Lab 2：基于搜索的人工智能：五子棋AI

231880101 孙俊晖

1 引言

1.1 背景

五子棋是一种古老而富有策略性的棋类游戏，因其简单易学、变化无穷而深受人们喜爱。它满足双人零和博弈的特点，即五子棋是信息确定、全局可观察、竞争对手轮流行动、输赢收益零和假设下的双人博弈问题。在学习完博弈搜索后，我想完成一个基于博弈搜索(对抗搜索)的人工智能：五子棋AI。

1.2 目标

利用Python语言和pygame库，以及所学的经典搜索算法完成一个AI五子棋游戏，该游戏包括**棋盘与棋子设计、用户交互、判断胜负、AI对手、游戏循环、可视化界面**等部分。

2 方法

2.1 技术栈

开发环境：Visual Studio Code 1.93.0

编程语言：Python 3.7.9 64-bit

2.2 棋盘与棋子设计

2.2.1 棋盘设计

棋盘是一个15 * 15的网格，每个格子可以放置一个棋子。在代码中，我用一个15 * 15的二维列表board来储存每个格子上棋子的状态，board [i] [j]储存的是第i + 1行第j + 1列的格子上棋子的状态，board [i] [j] = 0表示在这个格子上没有棋子，board [i] [j] = 1表示这个格子上是黑子，board [i] [j] = 2表示在这个格子上是白子。

2.2.2 棋子设计

棋子一共两种：黑棋和白棋。在代码中，我用一个列表exist_chess来存储棋盘上所有棋子的状态，包括所有棋子的坐标和颜色。在exist_chess中添加黑子和白子的代码示例如下：

```
if len(exist_chess)%2==0:#黑子
                exist_chess.append([[x,y],black_chess_color])
            else:#白子
                exist_chess.append([[x,y],white_chess_color])
```

2.3 用户交互

用户首先点击棋盘下方的"choose black"或"choose white"按钮来选择自己执黑还是执白(规定执黑棋的一方执先手)，然后在轮到自己落子时，在棋盘中点击想要落子的格子即可落子；在游戏中如果想要重新开始新的对局，点击棋盘下方的"restart"按钮即可开始新的对局；如果想要退出游戏，点击游戏窗口右上角的"x"号关闭游戏窗口即可。下面是实现用户交互的关键代码：

1.用户落子的实现

```
#判断鼠标的位置是否位于棋盘内部
def is_valid(x,y):
    if x > 800 or y > 800:
      return False
    return True
#根据鼠标的位置定位到棋盘上的点
def find_pos(x,y):
    for i in range(22,800,54):
        for j in range(22,800,54):
            if x >= i - 27 and x <= i + 27 and y >= j - 27 and y <= j + 27:
                return i,j
    return x,y
#判断该位置有没有棋子
def check_exist(x,y):
    global exist_chess
    for val in exist_chess:
        if val[0][0] == x and val[0][1] == y:
            return True
    return False
#添加棋子
def add_chess():
    global turn
    global flag
    global tim
    x,y = pygame.mouse.get_pos()
    x,y = find_pos(x,y)
    if is_valid(x,y) and not check_exist(x,y) and game_start:
      pygame.draw.rect(screen,rect_color,[x-20,y-20,40,40],2,1)
    mouse_pressed = pygame.mouse.get_pressed()
    if mouse_pressed[0] and tim==0:
        flag=True
        if not check_exist(x,y) and is_valid(x,y) and game_start and turn == player_num:#判断是否可以落子，再落子
            if len(exist_chess)%2==0:#黑子
                exist_chess.append([[x,y],black_chess_color])
            else:#白子
                exist_chess.append([[x,y],white_chess_color])
             #交替落子
            if turn == 1:
              turn = 2
            else:
              turn = 1
    #鼠标左键延时作用,延时200ms
    if flag:
        tim += 1
    if tim == 200:
        flag = False
        tim = 0
```

