# 基于强化学习的游戏 AI 寻路问题研究

**作者姓名** 王铭

**指导教师姓名、职称** 刘鼎 副教授

**申请学位类别** 工学硕士

中国知网 https://www.cnki.net

学校代码 10701　　学　号 20041211906
分 类 号 TP29　　密　级 公开

# 西安电子科技大学

## 硕士学位论文

# 基于强化学习的游戏 AI 寻路问题研究

**作者姓名：**王铭

**一级学科：**控制科学与工程

**二级学科（研究方向）：**控制理论与控制工程

**学位类别：**工学硕士

**指导教师姓名、职称：**刘鼎 副教授

**学　　院：**机电工程学院

**提交日期：**2023 年 6 月

中国知网 https://www.cnki.net

# Game AI Pathfinding in Dynamic Environment via Reinforcement Learning

A thesis submitted to

XIDIAN UNIVERSITY

in partial fulfillment of the requirements

for the degree of Master

in Control Science and Engineering

By

Wang Ming

Supervisor: Liu Ding Title: Associate Professor

June 2023

中国知网 https://www.cnki.net

# 摘要

游戏 AI 是一款电子游戏的重要组成部分，游戏 AI 设计的好与坏也是评价一款电子游戏是否优秀的重要依据之一。优秀的游戏 AI 能够给玩家提供良好的游戏体验，提高游戏的品质。游戏 AI 如何寻路是游戏 AI 设计中常见的问题。目前传统的游戏 AI 寻路算法有基于启发式搜索的 A* 算法，以及基于导航网格的 NavMesh 算法。这两种算法都是根据算法本身的规则进行计算，在固定的游戏环境中计算得到的路径是固定的。当游戏环境发生变化，例如游戏环境中存在动态障碍物，上述两种算法设计出的游戏 AI 在动态变化环境中寻路的能力较差。

强化学习是重要的机器学习范式和方法论之一，在游戏 AI 的设计中得到了广泛的应用。强化学习受行为主义心理学启发，以一种“试错”的方式进行学习。强化学习智能体与它所处的环境进行交互并获得反馈，根据获得的反馈，更新交互策略以使智能体达到某种既定的目标。零和博弈是博弈论中的一个概念，属于非合作博弈。在零和博弈中，博弈双方是不合作的，博弈双方必定一方有得，另一方有失。

本文将传统搜索算法 A* 算法、强化学习算法 Q-Learning 算法、零和博弈算法 Minimax-Q 算法三种算法相结合，来探索游戏 AI 智能寻路的方法，设计出一种游戏 AI，并以吃豆人作为游戏 AI 寻路的测试环境，提升游戏 AI 在吃豆人游戏中的寻路能力，并以此来提高玩家在游戏中的体验。本文的主要工作内容如下：

(1) 将 Q-Learning 算法与启发式搜索算法结合，提出 Q*-Learning 算法。本文利用 A* 算法计算游戏 AI 搜索地图的最短路径，将游戏 AI 搜索完最短路径所需要的消耗引入到 Q-Learning 算法中的 Q 值更新中，能够使游戏 AI 的 Q 值朝着缩短搜索路径的方向更新，以加快算法收敛速度。同时，本文利用强化学习算法 Q-Learning 进行路径动态规划，将游戏 AI 触碰到动态障碍物的奖励设置为极小，使得游戏 AI 在遇见动态障碍物后会重新规划路径，让游戏 AI 具备传统搜索算法所不具备的，规避动态障碍物的能力。

(2) 将 Q*-Learning 算法与零和博弈算法结合，提出 Minimax-Q* 算法。本文引入零和博弈的博弈论内容，考虑游戏 AI 与动态障碍物进行博弈，赋予了动态障碍物一定的智能。游戏 AI 在更新 Q* 值时，会考虑将动态障碍物的动作收益最小化，从而进一步提升游戏 AI 的规避动态障碍的能力。

(3) 将本文所提出的算法应用在经典游戏吃豆人中。本文利用吃豆人游戏环境设计了三张测试地图和一张应用地图。第一张地图是单纯的搜索任务，吃豆人吃完所有豆子即可完成任务；第二张地图包含一个可移动障碍物鬼怪，吃豆人需要规避鬼怪的路径，并吃完所有豆子；第三张地图，地图面积更大，鬼怪的数量更多。三张地图的



中国知网 https://www.cnki.net

难度依次上升，以便综合评判 Q*-Learning 算法和 Minimax-Q* 算法的性能，以及分析并对比传统的寻路算法，Q*-Learning 算法和 Minimax-Q* 算法所具有的优势。最后一张是由 Unity3D 平台构建的应用地图，将本文算法应用到该地图中，观察游戏 AI 的执行过程并分析其执行结果。

**关键词：**游戏 AI，强化学习，零和博弈，寻路算法



中国知网 https://www.cnki.net

# ABSTRACT

Game AI is an important part of a video game. The design of game AI is also the top priority of video game development. Excellent game AI can improve the quality of the game and provide players with satisfying game experience. How to search the best path for game AI is a common problem in game AI design. The traditional game AI pathfinding algorithms include A* algorithm based on search or NavMesh algorithm based on navigation grid. These two algorithms search path via its calculation rules, and the path calculated by them is always fixed in a static and stable game environment. When the game environment changes, for example, there are some movable obstacles in the game environment, the game AI designed by the above two algorithms has poor performance.

Reinforcement learning is an important machine learning paradigm and methodology, which had been widely used in the game AI design. Enlightened by behaviorist psychology, reinforcement learning is conducted in a ”trial and error” way. Agent, based on reinforcement learning algorithm, interacts with the environment and gets feedback from it. Then, intending to achieve some goals, agent updates its interaction policy through the feedback. Zero sum game is a concept of the Game Theory, which belongs to non-cooperative game. In a zero-sum game, both two sides of the game are uncooperative. It means that only one player can win this game.

This research combines the A* algorithm, the Q-Learning algorithm, and the Minimax-Q algorithm, to explore the intelligent way of game AI pathfinding. In order to improve experience of players in the game, this search will design a game AI, and use Pacman game as the testing environment for game AI pathfinding, which can enhance the pathfinding ability of game AI in Pacman game. The main contents of this research are as follow:

(1) Q*-Learning algorithm is proposed by combining A* algorithm with Q-Learning algorithm. In this study, A* algorithm is used to calculate the shortest path when game AI navigate in a maze, and the cost, required for game AI to search the shortest path, is introduced into the Q value update of Q-Learning algorithm, which can make the converge of game AI training faster. At the same time, this research uses the Q-Learning algorithm to dynamic navigation. This research minimize the reward, when game AI and movable obstacles col-



中国知网 https://www.cnki.net

lide, so that the game AI can replan the path and have the ability to avoid some movable obstacles.

(2) The Minimax-Q* algorithm is proposed by combining Q*-Learning algorithm with Minimax−Q algorithm. This research introduces the game theory of zero-sum game, and gives movable obstacles with certain intelligence. When updating the Q* value, game AI will consider minimizing the reward of movable obstacles(regarded as opponent agents) actions, so as to enhance the ability of game AI to avoid movable obstacles further.

(3) The third part of this research is applying Q*-Learning algorithm and Minimax-Q* algorithm to, a classic game, Pacman. This research has designed four maps of Pacman . The first map is a simple search mission. The Pacman can finish the mission after eating all beans; The second map contains a movable obstacle, the ghosts. Pacman need to avoid the path of ghost and eat all beans; The third map has a larger area and more ghosts. These three maps are used to comprehensively evaluate the performance of new algorithm proposed by this research, and analyze the advantages of them, comparing to traditional pathfinding algorithm. The last map, composed by Unity3D, is used to application, to observe the game AI actions and analyze result of the game.

**Keywords:** Game AI, Reinforcement Learning, Zero-Sum Game, Pathfinding



中国知网 https://www.cnki.net

# 插图索引

图 2.1 怪物 NPC 的决策层有限状态机 . . . . . 10
图 2.2 怪物 NPC 的有限状态机 . . . . . 10
图 2.3 怪物 NPC 决策层行为树结构 . . . . . 13
图 2.4 怪物 NPC 战斗行为树结构 . . . . . 13
图 2.5 怪物 NPC 移动行为树结构 . . . . . 13
图 2.6 怪物 NPC 死亡行为树结构 . . . . . 14
图 2.7 蒙特卡洛树搜索步骤流程[1] . . . . . 14
图 2.8 简单的 ANN 结构示意 . . . . . 16
图 2.9 强化学习框架示意 . . . . . 17
图 3.1 一个游戏回合构成图 . . . . . 25
图 3.2 Q*-Learning 算法流程图 . . . . . 29
图 3.3 游戏 AI 架构图示意 . . . . . 29
图 3.4 吃豆人测试环境 1 . . . . . 32
图 3.5 吃豆人测试环境 2 . . . . . 32
图 3.6 吃豆人测试环境 3 . . . . . 33
图 3.7 Q*-Learning 吃豆人在测试环境 1 的训练过程 . . . . . 33
图 3.8 Q*-Learning 吃豆人在测试环境 2 的训练过程 . . . . . 34
图 3.9 Q*-Learning 吃豆人在测试环境 3 的训练过程 . . . . . 35
图 4.1 Minimax-Q* 算法流程图 . . . . . 41
图 4.2 游戏 AI 架构示意 . . . . . 42
图 4.3 吃豆人游戏测试环境 . . . . . 43
图 4.4 Minimax-Q* 吃豆人在测试环境 1 中的训练过程 . . . . . 44
图 4.5 Minimax-Q* 吃豆人在测试环境 2 中的训练过程 . . . . . 45
图 5.1 智能寻路算法游戏地图 . . . . . 49
图 5.2 吃豆人游戏 AI 架构图 . . . . . 51
图 5.3 Minimax-Q* 算法流程图 . . . . . 51
图 5.4 吃豆人游戏过程一 . . . . . 53
图 5.5 吃豆人游戏过程二 . . . . . 53



中国知网 https://www.cnki.net

# 表格索引

表 3.1 实验配置信息 . . . . . 31
表 3.2 Q*-Learning 吃豆人测试环境 1 的算法运行结果 . . . . . 34
表 3.3 Q*-Learning 吃豆人测试环境 2 的算法运行结果 . . . . . 34
表 3.4 Q*-Learning 吃豆人测试环境 3 的算法运行结果 . . . . . 35
表 4.1 零和博弈基本情形的收益矩阵示意 . . . . . 37
表 4.2 实验配置信息 . . . . . 43
表 4.3 Q*-Learning 吃豆人测试环境 3 的算法运行结果 . . . . . 44
表 4.4 Q*-Learning 吃豆人测试环境 3 的算法运行结果 . . . . . 45
表 5.1 吃豆人游戏 AI 执行环境 . . . . . 52
表 5.2 吃豆人游戏 AI 执行结果 . . . . . 54



中国知网 https://www.cnki.net

# 符号对照表

| 符号 | 符号名称 |
| --- | --- |
| $S$ | 有限状态集合 |
| $F$ | 终止状态集合 |
| $\tau$ | 状态间的变迁集合 |
| $E$ | 表示触发事件集合 |
| $N$ | 有限状态机集合 |
| $C_e$ | 探索常数 |
| $n_i$ | 节点被访问次数 |
| $\mathcal{A}$ | 动作空间 |
| $P_a(s,s')$ | 动作选择概率 |
| $R$ | 环境奖励 |
| $G_t$ | 累计奖励 |
| $\pi$ | 智能体行为策略 |
| $\pi^*$ | 最优策略 |
| $\theta_t$ | 策略参数 |
| $s_t$ | 智能体当前状态 |
| $a_t$ | 智能体动作 |
| $\gamma$ | 折扣因子 |
| $\alpha$ | 更新步长 |
| $V_t(s,a)$ | 值函数 |
| $Q_t(s,a)$ | 动作值函数 |
| $J(\theta)$ | 策略参数的衡量指标 |
| $P$ | 游戏 AI 路径 |
| $C_{n_i}$ | 节点损耗 |
| $F_p$ | 路径损耗 |



中国知网 https://www.cnki.net

# 缩略语对照表

| 缩略语 | 英文全称 | 中文对照 |
| --- | --- | --- |
| AI | Artificial Intelligence | 人工智能 |
| ANTT | Automated Navigation Turing Test | 自动导航图灵测试 |
| AR | Augmented Reality | 增强现实 |
| BT | Behavior Tree | 行为树 |
| DFS | Deep First Search | 深度优先搜索 |
| DRL | Deep Reinforcement Learning | 深度强化学习 |
| FSM | Finite State Machine | 有限状态自动机 |
| Game AI | Game Artificial Intelligence | 游戏人工智能 |
| GT | Game Tree | 博弈树 |
| HFSM | Hierarchical Finite State Machine | 分层有限状态自动机 |
| MC | Monte Carlo | 蒙特卡洛方法 |
| MCTS | Monte Carlo Tree Search | 蒙特卡洛树搜索 |
| MDP | Markov Decision Process | 马尔科夫决策过程 |
| NPC | Non-Player Character | 非玩家角色 |
| PC | Personal Computer | 个人主机 |
| RL | Reinforcement Learning | 强化学习 |
| TD | Temporal Difference | 时序差分 |
| TS | Tree Search | 树搜索 |
| UCB | Upper Confidence Bounds | 可置信上界 |
| VR | Virtual Reality | 虚拟现实 |



中国知网 https://www.cnki.net

# 目　录

第一章 绪论.....1
1.1 引言.....1
1.2 研究背景及其意义.....2
1.3 国内外研究现状.....3
1.3.1 游戏 AI 应用功能.....3
1.3.2 游戏 AI 实现技术.....4
1.4 研究内容与方法.....5
1.5 本文结构.....6
第二章 基础理论知识.....9
2.1 游戏 AI 常用实现技术.....9
2.1.1 有限状态自动机.....9
2.1.2 行为树.....11
2.1.3 树搜索.....14
2.1.4 机器学习.....15
2.2 强化学习理论基础.....17
2.2.1 强化学习基本概念.....17
2.2.2 马尔科夫决策过程.....18
2.2.3 基于值的方法.....19
2.2.4 基于策略的方法.....22
2.3 本章小结.....23
第三章 结合 A*算法和 Q-Learning 算法的游戏 AI 动态避障.....25
3.1 Q-Learning 算法和 A*算法简介.....25
3.1.1 Q-Learning 算法简介.....25
3.1.2 A*算法简介.....26
3.2 结合启发式搜索的 Q-Learning 算法改进.....27
3.2.1 算法基本原理.....27
3.2.2 算法基本实现.....29
3.2.3 算法测试环境.....31
3.2.4 算法测试结果.....33
3.2.5 算法优劣分析.....35
3.3 本章小结.....36
第四章 结合 Minimax-Q 算法的游戏 AI 动态避障优化.....37
4.1 零和博弈和 Minimax-Q 算法简介.....37
4.1.1 零和博弈.....37

中国知网 https://www.cnki.net

4.1.2 Minimax-Q 算法.....38
4.2 结合零和博弈的 Q*-Learning 算法改进.....40
4.2.1 算法基本原理.....40
4.2.2 算法基本实现.....42
4.2.3 算法测试环境.....43
4.2.4 算法测试结果.....44
4.2.5 算法优劣分析.....45
4.3 本章小结.....46
第五章 游戏 AI 智能寻路算法应用.....47
5.1 游戏平台介绍.....47
5.1.1 Unity 游戏设计平台简介.....47
5.1.2 吃豆人游戏地图设计.....48
5.2 游戏 AI 智能寻路算法.....50
5.2.1 吃豆人游戏 AI 寻路策略分析.....50
5.2.2 吃豆人游戏 AI 设计实现.....50
5.2.3 吃豆人游戏 AI 运行过程分析.....52
5.3 吃豆人游戏 AI 运行结果分析.....54
5.4 本章小结.....55
第六章 总结与展望.....57
6.1 总结.....57
6.2 展望.....58
参考文献.....61

中国知网 https://www.cnki.net

# 第一章　绪论

## 1.1　引言

随着时代的发展，许多旧事物伴随着新时代的到来而展现出了新的面貌，游戏也是如此。在过去，游戏一直是充满着争议，因容易让人成瘾而不被相当一部分人所接受；但随着游戏行业的不断发展以及严肃游戏概念的提出，人们对游戏的观念逐渐改变，将游戏称为“第九艺术”。时至今日，作为游戏的硬件载体，GPU 和 CPU 的算力不断提升，助力了游戏行业的飞速跃进。同时，不少游戏的设计中引入了中国传统文化的元素，如《黑神话：悟空》，向全世界展示了中国传统文化的魅力。游戏正在成为一个新的文化载体，为社会文化的多元化发展注入了新的活力。此外，游戏也能做为现实世界向虚拟世界的映射，在游戏世界中完成对现实世界中的模拟和仿真，例如数字孪生技术。也因为其低成本的特性，游戏广泛用于各种软硬件的仿真和模拟中，比如微软的《模拟飞行》游戏。

