华东师范大学计算机综合实践—实验报告

<table>
<tr><td>作品名称</td><td colspan="2">不围棋 NoGo</td><td>小组编号：A-24 组</td></tr>
<tr><td>姓名</td><td>冯祥旭</td><td>专业</td><td>地理科学</td></tr>
<tr><td>学号</td><td>10205102478</td><td>任课教师</td><td>郑凯</td></tr>
<tr><td>电话</td><td>13118901383</td><td>电子邮件</td><td>2711835261@qq.com</td></tr>
<tr><td rowspan="2">小组成员 2</td><td>学号：</td><td rowspan="2">小组成员 3</td><td>学号：</td></tr>
<tr><td>姓名：</td><td>姓名：</td></tr>
</table>

原创承诺

本人承诺本作品为本队成员独立创作完成，不侵犯任何第三方的知识产权或其他权利，同时符合中华人民共和国的有关法律，如本作品发生侵权行为由本队成员自负责任。

冯祥旭

小组成员（签字）：

作品简介

（设计目标与意义、关键技术、作品特色）约 250 字

设计目标：一个基本没有 bug 的，能够正常运作的 python 程序，能够正确判断输赢，正确的显示落子、棋谱，电脑落子有一定的合理性、前瞻性，能够打赢普通水平的人类等等意义：检验自己的编程水平是否能够独立完成一个项目，加深对一些算法的学习和应用，提高自己利用 python 做开发的经验。

关键技术：pygame 提供的各种函数，悔棋的设计，价值评估函数的设计，如何利用好 pygame 窗口实时刷新的机制，怎么设计程序的流程……

作品特色：一般是只写算法然后接入对战平台和别人对战，实现 UI 还是很少见的，电脑落子的音效包也是很有特色的。

游戏规则：棋盘同九路围棋棋盘，9×9；黑子先手，双方轮流落子，落子后棋子不可移动；对弈的目标不是吃掉对方的棋子，不是占领地盘；恰恰相反，如果一方落子后吃掉了对方的棋子，则落子一方判负；对弈禁止自杀，落子自杀一方判负；对弈禁止空手(pass)，空手一方判负。

完成作品涉及到的软件（请给出软件名称、版本）：

软件名称：python；软件版本：3.8；

软件名称：pygame 包；软件版本：2.1.2；

软件名称：pycharm；软件版本：2020。

展示作品需要的软件（请给出软件名称、版本）：

软件名称：python；软件版本：3.8；

软件名称：pygame 包；软件版本：2.1.2；

软件名称：pycharm；软件版本：2020。

完成作品的步骤（以图文并茂的形式具体说明制作的过程，不少于 1000 字）

1.灵感来源

在课程开始之时先确定要做什么内容，不围棋的灵感来源于以前看到过的一篇文章，惊讶于还有这样的一种和围棋的规则如此大相径庭的棋类。正好其规则比较简单，想来代码量不会很多，便将其选定为本次计算机综合实践的作品。

2.基本显示功能

在确定了要完成什么作品后，我最先确定实现的是一个能够使两名玩家在同一台电脑上能够进行对战的程序。作为一个棋类游戏，程序里面首先考虑的就是棋盘和棋子，棋盘我选用了网络上的图片素材来应用，棋子的展示方法我利用了 pygame 中的 pygame.draw.circle() 方法来实现，该方法的作用是在窗口绘制一个圆形，可以通过传递参数的方式来更改圆形的位置，大小，颜色等性质。

