画着卡通人物

低可信度描述已自动生成

**实验报告**

**题 目**：**与基于强化学习的AI对战的黑白棋（Reversi）小游戏**

**学 院：信息学院**

**专 业：计算机科学与技术**

**组 员： A**

**B**

**C**

**D**

**2022年 5 月**

1. **实验内容**

基于MFC框架，实现一个黑白棋小游戏，玩家可以用鼠标在前端图形化界面操作，与后端运行的基于强化学习的AI对战。

本次实验内容主要可以分为四个模块，小组分工如下：

A：强化学习AI

B：MFC图形化界面

C: 前后端网络通信

D：依赖库管理

1. **实验环境**

操作系统：Windows 10 64位

集成开发环境：Visual studio 2022，pycharm 2022.1

AI开发与运行环境：python 3.9，tensorflow 2.8

1. **实验步骤**

**3.1. MFC部分：**

1. **需求分析，程序界面与控件设计**

在程序设计的前期需求分析阶段，我们组内经过讨论，对程序界面进行了以下设计

电脑屏幕截图

描述已自动生成

该程序只需要一个主界面即可，因此选择Visual Studio 2022中基于对话框的MFC应用模板

对话框主界面，大小为510\*540，有一个字符串与四个按钮

图形用户界面

中度可信度描述已自动生成

图形用户界面

中度可信度描述已自动生成

其中字符串用于实时显示当前棋盘上黑棋与白棋各自的数量

#define IDC\_CHESSCOUNT 1001

各按钮依次为：

开始游戏按钮，点击后用于开始一局新游戏。

#define IDC\_START 1002

提示按钮，默认不可点击，点击会在当前执棋方可以落子的空格中心显示一个红点。

#define IDC\_PROMPT 1003

结束本局按钮，默认不可点击，点击后会结束本局游戏。

#define IDC\_ENDGAME 1004

退出按钮，结束后会直接退出程序。

#define IDC\_QUIT 1005

另外有两个光标资源，用于在之后替换鼠标箭头光标，显示当前执棋方颜色

图片包含 游戏机, 笼子

描述已自动生成 图表

描述已自动生成

光标的ID设置为：

#define IDC\_BLACK 131

#define IDC\_WHITE 132

1. **黑白棋的规则与实现**
   1. 棋子与棋盘

棋子：黑白棋棋子每颗由黑白两色组成，一面白，一面黑，共64个（包括棋盘中央的4个）。棋子呈圆饼形。本次实验图形化界面为2D平面，所以使用黑白圆形棋子。

棋盘：黑白棋棋盘由64格的正方格组成，游戏进行时棋子要下在格内。现今的棋盘多以8x8较为普遍。本次实验中采取8\*8 棋盘，棋盘内信息保存在8\*8的二维数组当中。

以上的实现方法是，在对话框类的头文件ReversiDig.h中定义成员变量，各变量含义见代码注释。

文本

描述已自动生成

* 1. 落子和判定胜负的规则与实现

①黑方先行，双方交替下棋。把自己颜色的棋子放在棋盘的空格上

②当自己放下的棋子在横、竖、斜八个方向内有一个自己的棋子，则被夹在中间的全部翻转会成为自己的棋子。

③并且，只有在可以翻转棋子的地方才可以下子。除非至少翻转了对手的一个棋子，否则就不能落子。

④如果一方没有合法棋步，那他这一轮只能弃权，而由他的对手继续落子直到他有合法棋步可下。如果一方至少有一步合法棋步可下，他就必须落子，不得弃权。

⑤棋局持续到棋盘填满或者双方都无合法棋步可下时，棋盘上存在的哪一方的棋子多，哪一方获胜。

在讲解实现思路前要说明的是，计算机中数组下标是先行后列（先y轴后x轴），与我们在平时在平面直角坐标系表达位置的习惯先列后行（先x轴后y轴）不符，所以本次实验中为了符合直觉，方便运算，先创建两个函数转换这个差异。

屏幕上有字

描述已自动生成

* + 1. 规则①实现为：

在处理单次落子的函数bool RefreshChessBoard(int x, int y);中， 每次落子前判断该位置是否为空，不为空则落子失败，函数直接返回。

文本

描述已自动生成

* + 1. 规则③实现为：

落子前先判断该坐标是否是符合要求的可落子的点，其中，bool ChessPlaceableAt(int, int, int);函数会在上，下，左，右，左上，右上，左下，右下八个方向调用CPoint GetNextSameColorChessPos(int, int, int, int);进行穷举搜索，寻找与己方同色的棋子的坐标，以及判断两个坐标中间是否有可翻转的对方棋子。如果可以落子，则返回true，否则返回false。（上图中仅保留y轴方向的代码用于举例）

