四子棋实验指导书

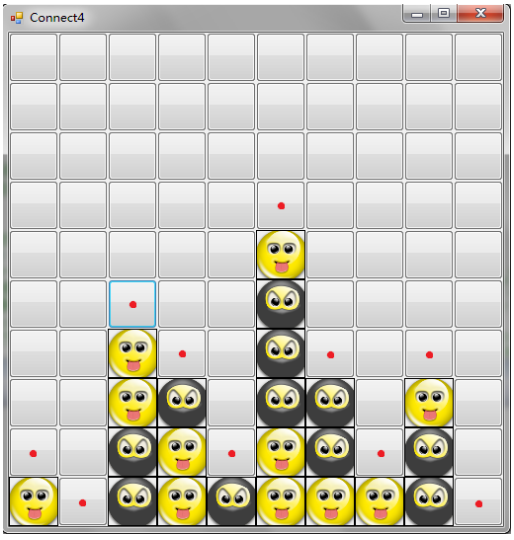
一、关于本实验

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。

本实验以四子棋博弈游戏为背景，主要包括了剪枝算法、棋局评价、威胁检测与排除以及同学们自己的创新性算法等方面，主要锻炼大家对人工智能算法的实际应用能力，从而使大家对人工智能有一个更为直观和真切的体会。

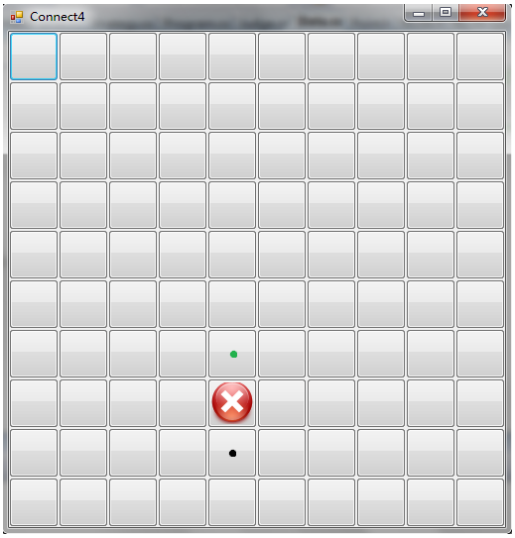
二、四子棋游戏介绍

1. 基本游戏规则



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上点击，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。先形成四连子的一方获胜，如果直到棋盘落满双方都没能达到目标，则为平局。



2. 本实验对规则的扩展

如果你查阅一下相关文献和资料，就会发现某些棋盘规模下的四子棋游戏是具有必胜策略的，一个典型的例子是 6（行数）× 7（列数）。

在 1987 年，美国计算机学家 James Dow Allen[1]，首先提出了该棋盘下的算法策略，在该策略下先手一方可以保证不输，后手一方

[1] James Dow Alllen 的个人主页 http：//fabpedigree.com/james/index.htm

最多可以拿到平局。仅在几周之后，荷兰计算机科学家 Victor Allis[2]也宣布独立地完成了四子棋必胜策略的工作并发表了相关论文[3]。

由于我们实验的评测方法是让大家的策略与已有的样例策略进行对弈，所以如果某一策略完全使用了必胜算法，将造成评测结果没有实际意义，因此我们对规则进行了如下改进。

首先，当两个策略进行对弈时，棋盘的大小是随机的，而不是固定不变的（宽度和高度的范围均为[9, 12]）。由于并不是所有规模的棋盘都有必胜策略，也不是所有的必胜策略都是先手必胜，所以这里带来了不确定因素，你应该使自己的算法尽可能普适。

其次，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处时，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。这样防止同学们在自己的策略中针对每一种棋盘大小分别实现必胜算法，再根据棋盘的实际大小选择对应的算法。

当然，以上两个小修改的目的并不是完全阻止你研究和使用必胜策略，而是防止你不加思考地使用必胜算法。实际上，必胜算法的很多思想即使在这两条规则的限制下也是可以发挥作用的，如果你想使自己的策略更加智能、更加有力，就可以学习和研究相关算法，加以吸收和改进，并应用到你自己的策略中，这就需要你发挥自己的创造力了。

三、实验要求

本次实验的实验要求为，在我们提供的实验框架下完成四子棋游戏的人工智能决策部分，并将你的策略文件（Windows：dll 文件，Mac：dylib 文件，Linux：so 文件，后面会具体介绍）与我们提供的不同级别的样例策略文件进行对抗，如果你的策略文件能够打败某一级别的样例，就能获得该级别的分数。