3.如何落子

解决了棋盘和棋子的展示问题后，要解决的问题是怎么将棋子正确的落到点上，这就涉及到了 pygame 窗口的坐标系统，其以左上角为原点，向下为 Y 轴，向右为 X 轴。根据这个原则，经过多次的调参，我找到了棋盘点与坐标位置之间的函数关系，绘制出了点与线，确定了每一个点的坐标是怎么计算的，并建立一个 9x9 的列表（一个列表里面的元素是 9 个小列表，小列表里面是每一行的坐标）来记录当前棋盘上的各个点的落子状况，棋盘落子会更改列表中相应位置的值，然后程序的其他部分会遍历该列表，获得状态值改变的的元素的索引，将索引坐标转换到棋盘上（即窗口坐标），利用 pygame 的刷新机制来绘制棋子，做到实时更新游戏的进程

```
# 绘制棋盘
def draw_map():
    screen.blit(pygame.image.load(r"棋盘.jpg"), (0, 0))  # 背景图片填充
    for item in range(0, size):  # 绘制行和纵坐标
        pygame.draw.line(screen, black, [border_width, border_width + item * unit],
                         [border_width + (size - 1) * unit, border_width + item * unit], 1)
        screen.blit(ziti.render(f'{item + 1}', True, black), [border_width - 30, border_width + item * unit - 10])
    for item in range(0, size):  # 绘制列和横坐标
        pygame.draw.line(screen, black, [border_width + item * unit, border_width],
                         [border_width + item * unit, border_width + (size - 1) * unit], 1)
        screen.blit(ziti.render(f'{item + 1}', True, black), [border_width + item * unit - 5, border_width - 30])
```

```
def __init__(self):
    self.chess_position = [[0 for y in range(0, 9)] for x in
                           range(0, 9)]  # 9 * 9的二维列表，用于表示棋盘：0 ~ 无棋子， 1 ~ 黑棋，-1 ~ 白棋
    self.current_record = []  # 列表表达式二维列表记录走棋的路径，用于悔棋和搜索。它的元素是一个元组(棋子类型，chess_position.search_hang，chess_posit
    self.chess_status = 0  # 棋子状态，黑棋先走，最开始会赋值为1：0 ~ 未开局和未分出胜负；1 ~ 等待黑棋落子；-1 ~ 等待白棋落子；3 ~ 结束（人类胜）；4 ~ 
    self.number_white = 0  # 白子个数
    self.number_black = 0  # 黑子个数
    self.visit = [[0 for y in range(0, 9)] for x in range(0, 9)]  # 搜索记录列表，0是没搜索过，1是搜索过
```

```
    index = 1
    for item in self.current_record:
        pygame.draw.circle(screen, black, [border_width + item[1] * unit, border_width + item[2] * unit],
                           int(unit / 2.5)) if item[0] == 1 else pygame.draw.circle(screen, white,
                                                                                    [border_width + item[1] * unit,
                                                                                     border_width + item[2] * unit],
                                                                                    int(unit / 2.5))  # 根据状态判断画哪个颜色
```

4.判断输赢

接下来就是实现怎么样判断游戏胜负，规则是基于“气”来实现的，“气”的规则与围棋类似，利用一个全局搜索来实现每一块气的计算，遍历棋盘获得白棋或者是黑棋的全部气。

当某一方不论怎么下都能使另外一方的一块棋子没有气时，游戏结束，这便是提前判断输赢，有利于在后期棋子非常杂乱时让棋手们知道二者之间是否已经决出胜负，因为了解这个游戏规则的玩家很难在一开始就输（因为这个游戏的精髓在于躲避），所以特意做了一个这样的设计。

```
    def search_qi(self, i, j):
        search_hang = [1, 0, -1, 0]  # 纵向搜索步伐，行
        search_lie = [0, 1, 0, -1]  # 横向搜索步伐，列
        chess_qi = 0
        self.visit[i][j] = 1
        chess_qi = 0  # 记录气数
        hang = 0
        lie = 0
        for k in range(4):
            hang = i + search_hang[k]
            lie = j + search_lie[k]
            if (hang < 0 or hang > 8) or (lie < 0 or lie > 8): continue
            if self.chess_position[hang][lie] == 0 and self.visit[hang][lie] == 0:
                chess_qi += 1
                self.visit[hang][lie] = 1
            elif self.chess_position[hang][lie] == self.chess_position[i][j] and self.visit[hang][lie] == 0:
                chess_qi += self.search_qi(hang, lie)
        return chess_qi

    def is_win(self):
        is_chess_death = 0
        chess_live = [[-1 for y in range(0, 9)] for x in range(0, 9)]
        for i in range(9):
            for j in range(9):
                if self.chess_position[i][j] == 0:
                    chess_live[i][j] = -1
                    continue
                else:
                    self.visit = [[0 for y in range(0, 9)] for x in range(0, 9)]
                    chess_live[i][j] = self.search_qi(i, j)
                    if chess_live[i][j] == 0: is_chess_death = 1
```

