五子棋人机博弈问题

摘 要

五子棋博弈是对抗搜索中的经典案例。该课程设计能实现人机五子棋博弈。在智能体设计上，选取了恰当的估值函数，使用了 MinMax 算法，并配合 alpha-beta 方式进行剪枝来最终确定落子位置。本实验使用 C++语言和 Qt 框架编写，实现了图形化的 UI 界面，在完成实验要求之余，增加了信息提示和游戏难度选择等功能。

关键词：五子棋人机博弈，MinMax 算法，alpha-beta 剪枝

Man-Machine game of Gomoku

ABSTRACT

Gomoku is a classic case of game theory in the field of artificial intelligence. This assignment can realize Man-Machine game of Gomoku. I choose proper evaluation function, MinMax algorithm, alpha-beta pruning algorithm to decide the position when designing agent. In this experiment, UI is completed by using C++ language and Qt frame. I also add relative information and model select besides this project's request.

Key words：Man-Machine game of Gomoku，MinMax algorithm，alpha-beta pruning algorithm

目 录

装 订 线

1 引 言

1.1 五子棋游戏

五子棋是一种两人对弈的纯策略型棋类游戏，通常双方分别使用黑白两色的棋子，黑子先行，一人轮流一著下于棋盘空点处直线与横线的交叉点上，先在横线、直线或斜对角线上形成 5 子连线者获胜。

1.2 人机博弈典例

1952 年，亚瑟·李·塞缪尔编写了一个电脑西洋跳棋程序，该程序可以根据当前双方棋子的位置计算评估胜率，并以此作为落子的标准。为完善这一程序，他还采纳专业选手的意见进行改进，更通过程序自身的对抗来进行学习。1956 年 2 月 24 日，他的程序在一起电视公开对抗中打败了康涅狄格州的西洋跳棋冠军，并因此被称为“机器学习之父”，也被认为是计算机游戏的先驱。

IBM 开发的国际象棋超级电脑 Deep Bule(深蓝)，1997 年 5 月曾击败国际象棋世界冠军卡斯帕罗夫。

2014 年开始英国伦敦 Google DeepMind 着手开发人工智能围棋软件 AlphaGo。AlphaGo 使用了蒙特卡洛树搜索与两个深度神经网络相结合的方法，一个是以借助估值网络（value network）来评估大量的选点，一个是借助走棋网络（policy network）来选择落子，并使用强化学习进一步改善它。在这种设计下，电脑可以结合树状图的长远推断，又可像人类的大脑一样自发学习进行直觉训练，以提高下棋实力。AlphaGo 曾多次击败人类围棋世界冠军，如柯洁、李世石。

2 实验概述

2.1 实验目的

熟悉和掌握博弈树的启发式搜索过程、α-β剪枝算法和评价函数，并利用α-β剪枝算法开发一个五子棋人机博弈游戏。

2.2 实验内容

以五子棋人机博弈问题为例，实现α－β剪枝算法的求解程序（编程语言不限），要求设计适合五子棋博弈的评估函数。

要求初始界面显示15*15的空白棋盘，电脑执白棋，人执黑棋，界面置有重新开始、悔棋等操作。

设计五子棋程序的数据结构，具有评估棋势、选择落子、判断胜负等功能。

撰写实验报告，提交源代码（进行注释）、实验报告、汇报PPT。

2.3 本文所做的工作

本实验选择了适当的评估函数，使用了 MinMax+α-β剪枝算法进行最优落子位置的搜索。代码使用了C++语言编写并利用STL库简化了代码编写的难度。实验实现了五子棋的人机博弈，完成了实验内容中的全部要求。算法优化后的AI棋力较高，胜率明显高于普通的人类选手。

同时，使用了Qt这一跨平台C++图形用户界面应用程序来实现UI界面。用户可以在菜单栏中获得有关五子棋和该程序使用的相关知识，也可以经链接跳转至 Github 获取源码。用户可以在菜单栏中进行悔棋、重新开始、选择游戏难度的操作。程序能根据用户鼠标点击位置和 AI 的运算结果绘制棋子。在用户进行操作时，状态栏左侧会显示相应提示，状态栏右侧则会显示当前的执子方。

