**1\_里程碑任务报告**

**步骤一：**

**（1）题目：**

写程序从以下网址取得下一步的指示http://183.175.12.27:8001/step\_01

**（2）解决思路：**

使用 requests库发送 GET 请求到网址，获取并打印返回的下一步指示。在请求中不使用代理，并处理可能的错误和超时。

**（3）代码：**

1. import requests

2. response = requests.get("http://183.175.12.27:8004/step\_01", proxies={"http": None, "https": None})

3. print(response.text)

**步骤二：**

**（1）题目：**

请你告诉服务器你的名字是什么，你的学号是什么。名字使用字段name，学号使用字段student\_number。请把以上信息以GET方式交到以下地址：

http://183.175.12.27:8001/step\_02

提示：对你来说处理中文可能有点困难，你可以给自己起个英文名字。

**（2）解决思路：**

通过 requests.get() 发送 GET 请求，将名字和学号作为 URL 参数传递给服务器。请求成功后，使用 response.json() 获取并打印服务器返回的 JSON 数据。

**（3）代码：**

1. import requests

2. params = {

3. "name": "name",

4. "student\_number": "1234"

5. }

6. response = requests.get("http://183.175.12.27:8004/step\_02", params=params)

7. print(response.json())

**步骤三：**

**（1）题目：**

阅读以下材料，找出提示。

链接: https://pan.baidu.com/s/1D5mEOdu-3orSUhNlSUp8ag 密码: 7j4s

**（2）解决思路：**

阅读电子书PDF。

**（3）代码：**

下一步网址：

http://183.175.12.27:8004/context/f64710b4861e6ff10678036548bf7afa

**步骤四：**

**（1）题目：**

实现模幂算法，通过服务器的检验。

访问http://183.175.12.27:8004/step\_04服务器会给你10个问题，每个问题包含三个数(a,b,c)，请给出a^b%c的值。返回值写入字段ans，10个数字用逗号,隔开，提交到http://183.175.12.27:8001/step\_04。

提示：注意逗号必须是英文逗号。

**（2）解决思路：**

通过向服务器请求10个问题，每个问题包含三个数 (a, b, c)，使用快速幂算法计算每个问题的模幂结果 (a^b%c)，然后将所有答案以逗号分隔的格式提交到服务器。程序通过记录运行时间来优化效率，确保结果格式符合要求（英文逗号分隔）。

**（3）代码：**

1. import requests

2. def ksm(x):

3. result = 1

4. while (x[1] > 0):

5. if (x[1] & 1):

6. result = result \* x[0] % x[2]

7. x[1] = int(x[1] / 2)

8. x[0] = x[0] \* x[0] % x[2]

9. return result

10. answer = ''

11. getHtml = requests.get("http://183.175.12.27:8004/step\_04/")

12. for i in eval(getHtml.json()['questions']):

13. answer += str(ksm(i)) + ','

14. param = {'ans': answer[:-1]}

15. getHtml = requests.get("http://183.175.12.27:8004/step\_04/", params=param)

16. print(getHtml.text)

**步骤五：**

**（1）题目：**

把你的用户名作为user字段，加密后的密码作为password字段，提交到http://183.175.12.27:8001/step\_05

不要急着找下一步的地址。作为一个确定性的程序，你的密码每次加密得到的结果都是一样的，我们可以为密码加噪，扰乱窃密者。在加密时，我们可以编码的信息不大于公钥中的第二个数字，这个数字很大，因此我们可以编码很长的信息。在c/c++中字符串以字符\0作为结束标志，\0以后的所有数据都会被忽略掉，因此我们可以在密码后面添加一个字符\0，之后连接上一个随机生成的没用的字符串。这样每次都会产生不同的密文，就能更有效的保护你的密码了。去试试吧！

回到我们得到的信息，它也是被加密的，你需要解密，服务器使用它自己的私钥加密了数据，因为秘钥是对称的，你可以用公钥解密它。

**（2）解决思路：**

通过快速幂模算法加密密码，将加密后的密码和用户名作为参数提交给服务器。服务器返回加密信息后，使用公钥解密该信息。解密过程中将16进制的加密数据转换为数字，使用快速幂模算法恢复原始数据，并将数字转换回字符串，从而获得服务器返回的原始信息。

**（3）代码：**

1. import requests

2. def calculate\_mod\_power(values):

3. base, exponent, mod = values

4. computed\_value = 1

5. while exponent:

6. if exponent % 2:

7. computed\_value = (computed\_value \* base) % mod

8. exponent //= 2

9. base = (base \* base) % mod

10. return computed\_value

11. def encode\_string\_to\_integer(text):

12. numeric\_value = 0

13. text\_length = len(text)

14. for index, char in enumerate(text):

15. position = text\_length - index - 1

16. numeric\_value += ord(char) \* (256 \*\* position)

17. return numeric\_value

18. def decode\_integer\_to\_string(numeric\_value):