6.悔棋

在实现了基本的功能后，我根据其他游戏程序界面设计的经验，认为我的程序也应该有悔棋这一选项。

悔棋部分的函数设计，关键在于一个参量 self.current_record = []，它的元素是一个元组(棋子类型，chess_position.search_hang，chess_position.search_lie)，它记录了走棋路径，悔棋时候只要 pop 最后一个元素，同时 pop self.chess_position（一个 9*9 的二维列表，记录棋子在棋盘中的位置）中对应的位置，再调用绘制函数把棋子重新绘制一遍即可，并将状态改为等待悔棋方落子状态。

```
# 悔棋，游戏结束就不能悔棋了
def huiqi(self):

    if len(self.current_record) == 0: return  # 棋盘上无子，报错
    last_chess = self.current_record.pop()  # 棋盘上有子，从实时记录中弹出并清除最后一个棋子
    self.chess_position[last_chess[1]][last_chess[2]] = 0  # 历史记录中位置清除，状态保留，即last_chess[0]不变，下次绘制不画该棋子
    if len(self.current_record) != 0: last_chess = self.current_record.pop()  # 棋盘上有子，从实时记录中弹出并清除最后一个棋子
    self.chess_position[last_chess[1]][last_chess[2]] = 0  # 历史记录中位置清除，状态保留，即last_chess[0]不变，下次绘制不画该棋子
    self.chess_status = 1  # 刷新当前棋子状态，如果当前悔的是黑棋，那么状态切换为等待黑棋落子
    self.number_white -= 1  # 棋子数目-1
    self.number_black -= 1
```

7.棋谱

实现了悔棋功能后，我为了增加程序特色，联想到了以前看过的棋谱上每一个棋子上都写出了其是第几步落下的。因此基于上一段的记录行棋路径的列表，我实现了棋谱功能，游戏结束后就在棋子上展示该棋子时第几步落子的。至此，“人人模式”的大部分功能都已经实现完成。

```
# 绘制棋子，游戏结束时同时绘制棋谱
def draw_chess(self):
    index = 1  # 记录走棋路径
    for item in self.current_record:  # 得到棋子坐标
        pygame.draw.circle(screen, black, [border_width + item[1] * unit, border_width + item[2] * unit],
                           int(unit / 2.5)) if item[0] == 1 else pygame.draw.circle(screen, white,
                                                                                    [border_width + item[1] * unit,
                                                                                     border_width + item[2] * unit],
                                                                                    int(unit / 2.5))  # 根据状态判断画哪个颜色
        if self.chess_status == 3 or self.chess_status == 4: screen.blit(ziti.render(f'{index}', False, white),
                                                                         [border_width + item[1] * unit - 5,
                                                                          border_width + item[2] * unit - 10]) if \
        item[0] == 1 else screen.blit(ziti.render(f'{index}', False, black), [border_width + item[1] * unit - 5,
                                                                              border_width + item[
                                                                                  2] * unit - 10])  # 走棋路径
        index += 1
```

8.开始界面

接下来我实现了开始界面，并设计了“新开一局”按钮来使的玩家从“人人模式”的界面进入开始界面；“退出游戏”按钮来退出游戏。

```
def is_click(self, pos):
    if new_start_x[0] < pos[0] < new_start_x[1] and new_start_y[0] < pos[1] < new_start_y[1]:
        enter()
    elif exit_game_x[0] < pos[0] < exit_game_x[1] and exit_game_y[0] < pos[1] < exit_game_y[1]:
        sys.exit()
    elif huiqi_x[0] < pos[0] < huiqi_x[1] and huiqi_y[0] < pos[1] < huiqi_y[
        1] and self.chess_status != 3 and self.chess_status != 4:
        self.huiqi()
        screen.blit(pygame.image.load(r"棋盘.jpg"), (0, 0))
        draw_map()
        self.draw_chess()
        self.draw_panel()
        return 0
    else:
        return -1
```

9.人机模式

在实现完开始界面和“人人模式”界面后，我想提高整个程序的水准，便开始设计“人机模式”，人机模式的实现大部分借鉴与“人人模式”，主要是多出了电脑落子部分。这部分的实现我最初先用随机落子来代替，接着通过网上查阅文献设计了一个价值计算函数，遍历棋盘找出白子落子的价值最大点，使电脑落子于该点。

