五子棋 AI 实验报告

刘小江 Max1.0

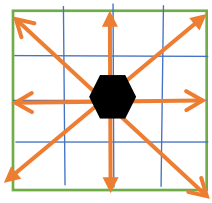
刘小江 Max1.0，是我耗时一周时间研制的 Bot，它并没有包含什么深刻高端的·算法，都是我单纯的理解，在这一周我几乎没有查阅资料，想看看自己白手起家能做到什么程度，这导致了我 AI 目前还不能轻易战胜我（在 Botzone 上赢过我一次），我也在寻找原因并且着手查找资料，优化我 AI 的决策。下面我开始我第一次的总结。

1.一个简单的想法

我最开始花了一些时间思考了我该从哪里开始我的 AI，最后发现了一些收获。在 Botzone 上，棋盘是一个 15*15 的数组，数组里不同的值表示了该位置不同的状态，从中我可以大致了解到，我设计的算法主要就是基于数组的操作，AI 通过对数组的分析，用算法计算出数组空位置中的最优坐标，并返回它。于是，问题就转化成为了空位置的搜索算法和分析算法上了。搜索算法，我的理解比较简单，可以通过两重 for 循环，搜索出棋盘的所有没有棋子的坐标，将它们压入 empty 数组中，并且一一去处理它们，找出算法认为的最优坐标并且返回最优解（下棋）。接下来就是分析算法了，我只需要去解决分析每个空坐标，并且求出最佳答案问题就能够解决了，我的初级 AI 就可以诞生。

2.实现算法分析

棋盘上没有落下棋子的点有很多很多，每一个空点都有可能和其他的任何坐标产生关系，形成更加复杂的情况，难以分析，即使把每一个空坐标和其他任何一个可能与它有关的点进行分析，我认为意义并不很大，而且大大加大了算法的复杂度，得不偿失。基于五子棋的规则，我只分析每一个空坐标的东南西北与东南，东北，西南，西北这八个方向（如图所示），而且分析距离只设置为 4，然后我根据这些方向的坐标信息进行分析，根据这些分析结果给出该空坐标的一个权值，最后把棋盘所有的空坐标分析完毕之后，选取权值最大的坐标返回（下棋）。好的问题来了，我该怎么计算权值呢，用什么好的公式呢？（此问题至今我还没有很好解决，准备打算查资料了，想不出来）目前我所设计的公式还不是很完善，遇到紧急情况（例如对方三连）有时还不一定能做出正确的反应。我会继续优化。



我的具体想法是这样的，在每一个空坐标的八个方向上都搜索一次，每一个方向都分为左边和右边，每一边连续棋子越多 temp 变量的值越高，但是每一边出现另一种棋子时（棋子被堵住）tempL 或则 tempR 就会清零，因为此时再也无法在此区域上构成 5 连（不会赢，也不会输），所以将它们重新计算，当搜索时遇到空棋时，temp--，这样来说，不管是敌方连续子多还是己方连续子多，都会有比较高的权值，这符合基本的逻辑；最终将这 8 个方向的 temp 累加，每种 temp 就是一种权值，最终暴力地取最大的权值下棋，完成

判断。这就是我的 AI1.0 的基本思路，还是有较大的缺陷，需要加以改进。

3.具体的实现

具体的实现如下，其中包括源代码和 Botzone 上的 Bot 以及对战截图：

1.总的布局：

```
#include ...  // C++编译时默认包含此库

using namespace std;

const int SIZE = 15;
int Grid[SIZE][SIZE];

//坐标点
struct point { ... };
//先手方
void placeAt1(int x, int y) { ... }
//后手方
void placeAt2(int x, int y) { ... }
Json::Value AI() { ... }
void start() { ... }
int main() { ... }
```

