基于 α-β 剪枝的五子棋人机对弈设计与优化

2021.05.12

摘要

本实验根据 Minmax 对抗搜索算法与 α-β 剪枝算法，利用 python 语言设计五子棋人工智能人机博弈游戏。本实验基于 pygame 交互界面，在线对战界面设置有悔棋，重开，认输等功能。智能体状态空间搜索深度为四层，包括两层智能体执子局面与两层人类执子局面，局面评估函数根据五子棋棋谱中的基本棋形设计。在原对抗搜索函数基础上加入了搜索面积，搜索顺序和迭代深化的优化项，有效缩小了搜索空间，提高了搜索速度。实验最后我们将不同训练程度的 Alphago Zero 与本实验进行博弈，并得出算法对比表。

关键词：五子棋，人工智能，Minmax 算法，α-β 剪枝，对抗搜索

目录

1 背景介绍

人机博弈是人工智能的重要分支，人们在对人机博弈搜索算法研究中产生了大量的研究成果。其中，Minmax 算法是其中最为基础的算法，该算法是一种零和算法，即一方要在可选的选项中选择将其优势最大化的选择，而另一方则选择令对手优势最小化的方法。而改进 Minmax 算法所产生的 α-β 剪枝算法则是其中最重要的算法之一，该方法可以有效地减少在搜索过程中生成的结点数从而提高搜索效率。α-β 剪枝算法的效率与平均分枝因子以及着法排列顺序息息相关，因此，根据游戏规则对算法搜索的平均分枝因子和着法排列顺序 [1] 进行改进成为提高 α-β 剪枝算法效率的关键。

2 Minmax 算法

对抗搜索（Adversarial Search）也称为博弈搜索（Game Rearch）在一个竞争的环境中，智能体（agents）之间通过竞争实现相反的利益，一方最大化这个利益，另一方最小化。对抗搜索的方法主要有三个：最小最大搜索（Minmax Search）Alpha-Beta 剪枝搜索（Pruning Search）蒙塔卡洛树搜索（Monte-Caelo Tree Search）最大最小搜索为对抗搜索中最基本的搜索方法；剪枝搜索是一种对最大最小搜索进行改进的算法，即在搜索过程中可以减去无需搜索的分支节点，且不影响搜索结果。蒙特卡洛树搜索通过采样而非穷举方法来实现搜索。

Minmax 搜索这里通过一个例子说明 Minmax Search 的计算过程。假设根据当前局面我们得到一个图 1 所示的博弈树：

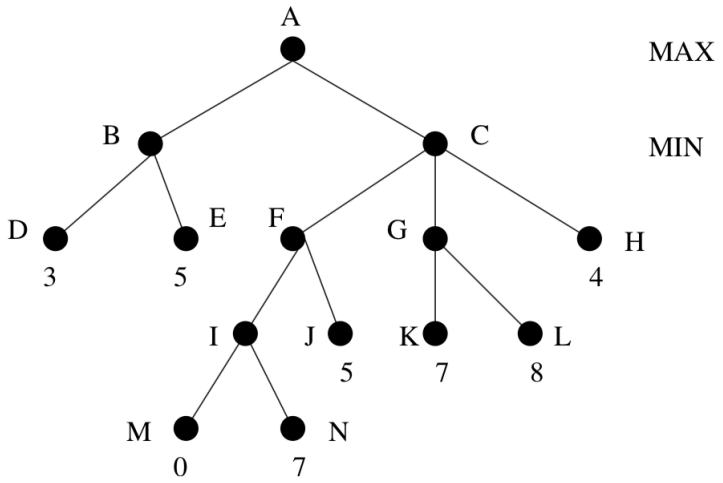


图 1: 博弈树

从上往下，单数层是我方行动，双数层是对方行动，我方行动需要选择对我最有利的行动，即 value 大的行动，对方行动则是选择使我方最不利的行动，即 value 小的行

动。

我们需要从最底层第四层开始考虑，双数层所以是对方行动。对于 node I，会选择值更小的 node M，node I 的值更新为 0。再考虑第三层，单数层是我方行动。node F 会选择 I，J 中值更大的 J，更新为 5，G 会选择 K，L 中值更大的 L，更新为 8。依次一层层向上类推，可以得到最终结果为图 2：

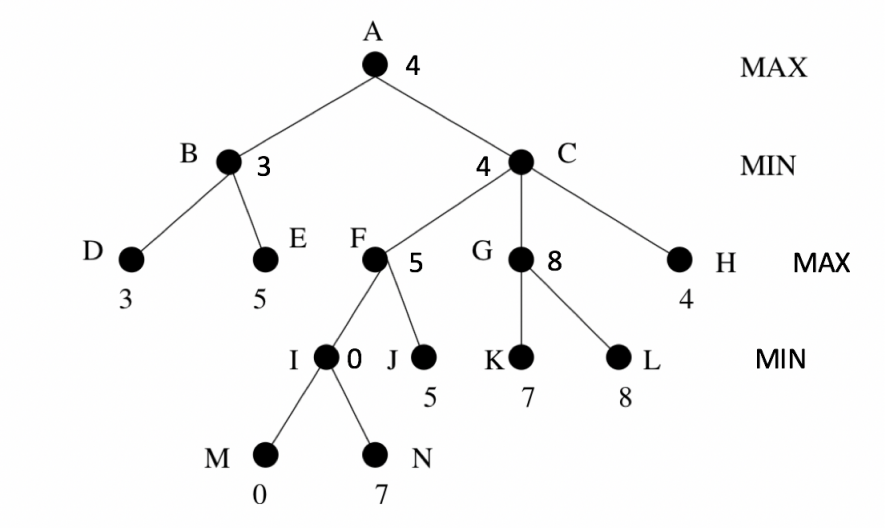


图 2: *Minmax* 搜索

通过以上例子可以得出对 Minmax 算法的性质与评价：

表 1: *Minmax* 算法性质

性质	评价	说明
完备性（complete）	是	决策树有限
最优性（optimal）	是	对手是理性的
空间复杂度（space）	O(b*m)[1]	深度优先探索
时间复杂度（time）	$O(b^m)$	
优点	简单有效	可返回最优结果
缺点	搜索空间过大	
优化	剪枝减少搜索节点	α-β 剪枝
优化	节点采样	MCTS 搜索

