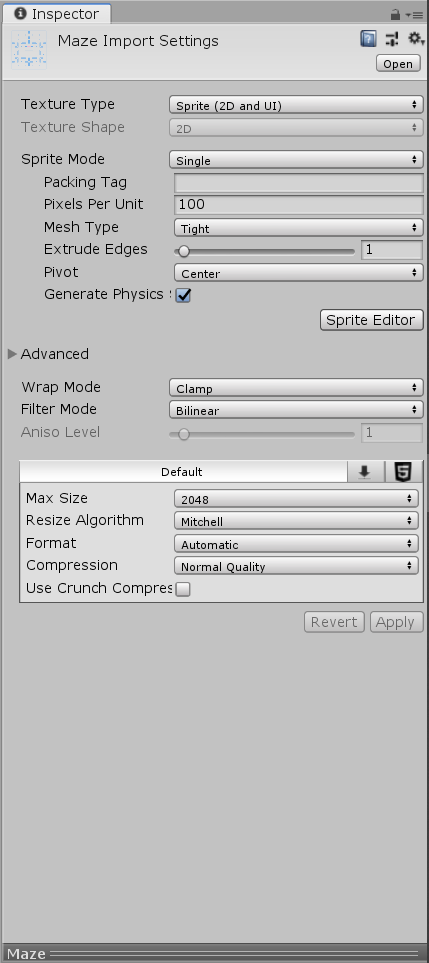
U2d

吃豆人

1. 如何在2D/3D工程中制作3D/2D场景

* 选择Edit->Project Settings->Editor->Default Behavior Mode改变当前模式
* 选择完后再按Ctrl+N在Scences文件夹里面新建新场景

1. 关于Sprite（2D and UI）



图中的Pixel Per Unit 代表每个单位多少个像素

比如该图

232×256 在Pixels Per Unit等于8的情况下 宽为 232/8=29 高为256/8=32

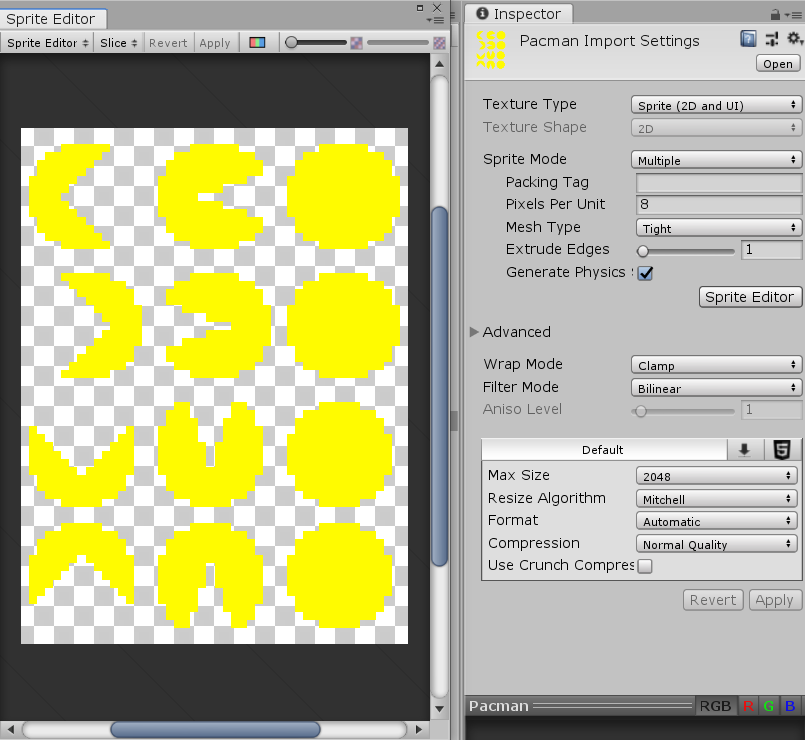


1. 迷宫的物理化
2. 直接给图片挂上多个碰撞器（无需多余物体）
3. 创建多个空物体来挂碰撞器（易于区分）

4.如何导出/导入素材

1.选中你要导出的资源  
2.Assets -> Select Dependencies  
3.Assets -> Export Package 弹出窗口中选 All (默认) 点击 Export... 按钮  
4.资源打包完成.  
5.将打包好的文件拖到你的目标项目中.  
6.双击你导入的资源包.  
7.弹出窗口中点击 Import 按钮  
8.导入完成