过去几年，中国的游戏产业发展态势良好。《2021 年中国游戏产业报告》[2](以下简称“报告”) 指出，2021 年，中国游戏市场实际销售收入已达 2965.12 亿元，相比去年收入增长达 178.26 亿元，同比增长 6.4%。虽然 2021 年游戏产业受疫情影响较大，但收入依然保持持续增长，中国的游戏产业依然有着光明的发展前景。随着元宇宙[3]概念的提出，以及包括微软、Facebook、阿里在内的各大科技企业全面推进元宇宙布局，游戏产业的发展也将迎来全新的机遇。近年来，VR、AR、深度学习[4]以及数字孪生等技术蓬勃发展，可实现将现实世界映射到虚拟世界的数字模型，在虚拟世界中构建真实的物理世界，也为游戏研究领域开辟了新的方向。

报告显示国内游戏用户规模 6.66 亿，同比增长 0.22%，用户数量趋于饱和，但伴随着防沉迷新规的落地以及未成年人保护工作的逐渐深化，用户结构将进一步合理。游戏的受众越来越广，不再局限于年轻人这一群体，游戏用户多元化。游戏带有的社交属性，也让受困于疫情的人们不再孤单，拉近了人与人之间的距离。报告中提出，中国移动游戏市场实际销售收入 2255.38 亿元，较去年增收 158.62 亿元，同比增长 7.57%，移动游戏占据着国内游戏市场的主流，总收入占比 76.06%。移动端技术、云游戏技术的发展，降低了用户的使用成本，进一步扩大了游戏受众范围，为游戏用户提供了更加便捷的服务以及与 PC 端游戏截然不同的游戏体验。

在如此多的人力、物力以及新兴技术助力下，游戏行业将会继续以高速、高效的态势发展。所以本文将重点关注游戏领域，以游戏作为载体，主要研究智能体的智能寻路算法及其应用。游戏 AI 寻路是游戏 AI 应用的典型场景，在以往的研究成果当



中国知网　https://www.cnki.net

中，游戏 AI 寻路的研究大多都局限与静态的游戏环境中。随着游戏设计技术的发展，游戏环境变化愈加复杂，由静态的游戏环境发展成动态环境，以往的静态环境中的算法的寻路效果下降，本文研究将设计出一种智能算法来缓解此类问题。

## 1.2 研究背景及其意义

游戏人工智能[5](Game AI, Game Aritficial Intelligence) 技术在游戏设计中扮演着重要的角色，也是游戏相关研究的重点和热点方向。从另一个方面讲，游戏同样也是人工智能研究中一个十分普及和重要的应用领域。游戏是一个灵活、低成本、高效的虚拟平台，为 AI 技术的研究和测试提供了极大的便利。从简单的棋类游戏到复杂的视频游戏，从基于 $\alpha$ - $\beta$ 剪枝的极小化极大算法的深蓝到由 Oriol 等人[6]开发的基于多智体深度强化学习的 AlphaStar，游戏一直助力着 AI 技术研究的创新和突破。同时，AI 技术的创新与突破也推动了游戏设计和研究的进步。由网易伏羲实验室开源的基于深度学习的虚拟智能形象创建工具 MeInGame[7]，可以从单张人像照片中创建长相相似的指定游戏拓扑结构的人物形象，为虚拟世界中构建虚拟人提供了解决方案。由此可见，游戏研究和 AI 研究相辅相成，联系紧密。

游戏 AI 是一个新兴研究领域，在早期的研究工作中，游戏 AI 通常是由树搜索[8](Tree Search) 算法或者其变体设计而成。大部分早期的游戏 AI 研究集中在传统的棋类游戏上，博弈树[9](Game Tree) 是一种常用的棋类游戏 AI 的建模形式，在博弈树上搜索到当前局面的最优解法就是最直接的实现方式。树搜索方法的效果与博弈树的复杂度呈负相关。如果博弈树的复杂度极高，树搜索方法需要消耗极大的计算资源才能得到一个可以接受的结果。从几个常见棋类游戏的复杂度上可以看到，单纯的搜索很难取得好的效果。于是研究者引入极小化极大算法[10]并利用 $\alpha$ - $\beta$ 剪枝[11]($\alpha$ - $\beta$ Pruning) 减少博弈树中所需评估的节点数，开发了国际象棋游戏 AI 深蓝，但此方法无法解决更加复杂的棋类游戏，如围棋。在后来的研究中，研究者引入了蒙特卡洛方法[12](Monte Carlo Method) 和马尔科夫决策过程[13](Markov Decision Process, MDP)，提出了蒙特卡洛树搜索 (Monte Carlo Tree Search, MCTS) 算法[14]。2013 年，基于 MCTS 的围棋 AI 已经可以在让子的情况下击败职业围棋选手。2015 年，DeepMind 的研究团队将上述算法与深度强化学习 (Deep Reinforcement Learning, DRL) 结合开发了 AlphaGo[15]。2016 年，AlphaGo 击败了世界冠军李世石，它的进化版本 AlphaGo Master 击败了当时的世界第一人柯洁。2018 年，DeepMind 团队的 David 等人[16]在 AlphaGoZero 的基础上开发了 AlphaZero。AlphaZero 不再仅仅只是围棋游戏 AI，它是一个更加通用的棋类游戏 AI，精通国际象棋、日本将棋、围棋。AlphaZero 在国际象棋、日本将棋领域击败了当时的世界冠军，而在围棋领域，击败了它的旧版本 AlphaGoZero。2019 年，OpenAI five 和 AlphaStar 横空出世，标志着游戏 AI 的探究场



中国知网 https://www.cnki.net

景已经从简单的棋类游戏转向了更为复杂的 3D 视频游戏，是游戏 AI 研究的重大突破，也进一步引发了游戏 AI 的研究热潮。

研究游戏 AI 是为了设计出更好的游戏。出色的游戏 AI 能够丰富游戏玩家的游戏体验，能让游戏厂商备受赞誉，能够提高游戏的商业价值和社会价值。游戏 AI 可以参与到游戏中，控制玩家或者非玩家角色，并优化其在游戏中的表现。此种功能的 AI 可以帮助游戏开发者进行自动化的游戏测试或者对游戏的整体设计进行评估。游戏 AI 也可以辅助游戏内容生成，做到游戏体验自适应，让不同的玩家体验到不同的游戏内容或者游戏难度。游戏 AI 同样可以进行游戏用户画像，对用户的游戏内行为进行数据采集和分析，以便游戏开发人员优化游戏内容，游戏厂商也能据此变更其营销策略。因此，本文研究的主要目的就是设计一个具有一定智能的游戏 AI，来提升玩家的游戏体验。

## 1.3　国内外研究现状

### 1.3.1　游戏 AI 应用功能

游戏 AI 的研究方向繁多，研究的主体和侧重点也各不相同。如果从游戏 AI 的应用功能来分类，游戏 AI 主要有以下几个研究方向：

(1) 游戏 AI 最主要的研究方向是开发玩游戏的 AI，甚至让 AI 参与到单人或多人游戏中与真实的人或其他游戏 AI 进行博弈，不论是合作博弈或是对抗博弈。从事于该方向的研究者致力于开发一个能够高效地进行游戏的 AI，让 AI 的游戏实力接近甚至超越人类的水平。Adrià 等人[17]提出了一种新型的内在奖励机制，以及一种可以自适应选择策略以优化训练过程的元控制器 (Meta-Controller)，并在这二者基础上开发了智能体 Agent57，有效地改善了强化学习智能体在复杂环境中不能有效探索的问题。Agent57 在 57 个 Atari 游戏中的表现超越了人类。Ouessai 等人[18]提出了一种启发式的蒙特卡洛树搜索算法 EvoPMCTS，参数化游戏的领域知识作为启发因子用以过滤和建模游戏决策空间，并出了一种用于优化阶段的遗传算法为 EvoPMCTS 搜索最佳参数。EvoPMCTS 在 RTS 游戏的多个地图上均有出色表现。

(2)AI 在游戏中的另一个研究方向是游戏的程序化内容生成。研究者利用 AI 辅助游戏内容的设计，在游戏运行时游戏的一部分内容被算法化的创建，而不是被游戏开发者提前设计好。这种设计方式可让游戏内容具备一定的变化性。该研究方向较为成功的案例有《血源诅咒》(Bloodborne, Sony Computer Entertainment, 2015)、《无人深空》(No Man Space, Hello Games, 2016)。

(3) 游戏分析也是游戏 AI 的研究方向之一，玩家建模、对玩家行为进行分析等是该研究方向中比较常见的研究工作，用以创建能够吸引不同受众的游戏内容，提升游



中国知网　https://www.cnki.net

戏用户的游戏体验。陶建容等人[19]针对不同游戏受众群体对 AI 模型的不同可解释性的不同目的，提出了一个由可解释 AI 驱动的多视角游戏作弊检测框架，有效地改善了游戏玩家作弊的问题，提升了绝大多数玩家的游戏体验。游戏用户画像分析是在线游戏提高产品设计品质和游戏营收的重要因素。赵世玮等人[20]提出了一种基于多数据源和多任务学习的玩家流失和玩家付费的预测方法，针对不同数据存在的自身特点不同，采用了不同的模型结构部提取相应数据的特征表示，并且以多任务学习 (Multi-Task Learning) 的方式同时训练和预测玩家流失和付费两个任务，在真实的游戏数据集上取得了显著的效果。

(4) 可置信智能体 (Believable Agent) 研究是游戏 AI 研究领域中新的研究方向，旨在让智能体能够通过基于游戏的图灵测试。游戏图灵测试 (Game Turing Test) 是图灵测试的一个变体，判断者必须正确地猜出某个被观测的游戏玩家是由真实的人类在操控，还是由 AI 操控。Sam 等人[21]通过一种新的自动导航图灵测试 (Automated Navigation Turing Test, ANTT) 解决了如何快速准确地量化拟人性，并演示了在复杂 3D 环境中，ANTT 在导航任务上的有效性。Dongkyu 等人[22]提出了一种基于经验推理架构的可信赖的游戏玩家建模方法，在此基础开发的智能体能够在的游戏场景中战斗，战斗方式与人类相似。

### 1.3.2 游戏 AI 实现技术

如果从游戏 AI 的实现技术或方法的角度来看，游戏 AI 主要有以下几种实现方法：

(1) 基于规则或者脚本实现的游戏 AI，常见的算法有 A* 算法，及其变种 JSP 算法等。谢吉刚[23]提出了一种分层避障 A* 寻路算法，解决了传统的 A* 的寻路算法会随着地图规模增大性能会急剧下降的问题。Harsani 等人[24]将模糊逻辑和 A* 算法结合。模糊逻辑用于确定敌人的行为，采用 A* 算法搜索最短路径，解决了山羊觅食游戏问题。Daniel 等人[25]提出的 JSP 算法是 A* 算法的变种。JSP 算法引入了跳点 (Jump Points) 的概念，对 A* 算法的搜索路径剪枝，不仅能搜索到更短的路径，跳点的引入也极大地提高了 A* 算法的运行速度。此类算法的缺点在于游戏 AI 的行为逻辑比较固定，游戏 AI 在运行时的计算复杂度比较高，在复杂的游戏环境下求解较为缓慢。

(2) 基于形式化方法的游戏 AI，常见的方法有有限状态自动机 (Finite State Mechine, FSM)、行为树 (Behavior Tree, BT) 等。蔡礼权[26]将模糊逻辑与行为树相结合，提出了模糊逻辑推理行为树，并应用到了 ARPG 游戏中验证了其有效性。Ramiro 等人[27]扩展了事件驱动 (Event-Driven) 行为树，提出了三种新型的节点，用于解决视频游戏中多个智能体之间协调的问题。此类游戏 AI 设计方法的缺点在于可移植性较差，



中国知网 https://www.cnki.net

不同行为模块之间的复用度较低，开发效率较低。

(3) 基于博弈论算法的游戏 AI，常见的算法是博弈树搜索算法，如极小化极大算法、蒙特卡洛树搜索算法等。孙一铃[28]将深度强化学习算法 Double DQN 与蒙特卡洛树搜索算法结合，对 Expectimax 搜索算法进行优化，改进其扩展方式和搜索策略，并将改进后的 Expectimax 算法应用到了麻将游戏，取得了比较好的效果。Tom 等人[29]提出了一种改进的蒙特卡洛树搜索算法应用到了吃豆人女士 (Ms Pacman) 游戏中。该算法主要包含四项改进：(1) 可变深度树；(2) 鬼和吃豆人的模拟策略；(3) 包括得分中的长期目标；(4) 使用衰减因子 $\gamma$ 对搜索树进行多次移动。Dennis 等人[30]提出了蒙特卡洛树搜索算法的八项改进技术：累进历史、N-Gram 选择技术、树重用、宽度优先树初始化、避免损失、基于新颖性的修剪、基于知识的评估和确定性游戏检测。基于此算法的游戏 AI 在不同的游戏中的胜率显著提高。树搜索算法再复杂程度低的游戏当中的应用效果比较好。当游戏的复杂程度变高、游戏的状态空间数量较大，树搜索算法的效率将大幅下降。

(4) 基于机器学习的游戏 AI，常见的方法论有强化学习、深度学习等。Volodymyr 等人[31]将强化学习与深度学习相结合，提出了深度 Q 网络 (Deep Q Network, DQN) 和经验回放 (Replay Memory) 技术，在七个 Atari 游戏中都有良好的表现，开辟了深度强化学习领域。John 等人[32]提出的 PPO 算法和 Tuomas 等人[33]提出的 SAC 算法，不仅在视频游戏中有出色的表现，成为了设计深度强化学习游戏 AI 的主流算法之一，还因为其优秀的性能，应用在现实世界的机械臂控制当中。此类游戏 AI 设计算法的缺点是算法的收敛性难以保证。

## 1.4　研究内容与方法

吃豆人游戏 (Pacman) 是电子游戏历史上经典的街机游戏。吃豆人游戏的构成要素比较简单，由吃豆人本体、鬼怪、豆子、能量药丸、迷宫构成。游戏的目标就是控制吃豆人规避鬼怪的攻击并吃掉迷宫中所有的豆子，路径消耗越短，分数越高。吃豆人游戏包含了游戏 AI 寻路、以及游戏 AI 与对手博弈的特性，游戏分数也能直观地体现游戏 AI 的性能，适合作为本文研究的算法测试和应用场景。本文研究的测试和应用游戏场景中的状态都是完全可观的。

A* 算法是比较常见的寻路算法，广泛应用于游戏 AI 的寻路中。A* 算法是一种启发式的最佳优先搜索算法，是深度优先算法 (Deep First Search) 的变种，将起点到某目标点的距离和该目标点到终点的预测距离之和作为启发因子，优先选择距离之和最小的方向进行搜索。A* 算法需要维护两个列表，一是开启列表 (Open List)。用以存放可以研所的节点；二是关闭列表 (Closed List)，用以存放已经探索过的节点。此外，A* 算法还需记录当前节点的父节点信息，以便作路径回溯。



中国知网　https://www.cnki.net

强化学习是重要的机器学习方法论之一，是目前热门的游戏 AI 设计方法。强学习包含五个要素：策略 $\pi$、动作 $a$、状态 $s$、奖励 $r$、环境。智能体在状态 $s$ 下，以动作 $a$ 与环境进行交互，将自身状态 $s$ 更新至 $s'$，观测交互结果得到计算动作 $a$ 的奖励 $r$，根据奖励 $r$ 更新行动策略 $\pi$，根据策略 $\pi$ 选择新动作 $a'$，直至终止状态。根据更新策略的方式不同，强化学习方法可以被分为两类，第一类是基于值的方法 (Value Based Method)，第二类是基于策略的方法 (Policy Based Method)。Q-Learning 算法属于基于值的方法，通过智能体当前动作得到的奖励计算 Q 值并更新当前 Q 表，智能体从 Q 表中选择 Q 值最大的动作执行。

零和博弈 (Zero-Sum Game)[34]是博弈论中的一个概念，属于非合作博弈。零和博弈表示所有博弈方的利益总合为零或者一个常数。在零和博弈中，参与博弈的各方是不合作的，有博弈方获利，必有其他的博弈方遭到损失。在一个有限零和博弈中，参与玩家需要使用一个混合策略。常见的解决零和博弈的方法有极小化极大算法、Minimax-Q 算法。

