

# 五子棋游戏技术文档

Gazettm and pajiini

2025 年 6 月 9 日

## 目录

<b>1</b>	<b>项目概述</b>	<b>2</b>
<b>2</b>	<b>类设计与实现</b>	<b>2</b>
2.1	GomokuBoard 类 . . . . .	2
2.1.1	成员变量 . . . . .	2
2.1.2	关键方法 . . . . .	2
2.2	AiPlayer 类 . . . . .	3
2.2.1	核心方法 . . . . .	3
2.3	GameWindow 类 . . . . .	3
2.3.1	游戏流程控制 . . . . .	3
2.4	Rating 类 . . . . .	4
2.4.1	核心方法 . . . . .	4
2.4.2	数据结构 . . . . .	5
2.4.3	文件存储格式 . . . . .	5
<b>3</b>	<b>关键算法</b>	<b>5</b>
3.1	AI 评估算法 . . . . .	5
3.2	胜率统计算法 . . . . .	5
<b>4</b>	<b>用户界面设计</b>	<b>6</b>
<b>5</b>	<b>测试</b>	<b>6</b>
5.1	图片展示 . . . . .	6
5.2	版本说明 . . . . .	6
<b>6</b>	<b>编译运行说明</b>	<b>7</b>
6.1	环境要求 . . . . .	7
6.2	编译步骤 . . . . .	7
6.3	文件说明 . . . . .	8
<b>7</b>	<b>分工</b>	<b>8</b>

## 1 项目概述

本项目是一个基于 Qt 框架的五子棋游戏，支持以下功能：

- 双人对战模式
- 人机对战模式（AI 玩家）
- 胜负判定与游戏结束处理
- 胜率统计与历史记录
- 图形化棋盘界面

项目采用 MVC 架构设计：

- **模型 (Model):** GomokuBoard 类管理棋盘状态
- **视图 (View):** GameWindow 类处理界面渲染
- **控制器 (Controller):** GameWindow 类处理用户输入和游戏逻辑

## 2 类设计与实现

### 2.1 GomokuBoard 类

棋盘核心逻辑，管理游戏状态。

#### 2.1.1 成员变量

- `m_size`: 棋盘尺寸（默认 15×15）
- `m_board`: 二维向量存储棋子状态

#### 2.1.2 关键方法

```
1 enum Piece { Empty, Black, White };
2 bool placePiece(int x, int y, Piece piece); // 落子
3 bool checkWin(int x, int y) const;         // 胜负判定
4 void reset();                               // 重置棋盘
```

Listing 1: gomokuboard.h

胜负判定算法：

```
1 bool GomokuBoard::checkWin(int x, int y) const {
2     const int directions[4][2] = {{1,0}, {0,1}, {1,1}, {1,-1}};
3     for (auto &dir : directions) {
4         int Count = 1;
5         // 双向检测连子数量
6         for (int i = 1; i < 5; ++i) { // 正向检测
7             int nx = x + dir[0] * i, ny = y + dir[1] * i;
8             if (nx < 0 || nx >= size() || ny < 0 || ny >= size()) break;
9             if (pieceAt(nx, ny) == currentPiece) Count++;
10            else break;
11        }
12    }
```

```

12     for (int i = 1; i < 5; ++i) { // 反向检测
13         int nx = x - dir[0] * i, ny = y - dir[1] * i;
14         if (nx < 0 || nx >= size() || ny < 0 || ny >= size()) break;
15         if (pieceAt(nx, ny) == currentPiece) Count++;
16         else break;
17     }
18     if(Count >= 5) return true; // 五连珠获胜
19 }
20 return false;
21 }

```

## 2.2 AiPlayer 类

实现 AI 玩家逻辑，包含位置评估和落子决策。

### 2.2.1 核心方法

```

1 QPoint calculateAIMove(GomokuBoard m_board); // 计算AI落子位置
2 int evaluatePosition(int x, int y, GomokuBoard::Piece aiPiece, GomokuBoard m_board); // 位置评估

```

Listing 2: aplayer.h

评估函数实现：

```

1 int AiPlayer::evaluatePosition(int x, int y,
2     GomokuBoard::Piece aiPiece, GomokuBoard m_board) {
3
4     // 防守评分（人类玩家威胁）
5     if (humanCount >= 4) score += 100000; // 阻断四连
6     else if (humanCount == 3 && !blocked) score += 10000; // 阻断活三
7
8     // 进攻评分（AI连珠）
9     if(aiCount == 5) score += 999999; // 五连绝杀
10    else if (aiCount >= 4) score += 5000; // 四连
11
12    // 中心区域加成
13    int center = m_board.size() / 2;
14    int distance = std::abs(x - center) + std::abs(y - center);
15    score += (m_board.size() - distance) * 10;
16 }

```

## 2.3 GameWindow 类

主游戏窗口，处理界面和游戏流程。

### 2.3.1 游戏流程控制

```

1 // 人机对战流程
2 void GameWindow::mousePressEvent(QMouseEvent *event) {
3     if (m_gameMode == HumanVsAI && m_currentPiece != Black)
4         return; // AI回合忽略点击
5
6     // 玩家落子
7     if (m_board.placePiece(x, y, m_currentPiece)) {
8         if (checkWin) ShowWinner(); // 胜负判定
9         else {

```

```

10         m_currentPiece = White; // 切换到AI
11         QTimer::singleShot(500, [this]() { // AI延迟落子
12             QPoint aiMove = m_aiplayer.calculateAIMove(m_board);
13             m_board.placePiece(aiMove.x(), aiMove.y(), White);
14             if (checkWin) ShowWinner();
15             else m_currentPiece = Black; // 切回玩家
16         });
17     }
18 }
19 }

