

武汉大学计算机学院

本科生实验报告

棋逢对手——基于 AI 棋类游戏之军棋

专 业 名 称 ： 计算机科学与技术

课 程 名 称 ： 软件构造基础

指 导 教 师 ： 祝园园

学 生 学 号 ： 2018302020183

学 生 姓 名 ： 辛林炫

二〇二〇年五月

郑 重 声 明

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名： 辛林炫

日期： 2020.05.28

摘 要

本实验为棋逢对手——基于 AI 的棋类游戏合集的军棋部分。整个军棋部分由报告人独立完成。项目主要包括离线双人对战、人机对战和在线双人对战三种不同的对战形式，并且有很多增加使用体验的辅助功能，如残局存储和读取、线上聊天、规则提醒、游戏音效及相关设置等辅助功能。

作为优秀的面向对象语言，C#不仅具有封装、继承和多态等特性，还有索引器、委托、事件等创新因素。而本次软件构造基础实验的实验目的则是复习、体会、运用在理论课上所学知识，自主设计、编写、调试一个程序，从而巩固所学知识、综合练习，提高逻辑思维能力和解决问题能力。

实验设计过程中采用自顶而下的设计方法，从整体到各个功能模块进行设计，而后进行模块化编程。

此外，实验中我有针对性的使用平日在课堂中学习的 C#语言中的特性，以起到提高 C#语言编程的熟练度的目的，如使用了委托与事件、异常处理、控件、socket 编程、序列化、流与文件 IO、多线程及异步编程等知识。

军棋规则复杂，棋子种类繁多。因此人机对战部分自创了打分评价体系用于机器落子，并在遍历所有棋子后寻找最优解输出。工兵寻径算法则是创造性的当作迷宫问题寻解问题来解决。

项目耗时较长且代码量充足，个人代码量为军棋模块中基础规则实现、人机对战、网络对战、辅助模块代码量综合，超过 2800 行。

关键词：军棋； 寻径算法；网络编程； 搜索算法

目 录

1 项目背景主要功能及目标.....	5
1.1 项目背景.....	5
1.2 项目主要功能及目标.....	5
1.2.1 军棋基础规则完整实现.....	5
1.2.2 人机对战的实现.....	6
1.2.3 网络对战的实现.....	6
1.2.4 残局的保存和读取.....	6
1.2.5 设置、帮助、关于.....	6
1.2.6 体验良好的 UI 界面.....	6
2 模块的详细设计.....	8
2.1 本地对战模块及基础规则实现.....	8
2.2 网络模块.....	10
2.3 AI 模块.....	12
3 模块实现采用的具体技术.....	15
3.1 本地对战模块及基础规则实现.....	15
3.2 网络模块.....	16
3.3 AI 模块.....	17
4 总体评价.....	18
4.1 难度.....	18
4.1.1 覆盖知识面广，考察能力全面.....	18
4.1.2 规则实现复杂，数学逻辑问题繁多.....	19
4.1.3 网络编程模式，涉及部分课外知识.....	20
4.1.4 人机对战中的机器设计.....	20
4.2 代码量.....	20
4.3 亮点.....	20
4.4 收获.....	21
5 附录.....	22
5.1 界面截图.....	22
5.2 代码截图.....	26

1 项目背景主要功能及目标

1.1 项目背景

本项目是棋逢对手——基于 AI 棋类游戏合集的一部分，项目主题为：实现一个能够实现双人离线对战、人机对战、网络对战的军棋，并具有聊天、保存残局、音效提醒等辅助功能，在良好的 UI 界面中给玩家舒适的游戏体验。

以下为项目主体——军棋的主要规则：

军棋是中国深受欢迎的棋类游戏之一。两人游戏时，玩家分占棋盘的上下两角，相互作战。

军棋棋盘：行走路线包括铁路线和公路线，每方有 5 个行营、23 个兵站（翻棋 25 个）、2 个大本营。准备好双方不同颜色的棋子。兵站是棋子的摆放位置；铁路线是棋子的快行线，只要在直线上棋子走的步数都不限（除了工兵以外，其他棋子都不许拐弯），公路线是慢行线，每次只能移动一步；行营是棋子的保护区，在行营中的棋子可以免受其他任何棋子的攻击，行营里的棋子每次只能移动一步；大本营其中一个为军旗所在位置，另一个可以摆放其他任何棋子，进入大本营的棋子不能再移动。

此外，工兵行棋比较特殊，只要在铁路线上，在没有挡路的情况下，可以拐弯。地雷可以胜工兵外所有有生命的棋子，与炸弹同归于尽。

1.2 项目主要功能及目标

1.2.1 军棋基础规则完整实现

棋类游戏中，军旗的规则相对复杂，棋子种类繁多，不同棋子在移动、吃子时都有自己独特的规则，胜利的条件是一方扛起另一方的军旗。

移动棋子需要考虑的是工兵寻径特殊线路的问题、铁路和公路不同以及兵站、行营、大本营的不同。

吃棋子需要考虑的是士兵、排长、连长、营长、团长、旅长、师长、军长和司令之间的大小级别，同时也应注意炸弹可以和任意棋子同归于尽，地雷只有工兵

可以排除的特殊情况。

实现以上规则后，加入游戏双方记边后即可实现离线双人对战。

1.2.2 人机对战的实现

人机对战的功能实现有轻松和简单模式，鉴于军棋 AI 设计有困难，我们采用的是打分机制来实现一个判断函数，通过遍历所有可移动棋子来对所有棋子的各种移动行为进行打分，最终选取出得分最高的行棋方式并输出。