该实验推荐的基本方法为$\alpha-\beta$剪枝算法，当然这不是硬性的限制。本实验中大家可以使用任何可能的算法，只要该方法能使你的策略更强大即可。

四、实验框架介绍

我们为你提供了两个工程：Strategy 和 Compete，下面将分别介绍。

1. **Strategy 项目**：该项目是你唯一需要从中编写代码的项目

Windows 系统：经编译后生成的名为 Strategy.dll.的文件即为包含你的策略的策

[2] Victor Allis 的个人主页 http：//chessprogramming.wikispaces.com/Victor+Allis

[3] A Knowledge-based Approach of Connect-Four The Game is Solved：White Wins（1988）

略文件（如果使用 Visual Studio 编译器，该文件在 Release 或 Debug 目录内）。

Mac 系统：经编译后生成的名为 libStrategy.dylib 的文件即为包含你的策略的策略文件（如果使用 Xcode 编译器，该文件在 Products 下的 Release 或 Debug 目录内）。

Linux 系统：经编译后生成的名为 Strategy.so 的文件即为包含你的策略的策略文件。

我们的评测系统对你的落子速度有时间限制，具体将在评测环节说明，为了提高你的程序速度，建议使用 Release 编译器。

- Judge.h、Judge.cpp：

用来进行胜局检测的函数，你可能使用得到也可能使用不到，相关函数的具体说明请见代码中的详细注释，这里只强调一些细节：由于在某一方落子得胜之前，一定不可能出现过任何一方的胜局，所以我们只需要在某次落子之后以本次落子点为核心判断是否有多于四个的连子即可，userWin 函数和 computerWin 函数就是这样做的，并且我们只需要在 user 落子时候判断是否 userWin，在 computer 落子之后判断是否 computerWin。由于 isTie 通过检测棋盘是否已满来判断是否为平局，这就意味着调用 isTie 必须在调用 userWin 或 computerWin 之后，假如某次为用户落子，那么之后当 userWin 为 false 时，调用 isTie 进行判断才有意义。

- Point.h：

Point 类，用来记录棋盘上的一个点。注意框架中对坐标的规定：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）点对应于棋盘中第 x 行第 y 列的位置（从 0 开始计而不是从 1 开始计）。

- 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，给出你在本步中的落子。

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在外面调用 clearPoint 函数来释放，你不需要修改该函数。

注意点：

1) 关于接口函数传入的参数

top 数组给定了当前棋局各列的顶端位置，这些顶端的上面是你可以落子的地方，如果你的策略给出了非法的落子，那么程序会给出提示。

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 为有用户棋子的位置，2 为有计算机（自己的策略）的棋，不可落子点处的 board 值也为 0.

对于不可落子点，外部调用程序已经做出了处理，比如某次落子点为 A 位置，而 A 上面的 B 位置为不可落子点，那么下次传递给你的 top 数组中该列列顶已被设为了 B，而不是 A，所以你的策略中不需要考虑对不可落子点进行特殊处理。当然，如果你想充分分析不可落子点，以做出更好的策略也是可以的，我们将不可落子点（noX，noY）直接传递给了你，你可以充分利用。

实际操作时，请注意看代码中详细的注释，将对你理解整个项目有很大的帮助。

2) 最好用 Release 编译器以保证你的落子速度，打印操作会严重影响你的策略的速度，所以在最终版本中最好去掉所有的打印调试信息。

3) 不要随意修改工程的各种属性。

4) （lastX，lastY）给出了上一次对方落子的位置，刚开局是传入的 lastX 和 lastY 值为-1。当然为了更好地组织你的策略，你可以不仅仅利用上一次落子这一步信息，可以在你的策略中将其保存下来，综合多步以便决策时使用。实际上，传入的参数中 top 和 board 给定了当前的状态，lastX、lastY 给定了过去的过程，这样也就为你提供了全部可用信息。

5) 进行思考时，可以认为自己的程序正与一个用户对抗，对方的棋子用 1 标示，自己的程序的棋子用 2 标示。

2. Compete 项目介绍

该项目实际上是一个非界面、命令行式的对抗平台，你不需要关心这里面的代码。Compete 可执行程序允许你将两个策略文件进行指定轮数的对抗，并给出结果统计，避免你在图形界面上频繁的点鼠标，我们最终也将使用这个平台来将你的策略同样例策略进行对抗。

