人工智能导论 Project-1 报告

黄翔 2017013570 清华大学软件学院

从业臻 2017013599 清华大学软件学院

1. 实验概述

1.1 实验选题

实现基于 Minimax 搜索和 α-β 剪枝算法的人机对弈五子棋 AI 程序。

1.2 实验环境

- ✓ 操作系统：Windows 10
- ✓ 集成开发环境：Visual Studio 2019/2017
- ✓ 编程语言：C++（C++ 11 标准）

1.3 实验内容

1.3.1 程序逻辑

- ✓ 对局：<newblack><newwhite><move x y>指令
- ✓ 提示：<tips>指令
- ✓ 悔棋：<withdraw>指令
- ✓ 复盘：<record x>指令

1.3.2 落子生成算法 (createmoves)

- ✓ 遍历寻找所有两格米字型范围内有棋子的位置
- ✓ 对候选落点启发式估值（evaluatePoint），以此为依据筛选与排序

1.3.3 游戏胜负判断算法 (gameover)

- ✓ 判断四个方向上，上一落点是否形成五子连珠

1.3.4 搜索算法 (searchmove)

- ✓ Minimax 搜索
- ✓ α-β 优化剪枝
- ✓ 迭代加深
- ✓ 散列表优化

1.3.5 局面评估算法 (evaluate)

- ✓ 字符串匹配的方式进行模式匹配，计算总得分

1.3.6 总结

总的来说，实验完成了所有必做项，并完成了如下额外项：

- ✓ 迭代加深，优化初试顺序
- ✓ 散列表优化
- ✓ 提示&悔棋&复盘操作
- ✓ 落子生成算法中依据启发式估值对候选落点筛选与排序，本身剪枝的同时也使得 α-β 剪枝效果更好。

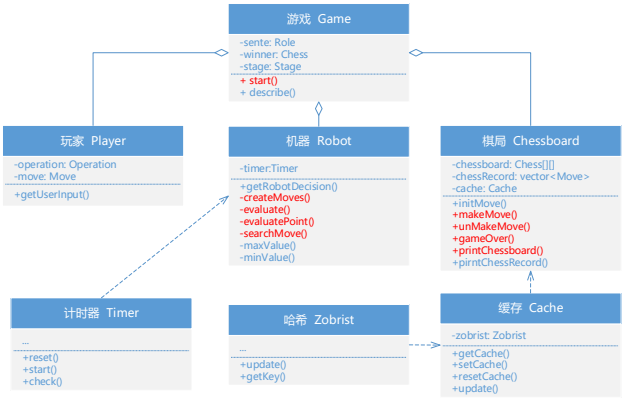
2. 程序架构

在助教提供的函数模板基础上，我们对程序进行了适当的重构，将相关的逻辑进行抽象封装，以进一步提升代码可读性与维护性，实现高内聚低耦合的逻辑结构。

2.1 实验选题

文件	描述
main.cpp	内含 main 函数，为程序入口点
chessboard.h/.cpp	内含 Chessboard 棋局类定义及实现，包括棋局信息记录，执行走法(makeMove)、撤销走法(unMakeMove)、胜负判断等功能(gameOver)
game.h/.cpp	内含 Game 游戏类定义及实现，包括游戏逻辑控制等功能
player.h/.cpp	内含 Player 玩家类定义及实现，包括接受玩家输入、解析玩家指令等功能
robot.h/.cpp	内含 Robot 机器类定义及实现，包括 AI 程序核心算法：获取合法落子点(createMoves)、搜索(searchMove)、估值(evaluate&evaluatePoint)等
timer.h/.cpp	内含 Timer 计时器类定义及实现，包括计时相关功能，用以辅助进行迭代加深
zobrist.h/.cpp	定义了 zobrist 类以及缓存类，用于实现五子棋局面的存储等，从而实现散列表优化
kmp.cpp	内含 KMP 字符串匹配算法实现，用于辅助局面判断