[1] m-树最大深度，b-平均分枝数

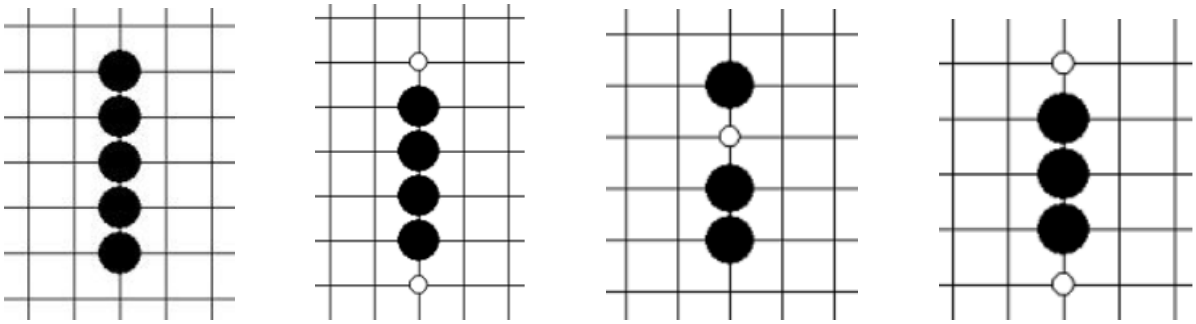
3 局面评估函数

五子棋最常见的基本棋型大体有以下几种：连五，活四，冲四，活三，眠三，活二，眠二。

连五顾名思义，五颗同色棋子连在一起。胜负已分，评分 10000。

活四有两个连五点 (即有两个点可以形成五)，活四出现的时候，如果对方单纯过来防守的话，是已经无法阻止自己连五了，评分 1000。

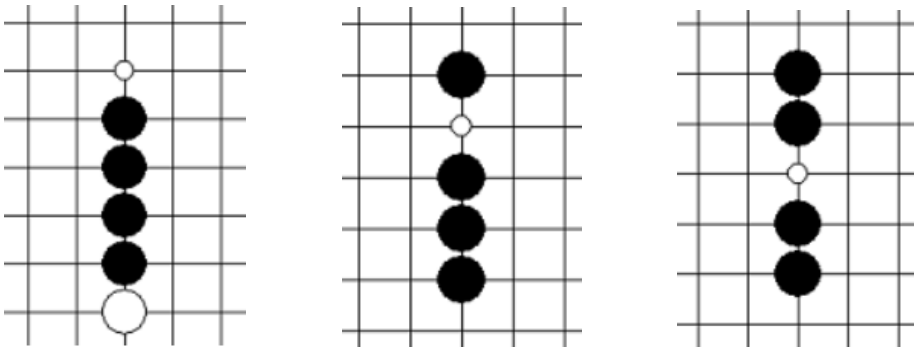
活三可以形成活四的三，当我们面对活三的时候，需要非常谨慎对待。在自己没有更好的进攻手段的情况下，需要对其进行防守，以防止其形成可怕的活四棋型。中间跳着一格的活三，也可以叫做跳活三，评分 100。



(a) 连五　(b) 活四　(c) 冲三　(d) 冲三

图 3

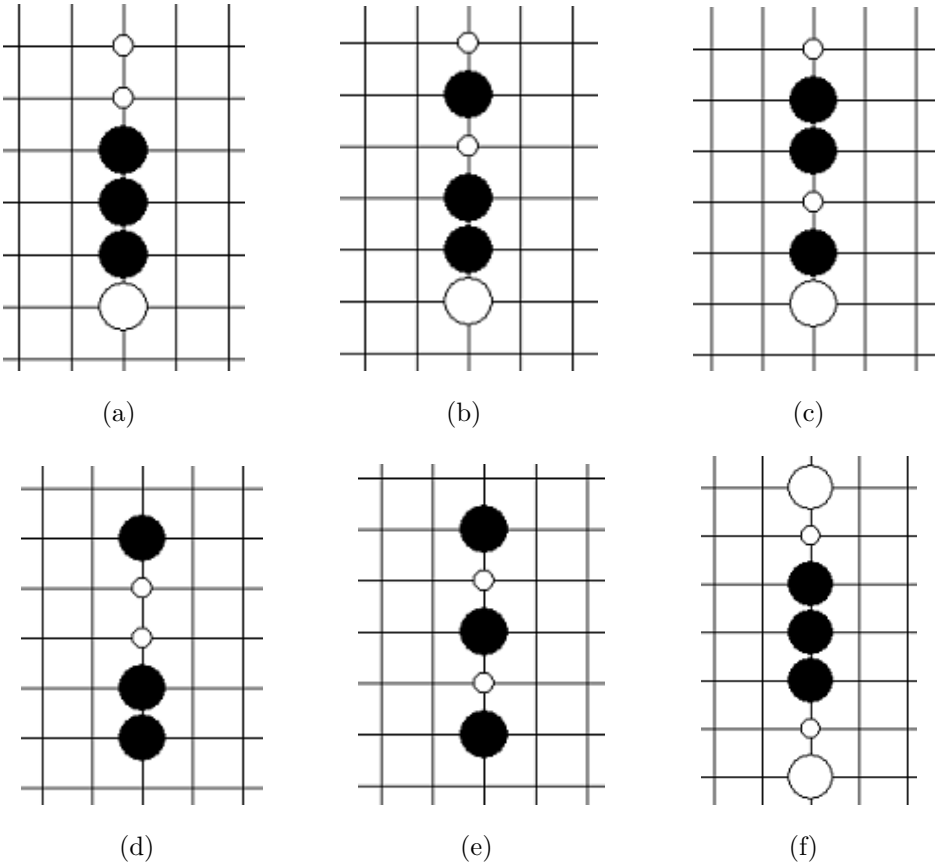
冲四有一个连五点，相对比活四来说，冲四的威胁性就小了很多，因为这个时候，对方只要跟着防守在那个唯一的连五点上，冲四就没法形成连五，评分 100。



(a)　(b)　(c)

图 4: 冲四

眠三只能够形成冲四的三，如下各图，分别代表最基础的六种眠三形状。眠三棋型即使不去防守，下一手它也只能形成冲四，而对于单纯的冲四棋型，我们知道，是可以防守住的，评分 100。

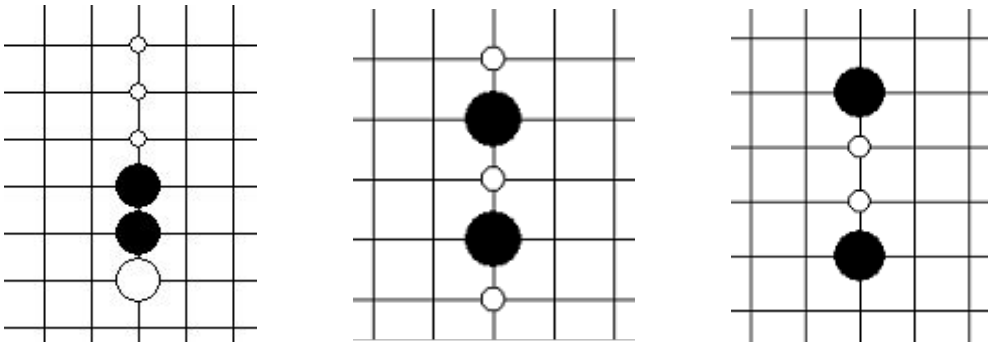


(a) (b) (c)

(d) (e) (f)

