t1 — 四子棋（ConnectFour）——策略组一

下发文件

提示

注意：本题共包含 5 组测试点，在 OJ 上以 5 道题目的形式呈现，具体内容详见【评分方式】一节。请选手务必阅读完本题目中的所有说明后再开始作答。

题目背景

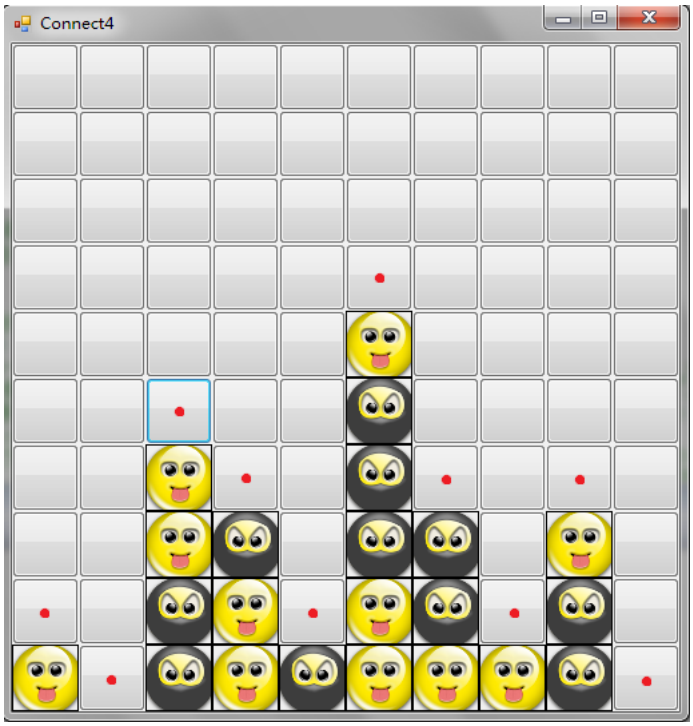
在人工智能领域，博弈问题一直以来都是一类具有代表性的研究课题，博弈问题中一些新课题的提出，也不断的推动着人工智能学科的发展。下发文件 `AdversarialSearch.pdf` 是博弈问题中对抗搜索的简介，选手可以阅读该简介以获取部分背景知识与相关算法。

本题改编自清华大学计算机系《人工智能导论》课程的一项大作业，以四子棋博弈游戏为背景，希望选手能够根据对抗搜索简介实现一部分对抗搜索算法，提高对人工智能算法的实际应用能力，更为直观和真切地体会人工智能。

题目描述

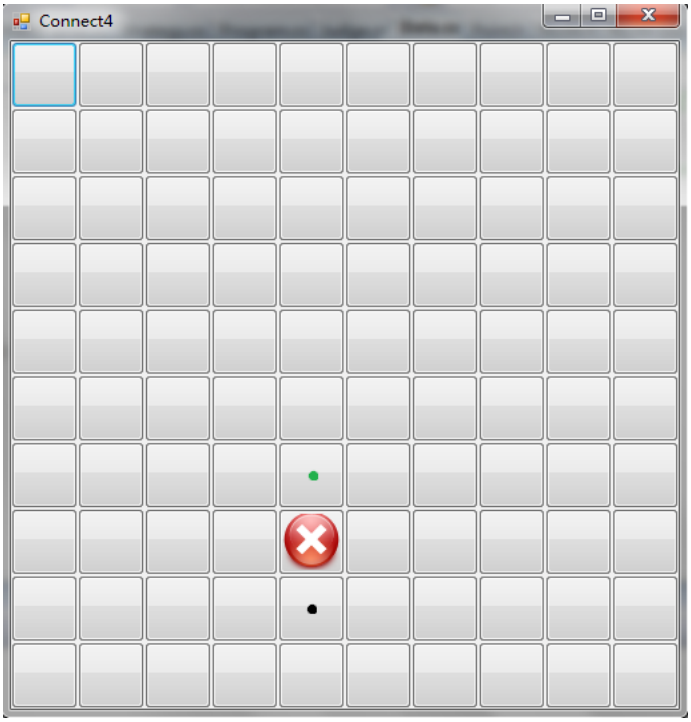
四子棋是一款经典的博弈游戏。其具体规则如下：

游戏双方分别持不同颜色的棋子，设 A 持白子，B 持黑子，以某一方为先手依次落子。假设为 A 为先手，落子规则如下：在 M 行 N 列的棋盘中，棋手每次只能在每一列当前的最底部落子，如图中的红点处所示，如果某一列已经落满，则不能在该列中落子。在图形界面中，如果在某一列的任意一个按钮处点击，会自动在该列的最低端落子。



棋手的目标是在横向、纵向、两个斜向共四个方向中的任意一个方向上，使自己的棋子连成四个（或四个以上），并阻止对方达到同样的企图。先形成一方棋子的一方获胜，如果直到棋盘落满双方都没能达到目标，则为平局。