2.2 类结构



注意：部分类成员及函数参数、返回值类型已省略。

2.3 核心代码路径

模块	核心函数 （省略返回值与参数）	所在文件
程序逻辑	Game::start(); Chessboard::makeMove(); Chessboard::unMakeMove();	game.cpp chessboard.cpp
落子生成	Robot::createMoves();	robot.cpp
判断胜负	Chessboard::gameOver();	chessboard.cpp
搜索模块	Robot::searchMove(); Robot::maxValue(); Robot::minValue();	robot.cpp
评估模块	Robot::evaluatePoint(); Robot::evaluate();	robot.cpp

3. 核心数据结构

3.1 棋盘状态

3.1.1 结构定义

Chess chessboard[GRID_NUM + 1][GRID_NUM + 1];

std::vector<Move> chessRecord;

3.1.2 结构描述

程序使用二维数组 chessboard 标记棋盘棋面状态；使用结构体 Move 的 vector 容器变量 chessRecord 标记落子记录。由于五子棋固定黑方先行落子，之后双方依次交替落子，根据 chessRecord 元素的索引，我们便可得知该步的棋色，根据 chessRecord 元素的个数，便可得知下一步的棋色。

执行走法与撤销走法时，显然仅需通过位置索引改变 chessboard 对应变量的值，以及对 chessRecord 压入或弹出元素，时间复杂度均为 O(1)。

3.2 棋盘辅助数组

3.2.1 结构定义

int possibleMoves[GRID_NUM + 1][GRID_NUM + 1];

char horizontals[GRID_NUM][GRID_NUM + 3];

char verticals[GRID_NUM][GRID_NUM + 3];

char up_diagonals[DIAGONAL_NUM][GRID_NUM + 3];

char down_diagonals[DIAGONAL_NUM][GRID_NUM + 3];

int horizontal_piece_count[2][GRID_NUM + 1];

int vertical_piece_count[2][GRID_NUM + 1];

int updiagonal_piece_count[2][DIAGONAL_NUM + 1];

int downdiagonal_piece_count[2][DIAGONAL_NUM + 1];

3.2.2 结构描述

上述数组中第一个数组 possibleMoves 记录了棋盘上每一位置 2 格的米字型范围内棋子的个数。如果该位置本身有棋子，那么数组对应值加 17（米字型范围内最多有 16 个棋子，这样就能区分该位置是否有棋子）。产生候选落点的时候，如果 possibleMoves 值小于 17 且大于 0，说明其米字型范围内有棋子，且该位置是空的，可以落子。possibleMoves 数组在落子和取消落子时维护，这样可以节省 createMoves 时的计算开销。

上述数组中的四个字符二维数组，代表棋盘的所有行、列、左上-右下对角线和右上-左下对角线，方便局面评估时直接作为字符串匹配算法的参数。在落子和撤销时维护这四个数组。

此外，由于大多数时候棋盘是比较空的，没有必要匹配所有模式，故又维护了四个整型数组，记录所有行、列、对角线中黑/白子个数。比如，某行只有 1 个白子，那么没有必要进行任何白棋模式的匹配了。同样在落子和撤销时维护这四个数组。

3.3 棋型模式与价值

3.3.1 结构定义

```
struct pattern {
    char P[7];
    int m;
    int pi[8];
};
pattern black_p[20];
pattern white_p[20];
int cost_self[20];
int cost_opp[20];
```

3.3.2 结构描述

pattern 结构包括模式本身 P（如“011110”）、其长度 m、KMP 算法定义下的前缀数组 pi。black_p 和 white_p 记录了所有黑白棋型的模式；cost_self 和 cost_opp 记录了这些模式己方和对方的价值，具体赋值见代码 Robot 类的构造函数。

4. 核心算法

4.1 落子生成算法 (createMoves)

为了减少 branching factor，只考虑两格远的米字型范围之内有棋子的空位置（一共 8*2=16 个位置）。然而，这样还是会有很多候选位置，因此对每个候选位置进行启发式的估值（这个估值和棋面整体估值不同，只考虑该位置落子的评分，复杂度也小的多）。估值方式是：如果有两个回合之内必胜/不堵必败的情况——具体包括己方活五、对方活五、己方活四、对方活四和己方多冲四或冲四活三，则只保留优先级最高的情况；否则，如果有较可能致胜/不堵较可能败的情况——具体包括己方多活三或单冲四、对方有冲四，则筛去更弱的情况；否则，仅对所有候选情况排序。排序的作用是可以让 α-β 剪枝更有高的剪枝比例。这里为了减少开销，采用了较为进攻性的策略，对于对方棋型判断止步于冲四，因为如果己方对方同时形成同等级必胜棋型，己方先胜。

