Report

基本思路

本实验需要实现的是四子棋的人工智能算法来进行AI之间的相互对抗。基本思路是使用课上学习过的蒙特卡洛树搜索MCTS，并使用UCB1算法后得到的信心上限树算法UCT。即：

从当前状态开始，只要还未到时间上限（在代码中定了一个上限轮数），便继续迭代过程。

每一轮迭代过程，从当前状态，即根开始，若当前节点的儿子节点全部拓展完，则会根据每个儿子节点计算得出的UCB1值 $I_j = \bar{X}_j + c\sqrt{\frac{2\ln n}{T_j(n)}}$ （其中 $\bar{X}_j$ 为儿子节点的胜率， n 为当前节点总次数， $T_j(n)$ 为儿子节点总次数），不断选取一个儿子节点节点作为当前节点。若存在儿子未被拓展，则拓展该节点，并对于拓展的节点进行一个随机模拟，得到一个收益，然后将该份收益从拓展节点往上传递，并进行下一次迭代。

除此之外，在使用以上实现的算法进行了几次对弈后，发现有许多对于人类显而易见的步骤，算法却并不能正确给出。推测是由于蒙特卡罗过程的纯随机方案无法对一些极端的情况进行很好的预测。因此除了以上算法给出的最佳点之外，代码中还对下了能直接赢或者不下就一定输的一步进行了判断。

实现方法

在 `TreeNode.hpp` 中定义了结构体 `TreeNode` ，用于储存搜索树中的节点。每一个节点中保存有以下信息：

```
struct TreeNode
{
  int M, N;
  double win;
  int tot;
  bool self;
  int **board;
  int *top;
  int x, y;
  int ava_ch, exp_ch;
  int noX, noY;
  TreeNode *fa;
  TreeNode **ch;
};
```

并且实现了相应的一些方法： `terminal` 调用了 `Judge.h` 中定义的相关函数，用于判断该节点储存的棋盘局势是否有玩家胜出或平局。 `all_expanded` 用于判断当前节点是否已经拓展过所有可能的孩子节点。 `expand` 用于拓展当前节点的某个未拓展过的孩子。 `rollout` 用于对当前节点进行随机模拟到终局，并返回收益。 `print` 用于在调试过程中输出树的相关信息。

在 `SingleChoice.hpp` 中定义了 `SingleChoice` 函数，用于判断当前局面是否有单个最优步。该函数通过枚举并使用 `Judge.h` 中定义的函数判断了以下情况：

1. 对于每一个能走的步，是否能直接让自己赢，若能，则下一步下在此处。
2. 对于每一个能走的步，对方下在此处是否能直接赢，若能，则下一步下在此处。

在 `Strategy.cpp` 中， `getPoint` 函数获取相关信息后，会调用 `UctSearch` 进行搜索。按照基本思路中的算法，其中选择拓展节点是通过调用 `select` 函数来实现的，若当前节点未拓展完，则使用当前节点的 `expand` 方法来拓展一个孩子节点，若已经拓展完，则使用UCB1值来选取最优的孩子来继续选取。选取完节点后，就使用节点的 `rollout` 方法来产生收益，并向上传递收益一直到树根。

需要进行调参的参数只有计算UCB1值时惩罚项用到的 c 值。经过测试后，发现 C 取 $0.55, 0.6, 0.65$ 后的结果相近，最终选择 $C = 0.65$

评测结果

使用批量测试后，得到在偶数测试点上的胜率为 91%

91	9	0	100	100	91%
胜	负	平	已测评局数	总局数	胜率