2.按钮功能的实现

```
for event in pygame.event.get():
        if event.type in (QUIT,KEYDOWN):#退出游戏
            sys.exit()
        elif event.type == MOUSEBUTTONDOWN:
            if button_black.collidepoint(event.pos) and not game_start:#执黑
              player_num = 1
              ai_num = 2
              exist_chess.clear()
              game_start = True
              board = np.zeros((15, 15), dtype=int)
              win_text = None
              print("Choose Black!")
            elif button_white.collidepoint(event.pos) and not game_start:#执白
              player_num = 2
              ai_num = 1
              exist_chess.clear()
              game_start = True
              board = np.zeros((15, 15), dtype=int)
              win_text = None
              print("Choose White!")
            elif button_restart.collidepoint(event.pos):#重新游戏
              turn = 1
              game_start = False
              exist_chess.clear()
              board = np.zeros((15, 15), dtype=int)
              win_text = None
              print("Restart!")
```

2.4 判断胜负

代码中使用了函数check_win()来判断游戏的胜负，check_win()函数通过判断每个方向上是否存在五子连珠的情况，如果有的话，返回五子连珠的棋子的颜色(黑子:1 白子:2);如果棋盘上不存在五子连珠的情况，返回0(表示无人获胜)。

```
directions = [(0, 1),(1, 0),(1, 1),(1, -1),(0, -1),(-1, 0),(-1, 1),(-1, -1)]
def check_win():
  for dx, dy in directions:
    for x in range(15):
      for y in range(15 - abs(dy)):
        temp_count = 1
        for step in range(1, 5):
          nx, ny = x + step * dx, y + step * dy
          if 0 <= nx < 15 and 0 <= ny < 15 and board[nx][ny] and board[x][y] and board[nx][ny] == board[x][y]:
            temp_count += 1
          else:
            break
        if temp_count == 5:
          return board[x][y]
  return 0  # 无人获胜
```

2.5 AI对手

这个部分是这个游戏的核心。我的想法是，先计算当前棋盘上敌我双方的棋型(活五、活四、冲四、活三、眠三、活二、眠二的个数)，基于检测到的棋型数量建立评估函数，AI的底层逻辑就是每次轮到自己落子时，搜索所有可以落子的位置，通过评估函数找到最优的落子位置。

首先，我需要计算当前棋盘上敌我双方每种棋型的个数。我使用了一个2 * 8的二维列表count来储存敌我双方每种棋型的个数，对于count[i] [j] ,i=0表示存储的是黑棋的棋型，i = 1表示存储的是白棋的棋型，j对应的是相应棋型的索引值(空=0,眠二=1,活二 = 2,眠三=3,活三=4,冲四=5,活四=6,活五=7), 如count[0] [4]存储的就是黑棋活三的个数。

我先定义了一个辅助函数getLine来获取以一个棋子为中心，在不同方向上连续9个格子的状态(如果超出边界则将超出的部分置为敌方的棋子):

```
direction_ = [(1, 0), (0, 1), (1, 1), (1, -1)] #一共四个方向
def getLine(x, y, direction, me, you):#direction取自direction_
  global board
  line = [0 for _ in range(9)]
  next_x = x + (-5 * direction[0])
  next_y = y + (-5 * direction[1])
  for i in range(9):
    next_x += direction[0]
    next_y += direction[1]
    if (next_x >= 15 or next_x < 0 or next_y < 0 or next_y >= 15):
      line[i] = you
    else:
      line[i] = board[next_x][next_y]
  return line
```