3　实验方案设计

3.1 总体设计思路与总体架构

将总体设计分为 UI 设计和内核设计两个部分，UI 部分负责与用户之间的交互和图形化的展示，内核部分主要进行搜索求解的过程。

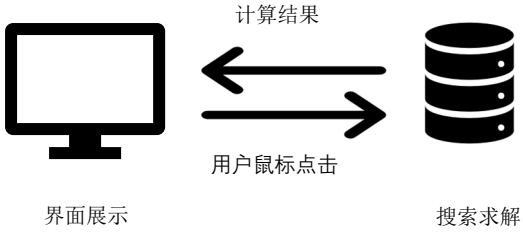

图 3.1 总体设计组成图

3.1.1 内核部分思路与框架

内核部分又可细分为人的内核和 AI 的内核。

人的内核部分是对鼠标位置和操作的判断以确定落子的位置。

AI 的内核则是通过启发式搜索确定当前局面下的最佳落子位置。

不论是人还是 AI 落子后，内核部分都需判断是否有一方获胜或有死局产生。

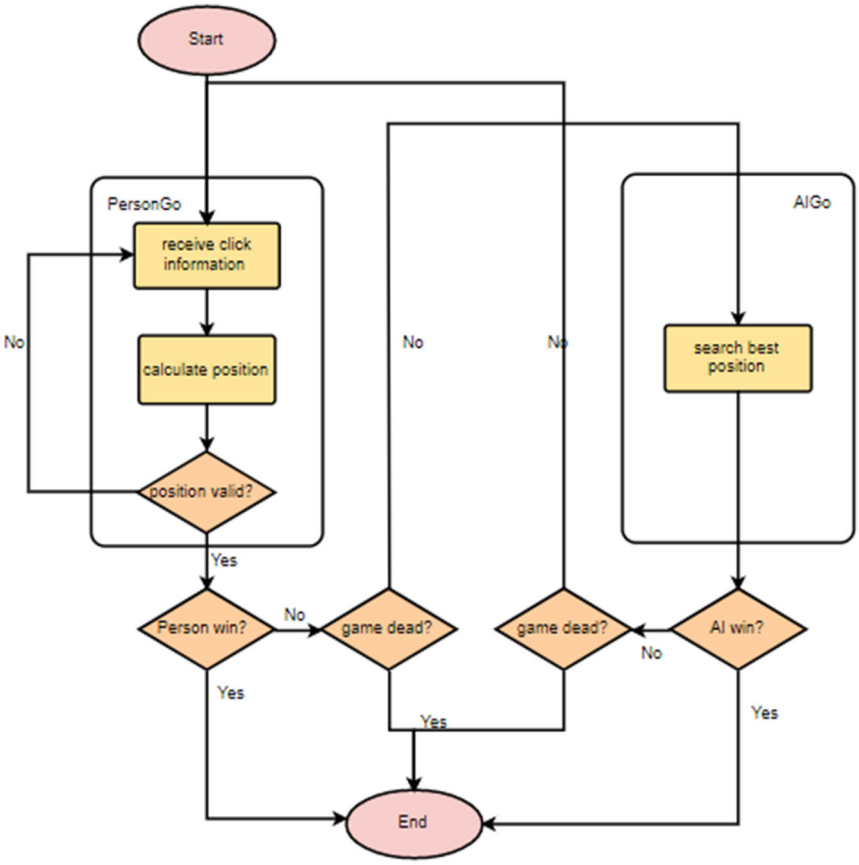

图 3.1.1 内核部分思路流程图

3.1.2 UI 部分思路与框架

UI 界面按功能分为菜单栏、棋盘、状态栏，共三大块。

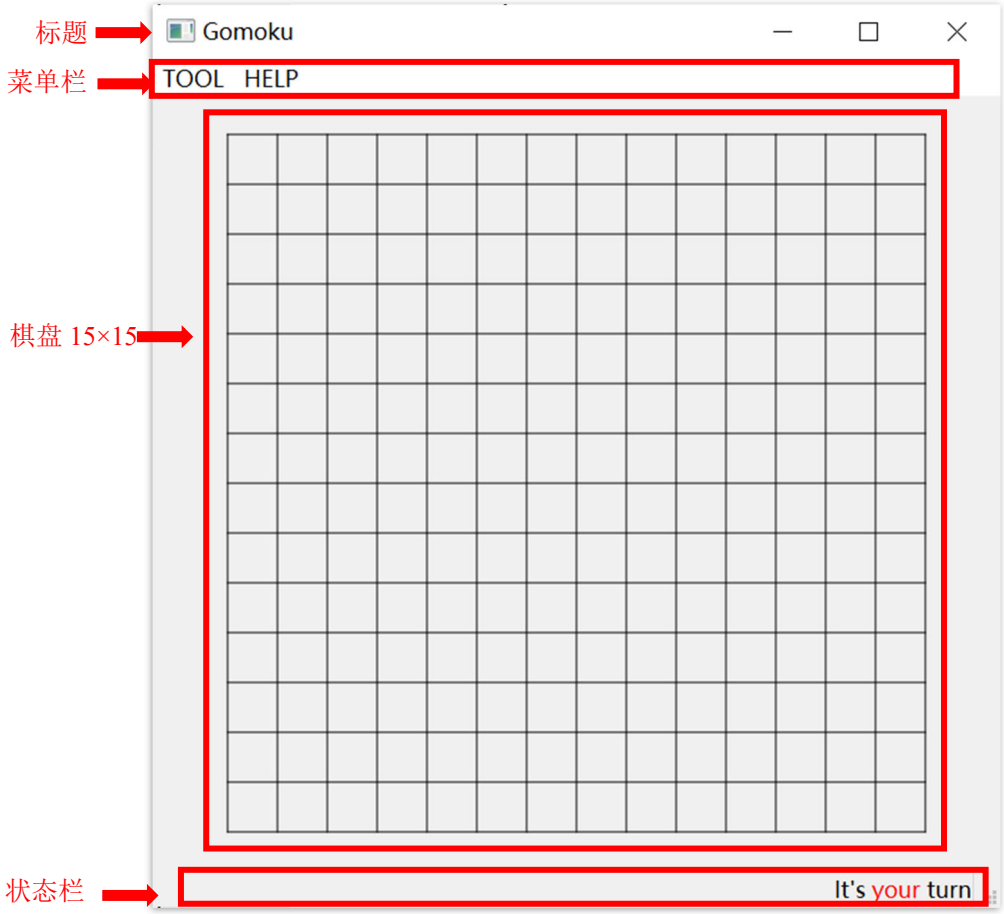

图 3.1.2.1 主界面展示

TOOL 中的下拉菜单可以进行悔棋、重置棋盘和模式选择。模式选择后会对棋盘进行重置，重置时如果棋盘上有棋子会有对话框提示。默认的模式是简单模式。

