CWJ 大作业——四子棋游戏
实验报告

目录

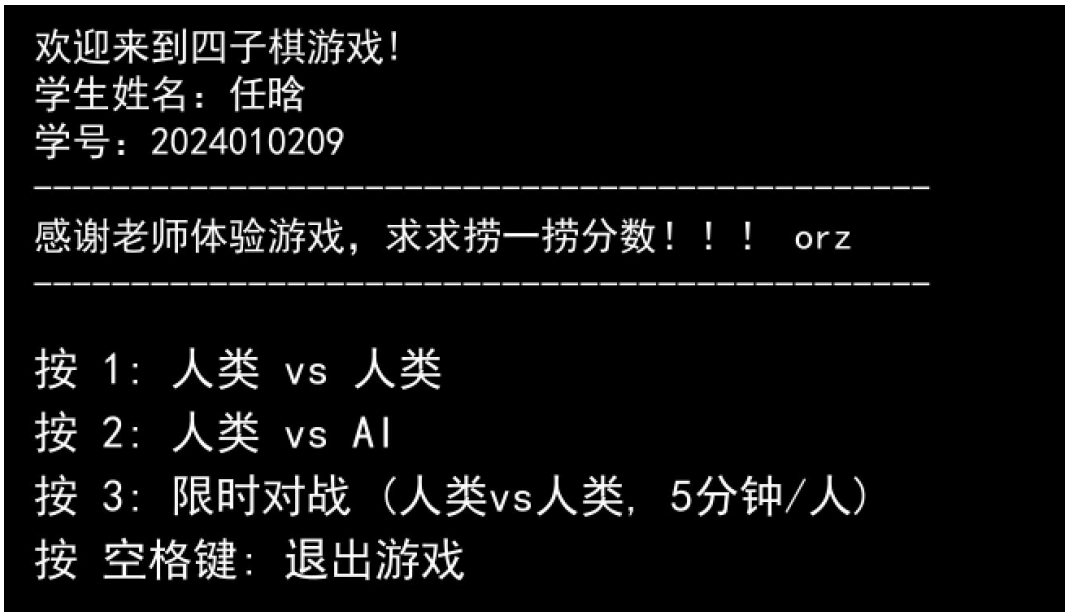
班级：机械 40
姓名：任晗
学号：2024010209
日期：2025-5-2

备注
IDE: Vscode -- version 1.99.3
Plugins： - Python @microsoft.com - Code runner @Jun Han
Compiler: Python 3.13
OS: Window 11 64 bit

一、简介与界面展示

首先根据 README.txt，requirements.txt 安装依赖项目并进入程序。

本次大作业的界面设计是 CLI 和 GUI 的混合使用，其中主菜单的设计更偏向 CLI，游戏界面的设计采用 GUI。我设计的四子棋游戏主要实现了如下图所示的三个功能：