4.2 游戏胜负判断算法 (gameover)

由于导致游戏结束的落子一定是最后一步，故只需要检查最后一步是否形成了五子连珠。最方便的判断方法就是计算四个方向上，与最后落子位置连续相邻的同色棋子的数量，只要有一个方向该数量大于 4 这说明有五子连珠。

4.3 搜索算法 (searchMove)

深度优先搜索：搜索算法的本质是深度优先搜索，根据 createMoves 给出的候选落子点，递归尝试，遇到边界条件（达到特定深度、超时或触发剪枝条件）时退出。

α-β 剪枝：结合课程内容，我们使用了 α-β 剪枝算法，通过 α、β 值的维护与比较，将显然已无法最优的分支进行剔除，减少了搜索树的分支数。

迭代加深：通过实现定时器类 Timer，在最外层循环添加时间判断条件，并设定搜索临界时间(THRESHOLD_TIME)、最小搜索层数(MIN_DEPTH)、最大搜索层数(MAX_DEPTH)，以使得搜索过程可以合理的时间内，尽可能多地进行深层搜索，在确保搜索质量的基础上避免算法大幅度超时。使用迭代加深后，AI 可以在正常的时间内完成落子（设置阈值为 3 秒）。此外，更深层的搜索可以优先尝试较低层数的最优路径，极有可能增大 α-β 剪枝的数量。

散列表优化：这部分参考了文档中提供的教程网站。Zobrist 键值技术提供了一种将可能情况是天文数字的棋盘局面用一个数字散列化的方法。建立一个[2][16][16]大小的 64 位随机数数组，新建一个随机数 key 作为表示棋盘局面的值，每进行一步落子操作就将 key 和数组对应位置数字进行异或运算。由于异或运算的性质，落子后再回退，key 不变。尽管一个 64 位整数比棋盘的可能情况数小得多，但发生碰撞的概率非常非常小。哈希表不可能存储 2^64 个表项，故设置哈希表的实际大小为 10M 的数量级，计算索要进行一次取模操作。哈希表的每一项存储了 key 的值、搜索深度、一个表示是准确值/上界/下界的 flag、局面评估值。

在搜索过程中，在每层开始的时候查找哈希表，若有该局面，且存储的该局面搜索深度大于当前搜索深度，则返回准确值/alpha/beta。此外，评估一个局面后，更新哈希表对应项的准确值；和 α-β 剪枝的思想类似，在搜索过程中，还可以更新哈希表对应项的上界/下界值。原理主要参考了助教提供的网页教程，不再赘述。

4.4 局面评估算法 (evaluate)

整个棋面的评估算法要调用很多次，因此减少开销很重要。根据五子棋的规则，有活五、活四、冲四、活三等模式。我们使用了 KMP 字符串匹配算法来优化模式匹配的效率，注意到由于模式是固定的，可以事先算出前缀数组，减少开销。为了进一步提高效率，我们维护了四个字符二维数组，代表棋盘的所有行、列、左上-右下对角线和右上-左下对角线，方便局面评估时直接作为 KMP 算法的参数。此外，由于大多数时候棋盘是比较空的，没有必要匹配所有模式，故又维护了四个数组，记录所有行、列、对角线中黑/白子个数。比如，某行只有 1 个白子，那么没有必要进行任何白棋模式的匹配了。

评估算法的好坏对 AI 的性能至关重要。在我们的实现中，综合借鉴了一些资料和文献中的棋型打分，可见代码中 Robot 类的构造函数。出于效率和性能的折衷考虑，并不考虑所有的棋型，而是考虑比较重要的大部分。最后，AI 能够在较占优势的黑方战胜一个成熟的五子棋 app 的最高难度，足以说明 AI 已经达到了可以与普通人一战的水平。