文本

描述已自动生成

其中，CPoint CReversiDlg::GetNextSameColorChessPos(int nx, int ny, int direction, int color)函数会返回指定方向上下一个同色棋子的坐标，如果找不到，则返回事先确定的的错误值

文本

描述已自动生成

经过以上两步判定后，可确认落子成功，修改数组中对应位置的值。

* + 1. 规则②实现为：

在处理单次落子的函数bool RefreshChessBoard(int x, int y);中，如果确认落子成功，则在上，下，左，右，左上，右上，左下，右下八个方向依次穷举会被翻转的对方棋子，并修改棋盘中对应位置的值为我方棋子，完成翻转。（图中保留两个方向的代码用于举例）

文本

描述已自动生成

* + 1. 规则④实现为：

在bool CReversiDlg::RefreshCurrentColor()函数中，每次成功落子后，都会先用对手颜色在整个棋盘的所有位置穷举，如果对手有可落子位置，则交换执棋方。如果对手无法落子，则用己方颜色在整个棋盘所有位置穷尽，如果己方有可落子位置，则己方继续落子，如果己方也无法落子，则返回false，结算本局游戏。

文本

描述已自动生成

文本

描述已自动生成

* + 1. 规则⑤实现为：

在void CReversiDlg::RefreshChessConut()函数中，会扫描棋盘所有位置，统计其颜色，保存在变量中，同时在界面底部的字符串中更新显示。如果双方都无法落子，需要判断胜负时，则会调用该函数更新变量中的统计数据，并结算。

文本

描述已自动生成

1. 图形化界面绘制
   1. 棋盘

棋盘每格大小为100\*100，棋盘左上角从界面左上角50\*50位置开始

文本

中度可信度描述已自动生成

在void CReversiDlg::OnPaint()函数中执行绘制

* 1. 棋子

void CReversiDlg::DrawAllChess()函数会穷举棋盘每个位置并且调用void CReversiDlg::DrawOneChess(int nx, int ny)函数绘制棋子

文本

描述已自动生成

void CReversiDlg::DrawOneChess(int nx, int ny)会先根据棋盘形状，计算出该坐标对应的棋盘格子的中心位置，以此为圆心，绘制半径为40的圆。圆形的填充颜色会根据该棋子的值，设定在Cbrush中。

文本

描述已自动生成

1. 消息处理

本次实验中主要对以下消息进行处理

文本

描述已自动生成

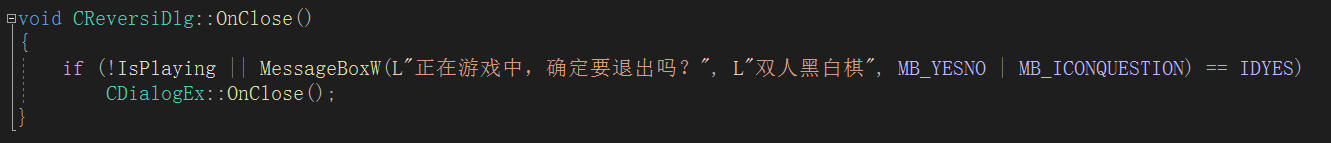
其中：

ON\_WM\_QUERYDRAGICON()完全使用系统默认设置不再赘述

ON\_WM\_PAINT()前文绘图部分已经讲解

* 1. ON\_WM\_CLOSE()在点击界面右上角X号时触发

如果此时正在游戏中，该函数会弹出框口，要求二次判断。如果不在游戏中或者二次确认，则会退出整个程序。



* 1. ON\_WM\_SETCURSOR()用于提示玩家当前执棋的是哪一方

该函数触发时，如果不在游戏中或者鼠标在棋盘范围外，则设置鼠标光标为默认光标。如果正在游戏中，且鼠标在棋盘上，则会根据当前执棋方的颜色设置鼠标为之前自定义的光标资源。当前执棋方为黑色，则鼠标光标显示为一个黑色的小棋子。白方同理。

文本

描述已自动生成

当按钮“开始游戏“被点击时，会触发 ON\_BN\_CLICKED(IDC\_START, &CReversiDlg::OnBnClickedStart)

如果此时正在游戏中，那么会弹出窗口要求二次确认。如果此时不在游戏中，或者二次确认要求重新开始。该函数会开始初始化一局新的游戏。

文本

描述已自动生成

首先会弹出一个窗口，要求玩家选择执黑棋还是白棋，并向服务器发送消息请求开始一局新的游戏。在这一过程中。会将连接服务器的情况进度会显示在下方字符串中。如果连接失败，则会显示错误信息。