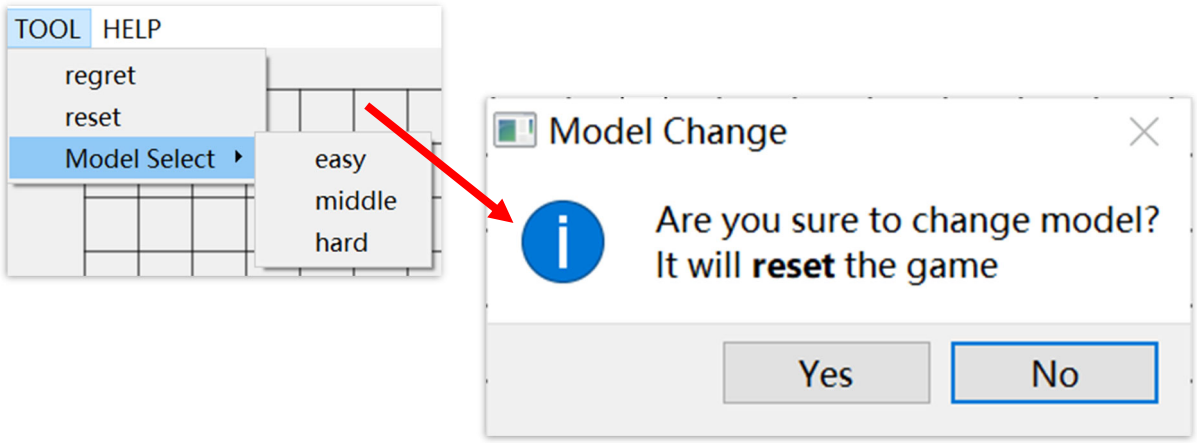

图 3.1.2.2 TOOL 下拉栏展示

HELP 中的下拉菜单可以对五子棋的有关规则、程序的使用进行查询，还可以了解更多有关该实验的内容。

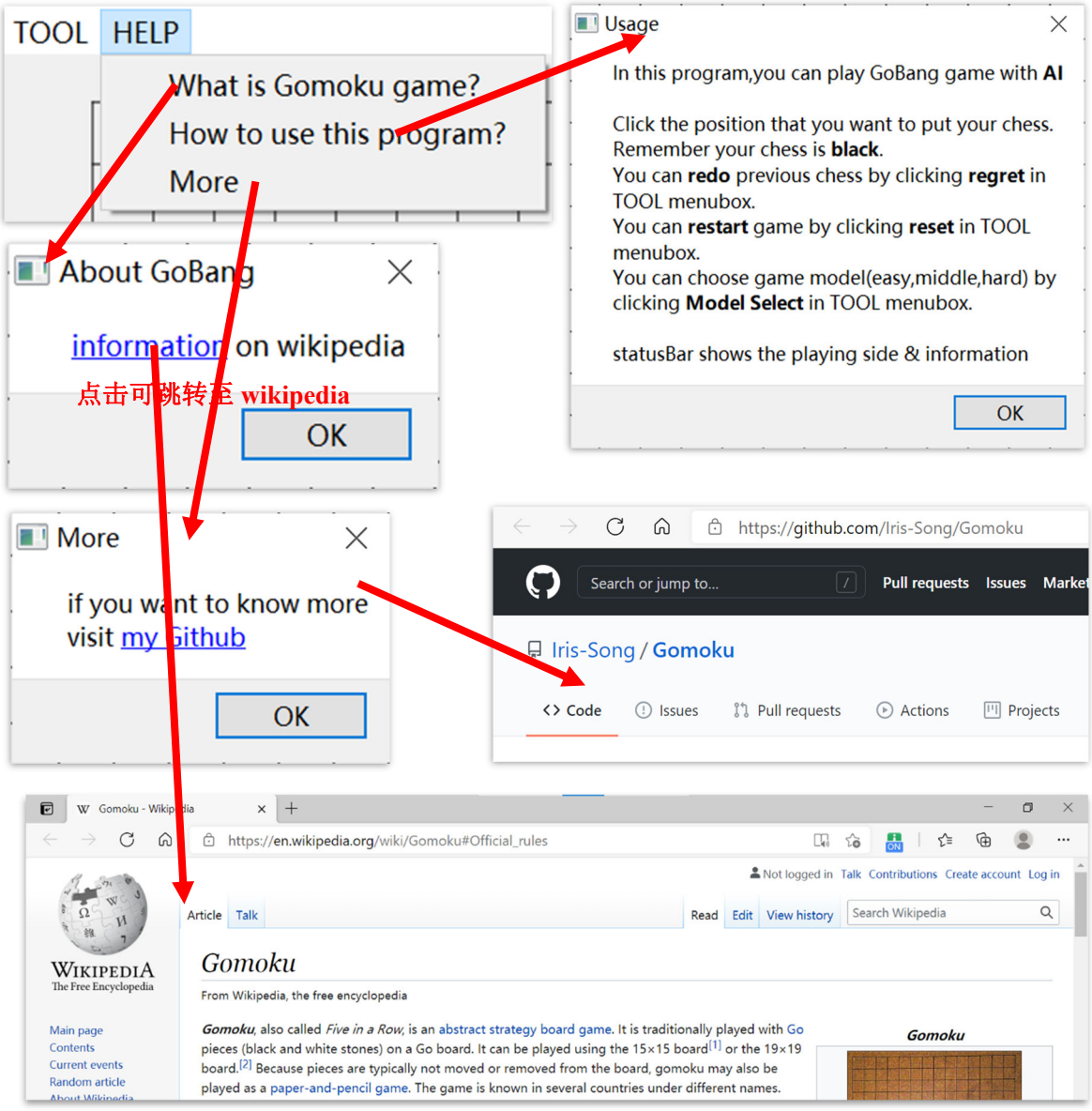

图 3.1.2.3 HELP 下拉栏展示

当鼠标移动到菜单栏中的下拉栏时，状态栏左侧会显示对应的提示。在程序的整个运行期间，状态栏右侧都会显示下一个落子的是哪一方。

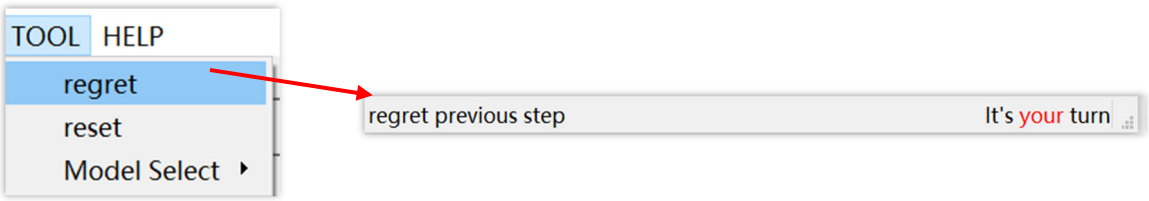

图 3.1.2.4 以鼠标移动到 regret 栏时的状态栏为例

点击窗口最右上角的退出按钮时会有对话框提示，询问是否确定退出，以免误触丢失现有棋局。当有一方胜利时会弹出对应的窗口并重置棋局。

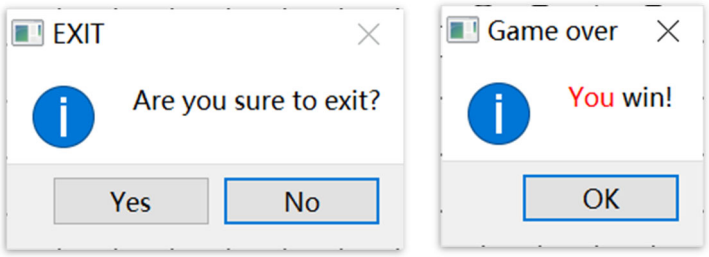

图 3.1.2.5 退出和胜利提示

棋盘部分，在人类玩家未落子前，会有小黑点跟随显示鼠标所指位置的落子点。点击后会对合法位置进行棋子绘制。当 AI 决策时，则不会有小黑点随着鼠标移动

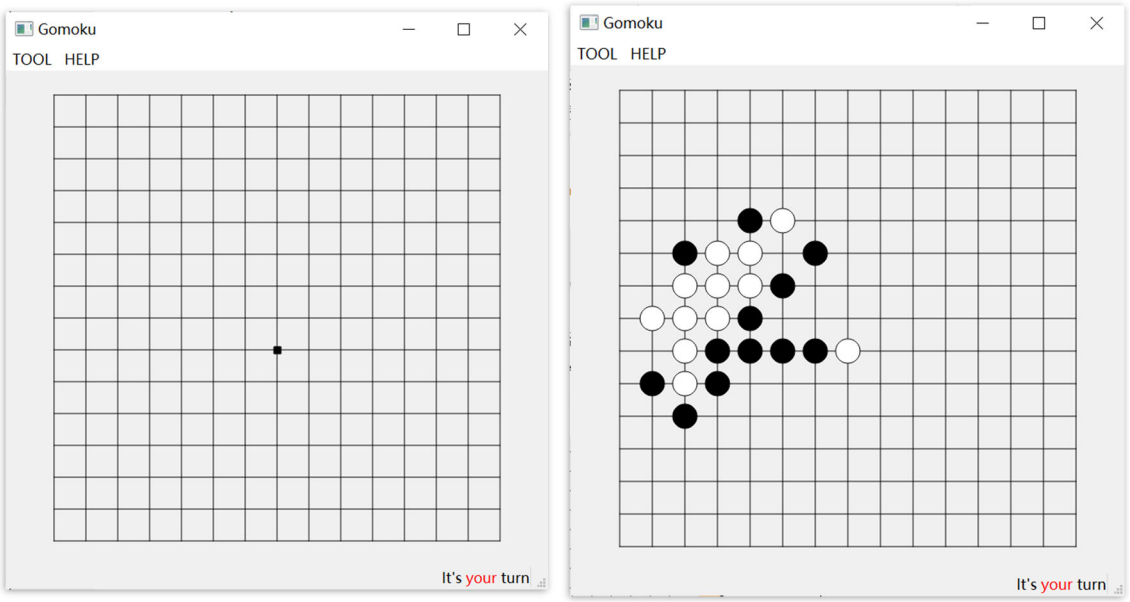

