深入探索AOI算法

co lin

发布于 2020-08-28 00:13 ，编辑于 2020-08-28 09:42

AOI概述

网上关于AOI算法的文章很多了，但大多语焉不详，一上来就9宫格十字链表，直接把人整懵。本文试图由浅入深的介绍AOI算法的形成，希望能把AOI解决的问题，以及它的核心逻辑讲清楚。如果读完之后能所有启发，那本文的目的就达到了。

AOI算法是大型多人在线的游戏服务器中一个非常重要的基础模块，它很大程度上决定了服务器的运行效率。那么什么是AOI呢？AOI全称为 `Area Of Interest` ，翻译过来叫 `感兴趣的区域` ，通俗的讲是一个游戏对象在场景中的视野，这个视野可以大到整个场景，也可以小到周围几米；它能观察到视野中的其它对象的一举一动，同时它也在某些对象的视野中，也被这些对象观察着。

每个对象需要维护到两个集合：

- **观察者集合**：就是关注我的对象集合，我的所有AOI行为都需要向这个集合发送事件，以便让他们观察到我的变化。
- **被观察者集合**：就是被我关注的对象集合，理论上只要有 `观察者集合` 就够了，为什么还需要维护一个 `被观察者集合` 呢？因为有时候想主动检查对象的状态，比如怪物AI会定时检查被观察者集合的距离，决定是否发动攻击；又比如释放技能需要遍历被观察者集合，判断它们是否命中。如果没有被观察者集合，就必须遍历整个场景的对象。

有些游戏对象同时拥有这两个集合，有些只拥有其中一个，假设场景中有玩家，怪物，NPC，掉落物：

- 对玩家来说：它能观察到所有类型的游戏对象；同时它会被其他玩家和怪物观察着；但它可能不会被NPC和掉落物观察。所以它的被观察者集合是所有类型的游戏对象，观察者集合是玩家和怪物。
- 对于NPC和掉落物来说：它不关心周围的对象，所以它没有被观察集合；但它有观察者集合，里面只有玩家类型。
- 怪物多样化一些：
  - 主动怪的被观察者集合是玩家；观察者集合也是玩家。
  - 被动怪可以设计为没有被观察者集合，因为在它没有被玩家攻击之前，它是不关心周围的环境，玩家攻击它之后，玩家会进入它的仇恨者列表，这是怪物AI的范畴了；它的观察者集合是玩家。

围绕这两个集合，AOI算法的设计要考虑以下因素：

- 对象视野的大小，这直接影响到集合的大小。
- 对象集合保存在哪里，有些做法是场景共享对象集合，有些则是每个对象单独保存这两个集合。
- 在进入离开移动等事件中，如何更新对象集合。

现在我们就一步步地探索AOI算法。

上帝视角

把整个场景当成可见范围，无论我走到哪里，都能看到场景中的所有对象。每个对象就像上帝一样，能观察到其他对象的行为。

这种做法是为最简单直接的AOI管理，场景管理器只需用一个字典保存所有游戏对象，另有一个字典按对象类型保存对象集合就可以了。被观察者集合和观察者集合在需要的时候直接从场景管理器中取。

知乎 首发于 游戏服务器编程

进入

我进入场景，取出所有对象，向我发送 `Enter(对象)` 事件，我会向客户端发送对象集进入的协议，这样我的客户端便能看到场景中的所有对象。

将我加入场景管理器。

取出其他玩家集合，向它们一个个发送 `Enter(我)` ，玩家会向它的客户端发送我进入的协议，这样它就能看到我在场景中，即便我其实离屏幕很远也是这样。

可能不需要向怪物集合发送Enter消息，因为怪物AI会自己去遍历敌人。

离开

我离开场景，将我从场景管理器删除。

取出玩家集合，向它们一个个发送 `Leave(我)` 事件，玩家会向它的客户端发送我离开的协议。

可能不需要向我发送 `Leave(对象)` 事件，因为我跳场景后，客户端会自动把场景中的对象删除。

更新

我在场景中的行为称为更新，比如移动，换装，发技能等等，其他玩家应该能看到我这些行为。

取出玩家集合，向它们发送相应的事件，他们会向客户端发送相应的协议，这样客户端就能看到我的行为了。

上帝视角的AOI其实是非常简单可靠的，如果预估场景中的对象最多几十个，那么我建议直接用这种方式，它即足够高效，也不用每个对象单独保存观察者集合和被观察者集合，对内存很友好，同时它很简单，几乎不大可能出错。