我利用了函数evaluate_type来计算不同棋型的个数,在evlauate_type函数中利用了上述辅助函数getLine：

```
def evaluate_type(x,y,dir_num,dir,me,you):
    global count
    line = getLine(x, y, dir, me, you)
    l_index = 4
    r_index = 4
    while r_index < 8:
        if line[r_index + 1] != me:
          break
        r_index += 1
    while l_index > 0:
        if line[l_index - 1] != me:
          break
        l_index -= 1
    l_range = l_index
    r_range = r_index
    while r_range < 8:
        if line[r_range + 1] == you:
            break
        r_range += 1
    while l_range > 0:
        if line[l_range - 1] == you:
            break
        l_range -= 1
    chess_range = r_range - l_range + 1
#chess_range < 5说明该方向上左右两边两个敌方的子中间夹着最多四个子，这种情况不属于任何棋型
    if chess_range < 5:
        record_move(x, y, l_range, r_range, dir_num, dir)
        return 0
    record_move(x, y, l_index, r_index, dir_num, dir)
    m_range = r_index - l_index + 1
    if m_range == 5:
        count[me-1][7] += 1

    if m_range == 4:
        #两边都为空则为活四 XMMMMX
        if line[l_index - 1] == 0 and line[r_index + 1] == 0:
          count[me-1][6] += 1
        #一边为空另一边为敌方棋子则为冲四 XMMMMP PMMMMX
        elif line[l_index - 1] == 0 or line[r_index + 1] == 0:
          count[me-1][5] += 1

    if m_range == 3:
        if line[l_index - 1] == 0:
            if line[l_index - 2] == me:  #冲四 MXMMM
                record_move(x, y, l_index - 2, l_index - 1, dir_num, dir)
                count[me-1][5] += 1
        if line[r_index + 1] == 0:
            if line[r_index + 2] == me:  #冲四 MMMXM
                record_move(x, y, r_index + 1, r_index + 2, dir_num, dir)
                count[me-1][5] += 1
        if line[l_index - 2] == me or line[r_index + 2] == me:
            pass
        elif line[l_index - 1] == 0 and line[r_index + 1] == 0:
            if chess_range > 5:  #活三 XMMMXX XXMMMX
                count[me-1][4] += 1
            else:  #眠三 PXMMMXP
                count[me-1][3] += 1
        elif line[l_index - 1] == 0 or line[r_index + 1] == 0:  #眠三 PMMMX XMMMP
            count[me-1][3] += 1

    if m_range == 2:
        left_three = False
        right_three = False
        if line[l_index - 1] == 0:
            if line[l_index - 2] == me:
                record_move(x, y, l_index - 2, l_index - 1, dir_num, dir)
                if line[l_index - 3] == 0:
                    if line[r_index + 1] == 0:  #活三 XMXMMX
                        count[me-1][4] += 1
                    else:  #眠三 XMXMMP
                        count[me-1][3] += 1
                    left_three = True
                elif line[l_index - 3] == you:
                    if line[r_index + 1] == 0:  #眠三 PMXMMX
                        count[me-1][3] += 1
                    left_three = True
```

```python
        if line[r_index + 1] == 0:
            if line[r_index + 2] == me:
                if line[r_index + 3] == me:  #冲四 MMXMM
                    record_move(x, y, r_index + 1, r_index + 2, dir_num, dir)
                    count[me-1][5] += 1
                    right_three = True
                elif line[r_index + 3] == 0:
                    if line[l_index - 1] == 0:  #活三 XMMXMX
                        count[me-1][4] += 1
                    else:  #眠三 PMMXMX
                        count[me-1][3] += 1
                    right_three = True
                elif line[l_index - 1] == 0:  #眠三 XMMXMP
                    count[me-1][3] += 1
                    right_three = True
        if left_three or right_three:
            pass
        elif line[l_index - 1] == 0 and line[r_index + 1] == 0:  #活二 XMMX
            count[me-1][2] += 1
        elif line[l_index - 1] == 0 or line[r_index + 1] == 0:  #眠二 PMMX, XMMP
            count[me-1][1] += 1

    if m_range == 1:
        if line[l_index - 1] == 0:
          if line[l_index - 2] == me and line[l_index - 3] == 0 and line[r_index + 1] == you: #眠二 XMXMP
              count[me-1][1] += 1
          if line[r_index + 1] == 0:
            if line[r_index + 2] == me:
              if line[r_index + 3] == 0:
                if line[l_index-1]==0:  #活二 XMXMX
                  count[me-1][2] += 1
                else:  #眠二 PMXMX
                  count[me-1][1] += 1
            elif line[r_index + 2] == 0:
              if line[r_index + 3] == me and line[r_index + 4] == 0:  #眠二 XMXXMX
                count[me-1][2] += 1

    # 以上都不是则为空棋型
    return 0
```