图 3.1.2.6 玩家鼠标移动落子预览和完整界面下的棋局

3.2 核心算法及基本原理

3.2.1 MinMax 搜索

MinMax 搜索（极小化极大搜索）是一种悲观搜索，即假设对手每一步都会将我方引入从当前看理论上价值最小的格局方向，即对手具有完美决策能力。树的每一个节点表示一种格局，而父子关系表示由父格局经过一步可以到达子格局。MinMax 通过对以当前格局为根的格局树搜索来确定下一步的选择。

3.2.2 alpha-beta 剪枝

如果按照 MinMax 算法来进行决策的话，需要的计算量是随着向后看的步数的增加而呈指数级增长的。但这些状态中包含很多不必要的状态，我们可以进行剪枝来提高运算速度。

Alpha-beta(α－β)剪枝的名称来自计算过程中传递的两个边界，这些边界基于已经看到的搜索

树部分来限制可能的解决方案集。其中，Alpha(α)表示目前所有可能解中的最大下界，Beta(β)表示目前所有可能解中的最小上界。

因此，如果搜索树上的一个节点被考虑作为最优解的路上的节点（或者说是这个节点被认为是有必要进行搜索的节点），那么它一定满足以下条件（N 是当前节点的估价值）：

α ≤ N ≤ β