图 5: 眠三

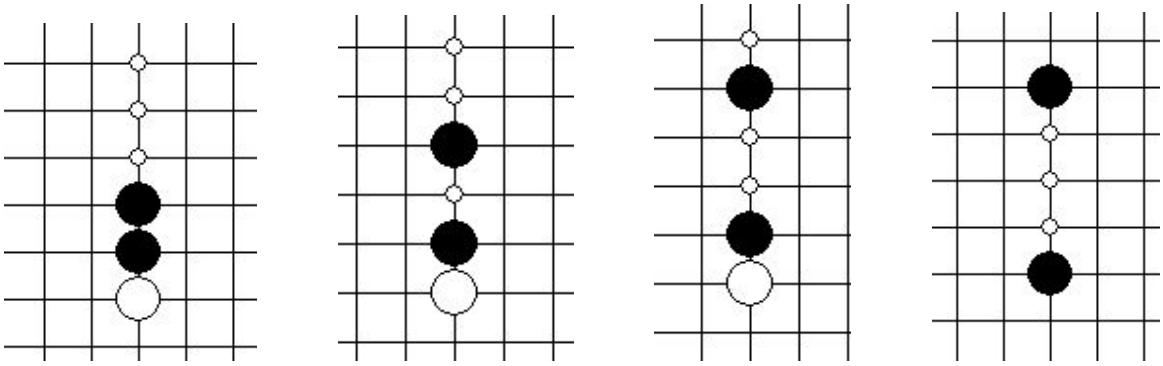
活二能够形成活三的二，下一手棋才能形成活三，等形成活三，我们再防守也不迟。但其实活二棋型是非常重要的，尤其是在开局阶段，我们形成较多的活二棋型的话，当我们将活二变成活三时，才能够令自己的活三绵绵不绝微风里，让对手防不胜防，评分 100。



(a) (b) (c)

图 6: 活二

眠二能够形成眠三的二。图中四个为最基本的眠二棋型，评分 100。



(a) (b) (c) (d)

图 7: 眠二

4 α-β 剪枝

α-β 剪枝的基本思想是对于每个 max 结点设置一个目前已知下界 α，每个 min 节点设置一个目前已知上界 β。α 代表我方可以搜索到的最好值，β 代表了对方可以接受的最坏值。如果某个行动的结果小于或等于 α，那么它就是很差的行动，我方可以选择更好的行动（当前 α 值的行动）。反之，如果某个行动的结果大于或等于 β，那么整个节点就作废了，因为对手不希望走到这个局面，而它有一定有别的行动（即走当前 β 值的行动）可以避免到达这个局面。

但发生下面两种情况时可以剪枝，即停止搜索该节点的其余子节点：1）当计算一个 min 结点时，如果它的 β 值小于等于其父结点的 α 值，则可以立即停止此结点的计算（α 剪枝）。2）当计算一个 max 结点时，如果它的 α 结点大于等于其父结点的 β 值，也可以立即停止此结点的计算（β 剪枝）。

简单来说，当 $\alpha \geq \beta$ 时，发生剪枝，因为后续搜索到的行动一定会差于之前的行动或是对方一定不会采取后续的行动。通过剪枝可以减少遍历的节点数，从而加快速度。即对于当前局面，我的选择最好可以达到价值 =4。

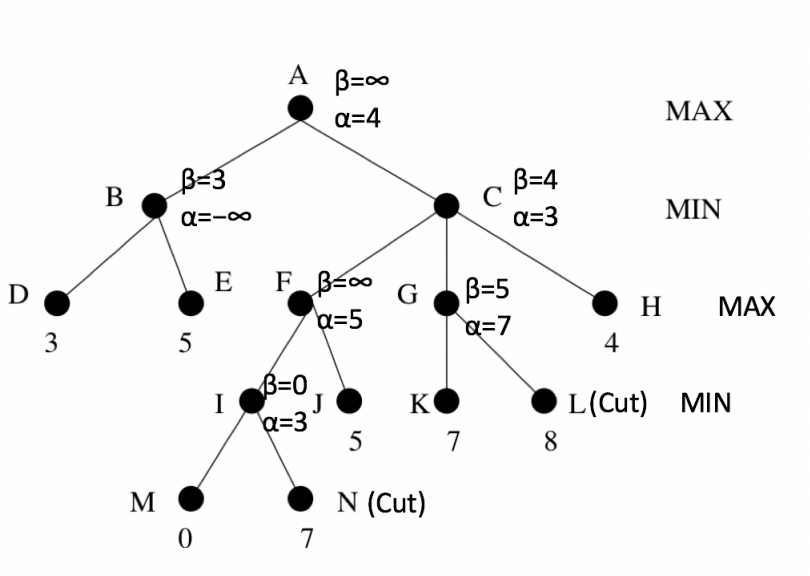


图 8: α-β 剪枝

5 剪枝算法优化

依据五子棋的特点，分别提出了 3 种改进方法 [2]。局部搜索，对棋盘进行局部化搜索，减少平均分枝因子；静态评价启发和迭代深化方法，优化着法排列顺序。下文将详细阐述。

5.1 局部搜索

在五子棋博弈的搜索中，所有空白位置都是合法的着法。对于 10×10 的棋盘，这意味着棋局刚开始时，博弈搜索的第 1 层的分枝因子为 10×10—1=99，第 2 层为 98，如果用 d 表示搜索深度，那么 d 层的分枝因子为 100-d。依据五子棋的特点，通过仔细分析可以看出，在这些大量的着法中，其实很多着法都是没有必要进行搜索的。因为在五子棋中，博弈的目的是阻止对方形成五连并尽量使己方形成五连，在实践中，发现最优走法应该是围绕在棋盘上已经形成的走法周围，所以没有必要对整个棋盘中的空白位置都进行搜索，而可以将棋盘进行剪切，形成一个局部棋盘，对局部棋盘上的空白位置进行搜索，这样将大大减少平均分枝因子。如果把棋盘看成是一个以左下角为坐标原点，两条边线为坐标轴的坐标系，对于每一个要进行搜索的节点，设其为 P，P 中具有最小纵坐标的棋子的坐标为 miny，P 中具有最大纵坐标的棋子的坐标为 maxy，P 中具有最小横坐标的棋子的坐标为 minx，P 中具有最大横坐标的棋子的坐标为 maxx，。可以为

户产生一个 m x n 的局部棋盘，其中，m=max-min+3，行 =max-min+3，这个棋盘包含 P 中所有的棋子，也就是说产生了一个等同于 P，但是却减少了很多无用分枝着法的棋盘。

5.2 静态评价启发

由 α-β 搜索算法原理可知，越早搜索到较优着法，那么剪枝就将越早得发生，α-β 搜索算法的效率也就越高。静态评价启发是本文提出的一种用来优化着法顺序的启发方法，它使得较优的走法能优先被搜索，因此可以简单而有效地提高 α-β 搜索算法的效率。在五子棋博弈中，当前节点最佳的着法可能不是在多层搜索的基础上的最佳的着法，但是它往往是一个较优的着法。例如，当前节点能产生一个形成活四的着法，那么这个着法将是一个最优的着法，不管还要进行多少层的搜索。因此，对于每一个要进行搜索的节点，设为 P，其每一个着法为 m_i，每一个着法 m_i。形成局面 P_i，那么可以对 P_i，进行评估，产生其评估值 v_i，如果 P 是极大方，则以 v_i，为关键字对 m_i。进行非递增排序；如果 P 是极小方，则以砖为关键字对 m_i，进行非递减排序。最终把这个排序的着法序列作为 P 的着法搜索顺序。