但当场景比较大，且对象数量达到数百上千的时候，这种方式就不适合了，因为每个对象的状态更新需要通知上千个其他对象，交叉起来能达到百万级别的量。再加上客户端承受不了上千的游戏对象，我们应该减少对象的视野。

减少视野

一旦限定了对象的视野，AOI算法就开始变得复杂；而且每个对象需要维护被观察者集合和观察者集合，内存占用会大大增加。

每种对象的视野可以不同，比如玩家的视野只比一个屏幕大一点点，有些BOSS需要更大的视野，而NPC可能视野为0，视野为0的对象不关注别人，只被别人关注。

看看减少视野后的处理

进入

我进入场景。

遍历场景中所有对象，逐一比较它们和我的距离，如果对象在我的视野之内，则向我发送 `Enter(对象)` 事件，此时**这些对象会加入我的被观察者集合**，同时，**我会加入到这些对象的观察者集合**。

同样如果距离小于对象的视野，则向对象发送 `Enter(我)` 事件，此时**我会加入到对象的被观察者集合**，同时**对象会加入到我的观察者集合**。

离开

我离开场景。

遍历我的被观察者集合，**将我从这些对象的观察者集合中删除**，将我的被观察者集合清空。

更新

这里的更新不包括移动，因为移动会导致对象集合变化。遍历我的观察者集合，向它们发送相应的更新事件。

移动

我移动的时候，被观察者集合和观察者集合都会发生变动，现在我们没有好的办法优化它，只能这样做：

遍历场景的所有对象：

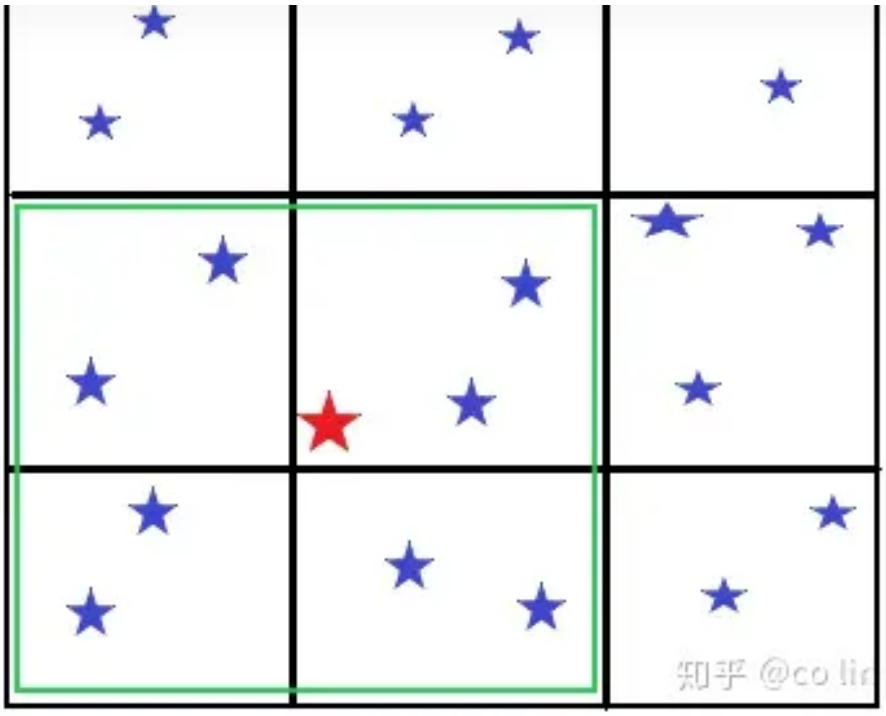
- 如果它原来在我的被观察者集合中，并且现在的距离已经大于我的视野，向我发送 `Leave(对象)` 事件，此时**对象会从我的被观察者集合删除**，同时**我会从对象的观察者集合删除**。
- 如果它原来在我的观察者集合中，并且现在的距离已经大于它的视野，则向它发送 `Leave(我)` 事件，此时**我会从它的被观察者集合中删除**，同时**它会从我的观察者集合中删除**。
- 向剩下的观察者集合发送移动事件。
- 如果它原来没有在我的被观察者集合中，并且现在的距离已经小于等于我的视野，向我发送 `Enter(对象)` 事件，此时**这些对象会加入我的被观察者集合**，同时，**我会加入到这些对象的观察者集合**。
- 如果它原来没有在我的观察者集合中，并且现在的距离已经小于等于它的视野，向它发送 `Enter(我)` 事件，此时**我会加入到对象的被观察者集合**，同时**对象会加入到我的观察者集合**。

