目　录

摘 要

机器博弈是人工智能研究的一个重要领域，规则简单却具有复杂博弈空间的围棋更是其中的代表，也是检验人工智能算法的重要应用场景。然而，尽管围棋游戏受到诸多研究者的关注，围棋博弈中仍有许多尚未被探索的课题，其中作为围棋游戏精髓的死活问题就是其中之一。基于蒙特卡洛方法的强化学习，及其衍生的蒙特卡洛树搜索、UCT 算法一直以来是解决围棋问题的主要算法，自然也适用于解决围棋死活问题。因此，本文基于围棋死活问题的游戏规则和目标，设计了一个专用于求解围棋死活题的 UCT 算法，并利用算法在自开发的围棋死活博弈系统中对不同难度的真实死活题进行求解实验，实现了一个死活题求解能力不亚于人类爱好者平均水平的强化学习智能体。最后，本文创造性地提出了一些算法优化方案，进一步实现了对智能体的改良强化。

关 键 词：强化学习；蒙特卡洛树搜索；围棋博弈；死活问题

ABSTRACT

Computer game is an important area of artificial intelligence, and the game of Go, with its simple rules but huge game space, is a representative of the perfect information game problems and a touchstone for a large number of artificial intelligence algorithms to be applied. However, although the game of Go has received much attention from researchers, there are still many unexplored topics in

the game of Go, one of which is Tsumego, which is a type of go problem based on life-and-death and the essence of the game of Go. Reinforcement learning based on Monte Carlo methods and its derived algorithms including Monte Carlo Tree Search and UCT algorithms have been the main algorithms for solving Go problems, and naturally they are also applicable to solving Tsumego problems. Therefore, in this paper, we design a UCT algorithm dedicated to solving Tsumego problems based on its unique game rules and objectives and conduct experiments by applying this algorithm to solving real Tsumego problems of different difficulties in a self-developed Go gaming system, which succeeds in realizing a reinforcement learning agent whose Tsumego solving ability is no less than the average level of human enthusiasts. Finally, this paper creatively proposes some algorithmic optimization schemes to realize the further improvement of the intelligent agent.

Key words: Reinforcement Learning; Monte-Carlo Tree Search; Computer Go; Tsumego

一、绪论

（一）研究背景

计算机博弈（机器博弈），是人工智能领域的主要研究问题之一。机器博弈旨在利用人工智能技术，在一定规则定义下，根据不同的局势采取正确的策略，从而实现己方利益最大化，最终赢得博弈胜利。棋类游戏因其自身游戏特点与博弈问题性质相符合，因此是计算机博弈问题的主要研究对象。其中以围棋为代表的双人零和完全信息博弈问题，因其不存在第三方因素干预、多方合作、规则简单等特点，更成为人们研究机器博弈问题的焦点。

围棋（Go），是一种历史悠久的双人策略博弈类游戏。对弈双方分别执黑白棋子在正方形棋盘上依次落子，从而占据棋盘区域。游戏最终通过比较双方棋子占据区域的面积大小判断输赢。其规则简单，但巨大的落子空间导致棋盘上的状态千变万化，双方棋手面临的棋面局势也错综复杂，如何做出最优决策成为决定输赢的关键难题，而这恰是人工智能技术在博弈问题中期望实现的。

强化学习（Reinforcement Learning），也称为增强学习，是机器学习的一个分支。2016 年 DeepMind 在《自然》上发表了一篇轰动世界的论文，其通过结合强化学习和深度学习，实现了棋艺比肩围棋大师的智能体[1]。同年该团队开发的围棋程序 AlphaGo 完胜了世界冠军李世石和柯洁，再一次将强化学习带入众多研究者的视野，同时也证明了强化学习在计算机围棋问题中的无穷潜力。

围棋死活题，是指在一个给定初始棋局内，双方以杀死对方棋子或救活己方棋子为目标的围棋题目。死活题可以理解为正式围棋比赛中的子问题，正式围棋比赛由无数个局部区域的死活问题组成，并影响最终胜负的。死活问题是围棋的精髓，因而是广大棋手们的研究课题和训练方法。

（二）研究现状

计算机围棋自上个世纪以来便一直受到研究者们的关注和研究。计算机围棋发展至今主要经历了两个重要阶段，分别为“传统计算机围棋”和“现代计算机围棋”。

1968 年美国威斯康星大学阿尔伯特佐布里斯特利用哈希算法对围棋棋面状态进行编码，并采用自设计的估值函数对棋面进行评估[2,17]，从而第一次实现了能模仿人类分析棋况并执行落子的围棋博弈智能体。1993 年，世界首个基于蒙特卡洛搜索树的围棋博弈算法诞生[3,17]。1994 年美国萨克生物研究学院的库隆提出利用时序差分算法解决围棋棋盘的估计问题[4,17]。传统计算机围棋主要采用专家系统的方法，主要是将围棋专家的下棋策略、经验转换成固定的计算机规则，使机器在遇到相应场景时可以执行相对正确的落子。专家知识的方法非常依赖于预备的围棋专业知识，需要给算法投喂大量“如果——就”的规则。但利用该方法也开发出许多优秀的围棋程序，其中具有代表性的是 GNUGO。

2005 年 UCT（Upper Confidence Bound Applied to Tree）算法的提出，标志着对计算机围棋博弈的研究进入到近代计算机围棋时期。2006 年法国国家信息与自动化研究院将上线信心界限算法（UCB）和蒙特卡洛搜索树搜索算法（MCTS）结合并首次应用于围棋上[5,17]，直至今日主流的围棋博弈程序仍是以

该框架为主。2007 年 David Silver 首次将强化学习应用至计算机围棋博弈上，并成功在 9*9 棋盘上实现大师水平的智能体[6,17]。随着深度学习技术的进步和算力的增强，人们开始尝试引入深度神经网络解决为其问题。2013 年 DeepMind 首次结合深度学习和强化学习应用于 Atari 游戏[7]，并在 2014 年利用深度卷积神经网络实现围棋走法估值[8]。2016 年，震惊世界的 AlphaGo 诞生，该算法使用了蒙特卡洛树搜索与两个深度神经网络相结合的方法，成功击败两位世界冠军，成为人工智能技术在计算机博弈领域应用的里程碑[1]。2017 年，DeepMind 开发的改良版 AlphaGo Zero 使用优化的强化学习框架轻松超越前代版本[9]。现代计算机围棋主要采用强化学习和深度学习结合的方法，借助强化学习实现无需依赖人类围棋经验的博弈智能体，同时借助深度学习解决信息存储问题和复杂计算。此后，大量研究者投身于对深度强化学习的改进，标志着计算机围棋迈入一个全新时代。

虽然计算机围棋取得的成绩众多，但对于围棋死活这类子问题，相关成果却屈指可数。对死活问题的研究者做出重要贡献的是加拿大 Brock 大学的 Thomas Wolf 教授，其开发的 GoTools 是解决围棋死活问题的经典程序[10]。GoTools 中采用了围棋专家的经验知识从而增加算法效率和准确性，并运用 Alpha-Beta 搜索算法求解封闭边界的死活问题。

（三）研究内容与意义

由于围棋死活问题在围棋游戏中具有举足轻重的意义，而计算机围棋又是计算机博弈问题，因此对围棋死活问题的研究有利于计算机围棋的发展，也可以为其他博弈问题提供方法参考。目前已有的大多数解决围棋死活问题的程序都是通过 Alpha-beta 搜索算法、最小最大搜索算法（Min-max Search）或其他一些传统方法实现，而强化学习方法在死活问题中并未受到重用。

因此本论文主要以计算机围棋为背景，强化学习为解决围棋死活问题的核心策略方法，蒙特卡洛树搜索作为支撑强化学习的框架，实现了基于蒙特卡洛树搜索方法的求解围棋死活问题的智能体。本论文的主要工作内容如下：

1. 概述了围棋博弈的基本规则以及围棋死活问题的定义，对围棋中的核心概念，如“气”、“眼”、“活棋/死棋”等进行详细介绍；
2. 介绍了强化学习的基本概念、常用方法和核心问题。重点阐述了不基于模型的强化学习方法——蒙特卡洛方法，以及其衍生的基于模拟探索的强化学习方法——蒙特卡洛树搜索算法，并给出蒙特卡洛树搜索进一步改良后的 UCT 算法框架；
3. 设计开发了一套支持“人机交互”的围棋死活题博弈系统，并基于 UCT 算法实现了一个可以解决围棋死活问题的智能体。通过对超参数、算法框架等方面的调整实现了对算法的调整优化，并利用该算法对一系列不同难度的死活题进行测试实验，取得了较好的效果。

二、围棋博弈

（一）围棋规则

随着围棋的发展和广泛传播，现代围棋规则主要衍生为中国围棋规则和日本围棋规则。两者的主要差别在于死活棋的判断规则不同。中国围棋的判别规则较为简单，仅利用最基本的提子规则和基于区域比较的输赢规则进行判断。而日本围棋规则相对复杂，通常包含附加规则。在计算机围棋研究中，人们普遍使用更为简单的中国围棋规则，本文所有内容亦以中国围棋规则为基础。