19. char\_values = []

20. while numeric\_value:

21. char\_values.append(numeric\_value % 256)

22. numeric\_value //= 256

23. char\_values.reverse()

24. return ''.join(chr(value) for value in char\_values)

25. ENCRYPTION\_EXPONENT = 65537

26. ENCRYPTION\_MODULUS = 135261828916791946705313569652794581721330948863485438876915508683244111694485850733278569559191167660149469895899348939039437830613284874764820878002628686548956779897196112828969255650312573935871059275664474562666268163936821302832645284397530568872432109324825205567091066297960733513602409443790146687029

27. user\_password = ‘12345’

28. password\_numeric = encode\_string\_to\_integer(user\_password)

29. encrypted\_password = calculate\_mod\_power([password\_numeric, ENCRYPTION\_EXPONENT, ENCRYPTION\_MODULUS])

30. encrypted\_hex = hex(encrypted\_password)

31. print(f"加密后的密码: {encrypted\_hex}")

32. request\_params = {

33. 'user': 'name',

34. 'password': encrypted\_hex

35. }

36. response = requests.get('http://183.175.12.27:8004/step\_05/', params=request\_params)

37. hex\_prefix = '0x'

38. response\_message = response.json()['message']

39. encrypted\_response = int(hex\_prefix + response\_message, 16)

40. decrypted\_value = calculate\_mod\_power([encrypted\_response, ENCRYPTION\_EXPONENT, ENCRYPTION\_MODULUS])

41. decrypted\_text = decode\_integer\_to\_string(decrypted\_value)

42. print(decrypted\_text)

**步骤六：**

**（1）题目：**

实现从坐标表示到图形表示的算法，通过服务器的检验。

访问http://183.175.12.27:8001/step\_06服务器会给你一个坐标表示，请给出每一步的棋盘图形表示，如上面的例子共有15步，请将每一步的棋盘图形表示，按顺序写入字段ans，每步用逗号,隔开，图形表示中换行可省略。结果提交到http://183.175.12.27:8001/step\_06。

**（2）解决思路：**

从服务器获取坐标数据后，初始化16x16棋盘并通过坐标转换确定每步棋的位置。轮流在棋盘上放置 "x" 或 "o"，记录每步棋的棋盘状态，最后按顺序提交所有棋盘状态。

**（3）代码：**

1. import requests

2. def calculate\_position(coordinate):

3. col = ord(coordinate[0]) - ord('a')

4. row = ord(coordinate[1]) - ord('a')

5. return int(col \* 16 + row)

6. api\_endpoint = "http://183.175.12.27:8004/step\_06/"

7. response = requests.get(api\_endpoint)

8. move\_sequence = response.json()['questions']

9. chess\_board = '\n'.join(['.' \* 15 for \_ in range(15)])

10. move\_count = 0

11. game\_states = []

12. for move\_index in range(0, len(move\_sequence), 2):

13. current\_move = move\_sequence[move\_index:move\_index + 2]

14. position = calculate\_position(current\_move)

15. board\_chars = list(chess\_board)

16. piece = 'x' if move\_count % 2 == 0 else 'o'

17. board\_chars[position] = piece

18. chess\_board = ''.join(board\_chars)

19. move\_count += 1

20. game\_states.append(chess\_board)

21. result\_data = {

22. 'ans': ','.join(game\_states)

23. }

24. final\_response = requests.get(api\_endpoint, params=result\_data)

25. print(final\_response.text)

**步骤七：**

**（1）题目：**

实现棋型算法，通过服务器的检验。访问http://183.175.12.27:8001/step\_07服务器会给你一个棋盘的坐标表示（board字段），以及一系列要考察的点的坐标（coord字段），坐标之间用逗号,隔开，整体用[]包围，事实上它是一个JSON的数组。

请给出每个考察点的四条线上的棋型。只考虑周围5个格子的范围。用.表示空白位置，x表示黑棋，o表示白棋。每个考察点返回四个表示棋型的字符串。所有考察点的棋型字符串按顺序返回，其中单个考察点的棋型字符串顺序、棋型字符串中棋子的顺序不做规定。

返回值写入ans字段，提交到http://183.175.12.27:8001/step\_07

**（2）解决思路：**

从服务器获取棋盘和考察点，计算每个考察点四个方向的棋型（横、纵、对角线），并返回这些棋型的字符串。最终将所有棋型拼接并提交给服务器。

**（3）代码：**

1. import requests

2. def calculate\_position(position):

3. col\_idx = ord(position[0]) - ord('a')

4. row\_idx = ord(position[1]) - ord('a')

5. return row\_idx \* 15 + col\_idx

6. def extract\_directions(position, chess\_state):

7. directions = ['', '', '', '']

8. center\_col = ord(position[0]) - ord('a')

9. center\_row = ord(position[1]) - ord('a')

10. for offset in range(-4, 5):