1.2.3 网络对战的实现

网络对战通过调用系统类来实现信息的监听、连接、发送信息。通过 IP 地址实现网络互联，能够实现网络对战双方的线上聊天以及下棋操作的实时同步。

1.2.4 残局的保存和读取

棋局对战时，支持随时将目前的棋局保存，也支持保存后随时将残局读取以便继续下棋。

1.2.5 设置、帮助、关于

程序有设置功能用于调整游戏的音效开关和提示框开闭等，同时有帮助用于显示游戏规则，还有关于显示我们的联系方式和程序版本。

1.2.6 体验良好的 UI 界面

程序配有定制的 UI 界面，可以有效提高玩家的游戏体验。

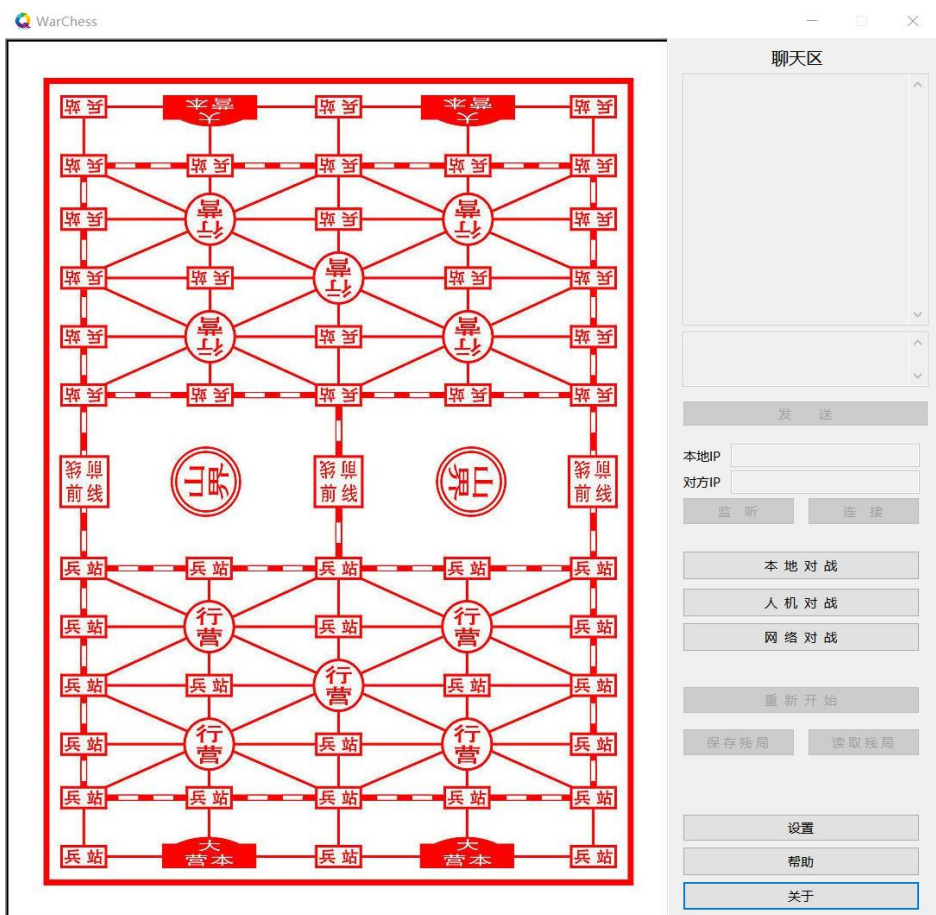


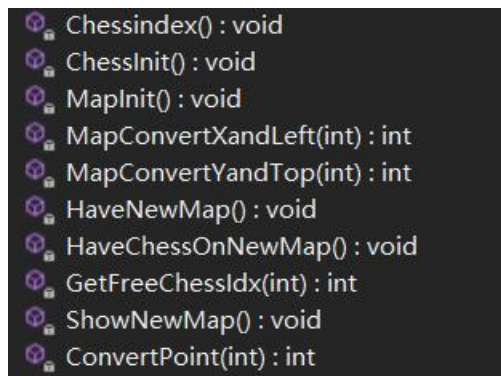
图 1 项目 UI

2 模块的详细设计

2.1 本地对战模块及基础规则实现

本模块是军棋游戏中最为基础的模块，主要有棋谱生成模块和棋局处理模块组成。棋谱生成模块负责随机生成一个符合规则的棋谱（双方军旗则通过随机数种子生成在任意一方），棋局处理模块负责处理选中棋子（鼠标点击），点击吃子或走到相应的空位置（鼠标点击），当一方的军旗被吃时，系统判定游戏结束，并作出相应提醒。

棋谱生成模块。首先通过数组 `PictureBox[]` 生成一个表示棋子的数组，建立一个枚举 `PlayerColor` 用来标志双方持方，生成初始棋盘 `Map[,]`。而后通过逻辑棋盘坐标和实际程序坐标进行转换的函数，根据随机函数生成的信息将棋子放置在对应的棋盘位置，下图所示为相关函数：



```
Chessindex() : void
ChessInit() : void
MapInit() : void
MapConvertXandLeft(int) : int
MapConvertYandTop(int) : int
HaveNewMap() : void
HaveChessOnNewMap() : void
GetFreeChessIdx(int) : int
ShowNewMap() : void
ConvertPoint(int) : int
```

图 2 生成棋盘相关函数

棋局处理模块。根据分析我们易知逻辑，首先点击一个 `PictureBox` 选定需要操作的棋子，然后点击一处 `PictureBox` 表示吃子或点击一处未放棋子的空兵站处表示尝试移动到该位置。然后调用逻辑判断能否移动、能否吃，并进行相关棋子移动。

判断棋子移动处有很多数据结构问题，如铁道上允许一次性直线移动多步，而工兵允许在棋盘上铁道处转弯移动，因此我自行设计了判断水平和竖直方向有无棋子挡路的算法 `VLine_Juge` 与 `HLine_Juge`，以及工兵寻径算法 `JudgeWay`。其中工兵寻径算法创造性的利用上学期数据结构课堂上的递归算法，将寻径过程转化为迷宫问题解决。

下图为相关函数：


```
MoveChess(int, int, int) : void
IsRedChess(PictureBox) : bool
FirstChessCanMove(PictureBox) : bool
IsBigHome(int, int) : bool
IsCamp(int, int) : bool
SecondChessAction(PictureBox) : void
SecondFreeAction(int, int) : void
AIAction() : void
IfOnRail(int, int) : bool
RailCanGO(int, int, int) : bool
JudgeWay(int, int, int, int) : bool
VLine_Juge(int, int, int) : bool
HLine_Juge(int, int, int) : bool
EatAction(int, int) : void
ReverseTurnColor() : void
```

图 3 棋局处理相关函数

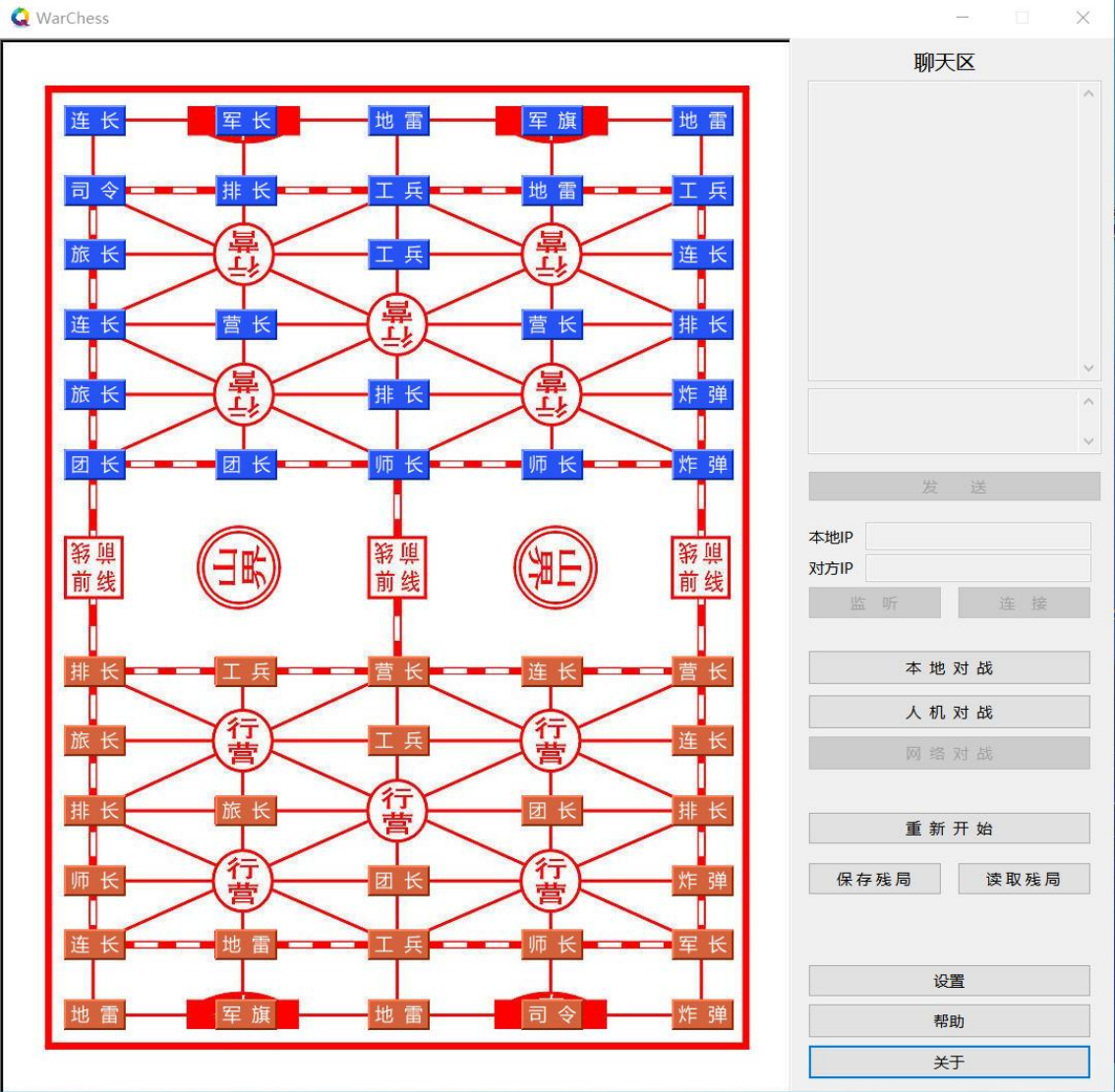


图 4 本地对战模式

2.2 网络模块

本模块用于局域网联机对战模式。在局域网联机对战时，下棋和输赢判断部分复用离线双人对战模式的代码，但在网络通信部分，加入了新的内容。

在实现联机对战时，采用 C#提供的套接字（socket）和 TCP/IP 协议，并重写了报文段部分，对缓冲区、面向应用的数据报的数据部分的格式进行设定。

在数据连接部分，采用 Client/Server 模式，并调用系统提供的 API。“监听”一方充当 Server，开启端口，等待连接；“连接”一方输入对方的 IP 地址，进行连接。连接后，使用 8880 端口进行发送数据，8881 端口则用来接收数据。

在数据报部分，数据报包含 class 和 content 两部分。Class 用于做消息类型标识，content 为交流内容。服务机根据不同的需求发出指令，由于服务机和客户机持方相对，接收移动信息之后接收方根据信息进行转换，按照转换后的坐标信息输出。