寻路问题[35]是游戏 AI 设计中最常见的问题，在短时间内计算得到游戏 AI 到达目的地的最短路径是寻路问题的评价指标之一。但随着游戏行业的发展，游戏的内容越来越丰富，游戏 AI 所处的游戏环境也变得越来越复杂。游戏 AI 规避动态的障碍物，在游戏中进行动态寻路也逐渐成为寻路问题的评价指标。传统的基于规则的寻路算法，如基于搜索的 A* 算法、基于导航网格的 NavMesh 算法，都不能很好的解决上述问题。而强化学习算法，如 Q-Learning 算法，在训练初期探索效率较低，算法的收敛速度较慢，有时不能规划出最短的路径。所以本文研究将结合 A* 算法快速计算最短路径的能力以及 Q-Learning 算法对变化游戏环境的适应能力，提出 Q*-Learning 算法，并设计一个能够规避动态障碍物的游戏 AI。同时，考虑游戏 AI 与动态障碍物进行零和博弈的情形，引入零和博弈算法 Minimax-Q 算法，提出 Minimax-Q* 算法以进一步提升游戏 AI 的动态避障能力。最后，用经典游戏吃豆人作为算法测试和应用平台，分析本文所提出的算法与传统算法之比存在哪些优劣。

## 1.5 本文结构

第一章为绪论部分内容，重点介绍了游戏行业的发展现状以及发展趋势等。绪论简要回顾了游戏 AI 的发展历程，简述了游戏 AI 的研究意义以及能为现实社会的发展带来怎样的价值。本文绪论从游戏 AI 的应用功能和游戏 AI 的实现技术两个方面，介绍并梳理了国内外研究游戏 AI 的现状。在绪论的最后部分，重点介绍了本文研究的研究内容以及完成研究内容所用的技术和理论知识。

第二章是理论知识介绍，主要分为两部分内容。第一部分是游戏 AI 常用实现技术：有限状态自动机、行为树、树搜索、机器学习，以及这些技术所依赖的理论知



中国知网 https://www.cnki.net

识，并通过一些简单的例子来说明有限状态机和行为树这两种方法是如何设计游戏AI的。第二部分是强化学习的理论介绍，包含马尔科夫决策过程内容、基于值的方法和基于策略的方法，以及这些方法的理论知识和应用算法。第二章内容是本文研究的理论知识部分，为第三章和第四章中的新算法的提出和实现提供了理论基础。

第三章和第四章是本文的重点内容。第三章是Q*-Learning算法内容介绍，首先简要介绍了A*算法和Q-Learning算法的实现方法，重点介绍本次研究所提出的Q*-Learning算法实现方法以及算法流程。之后，介绍测试Q*-Learning算法所用的吃豆人游戏的地图，分析基于Q*-Learning算法的吃豆人游戏AI的测试结果，以及与传统算法相比，本文所提出的算法拥有的优势。第四章是Minimax-Q*算法内容介绍，首先简要介绍了Minimax-Q算法的实现方法以及零和博弈的内容，重点介绍本文所提出的Minimax-Q*算法实现方法以及算法流程。之后，介绍测试Minimax-Q*算法所用到的吃豆人游戏的地图，分析基于Minimax-Q算法的吃豆人游戏AI的测试结果，以及与传统算法相比，本文所提出的算法存在的优势。

第五章是本文提出的新算法的应用部分内容。首先是Unity3D内容简介，利用Unity3D软件构建一个吃豆人游戏的原始应用场景，拥有更复杂的迷宫和更多数量的鬼怪，鬼怪的行为规则也略有不同。将Minimax-Q*算法应用到新的吃豆人游戏地图上，并对应用的结果进行分析，对算法应用效果的优劣进行评价。

第六章是有关本文研究的总结与展望。本文尝试将传统的A*算法、强化学习算法Q-Learning以及零和博弈算法Minimax-Q相结合来探索游戏AI智能寻路方法，整个研究过程一定存在许多不足之处，本章将会总结这些不足之处，并针对这些不足之处提出一些可行的方法。本章还会分析游戏行业、游戏AI的发展趋势，对未来的游戏和游戏AI的新研究方向提出一些观点和见解，对之后的研究工作作出展望，并为之做好准备。



中国知网　https://www.cnki.net

中国知网 https://www.cnki.net

# 第二章　基础理论知识

本章是本文研究的理论部分，将介绍与游戏 AI 设计相关的理论知识，为之后的算法设计、代码编写、算法性能测试等工作提供理论基础。主要包含以下几个部分：(1) 游戏 AI 常用的实现技术，分为四个小节：有限状态自动机、行为树、树搜索、机器学习。(2) 强化学习理论基础，分为四个小节：强化学习基本概念、马尔科夫决策过程、基于值的方法、基于策略的方法。最后，在本章小结中会对本章所介绍的内容进行简要的概括。

## 2.1　游戏 AI 常用实现技术

### 2.1.1　有限状态自动机

有限状态自动机[36-37]，简称有限状态机，是一种特定行为编辑的方法，属于一种专家系统，是游戏发展早期非常流行的非角色玩家 (Non-Player Character, NPC) 游戏 AI 设计方法。有限状态机是一种数学计算模型，用以表示在有限个状态内，状态间的转移和动作等行为。有限状态机在单个时间单位内只能处于一种状态，当有限状态机受到某种输入的影响，状态之间的变迁触发，有限状态机则会由当前状态向下一个状态转变。有限状态机可以描述一个智能体的状态集合、动作集合以及二者的控制流，适合应用于游戏 AI 的设计。有限状态机可由一组有向图表示，可以被定义成一个六元组：

$$\Sigma = (S, E, F, A, s_0, \tau) \tag{2-1}$$

其中，$s_0$ 表示有限状态集合 $S$ 中的起始状态，$F$ 表示终止状态集合，$\tau$ 表示状态间的变迁集合，$E$ 表示触发事件集合，$A$ 表示动作集合。

有限状态机的设计、实现、调试、可视化都比较简单。然而，在复杂的大型游戏环境中，有限状态机的设计将会变得十分复杂，因此，计算性能是基于有限状态机的游戏 AI 的限制之一。有限状态机的另一个限制是灵活性，一旦游戏 AI 的有限状态机设计完成，则该游戏 AI 的行为将会固定下来，基于该方法设计出的游戏 AI 缺乏一定的自适应性和演化性。上述问题可以通过为有限状态机中的状态变迁添加概率[38]或模糊逻辑[39]来进行一定程度的改善。此外，有限状态机的可移植性和模块化较差，在当前游戏环境设计好的有限状态机往往不能复用到另一个游戏环境中，造成开发效率上的低下。分层有限状态机[40](Hierarchical Finite State Machine, HFSM) 是解决该问题比较好的方法。分层有限状态机将游戏 AI 的行为逻辑进行解耦，分解成多个可复用的不同的行为模块，再通过一个顶层有限状态机对这些模块进行控制，很



中国知网　https://www.cnki.net

好地改善了有限状态机开发效率低下的问题。分层有限状态机可由一个五元组表示：

$$\Sigma_H = (N, E, n_0, \tau, N_f) \tag{2-2}$$

其中，$N$ 表示有限状态机集合，$E$ 表示有限状态机之间变迁触发事件结合，$n_0$ 表示初始状态机，$N_f$ 表示终止状态机集合，$\tau$ 表示变迁集合。

下面利用分层有限状态机设计一种简单的游戏 AI，用以控制游戏中的怪物 NPC。游戏 AI 主要包含两层，一是决策层，二是行为层。决策层主要功能是游戏 AI 根据当前游戏环境的不同，调动行为层的有限状态机执行不同动作，行为层是游戏 AI 的具体行为逻辑。决策层有限状态机的结构如下图 2.1 所示。

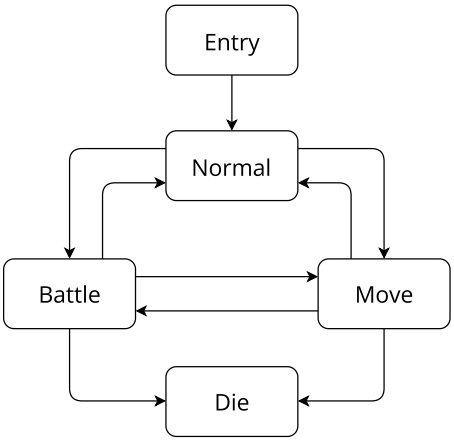
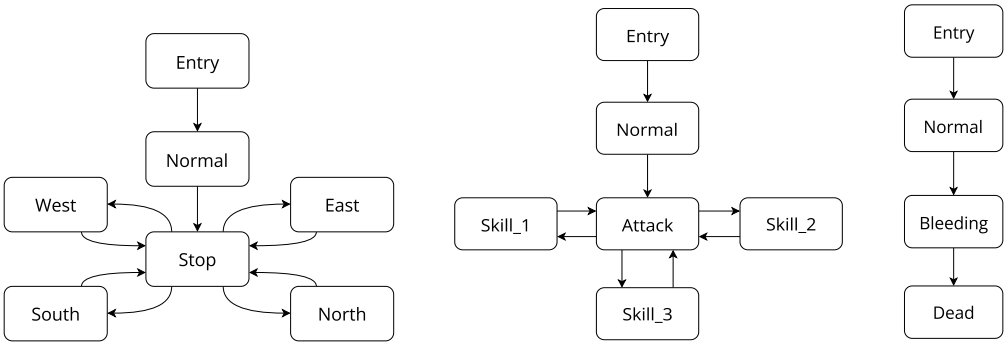



图 2.1　怪物 NPC 的决策层有限状态机

行为层主要包含：战斗模块、行动模块、死亡模块。怪物 NPC 在正常状态下可以调动行动模块，在地图上移动游走；当遇到玩家控制的角色时，可以调动战斗模块与玩家战斗；当怪物的血量耗光时，调动死亡模块。




(a) 行动　　(b) 攻击　　(c) 死亡

图 2.2　怪物 NPC 的有限状态机

行为层的行动模块用于怪物 NPC 的移动行为，包含五个基础动作：向东行、向西行、向南行、向北行、停止。行动模块的有限状态机如图 2.2(a) 所示。



中国知网　https://www.cnki.net

行为层的战斗模块用于怪物 NPC 的攻击行为，怪物由三种不同的攻击方式，分别是：技能 1、技能 2、技能 3。战斗模块的有限状态机如图 2.2(b) 所示。行为层的死亡模块用于表示怪物 NPC 的死亡，包含流血和死亡两个基本动作。死亡模块的有限状态自动机如图 2.2(c) 所示。

所有的有限状态机都包含了一个公共的状态 Normal，便于不同转态间的切换，以及不同有限状态机之间的相互调用。将 Normal 状态置于决策层当中又降低了不同模块间的耦合性。

### 2.1.2　行为树

行为树[41-43]是一种用于描述计划执行的数学模型，广泛应用于计算机科学、机器人学、控制系统以及视频游戏。行为树是一种与有限状态机相似的专家系统，其结构与分层有限状态机相似，同样易于设计、调试、可视化。但与有限状态机相比，行为树的模块化程度更高，可复用性更强。如果设计得当，几个简单的行为树可以组合成逻辑复杂的行为树。行为树以一种树状结构来描述游戏 AI 的复杂行为，底层的叶节点表示游戏 AI 的具体动作，顶层的控制节点用于游戏 AI 的决策控制。事件驱动行为树是传统行为树的变种，通过改变传统行为树节点的内部执行方式以及引入一种新节点来响应事件变化并且中止正在执行的节点，解决了传统行为树可扩展性不足的缺点。

行为树的节点可以分为三类：根节点、控制流节点、执行节点。基于行为树的游戏 AI 的行为逻辑以一种自上而下的方式执行，由根节点开始，由执行节点完成最终动作，口控制流节点主要负责控制执行节点来完成复杂动作。子节点根据动作的执行情况向父节点传递执行状态：成功、正在执行、失败。控制流节点主要分为以下三种：

(1) 选择节点 (Selector Node)：选择节点将会选择第一个没有返回失败状态的子节点执行，当所有子节点都返回失败时，选择节点返回失败。选择节点分为两种，一种是概率选择节点，另一种是优先级选择节点。优先级选择节点的子节点执行顺序依据其重要性进行排序，概率选择节点随机选择其子节点。选择节点执行逻辑的伪代码如算法 2.1 所示。

**算法 2.1** The Selector Node algorithm

```
1: procedure SELECTORNODE
2:     for i ← 1, n do
3:         status ← Signal(ChildNode(i))
4:         if status == running then return ChildNode(i)
5:         if status == success then return ChildNode(i)
6:     return failure
```

(2) 序列节点 (Sequence Node)：序列节点通常用于找到并执行第一个尚未成功执



中国知网　https://www.cnki.net

行的节点, 当所有子节点执行成功，父节点返回成功。序列节点执行逻辑的伪代码如算法 2.2 所示。

**算法 2.2** The Sequence Node algorithm

```
1: procedure SEQUENCENODE
2:     for i ← 1, n do
3:         status ← Signal(ChildNode(i))
4:         if status == running then return running
5:         if status == failure then return failure
6:     return success
```

(3) 修饰节点 (Decorator Node)：修饰节点主要用于强化单个子节点的行为或增加子节点的复杂性，比如为子节点添加执行次数或为子节点限定执行时间等。

从控制理论[44]的角度来看，行为树可以被定义成一个三元组：

$$T_i = (f_i, r_i, \delta_t) \tag{2-3}$$

其中，$i \in \mathcal{N}$ 表示行为树的节点序号，$f_i : \mathcal{R}_n \to \mathcal{R}_n$ 是常差分方程右侧的向量域，表示行为树不同状态之间的映射关系。$\delta_t$ 是时间步长，$r_i : \mathcal{R}_n \to \{R_i, S_i, F_i\}$ 表示返回状态，返回的状态包含：正在执行 $R_i$、成功 $S_i$、失败 $S_i$。行为树的执行可以用如下常差分方程描述：

$$x(t_{k+1}) = f_i(x(t_k)) \tag{2-4}$$

$$t_{k+1} = t_k + \delta_t \tag{2-5}$$

其中，$k \in \mathcal{N}$ 表示离散时间，$x \in \mathcal{R}_n$ 表示行为树模型的状态空间。序列操作符 $\mathcal{S}$ 可以将两个行为树 $T_i$ 和 $T_j$ 组合成一个更为复杂的行为树 $T_0$。

$$T_0 = \mathcal{S}(T_i, T_j) \tag{2-6}$$

行为树 $T_0$ 返回的状态 $r_0$ 及其不同状态间的映射关系 $f_0$ 定义如下：

$$r_0(x) = \begin{cases} r_j(x) & x \in S_1 \\ r_i(x) & otherwise \end{cases} \tag{2-7}$$

$$f_0(x) = \begin{cases} f_j(x) & x \in S_1 \\ f_i(x) & otherwise \end{cases} \tag{2-8}$$

下面将利用行为树对上一节中设计的游戏 AI 进行重构，来展示分层有限状态机和行为树之间的区别与相似之处。利用行为树构建的游戏 AI 的基本行为逻辑保持不变。首先构造一个决策层的行为树，用以调动不同的行为模块供游戏 AI 执行。



中国知网 https://www.cnki.net

决策层行为树的父节点为选择节点，即任意行为模块返回成功状态，则决策层行为树返回成功状态，决策层行为树结构如下图 2.3 所示。决策层行为树包含两个条件节点用以判断怪物 NPC 何时触发移动、战斗、死亡等行为。

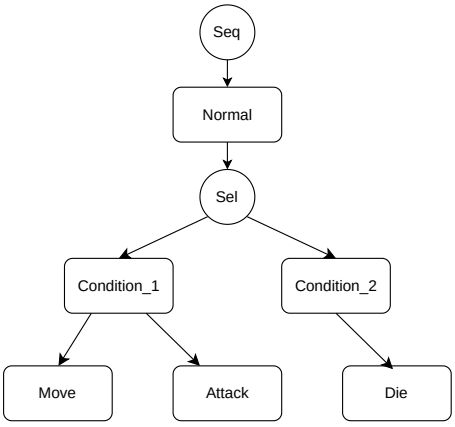



图 2.3　怪物 NPC 决策层行为树结构

战斗行为树包含三个叶节点对应三个可用的攻击技能，三个叶节点的父节点为选择节点，任意一个技能使用成功，则战斗行为树返回成功状态，否则返回失败。战斗行为树的结构如图 2.4 所示。

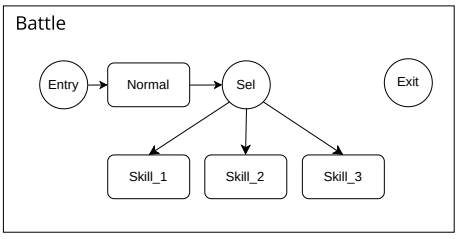



图 2.4　怪物 NPC 战斗行为树结构

移动行为树包含五个叶节点对应四个移动方向和一个停止动作，选择节点对其中任意一个动作进行选择，若存在返回成功的节点，则移动行为树返回成功，否则返回失败。移动行为树的结构如图 2.5 所示。

