Design and Analysis of Algorithms

Project Report #1

姓名	**罗鉴洪**
学号	09016414
当前版本	0

总览

在正式展开叙述项目细节之前，让我们对这几周的进展进行总览。

我们的五子棋 AI 名为 retardedAI，其 ID 为 5a9b9db9ed7b6b4cf832a3c，作者为冷安 Serica，亦即我本人。目前稳定版的版本号为 0，截止本周，到其在 botzone 上的排名为 6 到 8 名上下。

这是一个由 JavaScript 编写的 AI，起初选用这个语言的原因是为了方便图形化调试。了解到 botzone 平台上和 Bot 的交互方式，我们认为用 JavaScript 是一个不错的选择。事实上亦是如此，借助于语言特性，我同时让这个程序很轻松地跑在了我的服务器上，可以随时通过 https://iserica.com/#/lab 访问。

注意到我所提供的代码按照 botzone 的交互规范书写，因此他并不能在本地直接运行，你可以再 botzone 上创建游戏桌与之进行对局，或是在我的个人网站上对局，他们的版本一致。

何为 AI

目前人工智能浪潮再一次来袭，没错，是再一次。人工智能领域经历了几次兴衰，借助于目前唾手可得的海量数据和不断提升的计算能力，人工智能又迎来了发展的生机。于一个大二本科生而言，能在算法课上的大作业中写一个五子棋 AI，这十分有挑战性，但也给了我更广泛的视野。

在开始叙述我们的五子棋 AI 算法前，我们有必要对 AI 有一个认识。AI 应该有这样的表现，即可以像人一样行动和思考，并且这些思考和行动应该是合理的。理想情况是，我们的智能 Agent 在给定环境下应该采取最好的可能行为。

当然，这并不一定是 AI 的最宽广和准确的定义。实际上也有业内人士指出：“今天其实没有人在做 AI，但是每个人都说他们在做 AI，因为这是一个很酷而且很性感的流行语，与几年前的大数据一样。”（Erik Cambria，自然语言领域专家）

没错，类似大数据这样的数据分析科学，以及人工智能，这些研究方向都难免有重合。而现在的弱人工智能在很大程度上还只是基于“人工”经验的程序。无论是人赋予它的智能，如传统 AI 算法，或者是人教会他的智能，如机器学习，这背后的实质都是数学理论和大数据。

前面说到，我们的智能 Agent 应该做到再给定输入为状态 S 的情况下，要做出一个决策（或者说行为）R。对于五子棋 AI，环境就是五子棋的棋局，而行为就是下棋，我们希望这个棋的落点更接近于让棋局向 AI 获胜的方向发展。我们需要一个理性的 Agent，即是根据 Agent 已有的先验知识（如在某种状态下，应该下在哪里对自己最有力），选择（一定度量标准下）价值最高的行动。

为了实现 Agent 的理性，我们需要建立一个模型。简单说来，有两种方式获得模型。一种是基于人工经验，教会 Agent 在某种状态下应该采取什么行动。第二种是让 Agent 自己去学习到一个模型，而我们只作为一个“导师”的存在，仅仅告诉他这种行动对于这种情况而言是好是坏。以上两种，前者属于较为传统的人工智能，往往离不开搜索和推理，往往需要大量的人工经验。后者则是机器学习，里面又包括强化学习和深度学习（在博弈问题上，主要是深度强化学习）等，这种方法不需要很多人工经验，但需要很大的数据，经由数据训练出一个模型或是算法。

鉴于时间和水平有限，我们的 Agent 将采用前者实现。

价值评估

在后面我们会看到，我们的 Agent 很可能将通过搜索对问题进行求解。这实际上很符合人的一种行为模式。比如我们要开车去某一个陌生的地点，到这个地方有很多路。我们可能对路线并不熟悉，但我们如果有手机，那我们就会知道可能选择哪些路线，进而从这些路线中选一个合理的（如是否堵车、是否有很多红绿灯等）。在地图上，我们从一个点沿着路开往另一个点，其结果是确定的，不断寻找路线开到新的点的过程就是寻找一个任务序列的过程，这也是搜索的过程。

而搜索的空间不应该是无限的，这样我们将不知道什么时候停止搜索。因此，我们必须对搜索空间进行约束。约束条件首先就是我们所处的状态，即我们现在在哪里。我们所处的位置就是求解路线的起点，而不是其他地方。随后，我们要知道我们可能的行动，我们是开车，而不是坐公交。之后便是寻找可能的行为序列，如从起点开到 A 点，再开到 B 点进而到终点。我们需要有一个测试过程，它应该保证我们的行为序列是可行的，即我们最终可以到达终点。同时，我们应该尽量保证这个行为序列是较优的，可以为我们节省时间和燃油。

