计算概论 A　2025 秋 大作业(Amazons) 创作报告

信息科学技术学院 柏盛元 2500013113

摘要：

本文在大作业程序设计过程中撰写，将沿着作者设计各代 Amazons AI 的心路历程，从框架到决策方案，逐代分析各代程序在功能和算法上的改进。作者会将各代 Amazons 的源代码放在附件中供读者参考，也敬请指教。目前推出至第五代，后续可能会继续迭代更新。

正文：

初创（Amazons-R0）：PVP 框架

起初拿到这个作业的时候，我并没有听过这种棋类。在阅读了助教提供的介绍文件 Amazons.docx 并在 botzone 上亲自尝试了几局后,我对如何设计这种棋类的决策 AI 有了一个初步的想法：遍历所有走法，对每步走法做出量化评分并选择最优解。

但在制作 AI 决策算法之前，我们还需要先搭建一个能通过键盘输入实现交互的对弈界面。在实现人机对弈界面之前，我决定先设计一个较为简单的玩家对弈界面（这一界面在后续模型中也有保留，作为 PVP 模式）。

首先，我采用坐标读入，并定义了结构体 Position{int r(行), c(列)}。列坐标为 a~h，行坐标为 1~8，输入一个字母＋数字表示选定一个坐标（如：c4 表示第四行第三列）。每次读入三个坐标，分别表示起点，落点和箭点。每次行动后将起点清空，终点修改为该方棋子，再将箭点填充（注意，这里的**顺序很重要**，因为箭可能会落到起点上）。

接下来就是判断输入合法性。这一步比较枯燥，我参考了 ChatGPT 给出的建议，先设计了 inBounds 函数判断选点是否在棋盘内和 isClearPath 函数判断两点是否共线，然后再分别设计了 validMove 和 validArrow 判断合法移动与合法放箭（二者区别主要是起点是否需要为当前行棋方的棋子），这对后面的搜索算法产生了很大帮助。（以上两步在后续都归并到函数 turnForPlayer 中）

要特别提醒的是，如果输入的行棋方式非法，则要保持**棋盘和行棋方均不变**，输出“Invalid move”或“Invalid arrow”并要求再次输入。

然后是棋盘绘制。我最初画了几款棋盘，但输出效果都并不十分好看，于是这简单的图样设计我就丢给了 AI 完成（它画的确实相对美观，详见 printBoard 函数）。游戏进程中，开局会输出一次初始棋盘，每次行棋后会更新并输出新的棋盘。

再一个比较关键的点就是胜负判断，即在每步棋下完之后判断对手还是否有棋可走。

GPT 给了我一个正确但比较复杂的判定过程（**此处为本项目最后一次使用 AI 生成代码，后续内容均为独立完成**），而我们发现，**一方仍可移动当且仅当该方有至少一枚棋子周围（八个方向的相邻点）仍有空格**，于是在这里设计了一个更简单的 hasAnyMove 函数判断棋局是否仍可继续（在 Amazons-R1.cpp 中开始使用），这里不过多赘述。

一切对局函数设计完后，我们对棋盘初始化，分别用’.‘’ X ‘’ B ‘’ W‘表示空位，箭矢，黑棋和白棋，用变量 turn 表示当前行棋方，然后就可以开始游戏了。

Amazons-R1：引入 PVE 模式

在搭建完 PVP 框架之后，就要开始设计人机对弈模式了。刚刚我们的所有回合均为 Player's Turn，现在，我们在选定持方后，另一方的回合将由计算机执行。

怎么让计算机找到合适的走法呢？（**为方便后续介绍，以下讨论均认为人执白，AI 执黑**）在 R1 中，我先遍历所有格点找到四枚黑棋，对每枚黑棋在遍历所有格点找到所有合法的 move，再对每个合法的 move 遍历所有格点找到合法的 arrow，由此找出黑方所有行棋方式（此处复杂度最坏为 4*64*64，平均大约为 4*20*64，有较大优化空间）。