本题中，增设了以下一条游戏规则：**每次棋盘生成之后，会同时在棋盘上随机生成一个不可以落子的位置，如图中的红叉所示。**当某一次落子是在黑点上时，该列下一次可落子的位置就变为了绿点处，而不再是黑点上面的位置。



你需要在给定框架的基础上完善决策部分，形成一份完整的策略文件，并战胜尽可能多的样本策略文件。

评分方式

本题共包含 50 个不同的样本策略文件，按从弱到强的顺序分别编号为 `1.so` 至 `50.so`。其中 `2.so`、`4.so`、……、`50.so` 已随代码框架一同下发，`1.so`、`3.so`、……、`49.so` 将用于评测。

用于评测的 25 个样本策略文件按编号从小到大均分为五组（例如：第一组包含 `1.so`，`3.so`，`5.so`，`7.so`，`9.so`），在 OJ 上以五道题目的形式呈现。每道题目的评分方式如下：

- 将你提交的策略文件与 5 个样本策略文件先后手各对抗 1 次，共对抗 10 次。
- 与每个样本策略文件对抗时，M, N 会在 9 至 12 的整数（包含两端）中独立均匀随机，不可以落子的位置会在棋盘中均匀随机。与这个样本策略的两次对抗会采用**相同的**棋盘大小与不可落子位置，但与不同样本策略文件的**可能不同。**
- 每次对抗中，若你的策略获胜，得 10 分；若平局，得 5 分；否则不得分。每次提交的得分为 10 次对抗的得分总和。每道题目的得分为该题**所有提交中得分的最大值。**

本场比赛的最终得分为五道题目的得分之和。

实现与调试

下发文件提供了完整的代码框架与本地评测方式。**你只需要且只应当修改 `Strategy` 文件夹中的代码。**

`Strategy` 文件夹中包含以下文件：

- `Judge.h`、`Judge.cpp`：用来进行胜局检测，相关函数的具体说明请见代码中的详细注释；
- `Point.h`：定义了 `Point` 类，用于记录棋盘上的一个点。**注意：棋盘中左上角为坐标原点，纵向为 x 坐标轴，横向为 y 坐标轴，(x, y) 点对应于棋第 x 行第 y 列的位置（从 0 开始编号）**；
- `Strategy.h`、`Strategy.cpp`：定义了策略函数同外部调用程序之间的接口。
  - `getPoint` 函数：在对抗时，外部程序在每次需要落子时通过调用该函数获得你给出的落子点。你的策略最终应当封装到该函数中，给出你在本中的落子。以下是对传入参数的解释：
    - `M`、`N`：表示棋盘大小为 M 行 N 列，保证 $M, N \in [9, 12]$。
    - `lastX`、`lastY`：表示对手上一步的落子位置为 $(lastX, lastY)$，刚开局时传入 $lastX = lastY = -1$。
    - `noX`、`noY`：表示不可落子的位置为 (noX, noY)。
  - `top` 数组给定了当前棋局各列的顶端位置，这些位置上方的格子是你下一次落子的地方。形式化地说，你在编号为 y 的列的下一次落子位置是 $(top[y] - 1, y)$。特别地，如果你在这一列不能落子了，那么 $top[y] = 0$。
    - `board` 二维数组则给出了当前棋局的所有情况。你可以假设自己策略正在同某一用户进行对弈，那么 `board` 中 0 为空位置，1 为有对手棋位置，2 为有自己棋子的位置。不可落子点处的 `board` 值也为 0。
    - **更多详细内容可以参见代码中的注释。**
    - **注意：由于评测只会进行完整的对局，你可以通过全局变量记录完整对局的信息。**
  - `clearPoint` 函数：策略模块中动态定义的对象必须由策略模块中的函数来释放；`getPoint` 函数返回一个 `Point *`，该指针指向的对象应当通过面调用 `clearPoint` 函数来释放。**你不需要且不应当修改该函数。**
- `main.cpp`、`makefile`：用于提交时的编译与运行。**你不需要且不应当修改这两份代码。**
- 如有需要，你也可以自行添加辅助文件。

在进行本地测试之前，你需要完成以下步骤：

1. 在下发文件目录下运行 `mkdir so`，建立用于存放策略文件的目录；
2. 在下发文件目录下运行 `mkdir Results`，建立用于存放对抗结果的目录；
3. 在 `Compete` 目录下运行 `make`，得到进行对抗的可执行文件 `Compete`。