人机模式最重要的是对局面价值的评估，找到当前局面的最优点才能走出最有利的棋子，所以人机模式包含的函数都是以价值评估为核心，第一个 update_gas 函数用于更新当前局面每个棋子的气，由于棋串的存在，这部分采用了广度递归搜索的方法；第二个 value_base_mp 通过向下模拟一步行棋来判断局面的价值；pg_base_point 则是对某一个可以下的点模拟行棋后判断接下来的局面价值，进而确定该点价值；find_all_point 通过对所有点模拟落子后找到价值最大点的集合；select_point 则是在当前局面过于简单或复杂的情况下产生多个价值最大点时使用打散规则选择落子点，使局面更加开阔。（这里图片就不放出来了，具体的代码见 main.py ），以下放出参考的论文的部分：

文章编号：1000-5641(2019)01-0058-08

基于价值评估的不围棋递归算法

郭倩宇， 陈优广

(华东师范大学 计算中心, 上海 200062)

为了避免蒙特卡洛思想本身的弊端，本文选择用价值评估模型来实现不围棋人工智能，主要是从不围棋本身的博弈思路来考虑. 为了达到不输掉比赛的目的，就是努力不吃掉对手的子，那么，就会有两种行棋思路：第一，使自己的“气”比对手的少；第二，使自己的“眼”比对手的多.

基于以上两点，本文将被对方打吃的子数与己方“眼”的个数规定为权利数，以此来构造不围棋的价值评估模型和函数. 显而易见，在后盘，双方都在消耗自己的权利，而权利数多的一方将取得胜利，因此，不围棋行棋的主要目的可以描述为制造己方权利或是破坏对方权利. 本文中将权利的制造和破坏称为权利规则，这一规则容易想到的方法是把各种棋形摆出来，比如被打吃、“眼”等等. 但在实际局面中，形成权利的棋形千奇百怪，远不是能够一一列举的. 如果用模式库的方式，则难以避免占用空间较多的问题. 所以，本文利用“气”这一概念来思考，形成权利值的标志就是会在棋子周围的点中形成一个或多个对手无法落子的交叉点，即这个交叉点是己方的单个眼位或己方在此处有且仅有最后一口气(如图 1 和图 2 中标记处)，则这个交叉点就是自己的权利.

由此，在不围棋 9×9 棋盘中，记坐标点为(1, 1)至(9, 9)，从(1, 1)起，对于第 j 个交叉点($1\leqslant j\leqslant 81$)，其坐标为$(m, n)(1\leqslant m\leqslant 9, 1\leqslant n\leqslant 9)$，假定对于第 i 手($1\leqslant i\leqslant 81$)行棋颜色为黑，在坐标(m, n)处白方出现无法落子的情况，即(m, n)为黑方眼位或(m, n)处为黑方棋子或棋串的最后一口气，则记为 N_w(不能落子记为 1，可以落子记为 0). 其此时黑方权利值 B_i 可以评估为

$$B_i=\sum_{j=1}^{81}N_w^j. \tag{1}$$

同理，设第 i 手为白方，黑方无法落子记为N_b(不能落子记为 1，可以落子记为 0)，则此时

白方权利值W_i可以评估为

$$W_i=\sum_{j=1}^{81}N_b^j. \tag{2}$$

对于特定局面的价值评估模型得以构建，当进行到第 i 手时将局面总权利值记为 P_i，则

$$P_i=B_i+W_i. \tag{3}$$

在不围棋行棋过程中，黑白颜色的不同并不会影响当前局面的价值，也就是说，某一选点的优劣不会因当前行棋颜色不同而不同. 因此，无论黑白哪方在进行到第 i 手($1\leqslant i\leqslant 81$)时，只要选择 P_i 最大的点落子即可. 因此，可得出某一点(p, q)的价值 V 的评估函数为

