围棋（人人对战版）说明文档

围棋（人人对战版）规则介绍

围棋文化起源于中国，博大精深，历史悠久。现有的棋盘种类多样，包括 15 路棋盘、17 路棋盘和 19 路棋盘。本围棋程序以隋唐时期定型的 19 路棋盘为模板。同时，对一些复杂的围棋规则进行了简化，具体如下：

①该款围棋游戏中，分黑白两种颜色的棋子，下棋顺序为黑先白后、交替落子，双方每回合落一颗棋子，禁止悔棋；

②棋子只能落在棋盘的固定点上，同一点不能落两颗及以上棋子，且棋子不能相互替代；

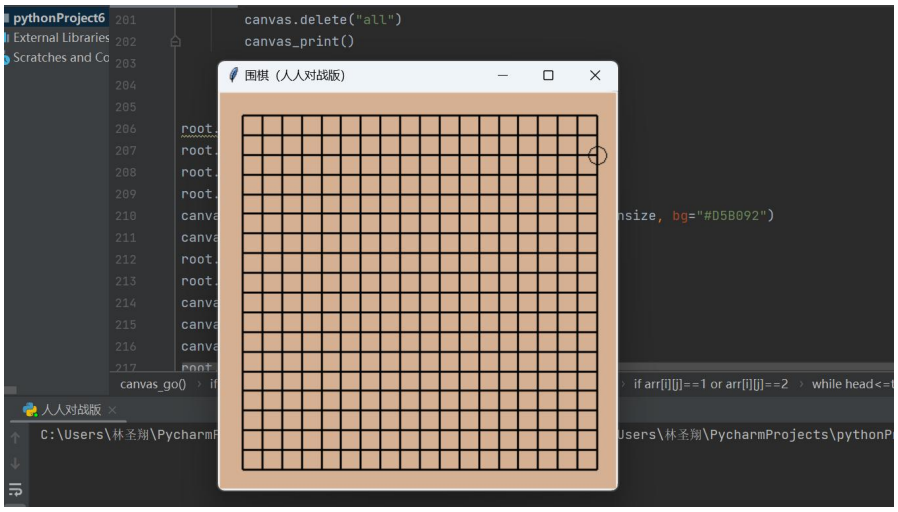
③按棋子的气判断棋子的去留。棋子被提走后，该位置不得下与与提走棋子相同色的棋子；

④双方思考到落子时间不得超过 1 分钟，超出者输，程序自动判对方胜利；

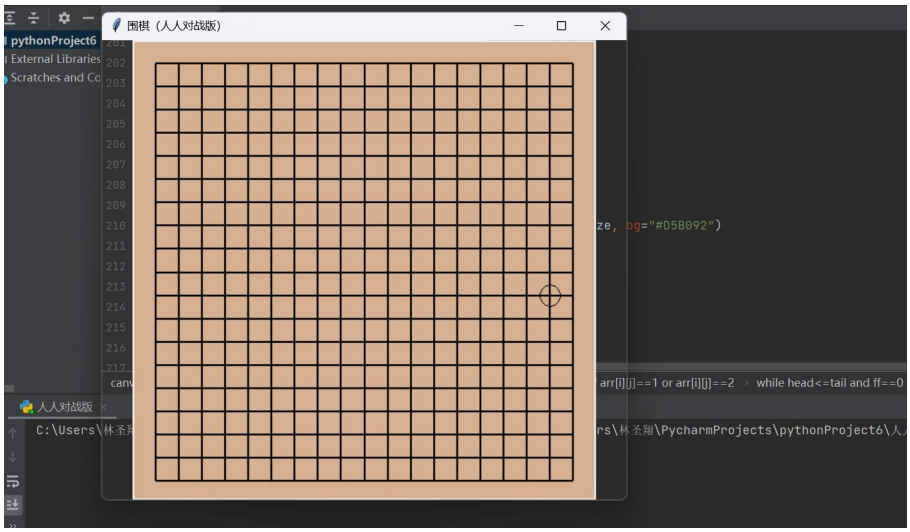
⑤若无超时的情况出现，程序采用中国大陆围棋规则中的数子法判定双方胜负，即对任意一方的活棋和活棋围住的点以子为单位进行计数。

