同济大学人工智能原理课程实验报告

实验题目:五子棋

小组成员: 1750844 周展田， 1752723 王松森

分工: 周展田：查找算法资料编写算法和文档
王松森：实现算法整合及图形化

一．实验概述

【实验目的】

通过实验加深对抗搜索中极小极大算法的理解，知道如何在搜索树中使用α-β剪枝，尽可能减少对结点的搜索，相对不使用剪枝更快地得到最优决策。

【实验问题描述】

本实验实现采用极小极大算法和α—β剪枝实现五子棋的人机智能对决。

（1）构建博弈树。

（2）实现极小极大算法对博弈树进行搜索。

（3）用α-β剪枝对博弈树进行搜索。

（4）对 2 个算法进行比较，分析算法的性能。

【实验原理】

1、**构造搜索树**。

每一个盘面布局为树中一个节点

2、**极大极小值算法**

3、**α－β剪枝**

MINMAX(root)=max(min(x1,x2,x3….),min(y1,y2…)…)

【实验环境】

Anaconda3 64bit Spyder

二．实验过程及结果

【算法详细设计】

1、Python 实现：

二维数组表示棋盘

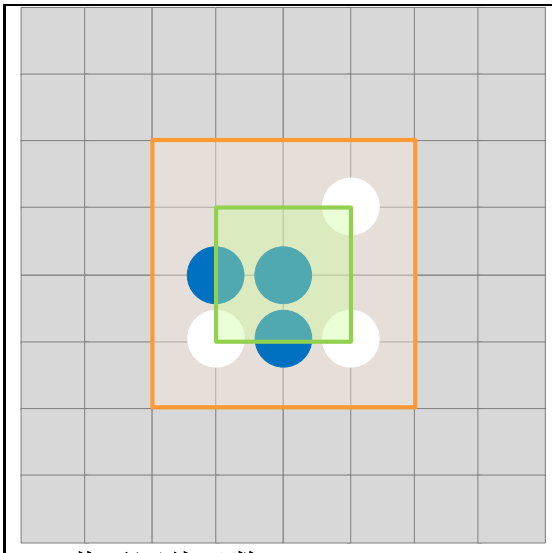
1，-1 表示黑白棋

每一个棋盘布局，即为一个结点

2、搜索范围

棋盘已占有棋子的外切矩形范围外扩一层

并判断是否超出棋盘范围