```

## 2.4 Rating 类

胜率统计系统，基于文件存储历史记录。

### 2.4.1 核心方法

```

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 }
```

2.4.2 数据结构

- rating: 胜率百分比
- Y: 胜利次数
- N: 失败次数
- message: 胜率显示信息

2.4.3 文件存储格式

```
1 Y // 人类玩家获胜记录
2 N // AI 获胜记录
```

3 关键算法

3.1 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：

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

3.2 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 用户界面设计

界面组件与功能：

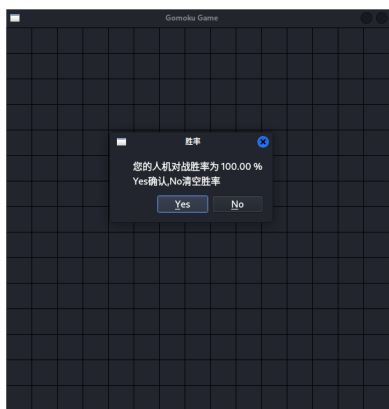
- **模式选择对话框**：游戏启动时选择人机/双人模式
- **棋盘绘制**：使用 QPainter 绘制  $15 \times 15$  网格
- **棋子渲染**：黑色实心圆（玩家），白色空心圆（AI）
- **胜率对话框**：显示历史胜率并提供清空选项
- **获胜提示**：模态对话框显示获胜方

界面操作流程：

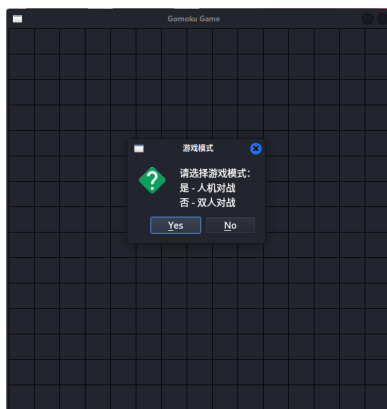
1. 启动游戏 → 选择对战模式
2. 显示历史胜率（可选择清空）
3. 玩家点击落子（人机模式下 AI 自动响应）
4. 五连珠时显示获胜对话框
5. 游戏结束记录胜率

## 5 测试

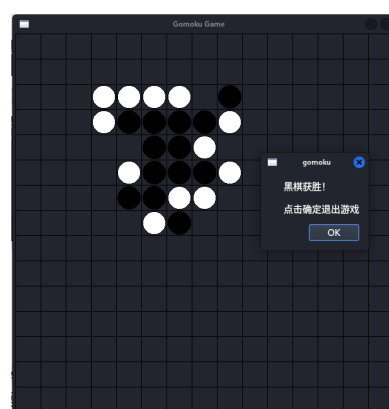
### 5.1 图片展示



(a) 图 1



(b) 图 2



(c) 图 3

图 1: 测试图例

### 5.2 版本说明

- May 14, 2025 测试
- May 24, 2025 linux 下 texmaker 测试 (使用 xelatex)

- May 27, 2025 添加 AI
- May 28, 2025 优化判定算法
- May 31, 2025 添加胜率
- Jun 1, 2025 修复 bug

## 6 编译运行说明

### 6.1 环境要求

- Qt 5.15.15
- c++ (Debian 14.2.0-19) 14.2.0
- g++ (Debian 14.2.0-19) 14.2.0

### 6.2 编译步骤

1. 编译项目 (提前写的 makefile 文件):

```

1 CXX = g++
2 CXXFLAGS = -std=c++11 -Wall -fPIC
3 QT_INCLUDE = -I/usr/include/x86_64-linux-gnu/qt5/ \
4             -I/usr/include/x86_64-linux-gnu/qt5/QtWidgets \
5             -I/usr/include/x86_64-linux-gnu/qt5/QtGui \
6             -I/usr/include/x86_64-linux-gnu/qt5/QtCore
7 QT_LIBS = -lQt5Widgets -lQt5Gui -lQt5Core
8
9 SRC = main.cpp gamewindow.cpp gomokuboard.cpp aplayer.cpp Rating.cpp
10 OBJ = $(SRC:.cpp=.o) moc_gamewindow.o # 添加 moc 生成的目标文件
11 TARGET = gomoku
12
13 all: $(TARGET)
14
15 $(TARGET): $(OBJ)
16     $(CXX) $(CXXFLAGS) $^ -o $@ $(QT_LIBS)
17
18 # 生成 moc 文件
19 moc_%.cpp: %.h
20     moc $< -o $@
21
22 # 编译 moc 文件
23 moc_%.o: moc_%.cpp
24     $(CXX) $(CXXFLAGS) $(QT_INCLUDE) -c $< -o $@
25
26 # 编译普通源文件
27 %.o: %.cpp
28     $(CXX) $(CXXFLAGS) $(QT_INCLUDE) -c $< -o $@
29
30 clean:
31     rm -f $(OBJ) $(TARGET) moc_*
```

2. 写 makefile 文件:

```
1 make
```

3. 运行程序：

```
1 ./gomoku
```

s

6.3 文件说明

文件	功能
gamewindow.cpp	主窗口和游戏流程控制
gomokuboard.cpp	棋盘状态管理和规则逻辑
aiplayer.cpp	AI 玩家决策算法
rating.cpp	胜率统计系统
main.cpp	程序入口点
Rating.txt	胜率数据存储

7 分工

Gazettm:

- gomokuboard
- gamewindows
- aiplayer
- Makefile
- LaTeX

pajiiii:

- aiplayer
- PPT
- markdown
- Rating