这是对目前已有的代码以及正在进行的重构的设计说明.

围棋的黑子和白子可以形成棋块，同时空白的棋子形成气块。大多数情况下这个气块是一个公共的气块。用处不是很大，保留气块的主要目的是用于记录眼位的情况。空白块属于同色棋子则称为眼。提子即可有眼，但不一定是真眼。我们说的眼位是否指真眼，还是泛指（不一定是真眼），要看上下文来理解。

黑棋块，白棋块和气块任意两者之间都可能相邻。

黑棋块周围可以有一到多个白棋块，一到多个气块；

白棋块周围可以有一到多个黑棋块，一到多个气块；

气块周围可以有一到多个白棋块，一到多个黑棋块。

这些概念是如下对应到Java类的。

黑白棋块：Block

气块（眼位或公气）：BlankBlock

棋盘：GoBoard

棋点： BoardPoint

点：Point

本来棋点和块是双向关系。历史的原因，Block中存的是Point的集合。

GoBoard 代表棋盘，最主要的功能是能够下一步棋，并处理提子，气的增减，棋块的合并，气块的分裂等等细节。 另一个功能是能够悔棋，将局面恢复到落子前的状态。

本来只要记录每步落子的位置即可悔棋，但是为了提高效率（比如某一步发生棋块的合并，悔棋时可以利用旧的棋块，不用另外生成新块），也为了简化算法。分裂前的气块和合并前的棋块都是记录在案的，供悔棋时用。

悔棋功能除了用于打谱之外，其实还有一个重要的用处就是在围棋的人工智能搜索时可以用来减少存储局面状态的内存。我们不需要保存过程中的每一个状态，我们只要有一个附近的状态再加上少数几步的棋谱即可表达该状态。

“*是否需要回退的功能.考虑了很久.*

*考虑过的选项*

*1. 不实现回退.功能.必要的时候重新前进.(这就需要保留旧局面的数据结构)*

*2. 实现回退(悔棋)功能,这样搜索计算时就不再需要保留旧的状态,而是可以方便的回退.对于增强计算能了有好处.*

“

现在的集合操作都是依赖equals和hashCode方法的。原先考虑过自定义这些方法。现在决定还是用默认的reference equality为好。

这部分代码比较复杂，容易滋生缺陷。目前的想法是通过清新的设计（包括算法的设计），充分的测试来保证质量。原先的测试主要是基于土人的五千局棋谱库。现在主要用网上下载的吴清源263局。其实有一两局就可以发现 大多数问题了。

因为以前实现过一个基于数组数据结构的GoBoard，可以和他对比处理结果是否一致。

另外就是数据结构的内部验证，看是否一致，是否复合理论上的不变量。

落子后的处理算法。

落子点的周围最多有四个点，最少有两个。

他们可能是敌方子，可能是己方子，也可能是空白点。分别讨论

空白点：

1. 可能分裂成多个气块，因为少了一个气点。
2. 可能气块没有分裂（这种可能性大很多）
3. 敌方子或者己方子：可能不再和原先周围的气块相邻。如果气块没有分裂，则简单删除相邻关系。如果气块分裂，原气块删除。则计算过程当中自动汇集相邻的数据。不论结果相邻与否都是自然正确的。

敌方子：

减少一气

1. 剩余一气或以上（但是不提子）
2. 提子

周围的棋块全部增加气。

己方子：

减少一气

合并棋块。

悔棋时的处理难点在于，能否直接用原先没有分裂之前的空白块。好像不行，因为原先的空白块已经不再和当时 的棋块相邻。

悔棋算法：

1. 同色块先分裂。（有旧的被合并的块记录，且相邻记录也是正确的。但是所指的空白块可能不正确。）
2. 提子块的恢复。
3. 气块的合并。（利用旧的完整气块，但是相邻关系需要重新维护。）

要点在于能使落子和悔棋的处理都比较简单（相互之间要配合）。

每步落子处理涉及到的棋块。

1. 落子点。（颜色改变，所属棋块改变）
2. 落子点所在的原气块。（颜色不变，但相邻关系有调整）

(如果气块分裂的话) 周围的子块

1. 落子点周围的己方块 （颜色不变，但相邻关系有调整）

（如果合并的话） 周围的敌块，气块

1. 落子点周围的敌方块

（若被提吃）周围的子块（与落子点同色）

改变颜色的是少数棋子。主要是棋块相邻关系的调整。

先分类可能的情况，再统计各种情况的频率。最后设定相关的测试用例。

1. 气块是否分裂（需至少两个空白点）
2. 棋块是否合并（需至少两个同色点）
3. 是否提子 （需至少一个异色点）

三者不会同时发生

先合并棋块, (则旧棋块的信息基本正确,指向的相邻块也正确,但是相邻块的反向连接可能不对,因为他们指向新建的棋块了.)

