四子棋作业 实验报告

展子航 2022010745

Intro

本次实验中虽然算法不受限制，但是我仍旧先使用了实验介绍中推荐的 `min_max` 搜索和剪枝算法。经过尝试后，发现算法的瓶颈在于**评估函数的编写**，也即是需要自己先成为四子棋高手。但由于本人的棋艺不精（尝试过，批量只有40%胜率），因此最后采用 **MCST 蒙特卡洛搜索树**的算法方案。

算法基本思路

算法简介

将每一次的当前局面作为根节点，建立信心上限树。对其之后的局面在时间范围内**不断进行模拟**，对胜利局面给予数值奖励，失败局面给予数值惩罚。最后选择信心最大的根子节点进行下子。注意，**每一个节点需要记录其胜负场的评分，被反向传播访问的次数，代表的局面。**

算法步骤

一次模拟的步骤如下：

1. 根节点置为当前节点。
2. 如果当前节点可以扩展，则选择一个可以扩展的局面进行随机模拟，转 5。
3. 如果当前节点已经全部扩展完毕，则根据信心上限树评分获取一个最优的子节点作为新的当前节点，转 2 。
4. 对当前节点进行模拟、评分，得到一个初始的战绩。如果当前节点已经处于终局，则跳过模拟，直接评分。
5. 反向传播，将模拟结果评分添加至**直到根节点的所有直系节点。**
6. 完成。

在四子棋的游戏规则下，当以上的算法进行了足够多的次数，我们就会得到一个拥有多个节点的树，并且根节点下的子节点**必定至少扩展过**，因此选择根节点下最好的子节点作为决策结果。

