人工智能导论: 四子棋

2021010761 计 13 班 程思翔

1 MCTS+UCT

1.1 MCTS

MCTS 是一种用于决策制定的启发式搜索算法, 适用于具有巨大搜索空间和不完全信息的问题. **MCTS** 通过模拟游戏的随机样本来构建和搜索一棵搜索树, 从而帮助决策者选择最优的行动.

MCTS 算法的核心思想是利用 **Monte Carlo** 方法进行随机采样和模拟, 通过对游戏状态进行随机模拟来评估不同行动的价值. 它通过不断扩展搜索树, 并根据每个节点的统计信息来指导搜索过程.

1.2 UCT

在 **MCTS** 算法的选择阶段, 根据已有的统计信息和一定的探索程度来选择下一步的行动. **UCT** 算法是一种基于上置信界的策略, 它在平衡已知价值 (利用) 和未知价值 (探索) 之间起到重要作用.

UCT 算法使用如下的选择公式:

$$UCB1(v_i) = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}}$$

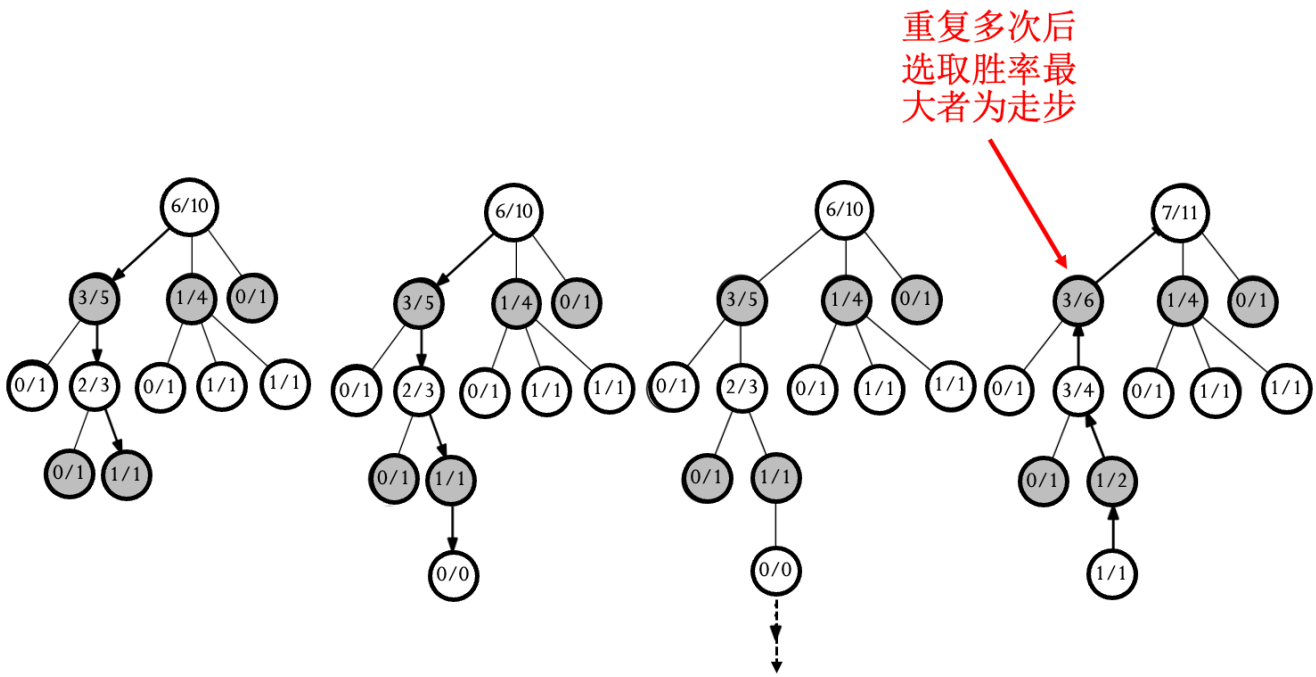
实现如下:

```
/* class UCT */
UCTNode* bestSolution(UCTNode* node) {
  double value = -FLT_MAX;
  UCTNode* best_solution = NULL;
  for (int i = 0; i < Glob_N; ++i) {
    if (node->child[i]) {
      double logN = C * sqrt(2 * log(double(node->visit)) / node->child[i]->visit);
      double val = (3 - 2 * node->player) * double(node->child[i]->win) / node->child[i]->visit + logN;
      if (val > value) {
        best_solution = node->child[i];
        value = val;
      }
    }
```

```
    }
    return best_solution;
}
```

其中 v_i 是候选子节点, w_i 是子节点的胜利收益, n_i 是子节点的访问次数, N 是父节点的总访问次数, C 是一个探索参数, C 越大越会考虑访问次数较少的子节点. 公式中的第一项表示子节点的平均胜率, 第二项表示子节点的探索程度. **UCT** 选择公式在探索和利用之间提供了平衡, 倾向于选择具有较高胜率和较少访问次数的节点, 但也给予探索未知节点的机会.

1.3 具体算法



- 选择: 根据当前获得所有子步骤的统计结果, 选择一个最优的子步骤. 从根节点 `root` 开始, 选择子节点向下至叶节点 `child` .
- 扩展: 当前统计结果不足计算出下一个步骤时, 创建一个或多个子节点并选取其中一个节点 `child` .
- 模拟: 从节点 `child` 开始, 模拟进行游戏.
- 反向传播: 根据游戏结束的结果, 更新从 `child` 到 `root` 路径上的获胜信息.
- 决策: 达到限定迭代次数或指定时间后, 选择根节点的最好子节点作为决策.

