**1.消除图像缩放影响**

**一、基本信息**

**测试版本：**v0.0.0.1

**测试平台：**Windows

**测试人员：**陈佳皓

**二、测试目标**

游戏对象的主要交互舞台在Stage/.../content/下，content具有组件，支持地图的缩放操作，中间有若干渲染组件影响对象的缩放比例。为此，若忽略此影响，则content上运行对象在地图缩放时会表现出奇怪的渲染表现，譬如地图缩小时对象放大，反之亦然，以及对象生成时会与预期地点产生漂移等。

由于这是一个广泛存在的代码缺陷，复现难度为0，故测试的主要目标在于定位bug起源以及对其进行修复。

**三、测试方法**

综合个人经验考虑，可辨明这其实是初学Unity时常见的错误，即遗漏实例对象在父子对象关系中受到的比例缩放影响，故只需在实例对象的生成初始化过程中加入消除缩放比例代码即可。

测试范围：content下有且仅有两类对象：Pawn类实例对象与ItemInstance类实例对象，对这两类对象各自进行测试，即可覆盖所有分支。

**四、测试结果**

加入消除缩放影响代码后，Stage/.../content/下运行的对象全部按照预期跟随content进行缩放，且不再产生漂移问题。

**五、总结**

Unity引擎中，会先根据原始数据，如世界坐标确定对象的绝对排布，再根据相对关系和相对缩放比例进行视觉层面的计算，最后通过渲染管线向Camera输出渲染出的画面，并反映在玩家屏幕上。而unity editor对象的属性中position是世界坐标，scale是相对缩放比例，在IDE环境中也存在LocalScale/LossyScale, position/local position等，非常易混淆，需要仔细理解并甄别。

**2.Pawn挤占同一地块问题**

**一、基本信息**

**测试版本：**v0.0.0.1

**测试平台：**Windows

**测试人员：**陈佳皓

**二、测试目标**

游戏对象的基本交互之一就是Pawn对象的移动，Pawn对象移动的起点与终点一定是与地格对应。在设计上，为了避免一些可能性的数据结构与交互上的不便，一个地块应只能同时站立一个Pawn。为此，需要构建测试用例，检查可能使得出现两个Pawn位于同一地块bug出现的样例并修复可能的bug。

**三、测试方法**

指定四类测试样例：

1.PawnA移动至地格1，在抵达之前让PawnB也移动至地格1

2.PawnA移动至地格1，抵达之后让PawnB移动至地格1

3.PawnA移动至地格1，抵达之前让PawnA移动至地格2，再让PawnB移动至地格1

4.PawnA移动至地格1，抵达之后让PawnA移动至地格2，再让PawnB移动至地格1

观察每组样例运行过程中各Pawn与各地格参数的取值情况，检查其是否符合预期。若不符合，需要定位bug成因和源头，并设法修复。

**四、测试结果**

经测试发现，样例1.的情况下，PawnA会和PawnB移动至同一格，产生bug，样例3.的情况下，PawnB无法移动至地格1，同样属于bug。故在之后引入与进程锁类似的“路程锁”，即：在PawnA试图移动到地格1时，让地格1先被PawnA占用，等待PawnA离开原先地格后，再让原先地格解除被PawnA的占用，如果PawnA半途被只会去别的地格，则立刻接触原先目的地的占用，从而修复此bug。

不过这样的做法是参照多线程朴素算法进行的设计，故而在今后更特殊的多线程情况下仍有可能会产生一定bug，有待进一步的优化和观察。

**五、总结**

不管是基于Managers架构还是组件化架构，当多个同类型对象对同一可交互对象发生交互时，一定会形成一个小型多线程问题，为此其实可以参考操作系统中解决多线程问题的结构，从而更好的解决和维护这些问题。

**3.管线渲染冲突问题**

**一、基本信息**

**测试版本：**v0.0.0.0

**测试平台：**Windows

**测试人员：**陈佳皓

**二、测试目标**

项目设计中，地图UI与系统UI产生了错误的图层关系，导致系统UI被地图UI覆盖，无法交互。需要解决这一问题。

**三、测试方法**

由于对于unity引擎中图形化的认知和学习较少，故而直接询问大预言模型以获取答案。

**四、测试结果**

经询问得以知晓，在unity引擎中，mesh对象的管线和ui组件的管线是分离的，其中mesh渲染以sorting layer与order in layer属性作为渲染顺序的参考，而ui渲染会因canvas render mode而不同。故经由调整之后即可解决图层关系问题。

