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	Gazettm
2410120036	pajiii

目录

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）
- 悔棋记录功能

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、悔棋
- **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
- 记录棋盘数据用于悔棋

3 程序设计

3.1 系统架构

3.1.1 逻辑结构图

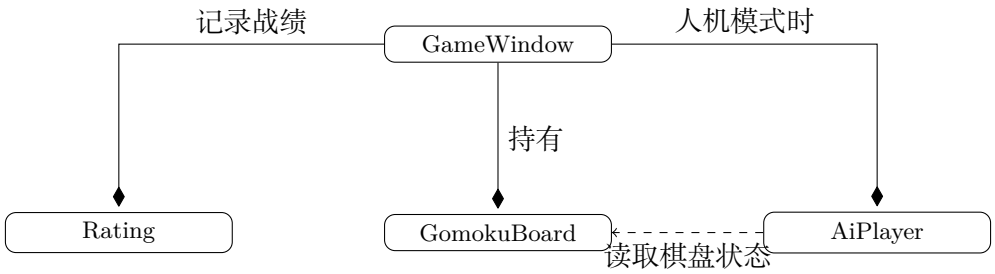


图 1: 五子棋游戏逻辑结构图

3.1.2 数据流图

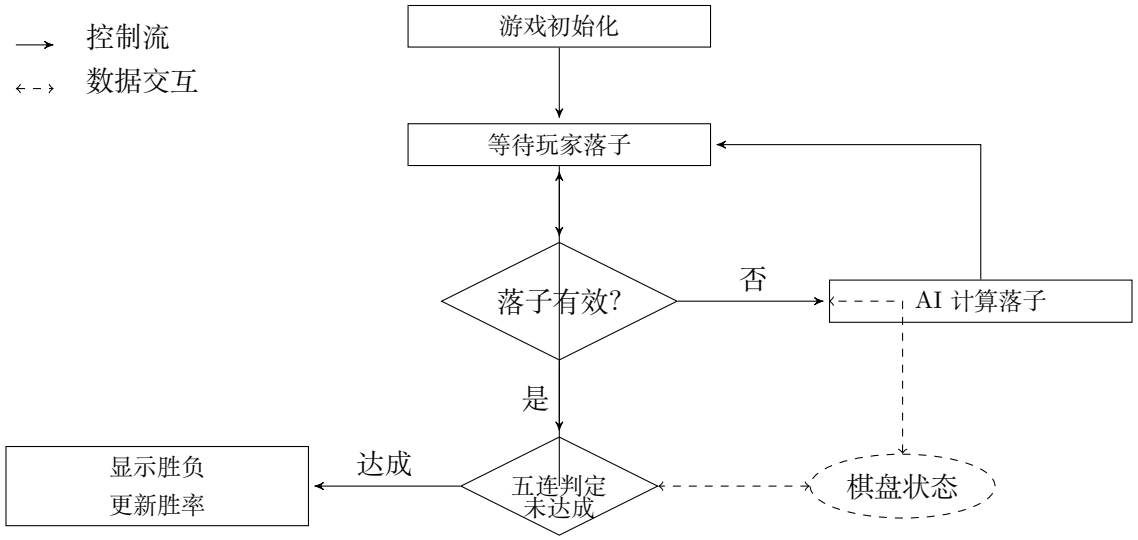


图 2: 五子棋游戏数据流图

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

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 使用头文件介绍

- QVector：QT 中用于生成动态数组的类，可以构建棋盘的基本逻辑。
- QPoint：QT 中提供的一个二维坐标点类，用于生成五子棋中的棋子。
- QTimer：用于计算 AI 落子时间并控制在 500ms 以内。
- Qpainter：QT 的绘图工具类，用于绘制棋盘，棋子，悬停提示，实现颜色渐变和抗锯齿效果。
- Qpixmap：QT 的图片导入类，用于设置棋盘背景图片。
- QMouseEvent：处理鼠标的移动，悬停，以及点击事件。
- QMessageBox：弹出消息提示框，比如选择游戏模式。
- QElapsedTimer：Qt 的高精度计时器类。

