|  |
| --- |
| MATLAB与科学计算课程报告 |
| 五子棋AI |
| 蒙特卡罗模拟树算法的实现 |
|  |
| **13328157 骆锎源 13328288 张一帆** |

|  |
| --- |
|  |

目录

[1. 选题阐述 2](#_Toc454554542)

[1.1 选题背景 2](#_Toc454554543)

[1.2 目标陈述 2](#_Toc454554544)

[2. 编程思路 2](#_Toc454554545)

[2.1 图形界面 4](#_Toc454554546)

[2.2 规则 4](#_Toc454554547)

[2.2.1 is\_legal判断 4](#_Toc454554548)

[2.2.2 EndGame判断 4](#_Toc454554549)

[2.2.3 Block 6](#_Toc454554550)

[2.2.4 Leap 7](#_Toc454554551)

[2.3模拟器结构（AIthink） 8](#_Toc454554552)

[2.3.1 MCTS模拟 9](#_Toc454554553)

[2.3.2 遍历 13](#_Toc454554554)

[3.完成作品 14](#_Toc454554555)

[3.1 人人对战 14](#_Toc454554556)

[3.2 纯粹模拟（胜率返回） 14](#_Toc454554557)

[3.3 纯粹模拟（访问次数返回） 14](#_Toc454554558)

[3.4 模拟+规则（胜率返回） 15](#_Toc454554559)

[3.5 模拟+规则（访问次数返回） 15](#_Toc454554560)

[3.6 范围遍历 15](#_Toc454554561)

[3.7 规则&模拟混合 16](#_Toc454554562)

[3.8 机机对战（设想） 16](#_Toc454554563)

[4.评价 16](#_Toc454554564)

[4.1难点综述 16](#_Toc454554565)

[4.2 未实现的优化 19](#_Toc454554566)

[4.3 感想 20](#_Toc454554567)

[5. 附录 22](#_Toc454554568)

[5.1 程序使用方式 22](#_Toc454554569)

[5.2 组内分工 22](#_Toc454554570)

[5.3 参考文献 22](#_Toc454554571)

# 1. 选题阐述

## 1.1 选题背景

选题之初，由于对风险管理中应用的蒙特卡洛模拟法有兴趣而查了一些相关的资料。在查资料的过程中了解了一族以随机模拟为核心的蒙特卡洛算法，并且发现著名的AlphaGo应用了其中的蒙特卡罗模拟树。在对蒙特卡罗模拟树进行进一步深入研究的过程中发现它可用于一大类博弈模型（combinatorial game），并且算法本身不要求对博弈策略有所研究即可通过随机模拟得到近似最优解。在这些知识的基础之上决定应用蒙特卡罗模拟树来做一些程序，选定五子棋是因为游戏规则相对简洁，而且在可应用的范围内。确定选题之后继续进行对MCTS的研究，通过阅读《A Survey of Monte Carlo Tree Search Methods》（Browne, Cameron et al, 2012）对MCTS的流程进行了全面的了解，并形成了实现五子棋AI的基本想法。

## 1.2 目标陈述

1）制作一个五子棋AI，使其能够根据棋局的进行给出（近似）最优的步数，且步数的给出由随机模拟决定，不依赖于基本游戏规则以外的策略型规则；必要时可以给出策略型规则作为辅助。

2）制作允许人机交互的五子棋对弈程序，其中机器部分应用五子棋AI。

# 2. 编程思路

主程序按照下棋的流程进行，如下图。

具体而言，棋盘采取画图的方式实现可视化，在程序内部采取15\*15的矩阵形式，并分为两层：记录棋局的棋盘矩阵（Chessboard）和记录落子情况的计数矩阵（CountZeros）。棋盘矩阵初始为零矩阵，随着棋局的进行，机器落子位置元素变为1，玩家落子位置元素变为2；计数矩阵初始为全1矩阵，随着棋局的进行，有子的地方变为0，无子的地方保持不变。

在经过针对模拟器的优化之后对每一次模拟的范围进行了限定，因此在机器和玩家落子之后增加了更新范围边界的步骤。

初始化游戏界面

机器随机生成第一步

是否结束游戏

机器生成一步

是否结束游戏

输入合法坐标值作为人下的一步

游戏结束

棋盘未满

否

否

是

是

## 2.1 图形界面

图形界面使用plot函数为主，首先在15\*15的范围内，坐标选择为（1-15）而不从0开始，因为在边界的棋子也可以显示出来。每次一方落子输入的是相对应的坐标，用fill函数填充不同的颜色加以判断。