别被这段话绕晕，实际上它的逻辑是很清楚的，请仔细理解这段话。

可以看到移动是最大的性能瓶颈，每次移动需遍历场景中的所有对象，如果每个人都在移动，那这个服务器的承载力可想而知。现在的优化方向转向如何减少对象的遍历，稍微思索后，我们能得到一个解决方法：将场景划分格子。

网格化

将场景划分成等大的格子，1个格子大约为1/4屏幕大小，每个进来的对象根据坐标加入对应的格子中，如下图所示：



绿色框为屏幕大小，红星为我，现在我们把最大视野限定为9宫格，这样搜索范围就缩小为9个格子，需要遍历的对象数量大大减少。

进入

我进入场景，马上计算出我所在的格子，并加入这个格子。

接下来的做法和上面的进入完全一样，只不过搜索的范围变成这9个宫格子，具体不再描述。

离开

我离开场景，将我从格子删除，

接下来的做法和上面的离开完全一样。

移动

我移动的时候，遍历9宫格的所有对象，然后执行和上面的移动完全一样的逻辑

如果我移动到新的格子上时，还要多处理一些事件：

- 把我从原来的格子删除，加入新的格子。
- 假设旧的9宫格为OldGrid，新的9宫格为NewGrid，计算{NewGrid−OldGrid}集合，得到的这些格子即为新增的格子。然后对这些格子执行和 进入 完全一样的处理：

如果对象均匀的分布在场景中，且场景足够大，那这个优化的效果是非常显著的。

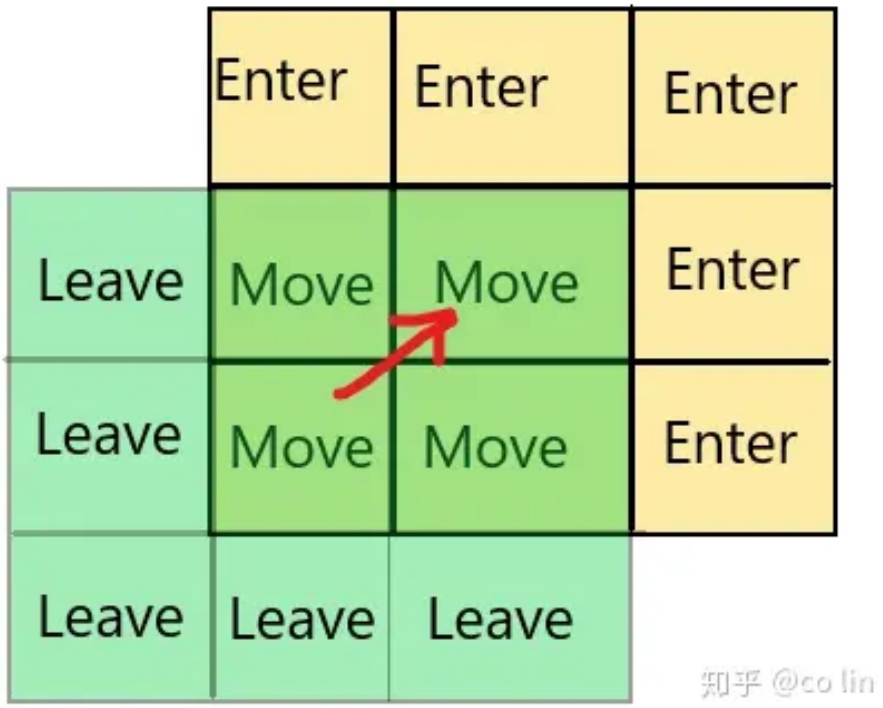
但如果像主城那样，玩家大多集中在一个区域，服务器的压力仍然会很大，原因是9个格子的玩家可能占了场景80%的玩家，这又退化成和遍历场景所有玩家差不多了。

