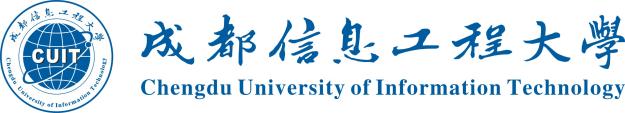
****

**实 验 报 告**

|  |  |
| --- | --- |
| **实验课程：** | **人工智能导论** |
| **实验项目：** | **实验二 基于Alpha-Beta剪枝搜索算法的五子棋游戏** |
| **指导教师：** | **陈美** |
| **学生姓名:** |  |
| **学生学号：** |  |
| **班 级：** |  |
| **分 数：** |  |

20 24 年 9 月 19 日

成都信息工程大学 计算机学院

## 问题描述

在博弈问题中，每一个格局可供选择的行动方案都有很多，因此会产生十分庞大的博弈树，如果只采用极大极小搜索算法，博弈树则会存在着一定的数据冗余。因此，将 Alpha-Beta 剪枝加入极大极小搜索算法中，可以极大地提升整体搜索效率，该算法称为 Alpha-Beta 剪枝搜索算法。

实现采用 Alpha-Beta 剪枝搜索算法实现五子棋游戏，棋盘大小为12×12，可使用 Java 或者 Python 语言实现。此题目参考分值80 分。

采用自己的照片（为避免看不清棋盘线，可采用一定的透明度）画12\*12的格子作为棋盘。此题目参考分值20 分，写一份解决方案阐述

## 解决方案阐述

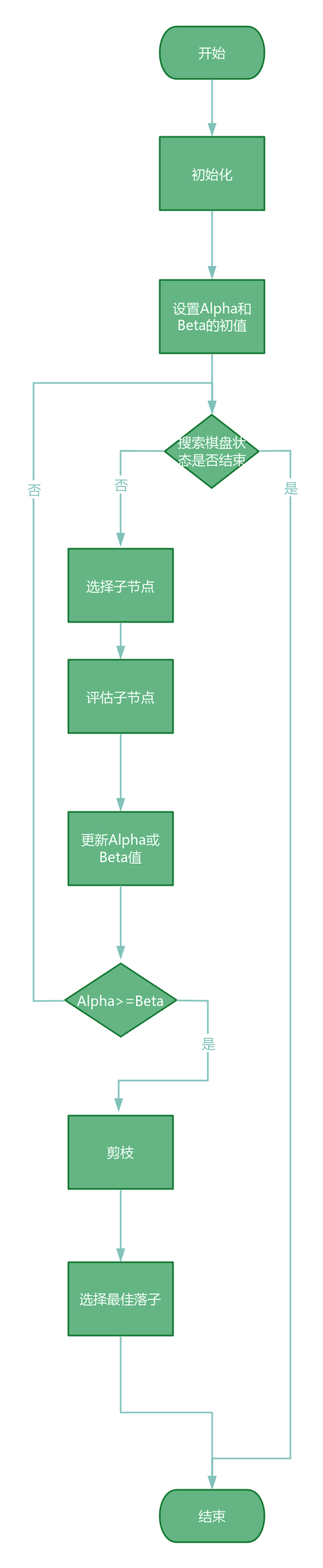
Alpha-Beta剪枝算法是一种在极小化极大算法（Minimax）的基础上的优化。它通过维护两个值，Alpha（当前局面的已知最小值）和Beta（当前局面的已知最大值），来减少搜索树中必须评估的节点数。这种方法可以有效地减少计算量，提高搜索效率。

在该实验中，我使用Python的Tkinter库来创建一个12x12的棋盘界面，棋盘使用透明度处理，以便在棋盘上清晰地显示棋子。

## 设计算法描述

1. 模块划分
2. 用户界面（UI）模块：负责绘制棋盘和棋子，处理玩家的落子操作。
3. 游戏核心模块：管理游戏逻辑，包括落子规则和胜负判定。
4. 人工智能（AI）模块：实现Alpha-Beta剪枝算法，AI根据算法决定自己的落子位置。
5. 数据管理模块：维护游戏状态和历史记录。
6. 用流程图描述关键算法，只需要列举算法中的主要步骤，不用细化到具体代码。