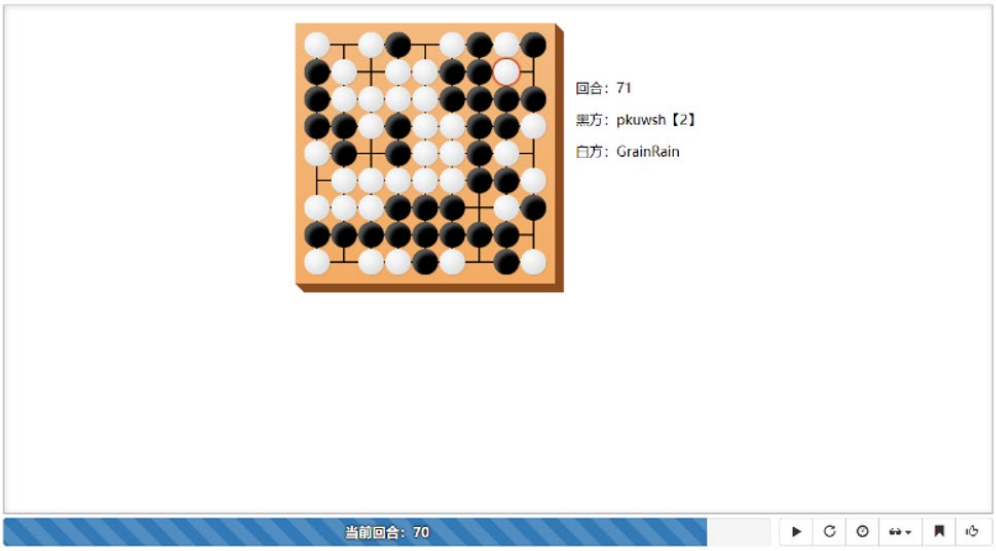
$$V(p,q)=\sum_{j=1}^{81}N_w^j+\sum_{j=1}^{81}N_b^j. \tag{4}$$

根据式(4)，可以对当前局面下所有可下点计算价值，其评估函数的伪代码如下.


```
Valuepoint(选点, 手数i)
{
  将此选点模拟为黑棋;
  While(81 个点均被遍历)
  {
    检测当前所有白棋不可落子的地方，记为 a;
    则黑棋权利值P_b←a−i;
  }
  将此选点模拟为白棋;
  While(81个点均被遍历)
  {
    检测当前所有黑棋不可落子的地方，记为 b;
    则白棋的权利值 P_w←b−i;
  }
此点评估价值为 P = P_b + P_w;
Return P;
}
```

10.网络对战

这是北京大学人工智能实验室开发的一个对战平台，里面就有不围棋的比赛，我利用有它来对我的算法进行改进，由于该网站需要用 c++编写接口，所以我用人工的方法来使我的 AI 和别人的 AI 进行对战，这是一次颇为有趣的体验



11.总结

至此，程序的总体已经实现完毕。之后经过了一系列的 debug（包括但不限于：电脑落子不够聪明、从模式退出到开始界面错误，悔棋没有考虑未落子时以及游戏决出胜负的时候等）以及音效包的补充，最终完成了作品的全部内容。

代码 100 行前主要为相关常量及通用函数，包括颜色，字体，窗口大小，按钮大小等常量以及主界面绘制函数，按钮绘制函数等等。

class men_machine_Chess 是一个实现游戏逻辑的类，里面包含了包括提前判断胜负，胜负判断，电脑评估当前局面，悔棋等方法。

class men_machine_PlayChess 是继承 class men_machine_Chess 的子类，包括了控制台文本绘制，落子等函数，承担接收玩家操作的功能。

后面的两个类则是仿造人机模式写的，有删减和改动。

def enter()是串联开始界面和两个模式的核心，是该程序的主函数。

关键参数如下：

self.chess_status = 0 # 棋子状态，黑棋先走，最开始会赋值为 1：0 ~ 未开局和未分出胜负；1 ~ 等待黑棋落子；-1 ~ 等待白棋落子；3 ~ 结束（人类胜）；4 ~ 结束（人类输），这是串联棋局的关键，标志着游戏处于什么状态。

self.chess_position = [[0 for y in range(0, 9)] for x in range(0, 9)] # 9 * 9 的二维列表，用于表示棋盘：0 ~ 无棋子， 1 ~ 黑棋，-1 ~ 白棋，一个存储已落子位置的的参量。

self.current_record = [] # 列表表达式二维列表记录走棋的路径，用于悔棋和搜索。它的元素是一个元组(棋子类型，chess_position.search_hang，chess_position.search_lie)

