人 工 智 能 （2023 春季）

实 习 报 告

班　　级 _____

学　　号 _____

姓　　名 _____

评　　分 _____

中国地质大学（武汉）计算机学院

2023 年 04 月

目　　录

（4）关键代码

（5）运行结果及分析

（6）小结

（7）参考文献

实习一　罗马尼亚度假问题

一、题目描述

编程实现：罗马尼亚度假问题

搜索策略：分别采用：代价一致的宽度优先、贪婪算法和 A*算法实现，并进行算法性能比较。

编程实现：编程语言自选；输入罗马里亚简化地图

运行结果：图形化界面包含地图、算法选择、耗散值、生成节点数统计、 运行时间、 路径搜索动态示意等。

二、算法思想

（1）代价一致的宽度优先

代价一致的宽度优先（UCS）算法的思想是在宽度优先搜索的基础上，从起点开始，每次选择当前代价最小的节点进行扩展，直到扩展到终点为止。在进行节点扩展时，需要考虑每个节点的代价，并更新其到起点的距离和前驱节点。

（2）贪婪算法

贪婪算法是指，在对问题求解时，总是做出在当前看来是最好的选择。在本题中，设立一个启发函数 h(n)，每次扩展时，会选择当前节点的所有相邻节点里启发函数值最小的节点进行扩展。

（3）A*算法

A*算法的算法思想是设立一个估价函数 f(n)=h(n)+g(n),其中 h(n)为启发函数，即当前节点到终点的估计距离，g(n)为从起点到当前节点的实际距离。每次扩展时，从当前节点的相邻节点里选择 f(n)值最小的节点进行扩展。

三、软件设计

1、软件说明

语言：C++

环境：MFC

2、需求分析

罗马尼亚度假问题是一个经典的搜索问题，目标是在罗马尼亚的几个城市之间找到最短的路线，以便游览所有城市。在这个问题中，每个城市都是一个节点，每个路径都是一个边。每个边都有一个代价，代表从一个城市到另一个城市的距离。因此，该问题可以被建模为一个图搜索问题。

3、总体设计

（1）读入罗马尼亚简化地图，将其转化为图的形式，并将图表示为邻接矩阵的形式。

（2）对于每个算法（代价一致的宽度优先、贪婪算法和 A*算法），进行搜索，找到最优解。

（3）在图形化界面上展示地图、算法选择、耗散值、生成节点数统计、运行时间、路径搜索动态示意等。

4、详细设计

（1）读入罗马尼亚简化地图：

a. 从文件中读取数据，包括城市名称、城市之间的距离以及起点和终点。

b. 将城市名称和对应的节点编号建立一个映射关系。

c. 根据城市之间的距离建立邻接矩阵。

（2）代价一致的宽度优先算法：

a. 初始化起点节点，并将其放入优先队列中。

b. 从队列中取出节点，并将其所有相邻的节点加入队列中。

c. 对于每个节点，计算其到起点的距离，并更新最短距离。

d. 重复步骤 b 和 c，直到找到终点或队列为空。

（3）贪婪算法：

a. 初始化起点节点，并将其放入优先队列中。

b. 从队列中取出距离终点最近的节点，并将其所有相邻的节点加入队列中。

c. 对于每个节点，计算其到起点的距离，并更新最短距离。

d. 重复步骤 b 和 c，直到找到终点或队列为空。

（4）A*算法：

a. 初始化起点节点，并将其放入队列中。

b. 从队列中取出节点，并将其所有相邻的节点加入队列中。

c. 对于每个节点，计算其到起点的距离和到终点的距离之和，并更新最短距离。

d. 重复步骤 b 和 c，直到找到终点或队列为空。

（5）图形化界面：

a. 展示罗马尼亚地图。

b. 提供算法选择功能，让用户选择代价一致的宽度优先、贪婪算法或 A*算法。

c. 在界面上展示生成的节点数统计和运行时间等信息。

d. 在地图上展示搜索过程中经过的路径，以及起点和终点等信息。

e. 提供动态示意功能，让用户可以观察搜索过程中节点的生成和扩展等情况。

四、关键代码

1、数据结构

（1）USC

```
// 创建一个初始值为-1 的 pre 向量，用于记录路径
std::vector<int> pre(CITY_NUM, -1);
// 定义一个结构体，用于存储城市 ID 和该城市到起点的距离
struct node
{
    int ID;  // 城市 ID
    int dis; // 距离起点的距离
    // 重载小于运算符，用于比较距离大小，以便使用优先队列
    bool operator<(const node& other) const { return dis > other.dis; };
};
// 记录已经访问的城市数量
int cnt = 1;
```

```
// 创建一个优先队列，用于存储每个城市到起点的距离，按照距离从小到大排序
std::priority_queue<node> q;
// 创建一个向量，用于存储每个城市到起点的距离，默认为无穷大
std::vector<int> dis(CITY_NUM, INF);
```

（2）贪婪

```
// 创建一个初始值为-1 的 pre 向量，用于记录路径
std::vector<int> pre(CITY_NUM, -1);
// 定义一个结构体，用于存储城市 ID 和该城市的估价函数值 h
struct node {
    int ID;  // 城市 ID
    int h;   // 城市的估价函数值
    // 重载小于运算符，按照估价函数值从小到大排序，以便使用优先队列
    bool operator<(const node& other) const {
        return other.h < h;
    }
};
// 创建一个优先队列，用于存储每个城市到起点的估价函数值，按照估价函数值从小到大排序
std::priority_queue<node> q;
// 创建一个向量，用于存储每个城市到起点的距离，默认为无穷大
std::vector<int> dis(CITY_NUM, INF);
```

（3）A*

```
std::vector<int> pre(CITY_NUM, -1); // 初始化 pre 数组，-1 表示没有前一个节点
struct node {
    int ID;
    int f;
    bool operator<(const node& other) const { // 重载小于号运算符，用于堆排序
        return other.f < f; // 优先级队列按照 f 值从小到大排序，f 值越小越优先
    }
```

```
};
int cnt = 1; // 记录节点扩展的次数
std::priority_queue<node> q; // 定义优先级队列，用于保存搜索中的节点
std::vector<int> dis(CITY_NUM, INF); // 初始化 dis 数组，表示从起点到各个节点的距离
```

2、搜索算法

（1）USC

```
// 将起点的距离设置为 0
dis[0] = 0;
// 开始进行 USC 算法
while (!q.empty())
{
    // 取出队列中距离起点最近的城市
    node cur = q.top();
    q.pop();
    // 如果该城市是终点，直接退出循环
    if (cur.ID == 1) break;
    // 遍历当前城市的所有邻居城市
    for (int i = 0; i < CITY_NUM; i++)
    {
        // 如果当前城市与邻居城市之间没有直接连接，则继续遍历
        if (distance[cur.ID][i] == INF || i == cur.ID) { continue; }
        // 如果当前城市到邻居城市的距离比之前记录的更短，则更新距离和路径
        if (distance[cur.ID][i] + dis[cur.ID] < dis[i])
        {
            dis[i] = dis[cur.ID] + distance[cur.ID][i]; // 更新距离
            pre[i] = cur.ID; // 记录路径
            q.push(node{ i,dis[i] }); // 将邻居城市加入优先队列
            ++cnt; // 记录已经访问的城市数量
        }
```

```
    }
}
```