5. 实验结果

5.1 优化效果测量

如上文所述，本项目主要使用了四种方式加速搜索（α-β 剪枝、Zobrist 散列表、候选落点筛选与排序、迭代加深优化初试顺序）。

为了测量四种方法的优化效果，我们取 release x64 模式下，两种不同开局下黑棋第 5 步 AI 决策的时间来进行衡量。第一种开局为黑 8-8，白 7-9，黑 7-7，白 9-9；第二种开局为黑 8-8，白 8-9，黑 7-9，白 9-7。第 5 步的时候棋型还未展开，因此搜索空间中的必走棋比例很小，候选落点的筛选起的作用非常小。

首先，由于迭代加深本身增加了开销，单独测量优化初试顺序的效果（其他三种优化也存在）。测量搜索 6 层的时间，结果如下（时间取两次测量平均值）：

是否优化初试顺序	开局 1 决策时间（秒）	开局 2 决策时间（秒）
√	0.164	0.271
	0.183	0.305

首先测量使用了 α-β 剪枝、无迭代加深时，搜索 6 层的时间，结果如下（时间取两次测量平均值）：

是否对候选落点筛选与排序	是否使用 Zobrist 散列表	开局 1 决策时间（秒）	开局 2 决策时间（秒）

√	√	0.164	0.256
√		0.208	0.379
	√	43.832	64.888
		58.144	74.037

如果不使用 α-β 剪枝，搜索 6 层的时间将会过长。为了展现 α-β 剪枝的优化效果，下表再取 AI 搜索 5 层的时间来进行衡量（默认使用 Zobrist 散列表优化、无迭代加深）：

是否使用 α-β 剪枝	是否对候选落点筛选与排序	开局 1 决策时间（秒）	开局 2 决策时间（秒）
√	√	0.073	0.061
√		1.429	2.559
	√	71.790	95.554
		66.675	77.314

从三张表可以看出，

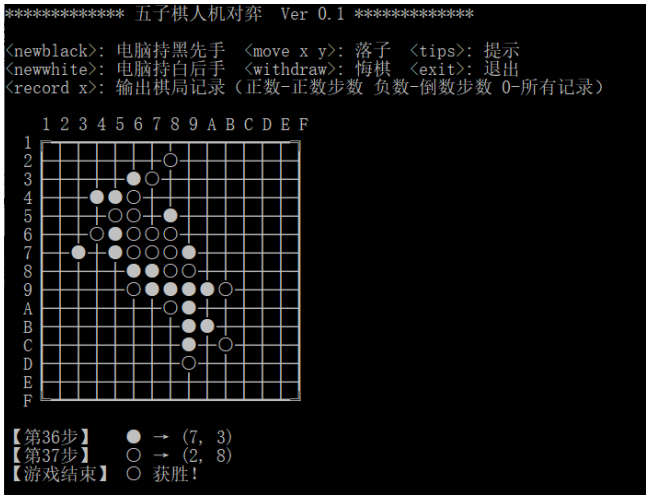
- ✓ 迭代加深时，优化初试顺序只有有限的效果，推测是因为浅层的最优选择未必对深层也是较优的（尤其开局）；
- ✓ 由于 α-β 剪枝是将复杂度的指数减半，其优化效果是极为明显的；
- ✓ 而对候选点的筛选与排序的排序部分出发点是使 α-β 剪枝尽可能剪去更多分支，因此在没有 α-β 剪枝的时候由于估值与排序的开销反而起了反作用，但是有 α-β 剪枝时效果非常明显；
- ✓ Zobrist 散列表优化的加速比较稳定，但增幅较小。