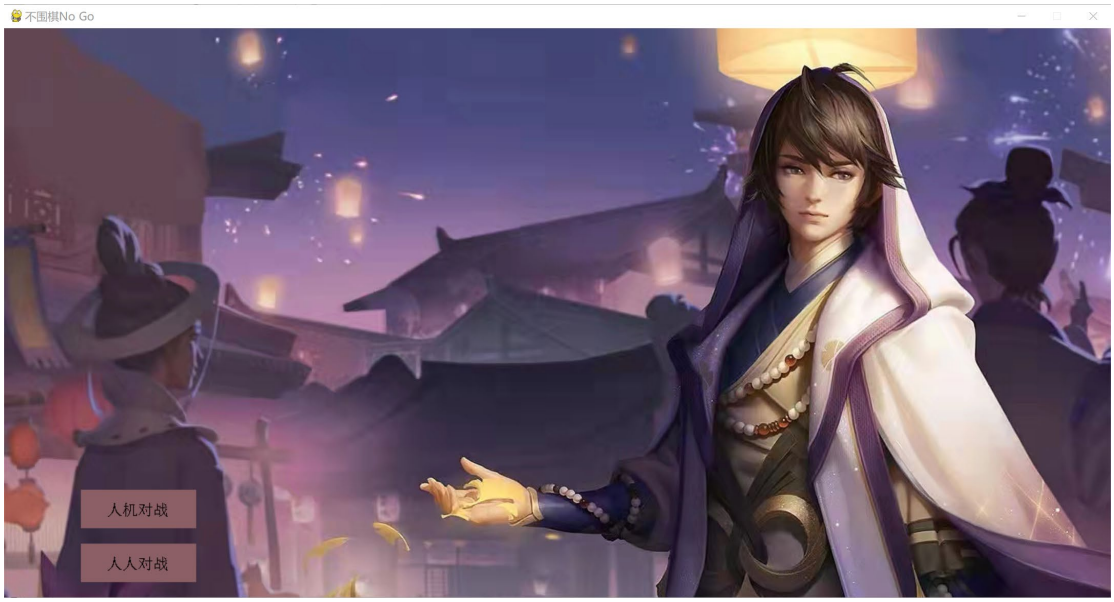
……

作品效果图（整体、关键点和特效）

要求截取屏幕反映作品整体、关键点和特效的画面若干张，并给出每张图对应的文字说明。

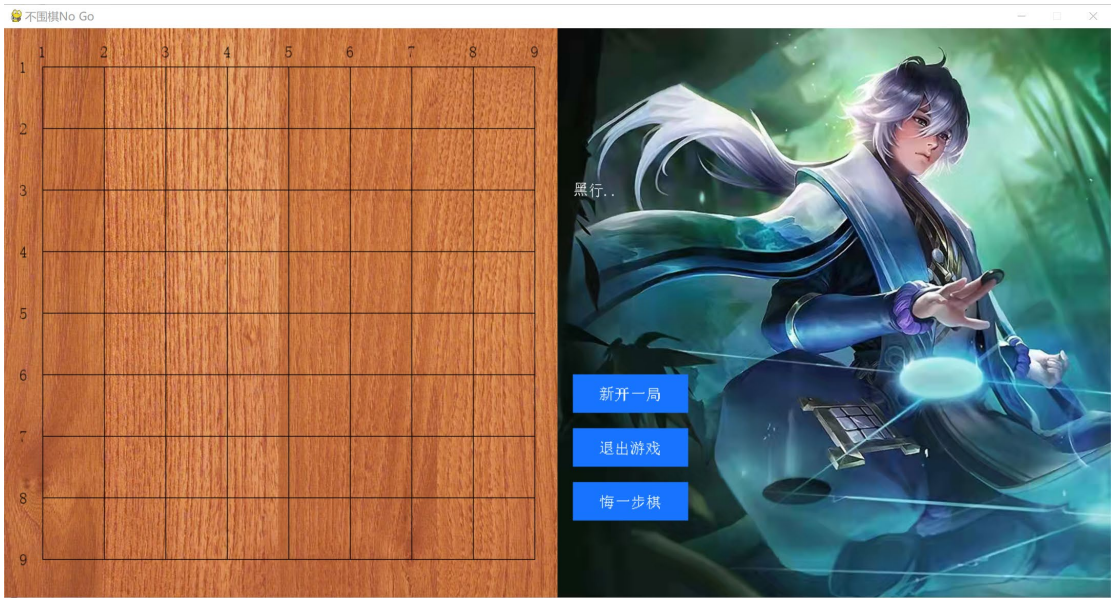
1.开始界面

运行程序最先看到的就是如下的界面，点击左下角“人机对战”和“人人对战”可分别开始两种模式下的游戏；