注意单个棋块的合并仍旧沿用旧棋块. (悔棋时.其相邻关系都要更新)

再分裂气块(则原棋块的气块信息也不受干扰.分裂出的气块会指向新生成的棋块.)

最后再提子.

悔棋是

气块先合并.

先删除刚分裂出的气块. 然后将这些点恢复为原先的气块.

原气块有正确的相邻关系(单向的,需要更新另外的反向).

不会更新到新分裂的棋块中,因为原气块指向旧的棋块)

新建的棋块先从邻块/气块中删除. 分裂出来. (不用更新相邻关系)

恢复旧的棋块(跟新单向的相邻关系)

参考文献：《打劫使您的棋力变强》第22页。

ISBN：7503906006

局面焦点：黑先则两眼活棋。外围白棋厚，且能确保连接。

不粘气点的原则需要修正，比如形成气点的棋块处于被打吃状态。

轮白走

轮黑走轮黑走

轮白走

轮白走

轮白走

轮白走

白a提：

轮黑走

黑提劫：

轮白走：

轮白走

1.最好不要考虑弃权，把这种情形合理剔除有利于简化问题。

2.弃权的结果可能对于区别二手劫和普通劫争是有必要的。

考虑二手劫的例子。

参考：克敌制胜的定式变着第177页。

ISBN：750431763

8

76

. . . . . .

5 . .    . .

4 . È  .

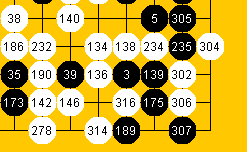
3 . .  .  .

2 .   .  . .

1 . .  . .

a b c d e f g h j k l m n

debugging tracking:



189 的周围气块有一个(原316所在位置的棋块)

是空的(size=0) 导致问题.

修正: 将落子后棋块的消失考虑在内.

这个问题主要出现在局面不断更新的过程中, 从初始局面读就没有这个问题.

为了性能考虑, 不能每次都作为初始局面读,而是保持上个局面的大部分数据,仅仅更新落子点附近的数据.

因为围棋的算法是个不断探索改动的过程,经常会修改代码,引入bug.建立一个高效的调试机制很重要.

1. MVC pattern. saparate UI class and core implementation.

2. 2011-8-22

It is very difficult for me to understand the old code, the current idea is to re-implement

it based on current requirement.

2012/03/23

Tomorrow 2:30 外教俱乐部 带画笔和纸

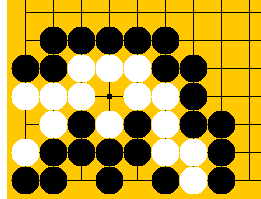
围棋棋盘上的状态一般来说和轮到谁走关系很大。比如说死活题吧，先后手不同往往结果不同，除非是死活状态已经确定的局面，如两眼活棋，大眼活棋。 所以我们判断形势的时候，应该指定先后手。做死活题目的时候也要指定先后手。

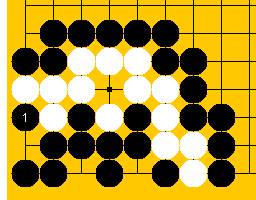
20120401

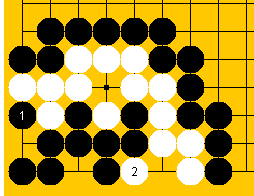
处理官子的时候,小棋盘有其方便性.因此决定支持任意棋盘大小.