5.3 迭代深化

在搜索的过程中，针对第 1 个要搜索的节点，为了得到一个较好的着法作为第 1 个着法，可以采用迭代深化的方法。当对一个节点进行深度为 d 的搜索时，可以首先对其进行一次 d-1 层搜索，得出的最佳着法作为 d 层搜索的最先搜索的着法。由于两层相邻之间的节点比较相似，因此这一着法很有可能便是最佳的着法或者是较优的着法。由此，在搜索过程中，将得到一个较高的剪枝效率 p，。在进行 d 层搜索时首先进行咖 1 层的搜索看似多进行了一次搜索，花费了更多的时间，但实际上搜索将变得更加有效。[3] 的实验表明，α-β 剪枝搜索 d 层所需时间大约是扣 1 层所需时间的 b 倍，其中，b 为平均分枝因子。五子棋中平均分枝因子大约为取 b=200，因此，每多搜一层就会花上原先的 200 倍时间。所以，对出 l 层的搜索大约只有进行 d 层搜索的 1 / 200，这个代价并不大，但却对 d 层的搜索提供了一个较优的着法的启发，这使剪枝效率将大大提高。

6 实验设计

6.1 实验环境

表 2: 实验环境

环境	版本
Windows 操作系统	10.0.19042
pycharm	2020.2.2 x64
python	3.8.3
pygame	2.0.1

6.2 算法流程图

图 9: 算法流程图

6.3 获取棋局状态

设当前棋局状态为 π，程序从当前状态扩展两层得到节点状态集合 $\Pi = \{\pi_i | i = 0, 1, 2, ...\}$，根据局面评估函数分别确定节点状态 π_i 的价值。判断状态价值的关键在于分别获取黑白棋子行、列和对角线上的同色棋子分布。getlist 函数返回棋子行，列，左右对角线同色棋子的四个列表，四个列表分别求解评估函数得到该状态的价值。
函数源代码如下：

Listing 1: 获取行列对角线上的同色棋子分布

```
#这里以水平线为例
def get_list(mx, my, color):
    global matrix
    # list_h:水平
    # 向右
    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
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty > SIZE:
        list1.append(-1)
    else:
        list1.append(0)

    # 删除拼接交叉点
    list1.pop(0)

    # 向左
    list2 = []
    tx = mx
    ty = my
    while matrix[tx][ty] == color:
        list2.insert(0, 1)
        tx = tx - 1
    if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty > SIZE:
        list2.insert(0, -1)
```

```
    else:
        list2.insert(0, 0)
    list_h = list2 + list1
    ...
    return [list_h, list_v, list_l, list_r]
```

6.4 α-β 剪枝函数

发生下面两种情况时可以剪枝，即停止搜索该节点的其余子节点：

1. 当计算一个 min 结点时，如果它的 β 值小于等于其父结点的 α 值，则可以立即停止此结点的计算（α 剪枝）。

2. 当计算一个 max 结点时，如果它的 α 结点大于等于其父结点的 β 值，也可以立即停止此结点的计算（β 剪枝）。

函数源代码如下：

Listing 2: α-β 剪枝函数

```
def round_ai():
    global min_x, max_x, min_y, max_y, flag_color, matrix
    time_s = time.time()

    if step != 0:  # step=0 步骤为玩家(对手)行棋
            min_tx1, min_ty1, max_tx1, max_ty1 = legal_range(min_x, min_y,
               max_x, max_y)
            eva_matrix1 = np.zeros((SIZE + 2, SIZE + 2), dtype=
               int)  # 第一层的估值矩阵
            value_max = -INF
            rx, ry = 0, 0

            for i in  range(min_tx1, max_tx1 + 1):
                for j in  range(min_ty1, max_ty1 + 1):
                    # 是否剪枝
                    flag_cut = False
                    eva_matrix2 = np.zeros((SIZE + 2, SIZE + 2), dtype=
                       int)

                    if matrix[i][j] == 0:
                        matrix[i][j] = flag_color
                        min_tx2, min_ty2, max_tx2, max_ty2 = legal_range(
                           min_tx1, min_ty1, max_tx1, max_ty1)
```

```
                    [list_h, list_v, list_l, list_r] = get_list(i, j,
                        flag_color)
                    eva1 = evaluation(list_h, list_v, list_l, list_r)

                    for _i in  range(min_tx2, max_tx2 + 1):
                        for _j in  range(min_ty2, max_ty2 + 1):

                            if matrix[_i][_j] == 0:
                                matrix[_i][_j] = -flag_color
                                [list_h, list_v, list_s, list_b] =
                                    get_list(_i, _j, -flag_color)
                                eva2 = -evaluation(list_h, list_v,
                                    list_s, list_b)

                                eva_matrix2[_i][_j] = eva2 + eva1
                                matrix[_i][_j] = 0
                                # 剪枝
                                if eva_matrix2[_i][_j] < value_max:
                                    eva_matrix1[i][j] = eva_matrix2[_i
                                        ][_j]
                                    flag_cut = 1
                                    break
                        if flag_cut:
                            break

                    if not flag_cut:
                        value_min = INF
                        for _i in  range(min_tx2, max_tx2 + 1):
                            for _j in  range(min_ty2, max_ty2 + 1):
                                if eva_matrix2[_i][_j] < value_min and
                                    matrix[_i][_j] == 0:
                                    value_min = eva_matrix2[_i][_j]

                        eva_matrix1[i][j] = value_min

                        if value_max < value_min:
                            value_max = value_min
```

```
                                rx, ry = i, j

                        matrix[i][j] = 0

        time_e = time.time()
        print("Time cost:",  round(time_e - time_s, 4), "s")
        add_chess(rx, ry, flag_color)
```

6.5 优化函数

对于 15x15 的棋盘，最优走法应该是围绕在棋盘上已经形成的走法周围，所以没有必要对整个棋盘中的空白位置都进行搜索，而可以将棋盘进行剪切，形成一个局部棋盘，对局部棋盘上的空白位置进行搜索，这样将大大减少平均分枝因子。

如果把棋盘看成是一个以左下角为坐标原点，两条边线为坐标轴的坐标系，对于每一个要进行搜索的节点，设其为 P，P 中具有最小纵坐标的棋子的坐标为 miny，P 中具有最大纵坐标的棋子的坐标为 maxy，P 中具有最小横坐标的棋子的坐标为 minx，P 中具有最大横坐标的棋子的坐标为 maxx。可以产生一个局部棋盘，其中，mintx=minx-3，maxtx=maxx+3，minty=miny-3，maxty=maxy+3，这个棋盘包含 P 中所有的棋子，也就是说产生了一个等同于 P，但是却减少了很多无用分枝着法的棋盘。