**五、总结**

尽管这样询问大模型可以快速解决眼下需要解决的问题，但是如果想要对这些问题，例如Unity渲染的底层原理有更进一步的了解和运用，仍需要发挥主观能动性手动追问或学习。

**4.MouseInteract获取Sprite冲突问题**

**一、基本信息**

**测试版本：**v0.0.0.2

**测试平台：**Windows

**测试人员：**陈佳皓

**二、测试目标**

MouseInteractManager负责捕获所有的鼠标操作，并利用状态机和多态模型记录用户目前行为并发生相应交互。而其获取此信息的主要来源是通过unity引擎的raycast接口，通过鼠标位置在模拟三维空间中射线与各对象collider的交叉检测，来判断用户的点击行为在与什么对象发生交互。

这样的实现一定会遇到一种问题，即当多个对象的collider重叠时，要如何确保用户可以通过操纵以选中其中任一对象，为此，需要对该行为的结果进行测试与分析，并通过一定算法实现需求。

**三、测试方法**

手动构建一个若干collider重叠在一起的情况，并对该重叠区进行若干次点击，观察其行为。其中重叠在一起的collider要保证其附着的对象类型覆盖尽量全面，并且同类对象间可以相互区分。

此次测试中，人工构建了UI类对象，ItemInstance类对象，Pawn类对象collider重叠的情况，并对其进行测试。

**四、测试结果**

进行若干次点击，其点击到的对象完全一致，这是由raycast性质导致的，传统情况下，它会选择第一个命中的collider信息并将其返回。

为此需要增加一步处理算法以达成目标功能：先获取raycast命中的所有collider，再通过mesh渲染层的sorting layer和order in layer顺序为其进行排序，排序后根据当前选中的目标对象，在再次点击后循环选中其下一个对象，从而可以选中其中任一对象。

**五、总结**

尽管这样的做法可以解决眼下的问题，实际测试发现，对于构建的用例中，这样的操作模式实则仍然不太方便，更具体来说，UI的交互优先级应时刻保持最高。因此在未来的更新中，仍需要对此开展优化，或者在以后的设计中，需要预先将这方面的功能交互作更细致的考量。

**5.HarvesrList引用传递问题**

**一、基本信息**

**测试版本：**v0.0.0.2

**测试平台：**Windows

**测试人员：**陈佳皓、丁一萌

**二、测试目标**

在测试Pawn与作物的Harvest交互中，观察到有且仅有第一次Harvest可以正常产生掉落物，而之后的Harvest都会涉及到NullReferenceExcept，这是显然且严重的bug。因此，需要对其进行定位和修复。

**三、测试方法**

假定问题源自于此问题，通过ToList将其转化为值传递后进行测试，检查其行为表现是否与原本一致。从而进行排查。

**四、测试结果**

在将原列表通过ToList接口进行先拷贝再传递后，所有的Harvest任务都能正常生成掉落物，说明定位成功，bug已修复。

**五、总结**

在unity，或者说C#中，值传递与引用传递是一组非常容易混淆的点（也是面试常见考点）。对于面向对象语言而言，C++主要通过指针和显式引用作以区分，在java中均是值传递，只是在传递引用时传递了引用对象指针的拷贝，C#中两者的区分度介于其它两种语言之间，更需要留个心眼。

**6.装载任务装载数目错误问题**

**一、基本信息**

**测试版本:** v0.0.0.2

**测试平台：**Windows

**测试人员：**李宇清

**二、测试目标**

在进行建筑任务时，发现pawn在运输时会将地上所有的建筑材料运走随后销毁item，并不符合如果要求数量少于物品数量时只装载对应数量的物品，更改原地item的数量并不销毁它的预期