总的来说，通过三种优化方式，程序在大多数情况可以做到 1 秒内完成 6 层搜索并做出决策，效率提升了很多个数量级。

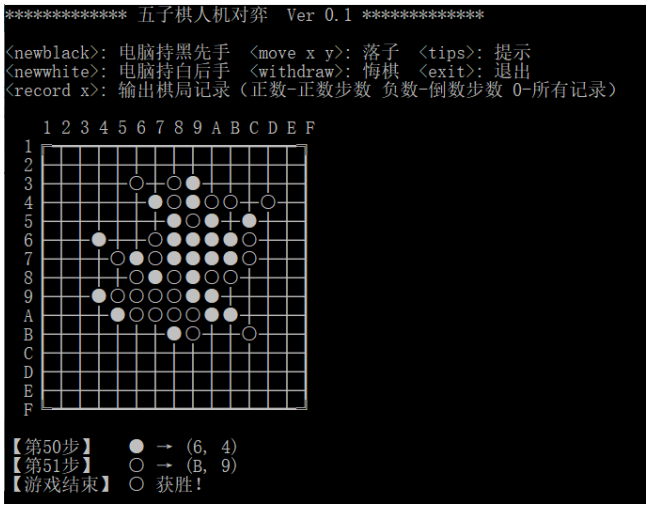
5.2 AI 水平测量

由于两位作者的五子棋水平稀松平常，因此使用华为快应用的五子棋 app 最高难度 AI 与本程序的 AI 对弈。

本程序 AI 持黑棋（占优），最大搜索层数 6，限时 3 秒，能够战胜敌方 AI 的最高难度，如下图所示。



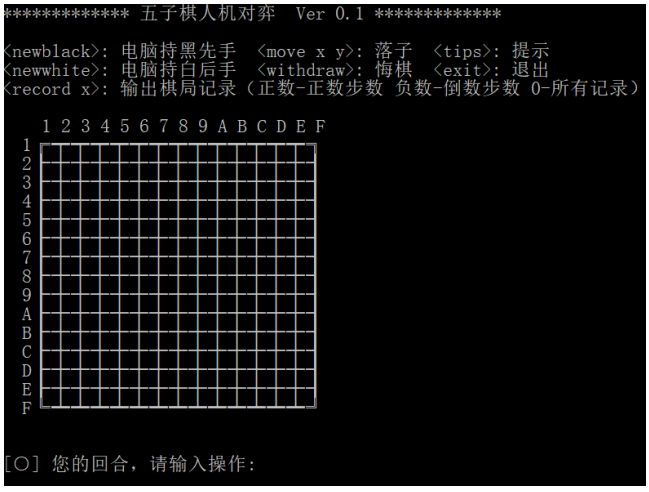
本程序 AI 持白棋虽会落败于敌方 AI 最高难度，但也能苦战 50 步后方落败。