（2）贪婪

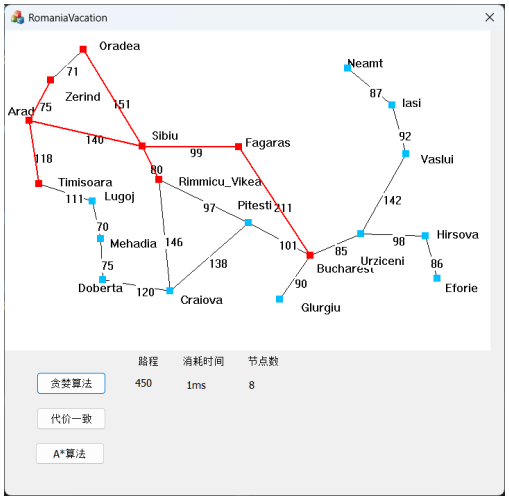
```
// 记录已经访问的城市数量
int cnt = 0;
// 开始进行贪婪算法
while (!q.empty()) {
    // 取出队列中估计值最小的城市
    node cur = q.top();
    q.pop();
    // 如果该城市是终点，直接退出循环
    if (cur.ID == 1) {
        break;
    }
    // 遍历当前城市的所有邻居城市
    for (int i = 0; i < CITY_NUM; i++) {
        // 如果当前城市与邻居城市之间没有直接连接，则继续遍历
        if (distance[cur.ID][i] == INF || i == cur.ID) {
            continue;
        }
        // 记录已经访问的城市数量
        ++cnt;
        // 将邻居城市加入优先队列，并更新距离和路径
        q.push(node{ i,h[i] });
        if (dis[i] > dis[cur.ID] + distance[cur.ID][i]) {
            dis[i] = dis[cur.ID] + distance[cur.ID][i]; // 更新距离
            pre[i] = cur.ID; // 记录路径
        }
    }
}
```

（3）A*

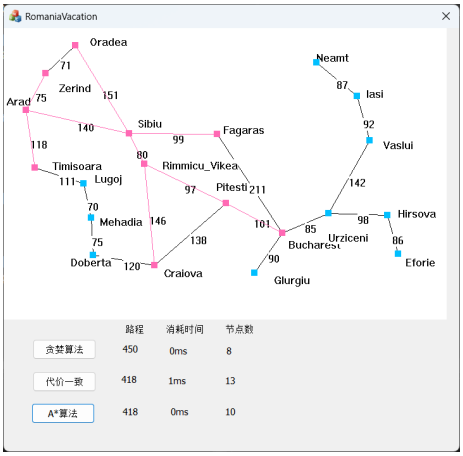
```
q.push(node{ 0, h[0] }); // 将起点加入优先级队列
dis[0] = 0; // 起点到自身的距离为0
while (!q.empty()) { // 循环直到队列为空
    node cur = q.top(); // 取出f值最小的节点
    q.pop(); // 从队列中删除该节点
    if (cur.ID == 1) { // 如果该节点是终点，则跳出循环
        break;
    }
    for (int i = 0; i < CITY_NUM; i++) { // 枚举当前节点的所有邻居节点
        if (distance[cur.ID][i] == INF || i == cur.ID) { // 如果当前节点和邻居节点之间没有边或者是同一个节点，则跳过
            continue;
        }
        q.push(node{ i, dis[cur.ID] + distance[cur.ID][i] + h[i] }); // 将邻居节点加入优先级队列
        ++cnt; // 统计节点扩展的次数
        if (dis[i] > dis[cur.ID] + distance[cur.ID][i]) { // 更新dis和pre数组
            dis[i] = dis[cur.ID] + distance[cur.ID][i];
            pre[i] = cur.ID;
        }
    }
}
```

五、运行结果及分析

1、运行结果



贪婪算法　　USC



A*算法

2、分析

在罗马尼亚度假问题中，三种不同的搜索算法都可以找到解，USC 和 A*算法找到的路程都为 418，而贪婪算法为 450。

USC 算法和 A*算法生成节点数较多，贪婪算法生成节点数较少。这表明贪婪算法的搜索效率更高，但是可能会牺牲搜索质量。而 USC 算法和 A*算法的搜索质量更高，但是可能需要更多的时间和计算资源。

另外，三种算法的运行时间都为 0ms，这是因为在这个题目中，搜索的时间非常短，无法被计算出来。

六、小结

罗马尼亚度假问题是一个经典的搜索问题，在本次实习中，我使用代价一致的宽度优先算法、贪婪算法和 A*算法三种不同的搜索算法对该问题进行了求解，并通过图形化界面展示了搜索过程和结果，加深了对搜索算法的理解和应用。通过比较三种不同的搜索算法的性能，可以发现，不同的算法具有不同的优劣性和适用场景。代价一致算法和 A*算法具有较高的搜索质量，但需要更多的时间和计算资源；贪婪算法具有较高的搜索效率，但搜索质量可能不够高。

七、参考文献

[数据结构 & 算法]A*算法--罗马尼亚度假问题（实验课作业） 芒果和小猫的博客-CSDN 博客

罗马尼亚度假问题 人工智能搜索算法全代码 C++（深度优先，广度优先，等代价，迭代加深，有信息搜索，A*算法，贪婪算法）_c++人工智能代码_我要做个小 Pro 的博客-CSDN 博客

实习二 采用最小冲突法求解 n 皇后问题

一、问题描述

采用最小冲突法求解 n 皇后问题，要求 N≥80。

二、算法思想

最小冲突法是一种解决约束满足问题（CSP）的贪心算法。最小冲突法的基本思想是，首先将每个皇后随机放置在一列中，然后逐个移动皇后，将其移动到所在列中冲突最少的位置。如果有多个位置具有相同的最小冲突数，则从中随机选择一个位置移动皇后。这个过程一直进行，直到没有任何皇后可以移动为止，此时就得到了一个解。

三、软件设计

1、软件说明

语言：C++

环境：Visual Studio 2022

2、需求分析

输入：皇后数量 N

输出：N≤100 时，打印棋盘；N＞100 时，输出每行每个皇后所处的列。

3、总体设计

（1）初始化一个长度为 N 的列表，随机将每个皇后放置在一列中

（2）循环执行以下步骤，直到没有任何皇后可以移动：

a. 对于每个皇后，计算其所在列的冲突数，即与其他皇后在同一列或同一对角线上的数量

b. 对于每个皇后，将其移动到冲突数最小的位置。如果有多个位置具有相同的最小冲突数，则从中随机选择一个位置移动皇后。

（3）返回皇后位置

四、关键代码

1、初始化

```
//row[i]表示当前摆放方式下第 i 行的皇后数，col[i]表示当前摆放方式下第 i 列的皇后数
int row[MAX];
int col[MAX];
int N; //放置 N 个皇后在 N*N 棋盘上
void init()
{
    for (int i = 0; i < N; i++) {//N queens
        R[i] = i;
    }
    randomize(R, 0, N);//初始化 N 个皇后对应的 R 数组为 0~N-1 的一个排列，即没有任意皇后同列，也没有任何皇后同行
    for (int i = 0; i < N; i++) {
        row[i] = 1;
        col[i] = 0;
    }
    for (int i = 0; i < 2 * N - 1; i++) {
        pdiag[i] = 0;
        cdiag[i] = 0;
    }
    for (int i = 0; i < N; i++) {
        col[R[i]]++;
        pdiag[getP(i, R[i])]++;
        cdiag[getC(i, R[i])]++;
    }
}
```