围棋（人人对战版）使用说明

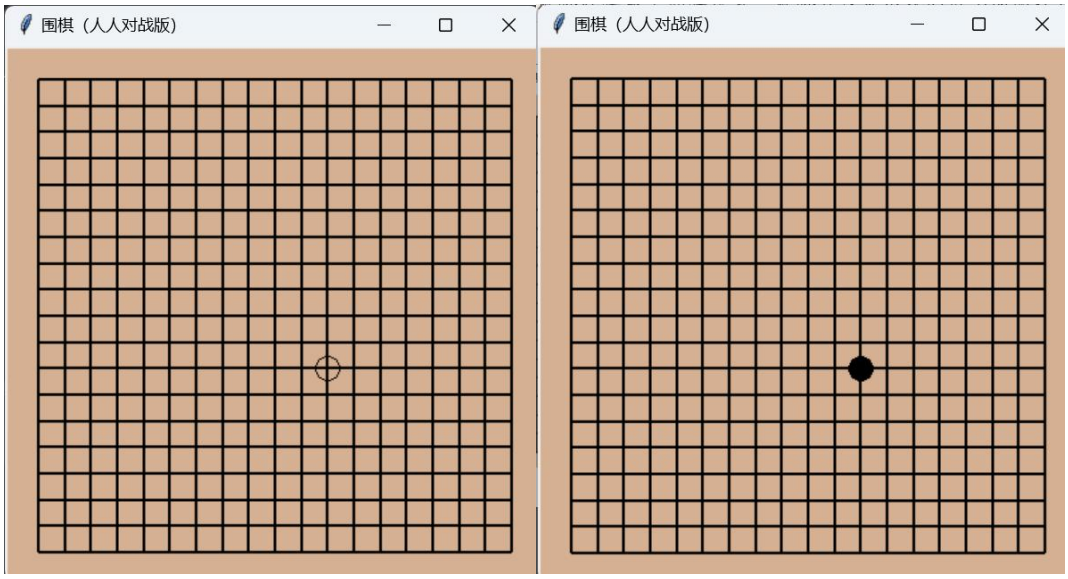
①打开程序，游戏窗口将以默认大小出现。



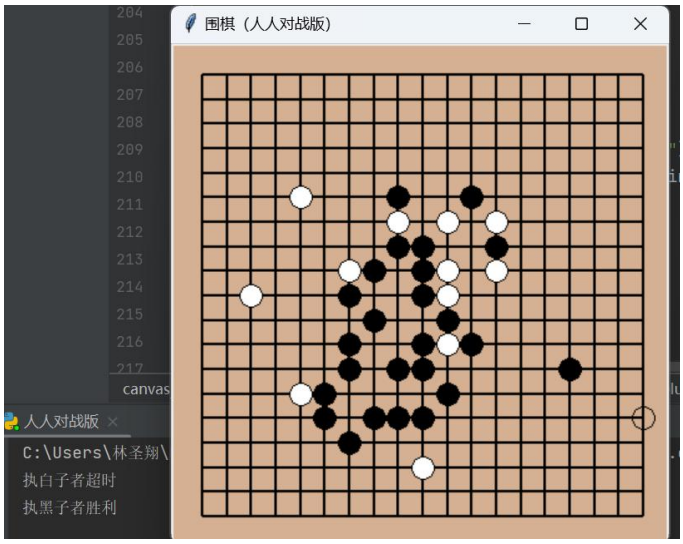
②玩家可以通过拉动边框的方式将棋盘大小调整至合适位置



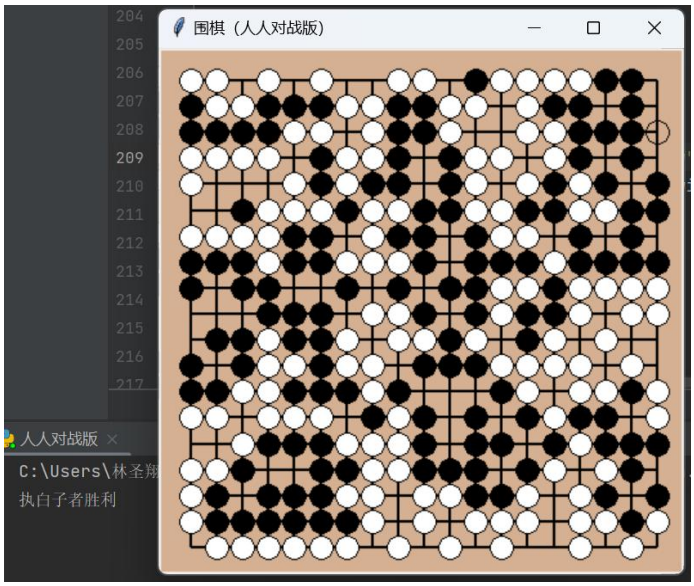
③用鼠标在棋盘上选择位置，鼠标经过处出现的透明圆圈即为待定落点位置，随后单击左键，完成落子