具体使用方法可参见该项目下的 readme.txt。

五、对抗测试

1. 本地对抗

通过系统下编译器编译 Strategy 项目得到策略可执行文件，然后调用 Compete 可执行文件进行对抗测试。

作业中“批量测试”目录下提供了 Windows、Mac 和 Linux 系统各自的批量编译、对抗脚本，需要将脚本中的待编译文件及编译器路径修改为本地目录，同时指定对抗轮数等参数。具体使用方法见各自系统目录下 readme.txt 文件。

2. 在线对抗

除了通过本地 Compete 平台进行对抗，还可以进入网站 **Saiblo** (https://www.saiblo.net/game/3)进行在线对抗。**Saiblo** 网站可以支持人类玩家之间、AI 之间、人类玩家与 AI 之间的对战，帮助同学熟悉游戏和调试策略。

在线对抗时，可以直接提交自己 Strategy 项目下源代码文件，网站后台自动进行编译。需要注意的是，网站搭建在 Linux 服务器，对于使用 Windows 和 Mac 系统的同学，需要代码中不包含平台特定的库才可以正常进行编译运行（例如 Windows.h），可以将自己的策略实现部分加入作业 Linux 版本相关文件进行提交测试。

六、评测方法

在本实验中，我们共有 100 个样本策略文件，它们分别为 100.dll/dylib/so~1.dll/dylib/so，这些 dll/dylib/so 文件事先通过循环赛的方法确定了排名，其中 100.dll/dylib/so 的棋艺最高超，1.dll/dylib/so 的棋艺最差，我们将偶数文件（100.dll/dylib/so，98.dll/dylib/so，...，2.dll/dylib/so）提供给大家用于训练和测试，保留奇数文件（99.dll/dylib/so，97.dll/dylib/so，...，1.dll/dylib/so）用于最终的评测。

1. 只要你的算法能够正常运行，不论其智能程度如何，都能得到基本分 40 分。正常运行主要包括生成的策略文件能够被测试平台正常载入和使用、程序不出 bug、不给出非法落子（棋盘外/不在列顶/不可落子点处落子）、不超时等方面。
2. 实验报告占 20 分，实验报告应描述智能算法的基本思路、方法和评测结果，中英文均可，我们会仔细阅读实验报告并给出综合得分。
3. 在如上的基础上，我们会让你的策略文件同保留的 50 个评测文件的每一个进行比赛，采用积分制。与每个评测文件对抗 2 次，分别为先手和后手，共对抗 100 次。每次对抗中，胜利得 2 分，平局得 1 分，失败不得分。最终的积分被划归到 40 分满分（总积分 * 40/200），作为及格分之上的加分。

在同每一个对手进行对弈时，你的程序可能出现的情况如下表所示：

你的程序	比赛判定结果	积分
胜出	你赢	2
失败	对手赢	0
平局	平局	1

出 bug	对手赢	0
给出非法落子	对手赢	0
某步落子超时	对手赢	0
无法载入策略文件	对手赢	0
找不到需要的接口函数	对手赢	0

单步落子的时限定为 3s，这个时限仅仅是为了对死掉的程序进行处理，所以为了保险起见，建议你不要为了搜的更深而让自己的程序单步时间特别接近 3s，考虑到测试服务器的性能在不同运行环境下有可能不同，所以特别接近于 3s 将是危险的。

七、实验上交内容

实验完成后，请建立一个命名为“<学号>_<系统>”文件夹（如 2020001000_Windows，2020002000_Mac，2020003000_Linux），并将如下四部分内容拷贝到该文件夹中打包提交：

1. Strategy 文件夹下所有的源代码文件，包括：Judge.cp，Judge.h，Strategy.cpp，Strategy.h，Point.h 以及自定义的*.cpp 和*.h 文件。注意：切勿提交除此之外的其他代码文件。在测试时，我们将对你的代码重新编译，所以请你不要在自己的代码中使用 C++标准库之外以及系统、平台特定的库函数。我们将使用 Compete 将你的策略文件同样例进行对弈，所以请确保你的策略可执行文件能在对应平台下正确运行。
2. 可执行文件（以学号命名）：Windows 下<学号>.dll 文件，Mac 下<学号>.dylib 文件，Linux 下<学号>.so 文件。
3. 与 50 个提供的样例正反手各对抗一次的（50 个）结果文件，放在命名为“results”的文件夹下。
4. 实验报告：格式为 PDF 或者 Word 文档，需要列出对抗结果的统计数据，包括<胜数、负数、平数>。在实验报告中，你应当对你的策略进行充分的说明，根据你自己的实现充地阐释你所设计的方法，以突出你为什么觉得这样的算法将是聪明的。在将你的程序与样例进行对抗的同时，我们也会认真评价你所采用的方法的创新性，这将让你获得额外的加分。