5. 如何编辑图片 Sprite Editor 其中的切割 Slice

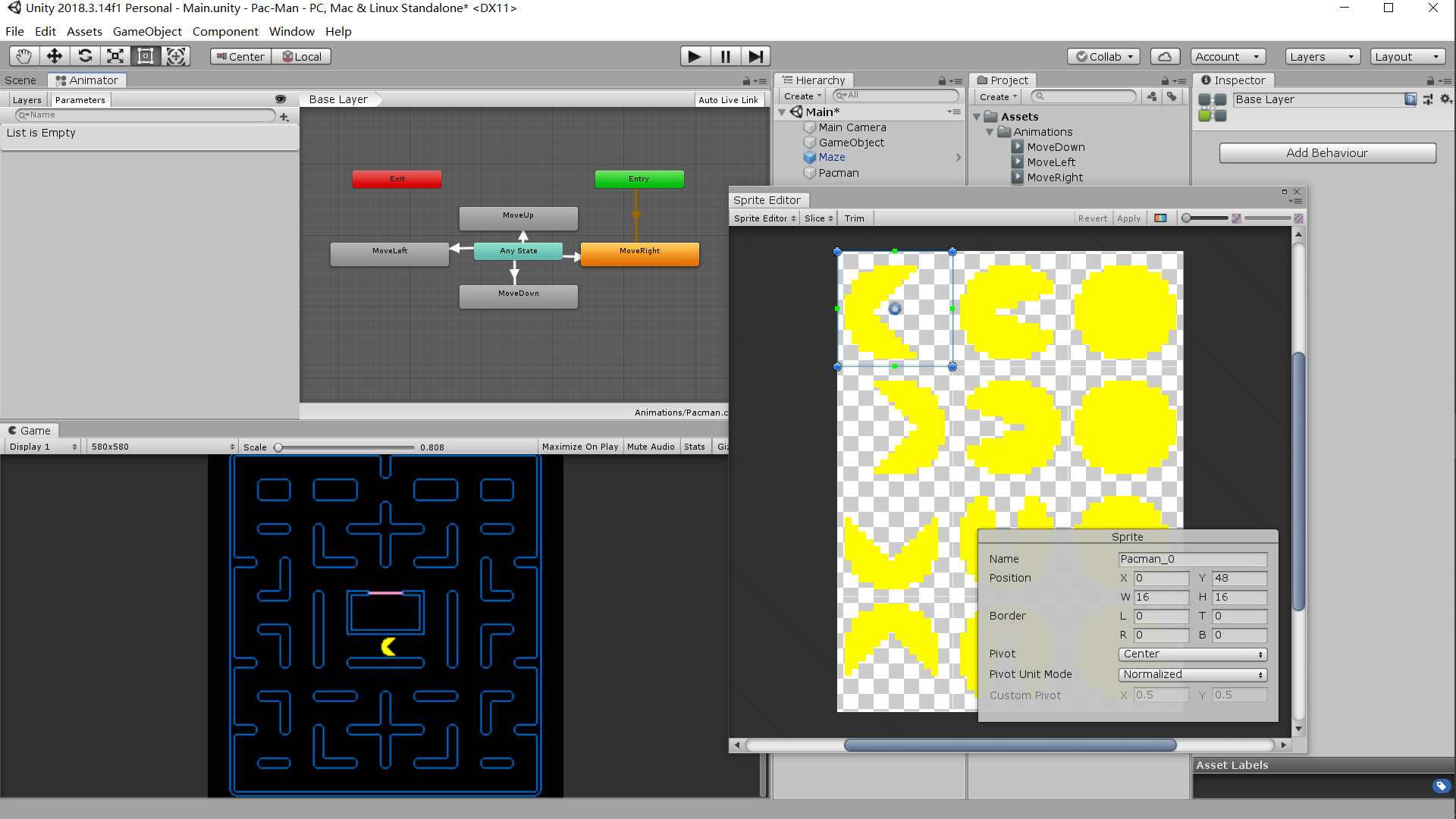


切割完后 点击apply 便会有子图片在父图片下方

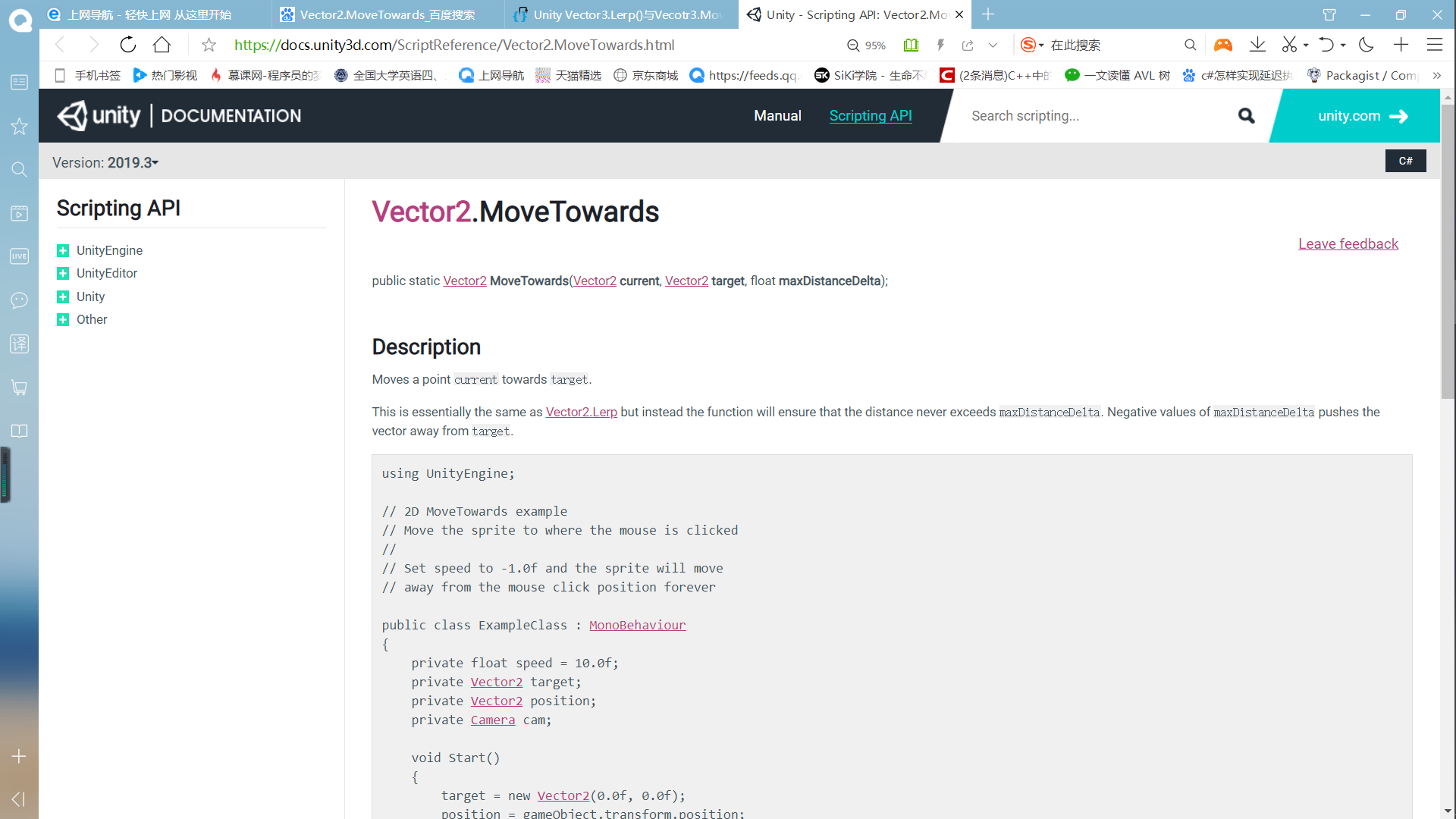
6. 状态机之间的状态转换如果需要两两之间都可以互相转化

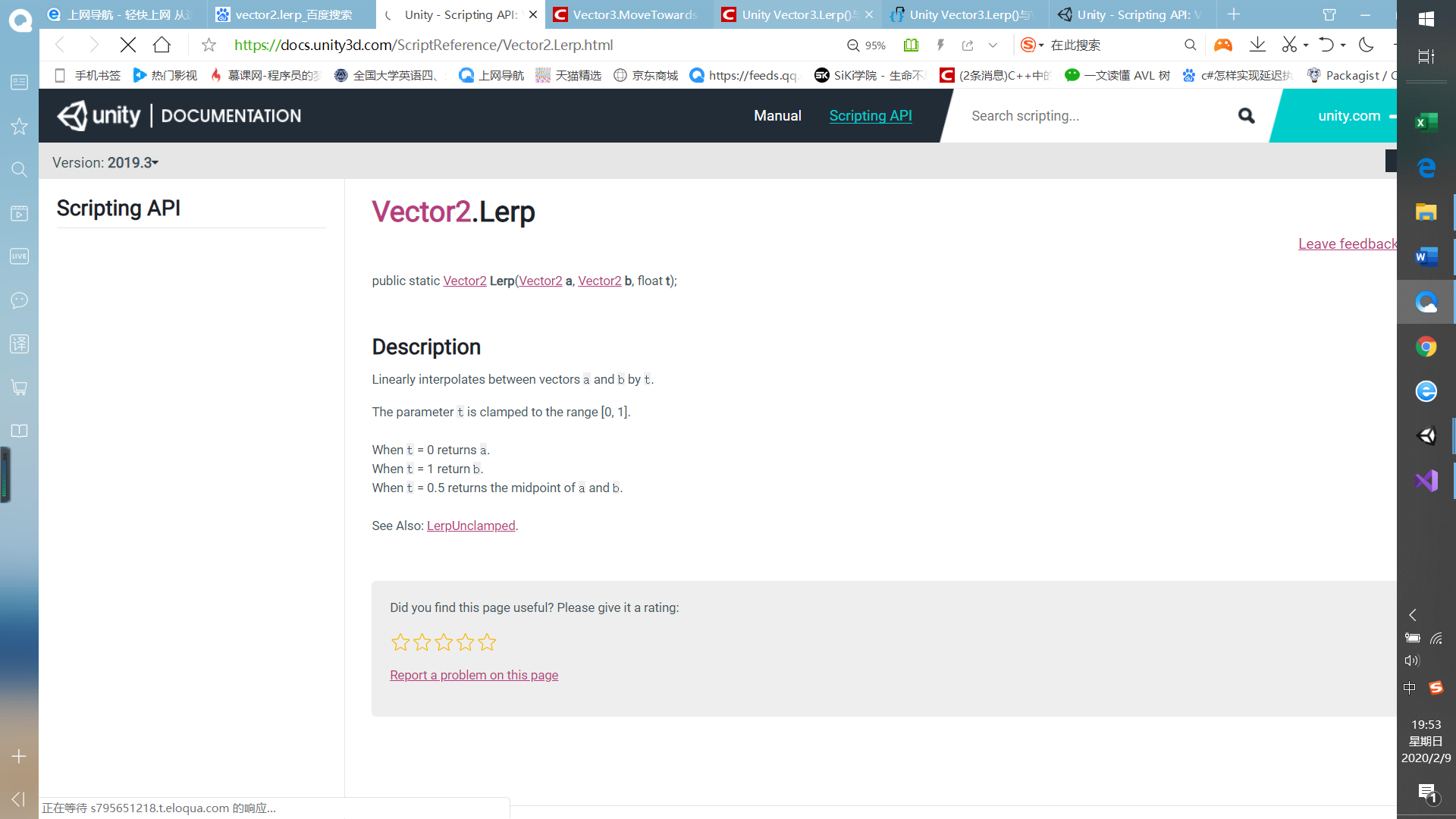
可以使用Any State 减少多余的连线

代表图中的任何状态



7.Vector2.MoveTowards方法

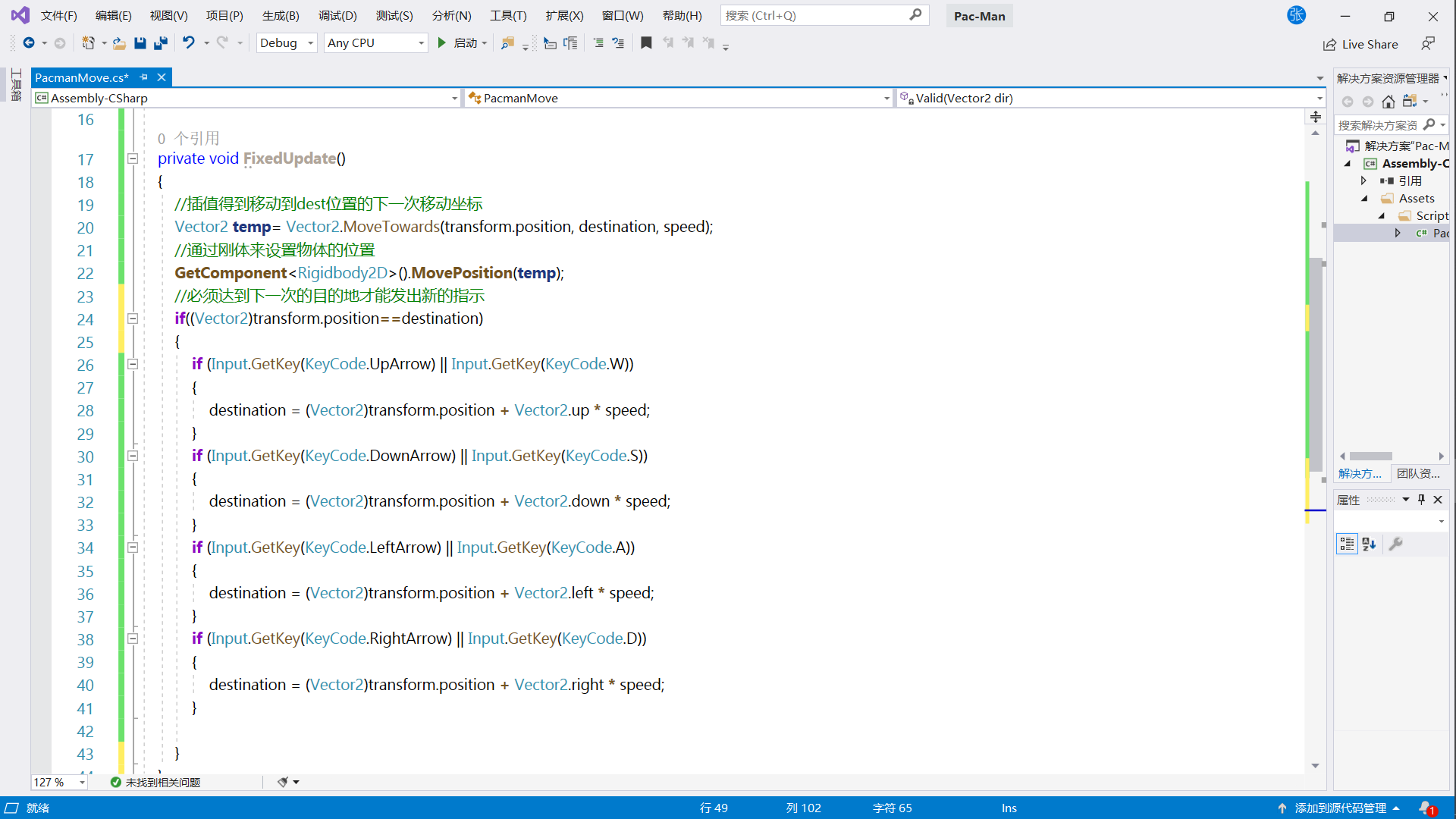
顾名思义 可以选择移动的目的地 ，移动的速度



基本上与Vector2.Lerp相同，但函数将确保距离永远不会超过maxDistanceDelta。maxDistanceDelta的负值将向量推离目标