假如我们把要求降低一些：强制对象的视野固定在9宫格上，也就是说，只要我在这个格子中，不管怎么移动，视野都是周围的9个格子，那事情就好办了。每个游戏对象不需要维护两个集合，直接从9宫格里取即可。现在逻辑简化成这样：

- 进入场景：进入一个格子，取出周围9格的对象，向它们发送 `Enter(我)` 事件，同时向我发送 `Enter(对象)` 事件。

知乎 首发于 游戏服务器编程

- 如果没跨格子，直接取9格的对象，向它们发送移动事件。
- 如果跨过格子，计算{OldGrid-NewGrid}，向它们发送 Leave(我) 事件，向我发送 Leave(对象) 事件；计算{NewGrid-OldGrid}集合，向它们发送 Enter(我) 事件，向我发送 Enter(对象 事件；计算{NewGrid*OldGrid}集合，向他们发送移动事件。



这个过程看起来就是上帝视角的缩小版，如果我们对视野要求不那么高，这个做法和上帝视角一样简单可靠。我相信很多游戏的9宫格处理都是用这种，或者基于这种去优化的。

这种做法的一个小小缺点是客户端会收到一些屏幕外的对象消息，有点浪费带宽吧，如果我们把发消息做成批量加压缩的方式，这种浪费并不大。

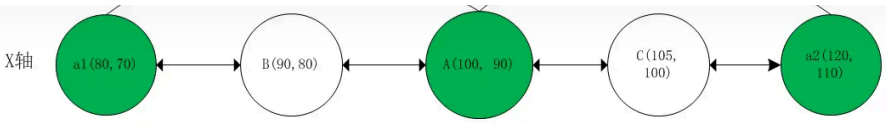
十字链表

基于网格的优化基本也就到这儿，如果想探索更多的优化方式，只能换一种数据结构，用十字链表，其核心思想是将所有对象结点链接在两个链表上，一个按X值排序，一个按Y值排序，对象进入场景后遍历两个链表，找到合适的位置插进去；移动的时候，从对象位置前后遍历两个链表，和其他对象进行判断。

简单的描述是这样子，可是当你按这个思路去实现十字链表时，你会发现搜索观察者和被观察者都很慢，因为每种对象的视野可能不一样，你没办法只前后遍历一点点，有可能在很远的地方有一个对象在观察着你，你只能整个链表遍历，这样的十字链表就没什么意义了。

真正的十字链表，除了对象结点外，还需要一种 哨兵结点 ，每个对象都带有两个哨兵结点，这两个哨兵结点同样被链接在X和Y链表上，哨兵结点也有坐标，它们的坐标刚好是对象坐标与其视野半径的差值，举个例子：

对象A的坐标是(100, 90)，假设对象的视野半径是20，那么哨兵1的坐标就是(80, 70)，哨兵2的坐标就是(120, 110)，它们按顺序加入链表，拿X链表举例，如下所示：



先不去管B和C，a1和a2是A的哨兵，并且A移动后，哨兵也会跟着移动，以保持它们的距离总是视野的半径。

那么哨兵结点的作用到底是什么呢？顾名思议，它们的作用是用来监控跨越它的对象结点的：

假如有一个D结点原来的坐标是(78, 69)，现在它移动到(82, 75)，坐标变动后，D结点需要从原来的位置向前移，必定会跨过a1这个结点，此时a1判断D是一个对象结点，且它原来的坐标在A的视野之外，现在的坐标到了A的视野之内，就可以确定D进入A的视野。哨兵向A发送 `Enter(D)` 事件：**D会加入A的被观察者集合**，同时，**A会加入到D的观察者集合**。

