基于博弈树 Negamax 搜索、$\alpha-\beta$剪枝、启发式搜索的井字棋与五子棋 AI

实验简介

自从 AlphaGo、AlphaZero、AlphaZeroGo 的问世，从蒙特卡洛树搜索到强化学习，棋类 AI 越来越厉害。本次实验使用了博弈树上的极大极小值搜索、negamax 搜索、$\alpha-\beta$剪枝和启发式搜索，实现了两个简单的棋类 AI——井字棋和五子棋。

实验目的

1. 掌握极大极小值搜索算法
2. 掌握 Negamax 搜索算法
3. 掌握$\alpha-\beta$剪枝算法
4. 熟悉使用 React.js 与 Electron 发布跨平台应用

实验相关原理与技术

算法方面

1. 极大极小值搜索算法 Minimax

极大极小值算法是双人博弈中的常用算法，主要思想是一方要在可选的选项中选择将其优势最大化的选择，另一方则选择令对手优势最小化的方法。具体是实现是对博弈树进行深度优先搜索，评估遍历到的节点的分数，在搜索过程中使自己的分数和对手分数之差最大化，以到达寻找形成对自己最优、对对手最劣的走法。

下面用井字棋的博弈树详细说明。井子棋的搜索树如下图所示。假设人类玩家先手，AI 后手，空棋盘是第 0 层，那么博弈树的偶数层代表人类棋手的落子，奇数层代表 AI 的落子，搜索的时候，在走到叶子结点时，使奇数层局面评估与偶数层局面评估之差最大。

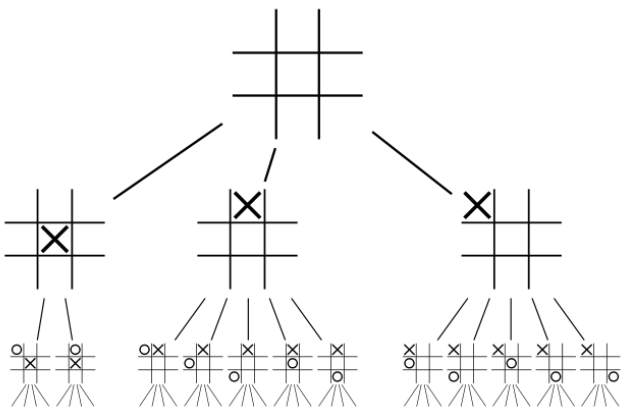

Figure 1 井字棋的博弈树

考虑井字棋的最大状态数不超过$9! = 362880$，搜索深度可以不限定，直到输/赢/平局位置停止，因此评估函数可以设置为：

$$evaluation(chessboard) = \begin{cases} -1, & player\ wins \\ 1, & AI\ wins \\ 0, & else \end{cases}$$

具体代码如下。遍历棋盘找空位，**假设在此处落子**，预测对方落子位置并**获得评估分数**，**回溯**。其中 `ai_move` 相当于 *max*（最大化自己的评估分数）的过程，`player_move` 相当于 *min*（最小化对方的评估分数）的过程。

```
const ai_move = (chessboard) => {
    let ret = { score: -INF, best_move: [] }, temp_score;

    let win_state = is_about_to_win(chessboard, AI);
    if (win_state.flag === true)
        ret = { score: AI, best_move: win_state.best_move };
    else if (is_full(chessboard))
        ret.score = TIE;
    else {
        for (let i = 0; i < 3; i ++)
            for (let j = 0; j < 3; j ++)
                if (chessboard[i][j] === 0) {
                    chessboard[i][j] = AI; // place here
                    temp_score = player_move(chessboard).score; // predict player move
                    chessboard[i][j] = 0; // rollback
                    if (temp_score > ret.score) { // update score and best move
                        ret.score = temp_score;
                    }
                }
    }

    return ret;
};


const player_move = (chessboard) => {
    let ret = { score: INF, best_move: [] }, temp_score;

    let win_state = is_about_to_win(chessboard, PLAYER);
    if (win_state.flag === true)
        ret = { score: PLAYER, best_move: win_state.best_move };
    else if (is_full(chessboard))
        ret.score = TIE;
    else {
        for (let i = 0; i < 3; i ++)
            for (let j = 0; j < 3; j ++)
                if (chessboard[i][j] === 0) {
                    chessboard[i][j] = PLAYER; // place here
                    temp_score = ai_move(chessboard).score; // predict AI move
                    chessboard[i][j] = 0; // rollback
                    if (temp_score < ret.score) { // update score and best move
                        ret.score = temp_score;
                        ret.best_move = [i, j];
                    }
                }
    }

    return ret;
};
```