Ps:在vector2的坐标表示（x，y，z）在二维场景中和日常的坐标轴是一样的

Lerp和MoveTowards的区别

* Lerp先快后慢
* MoveTowards不会无穷接近是一个匀速的过程
* 在脚本中用到的移动方法
* 

8.脚本中用到的巧妙防止无法到达destination而不进行任何运动的方法

private bool Valid(Vector2 dir)

{

Vector2 pos = transform.position;

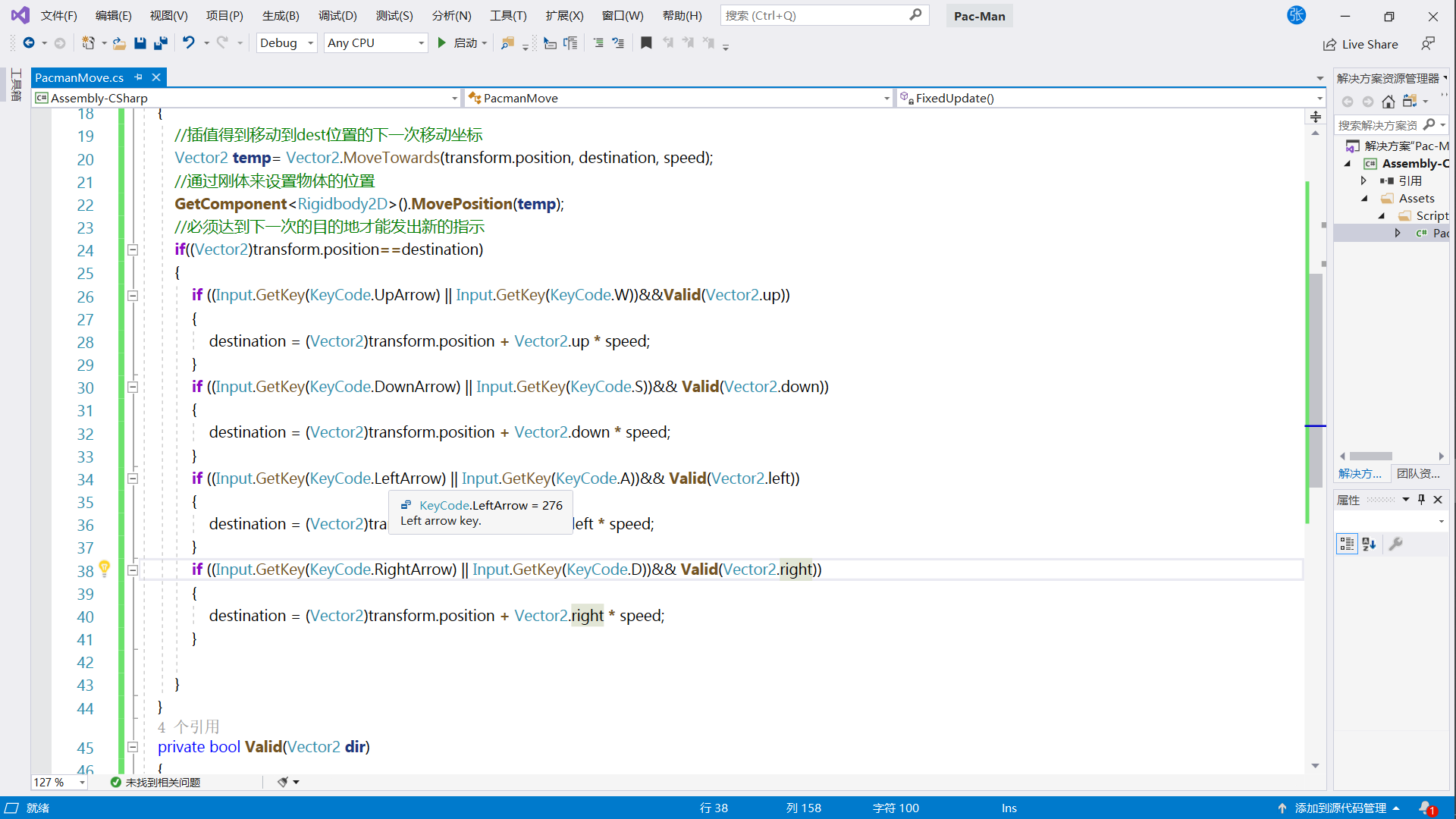
RaycastHit2D hit = Physics2D.Linecast(pos + dir, pos);

return (hit.collider == GetComponent<Collider2D>());

