南開大學
Nankai University

计算机学院

高级程序语言设计期末报告

内嵌智能 AI 的 Qt 黑白棋小游戏

姓名：张书睿

学号：2010521

专业：计算机科学与技术

2022 年 5 月 6 日

目录

1 开发环境 **2**

2 作业要求 **2**

3 实验主要流程 **2**

3.1 搭建窗口界面 2

3.2 游戏逻辑设计 3

3.2.1 棋盘底层逻辑 3

3.2.2 AI 逻辑设计 4

3.2.3 算法的多线程设计 4

4 附加功能实现 **4**

4.1 音乐播放 4

4.2 游戏计时 4

4.3 投降 6

5 项目设计中遇到的困难 **6**

5.1 利用子线程修改主体变量时主线程崩溃 6

5.2 调用 Python 文件时，难以设计传参与读参的方式 7

5.3 原始版本的困难模式的 AI 下棋算法运行时，界面线程阻塞，导致游戏计时器无法刷新 7

6 实际运行演示 **7**

1 开发环境

实验配置条件如下表1。

表 1: 实验条件配置

CPU	Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz 2.21 GHz
OS	Windows10(X86)
IDE(Qt)	Qt Creator 4.8.0 (Enterprise)
IDE(Py)	PyCharm Community Edition 2021.1.1 x64

2 作业要求

学生自选题目，使用 C++ 语言完成一个图形化的小程序。

- 图形化平台不限，可以是 MFC、QT 等。
- 程序内容主题不限，可以是小游戏、小工具等。

基于题目要求，选择做黑白棋小游戏。项目 Git 链接如下：Git_Othello。程序总代码量约 1500 行 (c++ 为 1200 行，python300 行)。黑白棋小游戏简介：Reversi。

3 实验主要流程

黑白棋小游戏的设计主要包括三大部分：游戏界面的设计搭建，游戏内部逻辑的设计搭建和附加功能的设计搭建。

3.1 搭建窗口界面

游戏包含三个窗口：开始界面，选择难度界面和游戏棋盘界面，分别如下图3.1, 图3.2, 图3.3所示。设计时运用了设计师界面的多种功能，包括设计样式表，调整槽函数等等。