在进行求解的过程中，α 和 β 会逐渐逼近。如果对于某一个节点，出现了 α ＞β 的情况，则说明该点一定不会产生最优解，所以，就不再对其进行扩展，这样就完成了对博弈树的剪枝。

3.2.3 估值函数

估值函数根据当前总的棋局进行估计。总的规则是己方的分数-对方的分数。

对于某一方来说，估值时会遍历该方的所有棋子进行一一估值。检查该棋子与周围 4 个或 5 个棋子是否满足某一棋型，并针对每一棋型进行打分，选择得分最高的棋型作为该子的得分。如果该位置棋子组成的最优棋型已经被打分过，则跳过该棋子。

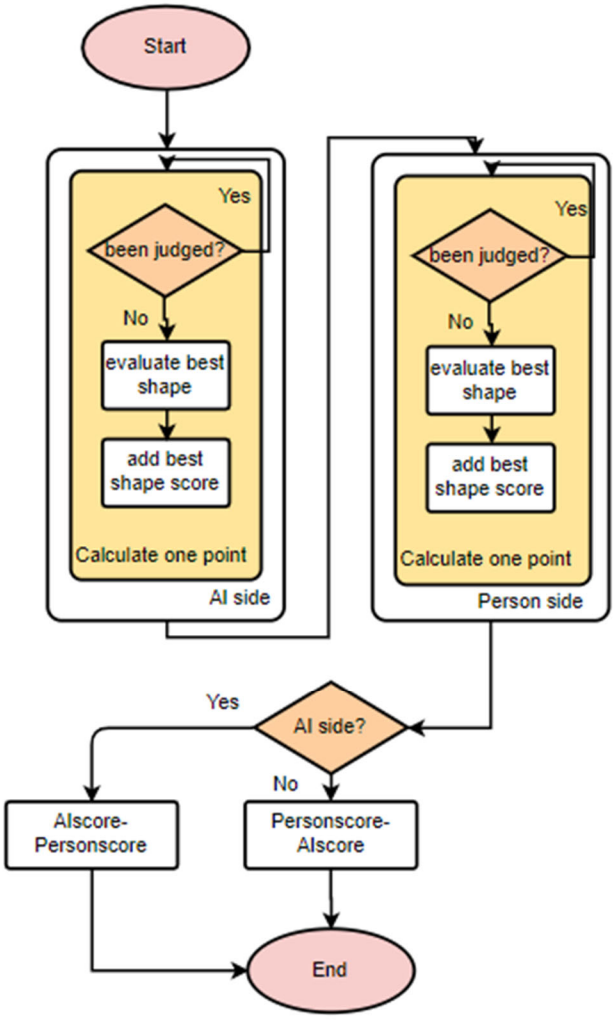

图 3.2.3 估值函数流程图

总的搜索算法由以上第三部分（MinMax 搜索、alpha-beta 剪枝、估值函数）共同组成，采用递归的方法，算法思路如下。

（1）如果是叶子结点或有一方获胜则返回估值。

（2）对于每个可以的落子位置（为了加快搜索的速度，只对周围有棋子的空白格进行搜索），放置对应方的棋子，并进行下一层的搜索得到估值 value。

（3）恢复棋局。

（4）如果得到的 value 大于 alpha：如果 value 大于等于 beta 则直接返回 beta,否则令 alpha 等于 value；如果该层正好等于搜索的最大层，则记录下该拓展节点的位置信息。

