

C++ 程序课程设计：五子棋游戏

24-1 班第 2 组

2025 年 6 月 11 日

学号	姓名
2410120028	侯成霖
2410120036	陶勇豪

目录

1	程序说明	2
2	程序分析	2
2.1	需求分析	2
2.2	设计挑战	2
3	程序设计	2
3.1	系统架构	2
3.2	核心类设计	2
3.3	AI 评估算法	3
3.4	胜率统计算法	3
4	程序实现	4
4.1	开发环境	4
4.2	关键实现	4
4.2.1	棋盘绘制	4
4.2.2	AI 决策流程	4
4.2.3	胜率计算	6
5	程序测试	7
5.1	测试用例	7
5.2	界面测试	7
6	任务分工	7
7	经验总结	8
7.1	遇到的问题与解决	8
7.2	改进方向	8
7.3	总结体会	8

1 程序说明

本程序是一个基于 Qt 框架的五子棋游戏，支持人机对战和双人对战两种模式。主要功能包括：

- 15×15 标准五子棋棋盘
- 智能 AI 对手（防守优先策略）
- 胜负判定与游戏结束处理
- 胜率统计系统
- 美观的图形界面（棋子渐变效果、悬停提示）
- 游戏模式选择（人机对战/双人对战）

程序采用面向对象设计，核心类包括 GomokuBoard（棋盘逻辑）、AiPlayer（AI 算法）、GameWindow（游戏界面）和 Rating（胜率统计）。

2 程序分析

2.1 需求分析

- 功能需求：**实现五子棋基本规则、AI 对战、胜负判定、历史胜率统计
- 性能需求：**AI 响应时间 <500ms，界面刷新流畅
- 用户体验：**直观的棋盘布局、棋子悬停提示、游戏状态反馈

2.2 设计挑战

- AI 算法需平衡进攻与防守
- 实现高效的五子连珠检测
- 保证界面响应性与美观性
- 胜率数据的持久化存储

3 程序设计

3.1 系统架构

图 1

3.2 核心类设计

- GomokuBoard：**棋盘状态管理、落子逻辑、胜负判定
- AiPlayer：**位置评估算法、最优落子点计算
- GameWindow：**界面绘制、事件处理、游戏流程控制
- Rating：**胜率计算、数据持久化

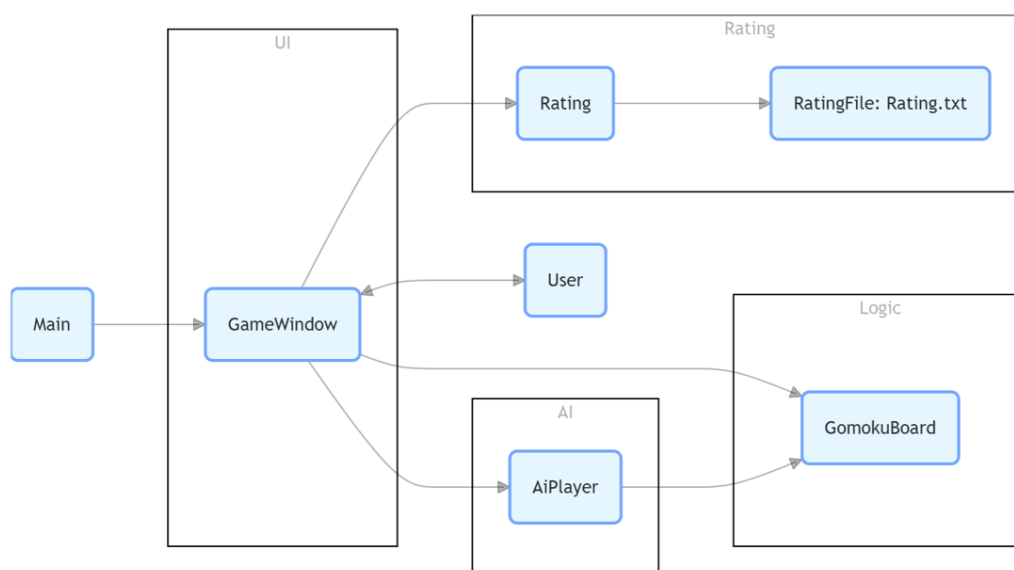


图 1: 系统架构图

3.3 AI 评估算法

采用基于规则的评估函数，考虑因素：

1. **防守优先级**：优先阻断对手四连（100000 分）
2. **进攻机会**：构建自身连珠（优先构建五连 999999 分）
3. **位置价值**：中心区域权重更高