11. row = center\_row

12. col = center\_col + offset

13. if 0 <= row < 15 and 0 <= col < 15:

14. idx = row \* 15 + col

15. directions[0] += chess\_state[idx]

16. else:

17. directions[0] += ' '

18. row = center\_row + offset

19. col = center\_col

20. if 0 <= row < 15 and 0 <= col < 15:

21. idx = row \* 15 + col

22. directions[2] += chess\_state[idx]

23. else:

24. directions[2] += ' '

25. row = center\_row + offset

26. col = center\_col + offset

27. if 0 <= row < 15 and 0 <= col < 15:

28. idx = row \* 15 + col

29. directions[1] += chess\_state[idx]

30. else:

31. directions[1] += ' '

32. row = center\_row - offset

33. col = center\_col + offset

34. if 0 <= row < 15 and 0 <= col < 15:

35. idx = row \* 15 + col

36. directions[3] += chess\_state[idx]

37. else:

38. directions[3] += ' '

39. return directions

40. response = requests.get("http://183.175.12.27:8004/step\_07/")

41. data = response.json()

42. move\_sequence = data['board']

43. target\_positions = data['coord']

44. chess\_board = '.' \* (15 \* 15)

45. current\_turn = 0

46. for i in range(0, len(move\_sequence), 2):

47. move = move\_sequence[i:i + 2]

48. idx = calculate\_position(move)

49. piece = 'x' if current\_turn % 2 == 0 else 'o'

50. chess\_board = chess\_board[:idx] + piece + chess\_board[idx + 1:]

51. current\_turn += 1

52. result = []

53. for position in target\_positions:

54. lines = extract\_directions(position, chess\_board)

55. result.append(','.join(lines))

56. final\_answer = ','.join(result)

57. response = requests.get('http://183.175.12.27:8004/step\_07', params={'ans': final\_answer})

58. print(response.text)

59.

**步骤八：**

**（1）题目：**

实现按照评分规则选择落子位置的算法，通过服务器的检验。

访问http://183.175.12.27:8001/step\_08服务器会给你几个棋局的坐标表示，保存到questions字段，以JSON数组表示。请给出分数最大的一个落子位置，写入到ans字段，按顺序给出坐标，坐标之间用逗号隔开。服务器不会告诉你，你是白棋还是黑棋，你需要根据规则自己确定。

**（2）解决思路：**

实现了按照规则下棋的功能。

**（3）代码：**

1. import requests as req

2. def calculate\_index(position):

3. return (ord(position[0]) - ord('a')) \* 16 + ord(position[1]) - ord('a')

4. def generate\_all\_positions():

5. positions = []

6. for row in range(15):

7. for col in range(16):

8. positions.append(chr(row + 97) + chr(col + 97))

9. return positions

10. def extract\_patterns(position, board\_state):

11. patterns = ['', '', '', '']

12. for offset in range(-4, 5):

13. col = ord(position[1]) - ord('a') + offset

14. if 0 <= col < 15:

15. patterns[0] += board\_state[(ord(position[0]) - ord('a')) \* 16 + col]

16. else:

17. patterns[0] += ' '

18. row = ord(position[0]) - ord('a') + offset

19. if 0 <= row < 15:

20. patterns[2] += board\_state[row \* 16 + ord(position[1]) - ord('a')]

21. else:

22. patterns[2] += ' '

23. row = ord(position[0]) - ord('a') + offset

24. col = ord(position[1]) - ord('a') + offset

25. if 0 <= row < 15 and 0 <= col < 15:

26. patterns[1] += board\_state[row \* 16 + col]

27. else:

28. patterns[1] += ' '

29. row = ord(position[0]) - ord('a') + offset

30. col = ord(position[1]) - ord('a') - offset

31. if 0 <= row < 15 and 0 <= col < 15:

32. patterns[3] += board\_state[row \* 16 + col]

33. else:

34. patterns[3] += ' '

35. return patterns

36. def determine\_player\_order(moves):

37. return 'MO' if (len(moves) // 2) % 2 == 0 else 'OM'

38. def pattern\_scoring\_system():

39. scoring = {

40. ("CMMMM", "MCMMM", "MMCMM", "MMMCM", "MMMMC"): 10000,

41. ("COOOO", "OOOOC"): 6000,

42. (".CMMM.", ".MCMM.", ".MMCM.", ".MMMC."): 5000,

43. ("COOO.", ".OOOC", ".OOCO.", ".OCOO."): 2500,

44. ("OCMMM.", "OMCMM.", "OMMCM.", "OMMMC.", ".CMMMO", ".MCMMO", ".MMCMO", ".MMMCO"): 2000,

45. (".MMC.", ".MCM.", ".CMM."): 400,

46. (".OOC", "COO.", "MOOOC", "COOOM"): 400,

47. (".MMCO", ".MCMO", ".CMMO", "OMMC.", "OMCM.", "OCMM.", "MOOC", "COOM"): 200,

48. (".MC.", ".CM."): 50,

49. ('.'): 20

50. }

51. return scoring

52. def find\_optimal\_move(move\_sequence, scoring\_system, position\_map):