（5）若（4）不满足，则返回 alpha。

3.3 模块设计

3.3.1 内核模块设计

A. 类设计

定义了 Game 类来实现内核部分。

```
class Game
{
public:
    Game();
    std::vector<std::vector<int>> chessBoard; // 存储当前游戏棋盘和棋子的情况
    std::stack<Point> PersonStk;//人落子情况
    std::stack<Point> AIStk;//AI 落子情况
    bool playerFlag; // 标示下棋方
    GameStatus gameStatus; // 游戏状态

    void startGame(); // 开始游戏
    void actionByPerson(int row, int col); // 人执行下棋
    void actionByAI(int &clickRow, int &clickCol); // 机器执行下棋

    bool isWin(int row, int col); // 按照点判断游戏是否胜利
    bool isDead(); // 判断是否死局
    void regret();//悔棋

private:
    int MinMax(int Side, int Depth, int alpha, int beta);
    std::vector<Point> GetMoves();

    void updateGameMap(int row, int col);
```

```
    bool isWin();//整个棋局有无胜利
};
```

B. 开始游戏

调用 startGame 函数，初始化化棋盘和 game 类中的 public 变量。

C. 人落子

根据鼠标点击的位置判断最合理的落子位置并将其放于 PersonStk 栈顶。

D. AI 落子

调用 MinMax 函数，找到最佳落子位置并落子。将落子位置放于 AIStk 栈顶。

E. AI 搜索最佳位置

见 3.2。

F. 悔棋

如果当前决策方是 AI，即 AI 还没有落子，则仅 PersonStk 栈顶元素出栈，并将该位置棋盘元素标记为空。若当前决策放是人，则 AI 已经落子，PersonStk 和 AIStk 栈顶元素都需要出栈。

G. 棋盘更新

人落子或 AI 落子后，将对应位置的 chessBoard 变量标记为对应棋子，然后交换执子方。

3.3.2 UI 模块设计

UI 模块主要通过 Qt 框架实现。通过对相关函数的重载实现棋盘的绘制、对话框的提示、鼠标的有关操作等。

与内核模块之间的联系，详见 3.3.3。

A. 棋盘的绘制

重载了 paintEvent 函数，使用了 QPainter 等头文件，根据点的坐标绘制棋盘，根据 chessBoard 变量绘制棋盘上对应位置的棋子。

B.鼠标的操作

重载了 mouseMoveEvent 函数以实现鼠标移动。当鼠标移动到合理的区间范围时，确定在棋盘上对应的点，并更新 UI。

重载了 mouseReleaseEvent 函数，当鼠标点击释放后，若此时正好是人执棋，根据鼠标移动确定的落子位置进行对应的落子操作，在游戏未结束的状态下，给予一定的延迟时间，然后由 AI 进行相应操作。

C. 菜单栏及对话框的实现

利用信号-槽结构实现。使用了 QmessageBox、QmenuBar 等头文件，使用到了简单的 html 语言来实现对话框中的提示和网页的跳转。

D. 状态栏的实现

状态栏的左侧主要是对菜单栏中鼠标指向的提示，在窗口初始化时使用信号-槽结构实现。状态栏的右侧是当前的执子方显示，实现方法是在状态栏中添加了永久的 label，并能在 paintEvent 函数中随着 playerFlag 变量进行绘制。