2. $\alpha-\beta$ 剪枝算法 Alpha Beta Pruning

考虑五子棋的棋盘大小，博弈树就很大一棵了，暴搜是不现实的，所以必须剪枝。$\alpha-\beta$ 剪枝的主要思想是**减少搜索树的分枝**，将搜索时间用在**分差更大**的子树上，继而提升搜索深度。具体实现为：α代表已经评估过的己方局面的最大值， β 代表已经评估过的对方局面最小值。如果当前在 min 层且该局面分数大于已经评估过的己方局面的最小值，就没有必要再继续搜索这个节点了，因为显然不是全局最优。如果当前在 max 层且该局面分数小于已经评估过的己方局面的最大值，同理，也没有必要再继续搜索这个节点了。过程如下图所示。具体代码在 negamax 算法说明部分给出。

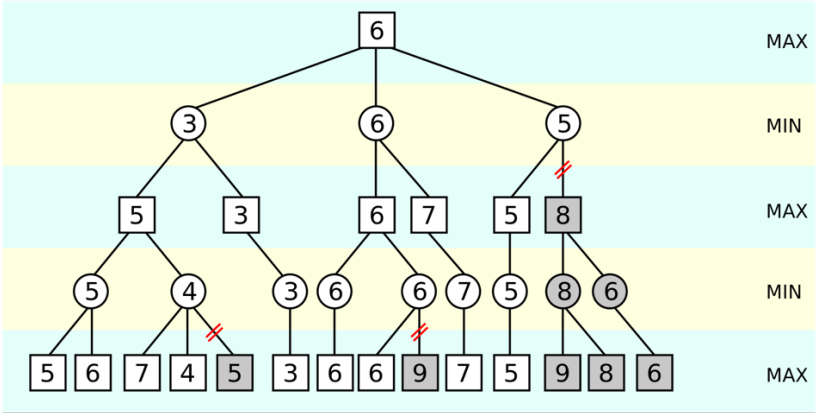

Figure 2 Alpha Beta 剪枝示例

3. 极大极小值算法的优化 Negamax

minimax 算法中，对自己评估获得的最大值和对对方评估的最小值，可以简化为：

$$\max(a,b) = -\min(-a,-b)$$

即可将博弈树转换为下图这种形式，与 minimax 并无二致，不过这样方便更新$\alpha-\beta$剪枝中的 alpha 和 beta 的值。

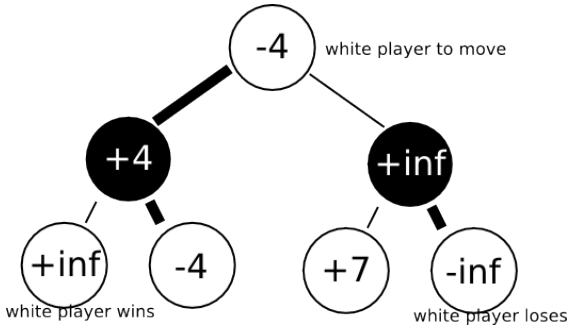

Figure 3negamax 生成的博弈树示例

所以我们可以把 minimax 中的两个函数集合成一个函数。如果是 AI 执子，则向评估结果大的子树前进；若果是人类执子，则向评估结果小的子树前进。配合 $\alpha-\beta$剪枝使用，代码如下所示。

```
const _negamax = (role, depth, alpha, beta) => {
    let e = new Evaluation(g_chessboard);
    if (depth <= 0 || win_check())
        return e.evaluate_chessboard(role);

    let search_list = e.generate_available_points(role);
    for (let i in search_list) {
        g_search_cnt ++;
        let x = search_list[i].point[0];
        let y = search_list[i].point[1];

        // place
        g_chessboard[x][y] = role;
        let val = -_negamax(-role, depth - 1, -beta, -alpha);
        // rollback
        g_chessboard[x][y] = 0;

        if (val > alpha) {
            if (depth === max_depth)
                g_next_move.push([x, y]);
             // alpha beta pruning
            if (val >= beta) {
                g_cut_cnt ++;
                return beta;
            }

            alpha = val;
        }
    }

    return alpha;
};
```

4. 启发式搜索算法 Heuristic search

我设置搜索深度是 3，即使加入了剪枝，搜索时间还是在 10s 上下。于是考虑用启发式搜索继续优化。其主要思想是通过指导搜索向最有希望的方向前进，降删除某些状态及其延伸，可以消除组合爆炸，并得到令人能接受的解，不过通常不一定是最优解。根据$\alpha-\beta$剪枝的过程我们可以发现，如果越早找到最大值，能剪的枝越多，所以节点的搜索顺序很重要。对于井子棋，我采用遍历棋盘找空位，对于五子棋可不能这样做了，否则搜到爆栈都搜不出来。具体实现是对棋盘每个点进行打分，**从分高的节点开始搜索**，可以大大减少搜索次数。具体代码见“实验方案与过程”。