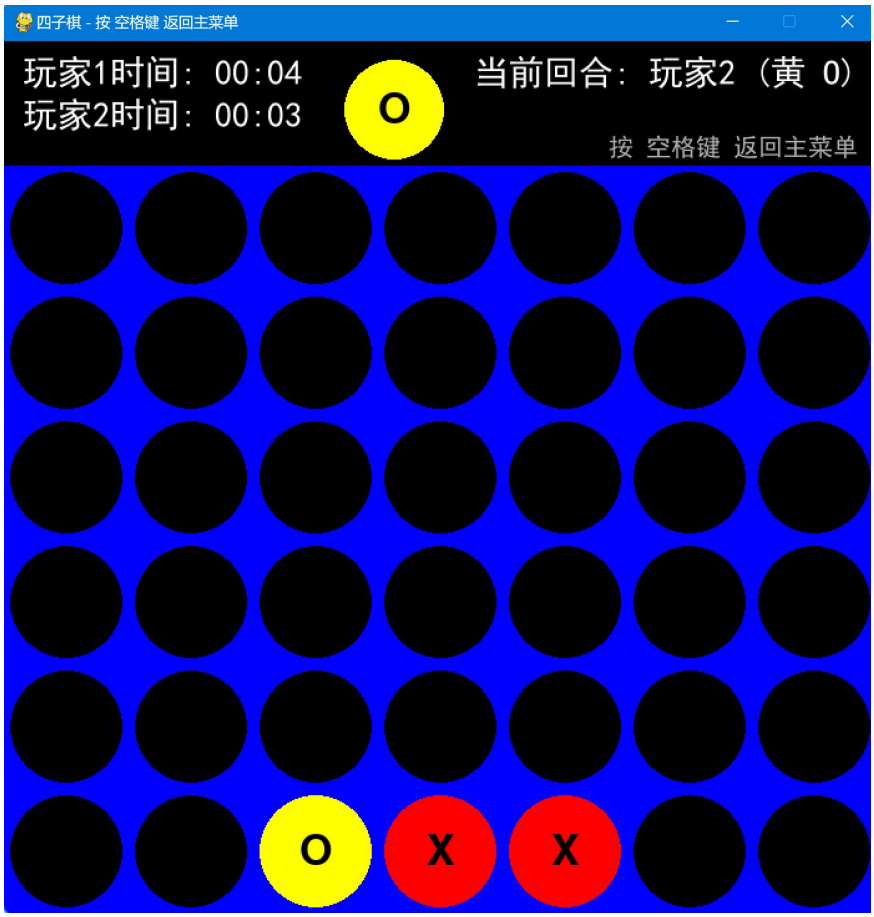


即：

- 玩家和玩家；
- 玩家和 AI；
- 限时的对战，其中每人回合时间总计不能超过 5 分钟

前两者也有正向计时的功能。

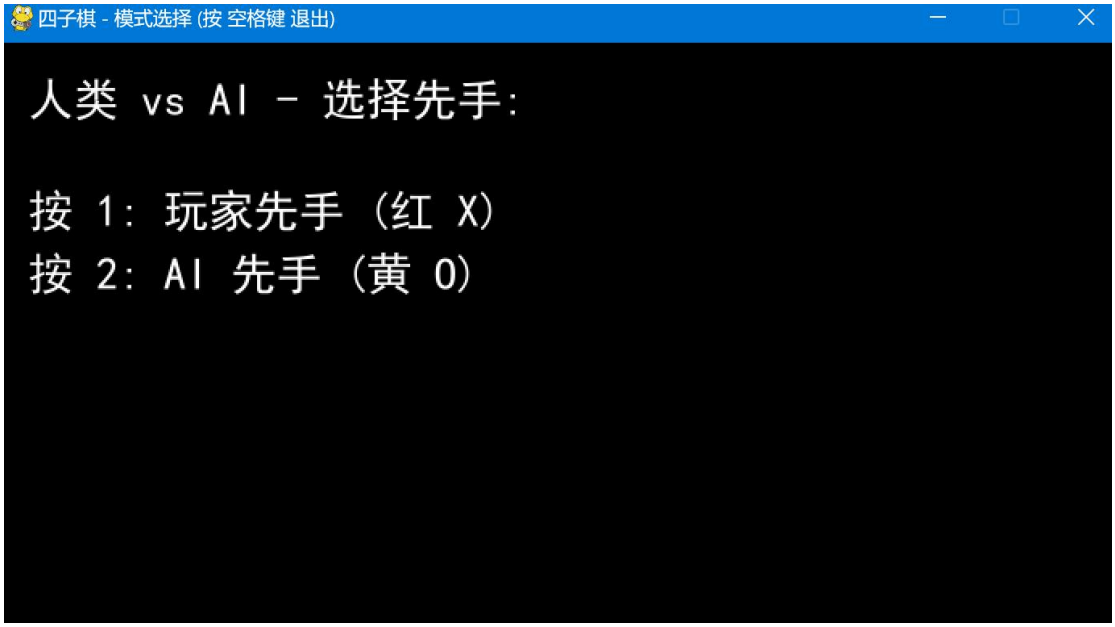