评分权重矩阵：

$$\text{Score} = \sum_{\text{方向}} \begin{cases} 100000 & \text{对手 4 连} \\ 10000 & \text{对手活 3} \\ 5000 & \text{AI 活 3/4 连} \\ 500 & \text{AI 活 2} \end{cases} + 10 \times (n - |x - c| - |y - c|)$$

3.4 胜率统计算法

$$\text{胜率} = \frac{Y}{Y + N} \times 100\%$$

- Y：从 Rating.txt 读取的”Y” 行数
- N：从 Rating.txt 读取的”N” 行数

4 程序实现

4.1 开发环境

- 操作系统: Linux
- 编译器: g++
- GUI 框架: Qt 5.15
- 构建工具: Makefile

4.2 关键实现

4.2.1 棋盘绘制

使用 QPainter 实现高质量棋盘渲染:

```
1 void GameWindow::drawBoard(QPainter &painter) {
2
3     painter.fillRect(..., QBrush(QColor(210, 180, 140, 180)));
4
5     for (int i = 0; i < m_board.size(); ++i) {
6         painter.drawLine(...);
7         painter.drawLine(...);
8     }
9
10    painter.drawEllipse(QPoint(x, y), 4, 4);
11 }
```

4.2.2 棋盘底层逻辑

gomokuboard.h

```
1 #ifndef GOMOKU_BOARD_H
2 #define GOMOKU_BOARD_H
3
4 #include <QVector>
5
6 class GomokuBoard {
7 public:
8     enum Piece { Empty, Black, White };
9
10    GomokuBoard(int size = 15);
11    virtual ~GomokuBoard() = default;
12
13    bool placePiece(int x, int y, Piece piece);
14    bool checkWin(int x, int y) const;
15    void reset();
16
17    int size() const { return m_size; }
18    Piece pieceAt(int x, int y) const { return m_board[x][y]; }
19
20 protected:
21     QVector<QVector<Piece>> m_board;
22     int m_size;
23 };
```

```
24
25 #endif
```

gomokuboard.cpp

```
1 #include "gomokuboard.h"
2
3 GomokuBoard::GomokuBoard(int size) : m_size(size) {
4     reset();
5 }
6
7 void GomokuBoard::reset() {
8     m_board.resize(m_size);
9     for (auto &row : m_board) {
10         row.resize(m_size);
11         row.fill(Empty);
12     }
13 }
14
15 bool GomokuBoard::placePiece(int x, int y, Piece piece) {
16     if (x < 0 || x >= m_size || y < 0 || y >= m_size || m_board[x][y] != Empty) {
17         return false;
18     }
19     m_board[x][y] = piece;
20     return true;
21 }
22
23 bool GomokuBoard::checkWin(int x, int y) const {
24     Piece currentPiece = m_board[x][y];
25     if (currentPiece == Empty) {
26         return false;
27     }
28     const int directions[4][2] = {{1,0}, {0,1}, {1,1}, {1,-1}};
29
30     for (auto &dir : directions) {
31         int Count = 1;
32         int dx = dir[0], dy = dir[1];
33
34         for (int i = 1; i < 5; ++i) {
35             int nx = x + dx * i, ny = y + dy * i;
36             if (nx < 0 || nx >= size() || ny < 0 || ny >= size()) break;
37             if (pieceAt(nx, ny) == currentPiece) {
38                 Count++;
39             } else {
40                 break;
41             }
42         }
43
44         for (int i = 1; i < 5; ++i) {
45             int nx = x - dx * i, ny = y - dy * i;
46             if (nx < 0 || nx >= size() || ny < 0 || ny >= size()) break;
47             if (pieceAt(nx, ny) == currentPiece) {
48                 Count++;
49             } else {
50                 break;
51             }
52         }
53         if(Count >= 5) return true;
54     }
55     return false;
```