实现方面

1. UI 界面 React.js

使 UI 组件化，方便快速开发 UI 界面。用 canvas 画棋盘，用 MVVM 的逻辑管理

2. Eelctron

使用 Electron 打包，构建跨平台的桌面应用。

实验环境

操作系统及硬件配置

编程语言

遵守 ES6 的 JavaScript

UI 界面搭建及应用发布

html5 + css3 + React.js + Electron

实验方案与过程

井字棋

1. 项目结构

```
├── src/TicTacToe
    ├── TicTacToe.css
    ├── TicTacToe.js
    └── minimax.js
```

其中 *TicTacToe.css* 控制页面样式，*TicTacToe.js* 控制游戏逻辑，*minimax.js* 包含了 minimax 和 evaluate 算法。

2. 算法流程图

其中，ai_move 和 player_move 的代码、evaluate 的公式已在“实验相关原理与技术”的“极大极小值搜索算法”部分给出，此处不再赘述。

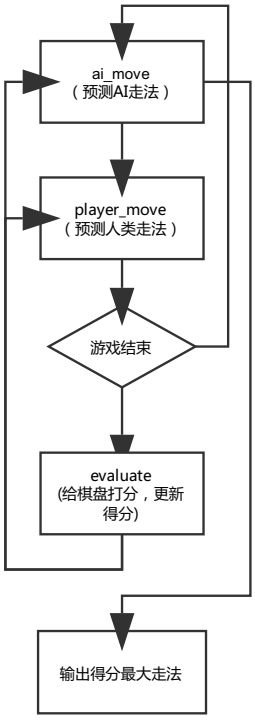

Figure 4 井字棋算法流程图s

五子棋

1. 项目结构

```
├── src/ GoBang
    ├── GoBang.css
    ├── GoBang.js
    ├── negamax.js
    └── evaluation.js
```

其中 GoBang.css 控制页面样式，*GoBang.js* 控制游戏逻辑，*negamax.js* 包含了使用 alpha 剪枝的 negamax 算法，evaluation.js 包含了棋盘的评估函数 evaluate 和启发式生成搜索点列表算法。

2. 算法流程图

（1） evaluate 棋盘评估

对棋盘打分，考察每一种棋形（活五、活四、活三、活二、冲四，冲三，冲二）的数量（统计数量就可以不用判断双三、双四、三四的情况），我设置的分数分别是[10000， 1000， 100， 10， 1000， 100， 10， 1]。期盘的最终得分是：

$$\text{evaluatoin(chessboard)} = \sum_{i=0}^{n} type[\text{i}].score \times type[i].cnt$$

在处理的时候使用了一个小技巧，就是把棋盘按行列展开成一位数组，然后写一个分析一位数组的棋型，就可以减少很多代码量。由于代码处理的细节

多、比较繁琐，就不在此处贴代码了，详细请见 evaluate.js。

（2） 启发式搜索生成搜索序列

详细思路在“实验相关原理与技术”的“启发式搜索”已经说明，这里说一下具体实现。给棋盘每个点打分的策略和给棋盘总体打分的策略一样，匹配不同的棋型，然后累加分数。打分先**选定角色**，先**寻找所有的空位**，让该角色在此处**落子**，**匹配棋形打分**，按照各个**空位得分排序**，选取分数大于某一阈值或者列表搜索列表数量小于某一阈值的那一部分返回，作为生成的搜索序列。这里的代码也比较冗长，详细请见 evaluate.js。

（3） negamax + alpha beta 剪枝

在“实验相关原理与技术”的“极大极小值算法的优化 negamax”部分给出，此处不再赘述。

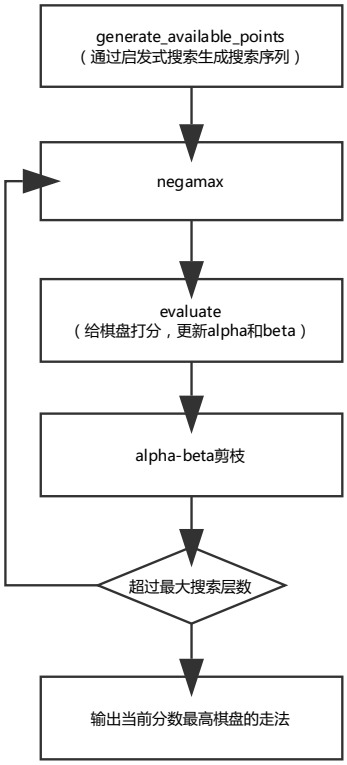

Figure 5 五子棋算法流程图

实验成果展示

1. 代码量统计

Language	files	blank	comment	code
JavaScript	9	291	94	1053
CSS	3	19	0	106
SUM:	12	310	94	1159