在函数evaluate_type中，我还利用了一个辅助函数record_move来标记已经遍历过的格子，避免重复计算(我使用了一个二维列表record来记录每个格子是否被遍历过)

```python
def record_move(x, y, left, right, dir_index, direction):
    global record
    x_ = x + (-5 + left) * direction[0]
    y_ = y + (-5 + left) * direction[1]
    for i in range(left, right):
        x_ += direction[0]
        y_ += direction[1]
        record[x_][y_][dir_index] = 1
```

在计算完当前棋盘上的棋型后，便可基于检测到的棋型数量建立评估函数getScore,根据敌我双方的棋型进行评分(威胁越大的棋型对应的分数越大，活五、活四、多个冲四/活三这种必胜棋型对应的分数极高):

```python
#空=0,眠二=1,活二 = 2,眠三=3,活三=4,冲四=5,活四=6,活五=7
def getScore(player_chess, enemy_chess):
        player_score = 0
        enemy_score = 0
        if player_chess[7] > 0:
            return (50000, 0)
        if enemy_chess[7] > 0:
            return (0, 50000)
        if enemy_chess[6] > 0:
            return (0, 10000)
        if enemy_chess[5] > 0:
            return (0, 9500)
        if player_chess[6] > 0 or player_chess[5] >= 2:
            return (9300, 0)
        if player_chess[5] > 0 and player_chess[4] > 0:
            return (9200, 0)
        if enemy_chess[4] > 0 and player_chess[5] == 0:
            return (0, 9100)
        if (player_chess[4] > 1 and enemy_chess[4] == 0 and enemy_chess[3]
== 0):
            return (9000, 0)
        if player_chess[5] > 0:
            player_score += 2000
        if player_chess[4] > 1:
            player_score += 500
        elif player_chess[4] > 0:
            player_score += 100
        if enemy_chess[4] > 1:
            enemy_score += 2000
        elif enemy_chess[4] > 0:
            enemy_score += 400
        if player_chess[3] > 0:
            player_score += player_chess[3] * 10
        if enemy_chess[3] > 0:
            enemy_score += enemy_chess[3] * 10
        if player_chess[2] > 0:
            player_score += player_chess[2] * 4
        if enemy_chess[2] > 0:
            enemy_score += enemy_chess[2] * 4
        if player_chess[1] > 0:
            player_score += player_chess[1] * 4
        if enemy_chess[1] > 0:
            enemy_score += enemy_chess[1] * 4
        return (player_score, enemy_score)
```

有了评估函数后，就可以计算当前的分数了。我使用了函数evaluate来计算当前的分数：

```
#计算每个点在指定方向上的棋型个数
def evaluatePoint(x, y, me, you):
  global record
  global direction
  for i in range(4):
    if record[x][y][i] == 0:
      evaluate_type(x, y, i, direction[i], me, you)
#计算当前分数
def evaluate(current_turn):
  global record
  global count
  record = [[[0,0,0,0]for _ in range(15)] for _ in range(15)]
  count = [[0 for _ in range(8)] for _ in range(2)]
  if current_turn == 1:
    player = 1
    enemy = 2
  else:
    player = 2
    enemy = 1
  for x in range(15):
    for y in range(15):
      if board[x][y] == player:
        evaluatePoint(x, y, player, enemy)
      elif board[x][y] == enemy:
        evaluatePoint(x, y, enemy, player)
  playerchess = count[player-1]
  enemychess = count[enemy-1]
  mscore, oscore = getScore(playerchess, enemychess)
  return (mscore - oscore)
```