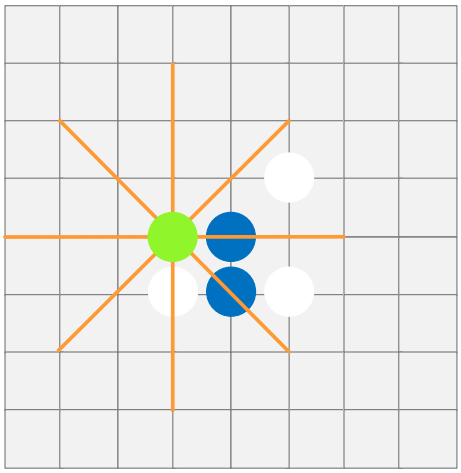


3、落子评估函数

（1）计算落子后每一个方向的评估值

0 为空，1 为我方棋子，-1 为敌方棋子，一组数据表示直线上一种排布方式

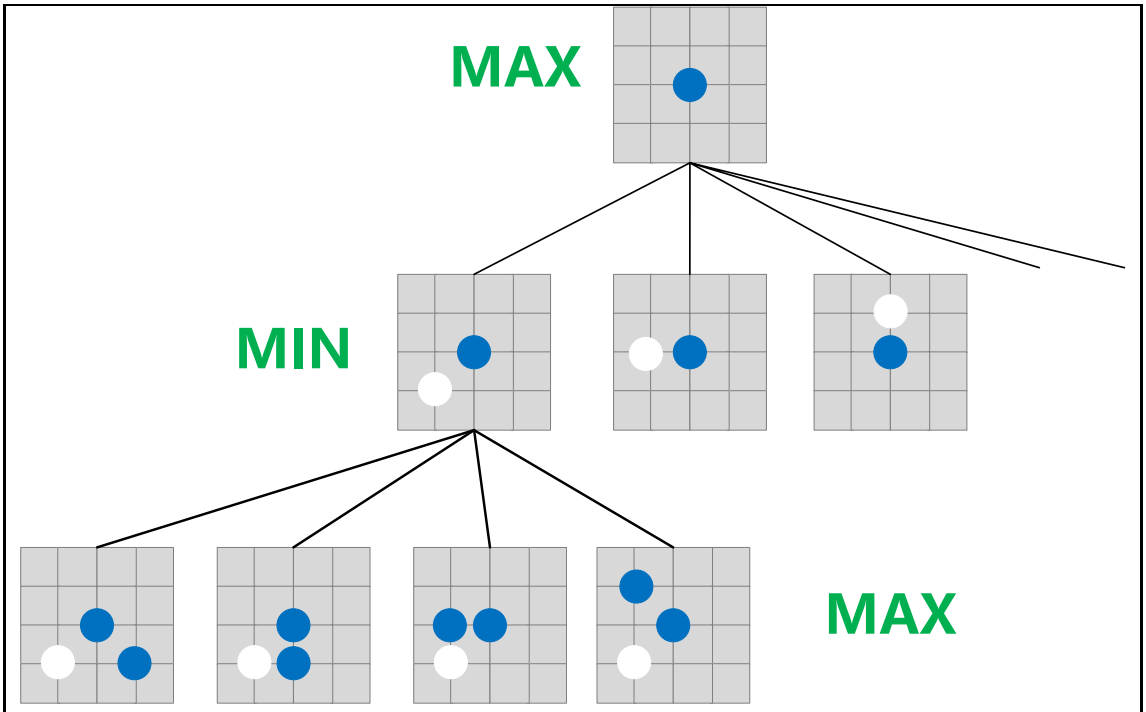
```
# 棋型评估分值
shape_score = {
    (0, 1, 0): 5,                   # 单子
    (0, 1, 1, -1): 10,              # 死2
    (-1, 1, 1, 0): 10,              # 死2
    (0, 1, 1, 0): 20,               # 活2
    (-1, 1, 1, 1, 0): 20,           # 死3
    (0, 1, 1, 1, -1): 20,           # 死3
    (0, 1, 1, 1, 0): 45,            # 活3
    (-1, 1, 1, 1, 1, 0): 60,        # 死4
    (0, 1, 1, 1, 1, -1): 60,        # 死4
    (0, 1, 1, 1, 1, 0): 120,        # 活4
    (0, 1, 1, 1, 1, 1, 0): 300,     # 成5
```