56 }

4.2.3 AI 决策流程

1. 遍历所有空位
2. 计算每个位置的攻防评分
3. 选择最高分位置
4. 随机选择最优位置（防预测）

```

1 QPoint AiPlayer::calculateAIMove(GomokuBoard m_board) {
2     int bestScore = -1;
3     QVector<QPoint> bestMoves;
4
5     for (int x = 0; x < m_board.size(); ++x) {
6         for (int y = 0; y < m_board.size(); ++y) {
7             if (m_board.pieceAt(x, y) == GomokuBoard::Empty) {
8                 int score = evaluatePosition(x, y, GomokuBoard::White, m_board);
9                 if (score > bestScore) {
10                     bestScore = score;
11                     bestMoves.clear();
12                     bestMoves.append(QPoint(x, y));
13                 } else if (score == bestScore) {
14                     bestMoves.append(QPoint(x, y));
15                 }
16             }
17         }
18     }
19
20     if (!bestMoves.isEmpty()) {
21         int randomIndex = QRandomGenerator::global()->bounded(bestMoves.size());
22         return bestMoves[randomIndex];
23     }
24     return QPoint(-1, -1);
25 }
26
27 int AiPlayer::evaluatePosition(int x, int y, GomokuBoard::Piece aiPiece, GomokuBoard m_board) {
28     GomokuBoard::Piece humanPiece = (aiPiece == GomokuBoard::White) ? GomokuBoard::Black : GomokuBoard::White;
29
30     int score = 0;
31
32     const int directions[4][2] = {{1,0}, {0,1}, {1,1}, {1,-1}};
33
34     for (auto &dir : directions) {
35         int dx = dir[0], dy = dir[1];
36         int aiCount = 1, humanCount = 1;
37         bool aiBlocked = false, humanBlocked = false;
38
39         for (int i = 1; i < 5; ++i) {
40             int nx = x + dx * i, ny = y + dy * i;
41             if (nx < 0 || nx >= m_board.size() || ny < 0 || ny >= m_board.size()) break;
42             if (m_board.pieceAt(nx, ny) == aiPiece) {
43                 aiCount++;
44             } else {
45                 if (m_board.pieceAt(nx, ny) == humanPiece) aiBlocked = true;
46                 break;
47             }
48         }
49     }
50 }

```

```

46     }
47
48     for (int i = 1; i < 5; ++i) {
49         int nx = x - dx * i, ny = y - dy * i;
50         if (nx < 0 || nx >= m_board.size() || ny < 0 || ny >= m_board.size()) break;
51         if (m_board.pieceAt(nx, ny) == aiPiece) {
52             aiCount++;
53         } else {
54             if (m_board.pieceAt(nx, ny) == humanPiece) aiBlocked = true;
55             break;
56         }
57     }
58
59     for (int i = 1; i < 5; ++i) {
60         int nx = x + dx * i, ny = y + dy * i;
61         if (nx < 0 || nx >= m_board.size() || ny < 0 || ny >= m_board.size()) break;
62         if (m_board.pieceAt(nx, ny) == humanPiece) {
63             humanCount++;
64         } else {
65             if (m_board.pieceAt(nx, ny) == aiPiece) humanBlocked = true;
66             break;
67         }
68     }
69
70     for (int i = 1; i < 5; ++i) {
71         int nx = x - dx * i, ny = y - dy * i;
72         if (nx < 0 || nx >= m_board.size() || ny < 0 || ny >= m_board.size()) break;
73         if (m_board.pieceAt(nx, ny) == humanPiece) {
74             humanCount++;
75         } else {
76             if (m_board.pieceAt(nx, ny) == aiPiece) humanBlocked = true;
77             break;
78         }
79     }
80
81
82     if (humanCount >= 4) score += 100000;
83     else if (humanCount == 3 && !humanBlocked) score += 10000;
84     else if (humanCount == 2 && !humanBlocked) score += 1000;
85
86     if (aiCount == 5) score += 999999;
87     else if (aiCount >= 4) score += 5000;
88     else if (aiCount == 3 && !aiBlocked) score += 5000;
89     else if (aiCount == 2 && !aiBlocked) score += 500;
90 }
91
92 int center = m_board.size() / 2;
93 int distanceToCenter = std::abs(x - center) + std::abs(y - center);
94 score += (m_board.size() - distanceToCenter) * 10;
95
96 return score;
97 }

```

4.2.4 胜率计算

```

1 Rating::Rating(): rating(0), Y(0), N(0){
2     std::string filename = "Rating.txt";
3     std::ifstream file(filename);