函数源代码如下：

Listing 3: α-β 剪枝函数

```
            miny,maxy=0,14
            for i in  range(np.array(matrix).shape[0]):
                for j in  range(np.array(matrix).shape[1]):
                    if matrix[i][j]!=0:
                        miny=i
                        break
            for i in  range(np.array(matrix).shape[0]-1,-1,-1):
                for j in  range(np.array(matrix).shape[1]):
                    if matrix[i][j]!=0:
                        maxy=i
                        break
            minx,maxx=0,14
            for i in  range(np.array(matrix).shape[0]):
                for j in  range(np.array(matrix).shape[1]):
                    if matrix[i][j]!=0:
                        minx= max(j,minx)
```

```
                    break
            for i in  range(np.array(matrix).shape[0]):
                for j in  range(np.array(matrix).shape[1]-1,-1,-1):
                    if matrix[i][j]!=0:
                        maxx= min(j,maxx)
                        break

            min_tx1, min_ty1, max_tx1, max_ty1 = legal_range(min_x, min_y,
                max_x, max_y)
            min_tx1 =  max(minx - 3, min_tx1)
            min_ty1 =  max(miny - 3, min_ty1)
            max_tx1 =  min(maxx - 3, max_tx1)
            max_ty1 =  min(maxy - 3, max_ty1)
```

6.6 游戏界面

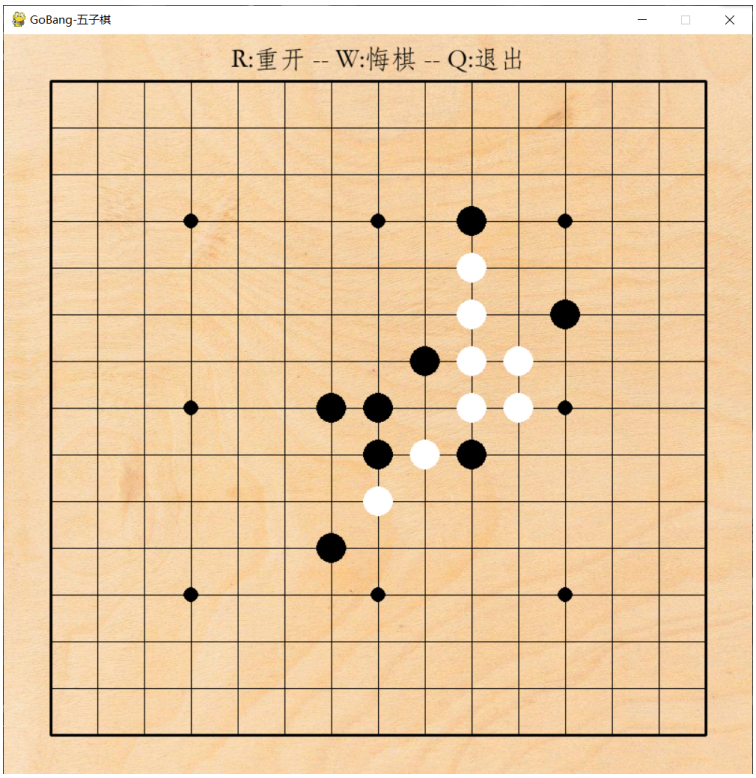


图 10: 游戏界面

7 实验评价

7.1 实验分工

表 3: 实验分工

实验内容	姓名
程序设计与交互界面	赵昊堃
剪枝算法改进与论文写作	苗成林
答辩 PPT 与资料查找	唐骏龙

7.2 方案扩展

为了提高本次实验五子棋的战力，我们在 Github 社区下载了 Alphago Zero[4] 五子棋源代码并进行训练，当训练次数为 8000 次时，我们的 α-β 剪枝状态空间搜索程序还可以战胜 Alphago Zero，但是当训练次数为 15000 次时，Alphago Zero 在对战 α-β 剪枝五子棋 AI 几乎无一败绩。这提示我们程序还有很大的待改进空间。

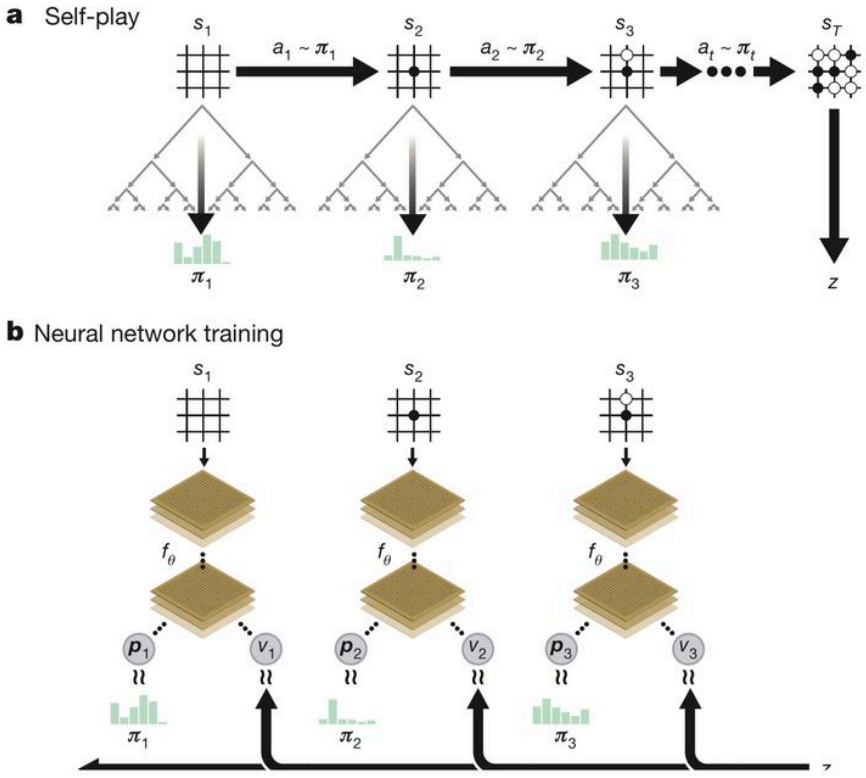


图 11: *Alphago Zero* 原理图

通过阅读原论文与相关博客我们得出两种算法的对比表，作为我们面对更大的状态空间时改进对抗搜索算法的一种可能。算法对比表 4：

表 4: 算法对比

智能体	对抗搜索五子棋	Alphago Zero
核心算法	α-β 剪枝对抗搜索	MCTS+ 策略价值网络
状态价值评估	预先设定评估函数	价值网络
状态空间	状态空间越大层数越少	更适合大状态空间
模拟方法	无模拟	策略网络代替 Rollout

参考文献

[1] 纪洪生. 基于概率的剪枝算法 [J]. 电脑知识与技术（学术交流）,2006(11):99-100.DOI:10.3969/j.issn.1009-3044.2006.11.055.

[2] 程宇, 雷小锋. 五子棋中 Alpha-Beta 搜索算法的研究与改进 [J]. 计算机工程,012,38(17):186-188.DOI:10.3969/j.issn.1000-3428.2012.17.051.

[3] Knuth D E，Moore R W-An Analysis of Alpha—Beta Pruning[J]．Artificial Intelligence，1975，6(4)：293—326.

[4] SILVER, DAVID, SCHRITTWIESER, JULIAN, SIMONYAN, KAREN, et al. Mastering the game of Go without human knowledge[J]. Nature,2017,550(Oct.19 TN.7676):354-359.