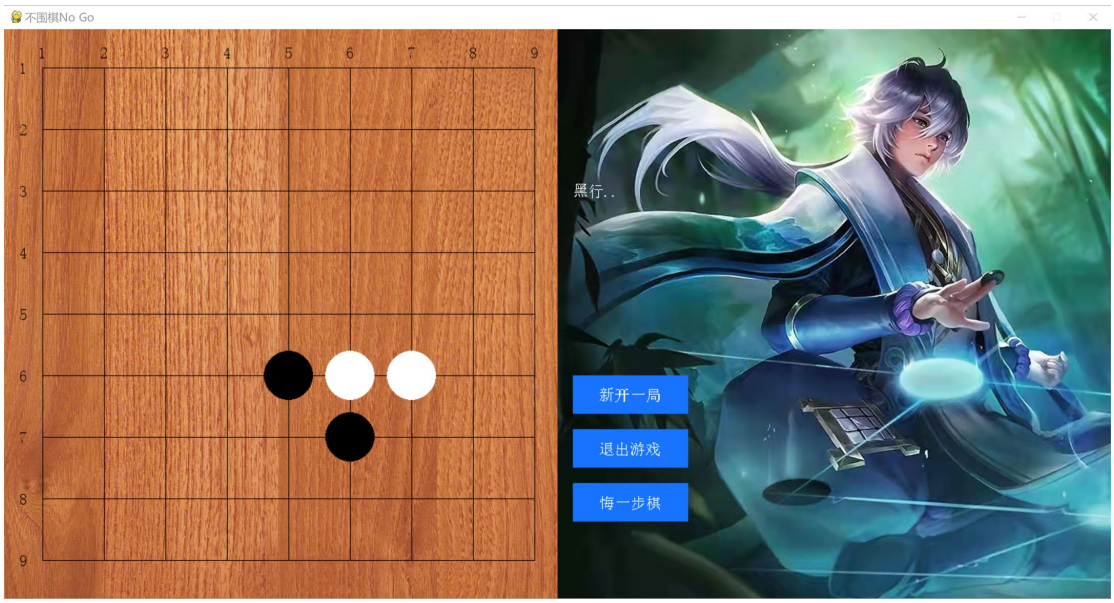
2.人人对战

点击“人人对战”按钮，开始游戏，右边的游戏选项分别是“新开一局”（点击后可以直接退出当前界面并回到开始界面）；“退出游戏”（点击可以直接推出程序）；“悔一步棋”（点击后可以悔棋，棋盘上棋子将消失，注意在开局未落子时和游戏结束时该按钮无效）