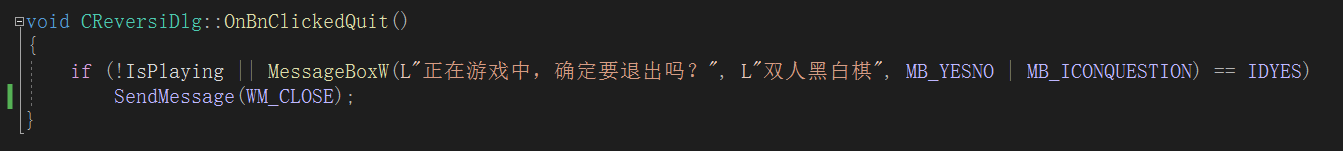
文本

描述已自动生成

之后会设置“开始游戏”按钮为“重玩”按钮，清空棋盘，设置游戏正在进行，并首次向服务器请求获取当前的棋盘。如果无法正确从服务器获取棋盘，也会在下方的字符串中提示，并弹窗提示错误的原因。在之后会设置“开始游戏”和“提示”的按钮为可用状态。获取到的黑白棋初始的棋盘有两黑两白共四个个棋子，最后会调用绘制棋盘的函数并重新统计棋子数量并显示在下方字符串中。

* 1. 当点击“退出”按钮时，会调用ON\_BN\_CLICKED(IDC\_QUIT, &CReversiDlg::OnBnClickedQuit)退出程序

如果此时正在游戏中，该函数会弹出框口，要求二次判断。如果不在游戏中或者二次确认，则会发出退出整个程序的消息。



* 1. 当点击“结束本局”按钮时，会调用 ON\_BN\_CLICKED(IDC\_ENDGAME, &CReversiDlg::OnBnClickedEndgame)

如果此时正在游戏中，该函数会弹出框口，要求二次判断。如果不在游戏中或者二次确认，则会调用void CReversiDlg::EndGame()函数结束本局游戏，而不是退出程序。

文本

中度可信度描述已自动生成

该函数会清空棋盘，设置游戏状态结束，重置各按钮的文字与状态。

文本

描述已自动生成

* 1. 点击“提示“按钮会触发 ON\_BN\_CLICKED(IDC\_PROMPT, &CReversiDlg::OnBnClickedPrompt)

该函数会扫描整张棋盘，找出所有当前执棋者可以落子的位置，并在对应位置画一个半径为10的红色圆

文本

描述已自动生成

清除提示的小红圆的函数为void CReversiDlg::CleanPrompt()该函数会设置画笔与画刷为背景色RGB（240,240,240），并在所有没有棋子的位置，都会画一个半径为11的圆，盖住之前的红色圆。

文本

描述已自动生成

* 1. 在程序页面任意位置点击都会触发ON\_WM\_LBUTTONUP()，该函数为本程序核心函数

该函数首先会将鼠标点击的位置计算转化为棋盘坐标，并调用落子函数 bool RefreshChessBoard(int x, int y);在对应棋盘坐标落子。如果落子函数判定该位置不能落子，则结束该函数。如果落子成功，则调用 bool RefreshCurrentColor();更新执棋方，并保存该函数返回的能否继续进行游戏的判断结果。。在落子过程中玩家可能点击过提示按钮，因此要消除提示的小红点。然后更新棋子数量的统计，重新绘制棋盘，之后还要根据新的执棋方更新鼠标光标的颜色。

文本

描述已自动生成

之后该函数会构造消息，将本次落子的情况告知服务器。服务器中的AI会根据玩家落子结果，计算自己的落子结果，然后返回新的棋盘数据，新的执棋方，并计算游戏能否继续进行。在与服务器通信的过程中，下方字符串会显示当前进度，并在出现错误时给出提示。然后根据服务返回的新的棋盘，重新绘制界面，更新棋子的数量并显示，以及更新光标的颜色。

文本

描述已自动生成

如果本函数之前的流程中判断双方都无字可下，则游戏结束，进入结算。根据双方棋子的数量判定胜负，然后调用EndGame()函数结束本局游戏。

屏幕上有字

描述已自动生成

3.2. 强化学习总体方案选择

在强化学习中，大多是基于 Q-learning 的算法。Q-learning的核心在于Q表格，通过建立Q表格来为行动提供指引，但当状态和动作空间非常庞大时，Q 表格也会随之变得无比庞大，因此，存储和查找整个 Q 表格将会变得十分困难甚至不可行。

因此，我们将使用 DQN 算法，这种算法将深度学习网络来替代 Q 表格，从该网络中获取下一步的决策 Q 值。这将带来两个好处：第一：不再需要存储有关整个状态空间的巨大 Q 表，而只需要存储深度学习的网络结构与参数，第二：相近的的输入会得到相近的输出，整个网络泛化能力将因此得到一定程度的提高。

1. Replay Buffer

之前所描述的一般的Q-Learning采用online update的方法进行更新，也就是伴随着学习过程的进行，每走一步（state变化一次）就会对Q网络进行一次更新，但是这种更新过于频繁，是导致不稳定情况出现的原因之一。