- QApplication：QT 中应用程序的核心类，实现全局应用程序控制和游戏循环。
- fstream：文件操作，存储胜负情况计算胜率。
- QLinearGradient：QT 中实现颜色线性渐变的类，增加棋子质感。
- QKeyEvent：处理键盘输入信号，本项目用于 ctrl+z 悔棋功能。
- QRandomGenerator: 用于生成范围内随机数，防止预测 AI 下一步。

4.3 关键实现

4.3.1 棋盘绘制

使用 QPainter 实现高质量棋盘渲染：

```
void GameWindow::drawBoard(QPainter &painter) {
    painter.fillRect(..., QBrush(QColor(210, 180, 140, 180)));
    for (int i = 0; i < m_board.size(); ++i) {
        painter.drawLine(...);
        painter.drawLine(...);
    }
    painter.drawEllipse(QPoint(x, y), 4, 4);
}
```

4.3.2 UI 设计及键鼠交互

gamewindow.h

```
#ifndef GAME_WINDOW_H
#define GAME_WINDOW_H

#include <QMainWindow>
#include <QMouseEvent>
#include <QMessageBox>
#include "gomokuboard.h"
#include "aiplayer.h"
#include "Rating.h"
#include <QTimer>
#include <QApplication>
#include <QRandomGenerator>
#include <QPixmap>
#include <cmath>

class GameWindow : public QMainWindow {
    Q_OBJECT

public:
    enum GameMode { HumanVsHuman, HumanVsAI };
    explicit GameWindow(QWidget *parent = nullptr);
    ~GameWindow() override = default;
    void ShowWinner(GomokuBoard::Piece winner);
    void exitGame();
    void WriteRatingY();
    void WriteRatingN();
  void undoLastMove();

protected:
```

```
    void paintEvent(QPaintEvent *event) override;
    void mousePressEvent(QMouseEvent *event) override;
    void mouseMoveEvent(QMouseEvent *event) override;
    void leaveEvent(QEvent *event) override;
  void keyPressEvent(QKeyEvent *event) override;

private:
    GomokuBoard m_board;
    GomokuBoard::Piece m_currentPiece;
    GameMode m_gameMode;
    AiPlayer m_aiplayer;
    Rating rating;
    QPoint m_hoverPos;
  bool isAIpending = false;

    void drawBoard(QPainter &painter);
    void drawPieces(QPainter &painter);
    void drawHoverIndicator(QPainter &painter);
};

#endif
```