• 按 1 进入 ***Mode1：人类 vs 人类*** 后：



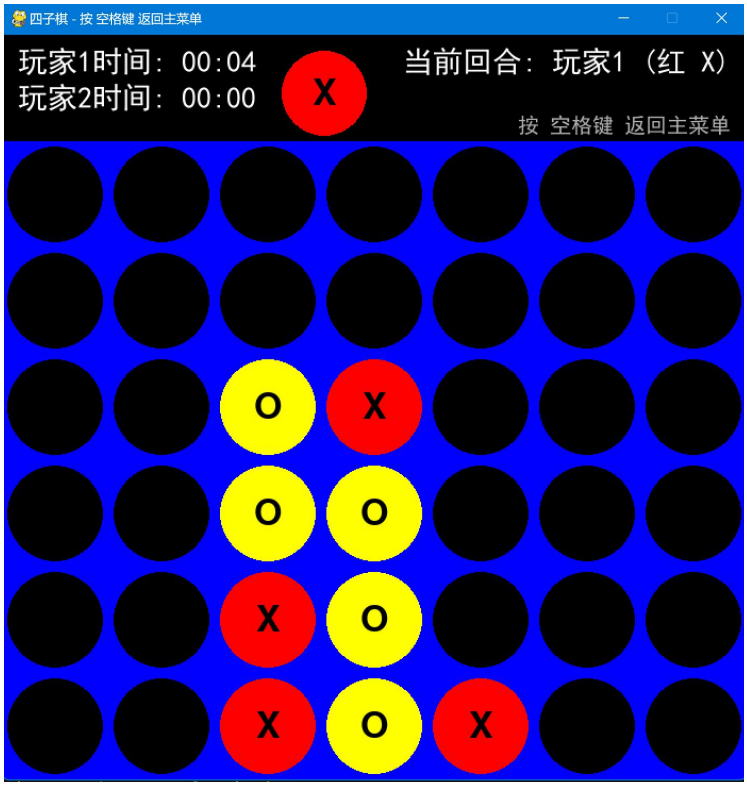
界面会显示 当前回合、两个玩家用时的正向计时、以及按空格键返回主菜单的提示语（标题栏也显示）