Alpha-Beta剪枝算法流程图：



## 算法实现（即完整源程序，带注解）

该核心搜索函数使用负值极大算法（negamax）加上 alpha-beta 剪枝来查找最佳的落子位置：

1. *# 负值极大算法搜索 alpha + beta剪枝*
2. def negamax(is\_ai, depth, alpha, beta):
3. *# 检查游戏是否结束或者递归深度是否已经达到边界*
4. if game\_win(list1) or game\_win(list2) or depth == 0:
5. *# 如果游戏结束或达到边界，调用评估函数返回当前局面的评估值*
6. return evaluation(is\_ai)
7. *# 获取所有可能的空位，并排除已经落子的位置*
8. blank\_list = list(set(list\_all).difference(set(list3)))
9. *# 对空位进行排序，提高剪枝效率*
10. order(blank\_list)
11. *# 遍历每一个候选落子位置*
12. for next\_step in blank\_list:
13. global search\_count
14. search\_count += 1  *# 增加搜索次数*
15. *# 如果要评估的位置没有相邻的子，则不进行评估，以减少计算量*
16. if not has\_neightnor(next\_step):
17. continue
18. *# 模拟 AI 或玩家的落子*
19. if is\_ai:
20. list1.append(next\_step)
21. else:
22. list2.append(next\_step)
23. list3.append(next\_step)
24. *# 递归调用 negamax 函数，模拟对手的落子*
25. value = -negamax(not is\_ai, depth - 1, -beta, -alpha)
26. *# 撤销落子*
27. if is\_ai:
28. list1.remove(next\_step)
29. else:
30. list2.remove(next\_step)
31. list3.remove(next\_step)
32. *# 如果当前值比已知的最大值 alpha 更大，则更新 alpha 值*
33. if value > alpha:
34. if depth == DEPTH:
35. *# 如果在最大深度找到更好的值，则记录下这个落子位置*
36. next\_point[0] = next\_step[0]
37. next\_point[1] = next\_step[1]
38. print(str(value) + "alpha:" + str(alpha) + "beta:" + str(beta))
39. print(list3)
40. *# 如果当前值大于或等于 beta，则表示对手在这个分支上不可能有更好的回应*
41. if value >= beta:
42. global cut\_count
43. cut\_count += 1  *# 增加剪枝次数*
44. return beta  *# 进行剪枝，返回 beta 值*
45. alpha = value  *# 更新 alpha 值*
46. return alpha  *# 返回当前局面的 alpha 值*