gamewindow.cpp

```
#include "gamewindow.h"
#include <QPainter>
#include <QMouseEvent>
#include <QMessageBox>
#include <QElapsedTimer>
#include <QApplication>
#include <fstream>
#include <QLinearGradient>
#include <QRadialGradient>
#include <QKeyEvent>

GameWindow::GameWindow(QWidget *parent) :
    QMainWindow(parent),
    m_board(15),
    m_currentPiece(GomokuBoard::Black),
    m_hoverPos(-1, -1) {
    setWindowTitle("Gomoku Game");
    setFixedSize(640, 640);
    setMouseTracking(true);
    show();
    rating.ShowRating();

    QMessageBox::StandardButton reply = QMessageBox::question(
        this,
        "游戏模式 ",
        "Ctrl+z可以悔棋\n请选择游戏模式: \n是 - 人机对战\n否 - 双人对战",
        QMessageBox::Yes | QMessageBox::No
    );
    m_gameMode = (reply == QMessageBox::Yes) ? HumanVsAI : HumanVsHuman;

}
void GameWindow::ShowWinner(GomokuBoard::Piece winner) {
    update();
    if (winner == GomokuBoard::Black) {
        if(m_gameMode == HumanVsAI){
            WriteRatingY();
```

```
        }

        QMessageBox msgBox;
        msgBox.setText("黑棋获胜！");
        msgBox.setInformativeText("点击确定退出游戏");
        msgBox.setStandardButtons(QMessageBox::Ok);
        msgBox.setDefaultButton(QMessageBox::Ok);

        connect(&msgBox, &QMessageBox::finished, this, &GameWindow::exitGame);

        msgBox.exec();
       } else if (winner == GomokuBoard::White) {
        if(m_gameMode == HumanVsAI){
            WriteRatingN();
        }

        QMessageBox msgBox;
        msgBox.setText("白棋获胜！");
        msgBox.setInformativeText("点击确定退出游戏");
        msgBox.setStandardButtons(QMessageBox::Ok);
        msgBox.setDefaultButton(QMessageBox::Ok);

        connect(&msgBox, &QMessageBox::finished, this, &GameWindow::exitGame);

        msgBox.exec();
    }
}
void GameWindow::paintEvent(QPaintEvent *event) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);
    painter.drawPixmap(rect(), QPixmap("baka.png"));
    drawBoard(painter);
    drawPieces(painter);
    drawHoverIndicator(painter);
}
void GameWindow::mousePressEvent(QMouseEvent *event) {
    if (m_gameMode == HumanVsAI && m_currentPiece != GomokuBoard::Black) {
        return;
    }

    int gridSize = width() / (m_board.size() + 1);
    int margin = gridSize;

    int x = qRound(static_cast<float>(event->x() - margin) / gridSize);
    int y = qRound(static_cast<float>(event->y() - margin) / gridSize);

  if (x < 0 || x >= m_board.size() || y < 0 || y >= m_board.size()) {
        return;
    }

    if (m_board.placePiece(x, y, m_currentPiece)) {
        if (m_board.checkWin(x, y)) {
            ShowWinner(m_currentPiece);
        } else {

            if (m_gameMode == HumanVsAI) {
        isAIpending = true;
                m_currentPiece = GomokuBoard::White;