1.围棋棋盘

围棋使用正方形格状棋盘及黑白二色圆形棋子进行对弈，常见的围棋棋盘有 9*9、11*11、19*19 三种大小，正式比赛通常使用 19*19 大小的棋盘。19*19 围棋棋盘包含纵横两个方向各 19 条线段，构成 361 个交叉点，其中包含 9 个用黑点加粗标记的特殊交叉点。棋盘中央的黑标记交叉点称为“天元”，其余标记交叉点称为“星位”。星位和天元作为参照物，可以帮助玩家更容易地确定棋盘上其他交叉点的位置。

由于本文重点解决死活问题，而围棋死活题是一种局部区域的问题，只关注局部的棋面、落子策略和死活状态，因此本文主要采用 9*9 大小的棋盘，从而更好地聚焦死活问题本身，同时也便于算法求解。本文采用的棋盘如图 2.1 所示。

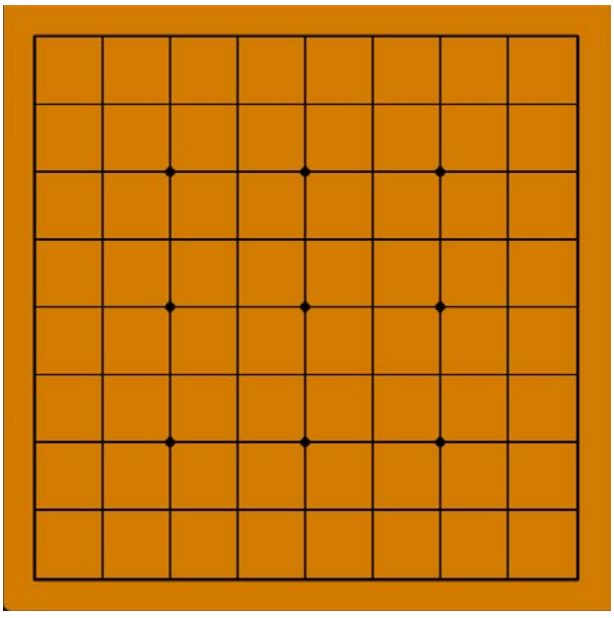


图 2.1 围棋棋盘（9*9）

2.基本概念与行棋规则

（1）基本下法

对弈双方各自执黑子和白子依次在棋盘上落子，一次一子。双方可选择的

落子点为棋盘上空白（尚无棋子）且非特殊禁着点的交叉点。落子无悔，棋子落定后玩家不得再操作棋盘上的棋子。轮至己方时，执子方可选择放弃本回合落子（虚着），同时将落子权力移至对方。若一方主动认输或双方连续选择“虚着”，则代表比赛提前结束，随后按规则判断输赢。

（2）棋子的气

当一个棋子被放置到棋盘的某个交叉点上后，其上下左右四个方向相邻的交叉点称为该棋子的“气”，“气”的单位为“口”。图 2.2 中用“X”表示气，展示了几种常见状态下棋子的气的情况：

1. 黑子 A 四个相邻的位置均为空交叉点，因此该子有 4 口气；
2. 围棋游戏中可将一方棋子放置在另一方棋子直接相邻的交叉点，从而堵住或填掉该棋子的气，图中黑子 B 的 4 个直接相邻交叉点中有两个位置被白子填占，因此该子只有 2 口气；
3. 当多个同色棋子直线相连时，这些棋子将被视为一个“块”。一块棋的气是共用的。图中黑子 C 与两颗同颜色棋子相连，形成一块棋，因此该棋块作为一个整体拥有 7 口气。反观黑子 D 和黑子 E 不直接相连，因此不构成“块”，两子的气相互独立，各为 4 口；
4. 黑子 F 棋子位于棋盘边线，因此在棋盘上最多只有 3 个与其直线相邻的交叉点，即 3 口气；
5. 黑子 G 位于棋盘的角上，因此在棋盘上最多只存在 2 个与其直线相邻的交叉点，即 2 口气；

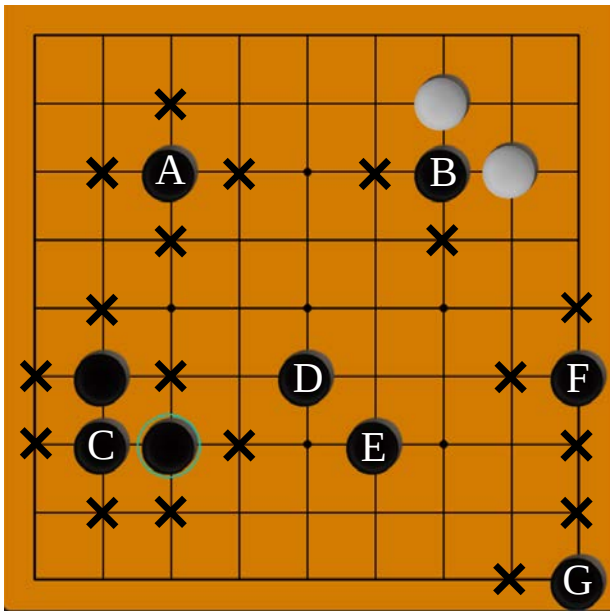


图 2.2 不同状态下棋子的气

（3）禁着点

基本下法中提到，双方不能将棋子下在“禁着点”。围棋中，禁着点主要分为两种情况：

1. 无气点：若己方棋子落入该点后，该子所在的棋块变为无气状态，即被

敌方棋子包围，同时己方落子后无法提取敌方棋子，则该落子无效，该落子点称为己方的禁着点。

图 2.3 展示了两种需要辨别的禁着点情况。对于白方而言，A 点为禁着点，因为白方棋子放入后与其上方相邻白子组成棋块，该块棋被黑子包围，呈无气状态；而 B 点非禁着点，因为在围棋规则中，一方落子后，对地方棋子状态的判定先于对己方的判定，图中尽管白棋落入后呈无气状态，但同时也可以使周围的黑子无气，因此在棋盘状态判定时，黑子先被提出，使得白棋获得新的“气”而存活。

2. 劫争点：己方棋子放入后使得对手面临其上一回合落子时相同的局面，则该落子无效，该落子点为己方的禁着点。

该情况在围棋中对应着“劫争”的规则，图 2.4 展示了“劫争”的基本结构。对于左方棋面，黑子可以落在 A 点，从而提掉一颗白子，得到右方棋面。此时若白棋反手立刻落在 B 点将上一步黑棋吃掉，则会导致下一回合的局面与上一回合完全相同，这种双方在可以同一位置反复吃掉对方一颗棋子的情况称为“劫”，黑白双方争夺“劫”所在位置的情况称为“劫争”。而为了避免黑白双方在此情况下陷入循环，围棋中特别规定“劫争”后的一回合中，“劫”所在位置为禁着点。

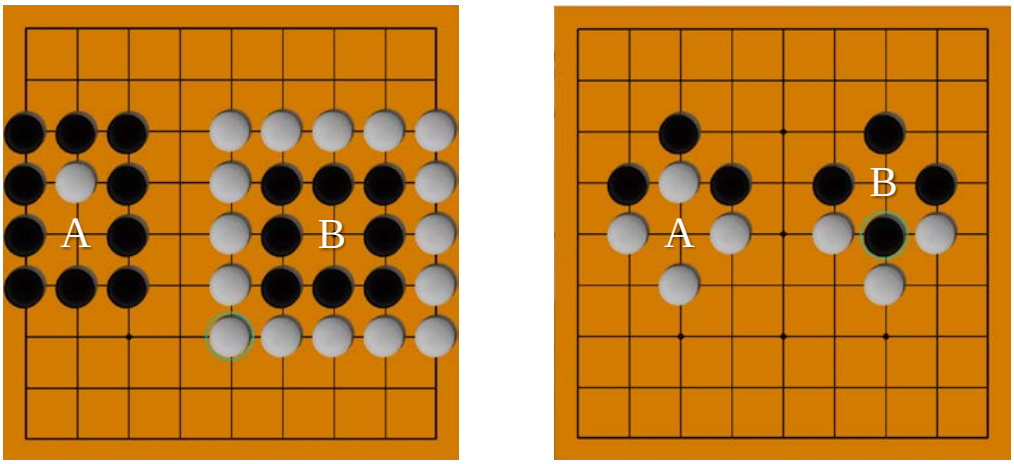


图 2.3 禁着点情形

（左为“无气”情况下的禁着点辨别，右为 “劫争”的基本情形）

（4）真眼与假眼

围棋中，尤其是死活问题中，输赢的根本准则在于对棋块的“真眼”个数的判断。因此在介绍死活问题前，还需介绍“眼”的概念。

图 2.5 展示了“眼”的基本情形。棋盘中，若一个空交叉点四周所有直线相邻的交叉点上均有同色棋子，则该空交叉点称为该颜色棋子的“眼”。由定义，图中交叉点 A、B、C 均为黑子的“眼”。