2、判断是否有冲突

```
bool qualify() {
    for (int i = 0; i < N; i++) {
        if (col[R[i]] != 1 ||pdiag[getP(i, R[i])] != 1 ||cdiag[getC(i, R[i])] != 1) {
            return false;
        }
    }
```

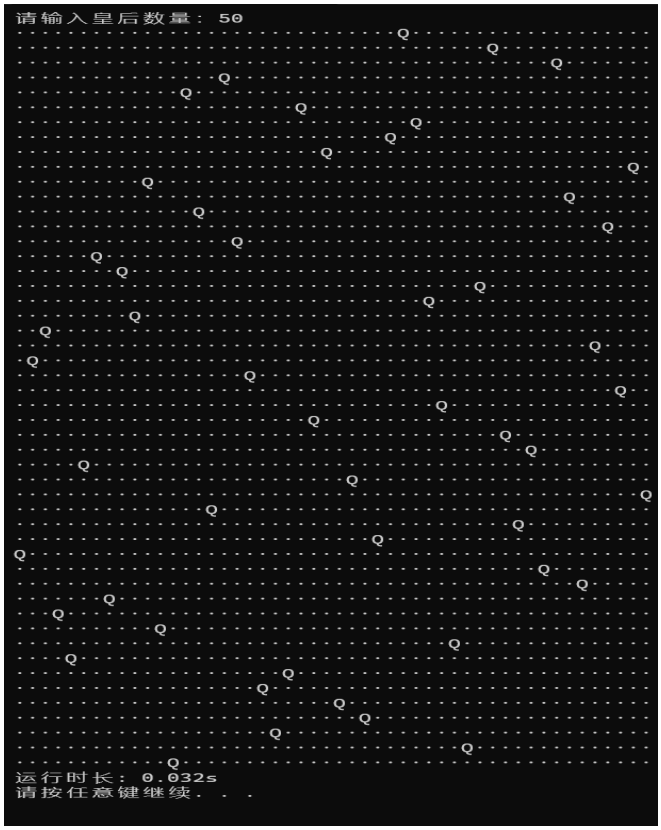
```
    return true;
}
```

3、调整皇后位置

```
bool adjust_row(int row) {
    int cur_col = R[row];
    int optimal_col = cur_col;//最佳列号，设置为当前列，然后更新
    int     min_conflict = col[optimal_col] + pdiag[getP(row, optimal_col)] - 1
        + cdiag[getC(row, optimal_col)] - 1;//对角线冲突数为当前对角线皇后数减一
    for (int i = 0; i < N; i++) {//逐个检查第 row 行的每个位置
        if (i == cur_col) {
            continue;
        }
        int conflict = col[i] + pdiag[getP(row, i)] + cdiag[getC(row, i)];
        if (conflict < min_conflict) {
            min_conflict = conflict;
            optimal_col = i;
        }
    }
    if (optimal_col != cur_col) {//要更新 col,pdiag,cdiag
        col[cur_col]--;
        pdiag[getP(row, cur_col)]--;
        cdiag[getC(row, cur_col)]--;

        col[optimal_col]++;
        pdiag[getP(row, optimal_col)]++;
        cdiag[getC(row, optimal_col)]++;
        R[row] = optimal_col;
        if (col[cur_col] == 1 && col[optimal_col] == 1
            && pdiag[getP(row, optimal_col)] == 1 && cdiag[getC(row, optimal_col)] == 1) {
            return qualify();//qualify 相对更耗时，所以只在满足上面基本条件后才检查
        }
    }
    //当前点就是最佳点，一切都保持不变
    return false;//如果都没变的话，肯定不满足终止条件，否则上一次就应该返回 true 并终止了
    //return qualify();
}
```