图 3.1: 开始界面

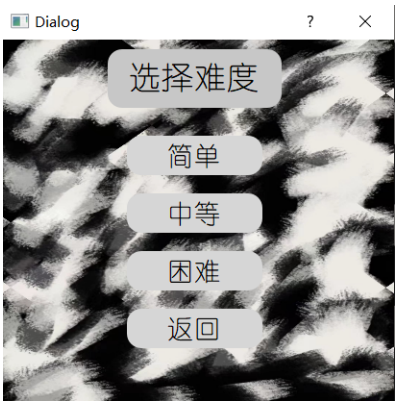


图 3.2: 选择难度界面

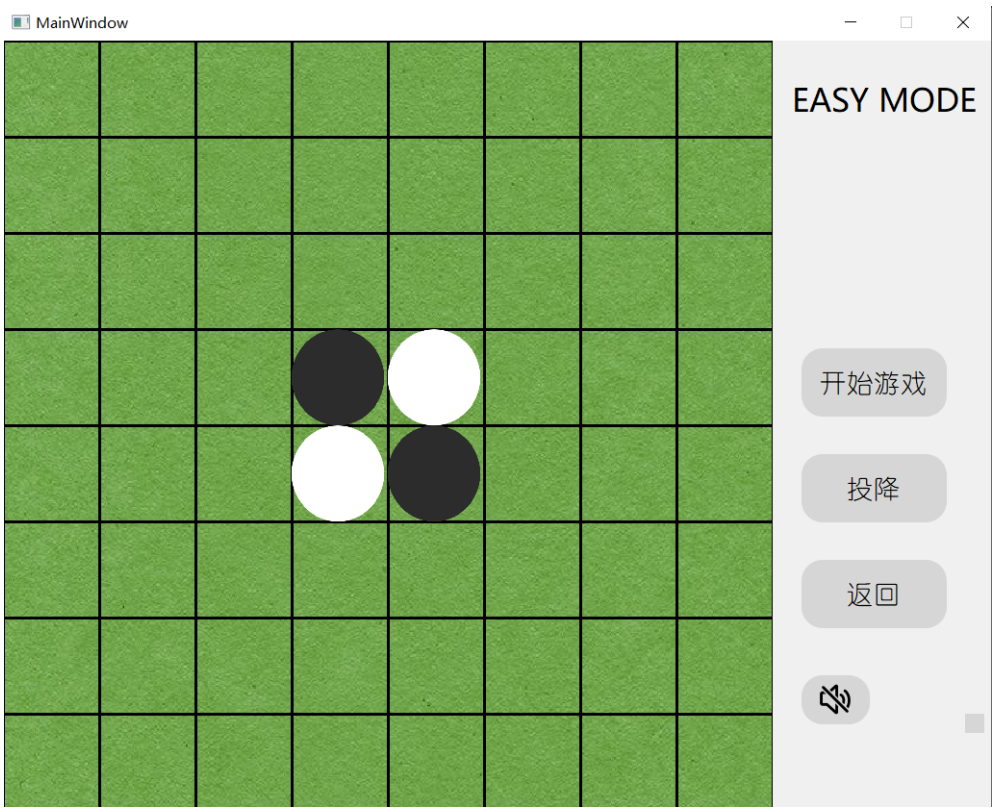


图 3.3: 游戏棋盘界面（简单模式）

3.2 游戏逻辑设计

游戏逻辑设计主要包含棋盘底层逻辑设计和 AI 逻辑设计。

3.2.1 棋盘底层逻辑

棋盘的主要操作包含下棋操作与翻转操作，其中，下棋操作隐含了检测已知可下位置的操作。检测可行操作和翻转操作都是通过向选定位置的八个方向（上，下，左，右，左上，左下，右上，右下）进行搜索来实现的。

下棋操作又可以分为人类玩家下棋与 AI 玩家下棋。

人类玩家：人类玩家的落子限定在能够下棋的位置（详情见黑白棋规则）。人类玩家落子的回合，paintevent 会将这些能够下棋的位置用虚线圆环标识出来；并且，棋盘设置了鼠标按下的监听器，如果落子位置合法，将会成功完成下棋与自动翻转的操作。

AI 玩家：由于本游戏可以选择难度，故不同 AI 的下棋策略与方式也略有不同。简单 AI 与中等 AI 都是基于简单逻辑，在主线程中实时进行推断然后落子的，而困难 AI 是基于自主编写的蒙特卡洛树搜索的 Python 程序（蒙特卡洛树搜索算法简介:MCTS），由子线程进行调用，等待并接收输出结果，最后落子。

3.2.2 AI 逻辑设计

AI 包含简单，中等和困难三种难度。

简单 AI: 简单 AI 的下棋逻辑仅仅基于随机算法。

中等 AI: 中等 AI 的下棋逻辑基于深度优先搜索 (DFS)。他将扫描每个合法的落子点，并以其为中心向之前提到的八个方向进行搜索，统计翻转的棋子数，再基于棋子落子点本身的含金量，给每个合法落子点一个分数，最终选取分数最高的点进行落子。

困难 AI: 困难 AI 的下棋逻辑基于蒙特卡洛树搜索 (Monte Carlo tree search, MCTS)，这是一种对抗搜索算法，算法内部的每次迭代操作包括选择，扩展，模拟和反向传播四步操作；由于进行多次模拟预测操作和反向传播操作，MCTS 倾向于得到更有利于自己的结果。由于使用 c++ 进行编写代码较为复杂，故采用 Python 进行编写，再导入 QT 项目，利用 c++ 调用 Python 的 API 进行操作以得到 AI 运行结果。

3.2.3 算法的多线程设计

为了保证界面展示的主线程不被堵塞以及鲁棒性，本游戏开启了多个线程，主要包括:

- **游戏运行线程：**主要运行正在执行的游戏，可以发送两种信号，分别用以告知主线程玩家的切换和游戏的结束。之所以采用发射信号的方式，是因为不可直接使用子线程去操作主线程中的主要数据变量（比如棋盘等），否则容易造成主线程崩溃。
- **AI 执行线程：**主要负责调用 Python API 操作，可以发送包含可落子点的信号。
- **计时器线程：**主要负责计时，发送一种信号，用以更新计时器界面。