图中“X”表示“眼”的角位，称为“眼角”。对于“真眼”、“假眼”的定义为：若该“眼”（如点 A）在棋盘中央（含四个“眼角”），且该“眼”所属方（黑或白）需同时占据三个“眼角”，则该“眼”为“真眼”，反之则为“假眼”。同理对于位于棋盘边线上的“眼”（如点 B）和位于棋盘角落上的“眼”（如点 C），则需占据所有的“眼角”才为真眼，否则为“假眼”。

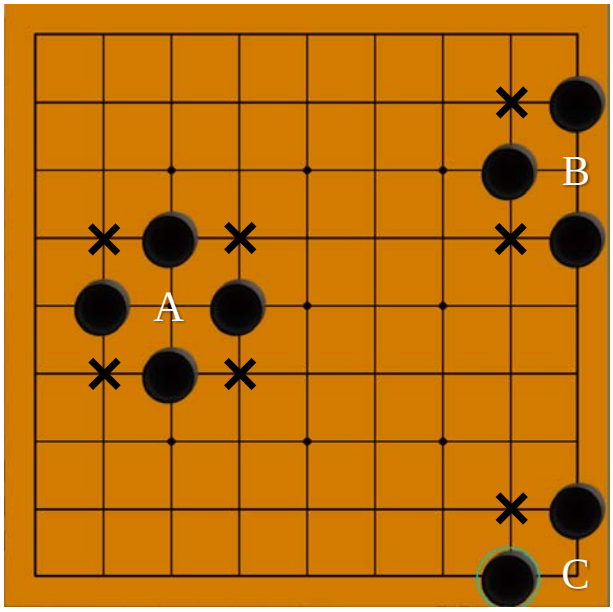


图 2.4 “眼”的基本情形

（二）围棋死活

1.死棋与活棋

死活题的最终结果通常可以被分为两大类：

1. 净杀/净活：双方以无条件达成杀棋或是活棋为目标，此时不允许最终存留“劫争”问题，当出现“劫争”时对双方而言都算作失败；
2. 劫杀/劫活：双方可以通过构造“劫争”的方式来实现杀棋或者活棋的目标，此时最终结果允许存在“劫争”情形；

在死活题游戏中，“净杀/净活”是双方玩家的首选目标，其结果优于“劫杀/劫活”，相应的，其对玩家的落子要求更高。若一道题目存在“净杀/净活”的解法，那么当玩家仅达成“净杀/净活”也算作失败。本文的目标是通过算法实现一个可以充分解决围棋死活题的智能体，我们希望它可以实现最优解、达到更高的游戏要求，因此本文中，我们采取更为严格的“净杀/净活”规则作为判定死活题输赢的准则。

对于以“净杀/净活”为要求的死活题，活棋棋形的判定可以用围棋语言进一步阐明——对一块棋而言，若该棋块内部拥有两个“真眼”，或拥有互为“眼角”的两个“假眼”，则该棋块为活棋，即另一方再无法满足吃子的条件。图 2.6 展示了活棋的常见情形，其中黑棋块 A、B 满足“拥有两个真眼”的条件，黑棋块 C 则满足“拥有两个互为眼角的假眼”的条件，因此图中黑棋块 A、B、C 均构成活棋。

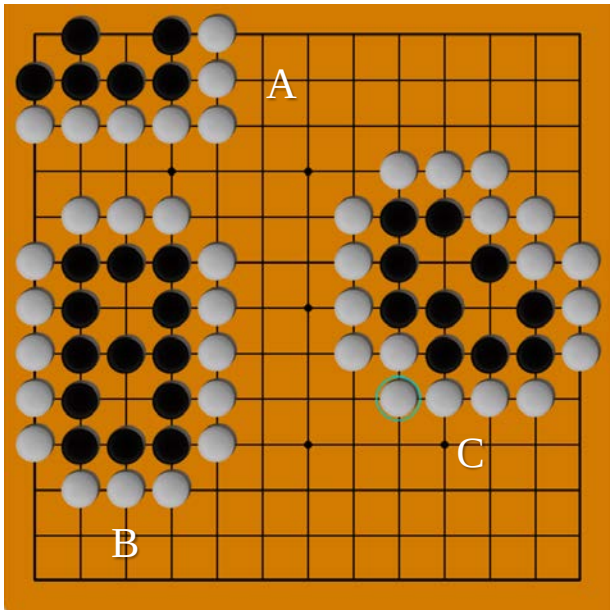


图 2.5 活棋的常见情形

2.围棋死活题

围棋死活题，是指在一个给定初始棋局内，双方以杀死对方棋子或救活己方棋子为目标的围棋题目。图 2.7 展示了几道死活题的初始棋面。

求解死活题，需要计算己方的正确着手，以及预测对方所有的有效抵抗，同时得出目标棋子在双方最优落子情况下的死活状态，从而推导出“正解”。死活题的正解，就是指在求解该题过程中，所有双方均以最优落子策略进行应对的博弈过程。

死活题目通常会规定先手方以及对黑白两方的要求，围棋选手通常会将自己代入其中一方并根据题目要求，实现己方活棋或者组织对方活棋，实现“净活”或“净杀”。本文中将目标为“净杀”的一方称为“进攻方”，目标为“净活”的一方称为“防守方”。

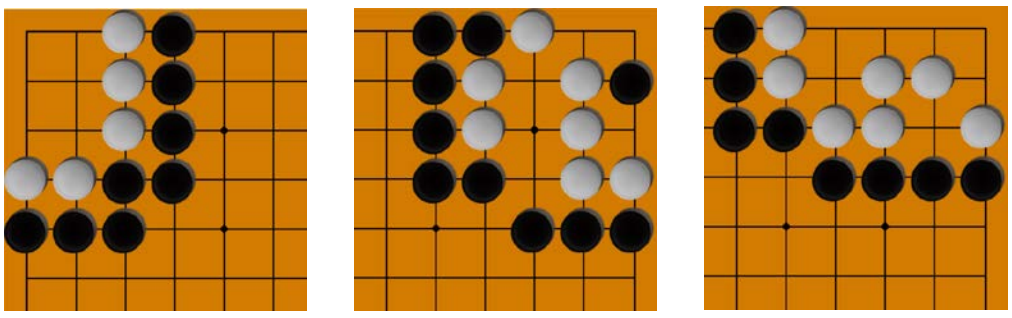


图 2.6 围棋死活题初始棋面

三、强化学习

（一）强化学习介绍

强化学习是机器学习的子领域，受生物可以自适应环境变化的启发而来[11]。强化学习的核心是让智能体以试错的方式和环境发生交互，同时接受环境的奖励反馈，并在反复的交互中优化自己的动作策略从而最大化奖励[12]，因此强化学习的目标通常是得到一个基于环境状态的最优动作策略，从而最终实现某种目标结果，比如赢得围棋游戏。强化学习属于无监督学习，因此它不需要预先给定的训练数据，而是在试探环境、评价动作的过程中不断学习。

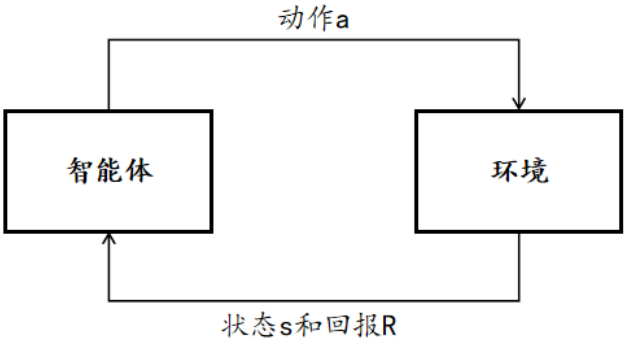


图 3.1 强化学习简单流程示意图

1.强化学习的基本要素

基于蒙特卡洛方法的强化学习模型主要包含以下七个基本要素：

1. S：表示环境的状态，t时刻环境的状态记作S_t；
2. A：表示智能体采取的动作， t时刻智能体采取的动作记作A_t；
3. R：表示交互过程环境给予的奖励，t时刻智能体在状态S_t中采取动作A_t则在$t+1$时刻得到回馈的奖励，记作R_{t+1}；
4. π：表示智能体的动作策略，即智能体会依据策略π来选择动作。通常动作策略可以表示为一个条件概率分布$\pi(a|s)$, 即在状态s时采取动作a的条件概率，其计算公式为：

$$\pi(a|s) = P(A_t = a|S_t = s)$$

5. $V_\pi(s)$：表示给定状态S和策略π后采取行动后的价值，动作价值函数通常可以表示为关于当前的延时奖励和后续的延时奖励的期望函数，其计算公式为：

$$V_\pi(s) = \mathbb{E}_\pi(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s)$$

6. γ：表示奖励衰减因子，通常在[0，1]之间，作为后续延时奖励的权重；

7. ϵ：表示算法探索率，通常在[0，1]之间，主要用于强化学习训练迭代过程中解决平衡“探索与利用(Exploration and Exploitation)”问题。ϵ越接近 1，则智能体更偏向于探索，反之则更偏向于利用已知经验；