图形界面的设计参考了《智能五子棋算法的设计实现》（王长飞等，2009）。

## 2.2 规则

首先对与人或者机器下的新的一步，需要先判断是否合法，即这个位置是否在棋盘范围内并且是无字的位置。某一方下完一步以后，需要判断这一步是否使游戏结束。某一方下完一步，另一方需要判断是否使用规则，即三个或者四个连在一起并且需要处理的情况。

### 2.2.1 is\_legal判断

每次无论人或者机器落子时都需要判断该位置上是否已经有子，如果是人落子位置已经有子，需要显示错误信息并重新落子。在机器模拟中同样需要判断，如果有子重新生成随即位置。islegal判断（文件中名称为is\_legal2）就是判断该位置是否有子的函数。

判断使用的是第二个矩阵CountZeros进行判断，CountZeros初始化为全是1的15\*15矩阵，每次落子会将对应位置的数值改为0。通过判断需要判断位置上CountZeros矩阵上的值可以判断出是否有子。

### 2.2.2 EndGame判断

EndGame是用于判断游戏是否结束的函数文件，在每一次在棋盘矩阵（Chessboard）和计数矩阵（CountZeros）更改后使用。其输入变量是更改后的上述矩阵和最新一次落子的坐标，返回变量为游戏是否结束（逻辑值）。

EndGame的基本思路为以五个判断判断游戏是否结束，即：

1）棋盘是否下满，通过计算计数矩阵所有元素的和实现；

2）左右方向上是否连成五子；

3）上下方向上是否连成五子；

4）左上右下方向上是否连成五子；

5）左下右上方向上是否连成五子；

其中1）至4）通过计数棋盘矩阵特定方向上相同数字个数实现。5个判断之间为逻辑或关系，即只要有一条满足游戏即结束，返回变量为真，否则为假。

EndGame流程图如下：

输入棋盘、计数、坐标

左右连5个？

游戏结束

上下连5个？

游戏结束

左上右下连5个？

游戏结束

左上右下连5个？

游戏结束

棋盘下满？

游戏结束

Y

N

Y

N

Y

N

Y

N

N

游戏继续

Y

### 2.2.3 Block

Block是策略型的规则，主要功能是在对方连成3个时进行截堵，以补充模拟的不足。现已被Leap规则替代。

Block的基本思路是，首先以和EndGame相同的思路判断对方是否连成3个，同时返回一个数组Situation（4元素），记录对方在哪个方向上连成三个。然后，以Defend函数生成一个坐标，截堵对方的三连。在Defend内部，首先根据Situation判断对方三连方向。针对四个方向分别有一个函数（is\_adjacent1至is\_adjacent4）返回可堵的坐标，并判断该坐标是否合法，根据Situation非零的位置调用不同的函数。在假定截堵任意方向的策略价值相等的情况下，按顺序进行判断，最先通过判断的坐标即为返回坐标。

Block的代码在文件夹Block中。

### 2.2.4 Leap

Leap是规则中的每次出现【空白白白空】或者【空白白空白空】或者【白白白白】的情况下需要在某一侧落子堵住或者避免4子连成的规则。

Leap中使用的是Leap1 Leap2 Leap3 Leap4 分别判断横竖斜四个方向上是否需要使用规则，将返回值返回到Leap中的一个矩阵，并记录每个方向判断后的需要使用规则的位置r0,c0，如果不需要规则则r0,c0为0,0。最后计算Leap中记录返回值的矩阵合是否为0，为0意味着不需要使用规则，若不为0则找出哪个方向不为0并使用该方向返回的r0,c0。

以Leap1为例解释某一方向上判断过程，假设输入位置为(r,c)，通过两层循环，分别判断(r,c-3)- (r,c)、(r,c-2)- (r,c+1)、(r,c-1)- (r,c+2)、(r,c)- (r,c+3)四个循环中每四个子中是否出现三个对方子与一个空位置，假如是则返回判断值为真与空白子的位置。四个循环可以覆盖上述需要使用规则的情况，甚至会覆盖部分不需要规则的情况例如 空白白白黑 白空白白黑 等，即便不使用规则也不会出错，故增加判断在循环的4个子前面与后面（第一次循环为例就是 (r,c-4)与(r,c+1)的位置）是否有一个我方的子，如果有则将判断返回值改为假。