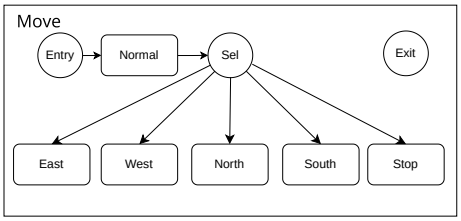



图 2.5　怪物 NPC 移动行为树结构

死亡行为树包含三个叶节点对应两个动作：流血、死亡。两个动作由序列节点控



中国知网　https://www.cnki.net

制，只有流血和死亡都成功时，死亡行为树返回成功。图中 Condition 为条件节点，当怪物血量为零时才触发死亡动作。死亡行为树的结构如图 2.6 所示。

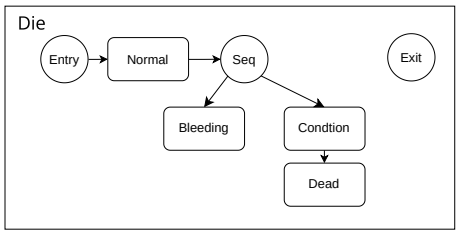



图 2.6　怪物 NPC 死亡行为树结构

### 2.1.3　树搜索

大部分游戏 AI 问题都可以转化为搜索问题，所以树搜索算法也被广泛地应用在游戏 AI 的设计问题当中。树搜索算法主要分为两大类，一类是非启发式搜索算法，另一类是启发式搜索算法。非启发式搜索算法在没有关于搜索目标任何信息的情况下，对搜索目标存在的状态空间进行搜索，主要包含深度优先搜索、广度优先搜索算法等。基本的非启发式搜索算法性能较差，很少直接用于游戏 AI 的设计。启发式搜索算法主要包含 A* 算法、极小化极大算法、蒙特卡洛树搜索算法等。

蒙特卡洛树搜索算法[45]将蒙特卡洛方法应用于博弈树搜索，在游戏 AI 当前状态的决策空间上搜索最佳决策。标准蒙特卡洛树搜索算法使用随机推演 (Rollouts) 来评估当前的游戏状态。在每一个随机推演中，游戏 AI 从某一个状态开始随机地选择游戏动作直到整个游戏结束。每一个随机推演的结果将作为博弈树上节点的权重，权重更大的节点将会被后来的随机推演优先选择。蒙特卡洛树搜索算法的核心迭代过程包含四个步骤：选择 (Selection)、拓展 (Expansion)、模拟 (Simulation)、反向传播 (Back Propagation)，如图 2.7 所示。标准蒙特卡洛树搜索算法伪代码如算法 2.3 所示。

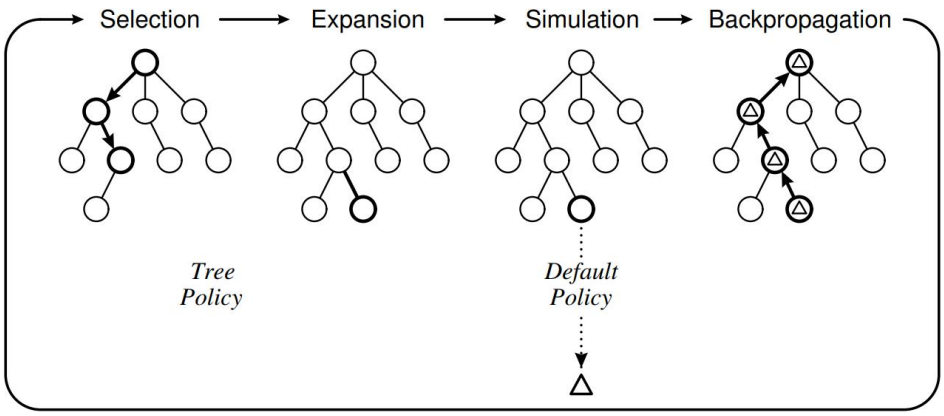



图 2.7　蒙特卡洛树搜索步骤流程[1]

(1) 选择：一般从博弈树的根节点 *R* 开始，依据树策略 (Tree Policy) 连续地访问



中国知网　https://www.cnki.net

其子节点，直至某个叶节点 $L$ 被访问。对应到游戏过程中，选择步骤表示当前游戏状态存在那些动作可供选择，随机选择某一动作执行。根节点表示当前的游戏状态，叶节点表示该节点的子节点还未被某个新的随机推演初始化。

(2) 拓展：如果游戏 AI 执行叶节点 $L$ 后，游戏没有结束，则该叶节点会创建一个新的子节点 $C$，新的子节点表示当前游戏状态存在哪些新的动作没有执行。如果游戏 AI 进入了叶节点 $L$ 对应的游戏状态，游戏 AI 可以选择新生成的子节点 $C$ 作为其下一步动作。

(3) 模拟：游戏 AI 从节点 $C$ 开始随机选择子节点执行，直到游戏进入终止状态，这一步骤也通常称为随机推演。

(4) 反向传播：利用随机推演的结果来更新博弈树上的路径 $C$ 到 $R$ 上每个节点的权重，以便在后续的模拟中能够选择更优的节点。

**算法 2.3** General MCTS Algorithm

```
1: procedure MCTS
2:     Initialize root node n_0 with state s_0
3:     for i ← 1, n do
4:         n_i ← TreePolicy(n_0)
5:         Δ ← DefaultPolicy(s(n_i))
6:         BackUp(Δ, n_i)
7:     return a(BestChild(n_0))
```

选择步骤中的树策略指的是博弈树的节点选择策略，最常见的树策略为 UCB 策略[46](Upper Confidence Bounds)，所选择的节点必须将 UCB 策略的计算公式最大化。除此之外，还有 $\epsilon$-greedy 策略、汤普森采样 (Thompson Sampling) 策略[47]、贝叶斯匪徒 (Bayesian Bandits) 策略[48]。UCB 策略的计算方法如下所示：

$$UCB = \frac{\omega_i}{n_i} + C_e\sqrt{\frac{\ln N_i}{n_i}} \tag{2-9}$$

其中，$\omega_i$ 表示执行当前节点 $i$ 能够获得的奖赏，$n_i$ 表示 $i$ 节点被访问的次数，$N_i$ 表示 $i$ 节点的父节点被访问的次数，$C_e$ 表示探索常数。

### 2.1.4　机器学习

机器学习[49]是人工智能的一个分支。机器学习算法通过采样数据构建一个模型，用来进行预测或者决策。机器学习分为四大类：监督学习[50]、非监督学习[51]、半监督学习[52]、强化学习[53]。监督学习通过一些已标记数据来提取这些数据本身的属性或特征，构建一个从数据到数据特征的映射，并用此映射来预测新数据的属性或特征。根据输入的标记数据的不同，监督学习可以处理三类任务：分类 (Classification)、回



中国知网　https://www.cnki.net

归 (Regression)、偏好学习 (Preference Learning)。常见的监督学习算法非常多，主要有人工神经网络 (Artificial Neural Network)、决策树 (Decision Tree)、随机森林 (Random Forests)、高斯回归、朴素贝叶斯分类器、支持向量机 (Support Vector Machine)、卷积神经网络[54](Convolutional Neural Network) 等。非监督学习通常应用于无标记数据，用以分析这些无标记数据之间的联系或发现它们潜在的特征。非监督学习主要用于数据聚类和数据挖掘，常见的算法包括 k-means 聚类算法、变分自编码器 (Variational AutoEncoder) 等。半监督学习应用的数据既存在有标签，也存在无标签的，是监督学习和非监督学习的结合。

人工神经网络是一种受到生物学启发的计算智能和机器学习方法，用于模拟生物大脑在某些任务中处理信息、操作、学习、执行的方式。人工神经网络的计算过程包括前向操作 (Forward Operation) 和反向传播 (Back Propagation)。人工神经网络的输入是由多种输入共同构成的一个向量 $x$，每一个输入都有一个权重参数，权重参数构成权重向量 $\omega$，输入向量 $x$ 和权重向量 $\omega$ 以内积的方式结合，并添加一个偏置权重 $b$，计算得到权重总和：$x * \omega + b$，最后再输入到一个激活函数 $f$ 中，上述整个过程就是人工神经网络的前向操作过程。人工神经网络的神经元会被划分成多个层级，第一层神经元与外部输入直接相连，被称之为输入层，中间的层次结构被称为隐藏层，最后一层用来输出人工神经网络的计算结果，被称之为输出层。多层感知机 (Multi-Layer Perceptron) 是人工神经网络最简单的形态之一，其结构示意如图 2.8 所示。

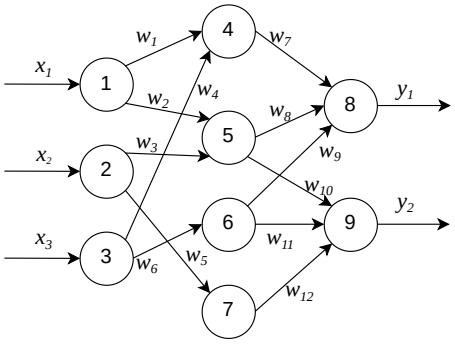



图 2.8　简单的 ANN 结构示意

反向传播[55]是一种基于梯度下降的算法，普遍用于人工神经网络的训练。误差函数 (Error Function) 用于衡量人工神经网络的输出与所期望的标记输出之间的偏差。反向传播会计算误差函数对人工神经网络中的每一个参数的梯度，利用链式法则沿着能够最小化误差函数的梯度方向来对人工神经网络的参数进行优化，直至误差函数更新到一个可接受的范围内。



中国知网　https://www.cnki.net

## 2.2　强化学习理论基础

### 2.2.1　强化学习基本概念

强化学习包含五个要素：策略 $\pi$、动作集合 $A$、状态集合 $S$、奖励信号 $R$、环境，以及一个可选的要素——环境模型，包含状态之间的转移概率等。强化学习模仿了自然界生物“试错”的学习方式，通过构建从环境到动作的映射，让智能体学会如何在复杂的环境中极大化从环境中获得的奖励，使得智能体在复杂环境中能够达到某种既定目标。智能体在状态 $S_t$ 下依据某个策略 $\pi$ 选择动作 $A_t$，观测动作 $A_t$ 的执行结果，获得奖励 $R_t$，根据奖励 $R_t$ 更新自身策略以及状态。强化学习示意图如 2.9 所示。

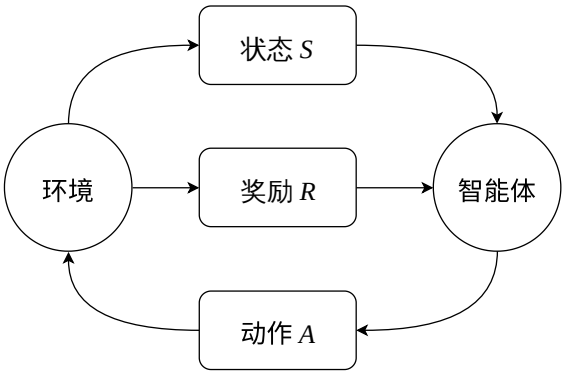



图 2.9　强化学习框架示意

强化学习不存在监督者，但有别于非监督学习，强化学习不利用非标签数据来学习数据之间隐藏的结构。与监督学习不同，标准的强化学习不依赖于标注数据进行训练，训练数据源自于智能体与环境的交互序列，而交互的结果会影响环境的变化，环境的当前时刻状态与下一时刻状态不同。因此，智能体交互得到的数据分布随着时间而变化，不满足独立同分布。在强化学习中，智能体的动作选择不是受迫的，智能体没有被告知下一步的正确动作，它的动作选择仅受环境奖励影响和指导，通过不断地尝试以获得有利动作。智能体需要探索未知的环境以及新的动作来获得潜在的更高的奖励，并利用已获得的经验来判断哪个动作是更加得利的，平衡智能体的探索 (Exploration) 和利用 (Exploitation)[56] 也是强化学习的关键问题之一。环境奖励是一个延迟奖励，只有过了一段时间之后环境才能告诉智能体，之前的行动是否有效。“试错学习”和延迟奖励也是强化学习的两大特征。另外，离线强化学习[57] 引入了监督学习的思想，利用专家知识数据对强化学习模型进行预训练，推动了强化学习的落地，为强化学习领域开辟了新方向。

强化学习问题的形式化利用了动态系统理论中的思想，尤其是不完全马尔科夫决策过程的最优控制，具体形式化细节将在下一节论述。马尔科夫决策过程旨在以最简单的形式包括这三个方面，即感觉、行动和目标，而不割离其中任何一个部分，适



中国知网　https://www.cnki.net

用于形式化强化学习问题。

### 2.2.2 马尔科夫决策过程

马尔科夫决策过程[58]是一种离散时间随机控制过程，为随机决策建模或者由决策者控制的决策建模提供了一种数学框架。马尔科夫决策过程是对马尔科夫过程(Markov Process)和马尔科夫奖励过程(Markov Reward Process)的改进，马尔科夫过程是拥有马尔科夫性的随机过程，而马尔科夫奖励过程是在马尔科夫过程的基础上添加了奖励函数，马尔科夫决策过程是在马尔科夫奖励过程的基础上添加了决策动作，来控制状态之间的转移。

预测和控制是马尔科夫决策过程的核心问题，通常通过动态规划来解决一些与马尔科夫决策过程相关的优化问题。在每一个时间步长中，某个过程处于状态 $s$ 中，决策者从状态 $s$ 的可选动作集合中选择动作 $a$，该过程在接下来几个时间步长中由状态 $s$ 转移到状态 $s'$，并且给予决策者奖励 $R_a(s,s')$，状态之间的转移概率由状态转移方程决定 $P_a(s,s')$。该过程的状态之间的转移满足马尔科夫性，即状态 $s'$ 仅仅只依赖于当前状态 $s$ 和动作 $a$，不受状态 $s$ 和动作 $a$ 之前的状态和动作影响。马尔科夫决策过程可以被定义成一个四元组：

$$M = (\mathcal{S}, \mathcal{A}, P_a, R_a) \tag{2-10}$$

其中，$\mathcal{S}$ 表示状态集合，称之为状态空间；$\mathcal{A}$ 表示决策者的动作空间，$\mathcal{A}_s \in \mathcal{A}$ 表示在状态 $s$ 下决策者可选择的动作空间；$P_a(s,s') = P_r(s_{t+1} = s' | s_t = s, a_t = a)$ 表示状态 $s$ 和动作 $a$ 在 $t$ 时刻下转移到 $t+1$ 时刻下的状态 $s'$ 的概率；$R_a(s,s')$ 表示状态 $s$ 转移到状态 $s'$ 获得的即时奖励。

马尔科夫决策过程的目标是为决策者找到一个良好的策略 $\pi$，使得决策者依据策略 $\pi$ 选择的动作能够最大化累积奖励，并使用折扣因子来调整即时奖励和延时奖励对整个累积奖励的影响。策略 $\pi$ 表示动作空间 $\mathcal{A}$ 的概率分布，既可以是确定的，又可以是随机的。策略 $\pi$ 下累积奖励的值函数定义如下：

$$V_{t+1}^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{t} \gamma^t R_{a_t}(s, s')\right] \tag{2-11}$$

将等式 2-11 改写成值函数 $V_t^{\pi}(s)$ 迭代的形式，结果如等式 2-12 所示。

$$V_{t+1}^{\pi}(s) = R_{\pi(s)}(s, s') + \gamma \sum_{t=0}^{t} P_{\pi(s)}(s,s') V_t^{\pi}(s) \tag{2-12}$$

其中，$\gamma \in [0, 1]$ 表示折扣因子，一般来说折扣因子的大小接近 1，折扣因子越小，决策者越重视即时奖励，因为对未来马尔科夫决策过程的模型的评估不一定准确，存在



中国知网 https://www.cnki.net

一定的不确定性，折扣因子能够体现这种不确定性。在当前状态下，策略的优劣通过值函数的大小来比较。一般来说，在标准马尔科夫决策过程中，每一个状态都包含一个最优策略 $\pi^*$ 能够最大化累计奖励的值函数。

等式 2-12 可以改写成值函数 $V_{t+1}^{\pi}(s)$ 对状态空间 $\mathcal{S}$ 中的状态 $s$ 累加的形式，并用动态规划的方式进行求解，主要包含两个步骤：一是值的更新，二是策略的更新。两个步骤在动态规划算法循环中递归地更新对最优策略的估计，两个步骤递归更新的顺序衍生出了多种不同的算法。

$$V^{\pi}(s) = R_{\pi(s)}(s, s') + \gamma \sum_{s' \in \mathcal{S}} P_{\pi(s)}(s, s') V^{\pi}(s') \tag{2-13}$$

$$\pi'(s) = argmax_{a \in \pi(s)} R_a(s, s') + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V^{\pi}(s') \tag{2-14}$$

### 2.2.3　基于值的方法

基于值的方法[59](简称“值方法”) 是强化学习方法的一个分支，马尔科夫决策过程理论是值方法的理论基础。值方法尝试找到一个最优策略 $\pi^*$ 以最大化马尔科夫决策过程中的值函数，通过现有策略 $\pi$ 的执行结果 $R$ 来估计最优策略。