以下为网络模式中，信息传递信号与作用：

Class 为 0 时，表示初始化棋局信息，content 为逗号分隔开的需要传递的棋局分配消息；Class 为 1 时，表示聊天信息，content 为聊天内容，判断内容后可以显示在聊天区显示聊天内容；Class 为 2 时，表示下棋至空兵站处，调用函数 MoveChess(idx, x, y)；Class 为 3 时，表示下棋至棋子处，调用函数 EatAction(idx1, idx2)；Class 为 4 时，表示一方胜利，游戏结束。

网络模块以及相关辅助模块的相关函数如下：

```

OpenWechat() : void
CloseWechat() : void
MapConvertLogic(int, int) : int
bt_MouseDownMap(object, MouseEventArgs) : void
bt_MouseMoveMap(object, MouseEventArgs) : void
bt_MouseDown(object, MouseEventArgs) : void
bt_MouseMove(object, MouseEventArgs) : void
LocalPlayBtn_Click(object, EventArgs) : void
AIPlayBtn_Click(object, EventArgs) : void
WebPlayBtn_Click(object, EventArgs) : void
RePlayBtn_Click(object, EventArgs) : void
SaveMapBtn_Click(object, EventArgs) : void
ReadMapBtn_Click(object, EventArgs) : void
SettingBtn_Click(object, EventArgs) : void
HelpBtn_Click(object, EventArgs) : void
AboutBtn_Click(object, EventArgs) : void
btnListen_Click(object, EventArgs) : void
listen() : void
btnConnect_Click(object, EventArgs) : void
btnSend_Click(object, EventArgs) : void
manageChessEvent(object, ChessEvent) : void

```

图 5 Form1.cs 中网络模块相关函数

```

C# WarChessWeb.cs
  ControllInternet
    skRec : Socket
    ipeRec : IPEndPoint
    skSend : Socket
    ipeSend : IPEndPoint
    OnReceiveMsg : ChessEventHandler(object, ChessEvent)
    ControllInternet()
    listen() : void
    connect(string) : void
    sendMsg(byte, byte, string) : void
    receiveMsg(object) : void
    close() : void
    ~ControllInternet()

```

图 6 网络模块专用文件 WarChessWeb.cs

```

C# InterfaceAndDelegate.cs
  ISocket
    listen() : void
    connect(string) : void
    sendMsg(byte, byte, string) : void
    receiveMsg(object) : void
    close() : void
  ChessEvent
    Iclass : string
    content : string
    flag : string
    ChessEvent(string, string, string)
    ChessEventHandler(object, ChessEvent) : void

```

图 7 接口与协议定义文件中网络模块相关部分

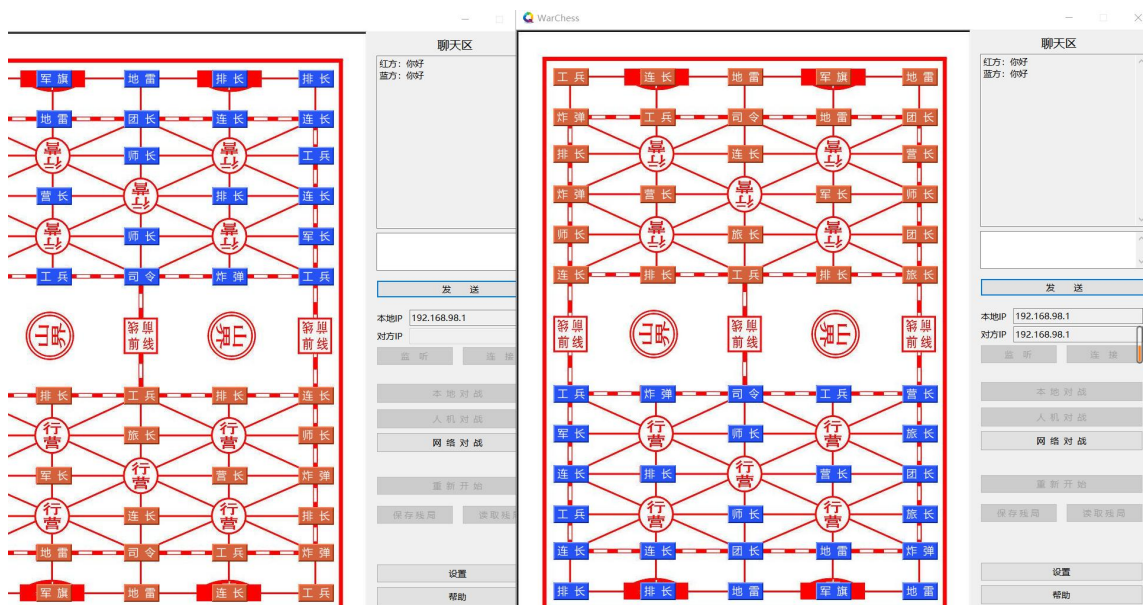


图 8 网络模式连接后棋局同步情况和聊天功能使用情况

2.3 AI 模块

本模块主要用于人机对战模式。在计算机下棋时，主要是要搜索最佳落棋点。在不同的难度设定中，所使用的算法也不同。

逻辑实现方法为：当玩家走完一步时，调用 `AIAction()` 对棋局情况进行得分情况判断并自动输出得分最高，即对机器最有利的一步棋。

军棋 AI 设计目前还没有算法，算法完全为我根据自身下军棋经验进行权值分配并根据大量实践对战调整所得，AI 棋力并不高，因此我将难度分配为“轻松”和“简单”两级。

轻松模式下，会遍历所有棋子的所有“相邻”可移动走法，对每个棋子的每种移动遍历进行评判打分，随后将打分最高的走法输出。

简单模式下，会遍历所有棋子的所有可移动走法，对每个棋子的每种移动遍历进行评判打分，随后将打分最高的走法输出。

因为算法整体过于复杂，代码量较大，这里展示主干逻辑部分：


```

588     for(int i=29;i<50;i++)
589     {
590         if(ChessPictureBox[i].Visible)
591         {
592             oldx = MapConvertXandLeft(ChessPictureBox[i].Left);
593             oldy = MapConvertYandTop(ChessPictureBox[i].Top);
594             if (IfToRail(oldx, oldy))...
1065             else if (IsCamp(oldx, oldy))...
1136             else if (!IsBigHome(oldx, oldy))//普通兵站
1137             {
1138                 //能否向左, 能否吃。空1分, 吃子(2+落差)分
1139                 if (oldx >= 1)...
1203                 //能否向右, 能否吃。空1分, 吃子(2+落差)分
1204                 if (oldx <= 3)...
1268                 //能否向上, 能否吃。空1分, 吃子(2+落差)分
1269                 if (oldy >= 1 && !( oldx ==1 && oldy == 6 ) && !(oldx == 3 && oldy == 6))...
1333                 //能否向下, 能否吃。空2分, 吃子(3+落差)分
1334                 if (oldy <= 10 && !(oldx == 1 && oldy == 5) && !(oldx == 3 && oldy == 5))...
1398             }
1399         }
1400     }
1401 }
1402 //遍历结束, 按照分数移动棋子
1403 if(idx == 25 && x==0 && y==0 )...
1410 if(Map[y,x]==999)...
1418 else...
1439 ReverseTurnColor();
1440 }