下图左边为Leap的判断，右图为Leap1作为例子的图示。

输入位置与当前棋盘

Leap判断

输出判断值与位置

输入位置与当前棋盘

ii=1：4

c+ii-4是否在

1到12内

是否三个对方棋子与一个空白

计算c+ii-4到c+ii-1中对方棋子个数与空白个数

c+ii-5与c+ii是否有一个为我方棋子

返回值为假

返回值为真

ii=ii+1

否

否

是

否

否

是

## 2.3模拟器结构（AIthink）

模拟器（AIthink）是一组函数，用以让电脑综合考察当前的棋局并且做出回应。输入变量是当前棋盘矩阵、计数矩阵以及预设的模拟范围，返回变量是电脑方下一步的坐标（在棋盘矩阵上的行标和列标）。模拟的方法属于蒙特卡罗模拟树（MCTS），为了解决耗时和模拟结果优度之间的矛盾的，我们结合五子棋的特点限定了模拟的范围。模拟范围根据棋盘上有子的位置决定，并随每一次落子而扩大，随着棋局的进行可能扩充到全棋盘。同时，模拟次数与模拟范围大小成正比，初步设定的比例为100:1（1为模拟的范围大小）。

### 2.3.1 MCTS模拟

蒙特卡罗模拟树方法主要是以随机取样（而非遍历）的方式来取得下一策略的最优近似解的一种算法，包括四个循环的步骤：选择（Selection）、扩展（Expansion）、模拟（Simulation）和反馈（Backpropagation）。在我们的程序AIthink中，选择和扩展两步使用Expand函数实现、模拟通过DefaultPolicy函数实现、反馈通过Backpropagation函数实现、最终的节点选择根据标准的不同，使用Choice函数（以胜率为标准）或SuperMax函数（以被访问次数为标准）实现。各个函数的具体运行方式将在稍后进行简单介绍

以下是模拟器的简单结构（框内为程序中的函数名）：

Expand

DefaultPolicy

Backpropagation

Choice/SuperMax

达到循环次数上限

N

Y

在进行模拟之前，首先要对几个需要用到的矩阵进行默认设置。需要用到的矩阵分别是：记录每一个点的累计被访问次数的矩阵CumVisitT，记录每一个点的净胜次的矩阵CumPayoff，以及记录每一个点评分的矩阵Rank。其中，被记录的点为被Expand函数选中的点，每一个矩阵都是和棋盘同样大小的15\*15矩阵，矩阵上的元素与棋盘矩阵上的一一对应；净胜次的计算方式为直接累加结果，其中胜记为1，负记为-1，平局不计。累计访问次数和累计净胜次矩阵的所有元素初始值均为零。评分矩阵根据当前计数（棋盘）矩阵设置，在计数矩阵元素为0的点，评分矩阵相应位置的值为负无穷（-inf），在计数矩阵元素为1的点，评分矩阵相应位置的值为正无穷（inf）。也即，有子的点评分为负无穷，没有子的点评分为正无穷。这些设置存放在函数DefaultForS中，在每次启动模拟器时调用。

接下来介绍Expand函数。Expand函数的主要功能是根据当前棋局和预设的模拟范围，给出下一步的一个可能的点。给出该点的方式为选取评分（Rank）矩阵在模拟范围内的最大值所在的坐标。根据评分矩阵的设置方式可知，棋盘上有子的点不会被选取，而没有子的点在模拟过程中至少会被选取一次。Expand的函数运作过程如下图：

用SuperMax排序Rank矩阵获取给定范围内的最大值，返回最大值的行标和列标

改变棋盘矩阵和计数矩阵相应位置的值

返回行标、列标和改变后的棋盘矩阵和计数矩阵

完成扩展之后，程序进入DefaultPolicy函数。DefaultPolicy函数的主要功能是根据Expand函数返回的坐标和棋局通过随机模拟的方式将棋局进行到终局，返回获胜方（变量Win）。机器方获胜时，Win为1；对手方获胜时，Win为0，平局Win为-1（这个设置来由主要是Win变量原本是逻辑变量，有1、0两个值，电脑赢时为真，输时为假；后来考虑到平局的情况增加了-1的值，为了减小修改范围，故没有改变原来1、0代表的情况）。在DefaultPolicy的返回变量中有一个i值，代表内部循环次数，是在调试程序中判断模拟过程是否正常进行而设置的参数，如果i值异常大或异常小，则程序可能出现问题。在正式的流程中，i并无特殊作用。