五、运行结果及分析

1、N=40



2、N=5000

```
请输入皇后数量： 2000
每一个Q在每一行的位置：
1820 369 1017 1470 92 231 1104 917 130 1486 1114 960 1410 556 1368 1117 1369 398 1896 74 1056 1360 1198 368 1643 1223 444 1180 172 1616 911 851 17 1336 297
1418 1632 1577 1671 1585 1285 1656 1472 83 1578 1551 276 1775 970 1662 1229 1142 1623 1389 1883 1619 1712 1043 1648 1667 1735 281 1721 1317 1589 1823 76 922
 410 467 989 400 957 1884 806 1404 1187 290 1517 482 952 1466 1710 981 1606 205 613 242 1450 497 1805 244 1173 1812 694 1185 1224 1799 1357 1750 764 1310 13
49 342 85 1430 437 794 340 1860 1749 349 1165 1672 1193 501 1618 1415 730 751 936 906 1708 512 710 381 277 523 1772 294 563 895 502 1335 332 1600 1867 1804
1864 1150 1518 938 1934 738 1939 732 1846 346 1009 947 810 1250 1440 842 941 1037 236 446 1128 1856 1818 795 1139 113 990 676 905 1006 1917 1829 364 1236 11
38 184 95 123 567 532 791 1506 958 1259 1994 951 1891 1239 1001 1680 1929 1685 1258 1561 370 111 1189 1526 1003 662 1294 1937 854 1000 612 699 401 554 1598
1941 84 1501 170 1219 1983 708 991 79 1352 978 131 811 1338 331 1916 1387 315 1748 807 1371 1878 1377 603 943 543 1066 1355 1837 995 932 1746 907 135 1839 6
41 617 537 344 195 1432 731 918 1718 850 929 772 1759 1550 1810 1195 686 1971 1304 1080 403 1925 1312 1564 457 883 1437 669 354 321 1620 1919 835 1882 222 5
44 578 1980 626 473 246 1795 994 1984 1132 1778 743 1862 1573 428 432 1959 1394 262 624 105 593 853 1525 780 756 150 1099 1840 1422 1308 904 229 1701 1580 4
66 1796 1920 864 133 1545 1442 786 1124 470 1990 896 396 1791 767 984 435 1719 1999 591 1332 1898 486 519 542 859 1900 722 1011 1306 271 68 1334 1833 856 66
1 307 1488 935 988 1038 279 1529 1844 779 1997 492 1232 1621 1599 620 1319 1838 982 71 949 489 857 430 1907 622 1754 1491 87 892 598 1103 1798 828 335 1956
434 1391 660 1687 1023 1935 1268 1965 797 1936 139 1571 1692 1348 119 1958 433 1966 201 156 439 1414 887 1943 733 755 203 696 1816 1852 481 509 1417 561 193
1 1307 323 1221 207 1579 461 689 971 1282 248 635 1530 1988 1498 1951 67 1100 326 1075 881 1881 1509 453 1703 180 1157 1200 320 1220 1073 900 1170 1794 1711
 1923 1973 1608 1215 1028 1566 1413 1586 1740 695 1698 424 177 1510 1271 1593 677 485 649 360 975 1341 1446 1657 1029 992 429 356 110 1444 1976 1955 1361 12
 1922 336 374 445 359 1806 1188 1383 1767 633 1305 232 1428 789 1255 588 1582 706 870 1670 1668 1828 143 1802 1584 500 385 1607 1507 1162 9 713 1893 69 1333
 16 1364 1981 1452 1849 325 871 781 1036 70 275 394 174 155 1111 716 1045 1480 324 1411 1192 966 800 869 705 843 682 404 264 21 768 998 1827 1263 1613 1690
253 1768 511 1051 785 1720 1398 579 834 245 1342 1265 258 1736 1190 655 923 639 1841 670 1125 144 460 1817 167 44 1485 1326 1622 1635 376 636 874 417 1933 1
859 1615 496 41 116 1202 1325 1482 228 1012 1562 469 1386 1253 1093 777 1438 1533 1967 1975 1553 118 1568 1544 1932 176 1630 285 1871 1899 515 1592 181 1363
 1320 1216 261 630 1039 604 1909 169 865 93 301 1109 1159 200 816 774 1061 1203 468 241 414 1068 546 209 94 559 1538 1921 844 471 618 142 596 531 1290 1998
609 925 1688 679 698 4 1858 1183 1603 1222 1513 1090 216 337 1733 723 187 1853 615 250 1855 1097 214 804 614 819 1181 1123 885 61 91 1801 256 1627 1176 1785
 348 846 916 1321 1689 1110 1490 312 291 808 422 304 1279 993 1962 1476 1072 1707 353 60 1781 1264 1500 504 1160 1964 909 443 683 894 1724 1793 1392 1836 67
2 1845 1337 1915 137 1633 141 863 1610 1654 1350 1876 124 536 538 784 628 431 1628 1249 1758 737 687 198 82 1832 1295 961 763 153 472 893 1256 1228 744 350
2 1274 194 1329 1972 1887 1783 1728 821 1892 1581 6 206 1094 1497 667 558 942 237 1763 1536 631 22 1 1194 1797 1121 8 13 1755 1626 1570 526 221 551 1235 627
 1693 1947 675 564 413 1021 1059 1468 462 602 1425 517 1044 494 1276 268 227 796 757 1447 629 1403 637 753 282 1106 477 643 306 146 26 1169 0 1372 406 1475
456 697 389 450 1163 1408 476 1961 1908 367 57 292 1948 540 1243 1658 78 1842 1088 393 11 1770 278 72 1013 463 52 996 260 1495 1637 1512 168 1591 1565 1697
1576 1725 878 1987 449 1024 375 302 1636 1067 969 829 20 77 721 1353 474 658 1004 274 90 19 1774 882 252 1523 65 1008 1885 1760 1070 1515 884 761 516 1863 7
35 233 1199 822 1135 1315 590 1664 343 1940 1345 924 955 574 1982 1611 1991 1380 42 1291 954 1723 728 452 1706 619 1522 1257 1505 179 1126 408 299 571 1743
384 154 464 1696 1244 1537 1460 1784 503 254 1769 739 397 1911 1819 1152 32 251 1309 107 1715 782 1645 392 977 597 953 1461 1650 1914 289 1426 1874 217 1877
 1986 1374 1527 7 1519 1149 718 293 160 1018 27 1370 937 534 97 607 775 1665 213 1102 1974 1060 223 1850 740 183 15 81 23 1605 681 1141 1609 1385 1095 80 14
64 35 1457 1996 24 548 1419 541 824 1382 14 48 1273 1745 1942 714 1653 29 382 726 1262 817 219 1443 1870 18 967 34 1122 799 1213 1732 1753 1154 96 566 373 1
47 101 623 1766 1727 455 98 818 1705 608 238 365 849 525 112 1246 1133 963 1449 1167 550 88 841 366 976 1926 1454 1354 1868 1297 204 164 1508 1120 1327 889
 950 1032 648 265 858 1405 1661 524 830 1779 1902 1217 345 1127 1809 411 1848 50 1927 1035 832 1875 1640 51 650 386 815 186 1729 145 1134 454 1751 1826 210 1
738 968 709 55 1339 45 313 1787 1206 1587 104 479 1164 930 685 652 1484 1242 39 573 1644 749 1747 788 266 1201 1400 873 1702 1299 801 827 102 1542 888 267 2
24 1624 1208 493 334 1594 296 1563 395 220 1714 1351 1479 379 1717 407 1467 303 1266 707 1993 399 1614 1524 1629 1146 1076 1156 891 1303 62 286 527 125 836
945 599 134 1469 1055 553 1641 1137 64 159 225 480 86 724 820 1757 973 1780 508 1676 1063 592 1054 1677 1761 1225 549 1596 322 1207 103 520 570 166 1311 114
8 1053 1709 1532 760 1071 249 314 73 1953 659 442 1481 595 25 793 163 1316 31 191 702 1041 1788 1834 383 1552 1433 719 1281 1254 158 1175 1096 1166 742 890
1427 717 645 1042 1954 1462 1963 1531 812 1098 270 1647 189 1359 1015 1434 1313 1362 1158 49 318 288 363 10 1455 1172 1651 416 1556 1649 1503 1057 915 752 6
63 1086 580 1910 127 1595 1366 420 1375 582 317 1048 1814 903 700 1546 152 1944 1079 1238 901 1625 1182 1928 215 1602 1014 506 1866 611 28 606 1376 487 1617
 1269 1675 727 860 1252 1186 1869 1652 3 30 441 759 193 459 1212 765 298 529 190 693 1673 931 745 1240 729 1356 149 263 586 688 328 1726 1950 715 202 1145 1
843 1930 638 1548 1101 46 1968 862 647 1949 1477 1267 877 1196 1318 295 1022 711 362 589 1390 1424 1459 919 1679 771 734 208 1441 1737 1016 1821 1323 121 12
37 330 1886 1583 1436 1280 329 1034 1776 505 1409 921 584 1691 1960 211 1069 1251 1062 1704 1171 1555 438 575 1741 666 284 1678 671 1040 1049 847 939 1803 9
26 1116 100 483 1904 880 837 691 1815 826 1731 185 1346 1560 762 813 1888 1129 1824 1002 59 867 33 1483 1782 1184 839 1504 1058 1210 1851 956 310 1590 498 1
205 1328 157 47 1218 1894 372 1078 840 54 547 1046 1020 1695 1716 1340 132 448 1631 269 528 656 908 1296 1813 1439 1209 1492 1989 415 912 1084 1952 1995 197
9 1083 690 37 1612 985 783 803 136 5 1765 1092 1493 1211 838 43 499 861 287 1082 257 161 665 750 1539 377 53 1901 1830 539 212 1396 402 668 1025 478 357 129
3 305 600 199 1115 358 1261 1686 1247 1905 1365 1808 300 1458 1278 572 866 197 1052 273 1574 1906 610 490 518 568 390 522 987 899 680 972 1516 243 712 510 1
050 962 1682 192 1178 594 311 1286 99 234 778 1197 1655 173 1105 63 1179 1471 565 1275 1064 1792 1373 1660 391 577 1287 792 823 876 1151 898 1448 1730 114 8
02 673 418 1300 421 495 75 1379 557 1289 178 1431 1639 1119 1588 338 255 1065 1226 1601 1230 1388 1807 1301 1140 605 959 1474 1298 1946 129 1324 426 1033 15
67 914 1347 1825 380 1284 974 1912 1091 651 736 151 1367 642 1674 122 371 1330 1314 845 552 1144 1288 1407 1604 1646 1331 283 555 1406 1985 1147 946 1131 14
8 115 1344 1713 1496 1597 1811 1756 933 1889 361 773 308 535 1031 1771 425 513 1231 1831 38 754 1547 419 58 1005 218 965 910 341 1292 1554 814 89 746 1694 1
75 1969 999 387 1502 1234 616 1241 1634 560 1499 427 339 833 1085 1161 725 1174 879 108 897 1402 162 1421 944 1451 1204 587 634 1456 701 1136 36 1534 1642 6
40 120 653 1019 475 986 1473 703 1572 1744 226 1540 423 1857 1272 1074 347 1113 1277 1638 1666 664 1549 678 126 872 247 1081 601 1699 530 766 1047 1734 1789
 1283 1378 409 875 1027 980 1722 1453 1700 948 488 868 1270 1429 188 1227 1890 1918 1865 1879 1521 927 1786 576 514 798 352 1010 790 1343 855 1401 1260 1155
 1420 1007 1873 770 940 997 1168 1659 1302 327 1393 412 272 1913 1118 1535 1895 1322 1684 928 581 1412 1938 646 1077 447 138 1970 1143 491 1130 1397 1108 77
6 240 319 1416 1854 920 747 521 769 787 674 355 886 848 964 1233 1903 1541 1847 1489 1248 979 533 704 40 1107 1435 109 259 333 692 583 1026 1880 165 235 155
9 436 1742 1381 1520 741 351 805 1358 1478 56 1924 1669 1153 1465 1511 545 825 1773 585 1543 1861 1764 1977 451 684 1514 569 562 1790 913 654 1575 1569 1177
 405 1897 621 1214 1463 748 196 458 465 758 1681 182 1494 1112 106 117 1945 983 902 1992 1423 1663 1872 440 809 388 1445 1557 1978 1558 1835 66 831 1395 108
7 507 1762 852 1487 140 1957 934 378 1739 280 484 1089 1245 1752 309 171 657 230 1683 625 316 720 1384 239 632 1191 1399 1030 644 128 1528 1800 1777
运行时长： 0.302s
请按任意键继续. . .
```