点击想要落子的对应列或对应列上方的空白处即可完成落子。

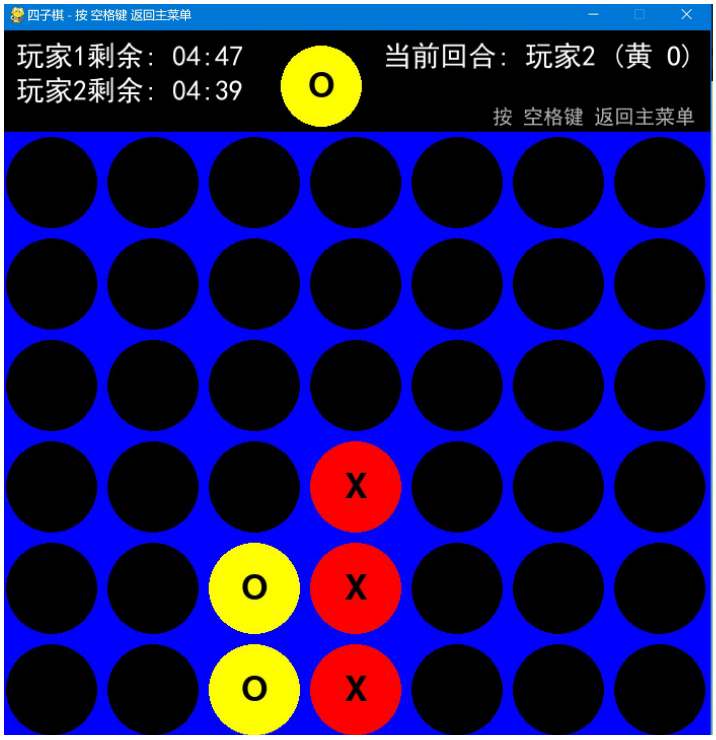
• 按 2 进入 ***Mode2：人类 vs AI*** 后：



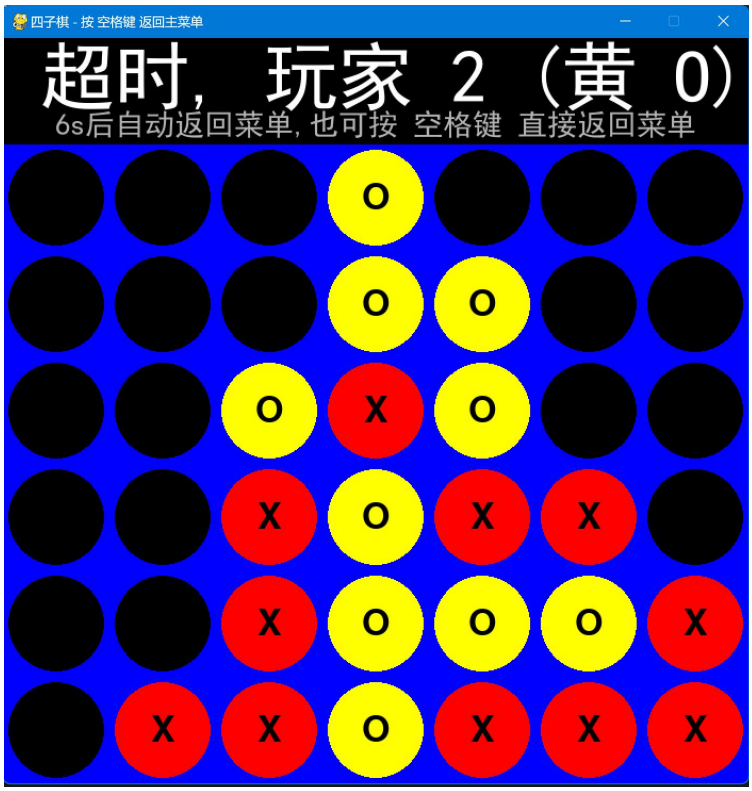
会提示玩家是选择先手还是后手，根据提示选择后同样可进入游戏界面。



• 按 3 进入 ***Mode3：限时对战（人类 vs 人类，5 分钟/人）*** 后：



两个玩家会进入 5 分钟的倒计时，如果超时会显示：（10s 后会自动返回主菜单，也可以直接按空格键退出）



二、代码和关键函数总览

2.1 程序整体框架

程序的整体框架采用经典的事件循环模式。程序启动后，首先进入模式选择界面。用户选择游戏模式后，程序进入游戏主循环。游戏主循环持续接收和处理用户输入（如鼠标点击、键盘按键）、更新游戏状态（落子、检查胜负、计时）、绘制游戏界面（棋盘、顶部信息）和处理 AI 的回合。游戏结束后，程序会显示结果并在一段时间后或用户操作后返回模式选择界面，形成一个可循环的游戏流程。

主要组成部分：

• 初始化阶段： 设置常量、全局变量、初始化 Pygame 和字体、配置窗口居中、设置 Tkinter 弹窗函数。

• 模式选择阶段 (mode_selection_screen): 负责显示游戏模式和人机对战先手选择菜单，根据用户输入设置全局游戏模式 (nGameMode) 和起始玩家 (nTurn)。

• 游戏主循环 (main function 的内层 while bGameRunning 循环): 负责一局游戏的进行，包括：

- 处理 Pygame 事件（退出、鼠标移动、鼠标点击、按键）。
- 根据用户点击或 AI 决策进行落子。
- 检查胜利、平局或超时。
- 切换回合。
- 更新计时（如果处于计时模式）。
- 绘制游戏界面（棋盘和顶部信息栏）。
- 处理游戏结束状态（显示结果，自动返回菜单）。

• 辅助函数模块： 包含棋盘操作、胜负判断、AI 逻辑、图形绘制、时间格式化等功能的函数。

• 退出处理： 在用户关闭窗口或按下退出键时，调用 Tkinter 弹窗显示感谢信息并退出程序。

2. 2 关键函数

• main()