（2）选取值最大的两个方向，其值和作为该点的评估函数值

4、棋面评估函数

本次落子值减去前一次落子值记为本次盘面值



5、程序界面设计

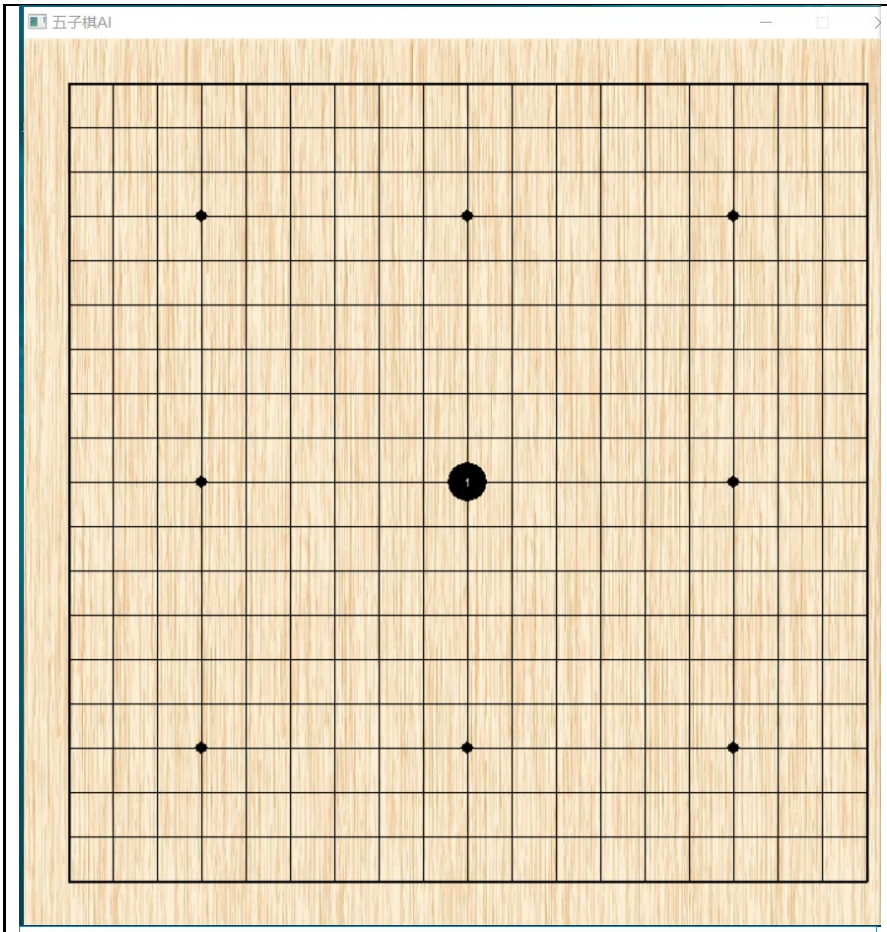
采用 Python 编程

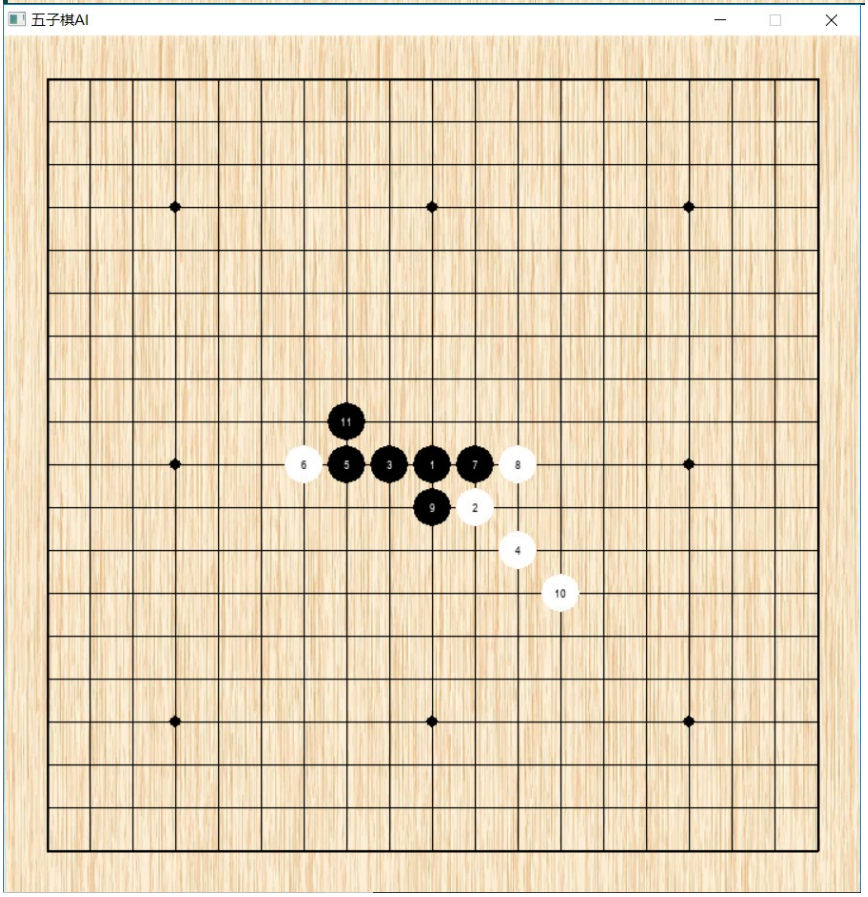
(1)根据提示输入是否使用 α - β 剪枝

```
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Do you need Alpha-Beta cutting strategy?(1/0:yes/no)
```