3、分析

最小冲突法的时间复杂度为 O(k*n^2)，其中 k 是迭代次数，n 是皇后数量。在 N 较大时，迭代次数可能会很大，因此该算法的时间复杂度可能会很高。但是，该算法通常能够在较少的迭代次数内找到一个解，因此在实践中通常比暴力搜索、回溯法更快。

六、小结

在本次实习中，我学习了如何使用最小冲突法解决 N 皇后问题。N 皇后问题是经典问题 8 皇后的扩展，可以很好地比较一些算法的性能，通过解决这个问题，提高了我的算法性能优化能力。

七、参考文献

n 皇后问题（回溯法-递归法和循环法，最小冲突法（较快解决 10000 级别问题））_肥宅_Sean 的博客-CSDN 博客

实习三 使用联机搜索算法求解 Wumpus 怪兽世界问题

一、题目描述

使用联机搜索求解 Wumpus 怪兽世界问题：

Wumpus 世界是一个山洞，有 4 x 4 个房间，这些房间与通道相连。因此，共有 16 个房间相互连接。我们有一个以知识为基础的 Agent，他将在这个世界探索。这个山洞里有一间屋子，里面有个叫 Wumpus 的怪兽，他会吃掉进屋的任何人。Agent 可 以射杀 Wumpus,但 Agent 只有一支箭。 在 Wumpus 世界中，有一些陷阱 PIT,如果 Agent 落在深坑中，那么他将永远被困在那里。其中在- -个房间里有可能找到 Gold。因此，Agent 的目标是找到金子并走出洞穴，而不会掉入坑或被 Wumpus 吞噬。如果 Agent 找到金子出来，他会得到奖励;如果被 Wumpus 吞下或掉进 PIT 中，他会受到惩罚。

注意:这里的 Wumpus 是静态的，不能移动。Wumpus 世界的示例图。它显示了在世界(1, 1)正方形位置上的一些带有陷阱 PIT 的房间，带有 Wumpus 的一个

房间和一个 Agent.

传感器: 如果 Agent 在 Wumpus 附近的房间里，他会感觉到恶臭 stench。(不是对角线的)。 如果 Agent 在紧邻陷阱 PIT 的房间内，他会感觉到微风 Breeze. Agent 可以感知到存在 Gold 的房间中的闪光。 Agent 走进墙壁会感觉到撞击。射杀 Wumpus 时，它会发出可怕的尖叫声，在山洞的任何地方都可以感觉到。 这些感知可以表示为五个元素列表，其中每个传感器都有不同的指标。

对 Wumpus 世界的 PEAS 进行了如下描述:

性能指标: 如果 Agent 带着金从洞穴中出来，则可获得 1000 点奖励积分。被 Wumpus 吃掉或掉进坑里的点数为-1000 分。 -1 表示每个操作，-10 表示 使用箭头。 如果 Agent 死亡或从山洞出来，游戏就会结束。

环境: 4 * 4 的房间网格。 该代理最初位于房间正方形[1, 1]中，朝向右侧。 除了第一个正方形[1, 1]以外，都是随机选择 Wumpus 和黄金的位置。 洞穴的每个正方形都可以是第-一个正方形以外的概率为 0.2 的坑(随机产生 PIT 陷阱)。

执行器:左转，右转,前进,抓，射击。

二、算法思想

Wumpus 世界问题可以使用基于规则的专家系统来解决，这种系统根据一系列规则进行推理，以确定 Agent 应采取的行动。具体来说，可以使用联机迭代加深搜索算法，它是一种递归深度优先搜索算法，该算法逐步增加搜索深度，直到找到解决方案或达到最大搜索深度为止。搜索过程中，需要根据感知信息和之前的动作，更新知识库，以便推理出下一步应该采取的行动。

三、软件设计

1、软件说明

语言：C++

环境：QT 5.12.12

2、需求分析

（1）确定感知信息和动作，包括 stench（恶臭）、breeze（微风）、glitter（闪光）、bump（碰撞）和 scream（尖叫）等感知信息，以及左转、右转、前进、抓和射击等动作。