然后就是最关键的形势判断环节。我们用 **Evaluate 函数**对每种局面进行评分（这是各代 AI 中最核心、最关键的函数）。对于每种走法得到的局面，我们根据双方对空格的占领程度进行评价。首先，我们用 GradeB/ GradeW 记录该局面双方得分，初始值为 0。在这版模型中，我们对每个点进行 BFS（这一过程使得 **Evaluate 函数非常耗时**，后续仍需优化），查看其需要多少步能找到黑方/白方棋子，分别记找到黑方/白方所用的步数为 stepB/ stepW（若无法找到则记 step=10）。对于所需步数，无疑是越少对该方越有利，于是我们先在双方的得分上减去该 step。然后，我们发现双方对每个空格存在占据关系，走到该空格所需步数越少，对该空格的占据能力越强。于是，我们用 stepB 和 stepW 的差值，定义了**相对占有，基本占有和完全占有**三个概念，并根据不同占有程度为占有方加上一定的分数。（具体判定方式及得分见附件 Amazons-R1.cpp>>Evaluate 函数）

实际上，R1 中三种占有程度的得分并**不能**很好的反映局面的形式，其相对 step 贡献的分数也显得过小，这会在后续的模型中得到改进，这其实也是模型数次改进的核心内容。

最终我们取得分最高的走法作为 AI 的决策并输出。

注：此版本由于只进行了一层搜索，用时较短，符合 botzone 的限时要求，故不在此讨论对时间复杂度的优化。

Amazons-R2:由单步搜索化为两步搜索，在玩家回合加入提示功能

上一版本中，我们只对每种走法走完后的局面进行了分析，至于后面会发生什么，我们

不得而知。这显然会导致我们的最优解判定方式具有一定局限性，比如：某一个格点，黑方占据后并不会对黑方的领地产生什么增益，但白方占据后可能会获得大量领地。因此，我们不得不对我方行棋后对手可能做出的决策加以考虑。

于是，在时间复杂度尚且允许的情况下，我们设计了**二层搜索**，在黑方每种走法的 Evaluation 后再对白方进行同样操作：找白棋，找合法的 move，找合法的 arrow。然后，对每次操作再进行 Evaluate。

注意，我们的 Evaluate 函数返回的均为 AI 方的得分，故内层（白方）的 Evaluate 要尽可能小，而外层（黑方）Evaluate 要尽可能大。

同时，在这一版本中，我们在玩家回合加入了提示功能。玩家输入“a hint needed”，则可由 AI 为玩家提供一个在 AI 视角下的玩家最优解。该过程与 AI 回合的决策大体相同，只需在执行前后各调换一次 AI 和玩家的持方即可。与 AI 回合不同的是，此功能只提供一个最优解，而不执行该下法，即只做提示，**不对棋盘做出改动**，这一点在输出函数中需注意（具体实现详见附件 Amazons-R2.cpp>>turnForPlayer 和 output 函数中“hinting”相关内容）。

同时，我们对上一版形势判断不合理的给分做出了调整，大大提升了占据效应对分数的贡献，使形式判断更加合理（详见附件 Amazons-R2.cpp>>Evaluate 函数）。

注：评分参数虽不存在最优质值，但通过不同模型对弈，可比较得出其相对合理性。

我们在运行时发现，由于增大了一层搜索深度，此版本的决策用时远远高于上一版本，在对局前期会有极长的等待时间（30s+）。于是我们在**内层搜索前加入了限制条件**：如果第一步得分比之前发现的第一步得分要低很多，则直接放弃该方案，不进行第二层搜索。这很好地帮我们在第一层搜索的时候排除掉了一些明显错误的下法，比如自己把自己往角落里围等。跟据不同的时间限制需求，我们可以调整上述“低很多”这一概念，及当本次的一层 Evaluate 评分比当前找到过的最高一层 Evaluate 得分要低 Δ Grade 或更多，就将当前走法舍弃。