2.强化学习的两类方法

强化学习的模型大致可以分为基于模型的强化学习方法和不基于模型的强化学习方法，两者的主要区别在于是否已知环境状态在智能体动作影响下的状态转移概率，若已知则为有模型强化学习，反之则为无模型强化学习。基于模型的强化学习方法的核心是马尔科夫决策过程，然而在很多实际问题中，我们无法预知环境如何随动作改变，因此基于模型的强化学习方法局限性较大。在计算机围棋中，更常用的方法是不基于模型的强化学习，而其中具有代表性的就是基于蒙特卡洛方法的强化学习。

3.强化学习的两类问题

强化学习主要为了解决两类问题。针对不基于模型的的强化学习，我们利用强化学习基本元素进行描述，两类问题分别是：

1. 预测问题：给定强化学习的 5 个基本要素——状态集S, 动作集A, 即时奖励R，衰减因子γ, 给定策略π，求解该策略的状态价值函数$V(\pi)$；
2. 控制问题：给定强化学习的 5 个基本要素——状态集S, 动作集A, 即时奖励R，衰减因子γ, 探索率ϵ, 求解最优的动作价值函数Q和最优策略π；

在本文中，我们关注的是让智能体在围棋死活题中找到最优落子策略，符合两类问题中的控制问题定义，因此也证明强化学习从理论上适用于解决围棋死活题。

（二）基于无模型的强化学习方法

1.前言

蒙特卡洛方法，又称随机抽样或统计试验方法，是一种基于概率统计理论的常用数值计算方法的计算机随机模拟方法[13,17]。其原理为，当所要求解的问题是某种事件出现的概率，或者是某个随机变量的期望值时，可以通过“试验”的方法，得到这种事件出现的频率，或者这个随机变数的平均值，并用它们作为问题的解。针对强化学习中的两类问题，我们也可以采用重复实验的思想。

2.蒙特卡洛方法

在不基于模型的强化学习中，由于智能体不知道环境状态的转移概率，因此唯一的方法是让智能体与环境不断交互，收集大量样本数据，包括经历的状态、采取的动作和收获的奖励，直到达到终点（比如围棋终局），此过程称为蒙特卡洛采样。每一次蒙特卡洛采样都会得到到一个完整的状态序列：$S_1, A_1, R_2, S_2, A_2, \dots S_t, A_t, R_{t+1}, \dots R_T, S_T$。随后我们基于强化学习思想，根据该状态序列对动作价值函数和策略进行更新迭代，并利用更新后的策略让智能体再次交互，获得新一轮采样数据。如此循环往复直到策略收敛，即得到最优策略。蒙特卡洛方法求解强化学习控制问题的算法流程如算法 3.1。

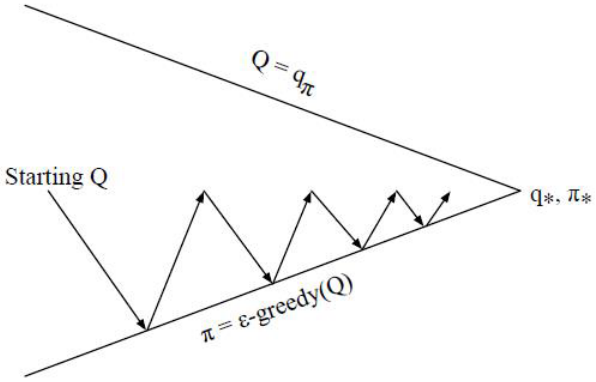


图 3.2 蒙特卡洛强化学习方法示意图

算法 3.1 基于蒙特卡洛的强化学习算法

输入：状态集合 S，动作集合 A，即时奖励 R，衰减因子γ ，探索率 ϵ

输出：最优动作价值函数 $q*$和最优策路 $\pi*$

1. 初始化所有的动作价值 $Q(s,a)=0$，状态次数 $N(s,a)=0$，采样次数 $k=0$，随机初始化一个策略 π
2. 循环：所有$Q(s,a)$未收敛
3. $k=k+1$
4. 蒙特卡洛随机采样得到完整序列$S_1,A_1,R_2,S_2,A_2,\dots S_t,A_t,R_{t+1},\dots R_T,S_T$
5. 循环：$t<=T$
6. $G_t=R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\cdots\gamma^{T-t-1}R_T$
7. $N(S_t,A_t)=N(S_t,A_t)+1$
8. $Q(S_t,A_t)=Q(S_t,A_t)+\frac{1}{N(S_t,A_t)}\big(G_t-Q(S_t,A_t)\big)$
9. 更新ϵ-贪婪策略，$\epsilon=\frac{1}{k}$
10. 更新动作策略，$\pi(a\mid s)=\begin{cases}\epsilon/m+1-\epsilon & \text{if } a^*=\arg\max\limits_{a\in A}Q(s,a)\\ \epsilon/m & \text{else}\end{cases}$，其中$m$为可选动作个数
11. 返回 $\pi(a\mid s)$

（三）基于模拟搜索的强化学习方法

1.前言

基于蒙特卡洛方法的强化学习实质上就是利用大量的实验拟合一个动作选择策略。而随着动作策略的收敛，环境状态受动作影响的变化也会趋于固定，我们可以理解为在收获最优动作策略的同时也收获了一个相对固定的状态转移概率。因此我们继而考虑将基于模型和不基于模型的强化学习算法相结合，让不基于模型的强化学习为基于模型的强化学习提供先验知识的指导。其中一种非常流行的结合强化学习方法就是基于模拟的搜索。

基于模拟搜索的强化学习方法主要分为两个交替进行的阶段——模拟和搜索。模拟阶段可以理解为重复地使用不基于模型的强化学习方法进行采样，获取大量的采样数据。搜索阶段可以理解为利用基于模型的强化学方法，以自身模拟结果作为模型输入，从而得到最佳动作策略，实现长期奖励最大化。

2.经典强化学习问题中的蒙特卡洛树搜索

蒙特卡罗树搜索（MCTS）是基于模拟搜索的代表方法，也是解决经典强化学习问题（包括博弈问题）的常用方法[14]。博弈问题常用博弈树来表示和解决，其基本构造是以初始状态为树的根节点，并根据每个博弈方做出所有可能的动作后得到的新的状态对博弈树进行扩展，直到每一条路径都到达博弈终局状态。这实际上就是对博弈状态的一种贪婪遍历方法，树的每一个节点表示博弈问题的一种状态，每一层的连接代表博弈方的一次动作，树搜索即指从根节点沿着树的某一条分支向叶节点探索的过程。蒙特卡洛树搜索求解传统强化学习控制问题的算法流程如算法 3.2 所示：

算法 3.2 经典强化学习问题中的蒙特卡洛树搜索算法

输入：状态集 S，动作集 A，即时奖励 R，衰减因子 γ，最大采样次数 K

输出：当前状态的最大动作价值Q对应的动作a_t

1. 初始化所有的动作价值 $Q(s,a)=0$，状态次数 $N(s,a)=0$，随机策略 π，$k=0$

2. 循环：$k \le K$

3. $k=k+1$

4. 蒙特卡洛随机采样得到完整序列$S_t, A_t^k, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, \ldots\ldots S_T^k$

5. 返回 K个完整的状态序列集合 $\{S_t, A_t^k, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, \ldots\ldots S_T^k\}_{k=1}^K \sim \pi$

6. 循环：动作a在集合A中

7. $Q(S_t,a)=\frac{1}{N(S_t,a)}\sum_{k=1}^{K}\sum_{u=t}^{T}1(S_{uk}=S_t, A_{uk}=a)G_t^k$，
其中$G_t^k = R_{t+1}^k + \gamma R_{t+2}^k + \gamma^2 R_{t+3}^k + \cdots \gamma^{T-t-1} R_T^k$

8. 返回 $a_t = \arg\max_{a\in A} Q(S_t,a)$

3.围棋问题中的蒙特卡洛树搜索

上述介绍了经典强化学习问题中蒙特卡洛搜索树的算法，状态序列中每一步都有给定的延时奖励。但是在以围棋为代表的零和问题中，中间状态是没有明确奖励的，只有在行至终局时才知道输赢结果从而对前面的动作进行评价和奖励。因此对于围棋问题，我们需要对蒙特卡洛搜索树算法进行步骤和结构上的调整。

由于围棋问题只有在终局时才知道结果，因此我们在基于蒙特卡罗方法的采样序列的结构中去除每一步动作的即时奖励$R_t\ (t<T)$，只关注游戏结束时的最终奖励R_t（T时刻游戏结束），然后通过回溯的方法将最终奖励R_t等价地赋予每一个中间状态。

经过结构和步骤的调整，以围棋游戏为背景的蒙特卡洛树搜索算法的模拟过程主要分为四个步骤：

1. 选择(Selection)：以当前状态作为根节点，从根节点开始，根据*“节点选择策略（Select Policy）”*依次向下选择最优的子节点，直到来到一个“存在未扩展的子节点”的节点。若该节点为博弈终止状态，则转至步骤 4，否则进入步骤 2.
2. 扩展(Expansion)：根据当前可选动作集合为其创建所有对应的子节点，根据*“节点扩展策略（Expand Policy）”*从新创建的子结点中选择一个节点当作当前节点。
3. 仿真(Simulation)：从当前节点开始利用一个*“默认走子策略（Default Policy）”*，不断选择下一步动作，直至游戏终局，得到一个胜负结果（回报）。
4. 回溯(Backpropagation)：将模拟的胜负结果等信息（包括终局回报）保存在当前叶结点上，并通过向上回溯对父节点保存的相关信息进行更新。