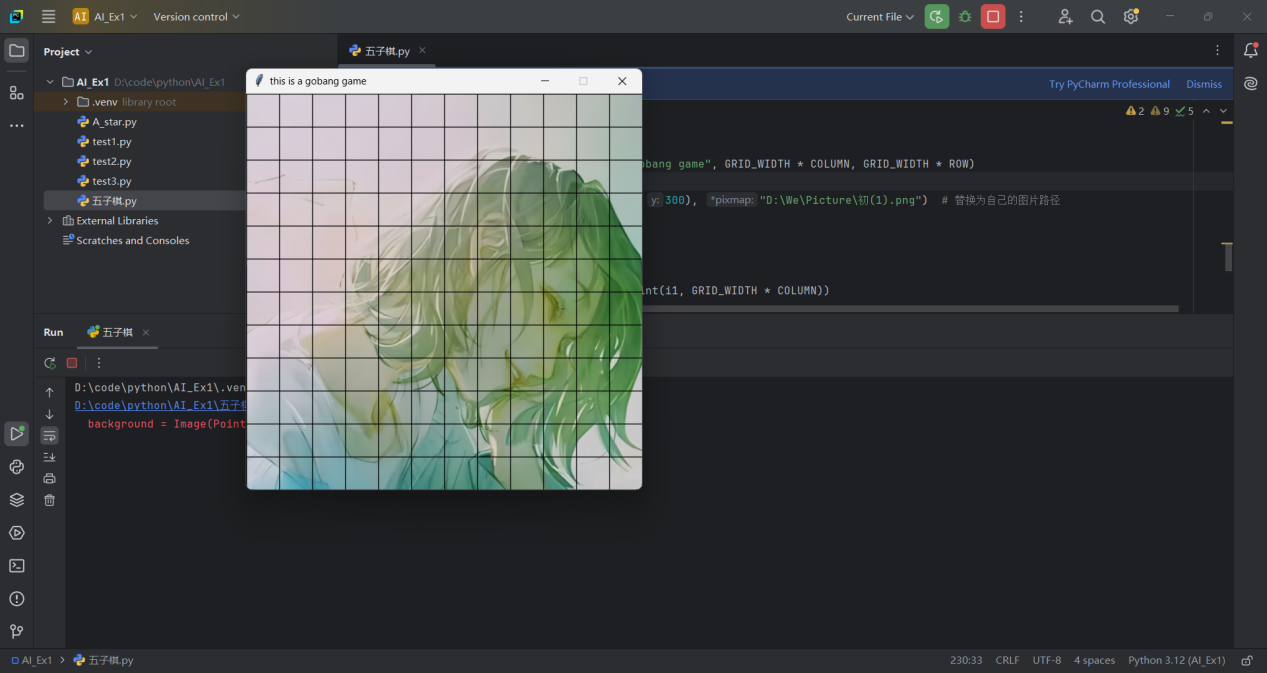
和采用自己的照片有关的代码部分：

1. def gobangwin():
2. win = GraphWin("this is a gobang game", GRID\_WIDTH \* COLUMN, GRID\_WIDTH \* ROW)
3. *#win.setBackground("yellow")*
4. background = Image(Point(300, 300), "D:\We\Picture\初(1).png")  # 替换为自己的图片路径
5. background.draw(win)
6. i1 = 0
7. while i1 <= GRID\_WIDTH \* COLUMN:
8. l = Line(Point(i1, 0), Point(i1, GRID\_WIDTH \* COLUMN))
9. l.draw(win)
10. i1 = i1 + GRID\_WIDTH
11. i2 = 0
12. while i2 <= GRID\_WIDTH \* ROW:
13. l = Line(Point(0, i2), Point(GRID\_WIDTH \* ROW, i2))
14. l.draw(win)
15. i2 = i2 + GRID\_WIDTH
16. return win

## 实验结果测试与分析

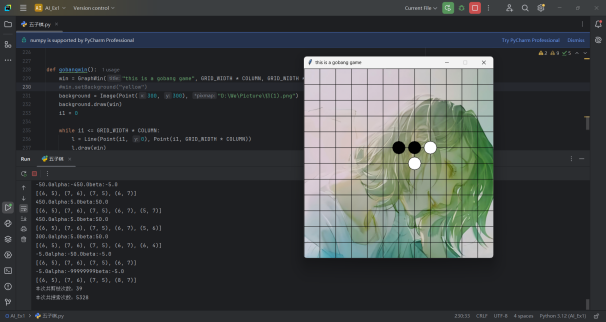
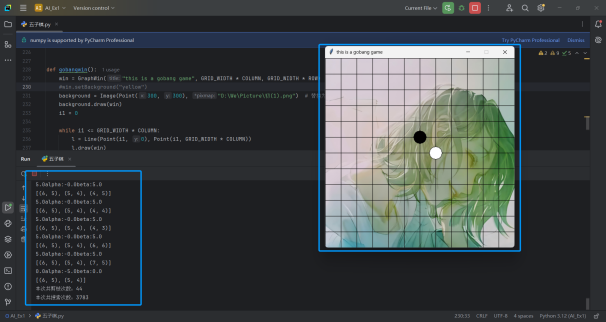
1. 实验结果