53. chess\_board = '.' \* 15 \* 16

54. player\_sequence = determine\_player\_order(move\_sequence)

55. for i in range(0, len(move\_sequence), 2):

56. pos = calculate\_index(move\_sequence[i:i + 2])

57. marker = player\_sequence[0] if (i // 2) % 2 == 0 else player\_sequence[1]

58. chess\_board = chess\_board[:pos] + marker + chess\_board[pos + 1:]

59. best\_position = ''

60. highest\_score = 0

61. for idx, cell in enumerate(chess\_board):

62. if cell == '.':

63. temp\_board = chess\_board[:idx] + 'C' + chess\_board[idx + 1:]

64. current\_pos = position\_map[idx]

65. direction\_patterns = ','.join(extract\_patterns(current\_pos, temp\_board))

66. current\_score = 0

67. for pattern\_group, score in scoring\_system.items():

68. for pattern in pattern\_group:

69. current\_score += score \* direction\_patterns.count(pattern)

70. if current\_score > highest\_score:

71. highest\_score = current\_score

72. best\_position = current\_pos

73. print(f"{best\_position} {highest\_score}")

74. return best\_position

75. def submit\_answer(endpoint, solution):

76. params = {'ans': solution[:-1]}

77. response = req.get(endpoint, params=params)

78. print(response.text)

79. api\_url = "http://183.175.12.27:8004/step\_08/"

80. response = req.get(api\_url)

81. move\_sequences = response.json()['questions']

82. scoring\_rules = pattern\_scoring\_system()

83. position\_lookup = generate\_all\_positions()

84. solution\_string = ''

85. for sequence in move\_sequences:

86. solution\_string += find\_optimal\_move(sequence, scoring\_rules, position\_lookup) + ','

87. submit\_answer(api\_url, solution\_string)

88.

**步骤九：**

**（1）题目：**

访问服务器http://183.175.12.27:8001/join\_game，会返回一个游戏编号game\_id。之后你可以使用这个游戏编号，进行游戏http://183.175.12.27:8001/play\_game/{game\_id}并查询游戏状态http://183.175.12.27:8001/check\_game/{game\_id}。实现消息循环，开始作战吧！

**（2）解决思路：**

通过获取游戏编号和状态，模拟棋盘每一步，计算最佳下棋位置并提交答案，实现消息循环，完成游戏的自动化操作。

**（3）代码：**

1. import requests as re

2. def getIndexNum(coords):

3. return (ord(coords[0]) - ord('a')) \* 16 + ord(coords[1]) - ord('a')

4. def allIndexStr():

5. spot = []

6. for i in range(0, 15):

7. for j in range(0, 16):

8. spot.append(chr(i + 97) + chr(j + 97))

9. return spot

10. def getLine(coord, board):

11. line = ['', '', '', '']

12. i = 0

13. while (i != 9):

14. if ord(coord[1]) - ord('a') - 4 + i in range(0, 15):

15. line[0] += board[(ord(coord[0]) - ord('a')) \* 16 + ord(coord[1]) - ord('a') - 4 + i]

16. else:

17. line[0] += ' '

18. if ord(coord[0]) - ord('a') - 4 + i in range(0, 15):

19. line[2] += board[(ord(coord[0]) - ord('a') - 4 + i) \* 16 + ord(coord[1]) - ord('a')]

20. else:

21. line[2] += ' '

22. if ord(coord[1]) - ord('a') - 4 + i in range(0, 15) and ord(coord[0]) - ord('a') - 4 + i in range(0, 15):

23. line[1] += board[(ord(coord[0]) - ord('a') - 4 + i) \* 16 + ord(coord[1]) - ord('a') - 4 + i]

24. else:

25. line[1] += ' '

26. if ord(coord[1]) - ord('a') + 4 - i in range(0, 15) and ord(coord[0]) - ord('a') - 4 + i in range(0, 15):

27. line[3] += board[(ord(coord[0]) - ord('a') - 4 + i) \* 16 + ord(coord[1]) - ord('a') + 4 - i]

28. else:

29. line[3] += ' '

30.