附录

Listing 4: 源代码完整版

```
import pygame
import os
import time
import sys
import tkinter.messagebox
import numpy as np
import random as rand

# 参数设置
WIDTH = 800
HEIGHT = 800
SIZE = 15  # 棋盘大小为15*15
SPACE = WIDTH // (SIZE + 1)  # 网格大小
FPS = 60  # 帧率
INF = 999999  # 定义无穷值
COLOR_WHITE = (255, 255, 255)
COLOR_BLACK = (0, 0, 0)

# init_pygame
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("GoBang-五子棋")
clock = pygame.time.Clock()
bg_img = pygame.image.load(os.path.join("images", "background.png"))
background = pygame.transform.scale(bg_img, (WIDTH, HEIGHT))
back_rect = background.get_rect()

# 全局变量
flag_win = 0  # 白子获胜:-1 黑子获胜:1
flag_color = 1  # ai执白子
flag_gg = True
flag_running = True
flag_start = True
# 步数记录
```

```
35 step = 0
36 # 棋盘矩阵
37 matrix = np.zeros((SIZE + 2, SIZE + 2), dtype= int)
38 # 搜索范围
39 min_x, min_y, = 0, 0
40 max_x, max_y = 0, 0
41 # 步骤记录
42 movements = []
43
44
45 # 绘制网格线
46 def draw_background(surf):
47     screen.blit(background, back_rect)
48
49     rect_lines = [((SPACE, SPACE), (SPACE, HEIGHT - SPACE)),
50                   ((SPACE, SPACE), (WIDTH - SPACE, SPACE)),
51                   ((SPACE, HEIGHT - SPACE), (WIDTH - SPACE, HEIGHT - SPACE)
                        ),
52                   ((WIDTH - SPACE, SPACE), (WIDTH - SPACE, HEIGHT - SPACE))
                        ]
53     # 边框线
54     for line in rect_lines:
55         pygame.draw.line(surf, COLOR_BLACK, line[0], line[1], 3)
56     # 网格线
57     for i in  range(17):
58         pygame.draw.line(surf, COLOR_BLACK, (SPACE * (2 + i), SPACE),
59                          (SPACE * (2 + i), HEIGHT - SPACE))
60         pygame.draw.line(surf, COLOR_BLACK,
61                          (SPACE, SPACE * (2 + i)),
62                          (HEIGHT - SPACE, SPACE * (2 + i)))
63     # 画棋盘上的黑色标记
64     dots = [(SPACE * 4, SPACE * 4),
65             (SPACE * 8, SPACE * 4),
66             (SPACE * 12, SPACE * 4),
67             (SPACE * 4, SPACE * 8),
68             (SPACE * 8, SPACE * 8),
69             (SPACE * 12, SPACE * 8),
```

```
70              (SPACE * 4, SPACE * 12),
71              (SPACE * 8, SPACE * 12),
72              (SPACE * 12, SPACE * 12)]
73
74      for dot in dots:
75          pygame.draw.circle(surf, COLOR_BLACK, dot, 8)
76
77  # 刷新棋盘已占有棋子的外切矩形范围
78  def update_range(x, y):
79      global min_x, min_y, max_x, max_y
80      if step == 0:
81          min_x, min_y, max_x, max_y = x, y, x, y
82      else:
83          if x < min_x:
84              min_x = x
85          elif x > max_x:
86              max_x = x
87          if y < min_y:
88              min_y = y
89          elif y > max_y:
90              max_y = y
91
92  # 棋型评估
93  model_score = {
94      # 一子:两端开
95      (0, 1, 0): 2,
96      # 死二
97      (-1,1,1, -1): -5,
98      # 死三
99      (-1,1,1, 1, -1): -5,
100     # 死四
101     (-1,1,1,1, 1, -1): -5,
102     # 二子:一端死
103     (0, 1, 1, -1): 5,
104     (-1, 1, 1, 0): 5,
105     # 二子:两端开
106     (0, 1, 1, 0): 20,
```

```
107     # 三子:一端死
108     (-1, 1, 1, 1, 0): 20,
109     (0, 1, 1, 1, -1): 20,
110     (-1, 1, 1, 1, 0,0): 20,
111     (0,0, 1, 1, 1, -1): 20,
112     (0, 1, 0,1, 1, -1): 20,
113     (-1, 1, 1, 0, 1, 0): 20,
114     (0, 1, 1,0, 1, -1): 20,
115     (-1, 1, 0, 1, 1,0): 20,
116     (1, 0, 0, 1, 1): 20,
117     (1, 1, 0, 0, 1): 20,
118     (1, 0, 1, 0 ,1): 20,
119     (-1, 0, 1, 1, 1, 0 ,-1): 20,
120     # 三子:两端开
121     (0, 1, 1, 1, 0): 40,
122     (0, 1, 0, 1, 1, 0): 40,
123     (0, 1, 1, 0, 1, 0): 40,
124     # 四子:一端死
125     (-1, 1, 1, 1, 1, 0): 80,
126     (0, 1, 1, 1, 1, -1): 80,
127     (0,1,0,1,1,1,0): 80,
128     (0,1,1,1,0,1,0): 80,
129     (0,1,1,0,1,1,0): 80,
130     # 四子:两端开
131     (0, 1, 1, 1, 1, 0): 160,
132     # 五子
133     (0, 1, 1, 1, 1, 1, 0): 320,
134     (0, 1, 1, 1, 1, 1, -1): 320,
135     (-1, 1, 1, 1, 1, 1, 0): 320,
136     (-1, 1, 1, 1, 1, 1, -1): 320
137 }
138
139
140 # 评估一个节点分值AI为正数 玩家为负数(调用的时候确定符号)
141 def evaluation(list_h, list_v, list_s, list_b):
142     score_h = model_score.get( tuple(list_h), 0)
143     score_v = model_score.get( tuple(list_v), 0)
```

```
144     score_s = model_score.get( tuple(list_s), 0)
145     score_b = model_score.get( tuple(list_b), 0)
146     rank = [score_h, score_v, score_s, score_b]
147     return  sum(rank)
148 
149 # 获得该结点在水平、竖直、左斜、右斜方向上构成的同色的棋子
150 def get_list(mx, my, color):
151     global matrix
152     # list_h:水平
153     # 向右
154     list1 = []
155     tx, ty = mx, my
156     while matrix[tx][ty] == color:
157         list1.append(1)  # 1表示是己方棋子，-1是敌方棋子
158         tx = tx + 1
159     if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
        SIZE:
160         list1.append(-1)
161     else:
162         list1.append(0)
163 
164     # 删除拼接交叉点
165     list1.pop(0)
166 
167     # 向左
168     list2 = []
169     tx = mx
170     ty = my
171     while matrix[tx][ty] == color:
172         list2.insert(0, 1)
173         tx = tx - 1
174     if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
        SIZE:
175         list2.insert(0, -1)
176     else:
177         list2.insert(0, 0)
178     list_h = list2 + list1
```

```
179
180     # list_v:垂直方向
181     list1 = []
182     tx = mx
183     ty = my
184     while matrix[tx][ty] == color:
185         list1.append(1)
186         ty = ty + 1
187     if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
            SIZE:
188         list1.append(-1)
189     else:
190         list1.append(0)
191     list1.pop(0)
192     list2 = []
193     tx = mx
194     ty = my
195     while matrix[tx][ty] == color:
196         list2.insert(0, 1)
197         ty = ty - 1
198     if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
            SIZE:
199         list2.insert(0, -1)
200     else:
201         list2.insert(0, 0)
202     list_v = list2 + list1
203
204     # list_l:向左斜
205     list1 = []
206     tx = mx
207     ty = my
208     while matrix[tx][ty] == color:
209         list1.append(1)
210         tx = tx - 1
211         ty = ty + 1
212     if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
            SIZE:
```

```
213         list1.append(-1)
214     else:
215         list1.append(0)
216     list1.pop(0)
217     list2 = []
218     tx = mx
219     ty = my
220     while matrix[tx][ty] == color:
221         list2.insert(0, 1)
222         tx = tx + 1
223         ty = ty - 1
224     if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
        SIZE:
225         list2.insert(0, -1)
226     else:
227         list2.insert(0, 0)
228     list_l = list2 + list1
229 
230     # list_r:向右斜
231     list1 = []
232     tx = mx