2 实现策略

在 `Strategy.cpp` 中实现了 `UCTNode` 和 `UCT` 类, 同时实现了相关策略.

2.1 边界特判

在随机过程中的每一步, 首先判断是否存在某一步使当前角色或对手立刻获胜. 若在某处落子后立刻获胜, 则在该处落子, 返回这次模拟的结果; 否则若对手在某处落子后立刻获胜, 则在该处落子, 返回这次模拟的结果.

```
/* class UCTNode */
int cornerCheck(int player) {
  int y = 0;
  for (y = 0; y < Glob_N; ++y) {
    if (tmp_top[y] > 0) {
      tmp_board[tmp_top[y] - 1][y] = PLAYER_SUM - player;
      if (isWin(player, tmp_top[y] - 1, y, Glob_M, Glob_N, tmp_board)) {
        tmp_board[tmp_top[y] - 1][y] = 0;
        return y;
      }
      tmp_board[tmp_top[y] - 1][y] = 0;
    }
  }
  for (y = 0; y < Glob_N; ++y) {
    if (tmp_top[y] > 0) {
      tmp_board[tmp_top[y] - 1][y] = player;
      if (isWin(PLAYER_SUM - player, tmp_top[y] - 1, y, Glob_M, Glob_N,
tmp_board)) {
        tmp_board[tmp_top[y] - 1][y] = 0;
        return y;
      }
      tmp_board[tmp_top[y] - 1][y] = 0;
    }
  }
  return -1;
}
```

2.2 收益值更新

在反向传播的收益值更新时, 起初我选择令 AI 获胜的收益为 1, 对手获胜的收益为 -1, 平局收益为 0. 考虑对妙手进行奖励, 收益值依据模拟步数定义如下:

```
/* class UCT */
int getGain(int count, int player, int x, int y, int** board, int* top) {
  if ((player == 2) && machineWin(x, y, Glob_M, Glob_N, board))
    return (count <= 5) ? 6 - count : 1;
  else if ((player == 1) && userWin(x, y, Glob_M, Glob_N, board))
    return (count <= 5) ? count - 6 : -1;
  else if (isTie(Glob_N, top))
    return 0;
  else
    return -6;
}
```

因为在每个 `UCTNode` 节点储存了当前的棋手状态, 进行上述反向传播的更新时不必区分极大与极小节点——若获胜, 将收益值加到节点统计结果中; 反之, 将收益值从节点统计结果中减去.

```
/* class UCT */
void backUp(UCTNode* node, double gain) {
  while (node) {
    node->visit++;
    node->win += gain;
    gain = (gain > 1)?(gain - 1):((gain < -1)?(gain + 1):gain);
    node = node->parent;
  }
}
```

2.3 落子权重

模拟过程中下在棋盘中部的棋子有更大的机会形成四连. 因此加入落子位置的权重, 从左到右侧权重依次为 $1, 2, \cdots, m-1, m, m, m-1, \cdots, 2, 1$. 在每次模拟循环中, 通过 `rand() % Glob_Sum` 生成一个随机数 `index`, 然后使用循环累加方式确定落子的位置 `y`. 当累加值超过 `index` 时, 确定落子位置. 使用条件判断 `(i <= (Glob_N - 1) / 2) ? (i + 1) : Glob_N - i` 计算得落子位置权重. 这样中部位置的权重较高, 角落位置的权重较低. 最后判断 `top[y]` 是否为 0, 以避免选择已满的列.

```
/* class UCT */
double defaultPolicy(UCTNode* node) {
  ...
  while (!isGameOver(...)) {
    player = PLAYER_SUM - player, y = 0;
```

```
    while (true) {
      int index = rand() % Glob_Sum, index_sum = 0;
      for (int i = 0; i < Glob_N; ++i) {
        index_sum += (i <= (Glob_N - 1) / 2)?(i + 1):Glob_N - i;
        if (index_sum > index) {
          y = i;
          break;
        }
      }
      if (top[y] != 0)
        break;
    }
    ...
  }
  ...
}
```

这样的调整可以在增加对中部落子位置的探索，更有可能选择中部位置进行落子, 适应四子棋游戏的特点, 强化了模拟效果.

2.4 参数选取

尝试对**UCT** 算法中 C 值进行调整. 一般而言, C 越大越会考虑访问次数较少的子节点, C 越小越会考虑胜率较高的子节点. 以下是本地批量测试的模拟结果:

C	**胜率** (/%)
1.2	91
1.1	93
1.0	96
0.9	95
0.8	92
0.7	91
0.6	92
0.5	90

综合上述结果, 选择 1 作为参数 C 的取值.

3 测评结果

最终版本的批量测试结果如下:

批量测试 #38004

96	4	0	100	100	96%
胜	负	平	已测评局数	总局数	胜率

被测试 AI

<2023IAI_2021010761>
~ci~c. v6

先手: 胜 47 负 3 **后手**: 胜 49 负 1

批量测试 #38015

97	3	0	100	100	97%
胜	负	平	已测评局数	总局数	胜率

被测试 AI

<2023IAI_2021010761>
~ci~c. v6

先手: 胜 48 负 2 **后手**: 胜 49 负 1

批量测试 #38032

97	3	0	100	100	97%
胜	负	平	已测评局数	总局数	胜率

被测试 AI

<2023IAI_2021010761>
~ci~c. v6

先手: 胜 49 负 1 **后手**: 胜 48 负 2

批量测试 #38068

95	5	0	100	100	95%
胜	负	平	已测评局数	总局数	胜率

被测试 AI

<2023IAI_2021010761>
~ci~c. v6

先手: 胜 50 负 0 **后手**: 胜 45 负 5