作用： 程序的入口点，控制整个游戏的流程。它包含一个主循环，负责在模式选择和游戏进行之间切换，管理游戏状态的初始化、事件处理、AI 回合、绘制和游戏结束判断。

• mode_selection_screen(oScreen)

作用： 显示游戏模式选择菜单和人机对战的先手选择菜单。根据用户的键盘输入设置全局变量 nGameMode (游戏模式) 和 nTurn (起始玩家)，然后返回到主循环开始游戏。

• create_board()

作用： 创建并返回一个表示棋盘状态的二维列表，初始化所有位置为 PIECE_EMPTY (空)。

• is_valid_location(a_a_nBoard, nCol)

作用：检查指定的列 nCol 是否可以合法落子（即该列最顶部是否还有空位）。

• get_next_open_row(a_a_nBoard, nCol)

作用： 在指定的列 nCol 中，从底部向上查找并返回第一个空行的行号。

• drop_piece(a_a_nBoard, nRow, nCol, nPiece)

作用： 将指定的棋子 nPiece 放置到棋盘的指定行 nRow 和列 nCol 上。

• check_win(a_a_nBoard, nPiece)

作用： 检查棋盘 a_a_nBoard 上是否存在由指定棋子 nPiece 组成的连续 N_WIN_LENGTH 个棋子（水平、垂直或对角线）。

• is_board_full(a_a_nBoard)

作用： 检查棋盘是否已经完全填满（用于判断平局）。

• get_valid_locations(a_a_nBoard)

作用： 返回当前棋盘上所有可以合法落子的列的列表。

• evaluate_window(aWindow, nPiece)

作用： (AI 相关) 评估一个包含 N_WIN_LENGTH 个位置的子窗口对指定棋子 nPiece 的潜在价值（如潜在的连子机会或被对手连子的风险）。

• score_position(a_a_nBoard, nPiece)

作用： (AI 相关) 计算整个棋盘状态对指定棋子 nPiece 的总评分，考虑了中心列的优势和各种潜在的连线组合。

• pick_best_move(a_a_nBoard, nPiece)

作用： (AI 相关) AI 的核心决策函数。遍历所有合法落子列，模拟在每个列落子后的棋盘状态，使用 score_position 评估模拟后的分数，并返回得分最高的合法落子列。

• draw_board(oScreen, a_a_nBoard)

作用： 在 Pygame 窗口 oScreen 上绘制当前棋盘（包括背景网格和已落下的棋子）。

• draw_top_bar(oScreen, nTurn, nMouseX, bGameOver, oWinnerTextRect)

作用： 在 Pygame 窗口 oScreen 的顶部区域绘制游戏信息，包括当前回合、玩家时间、鼠标悬停的棋子提示（游戏未结束时）、胜利/平局消息和自动返回菜单倒计时（游戏结束时）。

• show_message_popup(title, message)

作用： 使用 Tkinter 库创建一个通用的消息弹窗，用于显示游戏中的提示信息（如列满）或结束消息。

三、功能模块的设计和实现思路

程序根据功能被划分为多个逻辑模块，通过函数实现相应的功能：

3.1 基础功能及实现效果

3.1.1 初始化与常量定义

• 设计：

定义了游戏所需的各种常量，包括棋盘尺寸、连子获胜长度、玩家标识、棋子状态、颜色、界面尺寸、模式类型、时间限制等。这些常量使得代码更具可读性和易维护性。同时，初始化 Pygame 环境和字体，并设置窗口居中。

• 实现：

- 使用全大写字母定义常量，如 N_ROWS, N_COLS, COLOR_RED, MODE_H_VS_H 等。

- 定义了多个全局变量 (nGameMode, nTurn, fTurnStartTime 等) 来存储游戏状态，并在需要修改时使用 global 关键字。

- pygame.init() 初始化 Pygame。

- os.environ['SDL_VIDEO_CENTERED'] = '1' 设置环境变量使 Pygame 窗口居中。

- pygame.font.SysFont 或 pygame.font.Font 加载字体，使用 try...except 块处理字体加载失败的情况，提高程序的鲁棒性。