```

```
4  if (!file) {
5      std::ofstream createFile(filename);
6      if (createFile) {
7          createFile.close();
8          file.open(filename);
9      }
10 }
11 std::string line;
12 while (std::getline(file, line)) {
13     if(line[0] == 'Y')Y++;
14     if(line[0] == 'N')N++;
15 }
16 file.close();
17 if (N == 0 && Y == 0) {
18     rating = 0;
19     message = QString("无信息");
20     return;
21 } else if (Y == 0) {
22     rating = 0;
23 } else if (N == 0) {
24     rating = 100;
25 } else {
26     rating = Y / (Y + N) * 100;
27 }
28 message = QString("您的人机对战胜率为 %1 %").arg(rating, 0, 'f', 2);
29 }
30
31 void Rating::ShowRating(){
32     QMessageBox msgBox;
33     msgBox.setWindowTitle("胜率");
34     msgBox.setText(QString("%1\nYes 确认, No 清空胜率").arg(message));
35     msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);
36     msgBox.setDefaultButton(QMessageBox::Yes);
37     int ret = msgBox.exec();
38     if (ret == QMessageBox::No) {
39         std::ofstream file("Rating.txt", std::ios::trunc);
40         if (!file) {
41             return;
42         }
43         file.close();
44     }
45 }
```

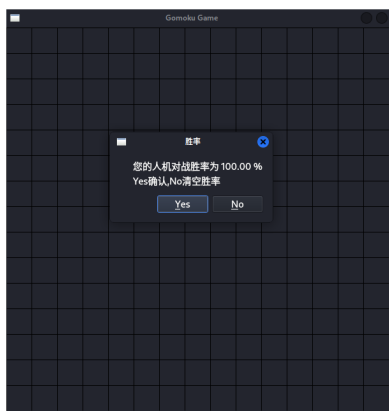
5 程序测试

5.1 测试用例

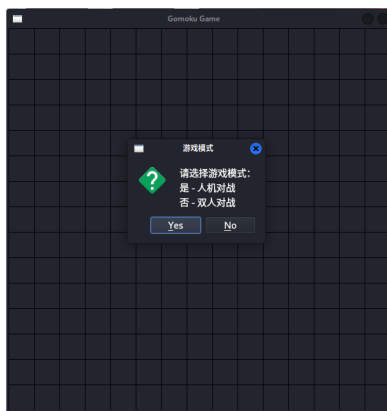
测试项	预期结果	实际结果
人机对战-玩家胜利	胜率记录增加	符合
五连珠检测	正确识别所有方向	通过
边界落子	无崩溃	通过

表 1: 测试结果

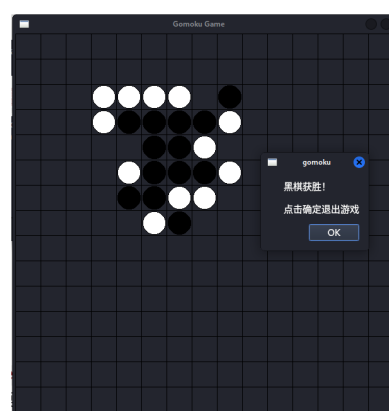
5.2 界面测试



(a) 图 1



(b) 图 2



(c) 图 3

图 2: 测试图例

6 任务分工

- **2410120028 侯成霖**: 负责底层逻辑, 算法, 界面开发 (GomokuBoard, AiPlayer, GameWindow), Latex 文档贡献度 60%
- **2410120036 陶永豪**: 负责文件处理, 界面开发 (GameWindow, Rating), PPT 贡献度 40%

7 经验总结

7.1 遇到的问题与解决

- **AI 只下左上角**: 调整分值评估, 越在中间的评分越高
- **边界检测错误**: 增加边界检查条件
- **显示程序内存溢出**: 调整大小加入检查

7.2 改进方向

- 增加难度级别选择
- 实现悔棋功能
- 添加音效和动画
- 优化 AI 算法 (Minimax+Alpha-Beta 剪枝)

7.3 总结体会

通过本项目, 我们深入掌握了:

- Qt 框架的图形编程技术

- 游戏 AI 设计原则
- 面向对象设计模式
- 团队协作开发流程

五子棋虽规则简单，但在实现过程中涉及到算法优化、用户体验、代码架构等多方面挑战，是一次宝贵的开发经验。