**三、测试方法**

在测试脚本中编写专门的装载测试函数，并在装载函数的两个分支语句以及他们前面设置debug语句检测程序运行。测试函数先创建可设置数量的木材，随后让小人调用装载任务进行装载，观察debug信息以及unity中实际的游戏显示。设置不同木材数量以及运载任务要求数量用来调试各种情况下的表现，分别在物品数量等于要求数量、物品数量大于要求数量、物品数量小于要求数量三种情况下设置了两次测试。

**四、测试结果**

无论采用何种数量对比情况下的测试，装载逻辑均进入了拿取所有物品数量并销毁的分支语句，结论是分支条件设置错误。

**五、总结**

设置分支语句需要更加小心，特别是在增加代码功能后一定要重新检查之前的代码分支逻辑，避免出现错误。

**7.SLEEP调用导致的运行游戏整体卡顿问题**

**一、基本信息**

**测试版本:** v0.0.0.1

**测试平台：**Windows

**测试人员：**李宇清

**二、测试目标**

在进行最初版建造农田时发现如果让小人进入工作时间既等待一段时间后再进行创建农田操作时会造成游戏整体进入停滞而非单独小人进入停滞，为实现工作时间的差异化必须对这个bug进行修复。

**三、测试方法**

为确定停滞位置，在建造农田任务各个部分前加上debug语句，经过测试确定了正是工作等待部分代码的错误，注释掉这部分后程序正常运行证实了这一点

**四、测试结果**

确定了sleep函数导致了问题产生，询问chatgpt后了解了sleep在unity中的效果是使整体休眠一段时间，故而改用协程方式实现等待，解决问题

**五、总结**

不熟悉sleep在unity中的效果导致的错误

1. **运输任务中小人直接前往目的地问题**

**一、基本信息**

**测试版本:** v0.0.0.2

**测试平台：**Windows

**测试人员：**李宇清

**二、测试目标**

在进行建筑任务时，注意到小人在初始运输阶段出现直接前往运输目的地点而没有先前往所需运输物品处再前往运输目的地点。这个关键bug会导致所有运输任务不符合预期，需要紧急修复。

**三、测试方法**

构筑测试运输任务的测试函数放入测试脚本，其主要流程是创建一定数量木头，发布一个对于这些木头的运输任务。同时在运输任务中设置debug语句对于任务进行流程进行检测，在controller中设置debug语句对于小人的当前位置、目标位置进行输出。

在unity的左上角实时显示每个小人的任务执行情况，从而检测小人是否是因为任务分配出现错误。

**四、测试结果**

经过多轮测试，定位问题到“由于没有等待第一次由运输任务发布的移动任务完成，在pawn还在ismoving的时候第二次移动任务将目标位置更改，由于代码执行速度很快而小人还没来得及移动，视觉效果上就是小人跳过向物品移动的过程直接向目标地点进行了移动” 因而我们需要检查更改我们的等待移动任务完成逻辑。 检查发现该段逻辑采用协程处理，但其等待效果是等待这段代码执行完毕（实际上移动任务只是设定好目标地点和ismoving就执行完了，小人并未直接完成运动）代码执行与实际运行不同。于是更改这部分代码为等待直到小人的ismoving参数变为false既完成移动后才执行下一步，于是概率性解决了这个问题。

**五、总结**

在编写代码时要注意实际操作的执行时间与代码运行时间的差别相当大，在执行等待逻辑的时候需要注意。

1. **运输任务中小人直接前往目的地问题（概率性失效）**

**一、基本信息**

**测试版本:** v0.0.0.2

**测试平台：**Windows

**测试人员：**李宇清

1. **测试目标**

紧接着上一个问题，在修复了上面的运输任务问题后，运输问题出现了概率性直接前往目的地的问题，说明仍有未解决的问题尚未发现，需要测试

**三、测试方法**

复用上一个问题中的测试脚本，针对性加入pawn.id的属性，针对性对于出现问题的小人id进行调试

**四、测试结果**

移动任务出现bug，原因是移动前会先修改目标地点的has\_pawn数值，如果这段代码的运行速度快于controller中的has\_pawn判定运行，就会让小人判断远处目的地点拥有小人而出现无法移动问题。