}

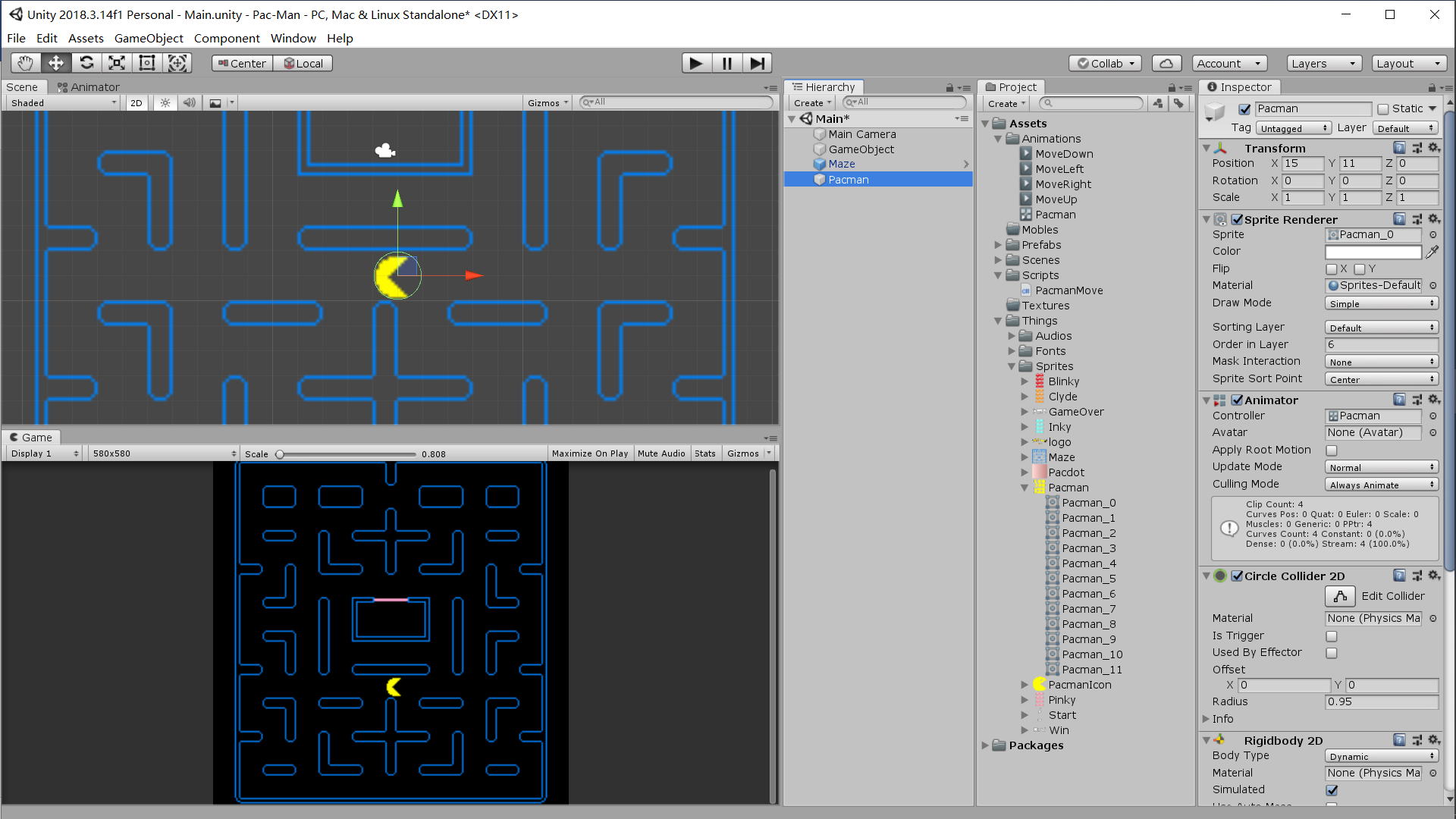
原理就在于从 pos+dir位置向当前所在位置pos发射射线 如果射线撞到墙体说明此处无法再往外穿墙

由此得到的新的移动函数



9.图片的渲染顺序 inspector面板中的 Order in Layer

层次越高越后渲染 从而覆盖前面的图片



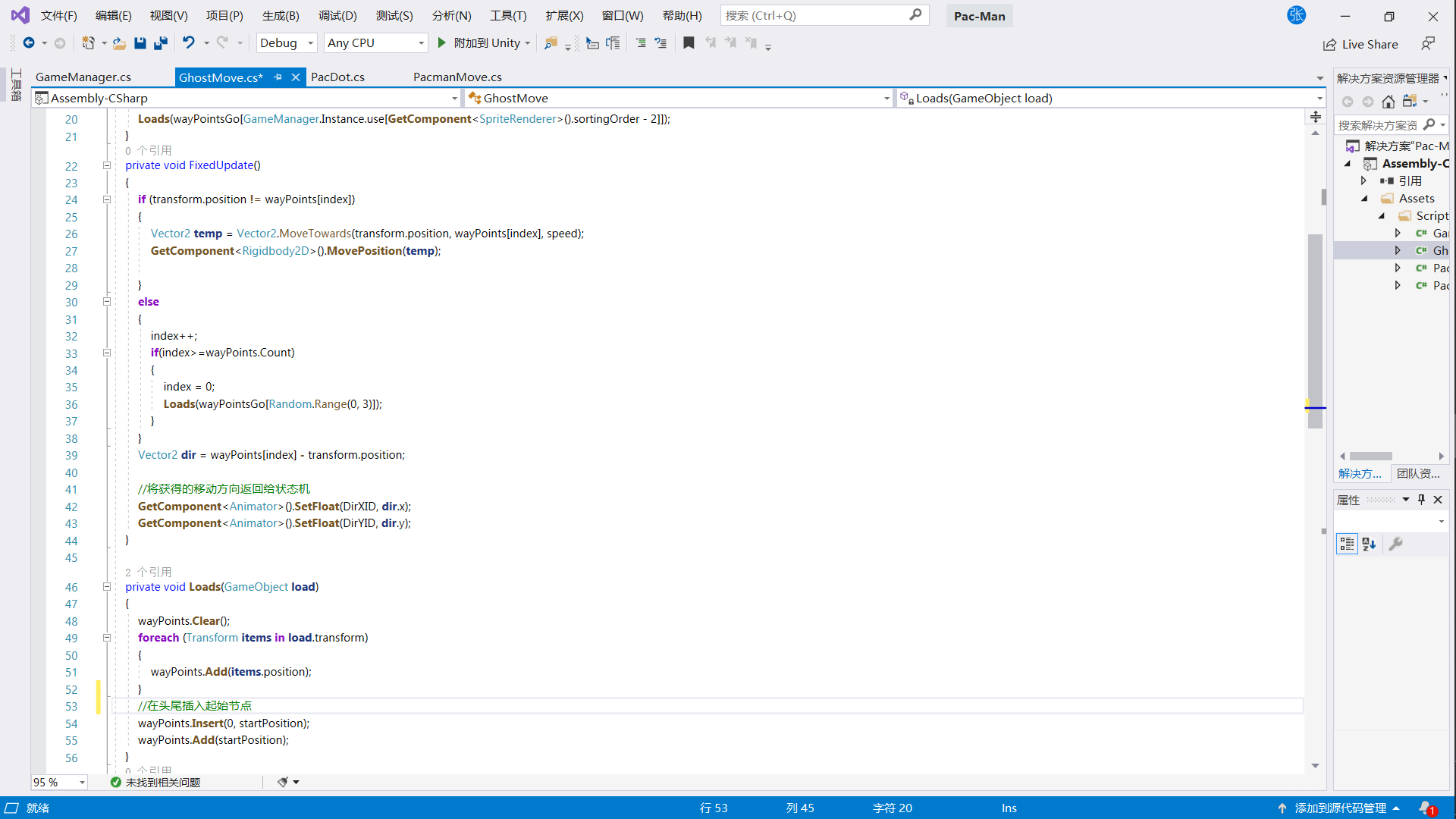
10.控制ghost 的移动方法

模拟ai的行为：

1.设置一系列的点作为ghost即将运行的路径的标点 全部设置为触发器来防止对角色移动产生影响.