31. i += 1

32. return line

33. def judge(testOrder):

34. if (len(testOrder) // 2) % 2 == 0:

35. return 'MO'

36. else:

37. return 'OM'

38. def RuleWithPoints():

39. RWP = {

40. ("CMMMM", "MCMMM", "MMCMM", "MMMCM", "MMMMC"): 10000,

41. ("COOOO", "OOOOC"): 6000,

42. (".CMMM.", ".MCMM.", ".MMCM.", ".MMMC."): 5000,

43. ("COOO.", ".OOOC", ".OOCO.", ".OCOO."): 2500,

44. ("OCMMM.", "OMCMM.", "OMMCM.", "OMMMC.", ".CMMMO", ".MCMMO", ".MMCMO", ".MMMCO"): 2000,

45. (".MMC.", ".MCM.", ".CMM."): 400,

46. (".OOC", "COO.", "MOOOC", "COOOM"): 400,

47. (".MMCO", ".MCMO", ".CMMO", "OMMC.", "OMCM.", "OCMM.", "MOOC", "COOM"): 200,

48. (".MC.", ".CM."): 50,

49. ('.'): 20

50. }

51. return RWP

52. def getMaxCoords(Order, RWP, indexSrc):

53. board = ''

54. for i in range(0, 15):

55. board += '...............' + '\n'

56. step = 0

57. BW = judge(Order)

58. for i in range(0, len(Order), 2):

59. index = getIndexNum(Order[i:i + 2])

60. if (step % 2) == 0:

61. board = board[0: index] + BW[0] + board[index + 1:]

62. else:

63. board = board[0: index] + BW[1] + board[index + 1:]

64. step += 1

65. maxCoord = ''

66. maxPoints = 0

67. for i in range(0, len(board)):

68. if board[i] == '.':

69. tempBoard = board[0: i] + 'C' + board[i + 1:]

70. coord = indexSrc[i]

71. lines4 = ','.join(getLine(coord, tempBoard))

72. points = 0

73. for rules, value in RWP.items():

74. for rul in range(0, len(rules)):

75. if rules[rul] in lines4:

76. points += value \* lines4.count(rules[rul])

77. if points > maxPoints:

78. maxPoints = points

79. maxCoord = coord

80. print(f"{maxCoord} {maxPoints}")

81. return maxCoord

82. def getNextStep(url, answer):

83. param = {

84. 'ans': answer[:-1]

85. }

86. getHtml = re.get(url, params=param)

87. print(getHtml.text)

88. url = "http://183.175.12.27:8004/step\_08/"

89. getHtml = re.get(url)

90. stepOrders = getHtml.json()['questions']

91. RWP = RuleWithPoints()

92. indexSrc = allIndexStr()

93. answer = ''

94. for order in stepOrders:

95. answer += getMaxCoords(order, RWP, indexSrc) + ','

96. getNextStep(url, answer)

**步骤十：**

**（1）题目：**

恭喜你到达第十步！

你已经完成了一个AI的设计，下面就需要发动你的智慧让你的机器大脑变得更聪明了！例如：

基于蒙特卡洛树搜索(AlphaZero思想)实现自我对弈的强化对抗训练，基于近端策略优化(Proximal Policy Optimization, PPO)实现与固定对手对战的强化对抗训练

**（2）解决思路：**

使用MCTS（蒙特卡洛树搜索）与神经网络（策略网络、价值网络）。

**（3）代码：**

1. import requests as re

2. import time as t

3. import random

4. import numpy as np

5. import torch

6. import torch.nn as nn

7. import torch.optim as optim

8. import torch.nn.functional as F

9. import os.path

10.

11. cached\_state = None

12.

13. MCTS\_SIMULATIONS = 50

14. MAX\_SKIPPED\_TURNS = 50

15. MODEL\_SAVE\_PATH = "gomoku\_model.pth"

16. SAVE\_INTERVAL = 2

17.

18. class AlphaZeroNet(nn.Module):

19. def \_\_init\_\_(self):

20. super(AlphaZeroNet, self).\_\_init\_\_()

21.

22. self.fc1 = nn.Linear(15 \* 15, 128) # 第一层

23. self.fc2 = nn.Linear(128, 64) # 第二层

24. self.fc3 = nn.Linear(64, 1) # 价值网络输出：局面的胜率

25. self.fc4 = nn.Linear(64, 15 \* 15) # 策略网络输出：每个位置的概率

26.

27. def forward(self, x):

28. x = x.view(-1, 15 \* 15) # 将棋盘展平为一维向量

29. x = F.relu(self.fc1(x)) # 第一层激活

30. x = F.relu(self.fc2(x)) # 第二层激活

31. value = torch.tanh(self.fc3(x)) # 价值网络输出

32. policy = F.softmax(self.fc4(x), dim=1) # 策略网络输出

33. return value, policy

34.

35.

36. model = AlphaZeroNet()

37.

38.

39. def fastModular(x):

40. result = 1

41. while (x[1] > 0):

42. if (x[1] & 1):

43. result = result \* x[0] % x[2]

44. x[1] = int(x[1] / 2)

45. x[0] = x[0] \* x[0] % x[2]

46. return result

47.

48.

49. def str\_to\_num(strings):

50. sum = 0

51. lens = len(strings)

52. for i in range(0, lens):

53. sum += ord(strings[i]) \* 256 \*\* (lens - i - 1)

54. return sum

55.

56.