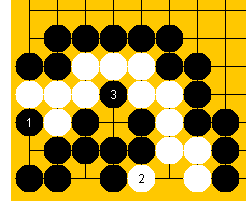
20120519:

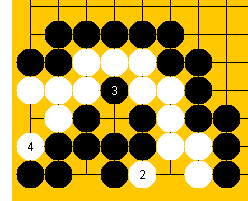
有眼三劫的分析

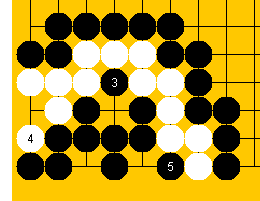
**001

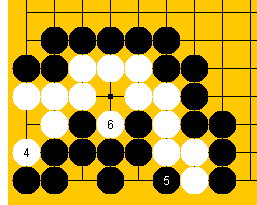
🡪101

🡪100

🡪110

🡪010

🡪011

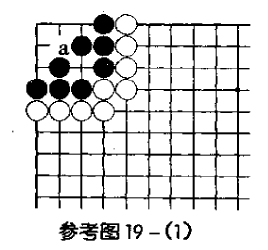
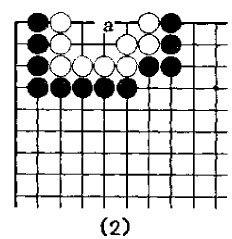
🡪

(局面循环了，所以白方011状态需要找了劫材之后才能白6提)

之后新的一轮循环，该黑方找劫材，黑方即使劫材不利。

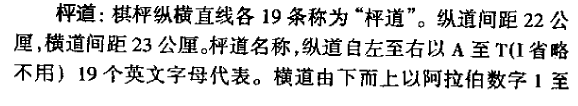
终局时隐藏的手段。白方A位断入，价值后手4目。考虑单官的价值，实际上可以差6目。

（轮黑下，黑补一手的结果和黑弃权白断入的结果比差6目，即3子）或者说 （轮白下，白弃权黑补一手的结果和白断入的结果比差6目，即3子）

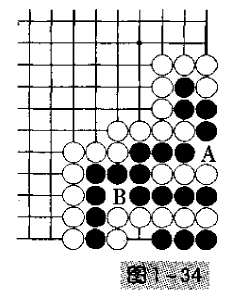
 

（举例而已，初学者也知道刀把五可以被点死。

如何计算这种隐藏手段。 （a位是卡眼的手段。）



特殊的活棋结果：（可以看成双活的扩展形式。）



六块棋卷入。其中外围白棋块看成普通的两眼活棋。

黑有三块：从上到下分别有一气，二气，一气。

白有两块；从上到下分别有一气，二气

一般的双活棋，涉及的每个棋块有两气。如果要吃对方必须送吃，对方反而成了大眼。

这里虽然有三块棋是一气，但是对方提了反而不利，

这里可以死算，候选点很少，最多三个。

如何识别这种活棋。算法一般是从一个棋块开始的，没有全局观念。

黑块1：形式上并不是是死棋，因为相邻百块也只有一气。（仅有一气可以被提掉。）

黑块2：形式上并不是是死棋，因为相邻百块也只有一气。（仅有一气可以被提掉。）

黑块1：形式上并不是是死棋，因为相邻百块也只有一气。（仅有一气可以被提掉。）

他们都没有两眼活棋。

从两气的块开始看，有双活可能。唯一剩下的变化是黑提子长气。但是对杀负。

考虑提子长气，皆不行。

也可以直接考虑提子的变化。（法现反而不利。）

关于大眼死活的知识表达。

全局的任何局部大眼，转化为左上角局部表达（移位的表达）涉及的点除了眼位之外，也包括眼位外面一层的包围之子，如此完整眼位和有缺陷的眼位可以区别。而且边角的眼位也可以区别（光移位是不够的！）。

检索结果的key设计如下。

**所成大眼包含的点数N**

**成眼的块数**

大眼的形状

所涉及的边数。（由最终**的矩阵大小**可知）

中间的板六为例：

N=6

XXX

XXX

3\*4 （不同于4\*3）3\*3可知

OOOOO

O XXX O

O XXX O

OOOOO

四个连接点决定了眼位完整与否。

还要能表达成眼 块的一些信息。比如气数，

还有包围的块，没有包含在内。假设他们很强，这里只是研究大眼是否能够得到两眼或者更多。

**矩阵大小 有足够的信息。4\*5在中央，4\*4在边上，3\*5在边上，3\*4在角上。**

**还有四个对应的，共八个。**

变化的结果如何保存。

SGF should be able to handle it, with minimum board size!

成眼块完整而且和外气无关式，存储成眼块信息已经足够。

若与外气相关，需要将成眼块周围的敌子也包括，如盘角曲四。

若眼位不完整（气紧），则与外气相关。需要存储成眼块周围的敌子， 且需要为他们预留气点，因此需要更大的矩阵来存储