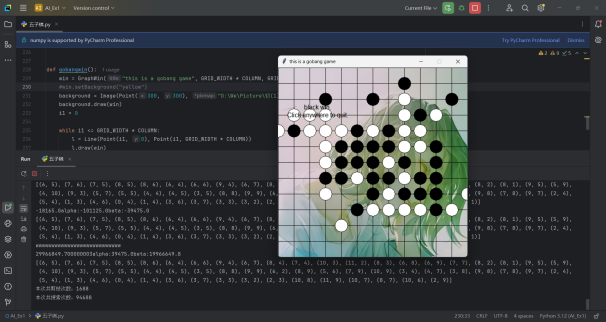
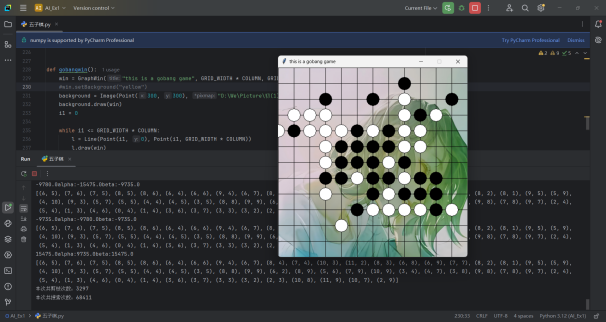
初始界面：



