**g象棋将军算法**

1. 主动将军（将棋子移动到制定坐标形成将军）：

还分为直接将军和间接将军：

直接将军：比如，将位于移动后的马，车，炮，兵的攻击范围内

这种情况只需要判断移动后的棋子是否正在将军

间接将军：移动了一枚卡在另一枚棋子攻击范围的棋子，使另一枚棋子造成了将军

这种情况还需要判断处于潜在将军状态的棋子在这之后是否变成了将军

棋子状态：直接将军态，间接将军态，无将军态

**名词解释：**

**有效移动坐标集合**：棋子根据自身移动规则，能移动到、或者是击败敌方棋子并取代其位置，且不会被己方或对方棋子挡住路径的坐标集合

**【总】有效将军坐标集合**：处于该集合中的将（帅）会被视为将军

棋子每移动一步，自动刷新有效移动坐标和有效将军坐标。如果敌方将的坐标处于此时的有效攻击目标内，那会形成将军。下一步地方必须做出反应，此时有三种方法解除将军，1. 消灭正在将军的那个棋子。 2. 挡住那个棋子的将军路径。3. 移动将的位置来解除将军。

*需要为棋子类增加一个成员变量，记录这枚棋子此时形成的“将军坐标集合”。*

*需要为玩家类增加一个成员变量，记录此时正在将军自己的敌方棋子的集合。*

*需要为棋子类增加一个成员变量，记录将军时的路径（坐标集合）*

如何判断是否被将死？**1.** **判断己方能否有棋子消灭敌方将军的棋子**：逐个检查自己所有进攻棋子的“有效移动坐标集合”，判断每一个敌方将军棋子是否处于这个集合内。**2. 判断能否有棋子挡住敌方将军棋子的路径。**

**1,2 两种情况还需要判断移动棋子后，是否会造成被动将军，如果会，那么此种解法无效**

**for (every pieces)**

**for (every NextMoves)**

**if NextMoves[i] == 将军棋子的坐标**

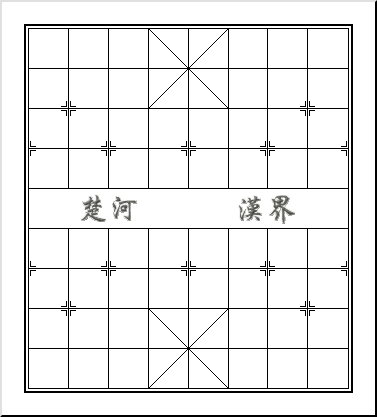
**if 被动将军 continue**

**3. 判断能否移动将的位置。**首先求出将的下一步有效移动坐标，然后逐个判断，将在新的位置是否还会被将军，即判断新的位置是否处于所有敌方进攻棋子的攻击范围之内，这就是为什么要用到“有效将军坐标”这一概念，当然这个“有效将军坐标”是针对将军而言，因为这里有个特殊情况就是双方的将不能出现在同一空的列，所以在计算“有效将军坐标”的时候，将的地位要这么考虑：首先逐个检查己方将的前方是否存在其他棋子，如果有，那就不必考虑己方将的“有效将军坐标”，如果没有，那就可以把将看做车，即，将的前方所有坐标都是将的“有效将军坐标”。

**各个棋子的“有效移动坐标(effective next moves)”算法：**

由于将只会出现在九宫格中，所以只需要考虑九宫格内的情况

0 1 2 3 4 5 6 7 8



这里假设棋子的坐标为(x, y)

**各个棋子的“有效将军坐标(effective checkmate coordinates)”算法：**

车：

1. 3 <= x <= 5，且y > 2：从y-1开始，依次-1， 判断坐标(x, y-1) 是否存在单位，直到y=0，若碰到将，直接

2. 当车的y坐标在0-2之间，若x坐标小于等于

马：当1 <= x <=7, 且0 <= y <= 4，（且除了(4, 1)）开始考虑马的攻击范围。计算马的“有效移动坐标”中落在九宫格内的部分 3<=x<=5, 0 <=y<=2

炮

兵：当 2<=x<=6, 且0<=y<=3, （且除了(2, 3), (6, 3)）开始考虑兵的攻击范围。

将：逐个计算坐标(x, y-1) 上是否存在棋子，

2. 被动将军（己方棋子若移动到某个地方会造成己方被将军）

造成被动将军有两种情况，一是移走棋子会导致原来的位置变空，从而暴露在敌方的火力之下，例如移走了“绊马腿”的棋子从而导致被对面的马将军。二是棋子移动到了新的位置后造成了被将军，例如，把棋子移到了别人的炮口。

由于“被动将军”是不被游戏规则允许的，所以要避免。

这里我有两种设想：

**1. 当试图移动一枚己方棋子时，临时计算移动棋子后，对方的“有效将军坐标集合”，如果此时己方的将位于这个坐标集内，则无法移动，从而避免了己方的两种情形的“被动将军”。**

2. 直接计算出己方的哪些棋子是第一种情况中不能被移走的，这种情况可能要引入一个新的collection，即“卡位棋子”，算法为：找出所有与敌方将有关联的进攻棋子，举个例子，就像与敌方将处于同一水平线或垂直线的车和炮，在将旁边或上方的兵，与将距离一步一斜格的马，然后找到卡在两棋子中间起到卡位作用的地方其他棋子，把它们标记为“卡位棋子”，这些棋子在那一方行动的过程中是不允许被移动的，所以避免了第一种情况的“被动将军”。

但可能第2种算法实现起来不如第1种方便，而且它无法应付第二种被动将军的情况，而第1种算法可以应付两种情况，所以我就使用第1种算法了