策略迭代 (Policy Iteration) 和值迭代 (Value Iteration) 是值方法中最为基础的方法。值迭代对状态空间 $\mathcal{S}$ 中的每一个状态 $s$，初始化值函数 $V^{\pi}(s) = 0$，然后选择能够使 $V^{\pi}(s)$ 最大的动作并执行，更新 $V^{\pi}(s)$ 直到整个迭代过程收敛。策略迭代方法的主要流程如下：首先利用等式 2-13(称为贝尔曼迭代方程，Bellman Iteration Equations) 对当前策略 $\pi$ 进行评估，并利用等式 2-14 进行策略更新，选择能够使 $V^{\pi}(s)$ 最大的动作作为状态 $s'$ 下的策略。经过多次迭代后，得到最优策略 $\pi^*$，该策略能够将贝尔曼迭代方程的值达到最优，得到贝尔曼最优方程 (等式 2-15)。

$$V^*(s) = max_{a \in \pi^*(s)} R_a(s, s') + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V^*(s') \tag{2-15}$$

利用状态来描述状态值值函数 $V^{\pi}(s)$ 足以解决许多马尔科夫决策问题，但更为普遍的方法引入了动作来描述动作值函数，即利用状态动作对 (State Action Pair) 来描述动作值函数 $Q^{\pi}(s, a)$(等式 2-16)。

$$Q^{\pi}(s', a) = R_a(s, s') + \gamma \sum_{s' \in \mathcal{S}} P_a(s, s') V^{\pi}(s') \tag{2-16}$$

值方法的实现方式及其改良的变种，主要包含以下几种：蒙特卡洛方法、时序差分学习方法 (Temporal Difference Method)、多步自助方法 (Bootstrap Method) 以及函数近似法 (Function Approximation Method) 等。上述方法往往利用当前策略的执行结



中国知网　https://www.cnki.net

果对最优策略进行估计，通常来讲是有偏估计。策略评估步骤得到的结果与最优策略之间存在一定偏差 (High Bias)，并且估计结果与最优策略之间的方差也较大 (High Variance)。

蒙特卡洛方法通过重复随机采样来获取数值结果，利用平均采样回报来解决强化学习问题，通常用于策略迭代算法中的策略评估步骤。智能体从初始状态与环境交互，能够得到多条交互序列，称之为轨迹 (Trajectory)。每一条轨迹都能获得累积奖励作为回报。

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots \tag{2-17}$$

进行策略评估时，蒙特卡洛方法从与环境交互的轨迹中采样 $N$ 个状态 $s$、动作 $a$、奖励 $r$ 序列，利用采样结果计算累积奖励的均值来近似替代策略评估中的累计奖励的期望。等式 2-18 就是蒙特卡洛方法对当前策略 $\pi$ 的累计奖励期望的近似估计。当采样的数量 $N$ 足够大时，根据大数定律，累积奖励的均值将会趋向于它的期望，蒙特卡洛方法产生的偏差也会变小。

$$V_t^{\pi}(s) \approx \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)} \tag{2-18}$$

$$V_t^{\pi}(s) = \mathbb{E}_{\tau \leftarrow \pi}[G_t | s_t = s] \tag{2-19}$$

蒙特卡洛方法的伪代码如算法 2.4 所示。与动态规划求解策略迭代不同，蒙特卡

**算法 2.4** Monte Carlo Methods Algorithm

**procedure** MC
    Initialize $V_s \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    Initialize $Return(s) \leftarrow$ empty list, for all $s \in \mathcal{S}$
    **for** $i \leftarrow 1, n$ **do**
        Generate a trajectory from $\pi$: $(s_i, a_i, r_i)$, $i \in [0, T]$
        $G \leftarrow 0$
        **for** $j \leftarrow 1, T$ **do**
            $G \leftarrow \gamma G + r_j$
        Append $G$ to $Return(s)$
        Compute $V_t^{\pi}(s) \approx \frac{1}{n}\sum_{i=1}^{n} G_t^{(i)}$
    **return** $V^{\pi^*}(s)$

洛方法不需要知道环境动态变化的模型，即当前环境各个状态之间的转移概率分布，仅仅从智能体与当前环境的交互中采样出状态、动作、奖励序列以进行学习。蒙特卡洛方法使用函数近似的方法，利用累积奖励的均值近似估计其期望，具有良好的收敛性，降低了估计的偏差的同是带来了较高的方差。蒙特卡洛方法只能从完整的序列中学习，必须等到轨迹结束后才能进行采样，因此学习速度较慢。



中国知网 https://www.cnki.net

时序差分方法[60]结合蒙特卡洛方法的优势，不需要知晓环境的动态变化模型，能直接从智能体与环境交互的经验中采样数据来进行学习。同时，时序差分方法也结合了动态规划方法的优势，可以利用当前的学习结果进行估计，不需要等待整个轨迹执行完毕。三种方法对于值函数的更新方式也不同，动态规划方法直接利用环境动态变化的模型的信息来更新值函数 (等式 2-20)；蒙特卡洛方法需要等到环境交互得到的回报已知，才用增量式的方法更新值函数 (等式 2-21)；时序差分方法的更新方式将二者进行了结合。最简单的时序差分算法是 TD(0) 算法，其值函数的更新方式如 (等式 2-22) 所示。

$$V(S_t) \leftarrow \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})] \tag{2-20}$$

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{2-21}$$

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2-22}$$

其中，$\alpha$ 表示更新的步长参数，为常量。估计回报 $R_{t+1} + \gamma V(S_{t+1})$ 被称之为时序差分目标，是带折扣因子的未来收益的总和。估计回报包含两个部分，一是走了某一步之后的实际奖励 $R_{t+1}$，二是通过自举的方式利用之前的估计结果估计 $V(S_t)$，估计的结果如等式 2-23 所示。TD(0) 算法的具体流程如算法 2.5 所示。

$$v_\pi(S) = R(s) + \gamma \mathbb{E}[v_\pi(S_{t+1}) \mid S_t = s] \tag{2-23}$$

**算法 2.5** Temporal Difference Methods Algorithm

**procedure** TD(0)
    Input the policy $\pi$
    Initialize $V_s$, arbitrarily $V(terminal) = 0$, for all $s \in \mathcal{S}$
    Initialize step size $\alpha \in (0, 1]$
    **while** $S$ is not terminal **do**
        Generate a trajectory from $\pi$: $(s_i, a_i, r_i)$, $i \in [0, T]$
        **while** each step in episode **do**
            $A \leftarrow$ action given by $\pi$ for $S$
            Take action $A$, observe reward$R$, state$S'$
            $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
            $S \leftarrow S'$

与蒙特卡洛方法相比，时序差分方法可以在线学习，即每走一步就可以更新，效率要高于蒙特卡洛方法。蒙特卡洛方法需要从环境中获得完整的状态、动作、奖励序列，只能在环境有终止状态的情况下学习。时序差分方法没有上述约束，通过自举的方式能够从不完整的序列中学习，不要求环境有终止状态。时序差分方法利用了马尔科夫性质，在马尔科夫环境下的学习效率更高。时序差分方法的估计结果有一定的偏差，但和蒙特卡洛方法相比，有更低的方差。



中国知网　https://www.cnki.net

### 2.2.4 基于策略的方法

基于策略的方法 (简称“策略方法”) 将策略 $\pi$ 参数化，直接在策略空间上搜索最佳策略。值函数依然用于策略方法中，但无需通过值函数来确定动作选择。根据是否使用梯度进行优化，策略方法主要包含两种，一种是无梯度方法 (Gradient Free)，另一种是梯度方法 (Gradient Based)。无梯度方法在搜索最优策略中不使用梯度信息，主要的方法有模拟退火算法、交叉熵搜索、进化计算等。

梯度方法通过策略的参数向量 $\theta \in \mathbb{R}^{d'}$ 来定义策略 $\pi(a \mid s, \theta) = P_r\{A_t = a \mid S_t = s, \theta_t = \theta\}$，该式表示在当前状态 $s$、策略 $\pi$ 参数为 $\theta$ 的情况下，选择动作 $a$ 执行的概率。策略梯度 (Policy Gradient) 是梯度方法中最简单的方法，策略梯度通过梯度上升的方法来更新策略参数 $\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$，其中，$J(\theta)$ 表示策略参数 $\theta$ 的衡量指标，$\nabla J(\theta_t) \in \mathbb{R}^{d'}$ 是一种随机估计，其期望近似于参数 $\theta_t$ 的性能指标的梯度。策略梯度得到的最优策略既可以是确定的 $\pi_\theta^*(s)$，也可以是随机的 $\pi_\theta^*(s) = P(a|s, \theta)$ 策略梯度主要的实现方法有蒙特卡洛策略梯度 (Reinforce) 等。蒙特卡洛策略梯度利用累计奖励 $G_t$ 的期望值作为动作值函数 $Q^\pi(s_t, a_t)$ 的无偏估计 (等式 2-24)。

$$\nabla J(\theta) = \mathbb{E}_\pi[Q^\pi(s_t, a_t)\frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)}] \rightarrow \nabla J(\theta) = \mathbb{E}_\pi[G_t\frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)}] \tag{2-24}$$

蒙特卡洛策略梯度的伪代码如算法 2.6 所示。蒙特卡洛策略梯度的策略参数更新

**算法 2.6** Monte Carlo Policy Gradient

```
procedure MCPG
    Input a differentiable policy parameterization π(a|s, θ)
    Initialize policy parameter θ ∈ ℝ^{d'}
    Initialize step size α ∈ (0, 1]
    for i ← 1, n do
        Generate a trajectory (s_i, a_i, r_i), i ∈ [0, T], following π(∗|∗, θ)
        for i ← 1, T do
            G ← Σ_T^{i=t+1} γ^{i−t−1} R_i
            θ ← θ + αγ^t G ∇lnπ(a_t | s_t, θ)
    return π(a|s, θ*)
```

方式如等式 2-25 所示。每一个增量都与收益 $G_t$ 和向量的乘积成正比，即采取实际行动的概率的梯度除以采取该行动的概率。向量表示参数空间中参数的更新方向，该方向能够最大程度增加在未来访问状态 $s_t$ 时，重复选择动作 $a_t$ 的概率。该更新方式使参数向量在该方向上增加，与返回成正比，与动作概率成反比。前者是有意义的，因为它使参数向有利于产生最高回报的动作的方向移动。后者是有意义的，因为否则，频繁选择的操作将处于优势 (更新将更频繁地朝着它们的方向进行)，即使它们没有产



中国知网 https://www.cnki.net

生最高的回报，也可能会获胜。

$$\theta_{t+1} \approx \theta_t + \alpha G_t \frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)} \tag{2-25}$$

蒙特卡洛策略梯度使用从时间 $s$ 开始的完整回报，其中包括直到该游戏回合结束的所有未来奖励。从这个意义上讲，蒙特卡洛策略梯度是一种蒙特卡罗算法，只在游戏回合完成后或者整个环境进入了终止状态才对所有参数进行更新。

## 2.3　本章小结

本章是本文研究工作的理论部分内容，第一部分讲述了常用的游戏 AI 实现技术，主要包含有限状态自动机、行为树、树搜索、机器学习，并介绍了这些常用方法的基本概念和理论、关键的数学原理以及实现上述方法的算法的主要流程。第二部分的内容主要包含强化学习的基本理论内容，介绍了强化学习的基本要素和智能体进行强化学习的直观流程，介绍了强化学习的数学基础——马尔科夫决策过程，以及实现强化学习的两种基本方法，简要说明了这两种方法的数学原理和基本实现算法，简要分析了实现算法之间的优劣。



中国知网　https://www.cnki.net

中国知网 https://www.cnki.net

# 第三章　结合 A* 算法和 Q-Learning 算法的游戏 AI 动态避障

本文第二章内容介绍了本文研究所用到的理论知识。在本章中，将利用之前介绍的强化学习和游戏 AI 理论知识，针对当前游戏 AI 存在的无法进行动态寻路的问题，提出一种解决方法。然后，利用经典游戏吃豆人作为算法的测试环境，对本章所提出算法的动态寻路能力进行测试。总结本章算法在吃豆人游戏中的运行结果，并分析和传统算法相比，本章所提出算法存在的优劣。

## 3.1　Q-Learning 算法和 A* 算法简介

### 3.1.1　Q-Learning 算法简介

Q-Learning 算法是一种无模型 (Model-Free) 策略无关 (Off-Policy) 的强化学习算法，由时序差分算法进化而来。时序差分算法不需要等待游戏回合结束，再对值函数进行更新，而是以一种自举的方式，用当前的执行结果来估计值函数。策略依赖 (On-Policy) 时序差分方法 SARSA[61]是 Q-Learning 算法前身，策略依赖方法仅通过当前的策略 $\pi$ 来估计每一个状态动作对的值函数 $q_\pi(s,a)$。一个游戏回合可以由状态和状态动作对序列构成，如图 3.1 所示。

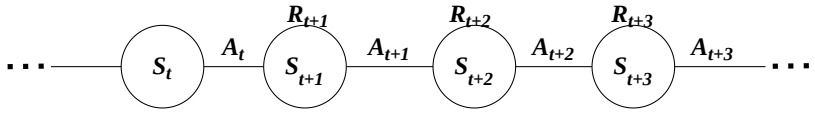



图 3.1　一个游戏回合构成图

时序差分算法考虑的马尔科夫决策过程中的转移是状态之间的转移，而 SARSA 算法考虑的是状态动作对之间的转移。两种应用例子是等价的，因为不管是状态间的转移还是状态动作对之间的转移都属于带奖励过程的马尔科夫链。SARSA 算法的动作值函数更新方式如式 3-1 所示。

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)] \tag{3-1}$$

该更新项利用下一步的 $Q$ 值 $Q(S_{t+1},A_{t+1})$ 来更新上一步的 $Q$ 值 $Q(S_t,A_t)$，$R_{t+1} + \gamma Q(S_{t+1},A_{t+1})$ 就是时序差分目标，是 $Q(S_t,A_t)$ 需要去逼近的值。实际上，时序差分目标是累积奖励 $G_t$ 的近似，累积奖励可由等式 2-17 改写成迭代的形式 (等式 3-2)。

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{3-2}$$

如果利用 $Q(S_{t+1},A_{t+1})$ 取近似累计奖励 $G_{t+1}$，则时序差分目标就会近似到累积奖励 $G_t$。该更新项将会一直执行直到达到终止状态，如果 $S_{t+1}$ 是终止状态，则 $Q(S_{t+1},A_{t+1})$



中国知网　https://www.cnki.net

将被定义为零。动作值函数 $Q(S_t, A_t)$ 的更新规则利用游戏回合 (Episode)，即状态动作对序列的完整五元组 $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$，因此该方法被称之为 SARSA 算法。

将 SARSA 算法的策略依赖性质变成策略无关性质，即策略无关时序差分方法，是 Q-Learning[62] 的主要改进的地方。策略无关方法在学习过程中包含两种策略，一种是目标策略，另一种是行为策略。目标策略是利用环境交互经验学习得到的策略，而行为策略可以是随机的。Q-Learning 算法的目标策略由动作值函数估计得到，而行为策略使用的是 $\epsilon$-greedy 策略，是不完全随机的，且是基于动作值函数逐渐改进的。这样做的好处就是能保证算法收敛的情况，尽可能的去探索未知的状态，来获得更高值的动作值函数。Q-Learning 算法对动作值函数的更新如式 3-3 所示。

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3-3}$$

从直观来讲，动作值函数 $Q$ 直接利用最优动作值函数 $q^*$ 来更新，而不依赖于当前策略，保证了算法的收敛性，只要所有状态动作对一直在更新，算法就会收敛。但当前策略依旧对算法的执行过程有影响，因为它决定了哪一组状态动作对应该被访问并更新。Q-Learning 算法的执行流程如算法 3.1 所示。

**算法 3.1** Q-Learning Algorithm

**procedure** Q-LEARNING
    Initialize $Q(s, a)$, arbitrarily $Q(terminal, *) = 0$, for all $s \in \mathcal{S}$ $a \in \mathcal{A}$
    **while** $S$ is not terminal **do**
        Initialize $S$
        **while** each step in Episode **do**
            Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
            Take action $A$, Observe $R$, $S'$
            Compute $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', a) - Q(S, A)]$
            $S \leftarrow S'$