经测试，当 Δ Grade 取 **60**，在本地运行时，前期单步思考时长在 15~25 秒不等。这一等待时长随对局进行而减少，当对局进行到约 20 步后，单步思考时长能控制在 3s 内。这已经满足了本地人机对弈的需求，同时决策仍能保持较高准确性，即这一剪枝未给搜索的结果带来太大变化。而当我们在 botzone 对局时，它对思考时间的限制是 1s，这就导致我们不得不将 Δ Grade 下调至 **20** 左右（其实 20 有时候也会超时），而这也导致了其决策智能性大大降低。经测评，该模型的水平与 botzone 上排名 400~500 的 bot 相当，而这显然不能满足我们的需求，于是我们不得不对我们的搜索和评分算法继续优化。

Amazons-R3：增强相对占领得分，减少非阻拦性行为的搜索

此版本改动较小，仅作简单说明。

我们在对局过程中发现，R2 的 AI 在决策中经常为了将较小一篇区域绝对占有（即白方完全无法进入），而忽视了更大的片区。尽管黑方此步完全占有了一片地盘，但却因此错过了与白方争夺另一片更大空间的良机。于是，我们调整了不同程度占有所带来的得分，使得这一情况虽仍然存在，但明显有所好转。

同时，为了进一步缩短思考时间，使得更大的 Δ Grade 也能满足 botzone 的限时要求，我们定义在一个位置放箭具有**阻拦性**当且仅当这个位置可被地方棋子一步到达。我们一般认为，阻拦性射箭往往比非阻拦性射箭更有价值。于是，我们在每个 Evaluate 之前加一个阻拦性判断。若该下法不具有阻拦性，则直接舍弃，这样可以大量减少 Evaluate 函数带来的耗时。尽管这可能让我们错过一些不具有阻拦性，但实则为妙手的下法，但其时间效益更为我们所需。

要注意的是，行至中后盘后，可能会不再存在阻拦性下法（即双方领地完全分离），这时候就不能再做此剪枝，否则会无法更新最优解法，使得输出仍为上一回合最优解导致报错。经实践测算，一般达到双方领地完全分离需要 **30~40 步**，故在 30 步之后结束该剪枝较合适（代码实现详见附件 Amazons-R3.cpp）。

Amazons-R4：优化了 Evaluate，新增棋谱记录和悔棋功能

先前提到的 Evaluate 函数需要跑大量 BFS 的缺陷在这一版本得到了优化。之前我们从每个空格点出发，各自进行 BFS 寻找目标棋子。而实际上，我们完全可以**从棋子出发**，以四枚棋子为初始点进行 BFS，这样可以一次 BFS 搜完所有空格点。这使得 Evaluate 的效率提升了十余倍！在进行了这一步优化后，我们把 Δ Grade 上调到 100，本地前期单步计算时间也能控制在 10s 内，这大大提高了我们程序的运行效率和决策智能程度。这一优化虽然叙述上比较简短，但却是对计算耗时最大程度的一次优化。

同时，课程大作业要求实现复盘和悔棋功能。对局复盘功能要使得某次被终止的对局可在下次继续。由于我们尚未搭建 UI 界面，遂先用棋谱记录代替此功能。开一个 step[64]，记录每步的 from，to，arrow。在对局过程中输入“make a record”，可查看截至当前的步数（以六个数字输出，分别为 from，to，arrow 的行、列坐标），对局结束时也会自动打印整局棋谱。

悔棋功能即棋盘更新的逆过程，这建立在棋谱记录的基础上。每步回退，需**依次**将上一步的箭矢清空，落棋点清空，出发点还原为该棋子。同时改变当前行棋方，并删去 step 栈

中最新添加的元素。注意，在 PVP 模式中一次悔棋只需回退一步，而 PVE 模式中一次悔棋需回退两步。（记录与悔棋功能的实现详见附件 Amazons-R4.cpp>>Regret 和 docuOutPut 函数以及 turnForPlayer 函数中相关判定内容）

Amazons-R5：分层搜索，全盘搜改为直线搜

之前，我们的两层搜索是深度优先进行的，即搜到黑方的一种下法后立马搜该下法对应的地方下法。这就使得我们 Δ Grade 的剪枝只能和先前最大值比较，而不能进行全盘比较。为此，我们尝试将其调整为**逐层搜索**。我们新建结构体 Step{Position f, t, a; int g;}，用 Step aiChoice[]来储存一层搜索找到的每一种黑方下法，对其按得分排序后，取一定数量的较优黑方下法再进行二层搜索。

