主要求解方法：

Boardgame\_simple::OnlyOneLeft():

dfs遍历所有状态空间，直至找到仅剩一个棋子的状态，或搜索完所有状态。输出仅剩一个棋子的解/最少剩余的解；找到一个棋子解就会退出，不存在一个棋子解情况时需要搜索全部状态空间。

Boardgame\_central::LeasStepsSolve():

视连跳为一步，求解初始状态到最终状态的最短步数解。

BFS队列元素使用ConState（step,route,state）,对象中存储路径字符串以及当前步数,上一次行动时的棋子序号可通过解码路径后三位获得，该解法比较八种等效状态去重，但在去重时规则有所不同：先求出所有存在的等效状态的步数，取其中最小步数（map<long long, int>存已访问过的状态和步数），若当前状态小于等于该最小步数则进队。事实上，到达同一状态的步数恐怕均会相等，这个剪枝几乎没有起到作用。

Boardgame\_central::LeastStepsSolve\_less\_memory():

在LeastStepsSolve()的基础上,考虑减少map<long long,int>带来的内存累积，在扩展每层去重只用得到本层的元素，故使用两个队列，交相运算，完成逐层遍历与map清空。

Boardgame\_central::LeastStepsSolve\_more\_less\_memory():

在less\_memory的基础上，考虑到队列出队后仅析构对象，队列中的内存并没有被释放，而是保留等待后续进队使用，又有该问题的状态空间树可能是上下窄，中间宽，故到运算后期，两个队列的空间都会是最宽那一层的状态节点数量，有浪费；另外，当一层的状态空间节点较多时，却有两个队列的空间等于这些节点的大小，造成了一倍的浪费，故考虑只使用一个队列完成逐层遍历与map清空。

Boardgame\_central::OnlyOneLeft()

BFS等效状态去重，求解最少剩余