### 3.1.2 A* 算法简介

A* 算法是一种启发式的最佳优先搜索算法，广泛用于在静态图中求解最短路径。在进行最短路径搜索时，A* 算法将待搜索区域量化成节点构成的图。A* 算法需要确定下一步可选择的节点，并将这些节点存入开放列表中。根据评估函数对开放列表中的节点进行评估，用以选择下一步的节点拓展搜索路径，该评估函数一般包含着启发信息，能够使得 A* 算法计算得到搜索区域的最短路径。评估函数的一般形式如等式 3-4 所示。

$$f(n) = g(n) + h(n) \tag{3-4}$$

其中，$n$ 表示搜索路径上的下一个节点。$g(n)$ 表示从起点到任意节点 $n$ 的实际距离，$h(n)$ 表示任意节点到目标节点的估算距离。如果 $g(n)$ 为 0，则算法转换为使用贪心



中国知网 https://www.cnki.net

策略的最优搜索算法，速度最快，但不一定能搜索到全局最优解。如果 $h(n)$ 为 0，则算法转化为 Dijkstra 算法，此时需要计算最多的节点。如果 $h(n)$ 小于等于节点 $n$ 到目标节点的实际距离，则一点可以计算得到全局最优解，且 $h(n)$ 越小，需要计算的节点数量越多，算法效率越低。最常见的距离度量方式有欧几里得距离、曼哈顿距离、切比雪夫距离。

标准的 A* 算法利用优先队列来维护开放列表，以评估函数作为排序的优先级。在算法循环的每一步中，A* 算法会计算周围未访问的邻居节点，计算它们的评估函数，并将它们存储到开放列表中。A* 算法选择开放列表中评估函数值最小的节点来扩展搜索路径，并将该节点从优先队列中剔除，加入到关闭列表以显示该节点被访问过。算法循环进行，直到达到目标节点。A* 算法的主要流程如算法 3.2 所示。

**算法 3.2** A* Search Algorithm

```
1:  procedure A* SEARCH
2:      Initialize Current node n_0, f(n_0) = 0, openList = [n_0], closedList = []
3:      while openList != Null do
4:          if n is goal then
5:              break;
6:          while child node n_i ∉ closedList do
7:              if n_i ∈ openList then
8:                  if f(n_i) is minimal then
9:                      Append node n_i into path
10:                     Delete node n_i from openList
11:                     Insert node n_i into closedList
12:             Compute f(n_i) = g(n_i) + h(n_i)
13:             Insert node n_i into openList
```

## 3.2　结合启发式搜索的 Q-Learning 算法改进

### 3.2.1　算法基本原理

传统的游戏 AI 寻路算法有基于最佳优先搜索的 A* 算法及其改进版本 JSP 算法，以及基于导航网格的 NavMesh 算法等。上述算法能够让游戏 AI 在给定的静态区域范围内，规划出从起始点到目标点的最短路径。但当游戏 AI 处于一种动态变化的游戏环境中，即游戏环境内的障碍物是动态的，能够在游戏环境中移动，甚至游戏 AI 触碰到障碍物会受到一定损害，则上述算法无法展现出良好的路径规划能力，来规避这些动态的障碍物。因为上述算法所设计出的游戏 AI 都是基于规则的，其行为逻辑一成不变，当游戏环境改变或者游戏规则变化，基于旧环境或旧规则的游戏 AI 的行为逻辑在新环境中不再适用。所以，本次研究将引入强化学习算法 Q-Learning 算法，与传统寻路算法——A* 算法相结合，提出 Q*-Learning 算法，来提升游戏 AI 对环境



中国知网　https://www.cnki.net

的适应能力，使得基于 Q*-Learning 算法的游戏 AI 能够避开动态的障碍物。

标准的 Q-Learning 算法以最大化 $Q(S_t, A_t)$ 值为目标来选择下一步的动作，由于 $Q(S_t, A_t)$ 是累积奖励 $G_t$ 的近似。所以，基于 Q-Learning 的游戏 AI 选择动作会以最大化累积奖励为目标。但是，在游戏 AI 训练的初期，游戏 AI 的 $Q$ 表上的 $Q$ 值是十分稀疏的，导致游戏 AI 的路径选择不能快速地达到目标，不能快速的获得有效奖励。所以，在游戏 AI 更新 $Q$ 值之前，先利用 A* 算法进行路径规划，规划出游戏 AI 当前位置到目标点的最短路径，并记录下游戏 AI 巡航最短路径所需要的损耗 $F_P$。

$$F_P = \sum_{n_i \in P} C_{n_i} \tag{3-5}$$

然后，将路径损耗加入到 $Q$ 值的更新中，并乘以影响因子 $\omega$ 来控制路径损耗对 $Q$ 值更新的影响程度。$C_{n_i}$ 表示节点 $n_i$ 的父节点到此节点的损耗。利用路径损耗来影响游戏 AI$Q$ 值的更新，进而影响其动作选择，对游戏 AI 的动作选择具有一定的指导性，使得游戏 AI 选择的动作能够让游戏 AI 更快的到达目标点，游戏 AI 能够更快的获得奖励。Q*-Learning 算法的动作值函数 $Q^*$ 更新方式如式 3-6 所示。

$$Q^*(S_t, A_t) \leftarrow Q^*(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q^*(S_{t+1}, a) - Q^*(S_t, A_t)] - \omega F_P \tag{3-6}$$

**算法 3.3** Q*-Learning Algorithm

```
1:  procedure Q*-LEARNING
2:      Initialize Q*(s, a), arbitrarily Q*(terminal, *) = 0, for all s ∈ S a ∈ A
3:      while S is not terminal do
4:          Initialize state S, position n
5:          while each step in Episode do
6:              Choose A from S using policy derived from Q* (e.g., ϵ-greedy)
7:              Take action A, Observe R, S', n'
8:              Derive path P via A* algorithm
9:              Compute F_P = ∑_{n_i∈P} C_{n_i}
10:             Compute :
                Q*(S_t, A_t) ← Q*(S_t, A_t) + α[R_{t+1} + γmax_a Q*(S_{t+1}, a) − Q*(S_t, A_t)] − ωF_P
11:             S ← S'
```

算法的执行流程图如图 3.2 所示，Q*-Learning 更新动作值函数 $Q^*$ 的具体流程，如算法 3.3 所示。与标准 Q-Learning 算法相比，Q*-Learning 算法利用 A* 算法来辅助提升游戏 AI 的训练效果，其执行过程主要包含以下几个步骤：

(1) 首先，游戏 AI 先初始化各项训练参数，并记录当前状态 $s$ 和位置 $n$，根据行为策略 $\epsilon$-greedy 选择下一步动作 $a$，即游戏 AI 会以 $\epsilon$ 的概率选择随机动作，以 $1-\epsilon$ 的概率根据 $Q^*$ 表中最大 $Q^*$ 值来选择动作。使用 $\epsilon$-greedy 策略能够适当的平衡游戏 AI 对未知环境的探索和已有经验的利用，并且易于实现。



中国知网 https://www.cnki.net

(2) 游戏 AI 执行动作 $a$ 后，更新当前状态 $s'$ 和位置 $n'$，计算目前获得的奖励 $R$，并启动 A* 算法模块。利用 A* 算法计算从当前位置 $n'$ 到目标位置的最短路径 $P$，计算巡航整条路径 $P$ 所需要的损耗 $F$，根据奖励 $R$ 和路径损耗 $F$ 计算 $Q^*$ 值，并更新 $Q^*$ 表。

(3) 游戏 AI 查询状态 $s'$ 是否为终止状态，若 $s'$ 为终止状态则整个算法流程结束，游戏 AI 的训练过程结束；若不为终止状态，则游戏 AI 继续执行步骤 (1) 直到游戏达到终止状态。

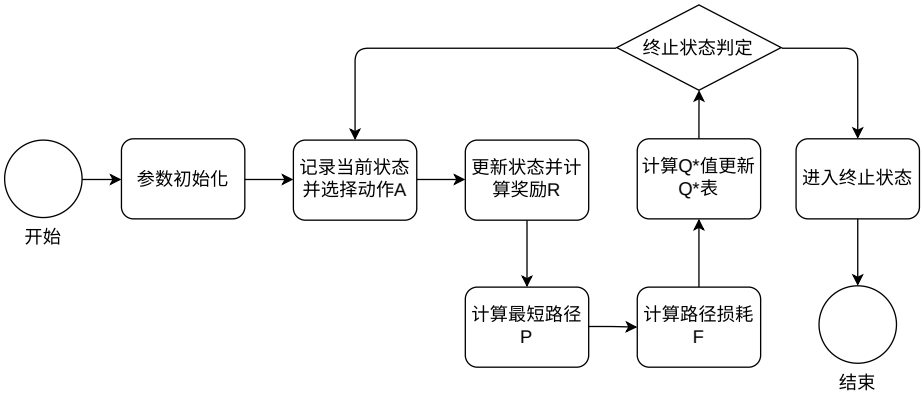



图 3.2　Q*-Learning 算法流程图

### 3.2.2　算法基本实现

基于 Q*-Learning 算法的寻路游戏 AI 实现分为三个基本模块，一是 A* 算法模块，二是 Q*-Learning 算法模块，第三是动作执行模块。动作执行模块包含游戏 AI 在游戏环境中具体执行的动作，例如：向东、向西、向南、向北、停止。A* 算法模块主

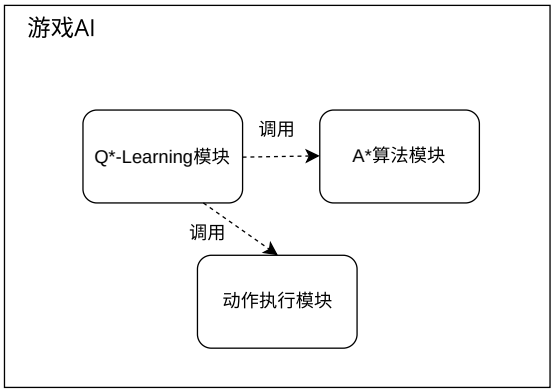



图 3.3　游戏 AI 架构图示意

要用于游戏 AI 规划出当前位置到目标位置的最短路径，并计算出游戏 AI 遍历该路径所需的损耗 (在本文研究中为路径长度)，Q*-Learning 算法模块是游戏 AI 的主体，



中国知网　https://www.cnki.net

主要执行游戏 AI 的训练过程，以及各项参数的更新。本文研究所设计的游戏 AI 架构图，如图 3.3 所示。

(1) A* 算法模块

A* 算法模块中的开放列表由优先队列维护，优先队列 (Priority Queue) 是一种数据结构，优先队列中的元素被赋予优先级，优先级最高的元素会被优先访问，通常由堆 (Heap) 实现。堆是一颗完全二叉树，其父节点比子节点具有更高的优先级。用数值来体现优先级，可将堆分为小顶堆和大顶堆。大顶堆中的根节点为堆中所有节点的最大值，所有父节点的值要大于子节点的值；小顶堆中的根节点为所有节点的最小值，所有的父节点的值要小于子节点的值。因为游戏 AI 搜素到的最短路径的长度越小越好，所以本次研究使用的是小顶堆来实现优先队列。

此外，A* 算法模块还拥有着将搜索区域量化成由节点构成的图的功能，该功能将游戏 AI 在不同时刻的位置记录成节点。将游戏 AI 的位置量化成节点，可以创建一个节点类去包含该位置更多的信息，例如：该节点的父节点的位置信息、该节点通往下一个子节点的方向信息、由该节点的父节点到达该节点的损耗和距离等等。将搜索区域量化成图，可以利用链表来存储搜索得到的最短路径，对于搜索区域复杂的场合，节省了大量存储空间，同时也能更加方便地计算最短路径的长度和遍历损耗。

算法的主体循环部分，首先将游戏 AI 的起始点加入到开启列表中，算法循环直到开启列表为空时结束，若游戏 AI 到达目标点则强制结束。从开启列表中取出启发函数 $f(n)$ 最小的节点，若该节点的邻居节点不在关闭列表中，计算所有邻居节点的 $f(n)$ 的值，以 $f(n)$ 的值作为优先级加入到开启列表中，然后将所有邻居节点加入到关闭节点中，表示这些节点已被访问。

(2) Q*-Learning 算法模块

该模块需要构建多个对象：游戏状态、游戏 AI 状态、游戏 AI 智能体等。游戏状态用于记录游戏的运行信息，包含游戏环境变化信息、游戏 AI 状态、游戏得分等。游戏状态作为一种标识，标记着在当前时刻对应的游戏 AI 的行为策略、动作执行的结果、下一步的动作选择、以及 $Q^*$ 表中的 $Q^*$ 值等，确保各项参数在更新时不会应用错误的值。游戏 AI 状态记录着游戏 AI 的运行信息，包含着游戏 AI 当前的位置、游戏 AI 的行进方向、游戏 AI 的行进速度等，作为游戏状态的一部分被用作于各项参数的标识，以便在需要时调用。

Q*-Learning 算法是游戏 AI 智能体的主体部分，$Q^*$ 表用 Python 中的字典来维护。字典，即哈希表，是一种数据结构，用于存储键值对，构建键和值之间的映射关系。字典中的键表示状态动作对。字典中的值表示 $Q^*$ 值的大小。游戏 AI 智能体对象中包含更新 $Q^*$ 表、更新参数、获取策略、获取下一步动作、计算 $Q^*$ 值等方法。上述方法都需要传入游戏状态作为标识，更新 $Q^*$ 表时，还需传入下一步的游戏状态、游戏



中国知网 https://www.cnki.net

AI 执行的动作和奖励等。游戏 AI 在进行动作选择时，会先筛选一次合法动作，以免游戏 AI 执行不合法动作触碰到静态的障碍物。

在算法运行的主体部分，游戏 AI 观测游戏状态之间的转移是否发生，若状态转移发生，则计算当前得到的奖励值，并调用更新方法来更新 $Q^*$ 表中的值，在新的状态中，游戏 AI 选择表中最大值作为下一步的动作。参数初始化会在游戏回合开始前结束，一个新的游戏回合开始时，记录上一次的状态和动作，当前游戏回合结束后，计算整个游戏回合的累积奖励。

### 3.2.3　算法测试环境

本次研究所使用的操作系统是 Windows 系统，编程语言使用的是 Python，用到 Python 标准库有用于科学计算的 numpy、用于构建优先队列的 heapq、用于存储运行结果的 json、用于构建游戏图形的 tkinter。实验平台的具体信息如表 3.1 所示。

表 3.1　实验配置信息

| 名称 | 配置 | |
|---|---|---|
| CPU | 型号 | AMD Ryzen 7 4800H |
| | 主频 | 2.90GHz |
| | RAM | 16.0GB |
| 开发环境 | Pycharm | |
| 操作系统 | Windows | |
| 编程语言 | Python | |
| Python 库 | numpy、json、tkinter、heapq | |

本文研究选用经典游戏吃豆人作为算法的测试环境。吃豆人游戏遵循马尔科夫性质，未来的任何游戏状态都仅依赖于当前的游戏状态，适合作为本文研究的算法测试平台。吃豆人的状态可以直接用当前的游戏画面来表示，游戏画面中包含了吃豆人的位置、鬼怪的位置、豆子的位置和能量药丸的位置。也可以利用间接地表示方式，来减小游戏的状态总数和 $Q^*$ 表的维度，例如吃豆人附近是否存在豆子或者鬼怪等。对于吃豆人来说，可能采取的动作可以是保持现在的动作、停止、向东行、向西行、向北行、向南行。关于吃豆人的强化学习[63]的奖励函数可以被设计成当吃豆人吃到豆子或能量药丸时给出正面奖励，当吃豆人碰到鬼怪时死亡，并给出负面奖励。

为了将吃豆人的游戏分数与寻路效果关联，本文研究改进了奖励函数的设计。吃豆人会有一个基本分数，如果吃豆人的一次动作没有吃到豆子，则扣一些基本分数。如果吃豆人在空地上循环行动，即使吃豆人完成了游戏，吃豆人能得到的分数也会较低。所以，吃豆人要在吃掉所有豆子的情况下，尽可能规划出更短的路径来获得更高



中国知网　https://www.cnki.net

的分数。该奖励函数的设计能够将游戏分数和寻路效果联系起来，算法寻路性能之间的差异也能更好的体现出来。同时，该奖励函数的设计能够缓解吃豆人游戏的奖励空间比较稀疏的问题，算法的收敛性进一步加强。

本章算法测试的第一个测试环境如图 3.4 所示。算法运行的各项参数设置如下：$\epsilon$-greedy 参数 $\epsilon$ 为 0.05、学习步长 $\alpha$ 为 0.2、累计奖励折扣因子 $\gamma$ 等于 0.8、路径损耗影响因子 $\omega$ 为 0.8。环境 1 是单纯的搜索任务，地图上一共有 17 颗豆子，对于吃豆人来说，只要能够吃掉所有豆子就能获得游戏胜利。在吃掉所有豆子的情况下，吃豆人规划出的路径越短，吃豆人能够得到的游戏分数越高，得到游戏分数更高的算法寻路性能越好。