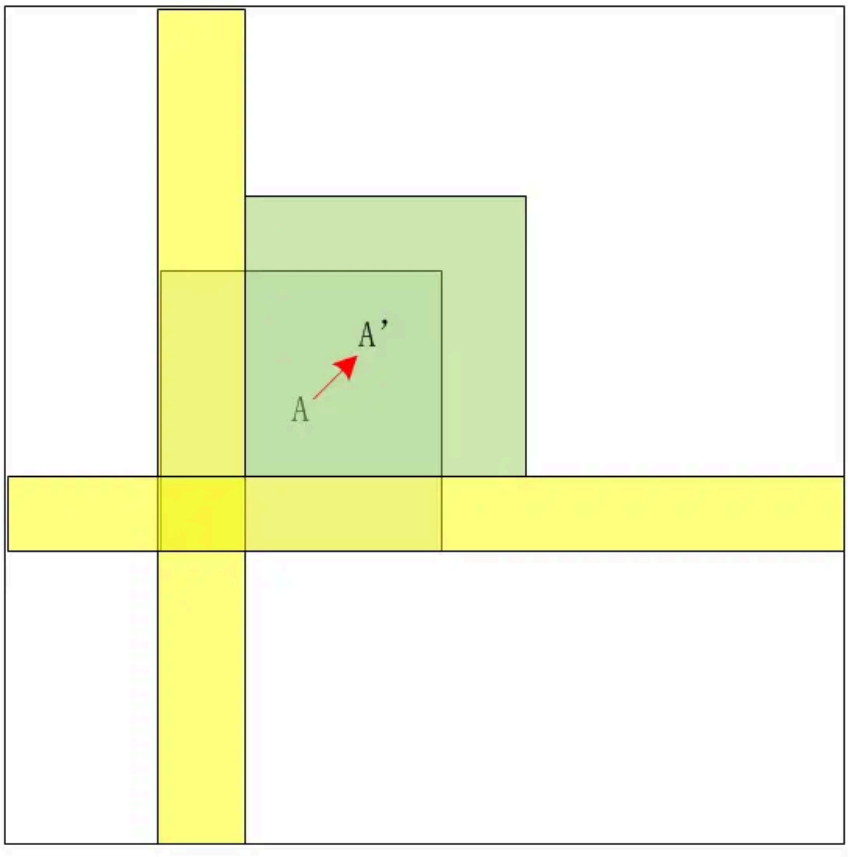
D在向前移动时，A也可能会进入D的视野，这个是怎么判定的呢？别忘了D也是有两个哨兵结点的，它们也跟着D向前移，其中一个哨兵越过A时判断：原来A的坐标在D的视野之外，现在A的坐标到了D的视野内，可以确定A进入D的视野，于是哨兵向D发送 `Enter(A)` 事件：**A会加入D的被观察者集合**，同时，**D会加入到A的观察者集合**。

如此一来一回，各方就慢慢维护起观察者集合和被观察集合。离开视野的处理也是类似，这里就不再罗索，留给你们去想。

总而言之，十字链表的的核心算法就是哨兵结点或对象结点在移动的时候，会跨过对方，在跨过的时候判断进入视野还是离开视野。

这个算法假设对象经常静止或小幅移动，对于大幅移动和进出场景是小概率事件，这个假设用在网游刚好非常凑效，所以十字链表非常高效。

下面用一个几何图来帮助理解十字链表：



当然也不是说十字链表很完美，它在角色经常进出场景的情况下就表现得不大好，因为角色每次进来场景，都需要从X和Y的链表头向前遍历，直到找到自己的位置。这个时间复杂度是O(N)。遇到那种经常跳场景去副本的游戏，十字链表可能会有点性能瓶颈。