3.1.2 游戏逻辑模块

• 设计：

负责维护棋盘状态，判断落子的合法性，找到下一行可落子的位置，执行落子操作，并判断游戏是否达到胜利或平局状态。

• 实现：

create_board(): 使用二维列表 (a_a_nBoard) 表示棋盘，初始化所有位置为 PIECE_EMPTY。

is_valid_location(a_a_nBoard, nCol): 检查指定列 nCol 是否还有空位（即第一行 a_a_nBoard[0][nCol] 是否为 PIECE_EMPTY）。

get_next_open_row(a_a_nBoard, nCol): 从最底行向上查找指定列 nCol 的第一个空行。

drop_piece(a_a_nBoard, nRow, nCol, nPiece): 在指定行 nRow、列 nCol 放置棋子 nPiece。

check_win(a_a_nBoard, nPiece): 检查棋盘上是否存在由 nPiece 组成的连续 N_WIN_LENGTH 个棋子的情况（水平、垂直、主对角线、副对角线）。

is_board_full(a_a_nBoard): 检查棋盘是否已满（即所有列的第一行都不为空）。

3.1.3 游戏状态表示

• 算法：

使用一个二维列表（矩阵）来表示棋盘状态。列表的每个元素存储一个整数，代表该位置的棋子状态（空、玩家 1 棋子、玩家 2 棋子）。

• 实现：

变量 a_a_nBoard 是一个 N_ROWS x N_COLS 的列表，例如 a_a_nBoard[r][c] 表示第 r 行、第 c 列的棋子状态。

3.1.4 胜利条件判断 (Brute-Force Search)

• 算法：

check_win 函数采用了一种暴力搜索 (Brute-Force Search) 的方法来判断是否存在四子连线。它不是基于复杂的游戏树搜索，而是直接遍历棋盘上所有可能的连线起点，检查从该起点开始的四个位置是否都属于同一个玩家的棋子。

• 实现：

水平检查： 遍历每一行，然后在每一行中从第一列开始，每次移动一格，检查当前位置及其右边三个位置是否颜色相同。循环范围确保不会越界 (N_COLS - (N_WIN_LENGTH - 1))。

垂直检查： 遍历每一列，然后在每一列中从第一行开始，每次移动一格，检查当前位置及其下方三个位置是否颜色相同。循环范围确保不会越界 (N_ROWS - (N_WIN_LENGTH - 1))。

主对角线检查 (左上到右下): 遍历棋盘的左上区域，检查从当前位置开始，向右下方移动的四个位置是否颜色相同。循环范围限制在能够形成主对角线的区域内。

副对角线检查 (右上到左下): 遍历棋盘的右上区域，检查从当前位置开始，向左下方移动的四个位置是否颜色相同。循环范围限制在能够形成副对角线的区域内。

在每种检查中，使用 Python 内置的 all() 函数结合生成器表达式简洁地判断连续四个位置是否满足条件。

3.2 创新功能及实现效果

3.2.1 AI 模块：使用贪心算法进行决策

• 算法：

本项目的 AI 采用了一种基于启发式评估的简单贪心算法。它不进行深度的游戏树搜索（如 Minimax 或 Alpha-Beta 剪枝），而是对当前棋盘状态下所有合法落子后的 *下一步* 棋盘进行静态评分，并选择得分最高的落子位置。

• 实现：

get_valid_locations(a_a_nBoard): 返回当前棋盘上所有可以合法落子的列的列表。

evaluate_window(aWindow, nPiece): 定义了一个评分函数，用于量化一个包含四个位置的窗口的价值。不同的棋子组合（如 3 连子 + 1 空位、2 连子 + 2 空位、对手 3 连子 + 1 空位等）被赋予不同的分数。己方形成连子潜力得分高，对手形成连子潜力得分低（负分）。

score_position(a_a_nBoard, nPiece): 评估整个棋盘状态对给定棋子 nPiece 的总分数。通过遍历棋盘上所有可能的水平、垂直、对角线窗口，并累加每个窗口的 evaluate_window 分数。此外，增加了对中心列的偏好评分。遍历整个棋盘的所有水平、垂直和对角线上的“窗口”（四个连续位置），累加每个窗口的 evaluate_window 分数，并加上中心列的偏好分数，得到当前棋盘状态的总评分。

pick_best_move(a_a_nBoard, nPiece):