```

图 9 AI 模块的主干判断部分

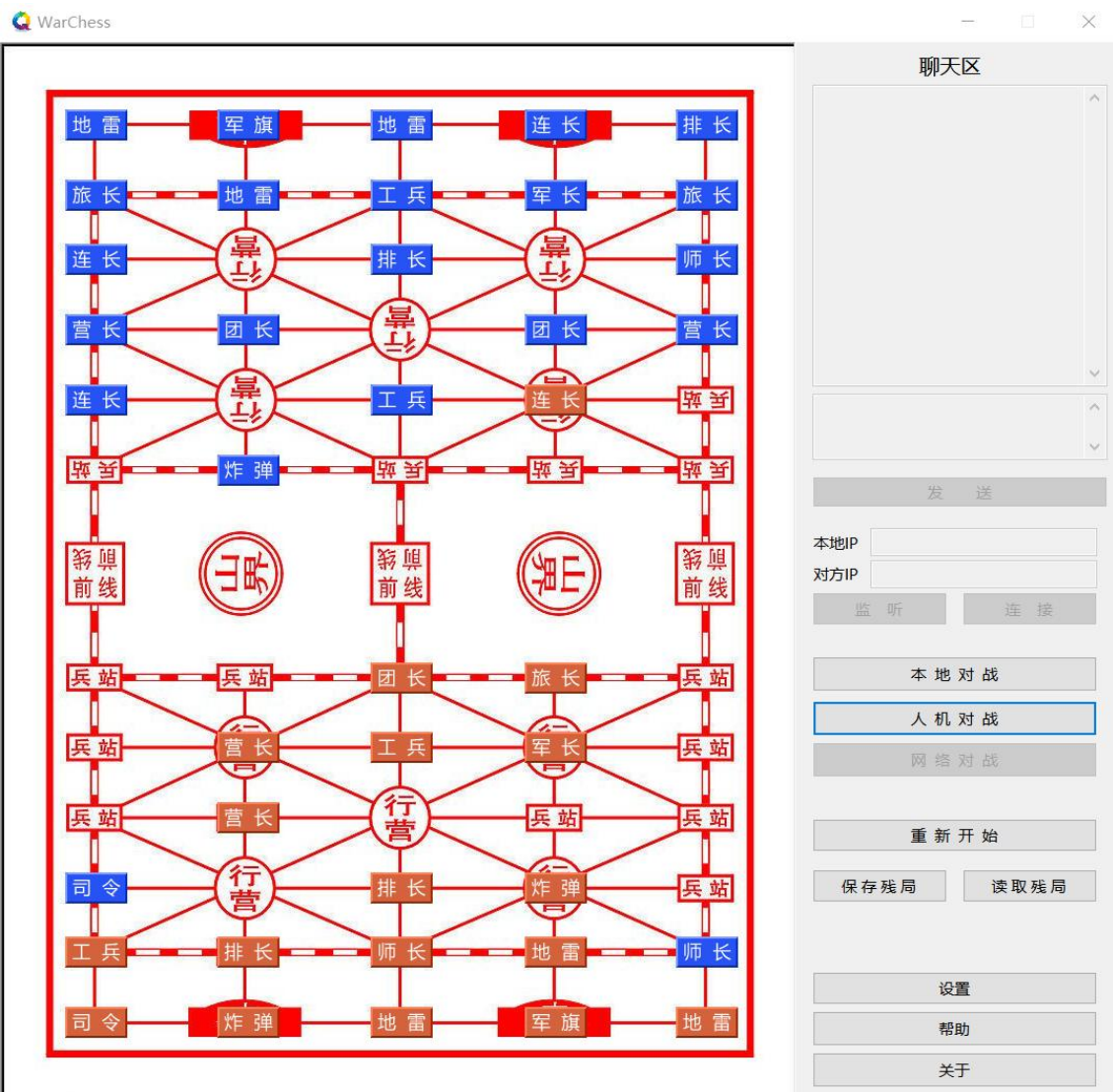


图 10 人机对战模式

3 模块实现采用的具体技术

3.1 本地对战模块及基础规则实现

本模块是军棋游戏中最为基础的模块，采用的技术主要在算法层面。其中值得一提的是棋局分析中的寻径算法。

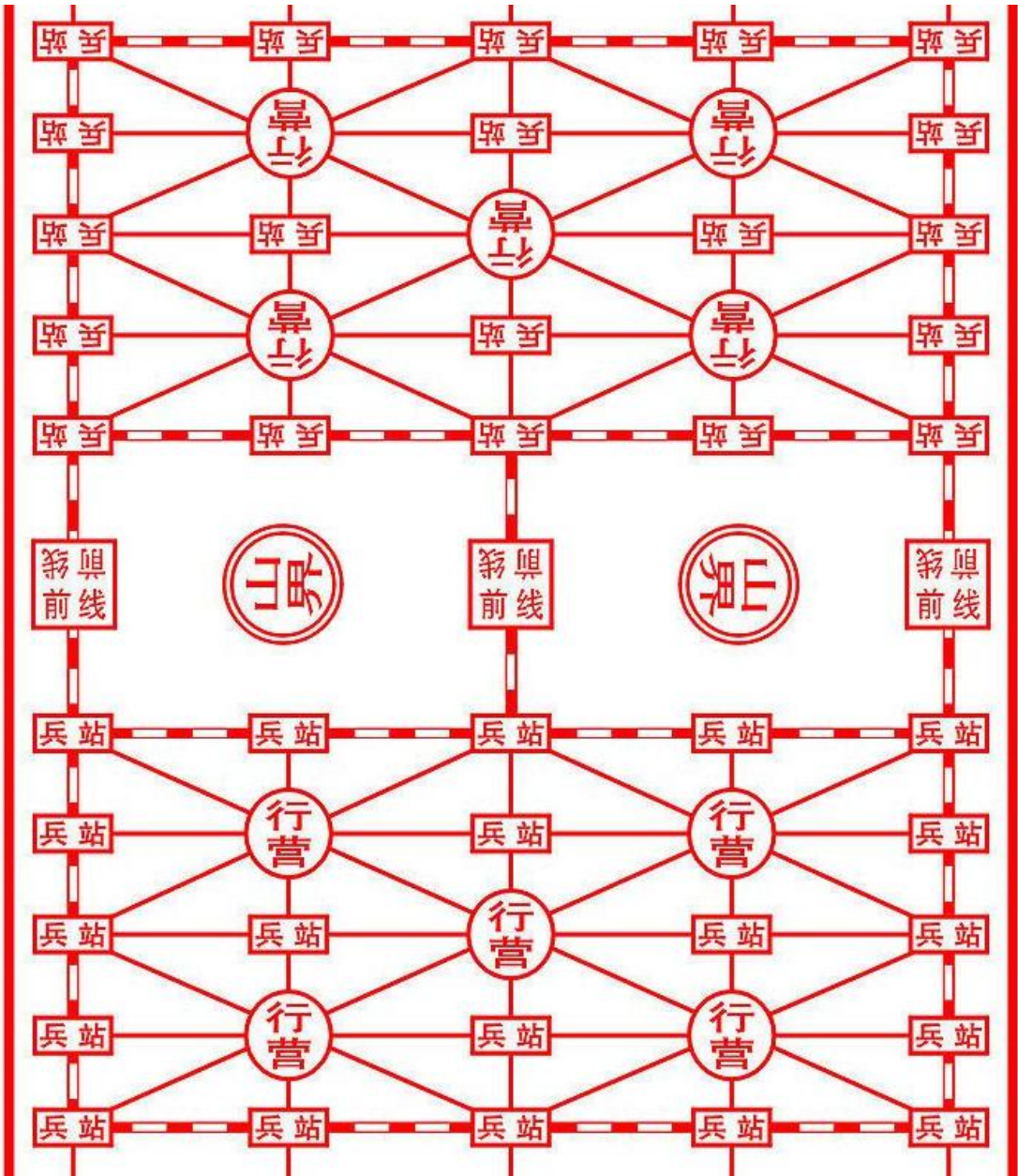


图 11 含铁道的军棋棋盘

所有可移动的棋子可以在铁道上直线移动，工兵可以在不被阻挡的情况下在铁道中自由移动，因此如何设计一个高效的算法成为基础规则实现中的重要问题。其中好解决的问题是直线移动的判断，可以分为水平和竖直两个角度来判断，

用 for 循环和逻辑坐标来判断是否有棋子挡路。

较为复杂的是工兵寻径算法，即自身在铁道上的工兵可以在不受阻碍时前往铁路上的任意一处兵站，这里创造性的采用了数据结构中学习的递归算法，将不能移动的位置视为迷宫围墙，从而将工兵寻径问题转化为迷宫问题，在设计算法之后发现十分高效可靠。

以下为工兵寻径算法代码，代码十分简洁，却是在众多方法中精简且自行设计的高效递归算法：

```
/// <summary>
/// 工兵寻径算法
/// </summary>
/// <param name="x1">初始横坐标</param>
/// <param name="y1">初始纵坐标</param>
/// <param name="x2">目标横坐标</param>
/// <param name="y2">目标纵坐标</param>
/// <returns></returns>
private bool JudgeWay(int x1, int y1, int x2, int y2)
{
    if (x1 == x2 && y1 == y2) return true;
    if (x1 < 0 || x1 > 4 || y1 < 0 || y1 > 11) return false;
    if (MIGONG[x1, y1] == 0) return false;
    MIGONG[x1, y1] = 0;
    return (JudgeWay(x1 + 1, y1, x2, y2) || JudgeWay(x1 - 1, y1, x2, y2) ||
        JudgeWay(x1, y1 + 1, x2, y2) || JudgeWay(x1, y1 - 1, x2, y2));
}
```