因此，我们首先，要有一个价值评估的函数。对五子棋而言，给定一个棋局 S，我们应该要给出每个落点的价值。显然，我们的 AI 根据每个落点的价值高低，可以做出决策。

具体地，在程序中，我们遍历整个棋盘，得到对手的棋型特征，如是否有连起来的三个子等。如果有，那我们有必要进行拦截。根据这些特征，程序对每个可能的落点进行打分，打分依据为这个落点的防御价值和进攻价值。如果防御价值高，说明这个点应该落子以进行防御，否则对手就连成四个子，对我们威胁很大。如果进攻价值高，说明这个点可以落子进行进攻，我们可能离连成五个子又接近了一步。

程序实现

我们这一版本的五子棋 AI 在学会价值评估后就基本告一段落了。虽然他尚不具有"算棋"的能力，但他已经知道哪些落点有价值，哪些落点可以不予考虑了。

在具体的程序实现上，我们分为两个模块：retardedAI.js 和 evaluator.js

retardedAI 就是我们的 Agent 实例，通过 Agent 实现对外的所有交互。因此在 botzone（或者网站服务器端）上的交互通过以下几行代码就可以实现：

```
import Agent from "retardedAI";
var retardedAI = new Agent();
retardedAI.init();
retardedAI.setState(input);
var response = retardedAI.decide();
```

在 botzone 上，由于单文件程序，因此第一行引入模块部分并不需要。

具体地，这两个模块的作用可以根据名称得出。前面说了 Agent 实现对外交互，而 evaluator 模块即为价值评估模块，通过他告诉我们的 AI 当前棋局什么落点是有价值的，AI 的决策即根据此。

两个文件对于变量和方法定义速览如下：

```
import Evaluator from './evaluator'

var Agent = function () {
    this.board = [];
    this.evaluator = new Evaluator();
}

// 初始化 Agent 中的内部成员变量，如棋局 board 数组
Agent.prototype.init = function () { ...

// botzone 中的输入和 Agent 的交互接口，Agent 根据输入设置棋局状态
```

```
Agent.prototype.setState = function (input) { ...

// 具体地，在(i, j)处下一颗子
Agent.prototype.put = function (i, j, role) { ...

// 根据棋局状态，返回当前认为最有价值的落点坐标
Agent.prototype.decide = function () { ...

export default Agent
```

```
var Evaluator = function () {
this.winStateCount = 0;
this.winState = []; // 这个数组存储所有可能的获胜方案

// 棋型判断的辅助数组，存储一些位置信息
this.winStateInnerEndPoint = [];
this.winStateOutterEndPoint = [];

// 存储人类在某种获胜方案上达成的进度
this.humanWinStateStat = [];
// 存储 AI 在某种获胜方案上达成的进度
this.agentWinStateStat = [];
// 某点的防御价值
this.defenceValue = [];
// 某点的进攻价值
this.attackValue = [];
// 棋局
this.board = [];
// 价值矩阵，存储每一点的价值
this.valueMatrix = [];
// 价值数组，可以对其进行排序得到最有价值的前 n 个点
this.valueArr = [];
}

// 每下一个子，就更新人类和 AI 在某种获胜方案上实现的进度
Evaluator.prototype.updateWinStat = function (i, j, role) { ...

// 评价(i,j)的价值
Evaluator.prototype.evaluatePoint = function (i, j) { ...

// 得到(i,j)的防守价值
Evaluator.prototype.getDefenceValue = function (i, j) { ...

// 得到(i,j)的进攻价值
Evaluator.prototype.getAttackValue = function (i, j) { ...

// 返回最有价值的点
```

```
Evaluator.prototype.evaluateFast = function () { ...

// 初始化各个数组信息
Evaluator.prototype.init = function () { ...

export default Evaluator
```

在如何快速从棋局信息中得到每个点的价值这一问题上，我做过许多种尝试。根据直觉，首先想到的就是遍历棋盘上所有的点，从上、下、左、右、左上、左下、右上、右下 8 个方向得到连子的信息。判断出这个点目前所处的状态，这个点是否有进攻作用，如 AI 在某方向上已经连了两个子；是否有防守作用，如对手在某个方向上连了三个子。

我们大可以写一长串逻辑判断来匹配所有的模式，如“活三”、“眠四”等等，但考虑每种棋型的所有可能状态上十分复杂的，如中间是否有空的点，是否有对手的子等等。一种优化的思路上是建立字典树（Trie Tree），建树的过程中枚举所有可能的情况，然后再进行搜索，搜索结束后我们可以得到棋型符合的模式（pattern），从而实现打分。

以上是基于理论分析可行性，但尝试后认为不够方便实现，而且遍历中会涉及许多重复计算，因此开始思考其他方法。是否能够减少运算量，同时更加简洁和灵活呢？参考了网上的一些博文，我发现可以借助一些辅助的“备忘录”，让 Agent 在一遍遍落子生成棋局的过程中，自动记住对手在哪些获胜方案上已经落了多少子了，而自己又在哪些获胜方案上迈进一步了。如此，在决策时我们只需要遍历这个备忘录，就可以很快得到哪些点具有高的进攻或防守的价值了。其实，这更像是人在下棋时的思维模式：我们在下棋时，并不是搜索一个个可能的落点，观察其周围的情况，而是在对弈的过程中，已经发现或计算得到了哪里可能起到进攻的作用，哪里必须要防守而不至于输掉比赛。