双方下棋的顺序和流程与主程序MainProject类似，决定每一步落点位置的函数为Simulation。在Simulation函数内部，用Matlab自带的randi函数经过线性变换生成给定范围内的二维随机整数组，并调用is\_legal2函数判断改组坐标是否合法，不合法则继续进行下一次生成，循环过程用while循环实现。

以下是DefaultPolicy函数的结构：

Expand的步数是否结束游戏？

用Simulation模拟对手方步数，判断是否结束游戏，更改临时棋盘，更改模拟范围

对手的步数是否结束游戏？

用Simulation模拟电脑方步数，判断是否结束游戏，更改临时棋盘，更改模拟范围

电脑的步数是否结束游戏？

根据结果改变Win变量的值

Y

Y

Y

N

N

N

以下是Simulation函数内部结构：

生成二维随机数组

是否合法？

返回

N

Y

获取棋局结果（Win变量值）后，程序进入Backpropagation函数。该函数的功能主要是根据模拟的终局结果改变累计访问次数矩阵、累计净胜次矩阵和评分矩阵上与Expand函数返回的行标和列标相对应位置的值，并返回所有改变后的矩阵，其中Rank矩阵在进行下一次模拟再次调用Expand函数时作为输入变量进入程序中。

具体而言，对于每一次终局，被扩展到的落子点对应位置的累计访问次数加1。在累计净胜次矩阵上，如果电脑赢，Expand落子点对应位置的净胜次加一；对手方赢，Expand落子点对应位置的净胜次减一；平局不作处理。在Rank矩阵的相应位置，以UCT（Upper Confidence Bounds for Trees）算法的公式进行赋值。值得注意的是，原公式为：

其中Xj为[0,1]上数值；Cp为常数，按经验取；n为母节点被访问数；nj为Expand给出的子节点被访问次数。显然，公式的第一部分使胜率高的节点评分高，第二部分使被访问次数少的节点评分高，但随着模拟的进行第一部分会越来越重要。使用时，我们令Xj等于（累计净胜次/累计被访问次数），n等于50（不取上一步被访问次数的原因在于表达式中n对值的影响不大，且要获取上一步被访问次数会令程序变得复杂）。

完成Backpropagation之后进行下一次模拟，重复直至达到模拟次数上限。

#### 2.3.1.1 胜率返回

在上述程序达到模拟次数上限之后，AIthink程序将选择当前模拟中表现最优的节点。选取标准有两个，一是根据节点的累计胜率，二是根据节点的被访问次数。其中，胜率返回依靠Choice函数实现。

在Choice函数中，首先初始化一个15\*15的Select零矩阵，然后对Select的每一个位置用for循环进行赋值。注意到我们在进入模拟器时设置过模拟范围，所以在全棋盘（15\*15的范围内），可能存在一次都没有被访问过的节点。因此，在累计访问次数为0的点，令Select相应位置值为负无穷（-inf）；否则令其值为（该点累计净胜次/该点累计被访问次数）。在这之后用SuperMax找出Select最大值的位置并返回行标和列标。

#### 2.3.1.2 访问次数返回

访问次数返回通过直接处理累计访问次数矩阵即可实现。用SuperMax函数找出累计访问次数矩阵中的最大值位置并返回行标和列标即可。

接下来简介SuperMax函数，该函数可以找到矩阵在一定范围内的最大值并返回其坐标。与Matlab自带的find函数不同，SuperMax只会返回一个坐标，即使矩阵中存在多个相等的最大值。在多个最大值的情况下，SuperMax返回的是最后一个被遍历到的最大值的坐标。之所以这样设置，是为了优化胜率返回的结果（Choice函数中引用了SuperMax）。在胜率返回中，最大值经常在范围的左上边缘和中间同时取到（原因不明），而中间位置要优于边缘位置，故此取靠后的最大值较好。SuperMax在多个函数中均被引用，是模拟器中重要的组件之一。