`TestCases` 文件夹中包含 `2.so`、`4.so`、……、`50.so` 共 25 个样本策略文件，你可以按照如下方式进行对抗：

1. 在 `Strategy` 目录下运行 `make so`，得到你的策略程序的可执行文件 `Strategy.so`，存放在 `so` 目录下；
2. 在 `Compete` 目录下运行 `./Compete <Strategy1> <Strategy2> <result> <n>`，其中 `<Strategy1>`、`<Strategy2>` 为两个需要进行对抗的 `so` 文件路径，`<result>` 表示对抗结果的存放路径，`<n>` 为对抗轮数。**注意：若 `so` 文件位于当前目录下，请在路径前加入 `./`，即使用 `./*.so` 表示当前目录下的 `*` 的路径。**

例如，如果你想将你的策略程序与 `2.so` 对抗 5 轮，并将结果存放在 `Results` 目录下的 `2.txt` 中，应在 `Compete` 目录下执行 `./Compete ../so/Strategy.so ../TestCases/2.so ../Results/2.txt 5`。

使用 `Compete` 进行本地测试时：

- Strategy1 和 Strategy2 将对抗 n 轮，每轮两次，分别为 Strategy1 先手和 Strategy2 先手。
- 每轮的 M, N 会在 9 至 12 的整数（包含两端）中独立均匀随机，不可落子位置会在棋盘中均匀随机。每轮的两次对抗会采用**相同的**棋盘大小与不落子位置，但不同轮之间**可能不同。**
- 每次调用 `getPoint` 函数的时间限制为 3 s，空间以硬件资源为限。每次对抗的总时间不作限制。
- 得到的对抗结果将存放在 `<result>` 文件中。

`<result>` 文件中的前 $4n$ 行，每四行表示一轮对抗结果，其中第二行和第三行的三个整数分别表示 Strategy1 和 Strategy2 先手时的返回值、Strate 总用时（以秒为单位）、Strategy2 总用时（以秒为单位）。返回值的具体含义如下表所示：

返回值	具体含义
0	平局
1	Strategy1 获胜
2	Strategy2 获胜
3	Strategy1 运行错误
4	Strategy1 给出的落点不合法
5	Strategy2 运行错误
6	Strategy2 给出的落点不合法
7	Strategy1 时间超过限制
8	Strategy2 时间超过限制
-1	载入文件 Strategy1 出错
-2	载入文件 Strategy2 出错
-3	Strategy1 文件中无法找到需要的函数接口
-4	Strategy1 文件中无法找到需要的函数接口

注意：当返回值不为 $0, 1, 2$ 时，Strategy1 与 Strategy2 的用时无意义。

`<result>` 文件的最后若干行为这 $2n$ 轮对抗的总体信息，具体含义见其中的文本。

提交与评测

提交时，你需要将 **`Strategy` 目录下的所有文件**压缩为一个 `zip` 格式的压缩包提交。**注意：压缩包内不应含有 `Strategy` 目录，应直接包含 `Strategy.cpp` 文件。**

提交代码时将会受到以下限制：

- 提交的 `zip` 文件大小不得超过 100 KB；
- **五道题的总提交次数不得超过 10；**
- 对于第 i 题（$i \in \{2, 3, 4, 5\}$），若第 $i-1$ 题得分 < 60，则第 i 题只能提交 1 次；
- 任意两次提交间需要间隔至少 1 分钟。
- **注意：由于评测时间较长，每一次提交都需要若干分钟才能够返回结果，请选手耐心等待。**
- **注意：由于评测资源有限，最后 30 分钟的提交不保证能在比赛结束前得到评测结果。请选手合理安排提交时间。**

每次提交代码将会使剩余的提交次数减 1，若编译失败，则剩余的提交次数加 1。

评测的时空限制如下：

- 内存限制为 1 GB；
- 每次调用 `getPoint` 函数的时间限制为 3 s。每次对抗的总时间不作限制。

评测结束后，你将获得你的每次对抗的得分与你的策略程序的运行结果。以策略组一为例，得到的 10 个测试点分别对应以下对抗的结果：

测试点编号	对抗
1	你的策略程序先手，`1.so` 后手
2	`1.so` 先手，你的策略程序后手
3	你的策略程序先手，`3.so` 后手
4	`3.so` 先手，你的策略程序后手
5	你的策略程序先手，`5.so` 后手
6	`5.so` 先手，你的策略程序后手
7	你的策略程序先手，`7.so` 后手
8	`7.so` 先手，你的策略程序后手
9	你的策略程序先手，`9.so` 后手
10	`9.so` 先手，你的策略程序后手

运行结果的具体含义如下表所示：

运行结果	具体含义
`You Win` / `Draw` / `You Lose`	运行正常
`Runtime Error`	运行时错误
`Time Limit Exceeded`	时间超过限制
`Memory Limit Exceeded`	内存超过限制
`Output Limit Exceeded`	输出超过限制
`Illegal instruction`	非法指令（例如策略程序给出的落点不合法）
`System Error`	系统错误（如有遇到请联系监考或技术人员）

注意：由于对抗具有随机性，同一份策略文件的多次提交有可能获得不同的结果。

语言

zip

代码 选择文件 未选择文件

提交