但是，网上博文中的方法有其局限性，因此亟待改进，这虽然提供了思路，但我们需要进一步探索。为什么有局限性呢？因为备忘录能记录的信息有限，这仅仅是一种数据存储，并不能和人的大脑匹敌。仅仅通过备忘录，我们的程序只能知道在某种获胜方案上自己或对手已经实现了几颗子，但并不知道这几颗子的具体情况，例如周围是否有堵截等。因此，我们也就引入了上面棋型判断的辅助数组，这方便我们判断这种赢法的真正价值（如侵略性或威胁）。

通过修正，我们的价值评估更加精确合理。对于价值上这样评判的，即分为进攻和防守的价值。进攻价值根据某种获胜方案上实现了几颗子以及周围情况评分，例如我们可以知道这是“活三”还是“眠

四”。防守亦然，不同的是，进攻上我还加入了“位置”这一影响因子。我们认为，在进攻上，往往有更大的“发展空间”是更有利的，即不能让棋子往“死胡同”里下。

对于同一种棋型，进攻价值高于防守价值，这使得我们的 Agent 不会为了防守而忘记自己的进攻，特别在还有一步或两步就可以获胜的情况下。同时，这可能也有利于我们今后加入搜索时对棋局的评判。对于某一落点的价值，取进攻价值和防守价值中的大者。

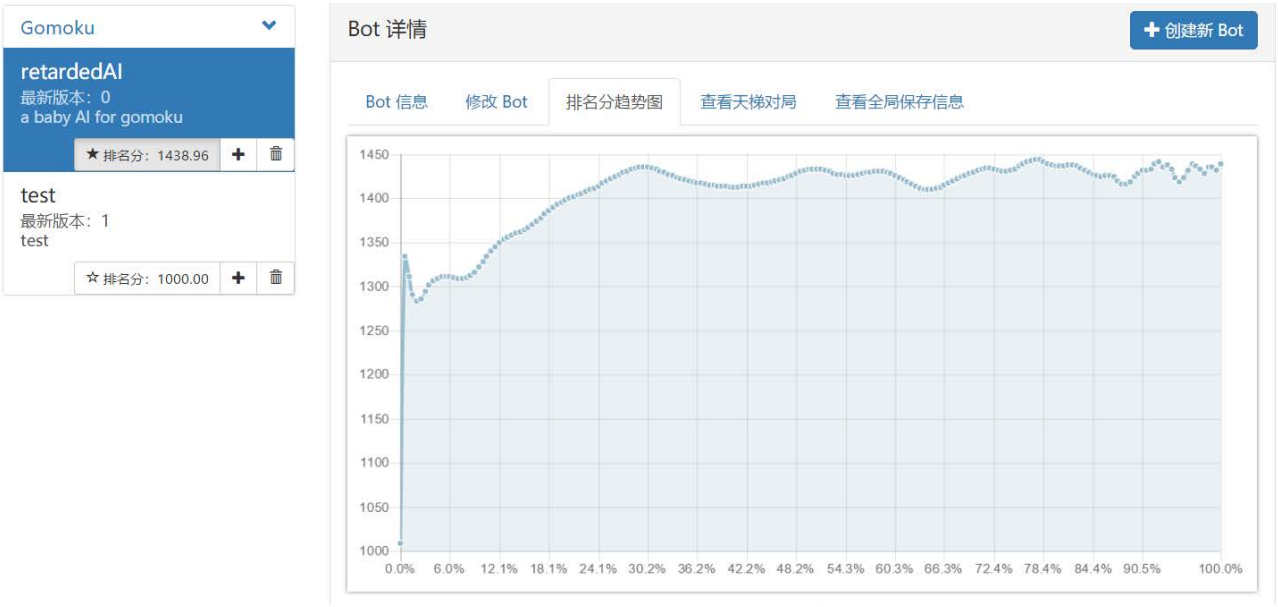
小结

本阶段主要实现了：

1. Agent 的基本程序架构
2. Agent 与 botzone 的交互
3. Agent 的价值评估
4. botzone 上不错的排名（当然，这可能主要是因为参与排名的 bot 较少，之后排名可能会下跌，但我会尽力保证其有更好的表现）
5. 将 bot 放在我的网站上运行，可以随时访问进行对局：https://iserica.com/#/lab
6. 未来将尽力实现搜索功能，使得 AI 可以多算几步棋，以实现更好的进攻和防守策略

附录

当前排名（2018.3.18）



参考文献

Stuart J. Russel, Peter Norvig 著．人工智能：一种现代化的方法（第 3 版）[M]．北京：清华大学出版社，2013．

部分参考还来自网络