而由于 DQN 同时也是一个 off-policy 的算法，行为准则可以脱离实验环境而学习，因此利用该特性，DQN 中引入了 replay buffer。因此我们引入replay buffer来降低更新的频次。这样可以防止网络过快拟合不必要的经验，可以真正实现强化学习的延迟奖励的特征。

Replay buffer包括两个部分：行为准则通过在探索或行动获取经验后会将经存储到经验池中，而目标准则则通过从经验池中随机抽取经验来更新这个网络。

这样将会带来两个好处：

随机抽取，可以切断经验的相关性。

每一条经验可以重复学习多次，提高了经验的利用率。

在具体实现上，replay buffer中的条目（即为 transition pair）通过四元组（s，a，r，s\_） 来表示。其中 s 和 s\_ 表示 state，分别代表下一个状态和当前状态，a 即为 action，表示需要何种动作可以实现由 s 到 s\_ 的转移。r 即为 reward，通过该动作可以获得何种奖励。

同时 replay buffer 将会有容量限制，随着探索的不断进行，将会有越来越多的内容加入到 replay

buffer 中。当 replay buffer 填满之后，深度网络将从 replay buffer 中随机采样，将一个 transition pair 条目中的奖励（r）和加一个状态（s\_）送入目标网络中，网络将推理出下一个 Q 值（theta），并使用该值计算 loss，以 loss 最小化为依据更新 Q 网络中的各个参数。一直训练直到 Q 网络最终收敛。图一所示即为该强化学习模型的学习训练流程。

Diagram

Description automatically generated

图一.DQN算法流程

1. 深度学习部分网络结构

DQN 网络中的深度学习部分，即替换掉传统的 Q-learning 中 Q 表的部分，我们经过多次尝试后使用了如下的网络结构：

四个全连接层，分别为 128, 128-128, 128-64, 64-1，都使用 elu 作为激活函数（防止出现梯度爆炸在训练时出现对 0 取对数而产生 nan，同时也不会像 sigmoid 那样过于平缓而过于难以收敛。同时使用均方差作为损失函数。具体实现如下：

关键代码：

self.eval = keras.Sequential([

keras.layers.Dense(128, kernel\_initializer=keras.initializers.HeNormal(), input\_dim=self.state\_len+self.action\_len),

keras.layers.Dense(128, activation='elu', kernel\_initializer=keras.initializers.HeNormal()),

keras.layers.Dense(64, activation='elu', kernel\_initializer=keras.initializers.HeNormal()),

keras.layers.Dense(1, activation='elu', kernel\_initializer=keras.initializers.HeNormal())

])

self.eval.compile(

optimizer=keras.optimizers.Adam(learning\_rate=self.lr),

loss='mean\_squared\_error',

metrics=['accuracy'])

eval 网络结构和 target 网络相同。

3.3. 网络模块依赖导入与导出

1. 网络模块代码

文本

描述已自动生成

1. 导出为依赖库

电脑萤幕截图

描述已自动生成

此处将函数声明为C语言函数，避免C++编译器对函数名的处理造成最终导出的函数名与预设不符

1. 导入

电脑萤幕画面

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

1. dll文件与Release文件夹结构

文本

描述已自动生成

1. **实验结果**

**项目文件结构**

/Old\_vision 不能悔棋单机版源代码1，能悔棋的单机版源代码2，控制台测试网络模块的源代码

/Release 可以直接玩的单机版和对战版，其中单机版有悔棋功能

/Reversi 对战版游戏源代码，需要导入编译完成的网络模块，注意导入依赖的路径

/Reversi\_AI AI源代码，需要tensorflow与flask与flask\_cors

/Reversi\_interop 网络模块，需要下载.gitmodules里的libhv依赖库并导入。

头文件路径：/libhv/win64/include/hv 库文件路径：/libhv/win64/lib/Debug

**运行单机版：**

1. 运行/Release/Reversi\_offline.exe

2. 点击开始游戏

**运行对战版：**

* 1. 运行/Reversi\_AI/backend.py，启动AI服务器后端

电脑屏幕的截图

描述已自动生成

* 1. 运行/Release/Reversi\_online.exe进入主界面

表格

中度可信度描述已自动生成

* 1. 点击开始游戏

玩家选择黑色先手，棋盘初始化为两黑两白，下方“开始游戏”按钮文字替换为“重玩”，“提示”与“结束本局”按钮激活为可以被点击。

图表, 气泡图

描述已自动生成

* 1. 落子

玩家点击期棋盘中方格落子，期间，鼠标光标也会被替换为对应执棋方的颜色。点击“提示”按钮可以显示当前执棋方所有可以落子的位置

图表, 气泡图

描述已自动生成

* 1. 游戏期间服务器也会打印接口的请求日志

文本

中度可信度描述已自动生成

* 1. 一局游戏结束，根据黑白棋子数量结算胜负

图表, 形状, 气泡图

描述已自动生成