┏━┳━┳━┳━┳━┳━┳━┓

┃○┃ ┃ ┃ ┃ ┃ ┃●┃

┣━╋━╋━╋━╋━╋━╋━┫

┃ ┃●┃●┃●┃●┃●┃ ┃

┣━╋━╋━╋━╋━╋━╋━┫

┃ ┃●┃●┃●┃●┃●┃ ┃

┣━╋━╋━╋━╋━╋━╋━┫

┃ ┃●┃●┃○┃●┃●┃ ┃

┣━╋━╋━╋━╋━╋━╋━┫

┃ ┃●┃●┃●┃●┃●┃ ┃

┣━╋━╋━╋━╋━╋━╋━┫

┃ ┃●┃●┃●┃●┃●┃ ┃

┣━╋━╋━╋━╋━╋━╋━┫

┃●┃ ┃ ┃ ┃ ┃ ┃○┃

┗━┻━┻━┻━┻━┻━┻━┛

玩家是黑子

①从open表中拿出后并不代表要移出open表（这可能与之前的深度优先搜索不同，因为我们这里是深度优先的α-β剪枝算法）

②在生成一个新的结点后把它放入open表的同时还要把它放入closed表（这可能违反通用的搜索算法，但不违背closed表的存在意义，即避免生成相同的结点）

③一次的扩展只会生成一个直接后继结点，并非生成全部的后继结点，这与之前的深度优先搜索不同

④只要一个结点移出了closed表，就释放掉这个结点

⑤只要一个结点没有可生成的结点了即不能扩展了或者这个结点由于剪枝不需要扩展了，那它就应该被移出open表，因为open表是用来盛放需要扩展的结点的

1.算法开始时open表只有一个初始状态即该AI方走步时的棋局状态，closed表为空

2.在有红字的框框里要注意：在释放结点时先做判断，如果（即将被释放的结点的父节点是初始状态的话）且（这个子节点的倒推估计值等于父节点的倒推估计值）的话，①先把这个子节点存赋值给bestchange这个变量里来保存AI最好的走步，②然后再释放这个结点。

如果这个条件不满足的话直接释放就好了

3.如果整棵搜索树保存了的话closed表其实是一个不必要的存在，因为在这里我们closed表的目的就是为了避免一个结点再次生成已有的子节点，那这样的话直接与子节点比较就好了，没必要再进去closed表里边进行比较了。那么相应的下边的流程图中涉及到从closed表中移除的地方就可以只释放结点就可以了

将这个结点移出open表，（因为不需要对他进行扩展了，所以就移出open表）删除它所有的直接后继（这时它的直接后继结点必定没有孩子，因此只需要释放当前结点的直接后继结点就行），同时要注意也删除它们在closed表中的记录（无closed表时就不用了）（也不必更新父节点的α或β值，因为就算有更新这个操作的话，父节点的α或β值也不会被更新，因为这个结点已经废掉了，父节点不管如何都不会选这个结点了）（不必删除在open表中的记录，因为此时肯定在open表中没有记录）（因为它的后继结点已经没有任何意义了，后继结点的任务仅仅是更新先辈结点的α、β或倒推估计值）

将当前结点的α或β的值赋值给倒推估计值，并更新父节点的α或β值，因为扩展完了的结点的倒推估计值一定等于此时最新的α或β值（部分直接后继结点没有被扩展时不能被称为扩展完了），然后把它移出open表，同时把它的直接后继结点移出closed表中（可能不需要），并释放掉这些直接后继结点（注意不要把它移出closed表），因为我们把这个结点移出了open表中，这就意味着我们不会再扩展它了，那就没必要保存它的已生成过的直接后继结点了，因为不需要避免生成相同的结点了，因为我们不扩展了

若（α！=INF&&β！=INF）且（α>=β）即不需要扩展这个结点且还能扩展

从open表（一个栈）中拿出栈顶结点而非移出（open表为空时算法终止，或说open表不为空时执行）

若不能扩展

若不是

计算它的静态估计值，然后更新父节点的α或β值，从open表中移除

若是

判断该结点是否为最底层结点

对照closed表（或已有儿子们）扩展一次不同的结点并把该结点放入Open表中，并把该结点放到closed表中（这步可能不要）（以免下次扩展相同的）

若（α<β或 当前结点或父节点没有α或β值）且这个结点还能扩展

判断当前结点的α或β值与父节点的α或β值的关系