图 12 工兵寻径算法

3.2 网络模块

网络模块主要是调用系统 System.Net 和 System.Net.Sockets，通过套接字来建立连接并收发数据。这里，ControlInternet 类（包含监听、连接、发送信息、接受信息函数等网络通信所用函数）继承自自定义接口 ISocket（该接口定义了通信相关函数），而 ControlInternet 类的数据成员的类型均来自系统预设，即 System.Net 和 System.Net.Sockets。

在实际操作中，“监听”一方充当 Server，在按下“监听”按钮后，开启端口，等待连接；“连接”一方则输入对方的 IP 地址，进行连接。连接后，使用 8880 端口进行发送数据，8881 端口则用来接收数据。

监听、连接、发送信息、接受信息函数等网络通信所用函数的具体实现，均是

通过调用系统 API 完成的，这些函数只是对 API 进行了封装。

3.3 AI 模块

AI 模块的重点是如何确定一个科学合理的估值函数来对棋局进行判断，并采用何种方式进行高效搜索，给出反馈。

这里我将每种棋子的各种行为都进行的讨论，并对每种行为进行了打分。

首先，工兵到司令每一种棋子都有对应的基础分数，地雷、炸弹、军棋三类特殊棋子在吃与被吃时有对应的分数。

当普通棋子移动时，可获得 1 分，当移动方向为对方大本营方向时，可获得 2 分，作为进攻激励。当普通棋子吃子时，可获得 $(2 + \text{吃子棋子分数} - \text{被吃棋子分数})$ 分，当炸弹吃子同归于尽时，可获得 $(\text{被吃棋子分数} - \text{炸弹棋子分数} - 5)$ 分，当普通棋子吃地雷炸弹时，可获得 $(\text{被吃棋子分数} - \text{吃子棋子分数} + 5)$ 分，当棋子扛军旗时，可以直接获得 20 分。

随后，对所有棋子进行遍历，首先考虑每个棋子的类型和位置，军旗和地雷不可移动，位置为本营则根据规则不可移动，其次考虑棋子在铁道上、行营还是普通兵站。

若棋子在铁道上，则代表棋子可能沿着铁道方向直走多步，需要判断棋子能否向左、向右可以移动到哪一位置、能否吃子，右下上三个方向同理，并得出分数；若棋子在行营，则遍历四周所有位置，先判断是否为行营，行营是否有棋子，如果不是行营则判断该位置是否为空，不为空时是否为敌方棋子，吃子分数，最终得出棋子最佳步骤分数；若棋子在普通兵站，则判断上下左右能否移动，是否有敌方棋子，能否吃子。

最终遍历结束，可以获得此时每种棋子每种落子可能中期望最大的落子方案，我们这里将该方案视为最佳方案。

由于军棋规则复杂，这里只采用了深度不高的剪枝算法，但随着多次实践中权值的调整，算法已经有了较大程度的优化。

4 总体评价

4.1 难度

4.1.1 覆盖知识面广，考察能力全面

本次实验我们小组的主要目的就在于，利用大作业的机会，对一学期 C# 学习到的所有知识进行系统巩固。

项目涉及到第二章到第十一章的各章大量知识。

第二章：C#语言基础。知识简单，不需要列举。

第三章：类、接口、结构。项目根据各个功能需求定义了类，定义了枚举结构用于判断下棋权的交换，并且构造了大量的索引器，网络对战部分定义了接口。

第四章：C#高级特性。网络部分大量运用委托和事件，并且本游戏对程序运行过程中所可能出现的各种异常都进行了完善地捕获和处理。

第五章：基础类及常用算法。项目中大量运用 Form 基础类和常见类库，如 String 类的信息处理，Console 类的测试输入输出等，此外也运用了遍历和递归等常见算法。

第六章：流、文件 IO。项目的残局保存和读取涉及到棋盘信息与文件的转化，进行文本的输入输出。

```
/// <summary>
/// 保存残局
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SaveMapBtn_Click(object sender, EventArgs e) ...

/// <summary>
/// 读取残局
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ReadMapBtn_Click(object sender, EventArgs e) ...
```

图 13 保存残局和读取残局函数

第七章：Windows 窗体及控件。由于程序是 Winform 程序，程序中使用了数十种控件以提高综合体验。

第八章：绘图及图像。棋盘和棋子绘制过程中，将棋子的图片（png 格式）巧妙地通过继承于 PictureBox 类别转换为能够直接添加进 Form 中的形式，可以算作某种形式上的自定义控件。

第九章：文本、XML 及网络信息获取。网络对战和网络聊天部分使用了本章知识，这里创建一个持续监听的子线程进行局域网沟通，并且使用 System.Net 中的 TcpClient 以及 TcpListener 以实现局域网内在线双人对战和对战实时聊天功能。

第十章：多线程和异步编程。由于网络对战和网络聊天部分需要，这里创建了一个持续监听的子线程进行局域网沟通，并且用 invoke 函数解决了线程之间的 bug。

第十一章：数据库、网络、多媒体编程。网络编程方面，采用 C#提供的套接字（socket）和 TCP/IP 协议，并重写了报文段部分，对缓冲区、面向应用的数据报的数据部分的格式进行设定。多媒体编程方面，程序调动了大量音效以提高游戏体验。

```
using System;
using System.Runtime.InteropServices; //add

namespace WarChess
{
    /// <summary>
    /// 用于播放音乐
    /// </summary>
    public class PlaySound
    {
        private const int SND_SYNC = 0x0;
        private const int SND_ASYNC = 0x1;
        private const int SND_NODEFAULT = 0x2;
        private const int SND_LOOP = 0x8;
        private const int SND_NOSTOP = 0x10;

        public static void Play(string file)
        {
            int flags = SND_ASYNC | SND_NODEFAULT;
            sndPlaySound(file, flags);
        }

        [DllImport("winmm.dll")]
        private extern static int sndPlaySound(string file, int uFlags);
    }
}
```

图 14 用于音效播放的文件 Playsound.cs

4.1.2 规则实现复杂，数学逻辑问题繁多

军棋规则较为复杂，棋子种类繁多，涉及到大量的判断。此外，棋盘中的众多地形则会在规则实现中转化为大量的数学问题和数据结构问题，处理项目中的数

学逻辑问题是项目的一个难点，也让做项目的过程中充满趣味。

4.1.3 网络编程模式，涉及部分课外知识

网络编程的实现是本项目的一个难点，在写网络模式时我阅读了大量的文档，并且与小组同学进行了大量交流，最终决定用 `socket` 来实现这一功能，项目实现了局域网对战以及局域网聊天功能。

4.1.4 人机对战中的机器设计

AI 算法部分涉及较为繁琐复杂，也称得上是难点。由于之前课程中学习过人工智能课程，自己也查阅学习过一些人工智能基础算法，总体设计工作处于总体顺畅，局部解决难题的状态。

4.2 代码量

由于本程序实现了军旗的基础功能、网络对战和 AI 算法，且均为纯手写代码，因此代码工作量相对较大，除去自动生成的代码，其余代码量的具体数据如下表所示：

表 1 代码量

文件名	代码量（行）
Form1.cs	2219
Formset.cs	105
InterfaceAndDelegate.cs	54
MessageControl.cs	253
PlaySound.cs	27
WarChessWeb.cs	132
总计	2790

4.3 亮点

- 1、基础规则实现中的工兵寻径算法，用数据结构中的知识，将工兵寻径算法视为迷宫问题，运用递归函数解决。
- 2、网络对战中 Socket 的使用，采用 C#提供的套接字（`socket`）和 TCP/IP 协

议，并重写了报文段部分，对缓冲区、面向应用的数据报的数据部分的格式进行设定。

3、AI 算法，自行设计并优化了评估函数作为 AI 落子判断，给出适当的激励和权重分配。

4.4 收获

项目工程量大，耗时长，覆盖知识面广，收获主要有以下几点：

1、C#课程基础知识的大规模自测和运用。在项目中，我有意识的复习和练习书本课堂上学习的 C#知识，一定程度上扎实了自己的基本功。

2、逻辑思维的提高。棋类游戏的规则实现，尤其是军旗的规则实现并不容易，需要自行进行大量的逻辑判断与代码编写，训练了我的逻辑思维。

3、网络编程与多线程。通过大作业项目亲身实践了网络编程，实现了网络对战和网络聊天，并且创建了子线程来实现监听沟通，让我深入探索了课堂上学习的概念，在实践中加深了认识。