④超时判负情况判定



⑤最终胜负情况判定（无超时的情况下）



值得一提的是，本款游戏设计时充分考虑到了双方玩家通过比赛实现相互学习这一目

的。故比赛结束后，程序不会自动关闭，玩家和观众可以通过最后的局面品味棋局之妙，同时，不服输的那一方可以继续战斗（不过已经无力回天的，无数对方选手胜利的信息将告诉你这一事实）。

围棋（人人对战版）设计思路及代码实现

该款围棋小游戏能够实现自动提字、判定胜负和判定超时等功能。该程序运用 Python 的 Tkinter 模块，通过自定义 go_printline(pos1,pos2)、canvas_print()、canvas_resize(event)、canvas_dot(event)、canvas_go(event)等五大函数组成。

主要函数设计

函数名	主要功能
go_printline(pos1,pos2)	输出一条线
canvas_print()	画棋盘主体及其上已有的点
canvas_resize(event)	改变窗体大小时重绘棋盘
canvas_dot(event)	移动鼠标时实时绘制棋子的备选位置
canvas_go(event)	实现程序的核心功能，能够实现落子、自动提子、判定胜负和判定超时等多项功能

①一些参数的初始化

```
from tkinter import *
import time
import os

window_minsize = 400                    # 窗体最小大小
go_row = 18                             # 棋盘最大行数（本程序默认第一行为0，故共有19行）
go_column = 18                          # 棋盘最大列数（本程序默认第一列为0，故共有19列）
go_boundary = 25                        # 棋盘边界空出的空间大小
go_width = 2                            # 棋盘绘制的线条粗细
go_dot = 10                             # 棋子的大小

root = Tk()
canvas = Canvas(root)
```

②go_printline(pos1,pos2)函数的实现

```
def go_printline(pos1, pos2):        # 画一条线
    canvas.create_line((pos1[0]+go_boundary, pos1[1]+go_boundary), (pos2[0]+go_boundary,pos2[1]+go_boundary), width=go_width)
    # 由于本程序需实现窗口大小的调节，故直线的起始位置和结束位置是动态的，无法通过简单的定值实现
```

③canvas_print()函数的实现

该函数主要分为两大部分：前半部分主要实现的是对棋盘的绘制，后半部分是绘制棋盘上已有的点。

```
def canvas_print():                     # 画棋盘主体
    [w, h] = [canvas.winfo_reqwidth()-go_boundary*2,canvas.winfo_reqheight()-go_boundary*2]
    go_printline((0, 0), (w, 0))
    go_printline((0, 0), (0, h))
    go_printline((w, 0), (w, h))
    go_printline((0, h), (w, h))
    for i in range(1, go_row):
        go_printline((1.0*w/go_row*i, 0), (1.0*w/go_row*i, h))
    for i in range(1, go_column):
        go_printline((0, 1.0*h/go_column*i), (w, 1.0*h/go_column*i))
```

```
for i in range(go_row+1):
    for j in range(go_column+1):
        if arr[i][j] == 1:
            [w, h] = [canvas.winfo_reqwidth()-go_boundary*2,canvas.winfo_reqheight()-go_boundary*2]
            [printx, printy] = [1.0*i*w/go_row+go_boundary,1.0*j*h/go_column+go_boundary]
            dot_size = 1.0*go_dot*w/400
            canvas.create_oval(printx-dot_size,printy-dot_size, printx+dot_size,printy+dot_size, outline="black", fill="black")
        elif arr[i][j] == 2:
            [w, h] = [canvas.winfo_reqwidth()-go_boundary*2,canvas.winfo_reqheight()-go_boundary*2]
            [printx, printy] = [1.0*i*w/go_row+go_boundary,1.0*j*h/go_column+go_boundary]
            dot_size = 1.0*go_dot*w/400
            canvas.create_oval(printx-dot_size,printy-dot_size, printx+dot_size,printy+dot_size, outline="black", fill="white")
```

④canvas_resize(event)函数的实现

该函数是通过一种非常巧妙的方法实现的，即每次鼠标点击时，用删除原有窗体绘制新的窗体来实现窗体大小的变化。由于程序执行速度很快，玩家看不出来窗体是否已重新绘制，会认为是窗体大小的变化。

```
last_resize = -1.0
def canvas_resize(event):      # 改变窗体大小时重绘棋盘
    global last_resize
    current_time = time.time()
    if (current_time-last_resize > 0.1):              #通过每次鼠标点击时重新绘制棋盘，实现该函数目的
        last_resize = current_time
        due = min(event.width, event.height)          #棋盘有最小标准
        canvas.configure(width=due, height=due)
        canvas.delete("all")
        canvas_print()
```