装 订 线

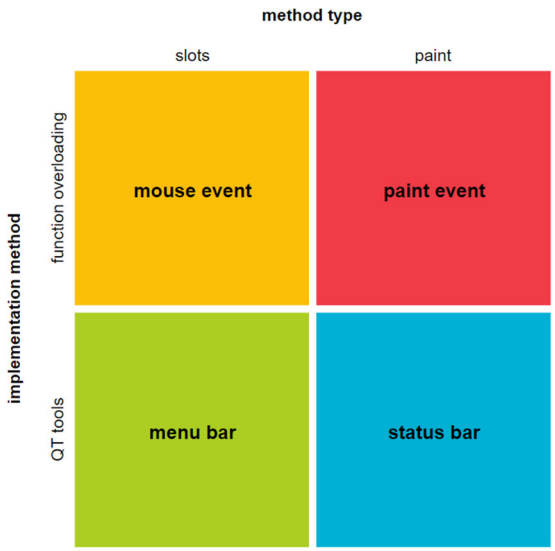

图 3.2.3 UI 模块示意图

3.3.3 内核部分与 UI 部分的联系

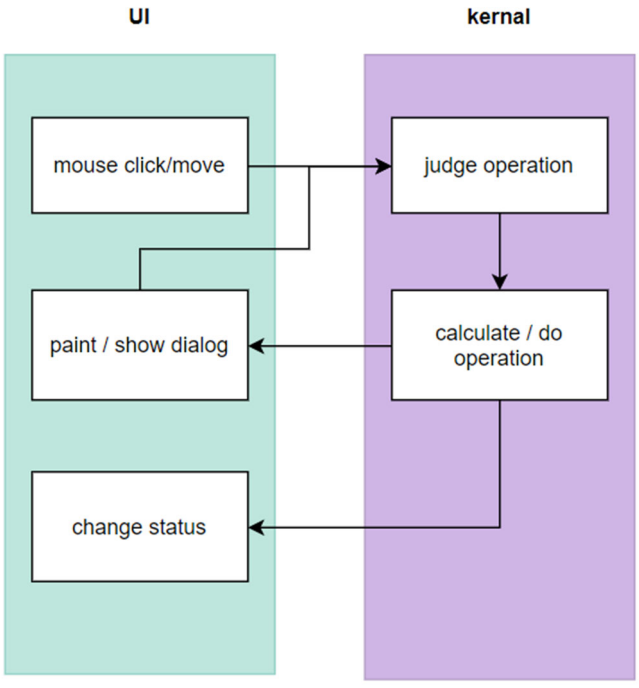

图 3.3.3 内核部分与 UI 部分的联系

3.4 创新内容或优化算法

A. 提高 AI 计算的速度

实际的博弈中，基于用户角度考虑，我只给了 AI 较短的决策时间。由于五子棋的比赛规则，一般在棋子周围的空白位置落子是最优选择。所以令 AI 仅搜索周围有棋子的空白格。

为了使 AI 在有限时间里能搜到尽量多的层数，选择在人的一方和 AI 的一方分别进行估值，

装 订 线

假设双方都会选择使自己得分最大的落子位置。这样不论搜索层数是奇数还是偶数，AI 都能获得当期情景下的最大值。

评估棋子前先检查该棋子原来有无被评估过，若被评估过则直接跳过。

多采用 if else 结构，加快判断速度。

B.难度选择

用户可根据需要选择游戏的难度，简单对应的是 MinMax 搜索 2 层。中等对应的是 MinMax 搜索 3 层。困难对应的是 MinMax 搜索 4 层。

C.落子预览

用户在点击棋盘之前能够看到将要落子的位置。

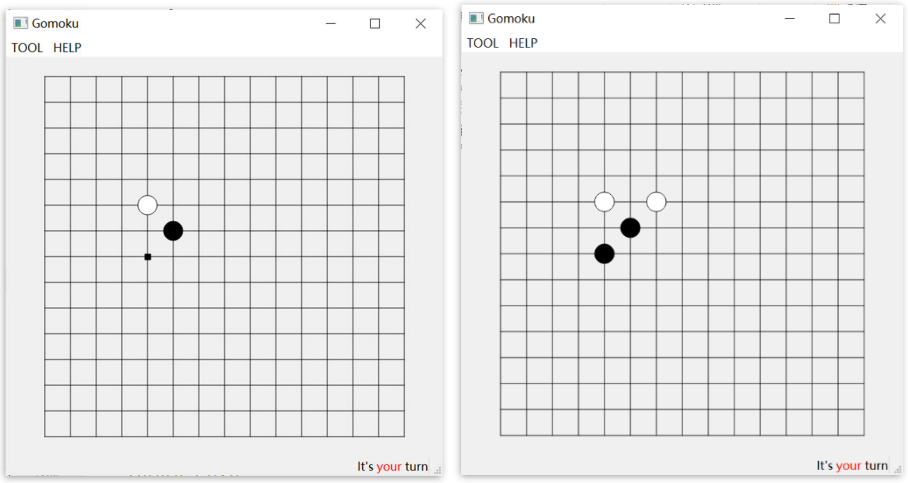

图 3.4 落子预览 （左：落子前，右：落子后）

4 实验过程

4.1 环境说明

硬件配置：Intel Core i5-10210U CPU

操作系统：64 位 Windows10

开发语言：C++

开发环境：Qt Creator (Qt 5.12.10 MSVC2017 64bit)

编译器：Microsoft Visual C++ Compiler 16.5.30104.148(x86_amd64)

调试器：CDB Engine

4.2 源代码文件清单及主要函数清单

4.2.1 头文件

A. head.h

定义整个程序中使用到的常量，包含关于 Point 类和 range 函数的声明。

B. mainwindow.h

包含与 UI 界面有关的头文件，声明了 MainWindow 类，该类中包含与界面有关的变量与函数(包括 slot 函数)。

C.game.h

声明了 Game 类，该类中包含与五子棋游戏内核部分有关的变量与函数。

D.score.h

定义了 Shape 结构体，该结构体能表示某一方向上的 5 个或 6 个棋子组成的棋型。该头文件中还声明了与评估棋局相关的函数。

4.2.2 源文件

A. main.cpp

包含 mainwindow.h

main 函数所在，显示主程序

B. mainwindow.cpp

界面源文件，包含 mainwindow.h

a. MainWindow 类的构造函数和析构函数

构造函数：对界面进行初始化；创建 Game 类并调用 initGame 函数

析构函数：删除 Game 类

b. void MainWindow:: initGame()

对游戏进行初始化，调用 Game 类中的 startGame 函数，开启鼠标跟踪并更新 UI。也是 TOOL 下拉菜单栏中的“reset”选项的槽函数。

c. void MainWindow::regret()

调用 Game 类中的 regret 函数并更新 UI。也是 TOOL 下拉菜单栏中的“regret”选项的槽函数。

d. void MainWindow:: EasyClicked()

void MainWindow::MiddleClicked()

void MainWindow::HardClicked()