(2)进入游戏界面，选择先手还是后手

```
Hello from the pygame community. https://www.pygame.org/contribu
Do you need Alpha-Beta cutting strategy?(1/0:yes/no)1
Do you want to choose black or white?(1/0:black/white)0
```





(3)最后先形成 5 子相连的棋局一方获得胜利，并且按下空格可进行下一局对决。

(4)记录电脑下每步棋所需的时间

```
F:\人工智能\五子棋\five_in_a_row.exe
Alpha-Beta pruning?(1/0:yes/no)0
Time cost: 0.2821 s
Time cost: 0.393 s
Time cost: 0.3716 s
Time cost: 0.3454 s
Time cost: 0.4483 s
Time cost: 0.4728 s
Time cost: 0.4721 s
Time cost: 0.4598 s
Time cost: 0.5178 s
Time cost: 0.4675 s
Time cost: 0.45 s
Time cost: 0.4998 s
```

【源程序】//提高可读性，标准文字解释

```
import pygame
import os
import time
import sys

# 参数设置
```

```
WIDTH = 720  # 屏幕宽度
HEIGHT = 720  # 屏幕高度
SIZE = 19  # 棋盘大小为19*19
GRID_WIDTH = WIDTH // (SIZE+1)  # 网格尺寸
FPS = 30  # 刷新频率

# 颜色设置
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)

alpha_beta_flag = int(input("Do you need Alpha-Beta cutting
strategy?(1/0:yes/no)"))
color_choice = int(input("Do you want to choose black or
white?(1/0:black/white)"))
# pygame初始化设定
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("五子棋AI")
clock = pygame.time.Clock()

# base_folder = os.path.dirname(__file__)
# img_folder = os.path.join(base_folder, "images")
img_folder = "images"
bg_img = pygame.image.load(os.path.join(img_folder, "background.png"))
background = pygame.transform.scale(bg_img, (WIDTH, HEIGHT))
back_rect = background.get_rect()

# 绘制网格线
def draw_background(surf):
    screen.blit(background, back_rect)
    rect_lines = [((GRID_WIDTH, GRID_WIDTH), (GRID_WIDTH, HEIGHT -
GRID_WIDTH)),
                  ((GRID_WIDTH, GRID_WIDTH), (WIDTH - GRID_WIDTH,
GRID_WIDTH)),
                  ((GRID_WIDTH, HEIGHT - GRID_WIDTH), (WIDTH - GRID_WIDTH,
HEIGHT - GRID_WIDTH)),
                  ((WIDTH - GRID_WIDTH, GRID_WIDTH), (WIDTH - GRID_WIDTH,
HEIGHT - GRID_WIDTH))]
    for line in rect_lines:
        pygame.draw.line(surf, BLACK, line[0], line[1], 2)
    for i in range(17):
        pygame.draw.line(surf, BLACK, (GRID_WIDTH * (2 + i), GRID_WIDTH),
```

```
                      (GRID_WIDTH * (2 + i), HEIGHT - GRID_WIDTH))
        pygame.draw.line(surf, BLACK,
                         (GRID_WIDTH, GRID_WIDTH * (2 + i)),
                         (HEIGHT - GRID_WIDTH, GRID_WIDTH * (2 + i)))
    circle_center = [(GRID_WIDTH * 4, GRID_WIDTH * 4),
                     (GRID_WIDTH * 10, GRID_WIDTH * 4),
                     (GRID_WIDTH * 16, GRID_WIDTH * 4),
                     (GRID_WIDTH * 4, GRID_WIDTH * 10),
                     (GRID_WIDTH * 10, GRID_WIDTH * 10),
                     (GRID_WIDTH * 16, GRID_WIDTH * 10),
                     (GRID_WIDTH * 4, GRID_WIDTH * 16),
                     (GRID_WIDTH * 10, GRID_WIDTH * 16),
                     (GRID_WIDTH * 16, GRID_WIDTH * 16)]
    for circle in circle_center:
        pygame.draw.circle(surf, BLACK, circle, 5)


win_flag = 0  # -1:white win;1:black win

color_flag = 1  # black
step = 0
matrix = [[0 for i in range(SIZE + 2)] for j in range(SIZE + 2)]  # 棋型
矩阵
min_x, min_y, max_x, max_y = 0, 0, 0, 0  # 搜索范围