### 2.3.2 遍历

见3.6范围遍历。

# 3.完成作品

在完善程序的过程中我们做出了不同版本的作品。这些版本存放在各自不同的文件夹中，同一个文件夹中还有我们测试的结果图。

## 3.1 人人对战

人人对战是最开始的模型，通过一个move的变量在1与-1之间的变化判断是哪一方先下，只有画图，判断输入是否合法，判断是否结束游戏等最基本的游戏组成部分组成。只能实现每一方都是人手动输入坐标值的游戏模式。

## 3.2 纯粹模拟（胜率返回）

胜率返回的纯粹模拟的组件以主程序、玩家输入区块、电脑模拟区块组成。其中电脑模拟区块中的选取节点方式采用胜率返回模式。测试结果显示该种程序表现不稳定。在好的情况下可以达到访问次数返回的效果，不好的情况下接近于在限定的模拟范围内随机选子。

## 3.3 纯粹模拟（访问次数返回）

访问次数返回的纯粹模拟的组件以主程序、玩家输入区块、电脑模拟区块组成。其中电脑模拟区块中的选取节点方式采用访问次数模式。测试结果显示该种程序表现较为稳定，通常在第二次运行主程序及以后表现较为良好。平常情况下可以识别玩家棋盘上的三连和四连并进行截堵，或者是识别到玩家的三子或四子中的空位进行填塞，识别率在80%左右；另一方面，也会积极自行构建使己方棋子连续的阵型，测试时经常会出现回过神来时它已经有很多线快完成，接近于“套路”的情况（这也可能是因为测试者技术不佳）。

## 

## 3.4 模拟+规则（胜率返回）

在胜率返回的纯粹模拟基础之上增加了Leap规则，每次电脑下棋时，首先判断规则，如判断为需要使用规则的情况则使用规则；否则进入模拟。对玩家三连、四连、空位的填塞率由于规则的存在达到100%，模拟时效果等同于相应的纯粹模拟。以下为几次测试结果的图示：

## 3.5 模拟+规则（访问次数返回）

在访问次数返回的纯粹模拟基础之上增加了Leap规则，每次电脑下棋时，首先判断规则，如判断为需要使用规则的情况则使用规则；否则进入模拟。对玩家三连、四连、空位的填塞率由于规则的存在达到100%，但构建己方获胜情形的积极程度有所降低，模拟时效果等同于相应的纯粹模拟。

## 3.6 范围遍历

由于模拟次数有限，曾经担心利用rank的排序判断有一些点可能由于随机下子的前几盘输了之后会出现永远访问不到的情况，并且这些子可能是比较重要的备选点。所以设计了这个范围遍历模型，在给定的有子范围的一定扩展范围内，每一个点随机下100盘，随机的方法仍然是先前的随机落点。最后每一个点返回一个胜率，最终选择胜率最高的一个点。尽管这样子实现了运算资源在每个可能子之间均匀分配，但同时也在很多不重要的点浪费了大量资源。

这个遍历的方法与前文的MCTS不同在与每次电脑模拟的第一步。在MCTS中的第一步会使用rank的矩阵记录依据公式的返回值，并在排序中选择出最大的值作为下一次的电脑模拟第一步expand。但是在遍历中每一次的第一步都是给定的，即每一个范围的点作为第一步一共产生100局模拟结果。

在试验中这种方法在前期显得比较有效，但是在后期情况复杂之后就出现了很多错误。一方面运算时间随着范围扩大线性增长，受限于每一个点的随即模拟盘数，另一方面并没有出现收敛于一个比较合理的落子，所以这种方法最后也没有作为最好的AI版本。

## 3.7 规则&模拟混合

由于实践中出现了堵塞的规则不一定是最优的一步，所以我们设计了一个标准，让70%的情况下要经过规则判断，即在出现连三等情况下会有立即判断。在另外30%的情况下直接跳过规则的判断过程，直接进入模拟得出模拟结果。实现的方法是生成一个0-1随机数，在大于0.3时进入规则判断的过程，小于0.3时跳过。其余的过程与模拟+规则（访问次数返回）相同。

## 3.8 机机对战（设想）

为了得出最佳的AI版本，我们考虑设计一个脚本使不同的AI自行比赛，得出最佳的AI版本。在前期试验中我们选出的是范围遍历与模拟+规则（访问次数返回）两个版本的AI进行模拟比赛，不过由于每一步的AI的都需要花费大量运算资源，在试验中没有得出最终结果。