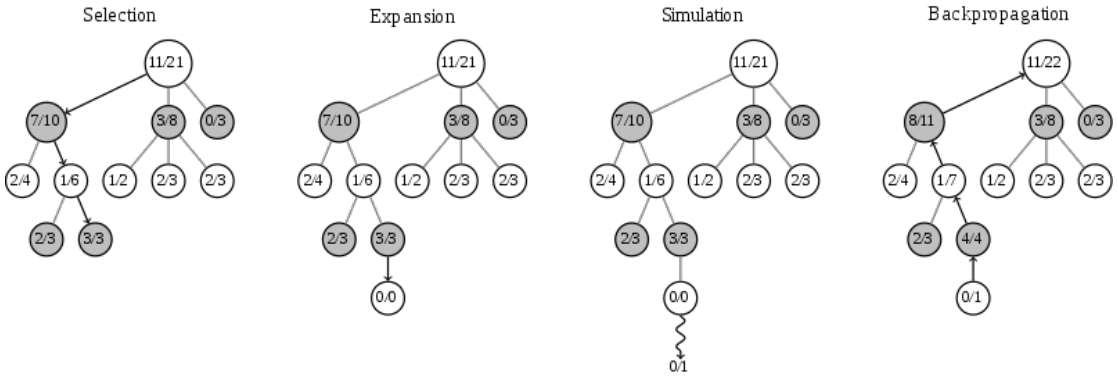


图 3.3 基本的蒙特卡洛树搜索模拟流程（图源自维基百科）

经过有限时间或有限次数的模拟后，算法将根据*“节点选择策略”*选出当前状态（根节点）下的最优动作（探索次数最多的子节点）作为真实决策。整体蒙特卡洛树搜索流程如算法 3.1 所示。

算法 3.3 围棋问题中的蒙特卡洛树搜索算法

函数 1：$MCTS(S_0)$

输入：为当前状态S_0创建一个根节点 N_0，且$S_0 = s(N_0)$

输出：当前状态的最佳动作a

1. 循环：在给定的模拟时间/次数内
2. $N_t \leftarrow TreePolicy\ (N_0)$
3. $Reward \leftarrow DefaultPolicy\ (N_t)$
4. $BackUp\ (N_t, Reward)$
5. 返回 $a(SelectPolicy(N_0))$

函数 2：$TreePolicy\ (N)$

1. 循环：直到N为终止节点
2. 若N节点未扩展，$N \leftarrow SelectPolicy\ (N)$

3. 否则，返回$ExpandPolicy(N)$

4. 返回 N

4.基于 UCB 的蒙特卡洛树搜索

（1）前言

上述算法过程中，*“节点选择策略”*、*“节点扩展策略”*、*“默认走子策略”*实际上是三个可调节的函数模块。通过改变这三种策略函数，可以对整体算法实现功能上的调整或性能上的优化。

“节点扩展策略”、*“默认走子策略”*主要决定了算法的搜索效率，AlphaGo 和 AlphaGo Zero 采用的是基于目标价值函数的策略，通过额外训练的深度神经网络（策略网络和价值网络）辅助蒙特卡洛树搜索，为节点扩展策略和默认走子策略提供经验知识，从而利于选择相对正确的节点、加快算法模拟过程，从而加速策略收敛。本文因缺少深度学习训练条件，因此将两种策略都默认为随机策略，即按照均匀概率随机选取一个可选对象。

*“节点选择策略”*主要决定了算法对动作好坏的评估方法，通常是一个基于节点保存信息的计算函数。通过调整该策略可以对最优动作的选取结果产生影响。通常来说，玩家会参考以往搜索得到的经验进行判定，其策略可以是每次都选择以往搜索结果中的最优动作。但由于搜索次数的有限，以往经验的最优解并不一定是全局最优解。在蒙特卡洛树搜索的模拟过程中，“以往经验”即为节点上保存的信息，然而只有被探索过的节点才会更新信息，未被探索过的节点始终维持初值，所以算法有可能会循环陷入某几条固定的选择路径，从而错失一些更好的动作选择。对于依赖历史经验还是广泛探索的策略平衡问题，就是强化学习常面临的“探索与利用(Exploration and Exploitation)”问题。

（2）上限界限信心算法（UCB）

Auer 在 2002 年提出了上限界限信心（Upper Confidence Bound, UCB）算法，该算法为博弈决策类问题定义了一个衡量“探索”价值和“利用”价值的加权和——UCB 值[15]。其中 UCB 值的算法如下：

$$UCB = \overline{V_i} + c\sqrt{\frac{2\ln\sum_i n_i}{n_i}}$$

其中，$\overline{V_i}$表示动作i的历史期望收益（历史回报奖励的平均值），该值代表了“探索与利用”问题中的“利用”价值，即对已有经验的利用；n_i表示动作i的探索次数，$\sum_i n_i$则代表所有动作的探索总数，c为一个探索常数（超参数），c 越大就越倾向广度搜索，c 越小就越倾向深度搜索，$c\sqrt{\frac{2\ln\sum_i n_i}{n_i}}$整体代表了“探索与利用”问题中的“探索”价值。

（3）围棋中的 UCT 算法

UCT(Upper Confidence Bound Applied to Tree) 算法[16]是 UCB 算法与 MCTS 算法的结合，即在 MCTS 的“节点选择策略”中选择拥有最大 UCB 值的动作，

从而解决了蒙特卡洛树搜索算法中的“探索与利用”问题。

结合“围棋中的蒙特卡洛树搜索”的算法步骤和“节点选择策略”、“节点扩展策略”、“默认走子策略”的设置，UCT 算法流程如算法 3.2 所示。

算法 3.4 围棋中的 UCT 算法

函数 1：$UCT(S_0)$

输入：为当前状态S_0创建一个根节点 N_0，且$S_0 = s(N_0)$

输出：当前状态的最佳动作 a

初始化：$Count(N) = 0, Q(N) = 0$

1. 循环：在给定的模拟时间/次数内

2. $N_t \leftarrow TreePolicy\ (N_0)$

3. $Reward \leftarrow DefaultPolicy\_Random\ (N_t)$

4. $BackUp\ (N_t, Reward)$

5. 返回 $a(SelectPolicy(N_0))$

函数 2：$TreePolicy\ (N)$

1. 循环：直到N为终止节点

2. 若N节点未扩展，$N \leftarrow SelectPolicy\_UCB\ (N, c)$

3. 否则，返回$ExpandPolicy_Random(N)$

4. 返回 N

函数 3：$SelectPolicy\_UCB(N, c)$

1. 返回 $\underset{N' \in childrenof(N)}{argmax} \left(\frac{Q(N')}{Count(N')} + c\sqrt{\frac{2\ln \sum_{N' \in childrenof(N)} Count(N')}{Count(N')}} \right)$

函数 4：$ExpandPolicy\_Random(N)$

1. 从节点N对应状态的可选动作集合$A(s(N))$中随机选择一个新的动作a

2. 将动作a对应节点N'添加到N的子节点集合中

3. 返回 N'

函数 5：$DefaultPolicy\_Random(N)$

1. 循环：状态S为非终止节点

2. 从节点N对应的状态$s(N)$的动作集$A(s(N))$中随机选择一个动作a

3. $S \leftarrow f(S, a)$

4. 返回 状态S的奖励回报$Reward$

函数 6：$BackUp(N, Reward)$

1. 循环：节点N不为空

2. $Count(N) \leftarrow Count(N) + 1$

3. $\mathrm{Q}(N) \leftarrow \mathrm{Q}(N) + Reward$

4. $N \leftarrow parentof(N)$

四、算法设计与实现

（一）围棋死活题博弈系统

本文以 aigagror 作者开源的 GymGo 项目为围棋程序内核，基于 pygame 实现了一套支持“人人对战”和“人机对战”的围棋死活题博弈系统——TsumeGo，其中“机器玩家”代表的是基于蒙特卡洛树搜索算法的智能体。本系统实现了完全端对端的学习，即直接输入初始棋盘局面即可自动运行算法，得到最佳落子策略。

开源围棋框架 GymGo 提供了传统围棋游戏的基本行棋规则的函数接口，本文设计的 TsumeGo 在其基础上根据围棋死活题的特殊要求和规则进行了规则上的修改和完善，包括判断对弈终局的逻辑以及判断输赢的方法，从而实现专门适用于围棋死活题的规则框架和 UCT 算法。本章将主要介绍该系统中针对围棋死活问题的蒙特卡洛树搜索算法实现。