```

```
                QTimer::singleShot(500, this, [this]() {

                    QPoint aiMove = m_aiplayer.calculateAIMove(m_board);
                    if (aiMove.x() != -1) {
                        m_board.placePiece(aiMove.x(), aiMove.y(), GomokuBoard::White);
                        if (m_board.checkWin(aiMove.x(), aiMove.y())) {
                            ShowWinner(GomokuBoard::White);
                        } else {
                            m_currentPiece = GomokuBoard::Black;
                        }
                        update();
                        isAIpending = false;
                    }
                });
            } else {
                m_currentPiece = (m_currentPiece == GomokuBoard::Black) ? GomokuBoard::White : GomokuBoard
    ::Black;
            }
            update();
        }
    }
}

void GameWindow::mouseMoveEvent(QMouseEvent *event) {
    int gridSize = width() / (m_board.size() + 1);
    int margin = gridSize;

    int x = qRound(static_cast<float>(event->x() - margin) / gridSize);
    int y = qRound(static_cast<float>(event->y() - margin) / gridSize);

    if (x >= 0 && x < m_board.size() &&
        y >= 0 && y < m_board.size() &&
        m_board.pieceAt(x, y) == GomokuBoard::Empty) {

        if (m_hoverPos != QPoint(x, y)) {
            m_hoverPos = QPoint(x, y);
            update();
        }
    } else {

        if (m_hoverPos != QPoint(-1, -1)) {
            m_hoverPos = QPoint(-1, -1);
            update();
        }
    }
}
void GameWindow::leaveEvent(QEvent *event) {
    Q_UNUSED(event);

    if (m_hoverPos != QPoint(-1, -1)) {
        m_hoverPos = QPoint(-1, -1);
        update();
    }
}

void GameWindow::keyPressEvent(QKeyEvent *event) {
  if(isAIpending){
    return;
  }
```

```
  if (event->key() == Qt::Key_Z && event->modifiers() & Qt::ControlModifier) {
    undoLastMove();
  }
  QMainWindow::keyPressEvent(event);
}

void GameWindow::drawBoard(QPainter &painter) {
    int gridSize = width() / (m_board.size() + 1);
    int margin = gridSize;

    int boardPixels = gridSize * (m_board.size() - 1);

    painter.fillRect(margin, margin,
                     boardPixels,
                     boardPixels,
                     QBrush(QColor(210, 180, 140, 180)));

    painter.setPen(QPen(QColor(13, 102, 171, 150), 4));
    painter.drawRect(margin, margin,
                     boardPixels,
                     boardPixels);

    painter.setPen(QPen(QColor(0, 0, 0), 1));
    for (int i = 0; i < m_board.size(); ++i) {
        int pos = margin + i * gridSize;

        painter.drawLine(margin, pos,
                         margin + boardPixels,
                         pos);

        painter.drawLine(pos, margin,
                         pos,
                         margin + boardPixels);
    }

    painter.setBrush(Qt::black);
    int starPoints[5][2] = {
        {3, 3}, {3, 11}, {7, 7}, {11, 3}, {11, 11}
    };

    for (auto &point : starPoints) {
        int x = margin + point[0] * gridSize;
        int y = margin + point[1] * gridSize;
        painter.drawEllipse(QPoint(x, y), 4, 4);
    }
}
void GameWindow::drawPieces(QPainter &painter) {
    int gridSize = width() / (m_board.size() + 1);
    int margin = gridSize;
    int pieceRadius = gridSize / 2 - 2;

    for (int x = 0; x < m_board.size(); ++x) {
        for (int y = 0; y < m_board.size(); ++y) {
            if (m_board.pieceAt(x, y) != GomokuBoard::Empty) {

                int centerX = margin + x * gridSize;
                int centerY = margin + y * gridSize;

                if (m_board.pieceAt(x, y) == GomokuBoard::Black) {
```

```
213
214                     QRadialGradient gradient(centerX, centerY, pieceRadius,
215                                              centerX - pieceRadius/3, centerY - pieceRadius/3);
216                     gradient.setColorAt(0, QColor(60, 60, 60));
217                     gradient.setColorAt(1, Qt::black);
218                     painter.setBrush(gradient);
219                 } else {
220
221                     QRadialGradient gradient(centerX, centerY, pieceRadius,
222                                              centerX - pieceRadius/3, centerY - pieceRadius/3);
223                     gradient.setColorAt(0, Qt::white);
224                     gradient.setColorAt(1, QColor(220, 220, 220));
225                     painter.setBrush(gradient);
226                 }
227                 painter.setPen(QPen(Qt::black, 1));
228                 painter.drawEllipse(QPoint(centerX, centerY), pieceRadius, pieceRadius);
229             }
230         }
231     }
232 }
233 void GameWindow::drawHoverIndicator(QPainter &painter) {
234
235     if (m_hoverPos.x() >= 0 && m_hoverPos.y() >= 0 &&
236         m_board.pieceAt(m_hoverPos.x(), m_hoverPos.y()) == GomokuBoard::Empty) {
237         int gridSize = width() / (m_board.size() + 1);
238         int margin = gridSize;
239
240         int centerX = margin + m_hoverPos.x() * gridSize;
241         int centerY = margin + m_hoverPos.y() * gridSize;
242         int pieceRadius = gridSize / 2 - 2;
243
244         painter.setPen(QPen(QColor(255, 97, 0, 250), 5));
245         painter.setBrush(Qt::NoBrush);
246
247         painter.drawEllipse(QPoint(centerX, centerY), pieceRadius + 2, pieceRadius + 2);
248     }
249 }
250 void GameWindow::exitGame() {
251     QApplication::quit();
252 }
253 void GameWindow::WriteRatingY(){
254     std::string filename = "Rating.txt";
255     std::string content = "Y";
256     std::ofstream file(filename, std::ios::app);
257     file << content << std::endl;
258     file.close();
259 }
260 void GameWindow::WriteRatingN(){
261     std::string filename = "Rating.txt";
262     std::string content = "N";
263     std::ofstream file(filename, std::ios::app);
264     file << content << std::endl;
265     file.close();
266 }
267 void GameWindow::undoLastMove() {
268   if (m_gameMode == HumanVsHuman) {
269     if (m_board.undoMove()) {
270       m_currentPiece = (m_currentPiece == GomokuBoard::Black)
271       ? GomokuBoard::White : GomokuBoard::Black;
```

```
272       update();
273     }
274   }
275   else if (m_gameMode == HumanVsAI) {
276     bool undoSuccess = false;
277     if (m_board.getMoves().size() >= 2) {
278       if (m_board.undoMove()) {
279         if (m_board.undoMove()) {
280           undoSuccess = true;
281         }
282       }
283     }
284     else if (!m_board.getMoves().empty()) {
285       if (m_board.undoMove()) {
286         undoSuccess = true;
287       }
288     }
289     if (undoSuccess) {
290       m_currentPiece = GomokuBoard::Black;
291       update();
292     }
293   }
294 }
```