# 4.评价

## 4.1难点综述

1）找出各个环节的具体实现途径。

虽然论文中已经对MCTS的思路进行了详尽的叙述，并且给出了伪代码，但是在具体的环节中如何实现这些想法则没有更进一步的说明。在进行编程之前，我们需要深入到每个环节的细节，通过绘制流程图找出适当的程序结构，并且构思尽量简单的实现方式，不断对程序进行优化。这个过程贯穿所有程序区块的构思和编写。

列举具体问题如下：

1. 棋盘的结构。目前的棋盘由多层矩阵组成，并且被包覆在图形界面下。具体而言，第一层是玩家可视的棋盘、与可视棋盘直接一一对应的棋盘矩阵、记录棋盘上空点的技术举证和三个在进入模拟器时生成的矩阵，记录每个点的相关参数。每一层都有特定的、不可替代的功能。在编写之初，识别所有必要的功能，以合适的形式把它表现出来十分具有挑战性。而且目前的版本也是优化后的产物。最初的版本中，模拟器里的矩阵采取了一维向量的形式，需要在向量和矩阵之间来回转化，排序时调用MyMax函数。后来由于转化过程增加了多余计算量而且容易出错而放弃，编写了SuperMax函数（在发现find函数会返回多个位置之后）对各个矩阵进行处理。

2.对于模拟器AIthink，初次进行编写时未进行区块化，而是把所有的流程都放在一个Simulation的函数文件中。编写完成后多次运行出错，在可以顺利运行之后模拟效果很不理想，而且难以找出到底是哪一步出现问题。鉴于这个情况，我们对Simulation进行了拆分，完全按照MCTS的四个步骤把不同的功能放入不同的函数文件中再重新进行组装，确保了结构的清晰，而且调试时更加容易定位问题的所在。

3. 确定规则的实现方式。每一个简单的规则在实现的时候都显得非常繁复，而且容易在细节上出错。规则的主要难点在于需要进行大量分类讨论。完全拆分会带来结构上的清晰，但会增加很多代码；合并则容易出错。即使是一个简单的Block规则，在有EndGame函数的基础上，也经过了多次调整才能正常运作。问题完全出现在细节上，非常难以发现。

4. 堵塞规则设计。规则的使用的难点主要体现在如何设计规则使电脑运用规则在三个连起来或者四个缺一的情况下会自动判断处理。并且这其中包含很多种情况，每种情况下最新一次的落子又可能是其中的任何一个，难以穷尽。并且很有可能同时出现多个需要使用堵或者塞的规则策略的地方，这时判断的顺序很重要，会决定做出反应的位置。

并且曾经出现过在对手已经出现连成三子的情况下，迅速去堵的策略不是最佳的策略，在单纯模拟下会出现更优的策略选择。此时的策略选择难以提前预判，难以在程序本身预先设定。

5. 循环次数限制。是限于游戏可玩性，每次的循环次数都没有设计的非常多，一方面次数较多会提高每一步的准确性，但同时也会增加每一步所花的时间。在游戏进行过程中，模拟的范围会进一步扩大，所以模拟的范围本身就已经在不断扩大，模拟时间在增加。

由于每一步使用的数据是上一步的返回值在rank里新的排序，而且每一盘返回结果之前需要很多判断，难以用矩阵同时模拟多场比赛，提高模拟的速度。

6. 禁手无避免。五子棋国际规则中存在禁手一说，有“三、三”、“四、四”和“长连”，包括“四、三、三”和“四、四、三”，即黑棋只能以“四、三”取胜等等。由于只存在于先下一方，即黑子。在我们未输入此规则前提下，为避免AI出于劣势，我们强制由电脑先下子。但是禁手的规则还是暂时未出现在整个游戏设计中。

7. GUI设计的不足。由于缺乏关于Matlab的GUI的设计知识，在人机交互上只能采用原始的坐标输入，而不能进行更直接的点击方式的人机交互，在与电脑交流的过程是比较麻烦的一个过程。交互时间比较长，在输入坐标的时候也容易出错，所以利用GUI克服交互的问题是未克服的难点。