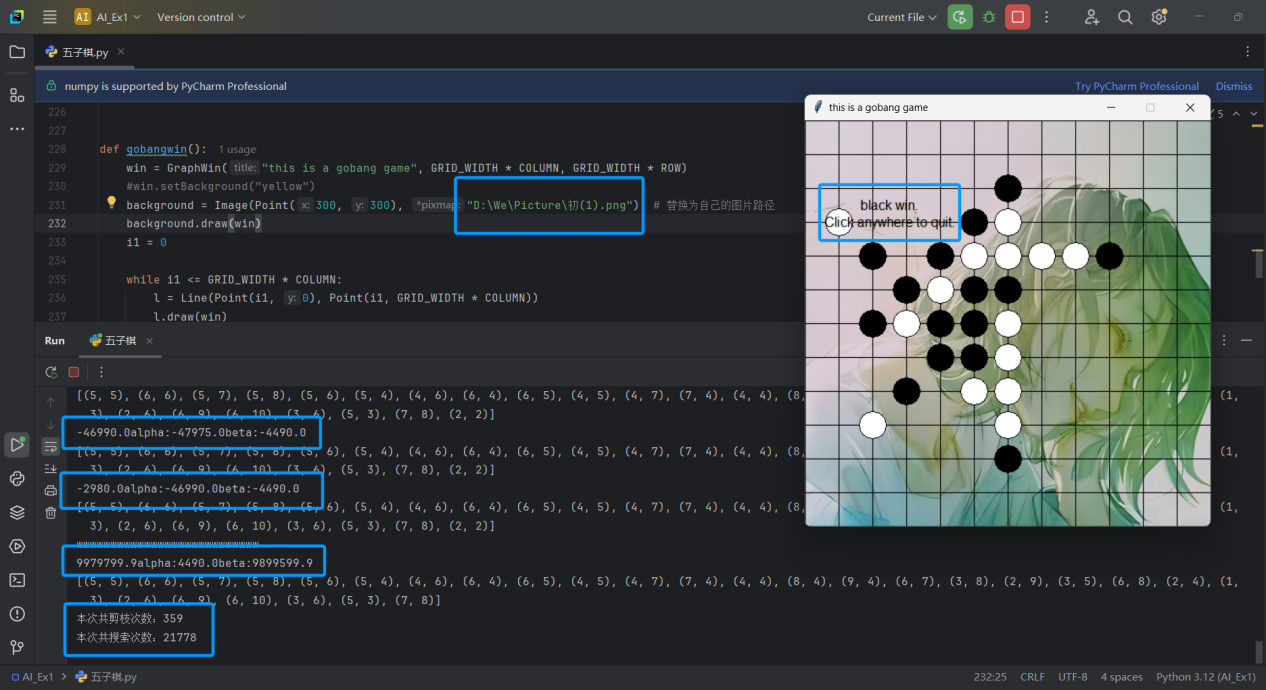
第一次人赢：

下第一步、第二步、中间一步和最后一步：

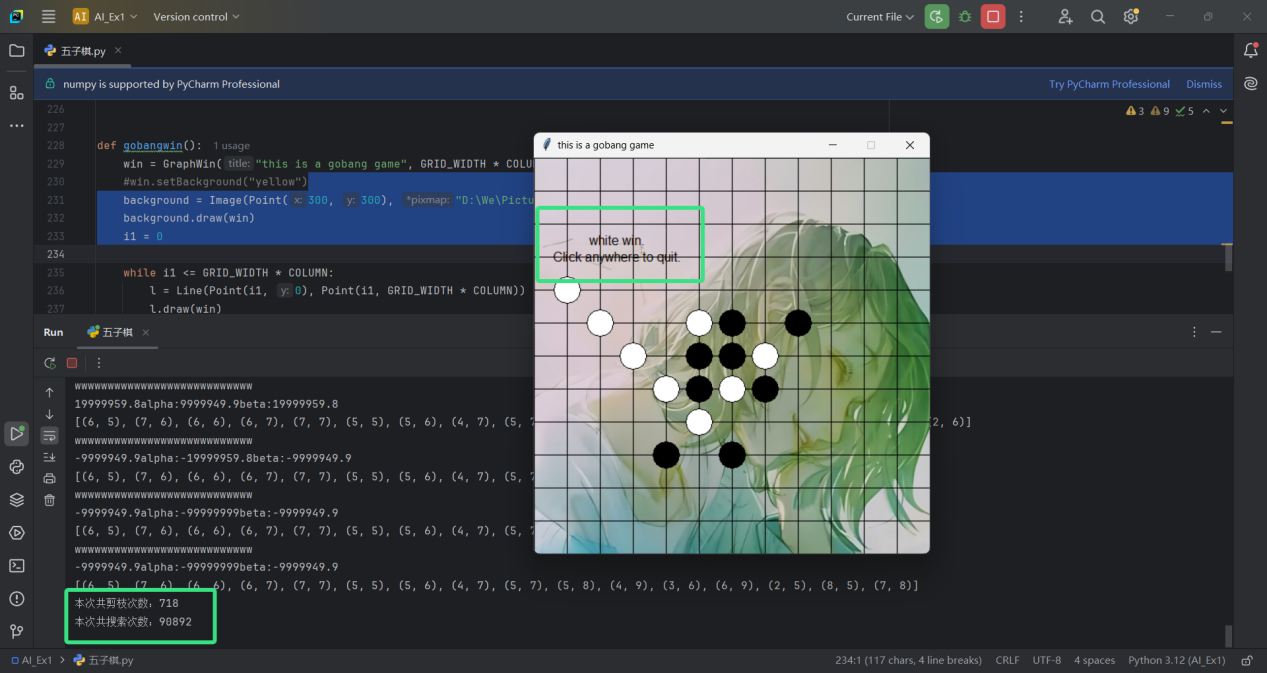




第二次人赢：



AI赢：



1. 分析

经过测试，控制台的遍历时间、次数墩、鼠标和图形化等都没有问题。

## 思考及学习心得

1. 随着棋盘尺寸的调整，原本的坐标计算方法不再适用。我深入分析图形库的坐标系统，重新计算了棋子和棋盘线条的位置，确保它们在新的棋盘尺寸下能够正确地显示。
2. 在这次实验中，我创新性地将个人照片作为五子棋游戏的棋盘背景，这一改变不仅为传统的游戏界面增添了个性化元素，也提升了视觉体验。通过这个过程，我学会了如何将图像处理与游戏逻辑相结合，并且在实践中深入理解了图形用户界面的设计与实现。这次尝试让我认识到，在软件开发中，创新的视觉设计可以有效提升用户的互动体验和游戏的吸引力。同时，我也体会到了在编程中追求细节和完美的重要性。

在这次五子棋游戏项目的实验中，我不仅深入理解并实现了高效的Alpha-Beta剪枝搜索算法，还通过亲手编码和调试，增强了自己在Python和Tkinter界面设计方面的实践经验。通过**将个人照片融入棋盘背景**，我探索了图像处理技术，同时学会了如何优化用户体验。项目模块化的设计让我意识到了代码结构的清晰对于维护和扩展的重要性。在调整算法参数的过程中，我体验到了性能优化的挑战，并在遇到问题时通过自学提升了解决问题的能力。此外，将创意融入项目的过程中，也锻炼了我的创新思维。

总的来说，这次实验不仅提升了我的编程技能，也加深了对人工智能算法的理解，是一次宝贵的学习经历。