⑤canvas_dot(event)函数的实现

该函数的设计出于人机交互的目的。由于本游戏规则明确规定，不得悔棋，为使玩家不应看错位置而留下遗憾，特意设置了该函数。能够使玩家在落子前看到自己棋子的待定位置，若不合适，在点击左键之前还可调整。这也是本款围棋游戏的一大特色。

```
def canvas_dot(event):      # 移动鼠标时实时绘制棋子备选位置
    [w, h] = [canvas.winfo_reqwidth()-go_boundary*2,canvas.winfo_reqheight()-go_boundary*2]
    [x, y] = [event.x-go_boundary, event.y-go_boundary]
    [pos1, pos2] = [int(min(1.0*x/w*go_row, go_row)+0.5),int(min(1.0*y/h*go_column, go_column)+0.5)]
    [printx, printy] = [1.0*pos1*w/go_row+go_boundary,1.0*pos2*h/go_column+go_boundary]
    canvas.delete("dot")
    dot_size = 1.0*go_dot*w/400
    assert isinstance(printy, object)
    canvas.create_oval(printx-dot_size,printy-dot_size,printx+dot_size,printy+dot_size, outline="black", fill="", tag="dot")
```

⑥canvas_go(event)函数的实现

该函数是本程序的核心，能够实现落子、自动提子、判定胜负和判定超时等多项功能，故极为复杂，下面根据其功能，进行分块讲解。

这是一些数组变量代表的含义

```
arr = []
for i in range(go_row+1):
    tmp = []
    for j in range(go_column+1):
        tmp.append(0)
    arr.append(tmp)        #arr可看作一个二维数组，存储每个点上的棋子信息，0表示还没有落过子无字，
                           #1表示此处有黑子，2表示此处有白子，3表示此处原是黑子但现在被白子占领，4表示此处原是白子但现在被黑子占领
step=0             # 记录已下的棋子个数（其奇偶性表示当前棋子的颜色，奇为白，偶为黑）
ff=0
dx=[0,0,1,-1]
dy=[1,-1,0,0]
last_time1=0       #上一枚棋子落子时间
current_time1=0    #当前棋子落子时间
```

超时判负功能是通过判断当前落子时刻与上次落子时刻的插值是否大于 60 秒来实现的

```
current_time1 = time.time()
last_time1
if current_time1-last_time1>=60 and step!=0:
    if step%2==1:
        print("执白子者超时")
        print("执黑子者胜利")
    else:
        print("执黑子者超时")
        print("执白子者胜利")
    last_time1 = current_time1;
else:
    last_time1=current_time1;
    #print("不超时")
```

落子功能中 ff 初始化为 0，分如下情况：1. 此处从未下过子，直接下；2. 此处无子，但曾有过黑子，那此处只能下白子；3. 此处无子，但曾有过白子，那此处只能下黑子；4. 不符合上述条件，无法落子，重新选择位置。

```
[w, h] = [canvas.winfo_reqwidth() - go_boundary * 2, canvas.winfo_reqheight() - go_boundary * 2]
[x, y] = [event.x - go_boundary, event.y - go_boundary]
[pos1, pos2] = [int(min(1.0 * x / w * go_row, go_row) + 0.5), int(min(1.0 * y / h * go_column, go_column) + 0.5)]
if arr[pos1][pos2]==3 and step%2==0:
    ff=1
if arr[pos1][pos2]==0:
    ff=1
if arr[pos1][pos2]==4 and step%2==1:
    ff=1
if ff==1:
    if step%2==0 :
        arr[pos1][pos2]=1
    else:
        arr[pos1][pos2]=2
    [printx, printy] = [1.0*pos1*w/go_row+go_boundary,1.0*pos2*h/go_column+go_boundary]
    dot_size = 1.0*go_dot*w/400
    step=step+1
    if step%2:
        canvas.create_oval(printx-dot_size,printy-dot_size,printx+dot_size,printy+dot_size,outline="black",fill="black")
    else:
        canvas.create_oval(printx-dot_size,printy-dot_size,printx+dot_size,printy+dot_size,outline="black",fill="white")
```