57. def encodeLogin(password):

58. power = 65537

59. modulus = 135261828916791946705313569652794581721330948863485438876915508683244111694485850733278569559191167660149469895899348939039437830613284874764820878002628686548956779897196112828969255650312573935871059275664474562666268163936821302832645284397530568872432109324825205567091066297960733513602409443790146687029

60. return hex(fastModular([str\_to\_num(password), power, modulus]))

61.

62.

63. def join\_game(user, myHexPass):

64. url = 'http://183.175.12.27:8004/join\_game/'

65. param = {

66. 'user': user,

67. 'password': myHexPass,

68. 'data\_type': 'json'

69. }

70. getHtml = re.get(url, params=param)

71. print(f"Open a new game{getHtml.text}")

72. return getHtml

73.

74.

75. def check\_game(game\_id):

76. url = f'http://183.175.12.27:8004/check\_game/{game\_id}'

77. getState = re.get(url)

78. game\_state = getState.json()

79. board = game\_state.get('board', [])

80. if not board:

81. print("The board is empty at the start of the game.")

82. return game\_state

83.

84.

85. def play\_game(user, myHexPass, game\_id, coord):

86. url = f'http://183.175.12.27:8004/play\_game/{game\_id}'

87. param = {

88. 'user': user,

89. 'password': myHexPass,

90. 'data\_type': 'json',

91. 'coord': coord

92. }

93. re.get(url, params=param)

94.

95. def getIndexNum(coords):

96. if isinstance(coords, int):

97. coords = str(coords)

98. if len(coords) == 2 and isinstance(coords, str):

99. return (ord(coords[0]) - ord('a')) \* 16 + ord(coords[1]) - ord('a')

100. else:

101. raise ValueError(f"Invalid coordinates format: {coords}")

102.

103. def getMaxCoords(state, RWP, indexSrc):

104. if isinstance(state, list) and len(state) == 225:

105. board = ''

106. for i in range(15):

107. for j in range(15):

108. idx = i \* 15 + j

109. if state[idx] == 1:

110. board += 'M'

111. elif state[idx] == -1:

112. board += 'O'

113. else:

114. board += '.'

115. board += '\n'

116. else:

117. Order = state

118. board = ''

119. for i in range(0, 15):

120. board += '...............' + '\n'

121.

122. step = 0

123. BW = judge(Order)

124. for i in range(0, len(Order), 2):

125. coords = Order[i:i + 2]

126. index = getIndexNum(coords)

127. if (step % 2) == 0:

128. board = board[0: index] + BW[0] + board[index + 1:]

129. else:

130. board = board[0: index] + BW[1] + board[index + 1:]

131. step += 1

132. maxCoord = ''

133. maxPoints = 0

134. for i in range(0, len(board)):

135. if board[i] == '.':

136. tempBoard = board[0: i] + 'C' + board[i + 1:]

137. coord = indexSrc[i]

138. lines4 = ','.join(getLine(coord, tempBoard))

139. points = 0

140. for rules, value in RWP.items():

141. for rul in range(0, len(rules)):

142. if rules[rul] in lines4:

143. points += value \* lines4.count(rules[rul])

144. if points > maxPoints:

145. maxPoints = points

146. maxCoord = coord

147. return maxCoord if maxCoord else None

148.

149. def allIndexStr():

150. spot = []

151. for i in range(0, 15):

152. for j in range(0, 16):

153. spot.append(chr(i + 97) + chr(j + 97))

154. return spot

155.

156. def getLine(coord, board):

157. line = ['', '', '', '']

158. i = 0

159. while (i != 15):

160. if ord(coord[1]) - ord('a') - 7 + i in range(0, 15):

161. line[0] += board[(ord(coord[0]) - ord('a')) \* 16 + ord(coord[1]) - ord('a') - 7 + i]

162. else:

163. line[0] += ' '

164. if ord(coord[0]) - ord('a') - 7 + i in range(0, 15):

165. line[2] += board[(ord(coord[0]) - ord('a') - 7 + i) \* 16 + ord(coord[1]) - ord('a')]

166. else:

167. line[2] += ' '

168. if ord(coord[1]) - ord('a') - 7 + i in range(0, 15) and ord(coord[0]) - ord('a') - 7 + i in range(0, 15):

169. line[1] += board[(ord(coord[0]) - ord('a') - 7 + i) \* 16 + ord(coord[1]) - ord('a') - 7 + i]

170. else:

171. line[1] += ' '

172. if ord(coord[1]) - ord('a') + 7 - i in range(0, 15) and ord(coord[0]) - ord('a') - 7 + i in range(0, 15):

173. line[3] += board[(ord(coord[0]) - ord('a') - 7 + i) \* 16 + ord(coord[1]) - ord('a') + 7 - i]

174. else:

175. line[3] += ' '

176. i += 1

177. return line

178.

179. def judge(testOrder):

180. if (len(testOrder) // 2) % 2 == 0:

181. return 'MO'

182. else:

183. return 'OM'

184.