然后，我选择使用极大极小搜索算法进行搜索得到最优解，并在极小极大搜索函数alpha_beta_search中使用alpha-beta剪枝来减少搜索空间并提高搜索效率：

```
#alpha_beta_search
def alpha_beta_search(depth, alpha, beta, is_maximizing,current_turn):
    global board
    if depth == 0:
        return evaluate(3-current_turn), None, None
    moves = get_empty()
    best_move = None
    best_score = -99999 if is_maximizing else 99999
    for score_heuristic, x, y in moves:
        board[x][y] = current_turn
        score, move_x, move_y = alpha_beta_search(depth - 1, alpha, beta, not is_maximizing,3-current_turn)  #current_turn的取值为1或2，用3-current_turn可实现交替落子
        board[x][y] = 0  # 回溯
        if is_maximizing:  #极大搜索
            if score > best_score:
                best_score = score
                best_move = (x, y)
                alpha = max(alpha, score)
                if beta <= alpha:  # 剪枝
                    break
        else:  #极小搜索
            if score < best_score:
                best_score = score
                best_move = (x, y)
                beta = min(beta, score)
                if beta <= alpha:  # 剪枝
                    break
    return best_score, best_move[0], best_move[1]
```

最后，利用封装好的搜索函数search()就得到了需要的AI函数：

```
def search():
    global turn
    global board
    depth = 4 #搜索深度
    alpha = -99999
    beta = 99999
    score, x, y = alpha_beta_search(depth, alpha, beta, True,turn)
    return (score, x, y)
def AI():
  max_score,ai_x,ai_y = search()
  if ai_num == 1: #ai执黑
    exist_chess.append([[ai_x*54+22,ai_y*54+22],black_chess_color])
  else: #ai执白
    exist_chess.append([[ai_x*54+22,ai_y*54+22],white_chess_color])
  return
```

2.6 游戏循环

在游戏循环中，除了包括上述提到的用户交互、判断胜负，还会利用全局变量turn来实现交替落子。turn的取值为1或2，turn为1表示当前为黑方落子，turn为2表示当前为白方落子，每次循环均会改变turn的值，以实现交替落子。在游戏循环中，还利用了一个全局变量game_start来判断游戏是否在进行，若游戏未开始或胜负已分，game_start置为False,根据add_chess()函数中落子条件的判定，此时不允许进行落子操作。

游戏循环的代码如下：

```
while True:
    #按钮功能
    for event in pygame.event.get():
        if event.type in (QUIT,KEYDOWN):
            sys.exit()
        elif event.type == MOUSEBUTTONDOWN:
            if button_black.collidepoint(event.pos) and not game_start:
              player_num = 1
              ai_num = 2
              exist_chess.clear()
              game_start = True
              board = np.zeros((15, 15), dtype=int)
              win_text = None
              print("Choose Black!")
            elif button_white.collidepoint(event.pos) and not game_start:
              player_num = 2
              ai_num = 1
              exist_chess.clear()
              game_start = True
              board = np.zeros((15, 15), dtype=int)
              win_text = None
              print("Choose White!")
            elif button_restart.collidepoint(event.pos):
              turn = 1
              game_start = False
              exist_chess.clear()
              board = np.zeros((15, 15), dtype=int)
              win_text = None
              print("Restart!")
    #绘制棋盘
    draw_map()

    #实现交替落子
    if game_start:
      if turn == ai_num:
        AI()
        if turn == 1:
          turn = 2
        else:
          turn = 1

    add_chess() #添加棋子

    update_chess() #更新棋子状态

    winner = check_win() #判断胜负

    #如果游戏胜负已分，不允许再进行落子操作
    if winner:
        if winner == 1:
            win_text = font.render("Black Wins!", True, (255, 0, 0))
            game_start = False

        else:
            win_text = font.render("White Wins!", True, (255, 0, 0))
            game_start = False
    pygame.display.update()
```