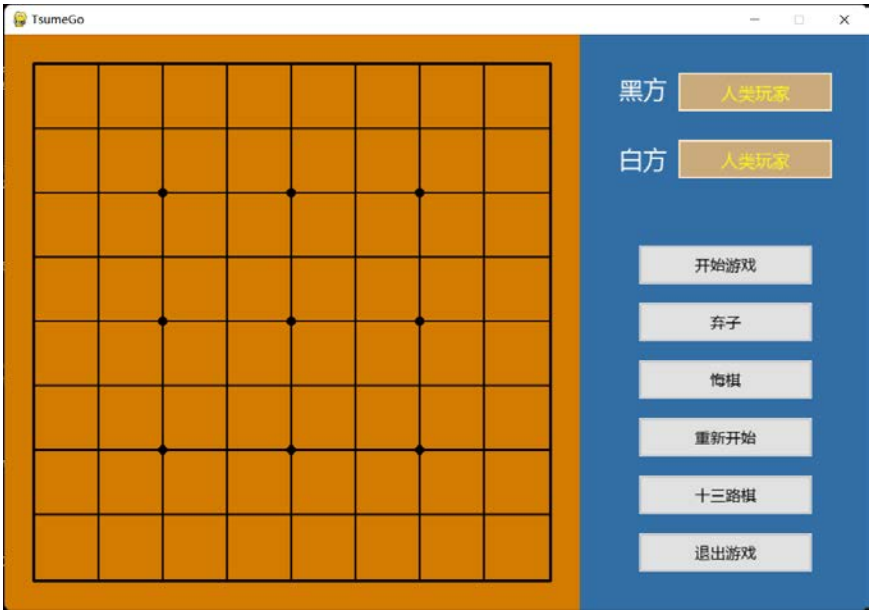


图 4.1 围棋死活题博弈系统

（二）死活问题中的 UCT 算法

本文基于第三章介绍的 UCT 算法实现了针对围棋死活题的 UCT 算法，并根据围棋死活题的一些规则特性做了三方面调整改进，分别是——博弈终局的判断条件、博弈结果的判断逻辑、模拟过程的回报设置。本节将具体介绍上述三方面改进。改进后的完整算法流程可见附录 A。

1.博弈终局的判断条件

在普通围棋比赛中，终局的条件通常为双方连续选择虚着，即双方自知胜负已分，则提前结束比赛；或是棋盘上无有效落子点，即棋盘上仅剩黑白子以及“禁着点”，则自动结束比赛。围棋死活题中，由于双方的最终目标分别是

“净杀”和“净活”，当任意一方达成目标时即可认为游戏结束，达成目标方胜利。因此在死活题中，判断终局的条件除了普通围棋中的两项判断依据外，还应增加对“净杀/净活”条件的判断。

本文所实现的死活题博弈系统中，对“净活”的判断主要通过判断防守方（通常会由题目给定）的棋块是否能构成第二章所介绍的“活棋”，即棋块中含有至少两个“真眼”或者两个互为“眼角”的假眼；对“净杀”的判断则为判断棋盘初始状态时的防守方的棋子是否被全部吃掉。判断游戏终局的完整过程如算法 4.1 所示。

算法 4.1 围棋死活题中的终局判断算法

函数 1：$game\_ended(S)$

1. 若$game_ended_by_over(S)$或$game_ended_by_killed(S)$

   或$game_ended_by_survived(S)$，返回 True

2. 否则，返回 False

函数 2：$game\_ended\_by\_over(S)$

1. 若双方连续虚着或当前棋盘状态无有效落子点, 返回 True

2. 否则，返回 False

函数 3：$game\_ended\_by\_killed(S)$

1. 若当前棋盘状态满足“净杀”，返回 True

2. 否则，返回 False

函数 4：$game\_ended\_by\_survived(S)$

1. 若当前棋盘状态满足“净活”，返回 True

2. 否则，返回 False

2.博弈结果的判断逻辑

任何一个博弈终局的条件满足后，我们就应对输赢结果进行判断。输赢结果基本可以从博弈终局的判断条件中体现出，此处对判断逻辑进行细化。

对于“净杀”或者“净活”提前出现的情况，胜者自然是达成“净杀/净活”的那一方；而对于双方连续“虚着”或棋盘无点可下，即游戏自动结束的情况，本文则默认为双方皆失败。因为此时我们认为黑白双方都没能达成自己的目标，这意味着双方都未能走出最佳的落子步骤、实现最优的题解，且由于游戏结束，双方也再无改变局势的可能。

这表示本文实现的博弈系统中不存在“和棋”，对于死活题来说结果只有“你死我活”和“两败俱伤”。这样设计的意义在于它让死活题中双方的获胜条件更苛刻，从而有利于训练出更专业的智能体。判断博弈结果的完整过程如算法 4.2 所示。

算法 4.2 围棋死活题中的结果判断算法

函数：$Winner(S)$

1. 若$game_ended(S)$:

2. 若$game_ended_by_killed(S)$，返回“进攻方”玩家

3. 若$game_ended_by_survived(S)$，返回“防守方”玩家

4. 否则，返回 -1，代表双方皆败

5.否则，返回 0

3.模拟过程的回报设置

第二章所述的“围棋中的蒙特卡洛树搜索算法”是一种基于模拟的搜索算法，其中模拟过程就是通过蒙特卡洛采样，不断模拟黑白双方的落子，直至博弈终局，并将此过程利用树结构进行表示和信息保存。因此树的每一层即代表着其中一方的一次落子状态，落子方层层交替。大量模拟后，智能体将站在根节点的己方视角，根据模拟的回报结果选择最优动作。

模拟是一个假想的过程，通过假设对方采用和己方相同的落子策略进行博弈，因此整个模拟过程是对称一致的。但由于最终决策时需要区分敌我收益，因此我们仍然有必要通过某种方式在树结构中对双方进行区分。

本文中，我们通过设置模拟过程结束后获得的回报值的正负符号对“敌我收益”进行区分，回报值为正则代表模拟结果“利我”，为负则代表模拟结果“利低”。比如当前状态的下一步落子方为“黑方”，即智能体当前代表的是黑方，若模拟的结果是“黑胜”，则当前叶节点的回报值+1，其父节点则-1，并依次返回更新至根节点；反之，则叶节点的回报值-1，同理依次改变正负符号回溯至根节点。最终智能体选择根节点的具有最大回报值的子节点即可。完整的回报更新算法如算法 4.3 所示。

算法 4. 3 围棋死活题中的回报更新算法

函数：$DefaultPolicy\_Random(N)$

1. 循环：状态S为非终止节点

2. 从节点N对应的状态$s(N)$的动作集$A(s(N))$中随机选择一个动作a

3. $S \leftarrow f(S,a)$

4. $winner \leftarrow Winner(S)$

5. $current\ player \leftarrow$ 当前状态$s(N)$的下一落子方

6. 若$winner = current\ player$，则$Reward = 1$

7. 否则，$Reward = -1$

8. 返回 $Reward$

五、实验与优化

围棋死活题中，“寻找正解”是优先于“赢得比赛”的根本目标，对于大部分简单死活题来说，“正解”中的第一步往往是固定的，而后续的步骤则会根据对手的动作进行改变，也称变招；通常情况下，走对第一步即大概率可以赢得比赛。因此本文基于死活题的核心问题——寻找“正解”，将智能体能否走对第一步作为评估算法效果的标准。

本文利用自开发的围棋博弈系统 TsumeGo 进行实验分析。选取了一系列死活题作为实验用例，利用改进的 UCT 算法对死活题进行求解，并根据解题结果对算法进行评估和优化。

（一）实验结果分析

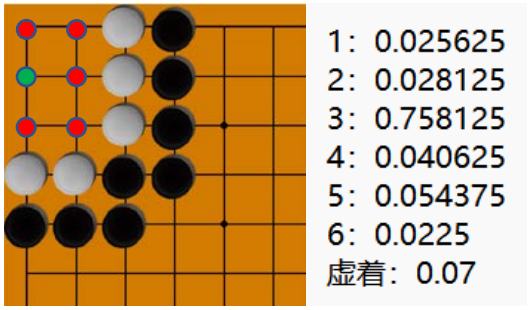
本文从“101 围棋网（https://www.101weiqi.com/book/daquan/）”随机选取了 30 道死活题（初级、中级、高级难度各 10 道），再利用算法 4.4 对每道题目分别进行 50 次求解。本节将算法中的部分参数设置默认值如表 4.1.