（2）设计知识库，包括规则、事实和推断等，以便进行推理。

（3）实现搜索算法，以便在有限时间内找到最优解决方案。

（4）设计评估函数，以评估代理的性能，并确定代理应该采取的行动。

（5）确定环境和执行器，包括 4 x 4 的房间网格、代理的初始位置和朝向、随机选择 Wumpus 和黄金的位置、可能的陷阱和箭数等。

3、总体设计

（1）初始化 Agent 的起始位置、方向和环境，设置初始得分为 0。

（2）不断执行下列步骤：

a. 根据当前感知，更新 Agent 对环境的知识和对环境中物体位置的推理。

b. 根据 Agent 的知识和推理，选择下一步操作，例如左转、右转、前进、抓、射击等。

c. 执行操作，更新 Agent 的位置、方向和得分。

d. 判断是否达到终止条件，如果是则输出得分并结束，否则返回步骤 a。

e. 根据上述算法，选择合适的搜索算法对该问题进行求解。

4、详细设计

（1）初始化 Agent 的位置和朝向，以及知识库和箭数等。

（2）迭代加深搜索，每次增加搜索深度，直到找到金子或死亡为止。

（3）在每个搜索深度上，根据感知信息和之前的动作，更新知识库。

（4）根据评估函数选择最优动作，包括前进、左转、右转、射击和抓。

（5）执行动作，并根据结果更新知识库和评估函数等。

（6）如果找到金子或死亡，则结束搜索。

四、关键代码

1、联机深度优先搜索

```
int Widget::BFS(Point src, Point tar, bool record)
{
    int Next[4][2] = { {0, 1}, {0, -1}, {-1, 0},{1, 0} };
    int steps = 0;
    map<Point,Point>bfs_path;
    bool flag = false;
    if((agent_world[tar.xx][tar.yy]&VISITED)==0)
    {
        agent_world[tar.xx][tar.yy] |= VISITED;
        flag = true;
    }
    queue<Point>Q;
    Q.push(src);
    bool vis[ROW_NUM][COL_NUM];
    memset(vis,false,sizeof(vis));
    vis[src.xx][src.yy] = true;
    while(!Q.empty())
    {
        Point cur = Q.front();
        Q.pop();
        ++steps;
        if(cur==tar)break;
        for(int i = 0;i < 4;i++)
        {
            int dx = cur.xx + Next[i][0];//下一个点坐标
            int dy = cur.yy + Next[i][1];
            if(dx >= 0 && dx < ROW_NUM && dy >=0 && dy < COL_NUM &&
((agent_world[dx][dy] & VISITED) == VISITED )&& vis[dx][dy] == false)
            {
```

```
                if(record == true)//记录路径
                {
                    bfs_path[Point{dx,dy}] = cur;
                    vis[dx][dy] = true;
                    Q.push({dx,dy});
                }
            }
        }
    }
    if(record == true)// 需要记录则根据搜索路径找到搜索结果
    {
        GetBfsPath(tar,src,tar,bfs_path);
    }
    if(flag)agent_world[tar.xx][tar.yy] &= ~VISITED;// 取消访问标记
    return steps;// 返回最短路径长度
}
```

2、联机宽度优先搜索

```
void Widget::DFS(Point current)
{
    vector<Point>neighbors;
    path_record[step_cnt++] = current;
    agent_world[current.xx][current.yy]                                    |=
    real_world[current.xx][current.yy];//获取当前点信息
    if((real_world[current.xx][current.yy]  &  GOLD)  ==  GOLD){find_gold  =
    true;return;}
    if((real_world[current.xx][current.yy]  &  PIT)  ==  PIT){game_over  =
    true;return;}
    PutFlag(agent_world,current,VISITED);
    UpdateNeighborsInformation(current);
    agent_world[current.xx][current.yy] &= ~CURRENT;
    GetNeighborPosition(current,neighbors);
    for(int i =0; i <neighbors.size(); i++)//周围可能有金子
```

```
{
    if((agent_world[neighbors[i].xx][neighbors[i].yy] & GOLD) == GOLD)
    {
        PutFlag(agent_world,neighbors[i],CURRENT);
        DFS(neighbors[i]);
        if(find_gold || game_over)return;
    }
}
//周围没有金子
if(find_gold == false)
{
    vector<Point>safe_place;
    for(int i =0; i <neighbors.size(); i++)
    {
        if (((agent_world[neighbors[i].xx][neighbors[i].yy] & SAFE) == SAFE)
        // 安全
                &&    ((agent_world[neighbors[i].xx][neighbors[i].yy]    &
VISITED)  == 0)) { // 未访问
            safe_place.push_back(neighbors[i]);
        }
    }
    if(safe_place.size()>0)//存在安全区域
    {
        int rand_next_pos = rand()%safe_place.size();//随机选择一个
        PutFlag(agent_world,safe_place[rand_next_pos],CURRENT);
        DFS(safe_place[rand_next_pos]);
        if(find_gold || game_over)return;
    }
    else
    {
        Point nearest_safe_pos = {-1,-1};
        for(int i =0; i<ROW_NUM; i++)
        {
            for(int j =0; j <COL_NUM; j++)
            {
                if((agent_world[i][j] & SAFE) == SAFE)
                {
                    if(nearest_safe_pos  == Point{-1,-1})nearest_safe_pos =
{i,j};
```

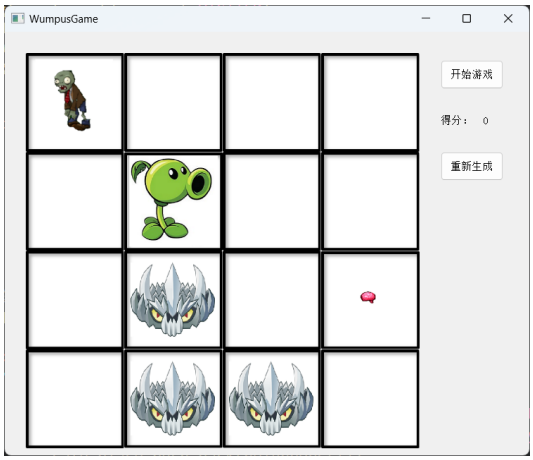
```
                    else{
                        int dismin = BFS(current,nearest_safe_pos,false);//
最近安全点距离
                        int discur = BFS(current,Point{i,j},false);//当前安
全点距离
                        if(discur < dismin)nearest_safe_pos = {i,j};
                    }
                }
            }
        }
        //机器人地图上有安全点
        if(nearest_safe_pos != Point{-1,-1})
        {
            PutFlag(agent_world, nearest_safe_pos, CURRENT);
            BFS(current, nearest_safe_pos, true);   // 记录行动路线
            DFS(nearest_safe_pos);  // 下一步
            if (find_gold || game_over) return;
        }
        else//没有安全点就选择一个相对安全的区域
        {
            vector<Point> good_neighbor;
            for (int i = 0; i < neighbors.size(); ++i)
            {
                if(((agent_world[neighbors[i].xx][neighbors[i].yy]        &
VISITED)  == 0) // 未被访问
                        &&  ((agent_world[neighbors[i].xx][neighbors[i].yy]
& PIT) == 0)    // 没有陷阱
                        &&  ((agent_world[neighbors[i].xx][neighbors[i].yy]
& WUMPUS) == 0)) // 没有怪兽
                    good_neighbor.push_back(neighbors[i]);
            }

            if (good_neighbor.size() > 0) { // 有相对安全的点
                int  rand_next_goodpos  =  rand()  %  (good_neighbor.size());
    // 随机选取一个相对安全的点
                DFS(good_neighbor[rand_next_goodpos]);  // 下一步
                if (find_gold || game_over) return;
            }else { // 没有相对安全的点，选择杀死怪兽
                vector<Point> kill_pos, pit_pos;
```

```
                for (int i = 0; i < neighbors.size(); ++i) {
                    if  ((agent_world[neighbors[i].xx][neighbors[i].yy]  &
VISITED) == 0) // 未被访问过
                        if((agent_world[neighbors[i].xx][neighbors[i].yy] &
WUMPUS) == WUMPUS) {    // 是怪兽
                            kill_pos.push_back(neighbors[i]);
                        }else {
                            pit_pos.push_back(neighbors[i]);
                        }
                }
                if(kill_pos.size() > 0)//怪兽存在则杀死
                {
                    int kill_wumpus = rand() % kill_pos.size();
                    vector<Point>wumpus_neighbors;

GetNeighborPosition(kill_pos[kill_wumpus],wumpus_neighbors);

agent_world[kill_pos[kill_wumpus].xx][kill_pos[kill_wumpus].yy]          &=
~WUMPUS;//取消怪兽标记
                    for(int i = 0; i < wumpus_neighbors.size(); i++)//取消
臭气标记
                    {

agent_world[wumpus_neighbors[i].xx][wumpus_neighbors[i].yy] &= ~STENCH;
                    }
                    PutFlag(agent_world,kill_pos[kill_wumpus],CURRENT);
                    DFS(kill_pos[kill_wumpus]);//下一步
                    if(find_gold || game_over)
                        return;
                }
                else if(pit_pos.size())
                {
                    int jump_pit = rand() % pit_pos.size();
                    PutFlag(agent_world,pit_pos[jump_pit],CURRENT);
                    DFS(pit_pos[jump_pit]);
                }
            }
        }
```