2.将标点的坐标赋值给一个数组存储,根据之前角色移动的脚本的思维来控制ghost按照目标点的方向进行移动

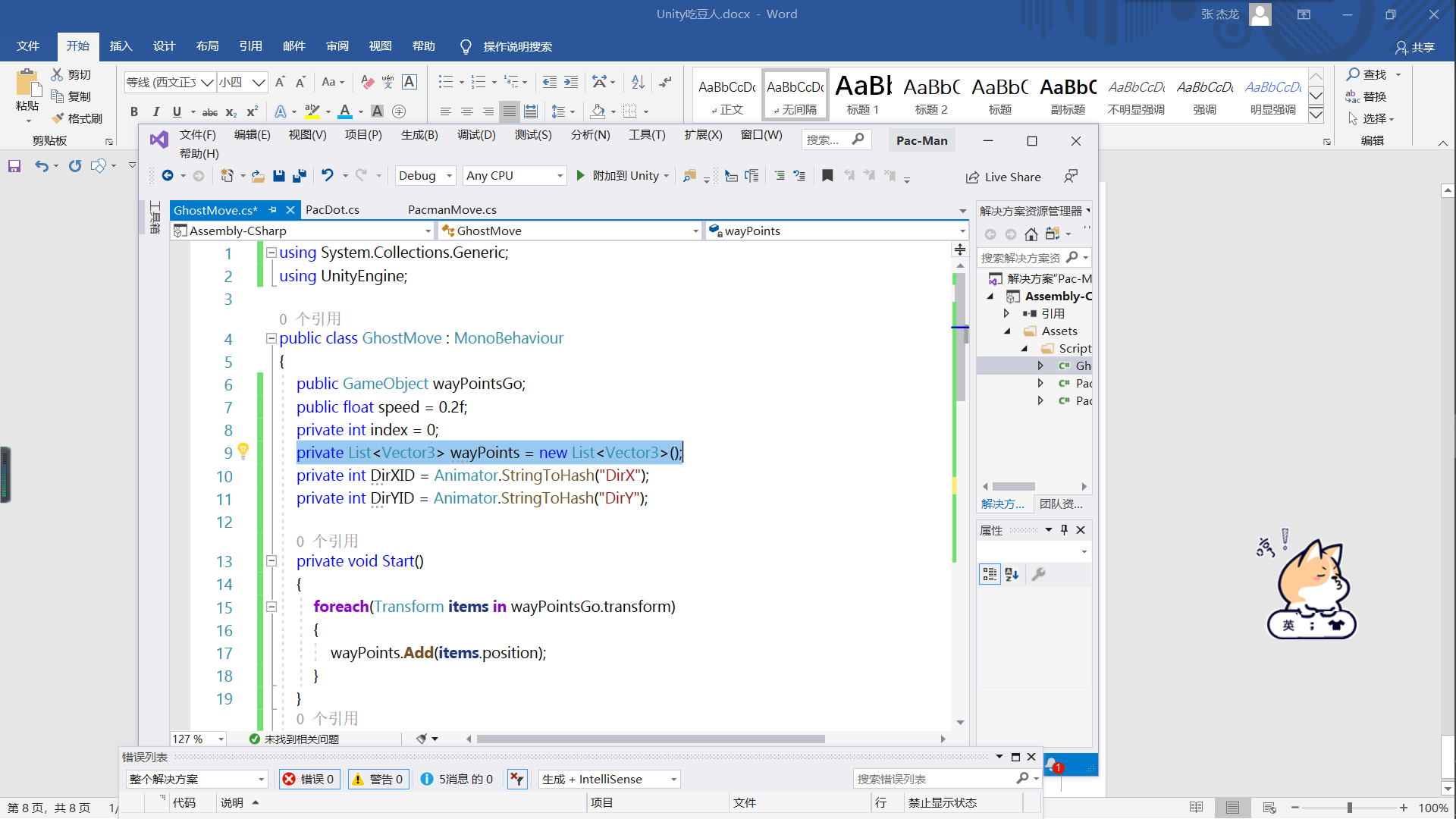
3.注意该情况下必须进行循环移动，所以数组的下标必须取余.