4、AI 的编写。之前没有编写过军棋这样规则复杂，少有人涉足的棋类 AI，本次实验中也是抱着试探的心态接触，值得开心的是优化后的 AI 效果不错。

5、更加具有想法，敢于创造。C#的学习让我接触了很多新的知识，也让我逐渐敢于提出大胆的设想，并通过自己的努力把设想实现，既提高了代码能力，又让我更加敢于构想创造。

5 附录

5.1 界面截图

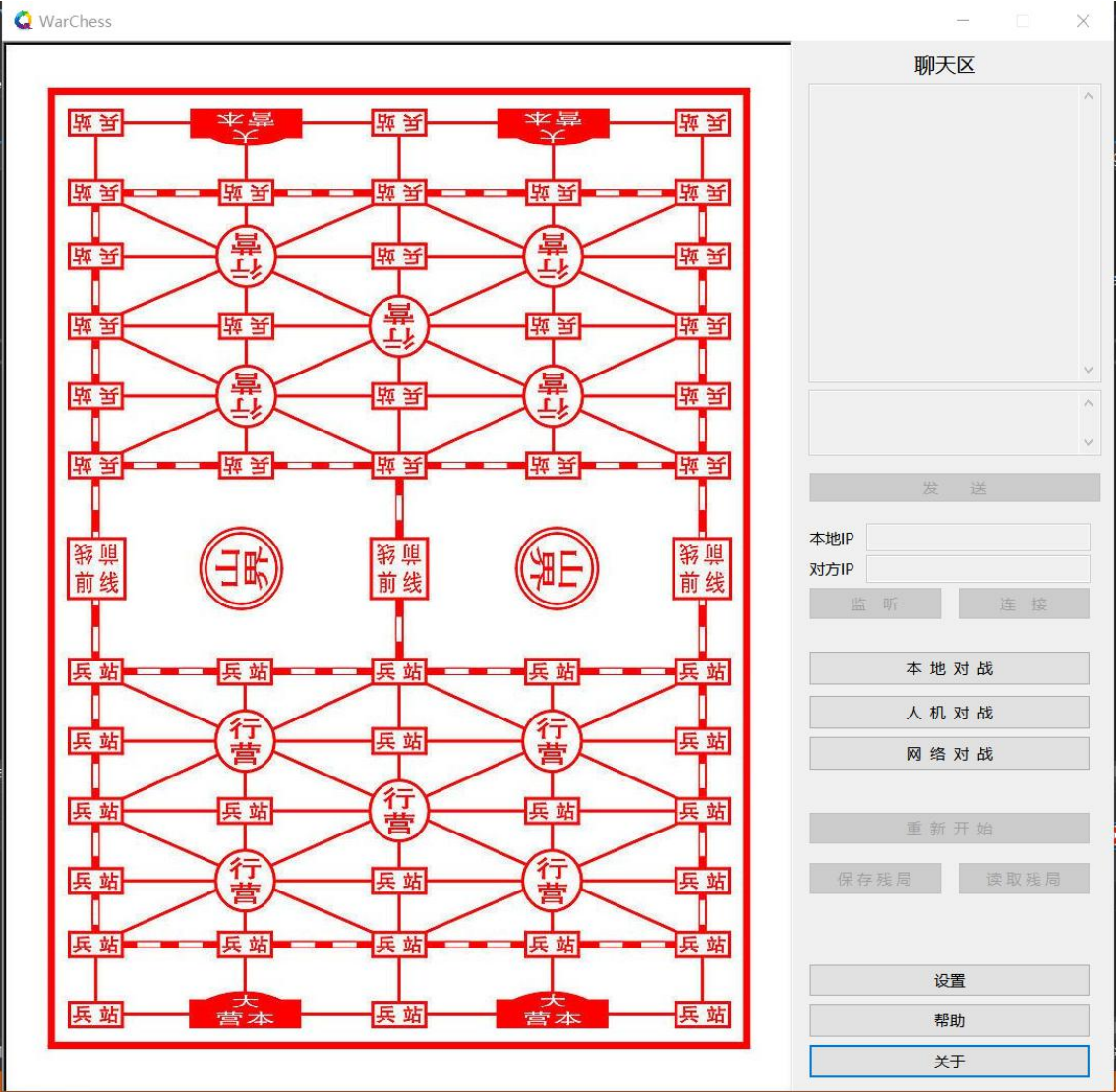


图 15 初始界面

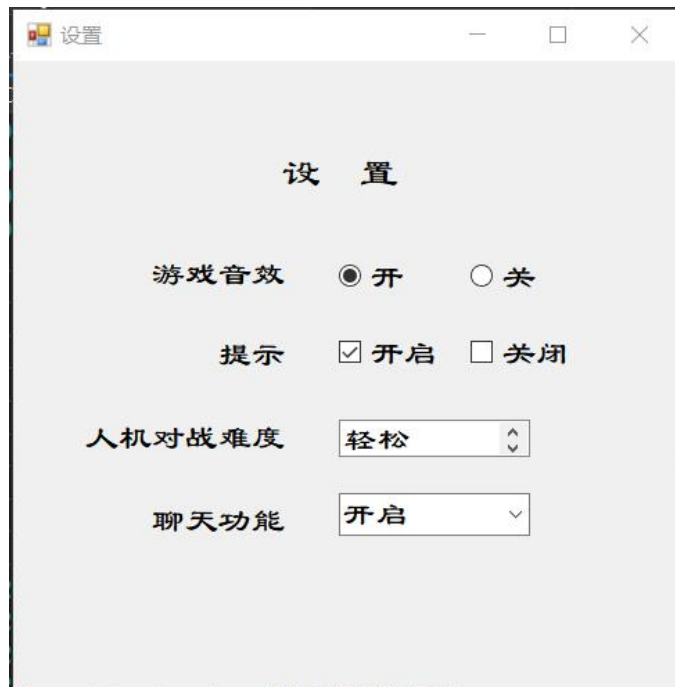


图 16 设置界面

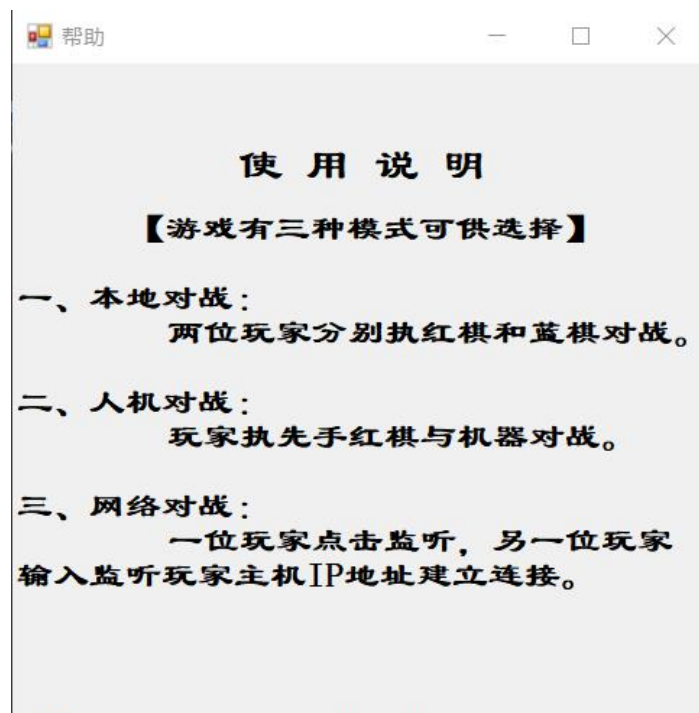


图 17 帮助界面

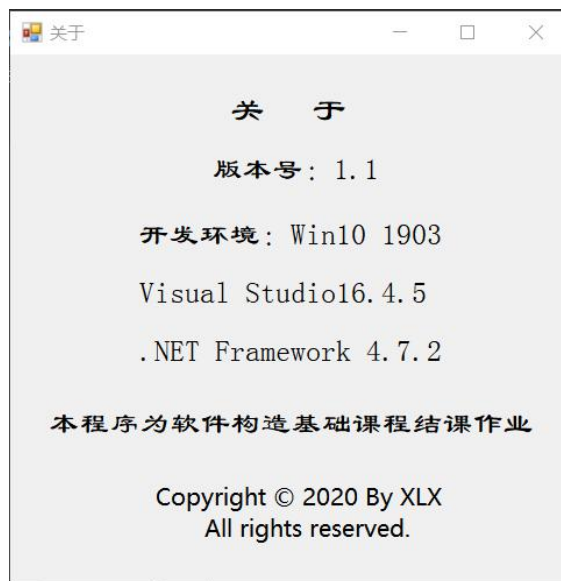


图 18 关于界面

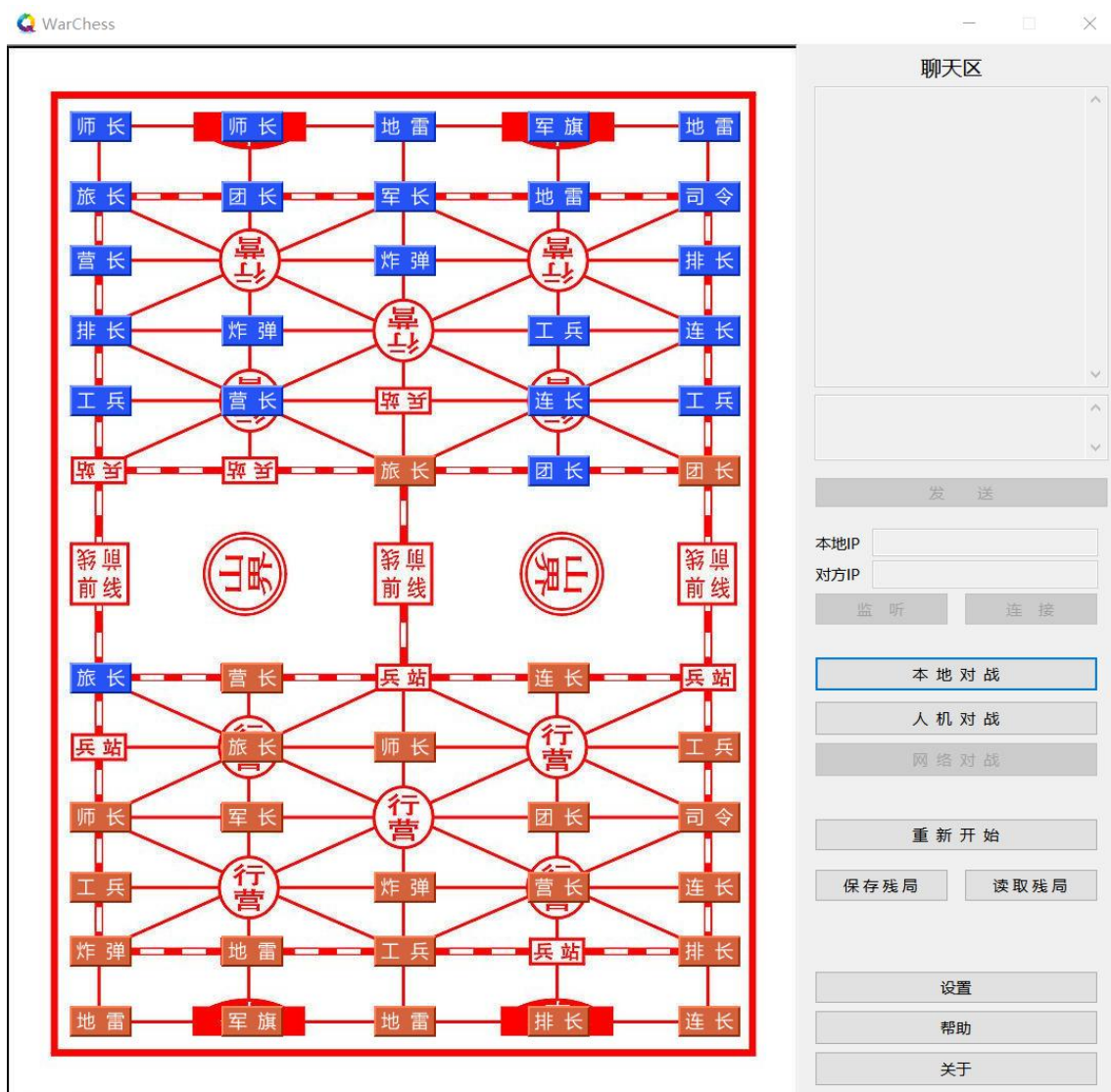


图 19 本地对战截图

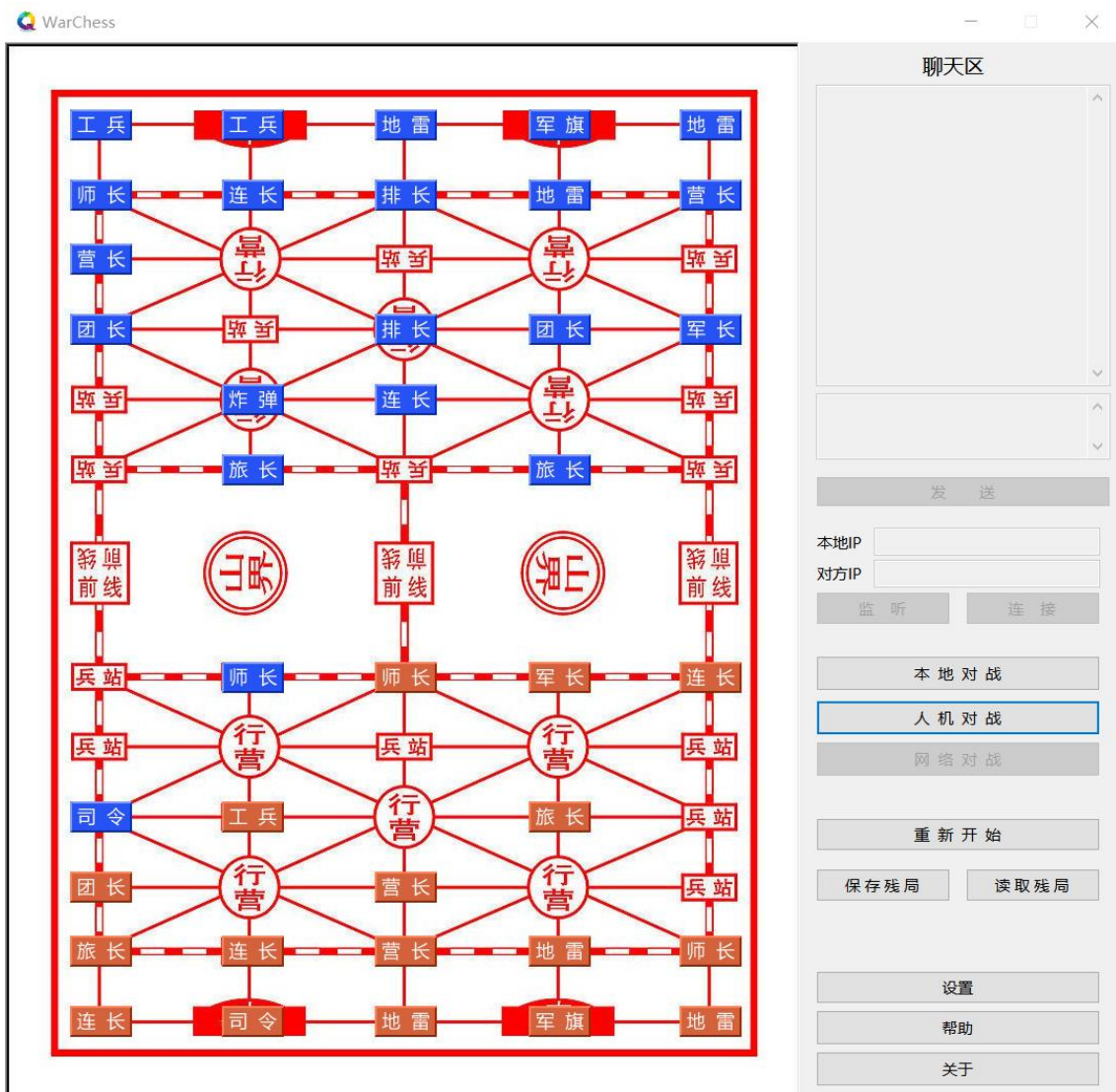


图 20 人机对战截图

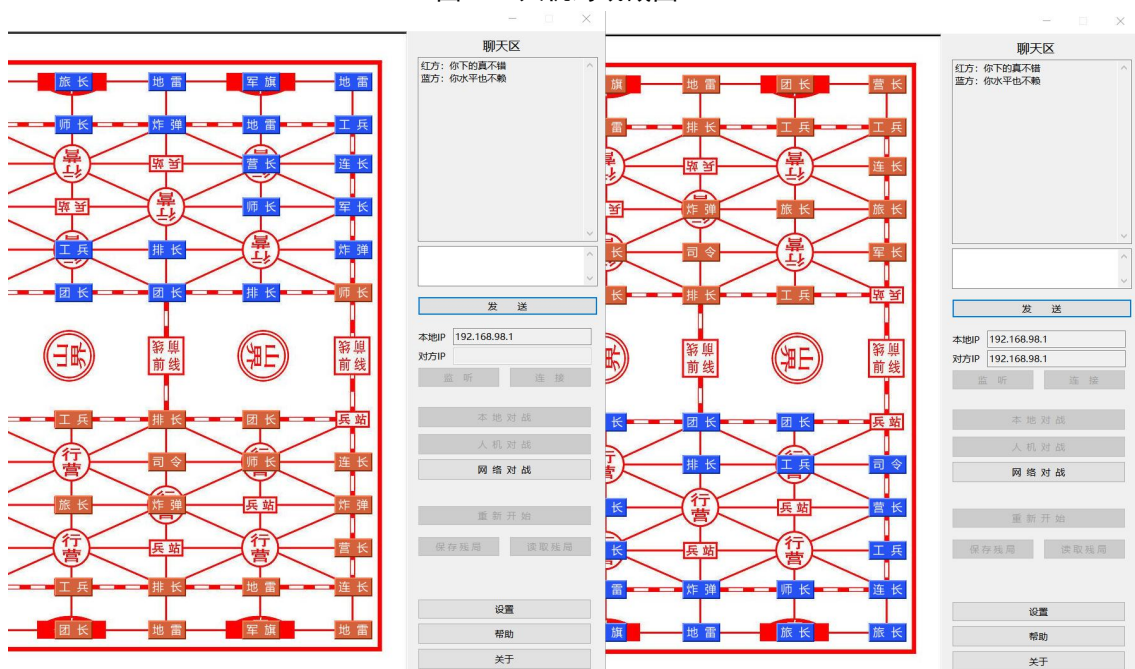


图 21 网络对战截图

5.2 代码截图

由于文件代码量较大，这里每个文件只截取一张图。

表 2 代码统计

文件名	代码量（行）
Form1.cs	2219
Formset.cs	105
InterfaceAndDelegate.cs	54
MessageControl.cs	253
PlaySound.cs	27
WarChessWeb.cs	132
总计	2790

```
540 private void SecondFreeAction(int x, int y){...}
575 /// <summary> AI模式中机器下棋，使用遍历打分机制，选取得分最高的走法
578 private void AIAction(){...}
1440 /// <summary> 棋子是否在铁路上
1446 private bool IfTonRail(int old_x, int old_y){...}
1453 /// <summary> 铁路上能否走到指定位置
1460 private bool RailCanGO(int idx, int x, int y){...}
1485 /// <summary> 工兵寻径算法
1493 private bool JudgeWay(int x1, int y1, int x2, int y2){...}
1503 /// <summary> 垂直方向判断是否有别的棋子挡路
1510 private bool VLine_Judge(int m, int n, int x){...}
1520 /// <summary> 水平方向判断是否有别的棋子挡路
1527 private bool HLine_Judge(int m, int n, int y){...}
1537 /// <summary> 走棋吃子: CI.SendMsg(3, 0, Str);调EatAction[idx1,idx2], CI.SendMsg(4, 0, ...
1542 private void EatAction(int idx, int i){...}
1633 /// <summary> 用于变换turncolor
1636 private void ReverseTurnColor(){...}
1642 /// <summary> 用于设置部分聊天功能
1645 public void OpenWechat(){...}
1651 /// <summary> 用于重新开始聊天功能
1654 public void CloseWechat(){...}
1660 /// <summary> 专门用于将空地图上点击转化为坐标t[y=(t/5),x=(t%5)]
1666 private int MapConvertLogic(int x, int y){...}
1742 /// <summary> 专门用于捕捉鼠标在空地图上的点击
1747 private void bt_MouseDownMap(object sender, System.Windows.Forms.MouseEventArgs e){...}
1762 /// <summary> 专门用于捕捉鼠标在空地图上的移动
1767 private void bt_MouseMoveMap(object sender, System.Windows.Forms.MouseEventArgs e){...}
1771 /// <summary> 专门用于捕捉鼠标在棋子上的点击
1776 private void bt_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e){...}
1800 /// <summary> 专门用于捕捉鼠标在棋子上的移动