# 刷新棋盘已占有棋子的外切矩形范围
def xy_range(x, y):
    global min_x, min_y, max_x, max_y
    if step == 0:
        min_x, min_y, max_x, max_y = x, y, x, y
    else:
        if x < min_x:
            min_x = x
        elif x > max_x:
            max_x = x
        if y < min_y:
            min_y = y
        elif y > max_y:
            max_y = y


# 棋型评估分值
shape_score = {
    (0, 1, 0): 5,                # 单子
```

```
    (0, 1, 1, -1): 10,           # 死 2
    (-1, 1, 1, 0): 10,           # 死 2
    (0, 1, 1, 0): 20,            # 活 2
    (-1, 1, 1, 1, 0): 20,        # 死 3
    (0, 1, 1, 1, -1): 20,        # 死 3
    (0, 1, 1, 1, 0): 45,         # 活 3
    (-1, 1, 1, 1, 1, 0): 60,     # 死 4
    (0, 1, 1, 1, 1, -1): 60,     # 死 4
    (0, 1, 1, 1, 1, 0): 120,     # 活 4
    (0, 1, 1, 1, 1, 1, 0): 300,  # 成 5
    (0, 1, 1, 1, 1, 1, -1): 300,
    (-1, 1, 1, 1, 1, 1, 0): 300,
    (-1, 1, 1, 1, 1, 1, -1): 300,
    (-1, 1, 1, 1, 1, 1, 1, -1): 300,
    (-1, 1, 1, 1, 1, 1, 1, 1, -1): 300
}


# 评估一个节点分值
def evaluate_node(list_h, list_v, list_s, list_b):
    score_h = shape_score.get(tuple(list_h), 0)
    score_v = shape_score.get(tuple(list_v), 0)
    score_s = shape_score.get(tuple(list_s), 0)
    score_b = shape_score.get(tuple(list_b), 0)
    rank = [score_h, score_v, score_s, score_b]
    rank.sort()
    rank.reverse()
    score = rank[0] + rank[1]  # 把最大的两个分值相加作为总分值
    return score


# 获得该结点在水平、竖直、左斜、反斜方向的一维向量
def get_list(mx, my, color):
    global matrix

    list1 = []
    tx, ty = mx, my
    while matrix[tx][ty] == color:
        list1.append(1)  # 1 表示是己方棋子，-1 是敌方棋子
        tx = tx + 1
        ty = ty
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
SIZE:
        list1.append(-1)
    else:
```

```
        list1.append(0)
    list1.pop(0)  # 删除自己 防止在合并的时候重复计算
    list2 = []
    tx = mx
    ty = my
    while matrix[tx][ty] == color:
        list2.append(1)
        tx = tx - 1
        ty = ty
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
SIZE:
        list2.append(-1)
    else:
        list2.append(0)
    list2.reverse()
    list_h = list2 + list1

    list1 = []
    tx = mx
    ty = my
    while matrix[tx][ty] == color:
        list1.append(1)
        tx = tx
        ty = ty + 1
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
SIZE:
        list1.append(-1)
    else:
        list1.append(0)
    list1.pop(0)
    list2 = []
    tx = mx
    ty = my
    while matrix[tx][ty] == color:
        list2.append(1)
        tx = tx
        ty = ty - 1
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
SIZE:
        list2.append(-1)
    else:
        list2.append(0)
    list2.reverse()
    list_v = list2 + list1
```

```
    list1 = []
    tx = mx
    ty = my
    while matrix[tx][ty] == color:
        list1.append(1)
        tx = tx + 1
        ty = ty + 1
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
SIZE:
        list1.append(-1)
    else:
        list1.append(0)
    list1.pop(0)
    list2 = []
    tx = mx
    ty = my
    while matrix[tx][ty] == color:
        list2.append(1)
        tx = tx - 1
        ty = ty - 1