- **gamewindow 类函数介绍**

1. `ShowWinner(GomokuBoard::Piece winner)`：处理游戏结束逻辑，显示胜负弹窗。如果是在人机模式，则调用 `WriteRatingY/N()` 记录结果到”Rating.txt” 文件。
2. `exitGame()`：调用 `QApplication::quit()` 退出游戏程序。
3. `paintEvent(QPaintEvent *event)`：绘制游戏界面（棋盘、棋子、悬停指示）。
4. `mousePressEvent(QMouseEvent *event)`：处理玩家落子行为，人机模式下触发 AI 响应（遍历棋盘计算最优步并落子）。
5. `mouseMoveEvent(QMouseEvent *event)`：显示鼠标悬停位置的高亮圆环指示。
6. `leaveEvent(QEvent *event)`：鼠标离开窗口时删除掉悬停指示。
7. `keyPressEvent(QKeyEvent *event)`：监听 `Ctrl+Z` 实现悔棋功能。
8. `undoLastMove()`：撤回上一步落子，人机模式需撤回两步。
9. `drawBoard(QPainter &painter)`：绘制 15x15 棋盘网格及星位标记。
10. `drawPieces(QPainter &painter)`：渲染所有棋子，使用渐变效果，增强立体感。
11. `drawHoverIndicator(QPainter &painter)`：在悬停位置绘制橙色高亮圆环。
12. `WriteRatingY()`：向文件追加字符 `Y`，记录玩家胜利。
13. `WriteRatingN()`：向文件追加字符 `N`，记录 AI 胜利。
14. `GameWindow(QWidget *parent)`：该类的构造函数，实现初始化游戏窗口和模式选择。

4.3.3 棋盘底层逻辑

gomokuboard.h

```h
#ifndef GOMOKU_BOARD_H
#define GOMOKU_BOARD_H

#include <QVector>

class GomokuBoard {
public:
    enum Piece { Empty, Black, White };

    GomokuBoard(int size = 15);
    virtual ~GomokuBoard() = default;

    bool placePiece(int x, int y, Piece piece);
    bool checkWin(int x, int y) const;
    void reset();

    int size() const { return m_size; }
    Piece pieceAt(int x, int y) const { return m_board[x][y]; }

protected:
    QVector<QVector<Piece>> m_board;
    int m_size;
};

#endif
```