233     ty = my
234     while matrix[tx][ty] == color:
235         list1.append(1)
236         tx = tx + 1
237         ty = ty + 1
238     if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
        SIZE:
239         list1.append(-1)
240     else:
241         list1.append(0)
242     list1.pop(0)
243     list2 = []
244     tx = mx
245     ty = my
246     while matrix[tx][ty] == color:
247         list2.insert(0, 1)
```

```
248         tx = tx - 1
249         ty = ty - 1
250     if matrix[tx][ty] == -color or tx == 0 or ty == 0 or tx > SIZE or ty >
        SIZE:
251         list2.insert(0, -1)
252     else:
253         list2.insert(0, 0)
254     list_r = list2 + list1
255 
256     return [list_h, list_v, list_l, list_r]
257 
258 # 判断搜索范围是否超出边界，返回合法的搜索范围
259 def legal_range(min_x, min_y, max_x, max_y):
260     change = 1
261     if min_x - change < 1:
262         min_tx = 1
263     else:
264         min_tx = min_x - change
265 
266     if min_y - change < 1:
267         min_ty = 1
268     else:
269         min_ty = min_y - change
270 
271     if max_x + change > SIZE:
272         max_tx = SIZE
273     else:
274         max_tx = max_x + change
275 
276     if max_y + change > SIZE:
277         max_ty = SIZE
278     else:
279         max_ty = max_y + change
280     return [min_tx, min_ty, max_tx, max_ty]
281 
282 # alpha-beta剪枝搜索
283 def round_ai():
```

```
284     global min_x, max_x, min_y, max_y, flag_color, matrix
285     time_s = time.time()
286 
287     if step != 0:  # step=0 步骤为玩家(对手)行棋
288         if step == 1:
289             # 第一步抢占中心,若中心被占则随机选择周围四点
290             if matrix[(SIZE + 1) // 2][(SIZE + 1) // 2] == 0:
291                 rx, ry = (SIZE + 1) // 2, (SIZE + 1) // 2
292             else:
293                 case = rand.randint(1, 4)
294                 if case == 1:
295                     dx, dy = 1, 1
296                 elif case == 2:
297                     dx, dy = 1, -1
298                 elif case == 3:
299                     dx, dy = -1, 1
300                 else:
301                     dx, dy = -1, -1
302                 rx, ry = (SIZE + 1) // 2 + dx, (SIZE + 1) // 2 + dy
303         else:
304 
305             min_tx1, min_ty1, max_tx1, max_ty1 = legal_range(min_x, min_y,
                    max_x, max_y)
306 
307             eva_matrix1 = np.zeros((SIZE + 2, SIZE + 2), dtype=
                    int)  # 第一层的估值矩阵
308             value_max = -INF
309             rx, ry = 0, 0
310 
311             for i in  range(min_tx1, max_tx1 + 1):
312                 for j in  range(min_ty1, max_ty1 + 1):
313                     # 是否剪枝
314                     flag_cut = False
315                     eva_matrix2 = np.zeros((SIZE + 2, SIZE + 2), dtype=
                            int)
316 
317                     if matrix[i][j] == 0:
318                         matrix[i][j] = flag_color
```

```
319                     min_tx2, min_ty2, max_tx2, max_ty2 = legal_range(
                            min_tx1, min_ty1, max_tx1, max_ty1)
320                     [list_h, list_v, list_l, list_r] = get_list(i, j,
                            flag_color)
321                     eva1 = evaluation(list_h, list_v, list_l, list_r)
322
323                     for _i in  range(min_tx2, max_tx2 + 1):
324                         for _j in  range(min_ty2, max_ty2 + 1):
325
326                             if matrix[_i][_j] == 0:
327                                 matrix[_i][_j] = -flag_color
328                                 [list_h, list_v, list_s, list_b] =
                                        get_list(_i, _j, -flag_color)
329                                 eva2 = -evaluation(list_h, list_v,
                                        list_s, list_b)
330
331                                 eva_matrix2[_i][_j] = eva2 + eva1
332                                 matrix[_i][_j] = 0
333                                 # 剪枝
334                                 if eva_matrix2[_i][_j] < value_max:
335                                     eva_matrix1[i][j] = eva_matrix2[_i
                                            ][_j]
336                                     flag_cut = 1
337                                     break
338                         if flag_cut:
339                             break
340
341                     if not flag_cut:
342                         value_min = INF
343                         for _i in  range(min_tx2, max_tx2 + 1):
344                             for _j in  range(min_ty2, max_ty2 + 1):
345                                 if eva_matrix2[_i][_j] < value_min and
                                        matrix[_i][_j] == 0:
346                                     value_min = eva_matrix2[_i][_j]
347
348                         eva_matrix1[i][j] = value_min
349
```

```
                            if value_max < value_min:
                                value_max = value_min
                                rx, ry = i, j

                        matrix[i][j] = 0

        time_e = time.time()
        print("Time cost:",  round(time_e - time_s, 4), "s")
        add_chess(rx, ry, flag_color)


def round_ai2():
    global min_x, max_x, min_y, max_y, flag_color, matrix
    time_s = time.time()

    if step != 0:  # step=0 步骤为玩家(对手)行棋
        if step == 1:
            # 第一步抢占中心,若中心被占则随机选择周围四点
            if matrix[(SIZE + 1) // 2][(SIZE + 1) // 2] == 0:
                rx, ry = (SIZE + 1) // 2, (SIZE + 1) // 2
            else:
                case = rand.randint(1, 4)
                if case == 1:
                    dx, dy = 1, 1
                elif case == 2:
                    dx, dy = 1, -1
                elif case == 3:
                    dx, dy = -1, 1
                else:
                    dx, dy = -1, -1
                rx, ry = (SIZE + 1) // 2 + dx, (SIZE + 1) // 2 + dy
        else:
            min_tx1, min_ty1, max_tx1, max_ty1 = legal_range(min_x, min_y,
                max_x, max_y)

            eva_matrix1 = np.zeros((SIZE + 2, SIZE + 2), dtype=
                int)  # 第一层的估值矩阵
            value_max = -INF
```

```
386             rx, ry = 0, 0
387             #第一层搜索
388             for i in  range(min_tx1, max_tx1 + 1):
389                 for j in  range(min_ty1, max_ty1 + 1):
390                     # 是否剪枝
391                     flag_cut = False
392                     eva_matrix2 = np.zeros((SIZE + 2, SIZE + 2), dtype=
                            int)
393 
394                     if matrix[i][j] == 0:
395                         matrix[i][j] = flag_color