表 4.1 UCT 算法默认参数

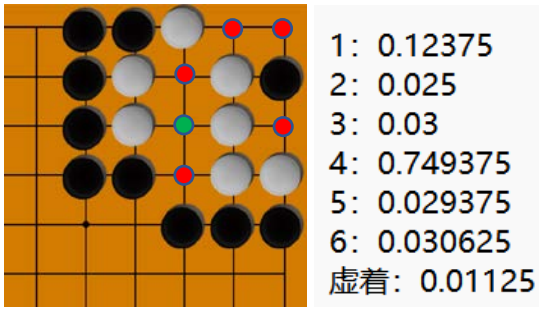
参数名	参数值
模拟次数N	1600
终局回报R	± 1 （赢为 1，输为-1）
探索常数c	2

图 4.1 展示了三道初级、中级、高级死活题示例以及对应的一次求解结果，棋盘中绿点表示 “正解”，红色序号表示其他备选落子点。UCT 算法在做最终动作决策时会根据模拟阶段子节点的探索次数选择相应动作，因此本文给出了算法模拟结束后的每个有效落子点的探索次数占比（所有落子点的探索次数之和为总模拟次数）。

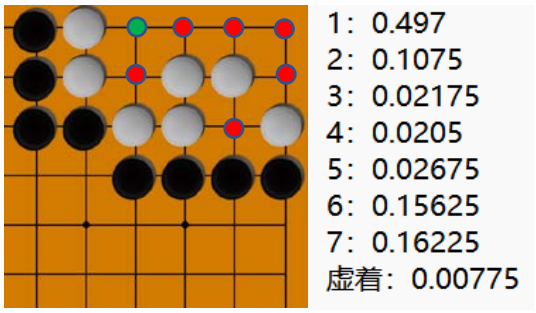
对于 UCT 算法，当模拟达到一定次数后，UCB 值将会逐渐收敛，智能体的动作策略也会趋于固定。因此若“正解”的探索次数占比越大，则代表算法的求解效果越好且越稳定。观察所给初级和中级题目的求解结果，可见“正解”位置被选择的次数占比很大，远超其他落子点，说明算法对该题的求解效果较好；高级题目中占比则相对较低，虽然智能体最终仍会选取该动作，但说明此时 UCB 值尚未收敛或模拟次数不足，鲁棒性较弱。



(a) 初级题 A（白先活白）



(b) 中级题 B（白先活白）



(c) 高级题 C（黑先杀白）

图 4.2 死活题示例及模拟探索结果（棋盘中可选落子点序号将其对应探索次数占比，序号按照落子位置从左至右、从上至下的顺序排列）

本文以算法作为死活题中的先手方（无论黑白），经过多次重复实验后，“正解”被选作首选动作的实验次数占比称为“准确率”；“正解”被视作前三选择动作的实验次数占比称为“次准确率”。比如对一道题进行 50 次求解实验，若其中 40 次算法将“正解”作为首选动作，则该题“准确率”为$40/50 = 0.8$；若其余的 10 次中还有 5 次将“正解”作为第二选择或第三选择动作，则“次准确率”为$(40 + 5)/50 = 0.9$。引入“次准确率”指标是为了增强对蒙特卡洛方法随机性的容忍度。

本文对选取的 30 道不同难度的死活题各进行 50 次实验，结果如表 4.2 所示。可见算法可以较好地解决初级、中级死活题，准确率远超“101 围棋网”上的人类玩家平均水平；此外“平均次准确率”显著高于“平均准确率”，说明在大多数实验中，“正解”都为头部备选动作。

表 4.2 UCT 算法求解结果

难度等级	平均准确率	平均次准确率	平均耗时(s)
初级	81.3%	94.7%	33.8
中级	67.3%	78.0%	44.4
高级	44.7%	53.3%	55.7

（二）UCT 算法优化

结合对围棋死活问题本质和规则的思考，对于本章实现的基于围棋死活问题的 UCT 算法，本节进一步挖掘了一些值得优化的参数或方法，包括 UCB 中的探索常数c、蒙特卡洛树搜索模拟次数、搜索空间、回报函数等。其中探索常数作为平衡“探索和利用”的超参数是一个经验值，通常需要根据具体问题设定，因此本文不对该参数的优化进行详细介绍。本节将具体介绍仅针对围棋死活问题的优化方案。实际实验中，算法的优化是一项参数组合问题，需要进行大量的组合尝试，不同参数组合的部分实验结果可见附录 C。

1.模拟次数优化

蒙特卡洛思想是蒙特卡洛树搜索的核心，通常情况下模拟采样次数越多，所得结果越接近于真实结果。对于围棋死活问题中的 UCT 算法而言，模拟次数越多即代表围棋博弈树的每一个节点都被充分探索，UCB 值趋于收敛，智能体的动作策略也将趋于固定（对某一动作的探索次数占比趋于 1）。在不考虑时间成本和算力成本，且其他参数设置合理的情况下，模拟次数或模拟时间越长则算法效果越稳定。然而实际情形中，围棋游戏往往有落子时间的限制，因此我们也要对其进行一定约束。

本文对算法模拟次数进行调整并对相同死活题用例进行实验，结果如表 4.3 所示。可见随着模拟次数的增加，算法对初级、中级题目的求解准确率明显增加；高级题目的求解准确率变化较小，主要是由于高级题目较复杂、搜索空间大，本文实验的模拟次数还不足支撑对该类难题的充分探索，因此效果稍差。

表 4.3 基于模拟次数调整的 UCT 算法求解结果

难度等级	平均准确率				平均次准确率			
	N=1600	N=2400	N=3200	N=4000	N=1600	N=2400	N=3200	N=4000
初级	81.3%	84.7%	86.7%	88.0%	94.7%	96.7%	97.3%	98.0%
中级	67.3%	68.7%	69.3%	70.7%	78.0%	78.7%	80.7%	81.3%
高级	44.7%	43.3%	45.3%	46.7%	53.3%	54.7%	56.7%	59.3%

2.搜索空间优化

计算机围棋博弈的难点在于其巨大的搜索空间，搜索空间决定了博弈树的广度和深度，更大的搜索空间也意味着需要更多的模拟次数和时间。围棋死活题作为一种局部问题，已相较于传统围棋问题减少了大量搜索空间。然而死活题所提供的初始棋盘依然包含许多无意义的落子点。以图 4.2 所给死活题为例，题目要求“黑先杀白”，根据基本的围棋知识和意识可知，棋盘左下方（红线框出区域）对于黑白双方来说都是无意义的落子点。因此我们在将棋盘状态输入至 UCT 算法时，可以对这些无意义落子点进行标定，将对应动作从算法可选取的动作集中剔除，从而进一步减小搜索空间，充分利用有限的模拟次数，提高算法效果。棋盘表示方法可见附录 B。

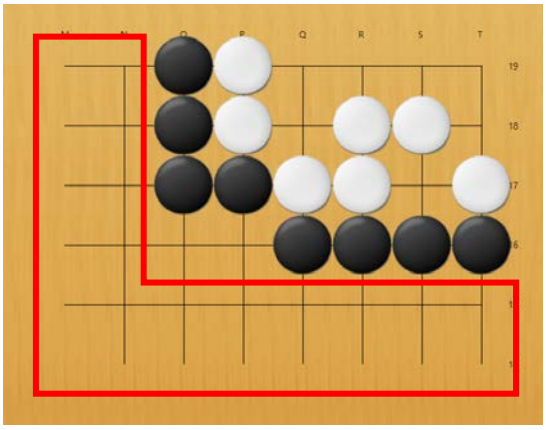


图 4.3 死活题示例（源自“101 围棋网”）

3.回报函数优化

死活题与普通围棋的区别之一在于它不仅对输赢结果有要求，也对玩家的落子步数有一定限制，即在获胜的基础上步数越少越佳。因此在求解死活题时，我们不仅需要找到可以赢得比赛的落子点，更要优中选优，选出能最快结束比赛的落子点。因此在模拟过程中，我们对终局回报值进行调整，添加一项用于衡量步数好坏的数据$k/stepNum$,则终局回报调整为$\pm(1+k/stepNum)$,其中步数回报系数k为常数项，默认为 1，$stepNum$为模拟过程中从游戏开始到游戏终局所花费的步数；当花费的步数越小时，回报绝对值越大，反之越小。

另外由于死活题存在如同标准答案的第一步“正解”，因此我们认为当玩家正确走对第一步后只有极小的概率落败，反之若未走对第一步，则落败的概率变大。因此为了让智能体找准第一步，我们可以进一步对回报函数进行调整，给输掉比赛时返回的回报添加一个惩罚系数p，默认为 3，即获胜时返回$+(1+k/stepNum)$，失败时返回$-p*(1+k/stepNum)$。由于动作价值函数实质上是回报的期望，因此该有利于增加正确动作与错误动作间的动作价值差值。

两项调整的目的都是让 UCT 算法更快更好地找到第一步的唯一“正解”。本文利用改良后的回报函数对相同死活题用例进行实验，结果如表 4.4 所示。

表 4.4 基于回报函数优化的 UCT 算法求解结果

	平均准确率		平均次准确率	
难度等级	$k=0,p=1$	$k=1,p=3$	$k=0,p=1$	$k=1,p=3$
初级	81.3%	84.0%	94.7%	95.3%
中级	67.3%	68.0%	78.0%	84.7%
高级	44.7%	49.3%	53.3%	60.0%

4.衰减因子优化

对经典的强化问题来说，智能体在进行一步动作后会得到一个即时回报R，而对于后续动作的回报会设置一个衰减因子作为延时奖励的权重，这样的设置

是为了减少未来行为对当前动作的影响。对于围棋游戏来说，只有在游戏结束时才得知结果，我们认为终局回报是由最后一步决定的，越远离游戏终局，则动作对最终结果的影响越小。因此本文尝试反其道而行之，在 UCT 算法的回溯过程中引入衰减引子γ，默认为 0.9，即从叶节点反向传播更新父节点的价值函数时，越接近根结点，则更新的回报值越小。