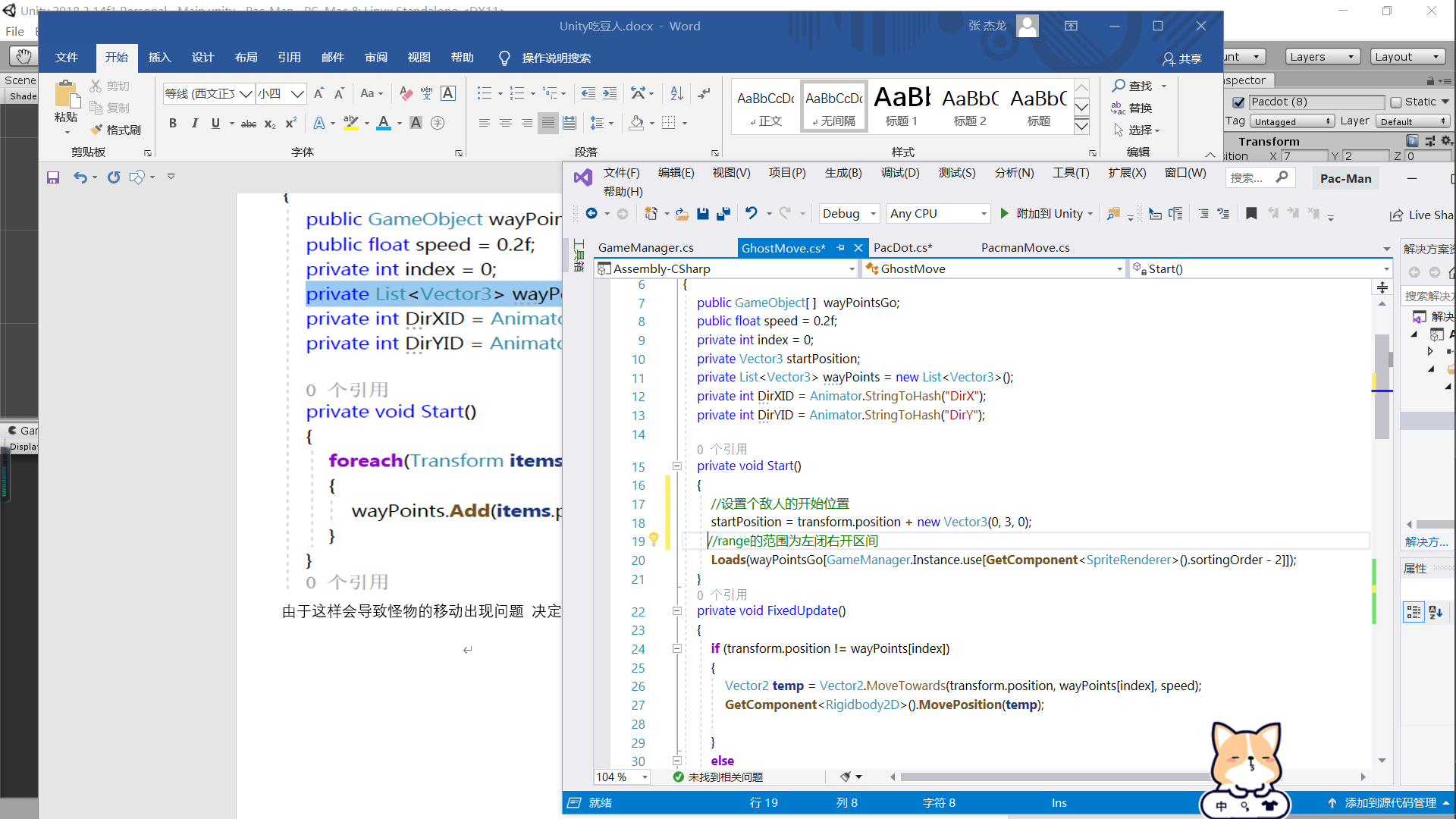
11.改进ghost的移动

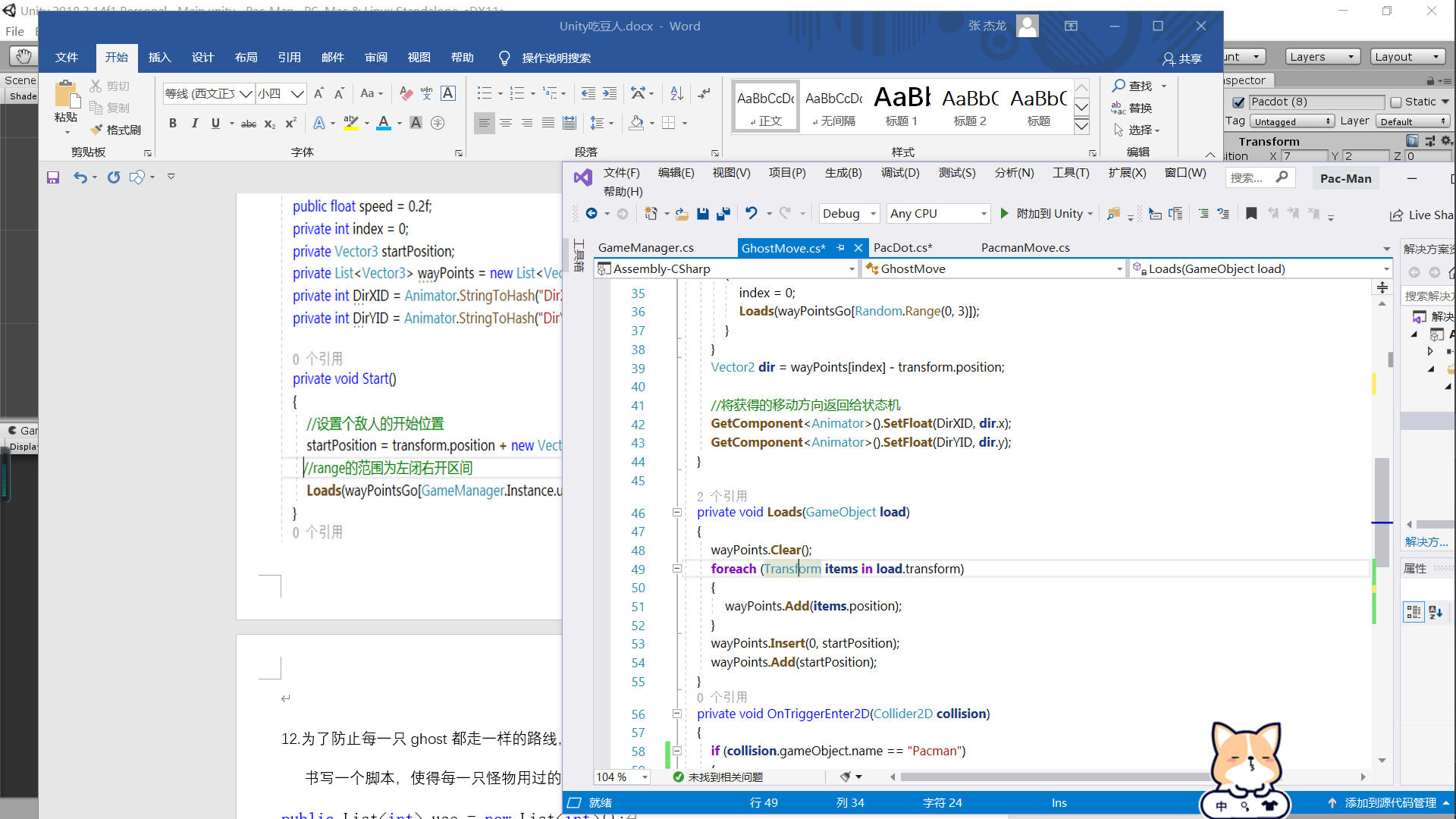
利用list存储position

Foreach的方法



由于这样会导致怪物的移动出现问题 决定把怪物的起始点分别设在不同地方 再把起始点插入list





12.为了防止每一只ghost都走一样的路线，进行再次改动

书写一个脚本，使得每一只怪物用过的index从list<int>中删除,从而保证不会重复

public List<int> use = new List<int>();

public List<int> wait = new List<int> { 0, 1, 2, 3 };

private void Awake()

{

\_instance = this;

int tempCount = wait.Count;