（代码实现详见附件 Amazons-R5>>findTheBest 函数中相关修改）

这样有两个好处：**1，与一层搜索遍历到的节点顺序无关。**举个夸张的例子，若一层搜索中，遍历到的黑方下法一次比一次好，那 Δ Grade 的剪枝就完全失效了；而逐层搜索可以将所有一层节点遍历完再进行整体比较，不会因为搜索到节点的顺序而影响剪枝效率。**2，二层搜索次数可直接调控。**Δ Grade 的剪枝要进行二层搜索的节点数取决于有多少节点落在一层[maxGrade, maxGrade - Δ Grade]这一区间内，而这个区间内的节点数是不稳定的。但调整后，我们只取一层节点中一定数量个较优节点进行二层搜索，这一数量是我们可以人为控制的。简单来说，就是直接控制前 k 个节点进入二层要比规定某一区间内节点进入二层更为稳定。

另外，我们在 R1 版本中提到过，全图找棋子、搜合法的移动和射箭是很费时的事。实际上，我们可以时刻记录八枚棋子去向，对每枚棋子只需要在其**八个运动方向**上寻找即可。并且，一个方向一旦中断即结束（不可能跳过一个空格抵达更远的空格）。由此，我们的合法行动搜索可由全盘搜简化为直线搜。（代码实现详见附件 Amazons-R5.cpp>>searchMove 等函数以及 dr[], dc[], chess[][]数组）

小结

经历了五代更新优化，作者的 Amazons 以及有了较为科学、完备的决策能力，在对弈水平上已超过作者本人。与人类相比，其优势在于**完备的形势判断能力**，可通过 Evaluate 函数（结合 searchB/W 函数）对每个局面进行彻底的分析，从而选出最优解。而其劣势则在于尚且未能进行往后**多步的计算**，使得在一些局部斗争中表现得不尽如人意，这也是我们后期要优化模型战力的一个重要方向。

写到这里，作者的 Amazons 已经在 botzone 上达到**约 150 名**的棋力。尽管后续仍有较大

优化空间，但由于诸多外部因素（如期末周等），暂且搁置此作。本学期结束后若仍有时间，会再对此模型进行优化，并于明年 1~2 月推出新的版本。届时会继续更新此报告，也会将新作同步至 botzone，敬请关注。

注：关于本地模型适配到 botzone 的代码实现，详见附件 Amazons-botzone.cpp，此处不过多赘述。

最后，是对本作品未来改进方向以及遇到的瓶颈的一些补充：

1，**计算时长**一直是一个较大的限制。

2，如何尽可能避免对**较明显的无意义下法的搜索**。现已在第二轮搜索中进行剪枝，下一步将试图寻找方法使得在进行 Evaluate 之前就将其舍弃，从而减少 Evaluate 的耗时。

3，如何再次拓展模型**计算的层数**。目前两层计算的模型的计算时长以接近限时极限，增加层数会大幅提高耗时。同时，此模型的代码要进行层数的增加也需要做出较大改动，如何**使层数也变为模型的一个简单参数**以便于后续优化，也是改进的方向之一。

4，优化当前的 Evaluate 方式，现行 Evaluate 只关注了一个点到它最近的棋子需要的步数。这会导致可能一个棋子同时照顾了多片区域的情形，而实际上一个棋子最终只能去往并作用于一片区域。一个可行的办法是《An Evaluation Function For The Game Of Amazons》中提到的用 **queenStep 和 kingStep** 记录到达一个点的**线动和邻动**所需步数，从而表示出一个棋子对某一点占领的**稳定性**。而作者自己的思路是分别测定四个棋子到某一点的距离，**实现对多棋子对一点联合控制能力的量化**。

5，课程资料中提到的 **MC/UCT 方法**以及相关论文，作者由于时间原因尚未研读，这或许是未来改进此模型的另一个新方向。

参考资料：

【1】计算概论课程资料 Amazons.docx；

【2】计算概论课程资料 AmazonSimpleSample.cpp；

【3】论文 An Evaluation Function For The Game Of Amazons -- J. Lieberum