表 4.5 基于衰减因子设置的 UCT 算法求解结果

	平均准确率		平均次准确率	
难度等级	$\gamma = 1$	$\gamma = 0.9$	$\gamma = 1$	$\gamma = 0.9$
初级	81.3%	86.0%	94.7%	97.3%
中级	67.3%	70.7%	78.0%	80.7%
高级	44.7%	47.3%	53.3%	54.7%

五、总结与展望

本文基于蒙特卡洛思想实现了蒙特卡洛树搜索这一强化学习方法，并结合UCB算法实现了UCT算法，解决了强化学习过程中的“探索和利用”问题，并针对围棋死活题的特殊规则和目标对 UCT 算法框架进行了结构上的调整。相较于传统的围棋死活求解方法，本文实现的 UCT 算法不需要喂入围棋死活知识即可达到更高的求解准确率，尤其对简单死活题有非常优秀的求解效果。本文最后提出的对算法优化方案的思考也是具有创造性和启发性的，有助于我们从问题需求和强化学习本质理解算法、改进算法。

本文实现的算法也仍有许多不足和改进空间。首先，本文 UCT 算法的*“节点扩展策略”*、*“默认走子策略”*采用的是随机策略，前期需要大量盲目试错，可以考虑模仿 AlphaGo 和 AlphaGo Zero 训练用于辅助扩展和走子的深度神经网络，为 UCT 算法提供先验知识，从而提升算法的效率和效果；其次，本文仅考虑了以“净杀/净活”为目标的死活题，没有考虑“劫杀/劫活”的情形，后续可以考虑针对“劫争”情况对相关规则函数进行修改。另外，本文选取的实验对象都是规模小于 9*9 的棋局，对于更大规模或者搜索空间更大的复杂死活题，仍需要通过增强算力或采用多线程的方法提升算法性能。最后，对于算法参数的组合仍有非常大的探索空间，未来可以继续挖掘死活题和强化学习的内在联系和规律，尝试寻找一组适用于绝大部分死活题的通用组合解。

附 录

A. 围棋死活题中的完整 UCT 算法流程

算法 4.4 围棋死活中的 UCT 算法

函数 1： $UCT\_Tsumego(S_0)$

输入：为当前状态S_0创建一个根节点 N_0，且$S_0 = s(N_0)$

初始化：$Count(N) = 0, Q(N) = 0$

1. 循环：在给定的模拟时间/次数内

2. $N_t \leftarrow TreePolicy(N_0)$

3. $Reward \leftarrow DefaultPolicy\_Random(N_t)$

4. $BackUp(N_t, Reward)$

5. 返回 $SelectPolicy(N_0)$

函数 2： $TreePolicy(N)$

1. 循环：若$game_ended(s(N))$

2. 若N节点未扩展，$N \leftarrow SelectPolicy\_UCB(N, c)$

3. 否则，返回$ExpandPolicy_Random(N)$

4. 返回 N

函数 3： $SelectPolicy\_UCB(N, c)$

1. 返回 $\underset{N' \in childrenof(N)}{argmax}\left(\frac{Q(N')}{Count(N')} + c\sqrt{\frac{2\ln \sum_{N' \in childrenof(N)} Count(N')}{Count(N')}}\right)$

函数 4： $ExpandPolicy\_Random(N)$

1. 从节点N对应状态的可选动作集合$A(s(N))$中随机选择一个新的动作a

2. 将动作a对应节点N'添加到N的子节点集合中

3. 返回 N'

函数 5： $DefaultPolicy\_Random(N)$

1. 循环：状态S为非终止节点

2. 从节点N对应的状态$s(N)$的动作集$A(s(N))$中随机选择一个动作a

3. $S \leftarrow f(S, a)$

4. $winner \leftarrow Winner(S)$

5. $current\ player \leftarrow$ 当前状态$s(N)$的下一落子方

6. 若$winner = current\ player$，则$Reward = 1$

7. 否则，$Reward = -1$

8. 返回 $Reward$

函数 6： $BackUp(N, Reward)$

1. 循环：节点N不为空

2. $Count(N) \leftarrow Count(N) + 1$

3. $\mathrm{Q}(N) \leftarrow \mathrm{Q}(N) + Reward$

4. $Reward \leftarrow -Reward$

4. $N \leftarrow parentof(N)$

函数 7：$Winner(S)$

1. 若$game_ended(S)$:

2. 若$game_ended_by_killed(S)$，返回“进攻方”玩家

3. 若$game_ended_by_survived(S)$，返回“防守方”玩家

4. 否则，返回 -1，代表双方皆败

5.否则，返回 0

函数 8：$game\_ended(S)$

1. 若$game_ended_by_over(S)$或$game_ended_by_killed(S)$

或$game_ended_by_survived(S)$，返回 True

2. 否则，返回 False

函数 9：$game\_ended\_by\_over(S)$

1. 若双方连续“虚着”或当前棋盘状态无有效落子点,返回 True

2. 否则，返回 False

函数 10：$game\_ended\_by\_killed(S)$

1. 若当前棋盘状态满足“净杀”，返回 True

2. 否则，返回 False

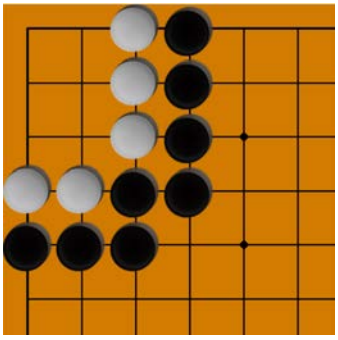
函数 11：$game\_ended\_by\_survived(S)$

1. 若当前棋盘状态满足“净活”，返回 True

2. 否则，返回 False

B. 初始棋盘表示方法

1. 将棋盘手动转换为一个大小为 9*9 的初始矩阵。若棋盘交叉点为黑子，则对应矩阵元素为 0；若为白子，则对应矩阵元素为 1；若交叉点空白且被认为是无意义的落子点，则对应矩阵元素为 2；若被认为有意义，则对应矩阵元素为 3。示例如下：




```
[3, 3, 1, 0, 2, 2, 2, 2, 2],
[3, 3, 1, 0, 2, 2, 2, 2, 2],
[3, 3, 1, 0, 2, 2, 2, 2, 2],
[1, 1, 0, 0, 2, 2, 2, 2, 2],
[0, 0, 0, 2, 2, 2, 2, 2, 2],
[2, 2, 2, 2, 2, 2, 2, 2, 2],
[2, 2, 2, 2, 2, 2, 2, 2, 2],
[2, 2, 2, 2, 2, 2, 2, 2, 2],
[2, 2, 2, 2, 2, 2, 2, 2, 2]
```

2. 程序自动将初始矩阵拆解成一个 Shape 为(NUM_CHNLS, SIZE, SIZE)，每个元素的值均为 0 或 1 的张量(Tensor)表示围棋棋盘状态，其中 NUM_CHNLS 的值为 7，表示张量的通道数，SIZE 表示棋盘的大小（9*9）。每个通道的数据意义如下：
   - CHANNEL[0]：第 0 通道为 BLACK_CHANNEL，表示黑棋棋子分布。有黑棋棋子位置为 1，否则为 0；
   - CHANNEL[1]：第 1 通道为 WHITE_CHANNEL，表示白棋棋子分布。有白棋棋子位置为 1，否则为 0；
   - CHANNEL[2]：第 2 通道为 USEL_CHNL，表示无意义搜索空间。判定为无意义的位置为 1，否则为 0；
   - CHANNEL[3]：第 3 通道为 INVLID_CHANNEL，表示下一步的落子无效位置，包括无意义位置和黑棋、白棋所占位置。无效位置为 1，其余为 0；
   - CHANNEL[4]：第 4 通道为 TURN_CHANNEL，表示下一步落子方，是一个全 0 或全 1 的矩阵。0：黑方，1：白方；
   - CHANNEL[5]：第通道为 PASS_CHANNEL，表示上一步是否为 PASS，是一个全 0 或全 1 的矩阵。0：不是 PASS，1：是 PASS；
   - CHANNEL[6]：第 6 通道为 DONE_CHANNEL，表示上一步落子之后，游戏是否终局，是一个全 0 或全 1 的矩阵。0：未终局，1：已终局。

C.不同参数组合的实验结果（部分）

以附录 A 中死活题为例，将五种参数的备选值进行组合（共 216 种），每种组合实验 50 次。最终将所有参数组合的实验结果依次按照准确率和次准确率进行排序，得到前十名参数组合结果如下：

参数组合					评估指标		
n $\{1600, 2400, 3200\}$	c {1,3,5,10}	k {1,5,10}	p {1,3,5}	γ {1,0.9}	准确率	次准确率	耗时(s)
3200	10	10	3	0.9	96%	100%	59.81
3200	5	1	3	0.9	94%	100%	60.27
3200	3	1	1	0.9	92%	98%	59.73
3200	5	1	5	0.9	92%	96%	59.28
2400	5	5	1	0.9	90%	98%	31.97
3200	10	5	3	1	88%	96%	58.84
2400	10	5	3	0.9	88%	92%	47.79
3200	3	5	1	0.9	88%	94%	59.49
3200	5	5	1	1	88%	92%	38.95
3200	10	5	5	1	86%	90%	58.19