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

{

int temp = Random.Range(0, wait.Count);//使用range从剩下的数中取值

use.Add(wait[temp]); //添加到我们需要的数组中

wait.Remove(wait[temp]);//在原来的list中删除

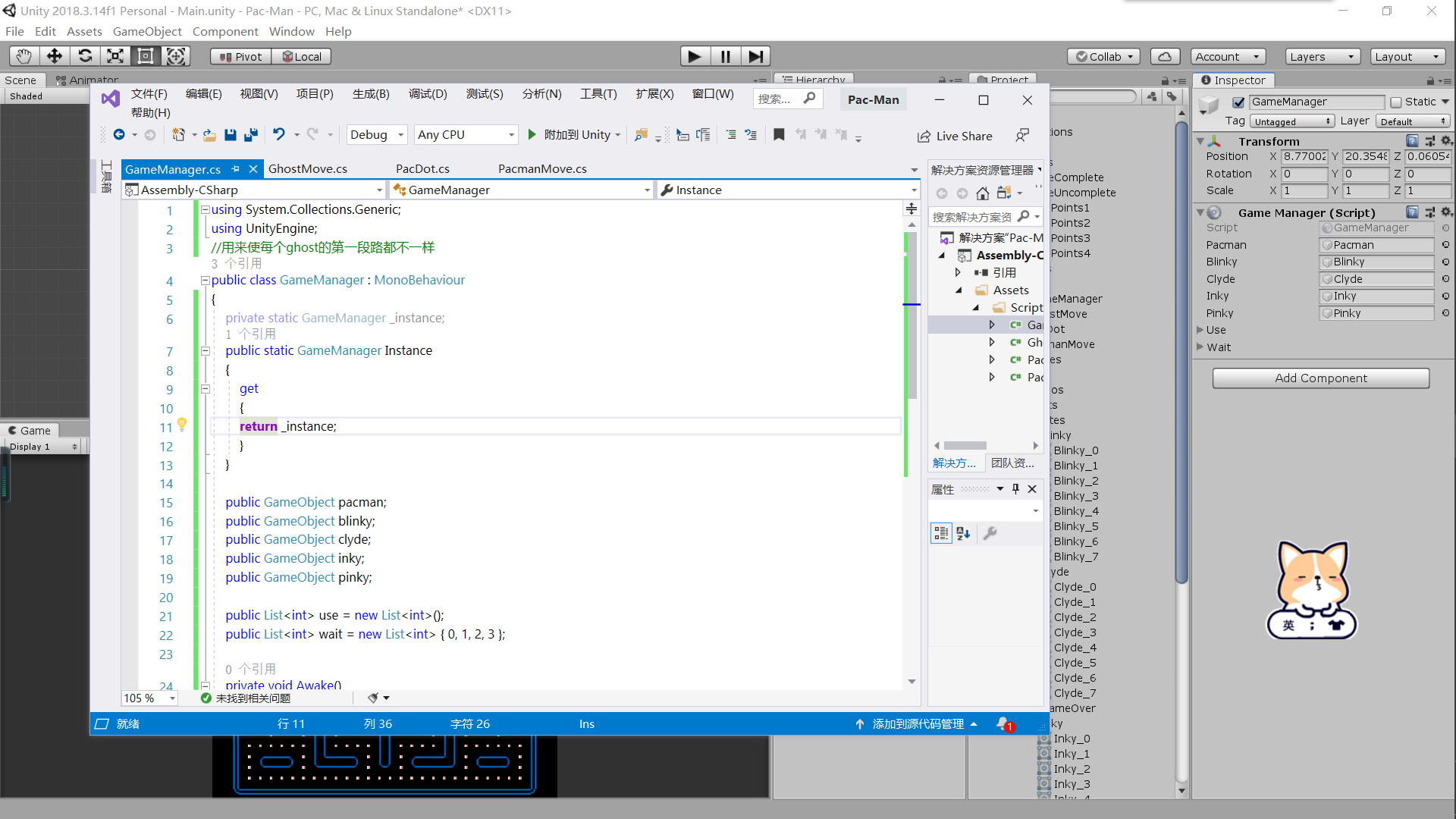
}

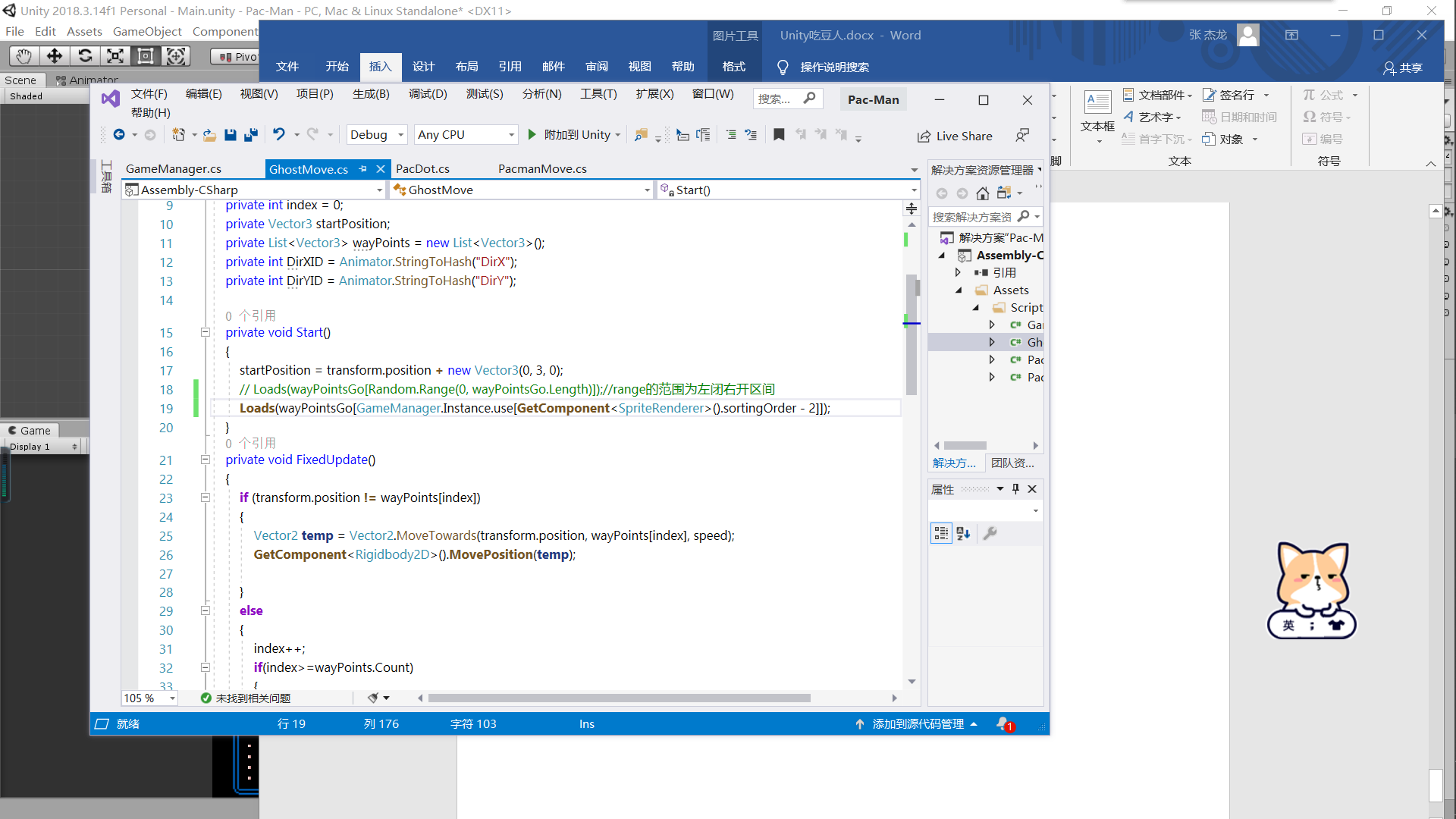
}

13.由于要调用上述脚本 需要用到单例方法

<https://www.cnblogs.com/Dean27/p/7744452.html>

对单例方法还不熟悉 但大致就是能够把自身的函数给别人使用





这句很长的话就是调用了单例里面的use方法 而后面的

[**GetComponent**<SpriteRenderer>().sortingOrder - 2]

获取了怪物身上的SpriteRenderer组件 根据渲染的层级来选择调用list的index

14.实现超级豆

超级豆的主要影响：

1.超级豆本身比普通豆大->

pacdotGos[tempIndex]**.transform.localScale** = new **Vector3**(3, 3, 3)

2.使主角能够吃掉怪物且怪物被吃后直接回家

**private void OnTriggerEnter2D(Collider2D collision)**

**{**

**if (collision.gameObject.name == "Pacman")**

**{**

**if (GameManager.Instance.isSuperPacman)**

**{**

**//回家**

**transform.position = startPosition - new Vector3(0, 3, 0);**

**index = 0;** **//回到起始点**

**}**

**else**

**{**

**Destroy(collision.gameObject);**

**}**

**}**

**}**

3.冻结敌人

private void FreezeEnemy()

{

blinky.GetComponent<**GhostMove**>().enabled = false;

pinky.GetComponent<**GhostMove**>().enabled = false;

inky.GetComponent<**GhostMove**>().enabled = false;

clyde.GetComponent<**GhostMove**>().enabled = false;

}

Ps：游戏物体的开启关闭用 **SetActive** 脚本的开启关闭用enabled

超级豆实现逻辑概括

1.用list存储一系列豆子的位置 被吃掉的豆子从该list中移除

2.随机在其中选择一个变成超级豆

3.吃下超级豆以后变成超级吃豆人

4.时效过后超级吃豆人消失

具体代码见u3d

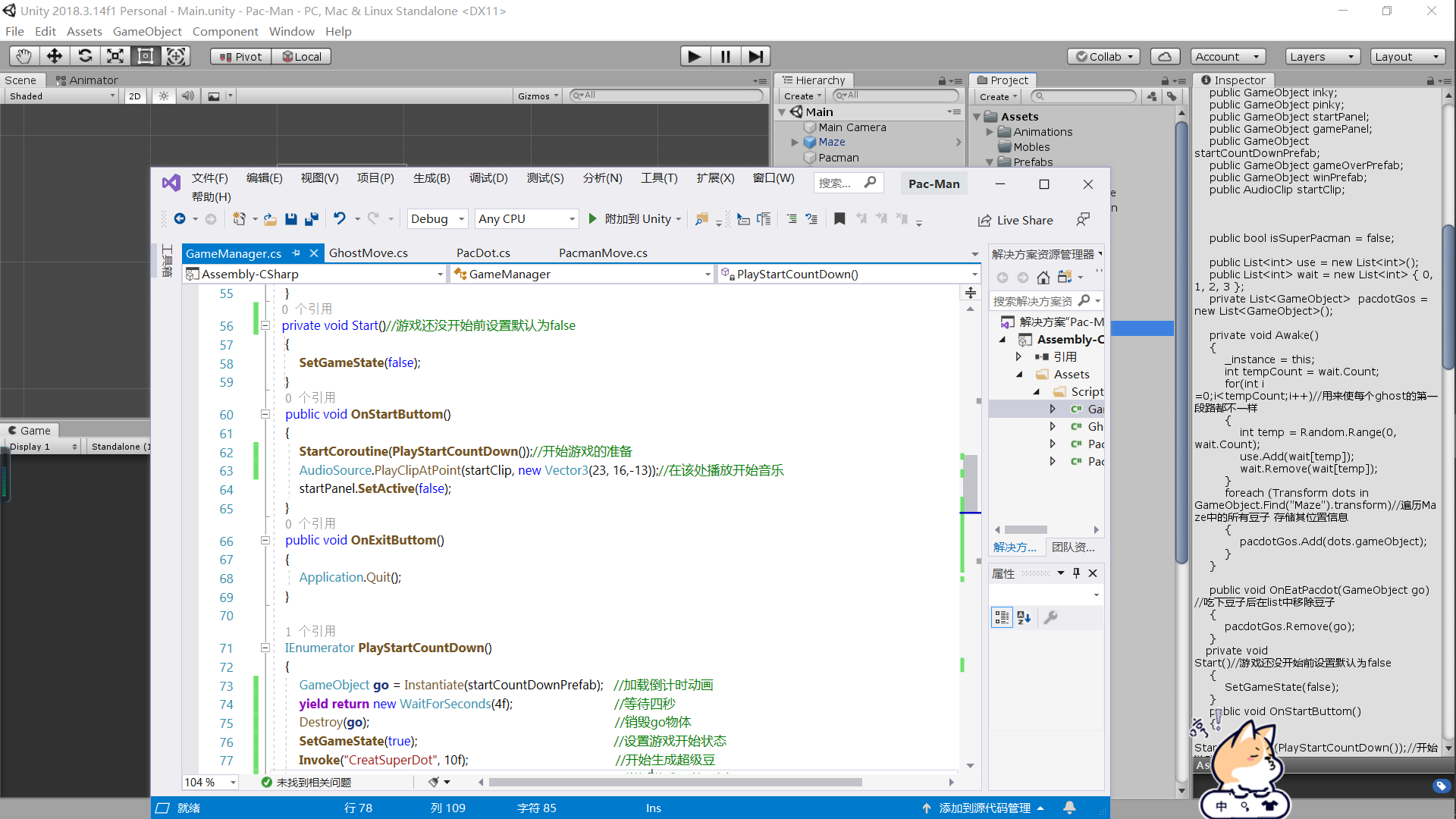
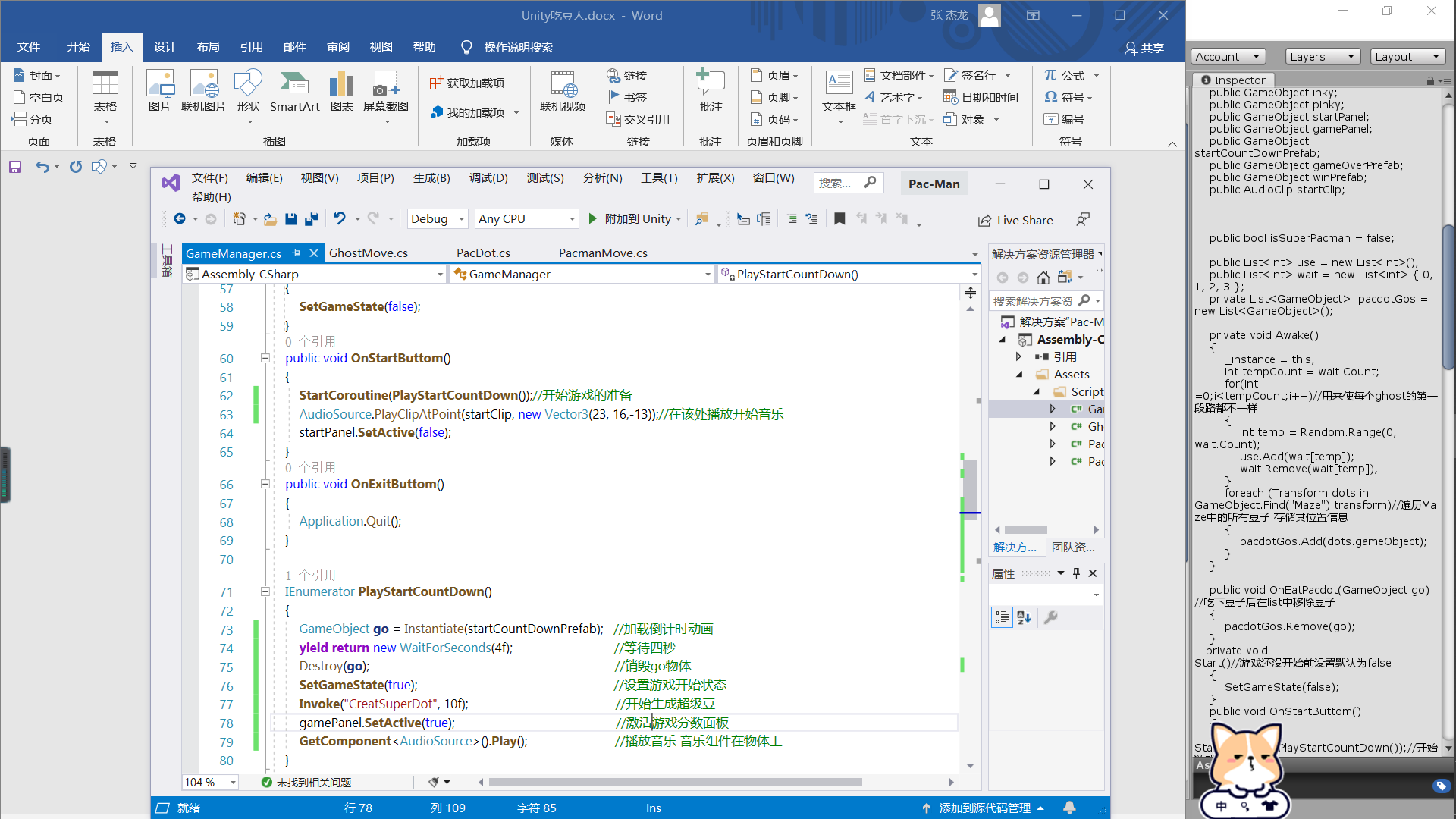
15.UI部分的逻辑实现

