**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    （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    (2)进入游戏界面，选择先手还是后手        (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    【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。 |