TOOL 菜单下拉栏中关于游戏模式选择的槽函数。会有消息框提示是否确认更改，若更改会清空棋局，用户可根据需要选择。

e. void MainWindow:: IntroClicked()

HELP 菜单下拉栏中关于五子棋介绍的槽函数。会有消息框介绍相关信息。

f. void MainWindow:: UsageClicked()

HELP 菜单下拉栏中关于该程序使用的槽函数。会有消息框介绍相关信息。

g. void MainWindow:: MoreClicked()

HELP 菜单下拉栏中关于更多情况的槽函数。会有消息框介绍相关信息。

h. void MainWindow::closeEvent(QCloseEvent *event)

void MainWindow::paintEvent(QPaintEvent *event)

void MainWindow::mouseMoveEvent(QMouseEvent *event)

void MainWindow::mouseReleaseEvent(QMouseEvent *event)

对 Qt 中已定义过的函数的重载。详见 3.3.2。

i. void MainWindow::JudgeGame()

判断游戏是否结束。若结束则弹出相应对话框并重置游戏状态。

C. head.cpp

包含 head.h

a. Point 类的构造函数的定义

b. bool range(int row,int col)

判断某一点是否在棋盘范围内

D. game.cpp

游戏内核源文件，包含 game.h，定义了全局变量 nextX=-1，nextY=-1 来记录下一个落子的位置和总的搜索层数。

a. Game 类的构造函数的定义

b. void Game::startGame()

开始游戏。对变量进行初始化，设置执棋方和游戏状态。

c. void Game::updateGameMap(int row, int col)

见 3.3.1 G。

d. std::vector<Point> Game::GetMoves()

获得当前可以移动的点的位置，并返回存储着这些点的 Point 型的 vector 中。

e. int Game::MinMax(int player, int depth, int alpha, int beta)

递归的进行 MinMax 搜索，返回当前执子方 player 的最终估值。详见 3.2。

f. void Game::actionByPerson(int row, int col)

void Game::actionByAI(int &clickRow, int &clickCol)
见 3.3.1 C D。

f. void Game::**regret**()
悔棋操作 3.3.1 F。

E. score.cpp

估值函数源文件，包含 score.h。

整体思路详见 3.2.3

a. int Evaluate(int player,std::vector<std::vector<int>> chessBoard)

调用 EvaluateOnePoint，计算 player 方的当前棋局下的得分。

b. int EvaluateOnePoint(int x, int y, int player,std::vector<std::vector<int>> gameMapVec)

调用 EvaluateOneShape 并使用 Shape 类，返回某个点的估值。

c. int EvaluateOneShape(int player, Shape sp)

返回 sp 棋型的估值得分。

d. bool JudgeNeighbor(int x, int y,std::vector<std::vector<int>> gameMapVec)

判断坐标(x, y)周围有没有棋子。

装

订

线

4.3 实验结果展示

4.3.1 UI 界面展示

简单起见，只展示一组结果。完整界面逻辑和展示详见 3.1.2

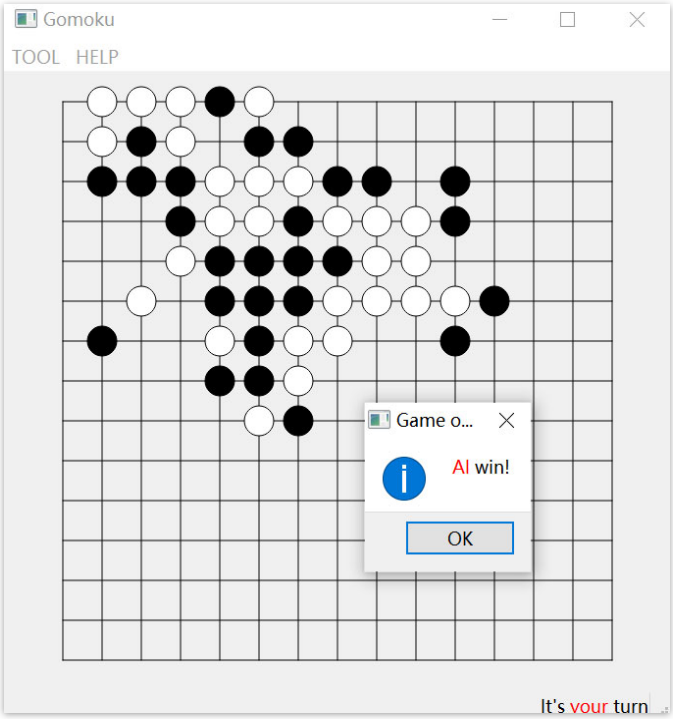

图 4.3.1 结果展示

4.3.2 结果分析

AI 的棋力随着搜索深度的增加而提高，但与此同时运算的时间也会显著增加。

5 总结

参考文献

[1] Jasmin Blanchette, Mark Summerfield. C++ GUI Qt 4 编程. 北京:电子工业出版社,2018，2000(2):5-8.

[2] Gomoku - Wikipedia

[3] Stuart Russell, Peter Norvig. Artificial intelligence: a modern approach 北京:人民邮电出版社, 2002

装

订

线