396                         min_tx2, min_ty2, max_tx2, max_ty2 = legal_range(
                                min_tx1, min_ty1, max_tx1, max_ty1)
397                         [list_h, list_v, list_l, list_r] = get_list(i, j,
                                flag_color)
398                         eva1 = evaluation(list_h, list_v, list_l, list_r)
399                         #第二层搜索
400                         for _i in  range(min_tx2, max_tx2 + 1):
401                             for _j in  range(min_ty2, max_ty2 + 1):
402 
403                                 if matrix[_i][_j] == 0:
404                                     matrix[_i][_j] = -flag_color
405                                     [list_h, list_v, list_s, list_b] =
                                            get_list(_i, _j, -flag_color)
406                                     eva2 = -evaluation(list_h, list_v,
                                            list_s, list_b)
407 
408                                     eva_matrix2[_i][_j] = eva2 + eva1
409                                     matrix[_i][_j] = 0
410                                     # 剪枝
411                                     if eva_matrix2[_i][_j] < value_max:
412                                         eva_matrix1[i][j] = eva_matrix2[_i
                                                ][_j]
413                                         flag_cut = 1
414                                         break
415                             if flag_cut:
416                                 break
417 
```

```
418                         if not flag_cut:
419                             value_min = INF
420                             for _i in  range(min_tx2, max_tx2 + 1):
421                                 for _j in  range(min_ty2, max_ty2 + 1):
422                                     if eva_matrix2[_i][_j] < value_min and
                                        matrix[_i][_j] == 0:
423                                         value_min = eva_matrix2[_i][_j]
424
425                             eva_matrix1[i][j] = value_min
426
427                             if value_max < value_min:
428                                 value_max = value_min
429                                 rx, ry = i, j
430
431                         matrix[i][j] = 0
432
433         time_e = time.time()
434         print("Time cost:",  round(time_e - time_s, 4), "s")
435         add_chess(rx, ry, flag_color)
436
437 # 玩家行棋
438 def round_player(pos):
439     x =  round(pos[0] / SPACE)
440     y =  round(pos[1] / SPACE)
441     if 1 <= x <= SIZE and 1 <= y <= SIZE and matrix[x][y] == 0:
442         add_chess(x, y, -flag_color)
443         return True
444
445 # 添加棋子
446 def add_chess(x, y, color):
447     global step, matrix
448     step = step + 1
449     movements.append((x, y, color, step))
450     matrix[x][y] = color
451     update_range(x, y)
452     is_gg()
453
```

```
454 # 撤销棋子
455 def withdraw_chess():
456     global step, matrix
457     i = 0
458     while i != 2 and  len(movements):
459         step = step - 1
460         x = movements[-1][0]
461         y = movements[-1][1]
462         del movements[-1]
463         matrix[x][y] = 0
464         i = i + 1
465 
466 # 绘制文本
467 def draw_text(surf, text, size, x, y):
468     font = pygame.font.SysFont("华文仿宋", size)
469     text_surface = font.render(text, True, COLOR_BLACK)
470     text_rect = text_surface.get_rect()
471     text_rect.center = (x, y)
472     surf.blit(text_surface, text_rect)
473 
474 # 绘制棋子
475 def draw_chess(surf):
476     for move in movements:
477         if move[2] == flag_color:
478             pygame.draw.circle(surf, COLOR_WHITE, (move[0] * SPACE, move[1]
                 * SPACE), 16)
479         else:
480             pygame.draw.circle(surf, COLOR_BLACK, (move[0] * SPACE, move[1]
                 * SPACE), 16)
481 
482 # 判断游戏是否结束
483 def is_gg():
484     global flag_win, flag_gg, flag_start
485     x = movements[-1][0]
486     y = movements[-1][1]
487     color = movements[-1][2]
488     [list_h, list_v, list_l, list_r] = get_list(x, y, color)
```

```
489     if  sum(list_h[1:-1]) == 5 or  sum(list_v[1:-1]) == 5 or
           sum(list_l[1:-1]) == 5 or  sum(list_r[1:-1]) == 5:
490         flag_win = color
491         flag_gg = True
492         flag_start = True
493
494 # 开始界面显示
495 def init_ui(surf):
496     global flag_win, movements, step, matrix, min_x, min_y, max_x, max_y,
           flag_gg, flag_start
497     if flag_start:
498         if flag_win != 0:
499             root = tkinter.Tk()
500             root.withdraw()
501             if flag_win == 1:
502                 tkinter.messagebox.showinfo("游戏结束", "你输了!")
503             else:
504                 tkinter.messagebox.showinfo("游戏结束", "你赢了!")
505         else:
506             screen.blit(background, back_rect)
507         draw_text(surf, "Enter键以开始游戏", 36, WIDTH / 2, HEIGHT / 2)
508
509     pygame.display.flip()
510     flag_win = 0
511     movements = []
512     step = 0
513     matrix = [[0 for i in  range(SIZE + 2)] for j in  range(SIZE + 2)]
514     min_x, min_y, max_x, max_y = 0, 0, 0, 0
515     flag_gg = False
516     flag_waiting = True
517     while flag_waiting and flag_start:
518         clock.tick(FPS)
519         for e in pygame.event.get():
520             if e. type == pygame.QUIT:
521                 pygame.quit()
522                 sys.exit()
523             elif e. type == pygame.KEYDOWN:
524                 if e.key == pygame.K_RETURN:
```

```
525                     round_ai()
526                     flag_waiting = False
527     flag_start = False
528 
529 # 主循环
530 while flag_running:
531     if flag_gg:
532         init_ui(screen)
533     clock.tick(FPS)
534     if step % 2 == 1:  # ai行奇数步
535         round_ai()
536     else:
537         for event in pygame.event.get():
538             if event. type == pygame.QUIT:
539                 flag_running = False
540             elif event. type == pygame.MOUSEBUTTONDOWN:
541                 round_player(event.pos)
542 
543             elif event. type == pygame.KEYDOWN:
544                 if event.key == pygame.K_r:
545                     flag_gg = True
546                 elif event.key == pygame.K_w:
547                     withdraw_chess()
548                 elif event.key == pygame.K_q:
549                     flag_running = False
550 
551     draw_background(screen)
552     draw_chess(screen)
553     draw_text(screen, "R:重开 -- W:悔棋 -- Q:退出", 28, WIDTH // 2, SPACE
        //2)
554     pygame.display.flip()
555 
556 pygame.quit()
557 sys.exit()
```