gomokuboard.cpp

```cpp
#include "gomokuboard.h"

GomokuBoard::GomokuBoard(int size) : m_size(size) {
  reset();
}

void GomokuBoard::reset() {
  m_board.resize(m_size);
  for (auto &row : m_board) {
    row.resize(m_size);
    row.fill(Empty);
  }
}

bool GomokuBoard::placePiece(int x, int y, Piece piece) {
  if (x < 0 || x >= m_size || y < 0 || y >= m_size || m_board[x][y] != Empty) {
    return false;
  }
  m_board[x][y] = piece;
  return true;
}

bool GomokuBoard::checkWin(int x, int y) const {
    Piece currentPiece = m_board[x][y];
    if (currentPiece == Empty) {
        return false;
    }
    const int directions[4][2] = {{1,0}, {0,1}, {1,1}, {1,-1}};

```

```
    for (auto &dir : directions) {
        int Count = 1;
        int dx = dir[0], dy = dir[1];

        for (int i = 1; i < 5; ++i) {
            int nx = x + dx * i, ny = y + dy * i;
            if (nx < 0 || nx >= size() || ny < 0 || ny >= size()) break;
            if (pieceAt(nx, ny) == currentPiece) {
                Count++;
            } else {
                break;
            }
        }

        for (int i = 1; i < 5; ++i) {
            int nx = x - dx * i, ny = y - dy * i;
            if (nx < 0 || nx >= size() || ny < 0 || ny >= size()) break;
            if (pieceAt(nx, ny) == currentPiece) {
                Count++;
            } else
                break;
            }
        }
        if(Count >= 5) return true;
    }
    return false;
}
```

- **gomokuboard 类函数介绍**

1. `GomokuBoard(int size)`：构造函数，初始化指定大小的棋盘（默认 15x15）。
2. `reset()`：重置棋盘状态，清空所有棋子及落子记录。
3. `placePiece(int x, int y, Piece piece)`：在合法位置放置棋子，并记录坐标。
4. `checkWin(int x, int y)`：检查当前位置是否形成五连棋，检测该位置坐标八个方向。
5. `undoMove()`：撤销最后一次落子，更新棋盘和落子记录。
6. `pieceAt(int x, int y)`：返回指定位置的棋子状态（Empty/Black/White）。
7. `size()`：获取棋盘尺寸（默认 15）。
8. `getMoves()`：返回所有落子记录的常量引用。

4.3.4 AI 决策流程

aiplayer.h

```
#ifndef AIPLAYER_H
#define AIPLAYER_H
#include "gomokuboard.h"
#include <QPoint>
class AiPlayer{
public:
  AiPlayer(){}
  int evaluatePosition(int x, int y, GomokuBoard::Piece aiPiece,GomokuBoard m_board);
```

```
  QPoint calculateAIMove(GomokuBoard m_board);
};
#endif
```