这是 AI 做出决策的核心函数。

- 获取所有合法落子列 (get_valid_locations)。

- 初始化一个较低的“最佳分数”。

- 遍历每一个合法落子列。

- 对于每个合法列，创建一个当前棋盘的临时副本。

- 在临时副本上模拟当前 AI 玩家在该列落子。

- 使用 score_position 函数评估模拟落子后临时棋盘的状态分数。

- 如果当前落子的分数高于记录的最佳分数，更新最佳分数和最佳落子列。

- 在所有合法列评估完成后，返回得分最高的落子列。如果多个列得分相同，则返回第一个找到的最高分列（或者像代码中那样，初始化时随机选择一个合法列作为最佳列，可以在平分时保持随机性）。

3.2.2 图形绘制模块

• 设计：

使用 Pygame 库负责将游戏状态可视化地呈现在屏幕上，包括棋盘、棋子、顶部信息栏等。

• 实现：

draw_board(oScreen, a_a_nBoard): 绘制棋盘网格（蓝色区域和黑色空洞）以及棋盘上已落下的棋子（红色 'X' 或黄色 'O'）。棋盘绘制区域从 TOP_BAR_HEIGHT 开始。

draw_top_bar(oScreen, nTurn, nMouseX, bGameOver, oWinnerTextRect): 绘制顶部的操作信息栏。在游戏进行中，显示当前玩家、玩家时间、鼠标悬停位置的提示棋子、返回菜单提示。游戏结束后，显示胜利/平局消息和自动返回菜单的倒计时。

format_time(fSeconds): 辅助函数，将总秒数格式化为 "MM:SS" 的字符串形式。

3.2.3 UI/菜单模块

• 设计：

提供用户友好的模式选择界面，引导用户开始不同类型的游戏。

• 实现：

mode_selection_screen(oScreen): 显示模式选择菜单。根据 current_selection_mode 变量在主菜单和人机先手选择菜单之间切换。绘制标题和选项文本，监听键盘输入 (K_1, K_2, K_3, K_SPACE, K_r) 来设置 nGameMode 和 nTurn，并在确定模式后退出循环并重设窗口大小。

show_message_popup(title, message): 使用 Tkinter 创建通用的消息弹窗，用于显示提示信息（如列满提示）和结束消息。弹窗居中显示，并始终在最上层。

show_exit_popup(): 调用 show_message_popup 显示程序退出时的特定感谢消息。

3.2.4 时间管理模块

• 设计：

在限时模式下，需要精确记录和显示每位玩家已用的时间，并判断是否超时。

• 实现：

fTurnStartTime: 全局变量，记录当前回合开始的时间戳。

fPlayer1TotalTime, fPlayer2TotalTime: 全局变量，累加玩家的总用时。在玩家落子或返回菜单时，计算当前回合持续时间 (time.time() - fTurnStartTime) 并累加到对应玩家的总时间。

draw_top_bar 函数中根据 nGameMode == MODE_TIMED_H_VS_H 计算并显示剩余时间。

fGameEndTime: 记录游戏结束时间戳，用于实现自动返回菜单的延迟

在游戏主循环中，检查玩家总时间是否超过 F_TIME_LIMIT_SECONDS，如果超过则判定超时并结束游戏。

3.2.5 Tkinter 弹窗实现

• 设计：

使用 Tkinter 库在 Pygame 窗口之上创建简单的信息弹窗，用于提示用户或显示游戏结束消息。

• 实现：

show_message_popup: 创建一个隐藏的 Tkinter 根窗口 (popup_root.withdraw())，然后在该根窗口上创建一个顶层窗口 (tk.Toplevel) 作为实际弹窗。设置弹窗标题、置顶属性、计算并设置弹窗在屏幕中央的位置。使用 tk.Label 显示消息文本，tk.Button 作为确认按钮，点击按钮时销毁弹窗和根窗口。popup.wait_window(popup) 阻塞程序，直到弹窗被关闭。