函数add_chess()中判断是否可以落子的部分代码如下：

```
if not check_exist(x,y) and is_valid(x,y) and game_start and turn == player_num:#判断是否可以落子，再落子
            if len(exist_chess)%2==0:#黑子
                exist_chess.append([[x,y],black_chess_color])
            else:#白子
                exist_chess.append([[x,y],white_chess_color])
```

游戏循环中更新棋子状态的函数update_chess()的实现如下：

```python
def update_chess():
  global exist_chess
  for pos, color in exist_chess:
    x = (pos[0] - 22) // 54
    y = (pos[1] - 22) // 54
    if color == black_chess_color:
      board[x][y] = 1
    elif color == white_chess_color:
      board[x][y] = 2
```

2.7 可视化界面

在代码中，我利用pygame库中的函数实现了可视化界面，包括棋盘、棋子、按钮以及获胜文本(分出胜负后会在屏幕中央显示"Black wins!"或"White wins! ")的绘制。可视化界面主要在draw_map()函数中实现，该函数在每次游戏循环时均会被调用。

```python
button_black = pygame.Rect(50,820,200,70)
button_white = pygame.Rect(550,820,200,70)
button_restart = pygame.Rect(300,820,150,70)
black_color = (255, 255, 255)
text_color = (0, 0, 0)
button_font = pygame.font.SysFont(['方正粗黑宋简体','microsoftsansserif'],30)
button_text = button_font.render("Choose Black", True, text_color)
button_text2 = button_font.render("Choose White", True, text_color)
button_text3 = button_font.render("Restart", True, text_color)
def draw_map():
    #清屏
    screen.fill(screen_color)
    #绘制"choose black"按钮
    pygame.draw.rect(screen, black_color, button_black)
    screen.blit(button_text, (button_black.centerx - button_text.get_width() / 2, button_black.centery - button_text.get_height() / 2))
    #绘制"choose white"按钮
    pygame.draw.rect(screen, black_color, button_white)
    screen.blit(button_text2, (button_white.centerx - button_text2.get_width() / 2, button_white.centery - button_text2.get_height() / 2))
    #绘制"restart"按钮
    pygame.draw.rect(screen, black_color, button_restart)
    screen.blit(button_text3, (button_restart.centerx - button_text3.get_width() / 2, button_restart.centery - button_text3.get_height() / 2))
    if not game_start and win_text: #绘制获胜文本
      screen.blit(win_text, (200, 330))
    for i in range(22,800,54):#绘制棋盘上的线，边界的四条线要略粗一点
        if i == 22 or i == 778:
            pygame.draw.line(screen,line_color,[i,22],[i,778],4)
        else:
            pygame.draw.line(screen,line_color,[i,22],[i,778],2)
        if i == 22 or i == 778:
            pygame.draw.line(screen,line_color,[22,i],[778,i],4)
        else:
            pygame.draw.line(screen,line_color,[22,i],[778,i],2)
    #绘制棋盘上的五个星位
    pygame.draw.circle(screen, line_color,[400,400], 8,0)
    pygame.draw.circle(screen, line_color,[184,184], 8,0)
    pygame.draw.circle(screen, line_color,[616,616], 8,0)
    pygame.draw.circle(screen, line_color,[184,616], 8,0)
    pygame.draw.circle(screen, line_color,[616,184], 8,0)
```

3 总结

通过本次实验，我成功实现了一个基于极小极大搜索算法的五子棋AI游戏。实验过程中，我学会了如何将学习到的数据结构和经典搜索算法应用到实际的问题中。此外，在创建可视化界面的过程中，我学会了如何使用pygame库的部分库函数。总的来说，本次实验是一次非常有意义的学习经历，不仅锻炼了我的编程技能，还让我对人工智能和博弈搜索有了更深入的理解。

4 参考文献

1.https://blog.csdn.net/m0_46403734/article/details/136535806

2.https://zhuanlan.zhihu.com/p/593096861