Report on GoMoku in stage I

Presented by 61516413 Y.Bin & 61516411 M.Lingtao

Current Version 1.0

2018/3/19

Summary

We have achieved:

1. Analyzed the problem and searched for the literature
2. Conceived and implemented the basic algorithm of our GoMoku program
3. Deployed our robot on the botzone, currently ranked tenth (March 18)

1. Problem Analysis

1.1 Introduction

GoMoku, also called Gobang or Five in a Row, is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board, using 15×15 of the 19×19 grid intersections.

Players alternate turns placing a stone of their color on an empty intersection. The winner is the first player to form an unbroken chain of five stones horizontally, vertically, or diagonally.

1.2 Background

After consulting literature, we found that there are several common algorithms for GoMoku

1) Scoring method

   Each time the player takes a step, the player and the computer score each of the empty positions on the board according to the winning table. The score for each position is related to the following sentence: The number of pieces already owned in each winning combination where the position is located.

   Then compare the scores generated by the player and the computer to determine whether the computer is offensive or defensive.

2) Mode method

   Set up certain types of chess that can be won in Gomoku and store them. Each operation simulates the game in storage.

1.3 Our solution

We fully analyzed the feasibility of the two options and decided to use the first type of solution. Because the second option is more aggressive, but it ignores defensiveness.

2. Program and Design

Our Idea:

Traversing every empty position on the board before each step, if bot places a chess piece in this space, determine the number of pieces in the space that are affected by this vacancy, score this position according to this information. Then judge how much the score is when the opponent falls to this point. Finally compare all scores, if our maximum score is greater than the opponent's maximum score, it will fall at this point. Otherwise, it will fall at the point where the opponent get maximum score.

The code is attached in the appendix.

3. Where to improve

1) Some five-tuples are calculated multiple times when calculating each point score, it can be optimized to reduce runtime.
2) Score function for each piece can be optimized
3) Due to the nature of the algorithm itself, the aggressiveness of the robot is not enough, it is more defensive.