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
SIZE:
        list2.append(-1)
    else:
        list2.append(0)
    list2.reverse()
    list_s = list2 + list1

    list1 = []
    tx = mx
    ty = my
    while matrix[tx][ty] == color:
        list1.append(1)
        tx = tx + 1
        ty = ty - 1
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
SIZE:
        list1.append(-1)
    else:
        list1.append(0)
    list1.pop(0)
    list2 = []
    tx = mx
```

```
    ty = my
    while matrix[tx][ty] == color:
        list2.append(1)
        tx = tx - 1
        ty = ty + 1
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
SIZE:
        list2.append(-1)
    else:
        list2.append(0)
    list2.reverse()
    list_b = list2 + list1

    return [list_h, list_v, list_s, list_b]


# 判断搜索范围是否超出边界，返回合法的搜索范围
def is_out(_min_x, _min_y, _max_x, _max_y):
    delta = 1
    if _min_x - delta < 1:
        min_tx = 1
    else:
        min_tx = _min_x - delta
    if _min_y - delta < 1:
        min_ty = 1
    else:
        min_ty = _min_y - delta
    if _max_x + delta > SIZE:
        max_tx = SIZE
    else:
        max_tx = _max_x + delta
    if _max_y + delta > SIZE:
        max_ty = SIZE
    else:
        max_ty = _max_y + delta
    return [min_tx, min_ty, max_tx, max_ty]


# 根据当前棋面向前搜索 2 步，利用极大极小算法及 alpha-beta 剪枝
def ai_go():
    global min_x, max_x, min_y, max_y, color_flag, matrix
    time_start = time.time()
    evaluate_matrix = [[0 for i in range(SIZE + 2)] for j in range(SIZE +
2)]  # 结点估值矩阵
    if step == 0:
```

```
        if color_choice == 0: #用户选择白棋
            add_chess((SIZE + 1) // 2, (SIZE + 1) // 2, color_flag)
    else:
        if step == 1 and color_choice == 1:
            if matrix[(SIZE + 1) // 2][(SIZE + 1) // 2] == 0:
                rx, ry = (SIZE + 1) // 2,(SIZE + 1) // 2
            else:
                if matrix[(SIZE + 1) // 2][(SIZE + 1) // 2] != 0 and
matrix[(SIZE + 1) // 2 + 1][(SIZE + 1) // 2 + 1] == 0:
                    rx, ry = (SIZE + 1) // 2 + 1,(SIZE + 1) // 2 + 1
        else:
            min_tx1, min_ty1, max_tx1, max_ty1 = is_out(min_x, min_y, max_x,
max_y)
            evaluate_matrix = [[0 for i in range(SIZE + 2)] for j in range(SIZE
+ 2)]  # 第一层的估值矩阵
            # evaluate_matrix2 = [[0 for i in range(SIZE + 2)] for j in
range(SIZE + 2)]  # 第二层的估值矩阵
            Max = -100000
            rx, ry = 0, 0
            for i in range(min_tx1, max_tx1 + 1):
                for j in range(min_ty1, max_ty1 + 1):
                    cut_flag = 0  # 剪枝标记
                    evaluate_matrix2 = [[0 for i in range(SIZE + 2)] for j
in range(SIZE + 2)]
                    if matrix[i][j] == 0:
                        matrix[i][j] = color_flag
                        min_tx2, min_ty2, max_tx2, max_ty2 = is_out(min_tx1,
min_ty1, max_tx1, max_ty1)
                        [list_h, list_v, list_s, list_b] = get_list(i, j,
color_flag)
                        eva1 = evaluate_node(list_h, list_v, list_s, list_b)
                        for ii in range(min_tx2, max_tx2 + 1):