2）Debug的过程。

Bug主要有两种形态。其一，由于不合理的结构或者未考虑到的细节问题，Matlab运行时报错；其二，实现效果极其不理想或者严重偏离预期。对于第一种，需要找到具体出错的位置（Matlab会提供它运行不下去的地方，但不一定就是造成问题的地方）和出错原因。对于第二种，则需对每个可能的地方进行地毯式的除错，设置断点手动运行多次，或者用测试用的输入变量检测各个函数的运行情况，甚至是检查整个思路上有没有问题。

列举具体问题如下：

1. 初代模拟器Simulation。在多次报错之后修理至可以正常运行，但模拟效果极其不理想，近似于随机落子。问题至今不明，并且在多次调试无果之后直接废弃，对功能进行拆分和组装。

2. 二代模拟器AIthink。虽然结构非常明晰，运行时也没有报错，但是在试运行时，发现模拟速度异常快，即使计算次数增加到百万次的等级仍然在几秒之内完成，而且模拟效果不理想。调试时在整个程序的各个函数中设置多个断点，并增加了一些指示性的变量（如DefaultPolicy中的i）。手动运行100次之后发现模拟过程正常，各个参数矩阵内元素值也正常变更，但整体仍然不正常。后来发现是模拟内部的棋盘矩阵和计数矩阵在每次模拟开始的时候没有恢复到进入模拟器时的初始值，而是随着每次模拟变更而变更，因此当棋盘上所有可模拟的位置均被模拟完一次之后，棋盘被占满，模拟自动结束。修复这个问题之后模拟器正常运行。

3. Block规则。在所有测试矩阵上试用均可行，但加入主程序之后则不能正常运行，无法正确识别是否需要截堵（表现为二连时即开始截堵），并且返回正确的截堵位置。后来发现是因为插入主程序的位置不对，未能获取最新的棋盘矩阵所致。

4. Rank矩阵内的评分设置，以及最终选取节点的标准。虽然基本是按照UCT给出的公式来进行，但由于不打算获取上一次选择节点的被访问次数，所以n值的设置由我们自行给出，可能影响了模拟效果，经过多次调整。常数Cp­一开始时被误设为，使公式中exploration部分的权重变大，推测这一错误使收敛速度变慢，造成本来就不足的模拟次数更加不足，进而影响了模拟效果。此外，最初的版本中选取节点标准均采取净胜率，返回值不理想；该为累计被访问次数标准后，模拟效果大幅提升。

## 4.2 未实现的优化

1） DefaultPolicy中模拟玩家和机器落子的随机模拟器Simulation的运作方式目前过于粗放，而且预计在棋盘快满的时候速度会变慢。我们打算用计数矩阵对其进行优化，具体做法是实时生成范围（1：sum(sum(CountZeros))）内的随机数N，然后把这个随机数还原到计数矩阵第N个非零点的位置。矩阵中的元素按行来计数（即第2行第1个元素编号是16）。

2） BackPropagation中用到UCT的公式中的第一项目前采取的是累计净胜率（若模拟中机器输了，累计胜次减一）。我们打算改成累计胜率（累计胜次只记录机器胜的盘数，对机器输或平局不做处理），这个改动不一定能优化模拟结果。UCT公式还有一些可操作的空间，例如第二项的常数n。这些我们都还没有尝试。

3） AIthink中选取最优节点的方式还有可以探究的空间。目前尝试过的是选取累计净胜率最高和被访问次数最多的节点两种方式。余下可以尝试的方式还有：累计胜次最高、累计胜率最高、累计胜次和累计被访问次数同时最高等。

## 4.3 感想

【学生一】 比较有趣的一点是做project时心境的变化。刚读完论文之后，觉得这个算法实现起来非常有难度。但是真的把所有流程想清楚，并且把程序写出来之后，又觉得这个算法实际上清晰又简洁，而且基本不能理解为什么之前会觉得难。

当然最初的debug确实非常困难。因为思路是正确的，但由于细节上的问题，一直都会出错，经常是接连很多个小时百思不得其解，最后发现是个小问题，简直崩溃又好笑。后来做多了几次之后，基本上看到代码就能有一种感觉，运行会不会错可以看得出；对需要注意的细节也有了一些直觉上的把握。仿佛是理解了机器的“想法”一样。