Appendix

```
#include<iostream>
#include<sstream>
#include<vector>
#include"jsoncpp/json.h"

using namespace std;


struct mark {
        int x;
        int y;
        long score;
        mark() :x(-1), y(-1), score(-1) {};
        mark(int m, int n, int s) :x(m), y(n), score(s) {}

        bool operator<=(const mark&mark2) {
                return this->score <= mark2.score;
        }
        bool operator>=(const mark&mark2) {
                return this->score >= mark2.score;
        }
        bool operator<(const mark&mark2) {
                return this->score < mark2.score;
        }
        bool operator>(const mark&mark2) {
                return this->score > mark2.score;
        }
        bool operator==(const mark&mark2) {
                return this->score == mark2.score;
        }
};

int directions[4][2]{ { 1,0 },{ 0,1 },{ 1,1 },{ 1,-1 } };

class gobang {
	int chalbox[15][15];
public:
	long marklist[5];
	gobang() {
		for (int i = 0; i < 15; i++) {
			for (int j = 0; j < 15; j++) {
				chalbox[i][j] = 0;
			}
		}
		for (int i = 0; i < 5; i++) {
			if (i == 0)marklist[i] = 1;
			else {
				int total = 0;
				for (int j = 0; j < i; j++) {
					marklist[i] = i > 0 ? 4 * marklist[i - 1] : 1;
				}
			}
		}
	}

	mark countScore(int x, int y, int who) {
		mark total;
		total.x = x;
		total.y = y;
		total.score = 0;

		/*for (int i = 0; i < 5; i++) {
		m.score += countLevel(x, y, i + 1, who);
		}*/
		this->chalbox[x][y] = who;
		for (int m = 0; m < 4; m++) {
			for (int i = 0; i < 5; i++) {
				int count = 0;
				bool flag = true;
				for (int j = 0; j < 5; j++) {
					if (x - (i - j) * directions[m][0] >= 0 && y - (i - j) * directions[m][1] >= 0
&& x - (i - j) * directions[m][0] < 15 && y - (i - j) * directions[m][1] < 15 && chalbox[x - (i - j) *
directions[m][0]][y - (i - j) * directions[m][1]] == who)
					{
						count++;
					}
					else if (x - (i - j) * directions[m][0] >= 0 && y - (i - j) * directions[m][1] >=
0 && x - (i - j) * directions[m][0] < 15 && y - (i - j) * directions[m][1] < 15 && chalbox[x - (i - j) *
directions[m][0]][y - (i - j) * directions[m][1]] == 0) {
                    }
                    else {
                        flag = false; break;
                    }
                }
                if (flag) {
                    total.score += count == 4 && who == 1 ? marklist[count - 1] +
marklist[count - 1] / 3 : marklist[count - 1];

                    if (count == 5 && who == 1)total.score = total.score * 2;
                }

            }
        }
        this->chalbox[x][y] = 0;
        return total;
        //return mark(x,y, 7857 * countLevel(x, y, 5,who) + 4152 * countLevel(x, y, 4,who) + 41 *
countLevel(x, y, 3,who) + 5 * countLevel(x, y, 2,who) + countLevel(x, y, 1,who));
    }

    mark ajust(int who=1) {
        mark myMaxScore, hisMaxScore, temp;
        vector<mark> maxMark;
        int pos;
        for (int i = 0; i < 15; i++) {
            for (int j = 0; j < 15; j++) {
                if (chalbox[i][j] == 0) {
                    temp = countScore(i, j, who);
                    if (maxMark.empty())
                    {
                        maxMark.push_back(temp);
                    }
                    else {
                        if (maxMark[0] < temp) {
                            maxMark.clear();
                            maxMark.push_back(temp);
                        }
                        else if (maxMark[0] == temp) {
                            maxMark.push_back(temp);
                        }
                    }
                    temp = countScore(i, j, -who);
                    if (maxMark[0] < temp) {
                        maxMark.clear();
                        maxMark.push_back(temp);
                    }
                    else if (maxMark[0] == temp) {
                        maxMark.push_back(temp);
                    }
                }
            }
        }
        if (who == -1) {
            pos = rand() % maxMark.size();
            return maxMark[pos];
        }

        vector<mark> maxScore2;
        mark least;
        for (auto per:maxMark) {
            this->Idown(per.x,per.y);
            temp = this->ajust(-1);
            if (maxScore2.empty()) {
                maxScore2.push_back(per);
                least = temp;
            }
            else if (temp < least) {
                maxScore2.clear();
                maxScore2.push_back(per);
                least = temp;
            }
            else if (temp ==least) {
                maxScore2.push_back(per);
            }
            chalbox[per.x][per.y] = 0;
        }
        pos = rand() % maxScore2.size();
        return maxScore2[pos];

    }
    void hedown(int x, int y) {
        if (x >= 0 && y >= 0 && x<15 && y<15)
            chalbox[x][y] = -1;
    }
    void Idown(int x, int y) {
        if (x >= 0 && y >= 0 && x<15 && y<15)
            chalbox[x][y] = 1;
    }
```

```
};

int main() {
	srand(time(0));
	// 读入 JSON
	string str;
	getline(cin, str);
	Json::Reader reader;
	Json::Value input;
	reader.parse(str, input);

	gobang go;
	// 分析自己收到的输入和自己过往的输出，并恢复状态
	string data = input["data"].asString(); // 该对局中，以前该 Bot 运行时存储的信息
	int turnID = input["responses"].size();
	int x, y;
	for (int i = 0; i < turnID; i++)
	{
		go.hedown(input["requests"][i]["x"].asInt(), input["requests"][i]["y"].asInt());
		go.Idown(input["responses"][i]["x"].asInt(), input["responses"][i]["y"].asInt());
	}

	// 看看自己本回合输入
	go.hedown(input["requests"][turnID]["x"].asInt(), input["requests"][turnID]["y"].asInt());
	mark res = go.ajust();
	Json::Value res2;
	res2["x"] = res.x;
	res2["y"] = res.y;
	Json::Value ret;
	ret["response"] = res2;
	Json::FastWriter writer;
	cout << writer.write(ret) << endl;
	return 0;
	return 0;
}
```