                            for jj in range(min_ty2, max_ty2 + 1):
                                if matrix[ii][jj] == 0:
                                    matrix[ii][jj] = -color_flag
                                    [list_h, list_v, list_s, list_b] =
get_list(ii, jj, -color_flag)
                                    eva2 = -evaluate_node(list_h, list_v,
list_s, list_b) #* 2
                                    evaluate_matrix2[ii][jj] = eva2 + eva1
                                    matrix[ii][jj] = 0
                                    # 剪枝
                                    if evaluate_matrix2[ii][jj] < Max:
                                        evaluate_matrix[i][j] =
```

```
evaluate_matrix2[ii][jj]
                                    if alpha_beta_flag == 1:
                                        cut_flag = 1
                                        break
                            if cut_flag:
                                break
                        if cut_flag == 0:
                            Min = 100000
                            for ii in range(min_tx2, max_tx2 + 1):
                                for jj in range(min_ty2, max_ty2 + 1):
                                    if evaluate_matrix2[ii][jj] < Min and
matrix[ii][jj] == 0:
                                        Min = evaluate_matrix2[ii][jj]
                            evaluate_matrix[i][j] = Min
                            if Max < Min:
                                Max = Min
                                rx, ry = i, j
                        matrix[i][j] = 0
        time_end = time.time()
        print("Time cost:", round(time_end - time_start, 4), "s")
        add_chess(rx, ry, color_flag)


movements = []  # 记录移动步骤


# 添加棋子
def add_chess(x, y, color):
    global step, matrix
    step = step + 1
    movements.append((x, y, color, step))
    matrix[x][y] = color
    xy_range(x, y)
    game_is_or_not_over()


# 绘制文本
def draw_text(surf, text, size, x, y, color):
    font_name = "arial"
    font = pygame.font.SysFont(font_name, size)
    text_surface = font.render(text, True, color)
    text_rect = text_surface.get_rect()
    text_rect.center = (x, y)
    surf.blit(text_surface, text_rect)
```

```
# 绘制棋子
def draw_movements(surf):
    for move in movements:
        if move[2] == color_flag:
            if color_choice == 1:
                pygame.draw.circle(surf, WHITE, (move[0] * GRID_WIDTH,
move[1] * GRID_WIDTH), 16)
                draw_text(surf, str(move[3]), 10, move[0] * GRID_WIDTH,
move[1] * GRID_WIDTH, BLACK)
            else:
                pygame.draw.circle(surf, BLACK, (move[0] * GRID_WIDTH,
move[1] * GRID_WIDTH), 16)
                draw_text(surf, str(move[3]), 10, move[0] * GRID_WIDTH,
move[1] * GRID_WIDTH, WHITE)
        else:
            if color_choice == 1:
                pygame.draw.circle(surf, BLACK, (move[0] * GRID_WIDTH,
move[1] * GRID_WIDTH), 16)
                draw_text(surf, str(move[3]), 10, move[0] * GRID_WIDTH,
move[1] * GRID_WIDTH, WHITE)
            else:
                pygame.draw.circle(surf, WHITE, (move[0] * GRID_WIDTH,