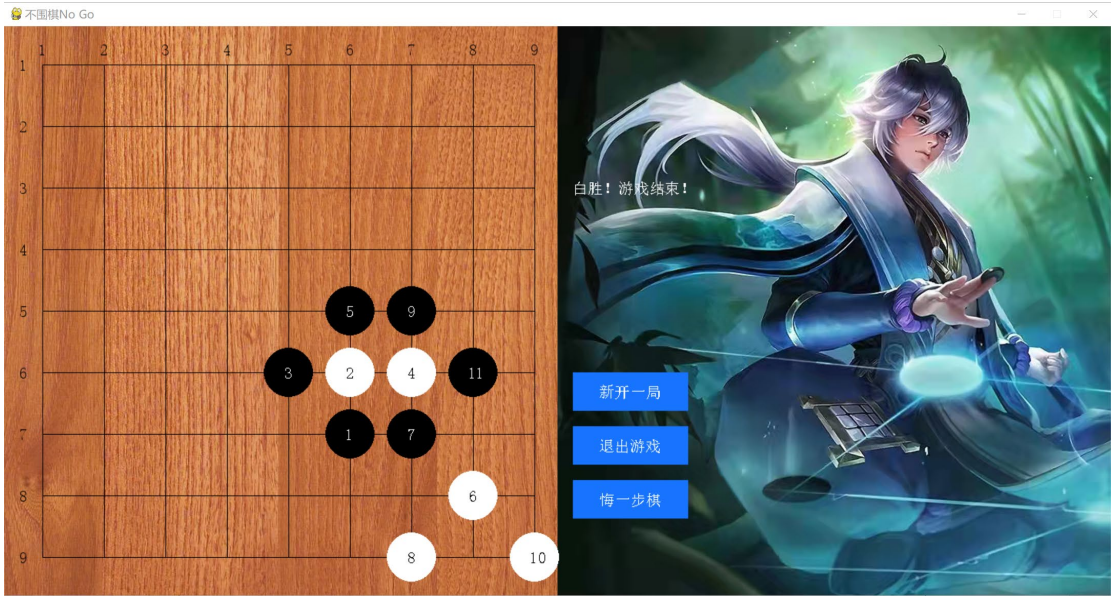
3.走棋提示

在下棋过程中，游戏选项上的状态面板会提示哪一方走棋，本程序无论哪个模式都默认黑棋先走



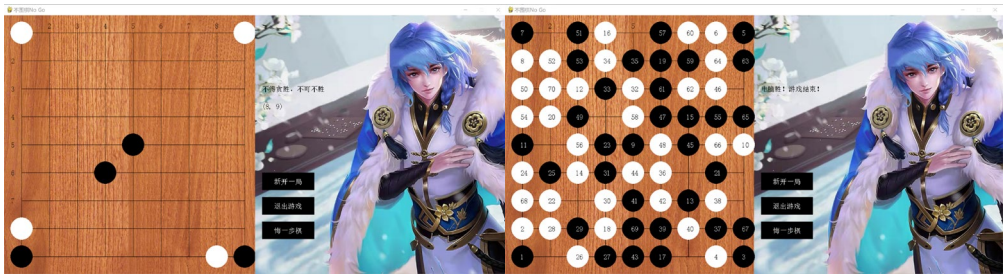
4.棋谱

在游戏决出胜负后，本程序还会将棋谱打印在棋盘上，方便复盘，另外，当有一方不管怎么下都是输的时候，程序也会自动结束游戏



5.人机模式

“人机模式”下的程序运作与“人人模式”大体相同，改变的地方有：在电脑落子时会有音效，在状态栏会有文本，并且电脑落子速度较快，为了让玩家辨认，在文本下面标出了电脑最近一次的落子坐标，坐标轴在棋盘上已经画好



本人工作（详细描述在本组作品实现过程中，本人参与完成的具体工作，对作品中自己原创的内容是哪些进行说明。不少于 300 字。）：

除了音效，三张人物背景图片和一张棋盘背景图片来源网络，算法设计参考了论文外，其余部分皆为原创。

参考文献：

中图分类号：TP399　　**文献标志码**：A　　**DOI**：10.3969/j.issn.1000-5641.2019.01.007

感想（不少于 300 字）：

这次做的《不围棋》是很有趣的一个尝试，以前都是玩游戏，这次变成了自己来做游戏。在做这个项目的过程中，遇见了许许多多的困难，包括但不限于素材的采集，程序框架是设计，与 bug 之间的斗智斗勇等等。但同时我也收获了很多，做这个项目的时候学到了蒙特卡洛搜索树，如何利用 pygame 制作游戏等相关知识，当自己写的程序能够运行起来的那一刻，其实是很激动的，最开心的是一个个 bug 被解决，一个个功能被实现，战胜排行榜上的一个个对手……

在这个项目的实现过程中，感谢北京大学人工智能实验室开发的游戏对战平台以用于测评电脑实力（https://botzone.org.cn/）；感谢郭倩宇，陈优广两位老师的文章《基于价值评估的不围棋递归算法》提供的思路，素材文件夹中有原文；感谢 Pete Shinners 提供的 pygame 库能让我的想法付诸实现。