2.核心代码（中间隐藏了部分代码）：

```
Json::Value AI()
{
    vector<point> empty;
    for (int i = 0; i < SIZE; ++i) { ... }
    //srand(time(0));
    //int rand_pos = rand() % xs.size();
    int ai_pos = 0;
    int mode = 0, cur = 0, curX = 0, curY = 0, sum1 = 0, sum2 = 0;
    int cPrior = 0, prior = 0, flgL = 0, flgR = 0, tx = 0, ty = 0;
    for (unsigned int i = 0; i < empty.size(); i++)
    {
        prior = 0; tx = empty[i].x; ty = empty[i].y;
        for (mode = 0; mode < 4; mode++)                    // 模式1为 — 向判断，模式2为 | 向判断
        {                                                   // 模式3为 \ 向判断，模式4为 / 向判断
            sum1 = sum2 = 0;
            flgL = flgR = 0;
            curX = empty[i].x;
            curY = empty[i].y;                              //改变模式后重置变量
            for (cur = -4; cur <= 4; cur++) { ... }
                                                            // 长连相对短连拥有绝对的优势
            if (sum1 == 1) prior = prior + 1;
            if (sum1 == 2) prior = prior + 10 + 2 * flgL;   // 数目相同敌方棋优先
            if (sum1 == 3) prior = prior + 100 + 20 * flgL;
            if (sum1 == 4) prior = prior + 1000 + 200 * flgL;
            if (sum2 == 1) prior = prior + 1;
            if (sum2 == 2) prior = prior + 10 + 2 * flgR;
            if (sum2 == 3) prior = prior + 100 + 20 * flgR;
            if (sum2 == 4) prior = prior + 1000 + 200 * flgR;
        }
        if (prior > cPrior) { ... }

        if (prior == cPrior && rand() < RAND_MAX / 5) { ... }
    }
    Json::Value action;
    action["x"] = empty[ai_pos].x;
    action["y"] = empty[ai_pos].y;
    return action;
}
```

8 个方向搜索代码（部分）

```
for (cur = -4; cur <= 4; cur++)
{
    if (mode == 1) //向→检查
    {
        curX = tx + cur;
        if (curX < 0) continue; //左越界继续
        if (curX > 14) break;   //右越界停止
    }
    if (mode == 2) //向↓检查
    {
        curY = ty + cur;
        if (curY < 0) continue; //上越界继续
        if (curY > 14) break;   //下越界停止
    }
    if (mode == 3) //向↘检查
    {
        curX = tx + cur;
        curY = ty + cur;
        if (curX < 0 || curY < 0) continue; // 左或上越界继续
        if (curX > 14 || curY > 14) break;  // 右或下越界停止
    }
    if (mode == 4) //向↗检查
    {
        curX = tx + cur;
        curY = ty - cur;
```

4.总结与不足

此算法具有一定的对抗性，代码量也比较精简，具有相对过得去的棋力（赢过我一次）。但是，这个 AI 还有相当的缺陷，比如算法在判断是会出现较为严重的错误，比如有时候不能够对敌方的三连棋子做出正确地回应，导致对战失败。我对此做出了一些分析，我认为这可能是由于我写的判断函数过于简单暴力，只是简单的将八个方向的权值做一个简单的线性累加，导致了 3+3>1+5 的情况发生。我在今后的代码中会努力加上特判代码，当算法搜索到并且判断出紧急情况时终止循环，堵住缺口，这保证代码能够正确回应攻击，但是这又有可能错误地破解活双三的情况，我会继续考虑。在下一次的“刘小江 Max2.0”中，我会查找一些五子棋几种较为典型的棋型资料，将其中的一些设置为宏，在搜索时进行特判，并且细化搜索判断的条件，使得 AI 的反应更加精确。以上就是我对刘小江 Max1.0 的总结和展望。