提子功能主要采用的是枚举算法和用队列方式实现广度优先搜索算法。该功能上的实现主要分为两大部分。第一部分是判断是否有没气的棋子，如果没有，就没必要进入第二部分；第二部分是找出所有没气的棋子，做好相应的标记，最后通过重新绘制棋盘，找出拿走那些没气了的棋子。第一部分实现如下，以棋盘中每一个棋子为起点分别进行搜索，判定其是否没气。没气的判定就是指该棋子或与其同色的同伴被异色棋子紧密包围，该判定过程的实现就是通过广度优先搜索算法，具体过程可参考代码。只要找到一个棋子没气或找完全部棋子才可以退出。由此，第一部分已经完成，进入第二部分。首先程序将判定是否有棋子没气，若都有气，就没必要进行搜索，若存在没气，则用广度优先搜索算法以每个棋子为起点进行搜索，如果发现这个棋子已经没气了，则给该位置表上记号（根据上文的定义选择标 2 或 3）

```
for i in range(go_row + 1):
    for j in range(go_column + 1):
        if arr[i][j]==1 or arr[i][j]==2:
            que = []
            for ii in range(5000):
                tmp=[]
                for jj in range(5):
                    tmp.append(0)
                que.append(tmp)
            vis = []
            for ii in range(50):
                tmp=[]
                for jj in range(50):
                    tmp.append(0)
                vis.append(tmp)
            head=tail=1
            que[head][1]=i
            que[head][2]=j
            m=arr[i][j]
            vis[i][j]=1
            ff=0
            while head<=tail and ff==0:
                for ii in range(4):
                    xx=que[head][1]+dx[ii]
                    yy=que[head][2]+dy[ii]
                    if xx>=0 and xx<=go_row and yy>=0 and yy<=go_column and arr[xx][yy]==0 and vis[xx][yy]==0:
                        ff=1
                        break
                    elif xx>=0 and xx<=go_row and yy>=0 and yy<=go_column and arr[xx][yy]==3 and vis[xx][yy]==0:
                        ff=1
                        break
                    elif xx>=0 and xx<=go_row and yy>=0 and yy<=go_column and arr[xx][yy]==4 and vis[xx][yy]==0:
                        ff=1
                        break
                    elif xx>=0 and xx<=go_row and yy>=0 and yy<=go_column and arr[xx][yy]==m and vis[xx][yy]==0:
                        tail=tail+1
                        vis[xx][yy]=1
                        que[tail][1]=xx
                        que[tail][2]=yy
                head=head+1
            if ff == 0:
                que = []
                for ii in range(5000):
                    tmp = []
                    for jj in range(5):
                        tmp.append(0)
                    que.append(tmp)
            vis = []
            for ii in range(50):
                tmp = []
                for jj in range(50):
                    tmp.append(0)
                vis.append(tmp)
            head = tail = 1
            que[head][1] = i
            que[head][2] = j
            m=arr[i][j]
            arr[i][j]=m+2
            while head <= tail:
                for ii in range(4):
                    xx=que[head][1]+dx[ii]
                    yy=que[head][2]+dy[ii]
                    if xx >= 0 and xx <= go_row and yy >= 0 and yy <= go_column and arr[xx][yy]==m and vis[xx][yy]==0:
                        tail=tail+1
                        vis[xx][yy]=1
                        que[tail][1]=xx
                        que[tail][2]=yy
                        arr[xx][yy]=m+2
                head = head + 1
sum1=0
sum2=0
```

数字法判断胜负，通过的是每一次落子完成且提子之后，对双方的棋子和棋子围住的点以子为单位进行计数，超过 17*12/2 的一方即判为胜利。

```
sum1=0
sum2=0
for i in range(go_row + 1):
    for j in range(go_column + 1):
        if arr[i][j]==1 or arr[i][j]==4:
            sum1=sum1+1
        if arr[i][j]==2 or arr[i][j]==3:
            sum2=sum2+1
if sum1>((go_row + 1)*(go_column + 1)/2) :
    canvas.delete("all")
    print("执黑子者胜利")
if sum2>((go_row + 1)*(go_column + 1)/2) :
    canvas.delete("all")
    print("执白子者胜利")
```

⑥主函数，各种函数的结合

```
root.title("围棋（人人对战版）")
root.minsize(window_minsize, window_minsize)
root.configure(bg="yellow")
root.wm_attributes("-transparentcolor", "yellow")
canvas.configure(width=window_minsize, height=window_minsize, bg="#D5B092")
canvas.pack()
root.update()
root.bind("<Configure>", canvas_resize)
canvas.bind("<Motion>", canvas_dot)
canvas.bind("<Button-1>", canvas_go)
canvas_print()
root.mainloop()
```

（以上代码仅代表程序的关键部分，全部代码详见 人人对战.py）