Figure 6 代码量统计

2. 界面展示

Figure 7 主页

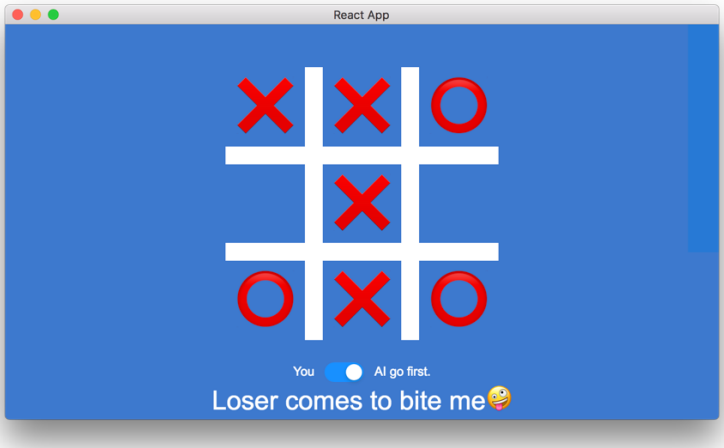

Figure 8 井字棋界

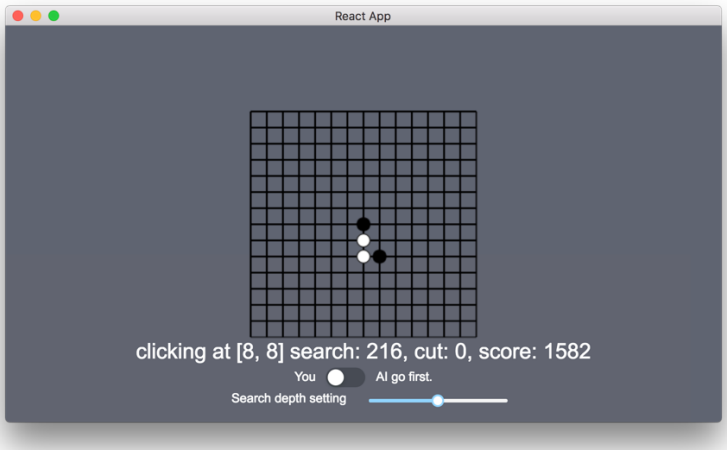

Figure 9 五子棋界面展示

3. 运行与安装说明

（1） Web 端

代码已部署到 gitpages 上：https://cjhahaha.github.io/ai-games/

Localhost：需要 node >= v10.6.0，npm >= 6.1.0。在项目目录下 `npm run start` 或者 `npm run build`。

（2） 桌面端

需要 `electron >= v1.4.13`。打包的配置文件已经写好，在项目目录下 `electron .` 或 `electron-packager . app --win --out build --overwrite --ignore=node_modules` 。

实验总结

本次实验从零开始编写界面、学习算法、调试、测试总共历时一周多。在这个过程中，虽然劳累，但是我掌握的博弈树的极大极小值搜索、negamax 搜索和 alpha-beta 剪枝，收获良多。不得不说，这一次的项目经验让我学会了很多东西。

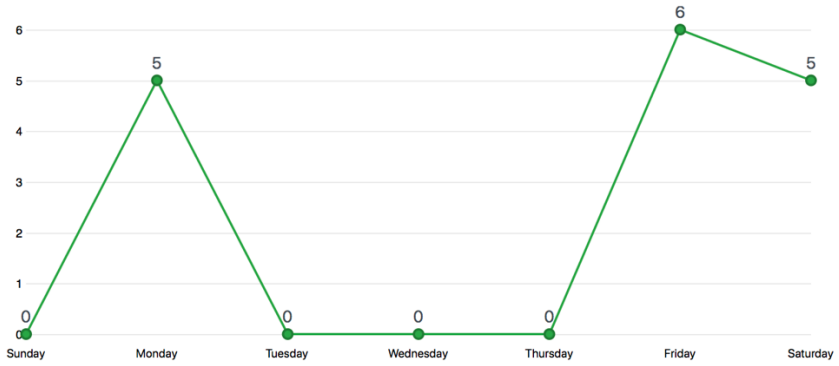

Figure 10 commit 统计

项目地址：https://github.com/cjhahaha/ai-games/tree/master

在线体验：https://cjhahaha.github.io/ai-games/