这种实现方式避免了 Tkinter 主事件循环与 Pygame 事件循环冲突的问题，可以在 Pygame 运行时弹出 Tkinter 窗口。

3.3 鲁棒性

·处理已满列的情况：

列已满的弹窗提示

- **处理非法列选择：**

由于对战界面是基于 GUI 的点击功能，所以无法选择非法列

- **处理非预期格式的输入：**

同理，也不会有非预期格式输入的问题

五、（附加）调试中出现的问题

将 pygame 的窗口最小化后再点开时会出现下图所示状况，只能显示部分界面，不能完整显示棋盘。

经网上冲浪查询得知，造成这种问题的根本原因是 Pygame 的绘制是“即时”的，而不是持久的。当使用 pygame.draw.rect, pygame.draw.circle, oScreen.blit() 等函数将图形绘制到 Pygame 的 Surface (通常是主显示窗口 oScreen) 上时，这些绘制操作是修改了 Surface 的像素数据。然后，调用 pygame.display.update() 或 pygame.display.flip() 命令图形系统将这个 Surface 的内容复制到屏幕上，使之可见。

但是，Pygame 的 Surface 不会自动记住或保留它上面的绘制历史。当窗口被最小化然后还原时，操作系统可能会处理窗口的内容，而 Pygame 底层（SDL 库）的图形上下文或 Surface 的内容可能被丢弃或标记为需要重新绘制。

在游戏主循环 (while bGameRunning:) 中：

draw_top_bar 函数被放在了 while bGameRunning: 循环的开头，这意味着它在每一帧都会被调用。所以当还原窗口时，顶部的操作栏会立即被重绘，显示当前的信息。

draw_board 函数主要在以下时机被调用：游戏刚开始初始化棋盘后；玩家成功落子后；AI 成功落子后。

它并没有在游戏主循环的每一帧都被无条件调用。

而当最小化窗口再还原时：操作系统让窗口重新可见，Pygame 的事件循环恢复运行，draw_top_bar 因为在循环开头，每一帧都会被执行，所以顶部区域被正常绘制。但是，除非正好在还原窗口后立即落子（手动或 AI），否则 draw_board 函数不会被再次调用来重绘棋盘区域。由于棋盘区域没有被重新绘制，这部分窗口的内容可能显示的是最小化前的旧内容（如果被丢弃，则可能是空白或不正确的显示），而不是当前的棋盘状态。

解决方案：

最简单的解决办法是确保棋盘绘制函数 (draw_board) 在游戏主循环的每一帧都被调用。这样，无论窗口是因为最小化还原、被其他窗口遮挡后重新暴露，还是其他任何原因导致需要刷新，棋盘都会在下一帧被正确重绘。

修改 main 函数中的游戏主循环部分：

```
569
570         # 单局游戏循环
571         bGameRunning = True
572         # 设置一个标志，表示当前局游戏正在运行。
573
574         while bGameRunning:
575         # 进入内层循环，执行一局游戏，直到 bGameRunning 变为 False。
576             oClock.tick(60) # 控制帧率
577
578             # 在绘制顶部栏之前先更新计时，确保实时性
579             draw_top_bar(oScreen, nTurn, nMouseX, bGameOver, oWinnerTextRect)
580             draw_board(oScreen,a_a_nBoard)
581
582             # 处理自动返回菜单
583             if bGameOver and fGameEndTime is not None:
584                 if time.time() - fGameEndTime >= F_AUTO_RETURN_DELAY:
585                     # 达到延迟，结束本局循环，返回主菜单
586                     bGameRunning = False
587                     continue # 跳过本帧剩余逻辑 (重要!)
588
```

在主函数的 while 循环中、if 前加上 draw_board(oScreen,a_a_nBoard)，确保每一帧都绘制。（上图代码中的 580 行）