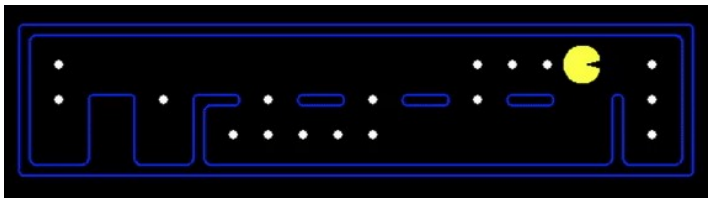


图 3.4　吃豆人测试环境 1

本章算法测试的第二个测试环境如图 3.5 所示。算法运行的各项参数设置如下：$\epsilon$-greedy 参数 $\epsilon$ 为 0.05、学习步长 $\alpha$ 为 0.2、累计奖励折扣因子 $\gamma$ 等于 0.8、路径损耗影响因子 $\omega$ 为 0.8。吃豆人在环境 2 不仅需要吃掉所有的豆子，并且需要规避动态障碍物鬼怪的路径，吃豆人触碰到鬼怪则会死亡。地图上共有 56 颗豆子，以及两颗能量药丸。当吃豆人吃掉能量药丸时，鬼怪速度减缓，吃豆人触碰到鬼怪不在触发死亡。吃豆人需要不触碰到鬼怪的情况下，规划出吃掉所有豆子的最短路径。吃豆人规划出的路径越短，得到的游戏分数也会越高，表示其寻路算法的性能越好。

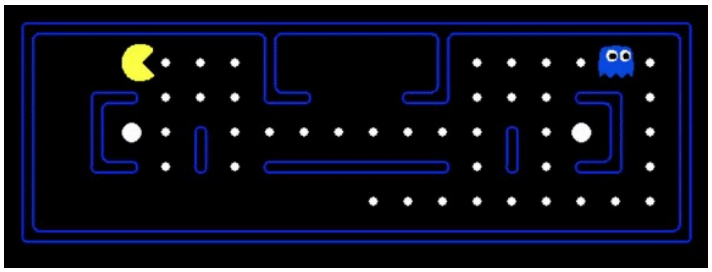


图 3.5　吃豆人测试环境 2

本章算法测试的第三个测试环境如图 3.6 所示。算法运行的各项参数设置如下：$\epsilon$-greedy 参数 $\epsilon$ 为 0.05、学习步长 $\alpha$ 为 0.2、累计奖励折扣因子 $\gamma$ 等于 0.8、路径损耗影响因子 $\omega$ 为 0.8。与测试环境 2 相比，环境 3 的豆子数量达到了 98 颗，地图面积更大，迷宫复杂程度更高，鬼怪的数量多一个。对于吃豆人而言，在环境 3 中吃掉所有豆子的难度会更高，规划出最短路径的难度也会更高。同样，如果吃豆人能在存活



中国知网　https://www.cnki.net

的情况下，规划出吃掉所有豆子的路径越短，吃豆人能够得到的分数也越高，算法的寻路性能也会更好。

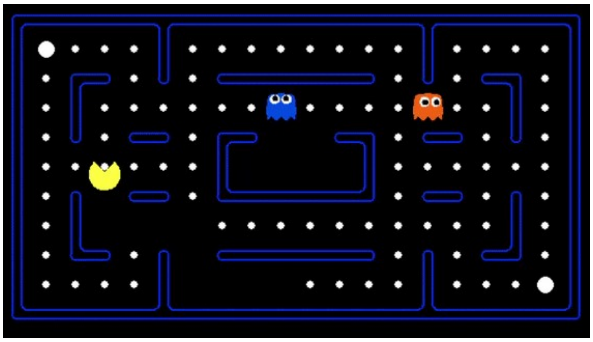


图 3.6　吃豆人测试环境 3

上述测试环境中的鬼怪是随机鬼怪，即鬼怪的动作选择是随机的。鬼怪的每次动作选择会先对非法动作进行过滤，以免出现鬼怪撞上迷宫墙等情况。鬼怪的合法动作由 Python 的字典维护，动作对应字典中的键，动作对应的概率存储在字典的值中，每一个合法动作的选择概率相等。利用 Python 的随机数库 random 随机出一个大小存在于区间 $[0, 1]$ 的数，随机数落到哪一个动作对应的概率区间内，则选择该动作。

### 3.2.4　算法测试结果

测试环境 1 的地图结构比较简单，地图搜索空间有限，游戏状态空间的维度相较于其他两个测试环境更小，吃豆人经历的状态数量较少。所以，Q*-Learning 算法、Q-Learning 算法、A* 算法控制的吃豆人都能够在测试环境 1 的地图中完成搜索任务，并规划出吃掉所有豆子所需的最短路径。但是与标准的 Q-Learning 算法相比，Q*-Learning 算法展现的优势有限，搜索到的平均路径更短，得到的平均奖励仅略高于 Q-Learning 算法。

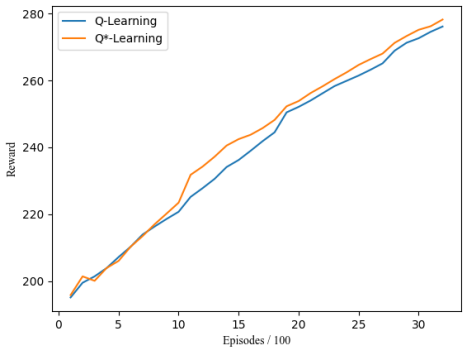



图 3.7　Q*-Learning 吃豆人在测试环境 1 的训练过程

因为 Q*-Learning 和 Q-Learning 算法都属于机器学习算法，在训练的初期，算法还未收敛导致游戏 AI 的寻路能力稍差，吃豆人游戏 AI 存在不能完成搜索任务的游



中国知网　https://www.cnki.net

戏回合，所以二者的平均奖励和平均路径相较于单纯依赖搜索的 A* 算法较低。等到训练后期，Q*-Learning 算法和 Q-Learning 算法已经收敛到一个比较好的结果，游戏 AI 得到的最高奖励和最短路径与 A* 算法一致。图 3.7 记录了 Q*-Learning 算法和 Q-Learning 算法的训练过程。表 3.2 记录了测试环境 1 的算法运行结果。

表 3.2　Q*-Learning 吃豆人测试环境 1 的算法运行结果

| 算法 | 最高奖励 | 平均奖励 | 最短路径长度 | 平均路径长度 |
|---|---|---|---|---|
| Q*-Learning | 278 | 233.3 | 34 | 38.8 |
| Q-Learning | 278 | 230.8 | 34 | 42.6 |
| A* 算法 | 278 | 278 | 34 | 34 |

相对于测试环境 1，测试环境 2 的地图结构变得更加复杂，地图搜索空间更大。由于可移动障碍物鬼怪的加入，游戏的空间复杂度变高，吃豆人经历的游戏状态数量也变多。由于吃豆人触碰到鬼怪死亡，吃豆人会得到一个极大的负面奖励，所以，即使基于 A* 算法的吃豆人能规划出吃掉所有豆子的最短路径，也不能在该地图上完成搜索任务。A* 算法得到的最高奖励和平均奖励都很低。

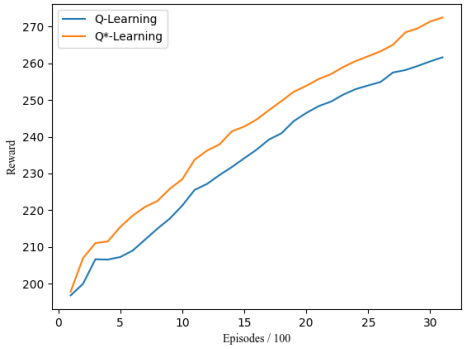



图 3.8　Q*-Learning 吃豆人在测试环境 2 的训练过程

表 3.3　Q*-Learning 吃豆人测试环境 2 的算法运行结果

| 算法 | 最高奖励 | 平均奖励 | 最短路径长度 | 平均路径长度 |
|---|---|---|---|---|
| Q*-Learning | 272 | 233.4 | 66 | 71.1 |
| Q-Learning | 261 | 225.7 | 74 | 82.2 |
| A* 算法 | 152 | 131.4 | - | - |

而 Q*-Learning 和 Q-Learning 算法具有学习能力，能够适应环境的变化，根据环境种的变化进行动作选择，具备规避鬼怪并改变原有路径的能力。由于测试环境 2 的地图迷宫更加复杂，Q*-Learning 算法的优势更加明显。因为 Q*-Learning 算法在进行



中国知网　https://www.cnki.net

动作选择时，会对目标点的最短路径进行搜索计算，所以 Q*-Learning 算法在相同的游戏回合数内，能够比 Q-Learning 算法获得更高的奖励。尽管 Q-Learning 算法也能完成搜索任务，搜索出最短路径，但是 Q*-Learning 算法得到的平均奖励和最高奖励更高，规划出路径的平均值更小。图 3.8 记录了 Q*-Learning 算法和 Q-Learning 算法的训练过程，表 3.3 记录算法的执行结果。

相比于测试环境 1 和测试环境 2，测试环境 3 中拥有两个随机移动的鬼怪，以及更为复杂的迷宫结构，规避鬼怪的同时规划出吃掉所有豆子的路径会将更加的困难。所测试结果来看，A* 算法依然不能够完成搜索任务，并且由于游戏环境更加的复杂，吃豆人碰上鬼怪的概率大大增加，与测试环境 2 相比，A* 算法能够得到的最高奖励和平均奖励更低。

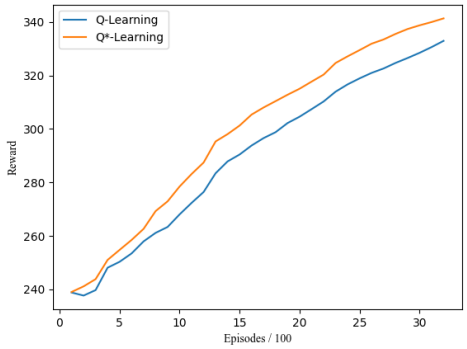



图 3.9　Q*-Learning 吃豆人在测试环境 3 的训练过程

表 3.4　Q*-Learning 吃豆人测试环境 3 的算法运行结果

| 算法 | 最高奖励 | 平均奖励 | 最短路径长度 | 平均路径长度 |
|---|---|---|---|---|
| Q*-Learning | 341 | 288.9 | 127 | 142.8 |
| Q-Learning | 332 | 280.2 | 138 | 154.6 |
| A* 算法 | 124 | 113.2 | - | - |

从训练过程和测试结果来看，Q-Learning 算法和 Q*-Learning 算法都能完成搜索任务。与 Q-Learning 算法相比，Q*-Learning 算法在更为复杂的环境中依然更有优势。Q*-Learning 算法得到的最高奖励和平均奖励都要高于 Q-Learning 算法；并且规划得到的最短路径和平均路径都要短于 Q-Learning 算法。Q*-Learning 算法在更复杂的环境中，寻路能力更强，规避动态障碍物的能力也更强。

### 3.2.5　算法优劣分析

综合上述测试结果来看，当吃豆人存在于一个状态空间复杂度较小且没有动态障碍物的游戏环境中时，基于强化学习的 Q-Learning 和 Q*-Learning 算法对比 A* 算



中国知网　https://www.cnki.net

法没有明显优势。在此类简单的环境中，基于强化学习的算法还需要包含游戏 AI 的训练过程，游戏 AI 训练初期的对未知环境的探索过程降低了算法的效率，花费了更多的时间，利用最佳优先搜索的算法能直接通过暴力搜索得到结果节约了训练时间，更有效率。在简单环境中，A* 算法的寻路效果尚可。

当测试环境的地图迷宫变得更加复杂并且加入动态障碍物，整个游戏环境的状态空间复杂度升高，A* 算法的寻路效果降低，几乎不能完成搜索任务。由于动态障碍物的动作是随机的，其运动轨迹自然也是随机的，而 A* 算法规划出的最短路径是固定，从而导致吃豆人游戏 AI 在搜索时会触碰到障碍物，在测试中，A* 算法没有规避掉障碍物导致游戏 AI 得分较低。基于强化学习的算法 Q*-Learning 和 Q-Learning 因为其具备学习能力，能够适应游戏环境的动态变化，能够自主的规避动态障碍物并完成路径搜索任务。又因为 Q*-Learning 算法结合了 A* 算法的路径规划能力，基于 Q*-Learning 算法的游戏 AI 在动作选择时，能够考虑到当前位置到目标位置的最短距离，所以 Q*-Learning 算法能够更快地收敛，其得到的平均奖励和规划出的平均路径要高于 Q-Learning 算法。

当测试环境的复杂程度进一步提升时，Q*-Learning 算法依然保有了其优势。对比 Q-Learning 算法，Q*-Learning 算法得到的最高奖励和平均奖励，规划到的最短路径和平均路径，均高于 Q-Learning 算法。所以，游戏环境的复杂程度提升，Q*-Learning 算法的寻路效果因此而受到影响的幅度较小。

## 3.3 本章小结

本章主要介绍了强化学习算法 Q-Learning 和最佳优先搜索算法 A* 的基本理论和算法流程，并将启发式搜索引入到 Q-Learning 算法中，提出了 Q*-Learning 算法，来改善基于搜索算法的游戏寻路算法不能规避动态障碍物的问题。本章将经典游戏吃豆人作为算法测试环境，利用复杂度依次升高的三张地图来测试算法的寻路效果，对测试结果进行了总结。并根据算法的测试结果，综合分析和评判，和传统算法相比，本章所提出的算法在寻路能力、避障能力上的优劣，以及在游戏表现上的差异。



中国知网 https://www.cnki.net

# 第四章　结合 Minimax-Q 算法的游戏 AI 动态避障优化

本章将对第三章中的测试环境做出一些改进，为测试环境中的鬼怪赋予一定的智能。本章将引入零和博弈算法对第三章所提出的方法进行改进，来进一步提升游戏 AI 的寻路和避障能力。本章将重点介绍算法的基本原理和实现流程，然后在改进的游戏环境中对算法进行测试，最后分析并总结本章所提出算法存在哪些优劣。

## 4.1　零和博弈和 Minimax-Q 算法简介

### 4.1.1　零和博弈

博弈论[64]是一种研究用于表述理性智能体之间策略化交互的数学模型的学科，广泛应用于社会科学、逻辑学、系统科学、计算机科学等。博弈论最早用来解决零和博弈问题，现在常常用于研究行为之间的联系，并用于智能体的决策。零和博弈是博弈论中的一种情形，为博弈双方一得一失、得失相等的博弈情形提供了数学表达。囚徒困境是最为常见的零和博弈情形，零和博弈也广泛存在于各类游戏中，例如：棋牌游戏、吃豆人等。最常用于解决零和博弈问题的方法是极小化极大理论，该理论与线性规划对偶和纳什均衡理论联系紧密。零和博弈拥有零和性质，意味着任何零和博弈

表 4.1　零和博弈基本情形的收益矩阵示意

| | player2 | | |
|---|---|---|---|
| player1 | | choice1 | choice2 |
| | choice1 | -A, A | B, -B |
| | choice2 | C, -C | -D, D |

情形的结果都是帕累托最优[65]。零和博弈的帕累托最优回报将会导致博弈双方以惩罚对手为标准来进行决策，即博弈双方会尝试最小化对方能得到的回报，同时最大化己方能得到的利益。

零和博弈的纳什均衡可以通过解决线性规划问题得到[66]。设零和博弈的回报矩阵为 $M$，其中元素 $M_{i,j}$ 表示玩家尝试最小化对方回报而执行策略 $i$，尝试最大化己方回报而执行策略 $j$ 得到的回报。假设回报矩阵 $M$ 中的所有元素均为正，则零和博弈至少存在一个纳什均衡，可由如下线性规划得到向量 $u$：

最小化：

$$\sum_i u_i \tag{4-1}$$



中国知网　https://www.cnki.net

通过约束：

$$u \geq 0 \tag{4-2}$$

$$Mu \geq 1 \tag{4-3}$$

第一项约束表示向量 $u$ 中的元素都是非负的，第二项约束表示回报矩阵和向量 $u$ 的乘积至少为 1。对于生成的 $u$ 向量，其元素之和的倒数就是博弈的值。将 $u$ 乘以该值给出一个概率向量，给出最大化玩家选择每个可能的纯策略的概率。如果博弈矩阵的所有元素不都是正的，那么给每个元素加上一个足够大的整数，使矩阵元素变成正的，也不会影响到纳什均衡的混合策略。求上述线性规划的对偶的解，可以得到最小化收益方的均衡混合策略。如果上述线性规划的所有解都被找到，它们就构成了零和博弈的所有纳什均衡。

### 4.1.2 Minimax-Q 算法

Minimax-Q 算法[67]属于多智体博弈强化学习算法,将马尔科夫博弈[68]框架 (Markov Game Framework) 应用到了强化学习中的马尔科夫决策过程。Minimax-Q 算法利用双人零和马尔科夫博弈作为算法的测试环境，即游戏环境中存在两个智能体和单个奖励函数，其中一个智能体尝试最大化奖励函数，另一个智能体尝试最小化奖励函数，该算法在石头剪刀布游戏的测试环境中取得了比较好的效果，成功将强化学习算法应用到了解决零和博弈问题中。