针对这个问题，我想到了一种解决办法，注意看了：

- 首先需要像9宫格那样限定一个最大视野，比如任何对象的视野都不会超过1.5个屏幕大小。
- 接着我们定义一些 地标结点 ，它们的坐标在设定之后就不会改变，像是地图里的地标一样。假如一个地图的大小是1000x1000，我们可以创建10个地标结点，每个结点的坐标相距100大小：M1(0, 0), M2(100, 100), M3(200, 200), M4(300, 300)。。。
- 创建场景的时候，就把地标结点分别插入到X和Y链表中，地标结点的坐标不会改变，所以永远不用移动它们。
- 接着把这些地标结点按顺序保存在一个数组中：|M1|M2|M3...|，有了这个数组，我们就不用从头移动了。
- A进入场景后，它算出： 移动点=A的坐标-最大视野直径 ，然后在地标数组中快速查找到最近的地标结点(用二分查找法)，从这个地标结点开始向前移动，移动过程中A就会进入其他对象的视野。
- A移动完成之后，A身边的两个哨兵结点开始向两边移动，移动过程中就会有一些对象进入A的视野。

地标结点越多越精确，但遍历的结点也越多，这个需要实际实现的时候做试验。

AOI算法大体就是这样，大多数游戏用固定视野的9宫格算法就足够了，如果人数太多，我们完全可以用场景分线或分层的方式，强制把人数降下来，因为人数太多，对客户端的体验也不会好到哪里去。

本文到此就结束了，希望对正在研究AOI算法的你有所帮助。

发布于 2020-08-28 00:13 ，编辑于 2020-08-28 09:42

算法 游戏服务器开发

墨菲定律
十字链表出场景，并不需要遍历，从自身往两头链表查找就好，这个不需要遍历整个链表
2021-08-23 回复 喜欢

莫日
作者大大，请问十字链表法的Y轴链表有什么用？在维护观察者集合时只要一条链表不就可以实现吗？ 即哨兵节点在x轴移动时直接就可以比较跨过节点的y是否超出范围，超出范围的就不进入视野。看有的文章提到x，y轴各前后遍历一次取交集，这不是脱了裤子放屁么。。。？
2021-05-16 回复 喜欢

co lin 作者
假如别的结点只移动Y坐轴，这时如果他离开你的视野，你没有y轴的哨兵，你就没法感知他离开了。
2021-05-16 回复 5

我晕

2021-04-19 回复 喜欢

belonger
九宫格那种 可以做一个视野优先级和上限 每个玩家自己维护自己的事业列表
2021-02-08 回复 喜欢

欢迎参与讨论
2021-01-03 回复 喜欢

陆海龙
就是和问道手游那种分线，其实就是不同场景，每个场景都会有人数上线的，这样保证每个场景的负载不会那么高
2021-08-21 回复 喜欢

co lin 作者
分线将一个场景复制为多个，类似于副本；分层只有一个场景对象，但角色进入这个场景后由不同的层对象管理，这样即使在同一个场景，也可能看不到对方，达到减少AOI消息的目的。
2021-01-04 回复 喜欢

黑心
哨兵会增加2倍的节点，感觉不一定好。这里遍历还有个优化方式是 快慢指针
2020-10-21 回复 喜欢

刘煜煜
要是遇到坐标一样的情况下怎么办
2021-11-02 回复 喜欢

znb sandy
能具体介绍一下么
2021-06-02 回复 喜欢

Luoluo
哨兵和地标节点妙啊
2020-08-28 回复 喜欢

点击查看全部评论 ›

知乎 首发于 游戏服务器编程

游戏服务器编程
专注于游戏服务器编程的技术分享

推荐阅读

ICML 2020 | 五篇精选论文，洞悉微软亚洲研究院机器学习...

微软亚洲研究院

【DL碎片2】神经网络中的优化算法

蝈蝈

【每周CV论文推荐】 初学者必须精读的5篇深度学习优化相...

龙鹏-笔名... 发表于有三AI学...

FFT的4096点verilog实现真到FPGA上板实测

三月份开始做FFT，由于在上周大概做个一两天，这样一直月初放假。 7月初的进度是基能测试完成，但是驱动的C代编写完成，以及测试的主机端没有编写完成，尚未上板...

在路上