aiplayer.cpp

```
#include <QRandomGenerator>
#include <cmath>
#include "aiplayer.h"
QPoint AiPlayer::calculateAIMove(GomokuBoard m_board) {
  int bestScore = -1;
  QVector<QPoint> bestMoves;

  for (int x = 0; x < m_board.size(); ++x) {
    for (int y = 0; y < m_board.size(); ++y) {
      if (m_board.pieceAt(x, y) == GomokuBoard::Empty) {
        int score = evaluatePosition(x, y, GomokuBoard::White,m_board);
        if (score > bestScore) {
          bestScore = score;
          bestMoves.clear();
          bestMoves.append(QPoint(x, y));
        } else if (score == bestScore) {
          bestMoves.append(QPoint(x, y));
        }
      }
    }
  }

  if (!bestMoves.isEmpty()) {
    int randomIndex = QRandomGenerator::global()->bounded(bestMoves.size());
    return bestMoves[randomIndex];
  }
  return QPoint(-1, -1);
}
int AiPlayer::evaluatePosition(int x, int y, GomokuBoard::Piece aiPiece,GomokuBoard m_board) {
  GomokuBoard::Piece humanPiece =
  (aiPiece == GomokuBoard::White) ? GomokuBoard::Black : GomokuBoard::White;
  int score = 0;

  const int directions[4][2] = {{1,0}, {0,1}, {1,1}, {1,-1}};

  for (auto &dir : directions) {
    int dx = dir[0], dy = dir[1];
    int aiCount = 1, humanCount = 1;
    bool aiBlocked = false, humanBlocked = false;

    for (int i = 1; i < 5; ++i) {
      int nx = x + dx * i, ny = y + dy * i;
      if (nx < 0 || nx >= m_board.size() || ny < 0 || ny >= m_board.size()) break;
      if (m_board.pieceAt(nx, ny) == aiPiece) {
        aiCount++;
      } else {
        if (m_board.pieceAt(nx, ny) == humanPiece) aiBlocked = true;
        break;
      }
    }

    for (int i = 1; i < 5; ++i) {
      int nx = x - dx * i, ny = y - dy * i;
      if (nx < 0 || nx >= m_board.size() || ny < 0 || ny >= m_board.size()) break;
```

```
55        if (m_board.pieceAt(nx, ny) == aiPiece) {
56          aiCount++;
57        } else {
58          if (m_board.pieceAt(nx, ny) == humanPiece) aiBlocked = true;
59          break;
60        }
61      }
62
63      for (int i = 1; i < 5; ++i) {
64        int nx = x + dx * i, ny = y + dy * i;
65        if (nx < 0 || nx >= m_board.size() || ny < 0 || ny >= m_board.size()) break;
66        if (m_board.pieceAt(nx, ny) == humanPiece) {
67          humanCount++;
68        } else {
69          if (m_board.pieceAt(nx, ny) == aiPiece) humanBlocked = true;
70          break;
71        }
72      }
73
74      for (int i = 1; i < 5; ++i) {
75        int nx = x - dx * i, ny = y - dy * i;
76        if (nx < 0 || nx >= m_board.size() || ny < 0 || ny >= m_board.size()) break;
77        if (m_board.pieceAt(nx, ny) == humanPiece) {
78          humanCount++;
79        } else {
80          if (m_board.pieceAt(nx, ny) == aiPiece) humanBlocked = true;
81          break;
82        }
83      }
84
85      if (humanCount >= 4) score += 10000;
86      else if (humanCount == 3 && !humanBlocked) score += 1000;
87      else if (humanCount == 2 && !humanBlocked) score += 100;
88
89      if(aiCount == 5) score += 99999;
90      else if (aiCount >= 4) score += 500;
91      else if (aiCount == 3 && !aiBlocked) score += 500;
92      else if (aiCount == 2 && !aiBlocked) score += 50;
93    }
94
95    int center = m_board.size() / 2;
96    int distanceToCenter = std::abs(x - center) + std::abs(y - center);
97    score += (m_board.size() - distanceToCenter) * 10;
98
99    return score;
100 }
```

- **aiplayer 类函数介绍**

1. `calculateAIMove(GomokuBoard m_board)`：计算 AI 最佳落子位置，遍历空位并调用评分函数，随机选择最高分位置。
2. `evaluatePosition(int x, int y, Piece aiPiece, GomokuBoard m_board)`：评估位置价值，检查四方向连子数和阻挡情况，结合中心距离计算得分。

4.3.5 胜率计算

Rating.h

```
#ifndef RATING_H
#define RATING_H
#include <QMessageBox>
#include <QString>
class Rating{
public:
  Rating();
  void ShowRating();
private:
  QString message;
  double rating;
  double Y;
  double N;
};
#endif
```