1805 private void bt_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e){...}
1809 /// <summary> 开启本地对战
1814 private void LocalPlayBtn_Click(object sender, EventArgs e){...}
1830 /// <summary> 开启人机对战
1835 private void AIPlayBtn_Click(object sender, EventArgs e){...}
```

图 22 Form1.cs 部分代码

```

11 namespace WarChess
12 {
13     public partial class FormSet : Form
14     {
15         public FormSet()...
19         private void radioButton1_CheckedChanged(object sender, EventArgs e)...
30         private void radioButton2_CheckedChanged(object sender, EventArgs e)...
41         private void checkBox1_CheckedChanged(object sender, EventArgs e)...
50         private void checkBox2_CheckedChanged(object sender, EventArgs e)...
58         private void FormSet_Load(object sender, EventArgs e)...
89         private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)...
97     }
98 }
99

```

图 23 FormSet.cs

```

15 namespace WarChess
16 {
17     /// <summary>
18     /// 联网对战接口
19     /// </summary>
20     public interface ISocket...
29
30     /// <summary>
31     /// 联网对战事件
32     /// </summary>
33     public class ChessEvent ...
46
47     /// <summary>
48     /// 委托
49     /// </summary>
50     /// <param name="sender"></param>
51     /// <param name="e"></param>
52     public delegate void ChessEventHandler(object sender, ChessEvent e);
53 }

```

图 24 InterfaceAndDelegate.cs


```

5      {
6      /// <summary> MemoryStream类, 用于处理信息流
9      public class MemoryStream
10     {
11         private byte[] _buffer;
12         private int _position;
13         private int _length;
14         private int _capacity;
15         public MemoryStream()...
22         private byte ReadByte()...
30         private int ReadInt()...
40         private byte[] ReadBytes(int count)...
67         public bool Read(out Message message)...
97         private void EnsureCapacity(int value)...
112        public void Write(byte[] buffer, int offset, int count)...
123        private void Remove(int count)...
137    }
138    /// <summary> Message类, 用于处理信息
141    public class Message
142    {
143        private byte _class;
144        private byte _flag;
145        private int _size;