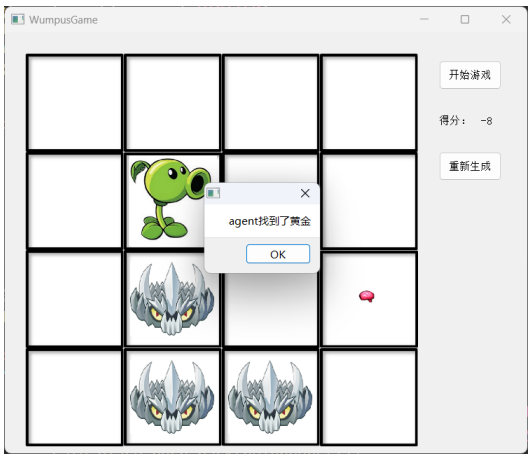
```
            }
        }
}
```

五、运行结果及分析

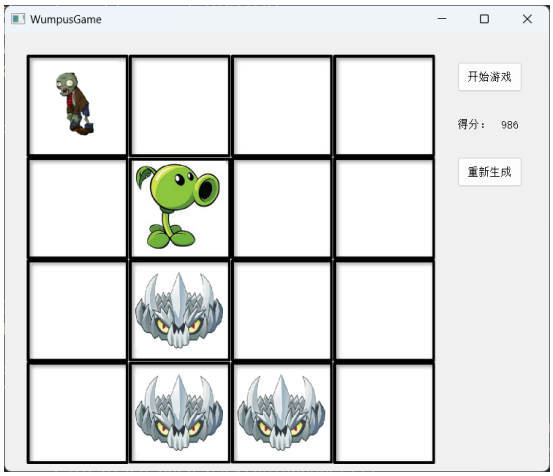
1、运行结果



初始化



找到黄金



返回起点

2、分析

在一些简单的情况下，比如金子附近没什么陷阱，金子离 Agent 也比较近，Agent 可以很容易找到金子并返回，但在一些复杂情况下，比如金子旁有陷阱和 Wumpus，那么 Wgent 的死亡概率会大大提高，因为此时 Wgent 是随机选择移动方向，如果没有最好的方向的话。

六、小结

在本次实习中，我要实现一个基于知识的 Agent 来解决 Wumpus 世界的问题，并通过定义各种感知和行动规则来描述 Agent 的行为。通过实现联机搜索的算法，我的 Agent 有概率成功找到黄金并成功离开了洞穴。通过本次实习，我了解了联机搜索的算法思想、具体实现以及应用到实际问题上，我对人工智能的应用和发展有了更深入的了解和认识。

七、参考文献

使用联机搜索求解 Wumpus World_xiongyuqing 的博客-CSDN 博客

实习四 采用 α－β 剪枝算法实现井字棋游戏

一、题目描述

采用 α－β 剪枝算法实现井字棋游戏

要求：图形化界面，选取先手后手

二、算法思想

α－β 剪枝算法是一种优化的极小极大算法，用于搜索博弈树的最佳决策。该算法通过对搜索树中的节点进行剪枝来减少搜索时间。具体来说，该算法在搜索树中使用两个参数 α 和 β，它们表示已经找到的最大值和最小值。在搜索过程中，如果当前节点的值已经超出了 α 和 β 的范围，则可以剪去该节点的子树，因为不论该子树的其他节点的值是什么，它们都不可能对当前节点的最优值产生影响。

三、软件设计

1、软件说明

语言：C++

环境：QT 5.12.12

2、需求分析

（1）用户界面：需要一个用户界面，以便玩家能够在屏幕上看到井字棋游戏的棋盘和移动情况。

（2）游戏规则：需要遵循井字棋的规则。

（3）算法实现：需要实现 α－β 剪枝算法。

（4）先手后手选择：需要允许玩家选择先手或后手。

（5）游戏结束判断：需要实现游戏结束的判断。

3、总体设计

（1）设计井字棋游戏的图形化界面，包括棋盘、棋子、先手后手选择按钮等。

（2）设计井字棋的游戏逻辑，包括初始化棋盘、下棋、判断胜负等。

（3）实现 α－β 剪枝算法，通过搜索博弈树找到最优决策。

（4）将搜索结果与当前棋盘上的局面相结合，得到下一步最优的落子位置。

4、详细设计

1. 图形化界面设计：
    - 设计棋盘：使用网格布局实现一个 3x3 的棋盘，每个网格代表一个棋子位置。
    - 设计棋子：使用圆形或叉形图片表示棋子，通过鼠标点击实现下棋操作。
    - 设计按钮：提供先手后手选择按钮，以及重新开始游戏的按钮。
2. 游戏逻辑设计：
    - 初始化棋盘：将所有棋子位置设置为空。
    - 下棋：根据当前玩家的选择，在空白的棋子位置上放置该玩家对应的棋子。
    - 判断胜负：在每次下棋后，检查当前玩家是否获胜，如果是则宣布胜利并结束游戏。
3. α－β 剪枝算法设计：

- 构造博弈树：以当前局面为根节点，将所有可能的下一步棋子位置作为子节点，递归生成博弈树。
- 极小极大算法：对于叶子节点，根据当前局面和当前玩家计算叶子节点的价值，对于极大节点，选择价值最大的子节点，对于极小节点，选择价值最小的子节点。
- α-β 剪枝：在搜索过程中，维护两个参数 α 和 β，表示已经找到的最大值和最小值。如果当前节点的价值超出了 α 和 β 的范围，则可以剪去该节点的子树。

4. 下一步最优落子位置设计：
   - 将搜索得到的博弈树中的每个节点的价值与当前局面相结合，得到每个子节点的最终价值。
   - 对于当前玩家的极大节点，选择价值最大的子节点作为下一步的落子位置，对于对手的极小节点，选择价值最小的子节点作为对手的下一步落子位置。