4 附加功能实现

本游戏附加功能包含音乐播放, 游戏计时和投降。

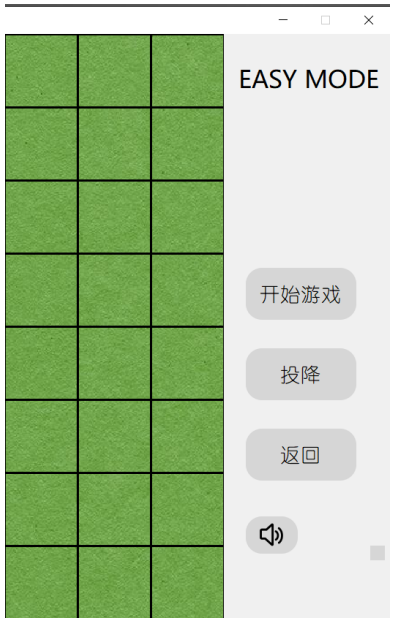
4.1 音乐播放

通过构建 QSoundEffect 对象实现播放, 可以通过按钮点击开启播放, 再次点击将取消。如图4(a)所示。激昂的音乐有助于增加下棋选手的紧张感。

4.2 游戏计时

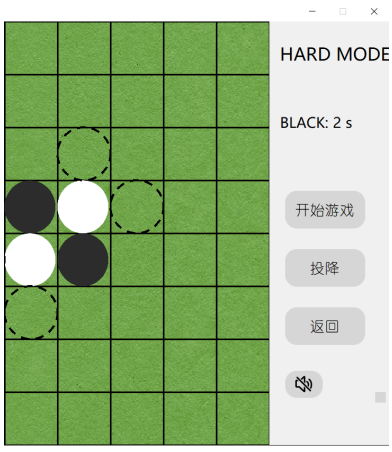
游戏进行时，计时器将对黑棋或白棋做思考的时间进行计时，在面板右侧进行显示，如图5(a)所示。时间的可感流逝，增加了游戏的火药味。