Rating.cpp

```
Rating::Rating(): rating(0), Y(0), N(0){
  std::string filename = "Rating.txt";
  std::ifstream file(filename);
  if (!file) {
    std::ofstream createFile(filename);
    if (createFile) {
      createFile.close();
      file.open(filename);
    }
  }
  std::string line;
  while (std::getline(file, line)) {
    if(line[0] == 'Y')Y++;
    if(line[0] == 'N')N++;
  }
  file.close();
  if (N == 0 && Y == 0) {
    rating = 0;
    message = QString("无信息");
    return;
  } else if (Y == 0) {
    rating = 0;
  } else if (N == 0) {
    rating = 100;
  } else {
    rating = Y / (Y + N) * 100;
  }
  message = QString("您的人机对战胜率为 %1 %").arg(rating, 0, 'f', 2);
}

void Rating::ShowRating(){
  QMessageBox msgBox;
  msgBox.setWindowTitle("胜率");
  msgBox.setText(QString("%1\nYes确认,No清空胜率").arg(message));
  msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);
  msgBox.setDefaultButton(QMessageBox::Yes);
  int ret = msgBox.exec();
  if (ret == QMessageBox::No) {
    std::ofstream file("Rating.txt", std::ios::trunc);
    if (!file) {
      return;
    }
```

```
43      file.close();
44    }
45  }
```

- **rating 类函数介绍**

1. `Rating()`：构造函数，读取 `Rating.txt` 文件内容并统计历史胜负记录（`Y/N`），计算百分比胜率。
2. `ShowRating()`：显示胜率对话框，提供清空记录的选项。当用户选择“No”时，则清空文件内容。

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	通过
落子位置检测	正确落在指定位置	通过
五连珠检测	正确识别所有方向	通过
边界落子	无崩溃	通过
人机模式悔棋	撤回两步	通过
双人模式悔棋	撤回一步	通过

表 1: 测试结果

5.2 界面测试

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 陶勇豪**：负责文件处理，界面开发（GameWindow，Rating），PPT 贡献度 40%

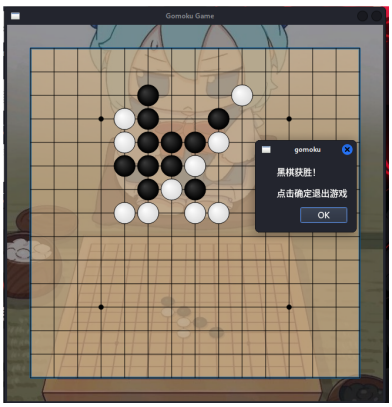
7 经验总结

7.1 遇到的问题与解决

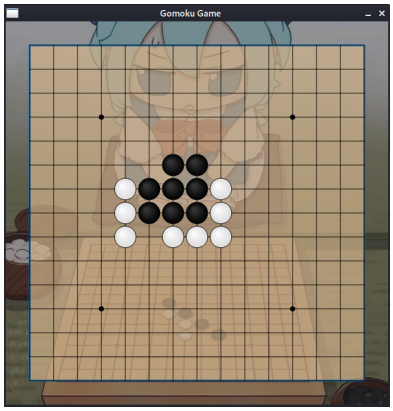
- **AI 只下左上角**：调整分值评估，越在中间的评分越高
- **边界检测错误**：增加边界检查条件
- **显示程序内存溢出**：调整大小加入检查

(a) 图 1

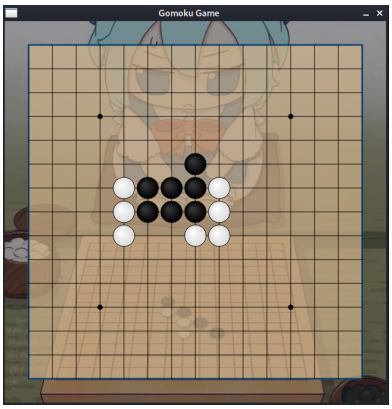
(b) 图 2



(c) 图 3



(d) 图 4（悔棋前）



(e) 图 5（悔棋后）

图 3: 测试图例

7.2 改进方向

- 增加难度级别选择
- 用栈保留对局信息
- 添加音效和动画
- 添加黑棋禁手规则，保证公平
- 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
- linux 环境编程操作
- 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。