move[1] * GRID_WIDTH), 16)
                draw_text(surf, str(move[3]), 10, move[0] * GRID_WIDTH,
move[1] * GRID_WIDTH, BLACK)


# 玩家行棋
def player_go(pos):
    x = round(pos[0] / GRID_WIDTH)
    y = round(pos[1] / GRID_WIDTH)
    if 1 <= x <= SIZE and 1 <= y <= SIZE and matrix[x][y] == 0:
        add_chess(x, y, -color_flag)
        return True


# 判断游戏是否结束
def game_is_or_not_over():
    global win_flag, game_over
    x = movements[-1][0]
    y = movements[-1][1]
    color = movements[-1][2]
    [list_h, list_v, list_s, list_b] = get_list(x, y, color)
```

```
    if sum(list_h[1:-1]) >= 5 or sum(list_v[1:-1]) >= 5 or
sum(list_s[1:-1]) >= 5 or sum(list_b[1:-1]) >= 5:
        win_flag = color
        game_over = True


# 开始界面显示
def show_go_screen(surf):
    global win_flag, movements, step, matrix, min_x, min_y, max_x, max_y,
game_over
    if win_flag != 0:
        if color_choice == 1: #选择黑棋
            draw_text(surf, "{0} win!!".format("WHITE" if win_flag == 1 else
"BLACK"), 64, WIDTH // 2, 350, RED)
        else:
            draw_text(surf, "{0} win!!".format("BLACK" if win_flag == 1 else
"WHITE"), 64, WIDTH // 2, 350, RED)
    else:
        screen.blit(background, back_rect)

    draw_text(surf, "Five In A Row", 64, WIDTH // 2, 100, BLUE)
    draw_text(surf, "Press space to start", 22, WIDTH // 2, 500, BLUE)
    pygame.display.flip()
    win_flag = 0
    movements = []
    step = 0
    matrix = [[0 for i in range(SIZE + 2)] for j in range(SIZE + 2)]
    min_x, min_y, max_x, max_y = 0, 0, 0, 0
    game_over = False
    waiting = True
    while waiting:
        clock.tick(FPS)
        for e in pygame.event.get():
            if e.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            elif e.type == pygame.KEYDOWN:
                if e.key == pygame.K_SPACE:
                    ai_go()
                    waiting = False


running = True
game_over = True
```

```
# 主循环
while running:
    if game_over:
        show_go_screen(screen)
    clock.tick(FPS)
    if color_choice == 0: #用户选择白棋，黑棋先走
        if step % 2 == 0:
            ai_go()
        else:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                elif event.type == pygame.MOUSEBUTTONDOWN:
                    player_go(event.pos)
    else:
        if step % 2 == 1:
            ai_go()
        else:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                elif event.type == pygame.MOUSEBUTTONDOWN:
                    player_go(event.pos)
    draw_background(screen)
    draw_movements(screen)
    pygame.display.flip()
pygame.quit()
sys.exit()
```

【实验结果及结论】

(1)本此实验采用极小极大算法和α-β剪枝实现了五子棋的人机对决，完成了五子棋游戏中机器方智能化，并记录的两种算法每一步棋的消耗时间。

(2)由于α-β剪枝应用在极小极大搜索树上，减去了那些不可能影响决策的分支，很大程度上减少了搜索所需的时间。

三．参考文献

1、https://baike.baidu.com/item/%E4%BA%94%E5%AD%90%E6%A3%8B/130266?fr=aladdin

2、https://blog.csdn.net/lihongxun945/article/details/50668253

3、基于α-β剪枝算法的智能五子棋

四．小结

1、本次实验对极小极大算法和α-β剪枝算法有了更深入的理解，对α-β剪枝
的整个剪枝过程更加清晰。

2、本实验采用了五子棋游戏的人机对决，由于棋局局势、评估函数较为复杂，最后搜索的深度比较浅，使机器一方不够智能，只能算一般般的五子棋棋手。

3、本次作业提高了小组成员的编程能力、团队协作能力。