(a) 播放关闭状态

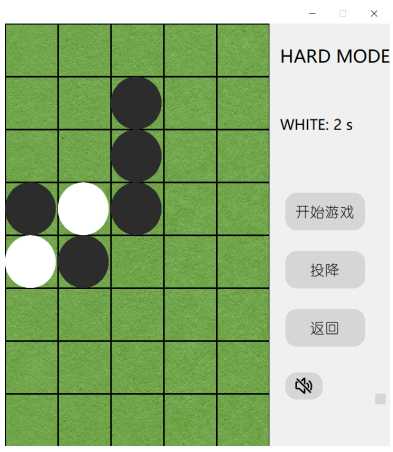


(b) 播放开启状态

图 4.4: 音乐播放演示



(a) 黑棋计时



(b) 白棋计时

图 4.5: 计时器演示

4.3 投降

点击投降即可立刻输给 AI，如图4.6，获得失败的痛苦。

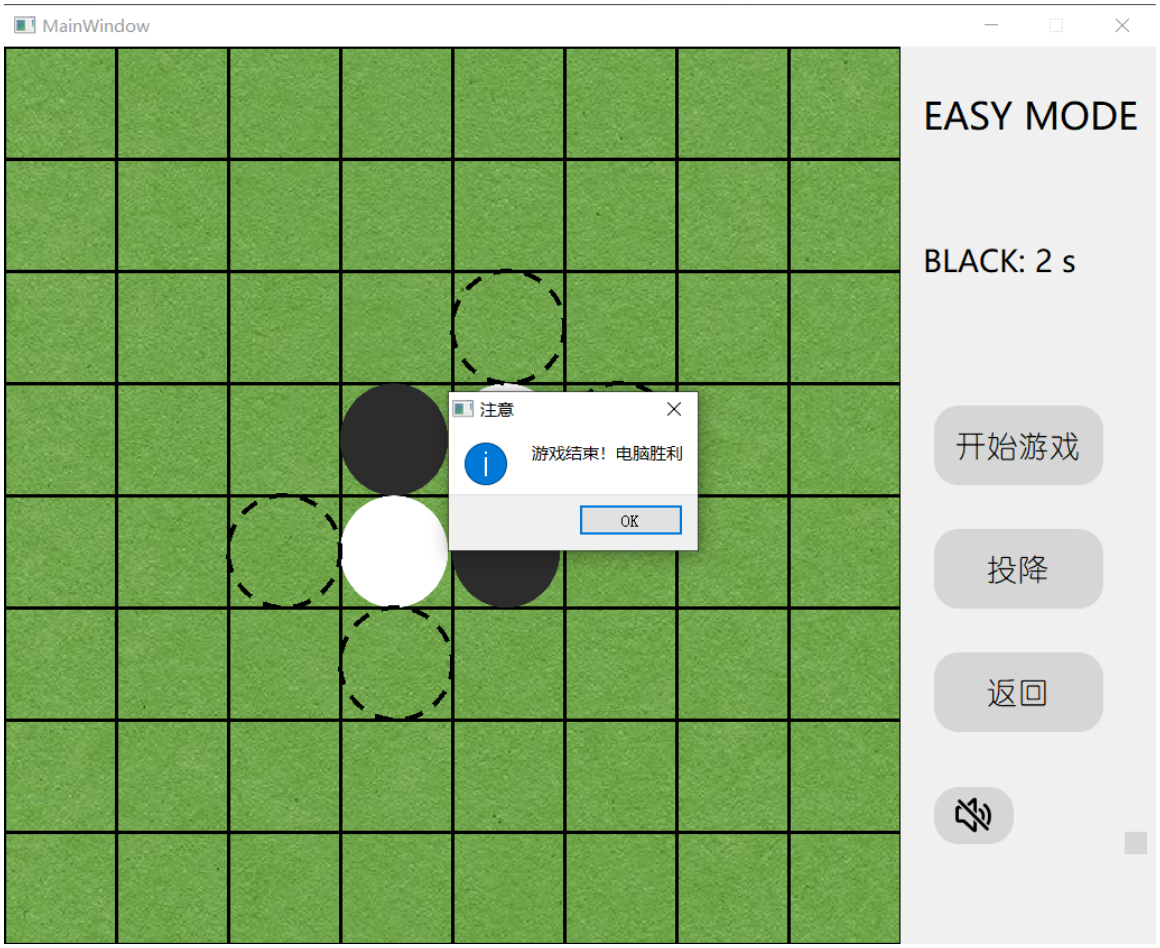


图 4.6: 游戏棋盘界面（简单模式）

5 项目设计中遇到的困难

遇到的困难主要包括以下几点。

5.1 利用子线程修改主体变量时主线程崩溃

主要原因：子线程与主线程可能在操作同一变量，而该变量会引起重绘事件的发生，进而导致主线程事件循环崩溃。

解决方法：操作其一，子线程通过发送信号，采用槽函数方式调用主线程中主体函数进行操作（使用 connet 连接时采用 auto 连接方式），这样就避免了事件循环的崩溃，维护了主线程鲁棒性。操作其二，为各个主体（包括 PaintEvent 也单独设置）设置各自的阅读当前棋盘合法落子点的 Qvector，防止同时操作同一 QVector 导致主线程崩溃。

5.2 调用 Python 文件时，难以设计传参与读参的方式

主要原因：原本的 AI 需要读取当前棋盘局面的 board 参数（一个自定义 Python 对象），然而 c++ 调用 Python API 时只能传入 int，double，const char等少数几种类型的基本变量，两者难以对接。

解决方法：为原本的 Py 文件单独设计一个 Func，接受 Qt 中由棋盘状态和 AI 棋子颜色构建字符串参数，Func 中实时创建一个 AIPlayer 对象，接受重新构建的棋盘 board 作为参数，最终输出选择的落子点。

5.3 原始版本的困难模式的 AI 下棋算法运行时，界面线程阻塞，导致游戏计时器无法刷新

主要原因：由于 MCTS 算法需要进行深度的搜索，故在设计算法时将搜索时间限定为了 8s；然而，如果将 MCTS 算法放在主线程执行，界面刷新等函数会在事件循环中被阻塞，导致 AI 思考时界面无法进行任何更新。

解决方法：单独为困难模式的 AI 设计一个线程类，在界面线程之外进行思考的操作，使得界面随时可以进行刷新。

6 实际运行演示

经过测试，程序能够正常运行，视频链接在Bilibili。