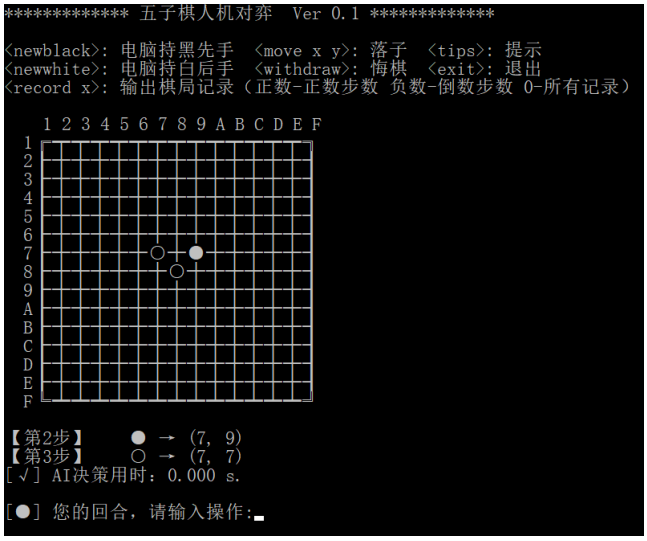
总的来说，本程序的 AI 能搜索 6 层，达到了不错的战斗力。

6. 使用说明

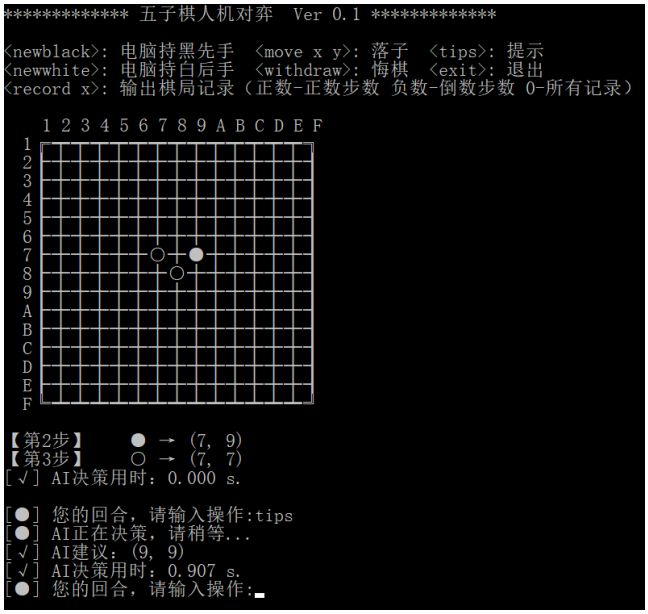
启动本程序后，会打印棋盘、所有选项：



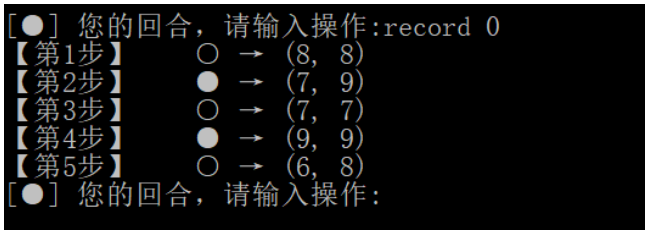
选择 newblack/newwhite 可以开始对弈或者重新开始新的一局；move x y 完成落子，其中 x 为竖直坐标，y 为水平坐标；withdraw 可以悔棋。决策过程中，程序会提示 AI 正在决策。完成落子后，程序会自动显示上两步，并输出 AI 决策用时。



在设计程序的时候，我们考虑到了 tips 选项，因此 AI 可以站在敌我双方的角度决策。tips 选项使用电脑 AI 一样的算法为玩家做出决策并将提示输出在屏幕上：



record x 是复盘功能，可以输出任意记录或者所有记录。



exit 退出程序。

7. 感想与收获

黄翔：在本次实验中，我们基于 α-β 剪枝 minimax 搜索，配合迭代加深与散列表剪枝，实现了一个简单的五子棋人机对抗 AI。在这次实验中，基于助教提供的模板思路，我们结合课程《面向对象程序设计》，将程序逻辑进行抽象封装，形成类与对象，提升代码的复用性、可读性与内聚性，降低其耦合性；结合课程《软件工程》，使用版本控制工具，在线上完成了两个人良好的协作开发；结合本课程《人工智能导论》，实现了对抗搜索的核心算法，确定了程序的基本架构；结合助教提供的参考论文，完成了迭代加深、散列表剪枝优化，设计了评估函数的初版。综合而言，这次实验，既是新知识的实践，也是旧知识的复习；实验开放性的解决方案，给了我们独立探索与发挥的空间，老师与助教良好的引导，则为我们指明了钻研的道路。尽管本次实验内容较为简单——在无禁手规则下的五子棋有着较为固定的套路，但本次实验无疑让我们更为深入地了解了搜索方法在人工智能中的应用，为我们之后的学习打下了良好的基础。

从业臻：如今我们对人工智能的认识基本都是机器学习、人工神经网络，但传统的搜索算法光芒不减，其思想依然是非常精妙和值得学习。通过经典的五子棋 AI 设计作业，我对 minimax、α-β 剪枝等都有了更加深刻的理解，也学到了 zobrist 这样精妙的算法。通过设计五子棋的评估函数，也对启发式函数的设计有了更深刻的理解。为了减少开销，提升效率，我也在优化数据结构和算法方面下了不少功夫，比如设计辅助数组来用空间换时间。

8. 参考资料

[1] http://www.xqbase.com/computer/search_hashing.htm

[2] http://www.xqbase.com/computer/search_iterative.htm

[3] https://github.com/lihongxun945/gobang

[4] https://www.cnblogs.com/maxuewei2/p/4825520.html

[5] 张明亮, 吴俊, 李凡长. 五子棋机器博弈系统评估函数的设计[J].计算机应用, 2012, 32(07):1969-1972.

[6] 裴博文.五子棋人工智能权重估值算法[J]. 电脑编程技巧与维护, 2008(6):69-75.