146        private int _x;
147        private int _y;
148        private int _chessFlag;
149        private byte[] _content;
150        public byte[] Content...
155        public int Size...
160        public byte Flag...
165        public byte Class...
170        public int X...
175        public int Y...
180        public int ChessFlag...
185        public Message()...
189        public Message(byte @class, byte flag, byte[] content)...
196        public byte[] ToBytes()...
214        public static Message FromBytes(byte[] Buffer)...
221    }

```

图 25 MessageControl.cs

```

namespace WarChess
{
    /// <summary>
    /// 用于播放音乐
    /// </summary>
    public class PlaySound
    {
        private const int SND_SYNC = 0x0;
        private const int SND_ASYNC = 0x1;
        private const int SND_NODEFAULT = 0x2;
        private const int SND_LOOP = 0x8;
        private const int SND_NOSTOP = 0x10;

        public static void Play(string file)
        {
            int flags = SND_ASYNC | SND_NODEFAULT;
            sndPlaySound(file, flags);
        }

        [DllImport("winmm.dll")]
        private extern static int sndPlaySound(string file, int uFlags);
    }
}

```

图 26 PlaySound.cs

```

15 namespace WarChess
16 {
17     /// <summary> 网络对战类
20     public class ControlInternet : ISocket
21     {
22         private Socket skRec = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
23         private IPEndPoint ipeRec;
24         private Socket skSend = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
25         private IPEndPoint ipeSend;
26         public event ChessEventHandler OnReceiveMsg;
27         public ControlInternet() { }
28         /// <summary> 监听函数
31         public void listen()...
51         /// <summary> 连接函数
55         public void connect(string ipStr)...
64         /// <summary> 发送信息函数
70         public void sendMsg(byte @class, byte flag, string content)...
84         /// <summary> 接收信息函数
88         public void receiveMsg(object obj)...
113         public void close()...
118         ~ControlInternet()...
123     }
124 }

```

图 27 WarChessWeb.cs

教师评语评分

评语： _____

评分： _____

评阅人：

年 月 日

（备注：对该实验报告给予优点和不足的评价，并给出百分之评分。）