四、关键代码

1、最大值最小值搜索

```
int Widget::MinMaxSearch(int depth)
{
    int value;//估值
    int bestValue = 0;
    int moveCount = 0;
    int i;
    Pos PosList[9];//保存可以下子的位置
    //如果在深度未耗尽之前赢了
    if(isWin() == COM || isWin() == MAN)
    {
        return evaluateMap();//返回递归
    }
    //如果搜索深度耗尽
    if(depth == 0)
    {
        return evaluateMap();
```

```
    }
    //如果深度未耗尽并且都没赢
    if(COM == player){ bestValue = -MAX_NUM; }
    else if(MAN == player){ bestValue = MAX_NUM; }
    //深度未耗尽并且都没赢的情况下，电脑需要获取到棋盘中剩余的位置，并且找到某一个位置下子
    // 获取棋盘上一共还剩多少步
    //moveCount = getMoveList(PosList);
    for(int i = 0; i < COL; i++)
    {
        for(int j = 0; j < ROW; j++)
        {
            if(board[i][j] == 0)
            {
                PosList[moveCount].x = i;
                PosList[moveCount].y = j;
                moveCount++;
            }
        }

    }

    if(moveCount < 1)
    {
        bestPos = PosList[0];
        return bestValue;
    }
    // 遍历棋盘上剩余的每一步，找到最优点
    for(int i = 0; i < moveCount; i++)
    {
        // 拿到棋盘剩余棋格中的一个棋格
        Pos curPos = PosList[i];
        //假装下个子
        board[curPos.x][curPos.y] = player;
        player = (player == COM) ? MAN : COM;
        // 假装下子完成后，调用 miniMax。
        // 调用完成后，获取返回值 2
        value = MinMaxSearch(depth - 1);
        //把假下的棋子清空
```

```
            board[curPos.x][curPos.y] = 0;
            player = (player == COM) ? MAN : COM;
            if(player == COM)
            {
                if(value > bestValue)
                {
                    bestValue = value;
                    // 防止出现递归未完成时，也调用了最优点
                    // 当递归return到最初开启递归那层时，赋值最优点
                    if(depth == currentDepth)
                    {
                        bestPos = curPos;
                    }
                }
            }
            else if(player == MAN)
            {
                if(value < bestValue)
                {
                    bestValue = value;
                    if(depth == currentDepth)
                    {
                        bestPos = curPos;
                    }
                }
            }
        }
        return bestValue;
    }
```

2、选取下一步位置

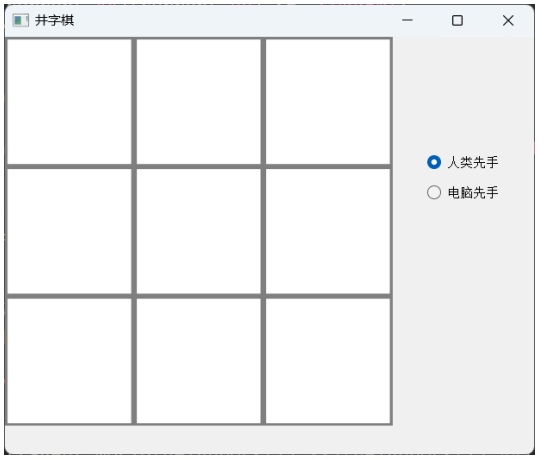
```
int Widget::evaluateMap()
{
    //如果计算机赢了，返回最大值
    if (isWin() == COM)
        return MAX_NUM;
    //如果计算机输了，返回最小值
    if (isWin() == MAN)
        return -MAX_NUM;
```

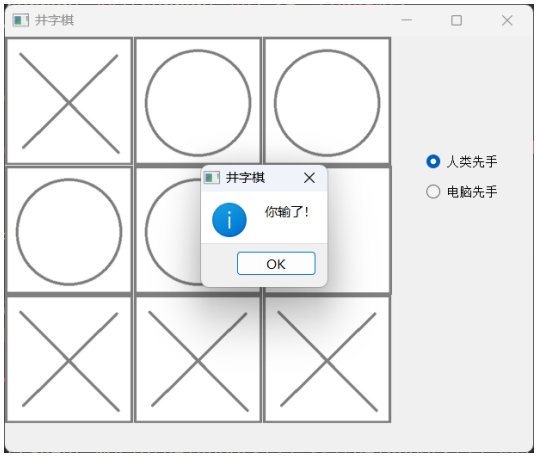
```
//该变量用来表示评估函数的值
int count = 0;
//将棋盘中的空格填满自己的棋子，既将棋盘数组中的 0 变为 1
for ( int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (board[i][j] == 0)
            tempBoard[i][j] = COM;
        else
            tempBoard[i][j] = board[i][j];
    }
}
//电脑方
//计算每一行中有多少行的棋子连成 3 个的
for (int i = 0; i < 3; i++)
    count += (tempBoard[i][0] + tempBoard[i][1] + tempBoard[i][2]) / 3;
for (int i = 0; i < 3; i++)
    count += (tempBoard[0][i] + tempBoard[1][i] + tempBoard[2][i]) / 3;
count += (tempBoard[0][0] + tempBoard[1][1] + tempBoard[2][2]) / 3;
count += (tempBoard[2][0] + tempBoard[1][1] + tempBoard[0][2]) / 3;
//将棋盘中的空格填满对方的棋子，既将棋盘数组中的 0 变为-1
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (board[i][j] == 0)
            tempBoard[i][j] = MAN;
        else tempBoard[i][j] = board[i][j];
    }
}
    //计算每一行中有多少行的棋子连成 3 个的
for (int i = 0; i < 3; i++)
    count += (tempBoard[i][0] + tempBoard[i][1] + tempBoard[i][2]) / 3;
for (int i = 0; i < 3; i++)
    count += (tempBoard[0][i] + tempBoard[1][i] + tempBoard[2][i]) / 3;
count += (tempBoard[0][0] + tempBoard[1][1] + tempBoard[2][2]) / 3;
count += (tempBoard[2][0] + tempBoard[1][1] + tempBoard[0][2]) / 3;
```

```
    // 返回的数因为包括了负数和整数，所以不会太大
    return count;
}
```

五、运行结果及分析

1、运行结果





2、分析

经过实现，使用 α－β 剪枝算法的井字棋游戏可以瞬间计算出最优的落子位置，而且在搜索过程中进行了剪枝，减少了搜索时间，提高了搜索效率。通过图形化界面的设计，用户可以方便地进行下棋操作，选择先手后手，观察游戏进程。然而，由于井字棋游戏比较简单，博弈树的规模较小，因此 α－β 剪枝算法在此应用场景下的优势不太明显。

六、小结

在本次实习中，我学习了 α－β 剪枝算法的原理和应用，学会了通过递归生成博弈树来搜索最优决策的方法。通过实现井字棋游戏，我对人工智能算法的应用和开发有了更深入的了解和认识，同时也锻炼了自己的编程能力和解决问题的能力。

七、参考文献

采用 α-β 算法实现井字棋游戏_采用 α-β 剪枝算法实现井字棋游戏。 图形化界面。 随机选取先手后手。_xiongyuqing 的博客-CSDN 博客

实 习 评 语

评阅时间：

评阅记录

实习题								
成绩								
上机成绩：				实习报告成绩：				
成绩：				评阅人（签字）：				