185. def RuleWithPoints():

186. RWP = {

187. ("CMMMM", "MCMMM", "MMCMM", "MMMCM", "MMMMC"): 10000,

188. ("COOOO", "OCOOO", "OOCOO", "OOOCO", "OOOOC"): 6000,

189. (".CMMM.", ".MCMM.", ".MMCM.", ".MMMC."): 5000,

190. ("COOO.", ".OOOC", ".OOCO.", ".OCOO."): 2500,

191. ("OCMMM.", "OMCMM.", "OMMCM.", "OMMMC.", ".CMMMO", ".MCMMO", ".MMCMO", ".MMMCO"): 2000,

192. (".MMC.", ".MCM.", ".CMM."): 400,

193. (".OOC", "COO.", "MOOOC", "COOOM"): 400,

194. (".MMCO", ".MCMO", ".CMMO", "OMMC.", "OMCM.", "OCMM.", "MOOC", "COOM"): 200,

195. (".MC.", ".CM."): 50,

196. ('.'): 1

197. }

198. return RWP

199.

200. MCTS\_SIMULATIONS = 50

201.

202. def get\_state(board):

203. if isinstance(board, str):

204. state = [0] \* 225

205. if board:

206. BW = judge(board)

207.

208. for i in range(0, len(board), 2):

209. if i + 1 < len(board):

210. index = getIndexNum(board[i:i + 2])

211. row = index // 16

212. col = index % 16

213. if 0 <= row < 15 and 0 <= col < 15:

214. pos = row \* 15 + col

215. if pos < 225:

216. state[pos] = 1 if (i // 2) % 2 == 0 else -1

217. else:

218. state = []

219. if not board:

220. state = [0] \* 225

221. else:

222. flat\_board = []

223. for row in board:

224. if isinstance(row, list):

225. flat\_board.extend(row)

226. else:

227. flat\_board.append(row)

228. state = [1 if cell == 'X' or cell == 'M' else -1 if cell == 'O' else 0 for cell in flat\_board]

229. if len(state) < 225:

230. state.extend([0] \* (225 - len(state)))

231. elif len(state) > 225:

232. state = state[:225]

233. print(f"State length: {len(state)}")

234. return state

235.

236. def get\_possible\_actions(state):

237. if len(state) != 225:

238. print(f"Warning: State length is {len(state)}, expected 225")

239. return []

240. return [i for i, cell in enumerate(state) if cell == 0]

241.

242. def mcts\_search(state):

243. if len(state) != 225:

244. print(f"Warning in mcts\_search: State length is {len(state)}, expected 225")

245. state = [0] \* 225

246. possible\_actions = get\_possible\_actions(state)

247. if not possible\_actions:

248. print("No possible actions found")

249. return None, 0

250. best\_action = None

251. best\_value = -float('inf')

252. for \_ in range(MCTS\_SIMULATIONS):

253. action, value = evaluate\_state(state)

254. if action is not None and value > best\_value:

255. best\_value = value

256. best\_action = action

257. return best\_action, best\_value

258.

259. def evaluate\_state(state):

260. if len(state) != 225:

261. raise ValueError(f"Expected state of length 225, got {len(state)}")

262. possible\_actions = get\_possible\_actions(state)

263. if not possible\_actions:

264. return None, 0

265. board\_tensor = torch.tensor(state, dtype=torch.float32).unsqueeze(0)

266. with torch.no\_grad():

267. value, policy = model(board\_tensor)

268. action\_probs = policy[0].detach().numpy()

269. valid\_action\_probs = []

270. for action in possible\_actions:

271. if action < len(action\_probs):

272. valid\_action\_probs.append(action\_probs[action])

273. else:

274. valid\_action\_probs.append(0.0)

275. if not valid\_action\_probs:

276. return None, 0

277. best\_idx = np.argmax(valid\_action\_probs)

278. best\_action = possible\_actions[best\_idx]

279. return best\_action, value.item()

280.

281. criterion = nn.MSELoss()

282. optimizer = optim.Adam(model.parameters(), lr=0.001)

283.

284. def train\_step(state, action, reward):

285. model.train()

286. state\_tensor = torch.tensor(state, dtype=torch.float32).unsqueeze(0)

287. action\_tensor = torch.tensor([action], dtype=torch.long)

288. value, policy = model(state\_tensor)

289. value\_loss = criterion(value, torch.tensor([reward], dtype=torch.float32))

290. policy\_loss = -torch.log(policy[0, action\_tensor]) \* reward

291. loss = value\_loss + policy\_loss

292. optimizer.zero\_grad()

293. loss.backward()

294. optimizer.step()

295. return loss.item()

296.

297. def choose\_action(state, RWP, indexSrc):

298. action\_coord = getMaxCoords(state, RWP, indexSrc)

299. if action\_coord:

300. try:

301. row = ord(action\_coord[0]) - ord('a')

302. col = ord(action\_coord[1]) - ord('a')

303. action = row \* 15 + col

304. print(f"Chosen action from heuristic strategy: {action\_coord} (index: {action})")

305. except (IndexError, TypeError):

306. print(f"Invalid action coordinate: {action\_coord}")

307. action = None

308. else:

309. print("Heuristic strategy did not find a valid action. Falling back to neural network.")