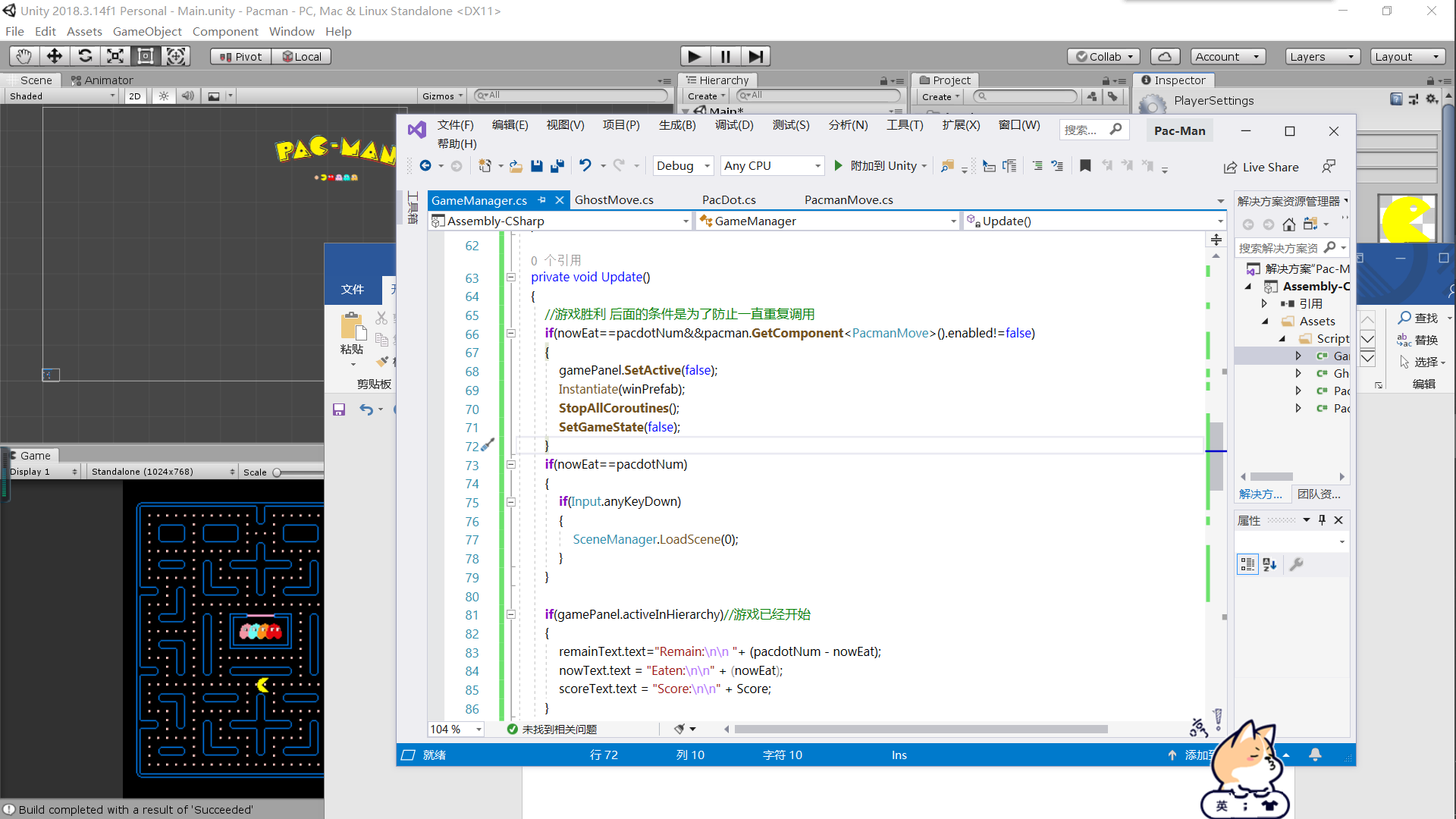
1.将文本Start和Exit设置为按钮

2.在start还没点击前禁用掉敌人和吃豆人的脚本

3.点击start后 等待四秒的同时运行开始动画

4.动画结束的同时将脚本激活并且开始超级豆函数



16.分数统计  
pacdoNum在awake/start方法中就先计算出来

分数的显示如下图 分数的计算在各种碰撞里面写

17.项目中用到的一些比较高级的方法

1.单例

public class GameManager : MonoBehaviour

{

private static GameManager \_instance;

public static GameManager Instance

{

get

{

return \_instance;

}

}

创建了该 Instance以后 可以利用Gamemanger.Instance.xxx来调用里面的东西

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.gameObject.name == "Pacman")

{

if (GameManager.Instance.isSuperPacman)

{

GameManager.Instance.Score += 500;

//回家

transform.position = startPosition - new Vector3(0, 3, 0);

index = 0;//回到起始点

}

else

{

collision.gameObject.SetActive(false);

GameManager.Instance.gamePanel.SetActive(false);

Instantiate(GameManager.Instance.gameOverPrefab);

Invoke("ReStart", 3f);

}

}

}

2.屏幕比例

Screen.SetResolution(1024, 768, false);

3.延时调用

Invoke("CreatSuperDot", 10f);//10s以后再生成超级豆

4.关闭进程

Application.Quit();

5.协程

Unity中的协程方法通过yield这个特殊的属性可以在任何位置、任意时刻暂停。也可以在指定的时间或事件后继续执行，而不影响上一次执行的就结果，提供了极大地便利性和实用性。   
协程在每次执行时都会新建一个（伪）新线程来执行，而不会影响主线程的执行情况。

下面是此案例中的两个例子

IEnumerator PlayStartCountDown()

{

GameObject go = Instantiate(startCountDownPrefab); //加载倒计时动画

yield return new WaitForSeconds(4f); //等待四秒

Destroy(go); //销毁go物体

SetGameState(true); //设置游戏开始状态

Invoke("CreatSuperDot", 10f); //开始生成超级豆

gamePanel.SetActive(true); //激活游戏分数面板

GetComponent<AudioSource>().Play(); //播放音乐 音乐组件在物体上

}

public void OnStartButtom()

{

StartCoroutine(PlayStartCountDown());//开始游戏的准备

AudioSource.PlayClipAtPoint(startClip, new Vector3(23, 16,-13));//在该处播放开始音乐

startPanel.SetActive(false);

}

public void OnEatSuperPacdot() //吃下超级豆以后

{

Score += 200;

isSuperPacman = true;//标识为超级吃豆人

FreezeEnemy();//冻住敌人

StartCoroutine(RecoveryEnemy());

Invoke("CreatSuperDot", 10f);//10s以后再生成超级豆

}

IEnumerator RecoveryEnemy()

{

yield return new WaitForSeconds(3f);//等待三秒

UnFreezeEnemy(); //敌人解冻

isSuperPacman = false; //重新变回普通吃豆人

}