另一个重要的感想在于似乎发现了AI和人类思维的一个重要区别。无论如何，AI的智能——至少是在我们用的这个算法中——纯粹基于计算次数，所以硬件计算能力的高低直接决定了它的反应速度和效果。相对地，我们在思考的时候不会重复进行这样模拟到终局的计算，而是根据经验直接排除了大量的选项，或者逻辑地找出了一些模式，是个模糊而神奇的过程。实际上，这个算法是通过大量计算排除了大量选项（像是在制造经验），然后非自觉地到达了一些模式，在测试几个最终版本的时候不时表现得非常“阴险”，一个不小心我就输了，没办法测试那些需要测试的功能。即使它已经这么“聪明”了，还是总给我一种似是而非的感觉，在模拟给出准确结果的时候会让人怀疑它是不是真的在思考了，但模拟不准的时候立刻像是露出马脚一般。

实际上，MCTS给我的印象像是一种土豪的算法。虽然不像遍历的算法那么挥霍，但总之还是依赖于大量计算。可想而知，如果我们拿到一台一秒钟能算几百万次的计算机，即使不像现在这样限制范围、根据范围大小限定次数，也可以拿到很好的模拟结果。在每一次进行到终局的模拟中机器都在乱下，通过大量乱下的叠加找到一个近似最优的点，这种做法总是让我想到猴子打字机。理论上来说，人类的智慧不可能超越猴子打字机，只要语言结构没有发生变化，语言能到达的一切真理和杰作最终都会出现在猴子打字机上。然而，像猴子打字机那样利用随机性的过程接近于绝对的浪费，丝毫没有效率可言。从这个意义上讲，给出了准确模拟结果的这个AI并没有什么了不起，值得敬佩的是那些给出了MCTS理论基础，证明了这个模拟方式最终会收敛到最优解的那些数学家；尤其是UCT的公式，感觉简直是天才的发明。

不管怎么说，教会电脑下棋这个过程非常有成就感，虽然有时候一编就是一整天也没觉得有多累；看着它慢慢变聪明最后到我不小心就会输给它的地步也很有趣。由于时间限制还有一些优化的想法没有实现，更别提做出一族利用蒙特卡罗算法解决问题的程序的这个野心。不过大概以后还有的是机会去写吧。

【学生二】 可以利用论文的知识与本学期的matlab知识设计出一个简单的AI模型，在看到AI通过模拟下出一些自己都没有看出来的可以取胜的步数的时候很高兴。一方面可以避免了网络上大量的类似五子棋AI都是用一个局面的评分系统，对不同的局部有不同的评分选择落子位置的常规做法，另一方面也证明了论文在推导上的正确性，虽然我们的模拟次数没有非常高的数量级。在做的过程中也出现了很多个可运行的版本，将我们设计的各个部分不同的组合方式，可以得到反应速度、准确度差距悬殊的AI。但我们一直在希望提高我们的AI的速度与准确度，期间每次的提高都会令人兴奋。当然我们也存在着GUI等没有克服的问题，需要更多地去继续学习matlab的知识，更多地去提高自己。

最后感谢老师一学期的教学，从基本的matlab操作知识到每个例子背后的算法思想，给我们带来了很多启发与实际技能的提高，为我们未来进一步运用这个工具打下了基础。

# 5. 附录

## 5.1 程序使用方式

打开每个文件夹中的主程序（文件名开头都是MainProject，后面根据版本不同有不同的命名），运行。建议用【模拟+规则（访问次数返回）】或【规则&模拟混合】进行测试。第一次运行的模拟精度可能不佳。如果要增加模拟器的运行次数，可以在AIthink函数里改变运行次数的系数，目前采取的次数是100\*（模拟范围大小）。

把可视化的棋盘看做xoy平面，按顺序输入的数组（x，y）对应的是横纵坐标而不是矩阵里的行标和列标，这一点请格外注意。

## 5.2 组内分工

骆锎源：看论文，整理总体思路，制作模拟器（MCTS）、EndGame规则以及Block规则，完善PPT，撰写报告、整理报告

张一帆：看论文，完善总体思路，制作主程序、图形界面和is\_legal规则以及Leap规则，制作模拟器（遍历），制作PPT，撰写报告

## 5.3 参考文献

1. Browne, Cameron et al. “A Survey of Monte Carlo Tree Search Methods”. *IEEE Transactions on Computational Intelligence and AI in Games*. Vol. 4. No. 1. March 2012.

2. 王长飞等：《智能五子棋算法的设计实现》，*系统仿真学报*，第21卷，第四期，2009年2月。