310.

311. action, \_ = mcts\_search(state)

312. if action is None:

313. print("Exploring with random action due to no valid action found.")

314. possible\_actions = get\_possible\_actions(state)

315. if possible\_actions:

316. action = random.choice(possible\_actions)

317. else:

318. print("No possible actions available.")

319. return None

320. return action

321.

322. epsilon = 0.1

323.

324. def get\_reward(state, action, winner):

325. if winner == 'None':

326. return 0

327. if winner == user:

328. return 1

329. else:

330. return -1

331.

332. def save\_model(model, optimizer, path=MODEL\_SAVE\_PATH):

333. torch.save({

334. 'model\_state\_dict': model.state\_dict(),

335. 'optimizer\_state\_dict': optimizer.state\_dict(),

336. }, path)

337. print(f"Model saved to {path}")

338.

339. def load\_model(model, optimizer, path=MODEL\_SAVE\_PATH):

340. if os.path.exists(path):

341. try:

342. checkpoint = torch.load(path)

343. model.load\_state\_dict(checkpoint['model\_state\_dict'])

344. optimizer.load\_state\_dict(checkpoint['optimizer\_state\_dict'])

345. print(f"Model loaded from {path}")

346. return True

347. except Exception as e:

348. print(f"Error loading model: {e}")

349. return False

350. else:

351. print(f"No saved model found at {path}")

352. return False

353.

354. def train\_agent(user, game\_id):

355. game\_state = check\_game(game\_id)

356. board = game\_state.get('board', '')

357. state\_str = get\_state(board)

358. skipped\_turns = 0

359. training\_steps = 0

360. while True:

361. action = choose\_action(state\_str, RWP, indexSrc)

362. if action is None:

363. print("Skipping turn due to no valid actions")

364. skipped\_turns += 1

365. if skipped\_turns > MAX\_SKIPPED\_TURNS:

366. print("Too many skipped turns, ending training...")

367. break

368. t.sleep(2)

369. game\_state = check\_game(game\_id)

370. board = game\_state.get('board', '')

371. state\_str = get\_state(board)

372. continue

373. row = action // 15

374. col = action % 15

375. coord = chr(97 + row) + chr(97 + col)

376. print(f"Playing move at position {coord} (action={action})")

377. play\_game(user, myHexPass, game\_id, coord)

378. t.sleep(1)

379. new\_game\_state = check\_game(game\_id)

380. new\_board = new\_game\_state.get('board', '')

381. new\_state\_str = get\_state(new\_board)

382. winner = new\_game\_state.get('winner', 'None')

383. reward = get\_reward(state\_str, action, winner)

384. loss = train\_step(state\_str, action, reward)

385. training\_steps += 1

386. print(f"Training step {training\_steps} - Loss: {loss}")

387. if training\_steps % SAVE\_INTERVAL == 0:

388. save\_model(model, optimizer)

389. if winner != 'None':

390. print(f"The winner is {winner}")

391. save\_model(model, optimizer)

392. break

393. state\_str = new\_state\_str

394. skipped\_turns = 0

395. if new\_game\_state.get('current\_turn', '') != user:

396. print(f"Waiting for opponent to play...")

397. waiting = True

398. while waiting:

399. t.sleep(2)

400. check\_state = check\_game(game\_id)

401. if check\_state.get('current\_turn', '') == user or check\_state.get('winner', 'None') != 'None':

402. waiting = False

403. board = check\_state.get('board', '')

404. state\_str = get\_state(board)

405.

406. user = 'name'

407. password = '12345'

408. myHexPass = encodeLogin(password)

409. RWP = RuleWithPoints()

410. indexSrc = allIndexStr()

411.

412. load\_model(model, optimizer)

413.

414. game\_id = join\_game(user, myHexPass).json()["game\_id"]

415. state = check\_game(game\_id)

416.

417. print("Looking for game partners ...")

418. while state['ready'] == "False":

419. state = check\_game(game\_id)

420. print(state['ready'], end=" ")

421. t.sleep(2)

422.

423. if state['creator'] != user:

424. opponent = state['creator']

425. else:

426. opponent = state['opponent\_name']

427.

428. train\_agent(user, game\_id)

429.

430. while state['ready'] == "True":

431. if state['current\_turn'] == user:

432. order = state['board']

433. coord = getMaxCoords(order, RWP, indexSrc)

434. play\_game(user, myHexPass, game\_id, coord)

435. print(f"Playing {coord}")

436. else:

437. print(f"Waiting for {opponent} to play")

438.

439. t.sleep(2)

440. state = check\_game(game\_id)

441.

442. if state['winner'] != "None":

443. print(f"The winner is {state['winner']}")

444. break

445.