个人所做修改

信心上限评分规则修改

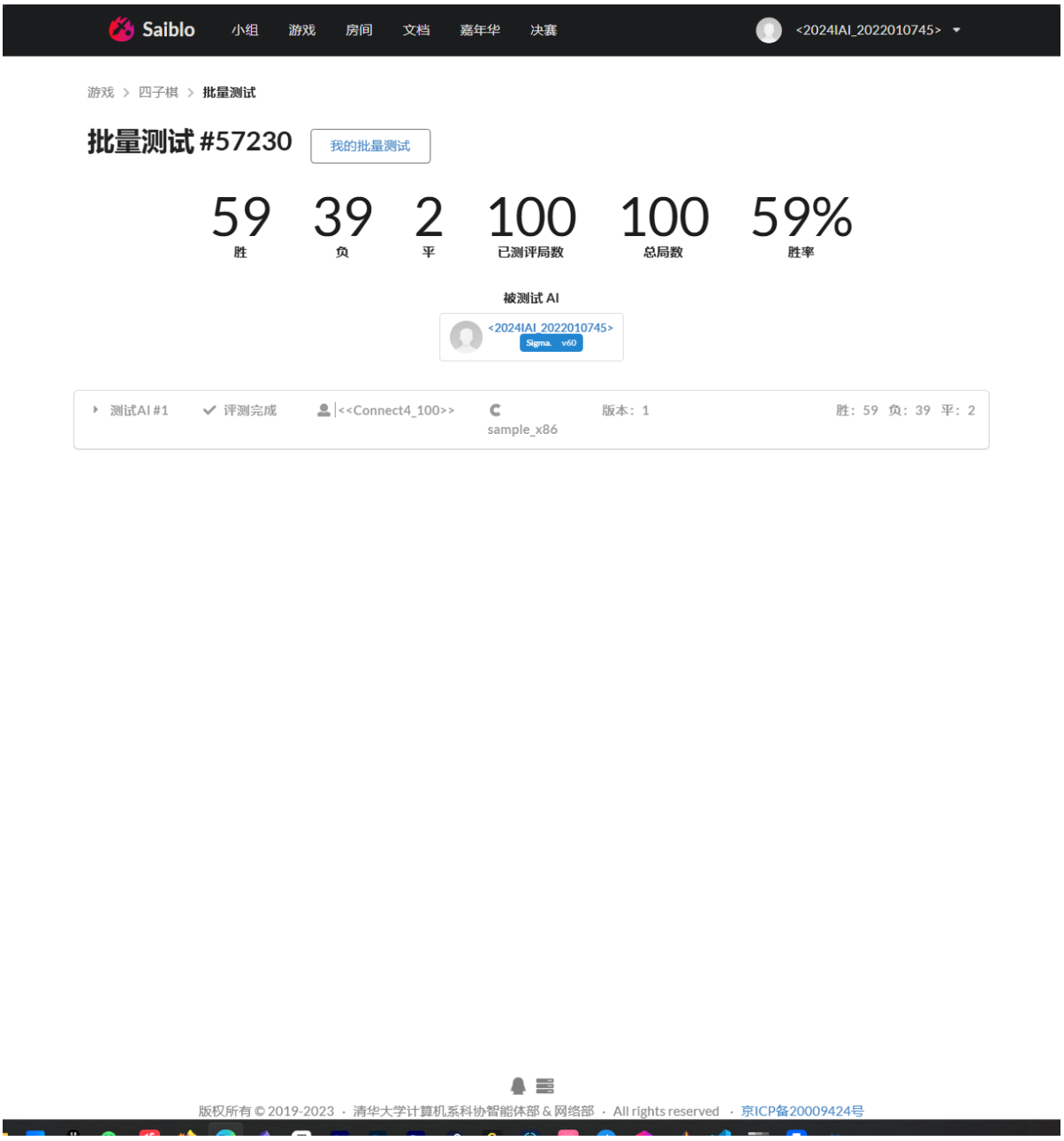
对于某个节点 `v` 和其父节点 `p` 信心上限树评分规定如下：

$$F(v) = site(v) * \frac{Win(v)}{Total(v)} + C * \sqrt{\frac{2 * \ln(Total(p))}{Total(v)}}$$

其中 `site` 为节点立场参数，`Win` 为该节点的胜负场评分，`Total` 为对节点被反向传播所访问次数，`C` 是自由参数，可以调整，**但在最终根节点进行评分选择时需要置为 0 ，因为此时的选择已经与访问率无关，只与胜率有关。**

节点立场参数代替了极大极小节点的区分存在。

最终使用 `C = 0.65`。此为与 `<<Connect4_100>>` 对弈数个 `100` 局选择。胜率约 60%。



加权模拟

如我一样的臭棋篓子都知道**下棋多下中间，出现四连的机会大。**对于模拟而言，我们可以在随机下棋时为下中间加权，这样下出有效模拟局的概率会变高。可以理解为改随机的分布。

基本的均匀分布模拟使用的是梅森旋转算法。

```
#include <random>
std::random_device rd;
std::mt19937 gen(rd());
```

加权的方式如下：

```
// 处于UCT初始化中
// 分段赋值 根据我的下棋经验 没人一开始下角上 因此中间大
int mid = (total_col - 1) / 2;
total_weight = (total_col % 2) ? (mid + 1) * (mid + 1) : (mid + 1) * (mid + 2);

for (int i = 0; i <= mid; ++i) {
 col_weight[i] = i + 1;
```

```
}

for (int i = mid + 1; i < _col; ++i) {
 col_weight[i] = col_weight[_col - i - 1];
}

// 取加权随机数
int rand_with_weight(int num_choices, int* weights, int total) {
   int rnd = int(gen()) % total;
   rnd = (rnd + total) % total;
   for (int i = 0; i < num_choices; ++i) {
      if (rnd < weights[i]) {
         return i;
      }
      rnd -= weights[i];
   }
   return (num_choices - 1) / 2;
}
```

额外必下手判定

对于必须下的棋（例如自身的四连，堵对手的三连）直接下，不进行树模拟。

时间参数优化

为了避免 TLE，模拟时间采用：

```
#include <ctime>
const double time_limit = 2 * CLOCKS_PER_SEC;
```

评测结果

其中输掉的两局分别为 `<<Connect4_92>>` 与 `<<Connect4_100>>` 。