以下是刘小江 Max1.0 的源代码：

```
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib>
#include "jsoncpp/json.h" // C++编译时默认包含此库

using namespace std;

const int SIZE = 15;
int Grid[SIZE][SIZE];

//坐标点
struct point
{
    int x, y;
};
//先手方
void placeAt1(int x, int y)
{
    if (x >= 0 && y >= 0) {
        Grid[x][y] = 1;
    }
}
//后手方
void placeAt2(int x, int y)
{
    if (x >= 0 && y >= 0) {
        Grid[x][y] = -1;
    }
}
Json::Value AI()
{
    vector<point> empty;
    for (int i = 0; i < SIZE; ++i)
    {
        for (int j = 0; j < SIZE; ++j)
        {
            if (Grid[i][j] == 0)
            {
                point temp; temp.x = i; temp.y = j;
                empty.push_back(temp);
```

```
            }
        }
    }
    //srand(time(0));
    //int rand_pos = rand() % xs.size();
    int ai_pos = 0;


    int mode = 0, cur = 0, curX = 0, curY = 0, sum1 = 0, sum2 = 0;
    int cPrior = 0, prior = 0, flgL = 0, flgR = 0, tx = 0, ty = 0;

    for (unsigned int i = 0; i < empty.size(); i++)
    {
        prior = 0; tx = empty[i].x; ty = empty[i].y;
        for (mode = 0; mode < 4; mode++)   // 模式 1 为 — 向判断，模式 2 为 | 向判断
        {                                  // 模式 3 为 \ 向判断，模式 4 为 / 向判断
            sum1 = sum2 = 0;
            flgL = flgR = 0;
            curX = empty[i].x;
            curY = empty[i].y;             //改变模式后重置变量
            for (cur = -4; cur <= 4; cur++)
            {
                if (mode == 1) //向→检查
                {
                    curX = tx + cur;
                    if (curX < 0) continue; //左越界继续
                    if (curX > 14) break;   //右越界停止
                }
                if (mode == 2) //向↓检查
                {
                    curY = ty + cur;
                    if (curY < 0) continue; //上越界继续
                    if (curY > 14) break;   //下越界停止
                }
                if (mode == 3) //向↘检查
                {
                    curX = tx + cur;
                    curY = ty + cur;
                    if (curX < 0 || curY < 0) continue; // 左或上越界继续
                    if (curX > 14 || curY > 14) break;  // 右或下越界停止
                }
                if (mode == 4) //向↗检查
```

```
{
    curX = tx + cur;
    curY = ty - cur;
    if (curX < 0 || curY>14) continue; // 左或下越界继续
    if (curX > 14 || curY < 0) break;  // 右或上越界停止
}

// 初始化棋子标志，用于判断反色
if (cur < 0 && flgL == 0 && Grid[curX][curY] != 0)
    flgL = Grid[curX][curY];

if (cur > 0 && flgR == 0 && Grid[curX][curY] != 0)
    flgR = Grid[curX][curY];

if (cur < 0 && Grid[curX][curY] == (-flgL))
{
    sum1 = -1;      // 左侧搜索到反色，相当于被堵住的棋
    flgL = -flgL;   // 同时标记也取反
}

if (cur < 0 && Grid[curX][curY] == flgL)
{
    sum1++; // 左侧搜索到同色
}

if (cur == -1 && sum1 > 0 && Grid[curX][curY] == 0)
{
    sum1--; // 左一位搜索到空棋
}
if (cur == 1 && sum2 > 0 && Grid[curX][curY] == 0)
{
    sum2--; // 右一位搜索到空棋
}

if (cur > 0 && flgL == flgR) // 如果左边的棋子和右边的同色
{
    sum2 += sum1; // 用 sum2 代替 sum1 继续搜索
    sum1 = 0;     // sum1 置 0 以防重复相加和影响后面的 cPrior
}

if (cur > 0 && Grid[curX][curY] == (-flgR))
{
    sum2--; // 右边出现反色则，相当于被堵住的棋
    break;   // 不需要继续搜索了
```

```
                }

                if (cur > 0 && Grid[curX][curY] == flgR)
                {
                    sum2++;// 右侧搜索到同色
                }

            }
            // 长连相对短连拥有绝对的优势
            if (sum1 == 1) prior = prior + 1;
            if (sum1 == 2) prior = prior + 10 + 2 * flgL; // 数目相同白棋优先
            if (sum1 == 3) prior = prior + 100 + 20 * flgL;
            if (sum1 == 4) prior = prior + 1000 + 200 * flgL;
            if (sum2 == 1) prior = prior + 1;
            if (sum2 == 2) prior = prior + 10 + 2 * flgR;
            if (sum2 == 3) prior = prior + 100 + 20 * flgR;
            if (sum2 == 4) prior = prior + 1000 + 200 * flgR;
        }

        if (prior > cPrior)
        {
            cPrior = prior;
            ai_pos = i;
        }

        if (prior == cPrior && rand() < RAND_MAX / 5)
        {   // 除第一个棋外很少有优先级相等的点
            cPrior = prior;
            ai_pos = i;
        }
    }
    Json::Value action;
    action["x"] = empty[ai_pos].x;
    action["y"] = empty[ai_pos].y;
    return action;
}
void start()
{
    // 读入 JSON
    string str;
    getline(cin, str);
    Json::Reader reader;
    Json::Value input;
    reader.parse(str, input);
```

```
    // 分析自己收到的输入和自己过往的输出，并恢复状态
    int turnID = input["responses"].size();
    for (int i = 0; i < turnID; i++) {
        placeAt1(input["requests"][i]["x"].asInt(),
input["requests"][i]["y"].asInt());
        placeAt2(input["responses"][i]["x"].asInt(),
input["responses"][i]["y"].asInt());
    }
    placeAt1(input["requests"][turnID]["x"].asInt(),
input["requests"][turnID]["y"].asInt());
    // 做出决策存为 myAction
    // 输出决策 JSON
    Json::Value ret;
    ret["response"] = AI();
    Json::FastWriter writer;
    cout << writer.write(ret) << endl;
}
int main()
{
    start();
    return 0;
}
```