基于 Minimax-Q 算法的智能体在动作选择时，会考虑对手的动作，并尝试将对手执行动作后得到的奖励最小化。Minimax-Q 将智能体的动作选择集合定义为 $A$，将对手的动作选择集合定义为 $O$，利用 $R(s,a,o)$ 表示智能体在状态 $s$ 下执行动作 $a$，而对手执行动作 $o$，智能体能得到的即时奖励。马尔科夫决策过程总是存在至少一个最优策略使得智能体得到的累积折扣奖励最大，而对于很多马尔科夫博弈来说，最优策略不一定是确定的，智能体的行动目标是最小化对手的动作收益，策略的选择和更新是和对手的策略息息相关的。这种衡量策略的指标得到的策略通常是保守的，这种策略可以迫使智能体与更大胆的对手取得平局，这种策略会对一些对手产生巨大的回报，而对其他对手则会造成巨大的损失。智能体的策略表示动作选择的概率分布 $\pi \in P(A)$，为了使得策略收敛到最优，需要最大化值函数 $V$，如等式 4-4 所示:

$$V(s) = max_{\pi \in P(A)} min_{o \in O} \sum_{a \in A} R(s,a,o)\pi_a \tag{4-4}$$

式 4-4 即为双人零和博弈中的线性规划项，同时也是值函数，在最大化己方收益、最小化对手收益的情况下，利用线性规划解出动作选择概率 $\pi_a$。

在马尔科夫决策过程中，Q-Learning 算法利用动作值函数代替了值函数，来更新贝尔曼方程。Minimax-Q 算法中利用了同样的方法，将值函数 $V(s)$ 用动作值函数



中国知网 https://www.cnki.net

$Q(s,a,o)$ 代替，$Q(s,a,o)$ 表示在状态 $s$ 在智能体执行动作 $a$，对手执行动作 $o$ 得到的 $Q$ 值。$Q(s,a,o)$ 在马尔科夫博弈中的更新过程如等式 4-5 所示。

$$Q(s,a,o) = R(s,a,o) + \gamma \sum_{s' \in S} P(s,a,o,s')V(s') \tag{4-5}$$

利用动作值函数来代替等式 4-4 中的值函数可得到等式 4-6。

$$V(s) = max_{\pi \in P(A)} min_{o \in O} Q(s',a,o)\pi_a \tag{4-6}$$

等式 4-5 的迭代过程，可写成如下形式。

$$Q(S_t,A_t,O_t) \leftarrow Q(S_t,A_t,O_t) + \alpha[R_{t+1} + \gamma V(s) - Q(S_t,A_t,O_t)] \tag{4-7}$$

$$V(s) = max_{\pi \in P(A)} min_{o \in O} Q(S_{t+1},a,o)\pi_a \tag{4-8}$$

等式 4-7 表示 $Q(s,a,o)$ 值的更新方向，朝着最大化己方收益，最小化对手收益的方向进行更新 $Q(s,a,o)$ 值。其中，$\alpha$ 表示更新步长，$\gamma$ 表示折扣因子。等式 4-8 是表示零和博弈中的线性规划项，以动作值函数来表示，智能体通过线性规划来确定动作选择概率 $\pi_a$

Minimax-Q 算法的整体流程如算法 4.1 所示。首先初始化 $Q(s,a,o)$ 值，动作选择概率 $\pi_a$ 值，智能体状态集合 $S$ 和动作集合 $\mathcal{A}$。初始化智能体初始状态 $S$。在游戏回

**算法 4.1** Minimax-Q Algorithm

**procedure** MINIMAX-Q
    Initialize $Q(s,a,o)$, arbitrarily $Q(terminal,*,*) = 0$, for all $s \in S$ $a \in \mathcal{A}$ $o \in O$
    Initialize $\pi_{s,a} = \frac{1}{|\mathcal{A}|}$
    **while** $S$ is not terminal **do**
        Initialize $S$
        **while** each step in Episode **do**
            Choose action $A$ with probability $\pi_{S,a}$
            Take action $A$, Observe $R$, $S'$ and opponent action $O$
            Compute $Q(S,A,O) \leftarrow Q(S,A,O) + \alpha[R + V(S) - Q(S,A,O)]$
            Use linear programming to find $\pi_{S,*}$ such that:
            $\pi_{S,*} = argmax(\pi'_{S,*}, min(o', sum(a', \pi_{S,a'} * Q(S,a',o'))))$
            Let $V(S) = min(o', sum(a', \pi_{S,a'} * Q(S,a',o')))$
            $S \leftarrow S'$

合的每一个步骤中，根据动作选择概率 $\pi_a$ 选择动作 $A$，观测即时奖励 $R$ 和对手动作 $O$，利用 4-6 式来更新 $Q(s,a,o)$ 的值，并利用 4-4 式进行线性规划更新智能体的动作选择概率 $\pi_a$，直到整个游戏达到终止状态。



中国知网 https://www.cnki.net

## 4.2 结合零和博弈的 Q*-Learning 算法改进

### 4.2.1 算法基本原理

在上一章中，Q*-Learning 算法的测试环境中的动态障碍物的动作是随机的，智能体规避动态障碍物的路径的难度较低。当动态障碍物具备一定的智能，比如动态障碍物会朝着智能体的位置进行移动，并不断追逐着智能体，则基于 Q*-Learning 算法的游戏 AI 的寻路效果降低，规避动态障碍物的能力下降。发生这种情况的原因是 Q*-Learning 算法并未考虑动态障碍物的动作收益，当前动作选择策略和最优策略相差较大。所以本章引入零和博弈算法 Minimax-Q，将二者结合提出 Minimax-Q* 算法。基于 Minimax-Q* 算法的游戏 AI 在寻路时，会进一步考虑动态障碍物的动作收益，并尝试将其最小化，在保证游戏 AI 能够规划出最短路径的同时，来进一步提升游戏 AI 规避具有一定智能的障碍物的能力。

Q*-Learning 算法将遍历 A* 算法规划的最短路径所需的消耗 $F$ 加入到 $Q$ 值的更新当中，使得基于 Q*-Learning 算法的游戏 AI 在规避动态障碍物时会考虑如何规划出更短的路径。引入 Minimax-Q 算法后，游戏 AI 在规避动态障碍物时会考虑最小化动态障碍物的动作收益。首先，游戏 AI 会以 $\pi_a$ 的概率进行动作选择，执行动作后观测执行结果、以及对手动作 $o$ 并利用 A* 算法进行路径规划，利用执行结果更新 $Q$ 值。然后，根据最小的 $Q$ 值以最大化 $V(s)$ 为目标进行线性规划，利用线性规划的结果来更新动作选择概率分布 $P(A)$。最后，根据动作选择概率分布 $P(A)$ 最大的概率值 $\pi'_a$ 来执行下一步动作。Minimax-Q* 算法的 $Q$ 值更新如式 4-9 所示，便于区分 Minimax-Q 算法，利用 $Q^*$ 符号来代替 $Q$。

$$Q^*(S_t, A_t, O_t) \leftarrow Q^*(S_t, A_t, O_t) + \alpha[R_{t+1} + \gamma V(s) - Q^*(S_t, A_t, O_t)] - \omega F_P \tag{4-9}$$

$$V(s) = max_{\pi \in P(A)} min_{o \in O} Q^*(S_{t+1}, a, o)\pi_a \tag{4-10}$$

Minimax-Q* 算法利用 Minimax-Q 算法中式 4-8 来代替 Q*-Learning 算法中的最大 $Q$ 值项，通过线性规划来获得下一步要进行的动作，而不是通过最大 $Q$ 值来获得。Minimax-Q 算法为 Q*-Learning 算法赋予了零和博弈的特性，使得基于二者结合的 Minimax-Q* 算法的游戏 AI 能够尝试最小化动态障碍物的动作收益，从而增强规避动态障碍物的能力。

Minimax-Q* 算法的主要流程如算法 4.2 所示，图 4.1 为算法主流程图，算法的主循环主要包含以下几个步骤：

(1) 第一步，游戏 AI 先初始化各项训练参数，并记录当前的初始状态 $s$ 和位置 $n$，根据行为策略 $\epsilon$-greedy 选择下一步动作 $a$，即游戏 AI 会以 $\epsilon$ 的概率选择随机动作，以 $1-\epsilon$ 的概率根据 $\pi$ 表中最大 $\pi_a$ 值来选择动作 $a$，$\pi$ 表中存储可选动作的选择



中国知网 https://www.cnki.net

概率，$\pi$ 表中的元素总和等于 1。使用 $\epsilon$-greedy 策略能够适当的平衡游戏 AI 对未知环境的探索和已有经验的利用，并且易于实现。

**算法 4.2** Minimax-Q* Algorithm

**procedure** MINIMAX-Q*
  Initialize $Q(s,a,o)$, arbitrarily $Q(terminal,*,*)=0$, for all $s\in\mathcal{S}$ $a\in\mathcal{A}$ $o\in\mathcal{O}$
  Initialize $\pi_{s,a}=\frac{1}{|\mathcal{A}|}$
  **while** $S$ is not terminal **do**
    Initialize $S$
    **while** each step in Episode **do**
      Choose action $A$ with probability $\pi_{S,a}$
      Take action $A$, Observe $R$, $S'$, $n'$ and opponent action $O$
      Derive path $P$ via A* algorithm
      Compute $F_P=\sum_{n_i\in P}C_{n_i}$
      Compute $Q(S,A,O)\leftarrow Q(S,A,O)+\alpha[R+V(S)-Q(S,A,O)]-\omega F_P$
      Use linear programming to find $\pi_{S,*}$ such that:
      $\pi_{S,*}=argmax(\pi'_{S,*},min(o',sum(a',\pi_{S,a'}*Q(S,a',o'))))$
      Let $V(S)=min(o',sum(a',\pi_{S,a'}*Q(S,a',o')))$
      $S\leftarrow S'$

(2) 游戏 AI 执行动作 $a$ 后，更新当前状态 $s'$ 和位置 $n'$ 并记录对手的动作 $o$，计算目前获得的奖励 $R$，并启动 A* 算法模块。利用 A* 算法计算从当前位置 $n'$ 到目标位置的最短路径 $P$，计算巡航整条路径 $P$ 所需要的损耗 $F_P$。之后，进行线性规划求解下一步各项可选动作的动作选择概率 $\pi'$，并更新 $\pi$ 表。根据之前得到的 $Q^*$ 值和 $\pi$ 值来更新 $V(s)$ 值，并更新 $V(s)$ 表。根据奖励 $R$ 和路径损耗 $F$，以及 $V$ 值计算 $Q^*$ 值，并更新 $Q^*$ 表。

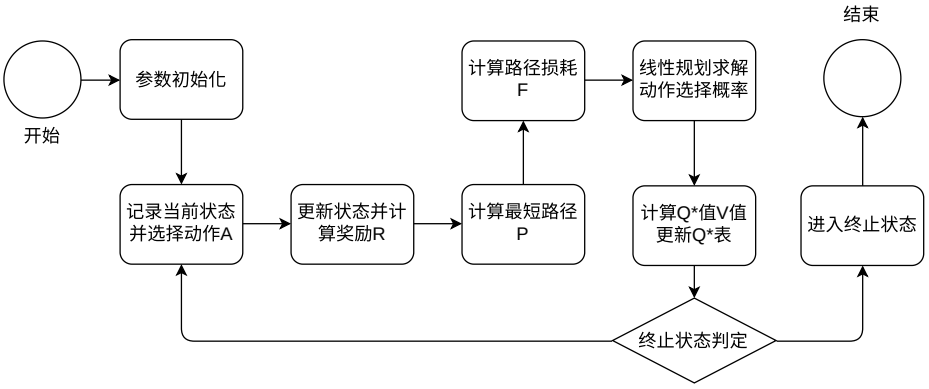



图 4.1　Minimax-Q* 算法流程图

(3) 游戏 AI 查询状态 $s'$ 是否为终止状态，若 $s'$ 为终止状态则整个算法流程结束，游戏 AI 的训练过程结束；若不为终止状态，则游戏 AI 继续执行步骤 (1) 直到游戏达到终止状态。



中国知网　https://www.cnki.net

### 4.2.2 算法基本实现

基于 Minimax-Q* 算法的游戏 AI 除了拥有 Q*-Learning 算法模块、动作执行模块、A* 算法模块，还有一个用于计算线性规划的模块。Minimax-Q* 算法的前三个模块的作用和工作原理与 Q*-Learning 算法中的模块基本一致，区别在于当进行动作选择时，游戏 AI 选择动作的依据不再是根据 $Q^*$ 表中的最大值来选择动作，而是依据动作选择概率 $\pi$ 表来选择，选择 $\pi$ 中概率最大的动作来执行。$\pi$ 表的维度与游戏 AI 能够执行的动作数量相等，但在进行动作选择时，游戏 AI 会对合法动作进行过滤，当 $\pi$ 表中的最大概率动作不合法，则选择概率次大的动作，以此类推，直到选择的动作是合法动作。本次研究所设计的游戏 AI 架构图，如图 4.2 所示。

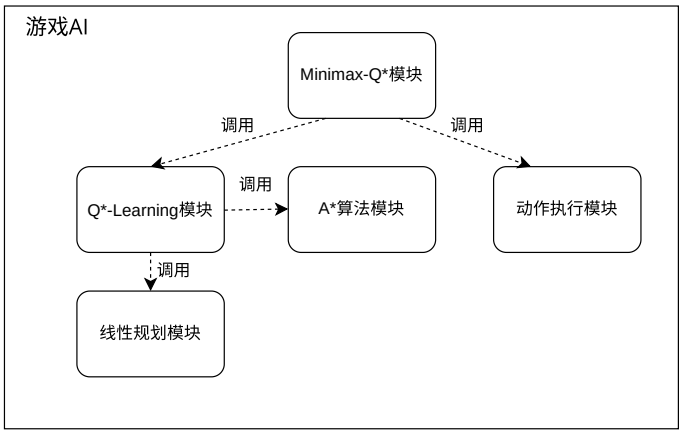



图 4.2　游戏 AI 架构示意

计算线性规划的模块使用 Python 库 pulp 来实现，pulp 是开源的第三方工具包，可以用于求解线性规划、整数规划、混合整数规划问题等。pulp 使用的主要步骤如下所示：

(1) 第一步先要确定线性规划问题。pulp 中的 LpProblem 方法是定义线性规划问题的构造函数。该函数输入的第一个变量类型为字符串，表示用户定义的问题名称，用于输出信息；输入的第二个变量名为 sense，用来表示优化方向 (求最大值还是最小值)，求最大值需要输入参数 pulp.LpMaximize，求最小值需要输入参数 pulp.LpMinimize。本次研究需要求解动作选择概率的最大值，所以选用参数 pulp.LpMaximize。

(2) 第二步需要确定线性规划变量。由于求解的是游戏 AI 的动作选择概率，所以线性规划变量的个数由游戏 AI 的动作空间维度确定。线性规划变量由 pulp.LpVariable 函数确定。LpVariable 的输入变量包括线性规划变量的名称，类型为字符串；线性规划变量的上下界以及类型，可设置的变量类型包括连续、离散、二进制。游戏 AI 的所有动作概率的应当处于 $[0,1]$ 的区间内。

(3) 第三步需要线性规划过程添加目标函数和约束条件，约束条件比较单一，即



中国知网 https://www.cnki.net

为游戏 AI 的所有合法动作的概率之和应当等于 1。

### 4.2.3　算法测试环境

本次研究的实验平台配置与第三章中的 Q*-Learning 算法的实现平台的配置基本相同，Python 工具包使用的种类和数量以及使用目的基本相同。本次实验多出了一个 Python 的工具包 pulp 来求解线性规划。实验平台的具体信息如表 4.2 所示。

表 4.2　实验配置信息

| 名称 | 配置 | |
|---|---|---|
| CPU | 型号 | AMD Ryzen 7 4800H |
| | 主频 | 2.90GHz |
| | RAM | 16.0GB |
| 开发环境 | Pycharm | |
| 操作系统 | Windows | |
| 编程语言 | Python3.7 | |
| Python 包 | numpy、json、tkinter、heapq、pulp | |

Minimax-Q* 算法的第一个测试环境与第三章中的 Q*-Learning 算法测试环境中的测试环境 2 相同，只拥有一个可随机移动的鬼怪，迷宫的复杂程度趋于一般。吃豆人在环境 1 不仅需要吃掉所有的豆子，并且需要规避动态障碍物鬼怪的路径，吃豆人触碰到鬼怪则会死亡。地图上共有 56 颗豆子，以及两颗能量药丸。当吃豆人吃掉

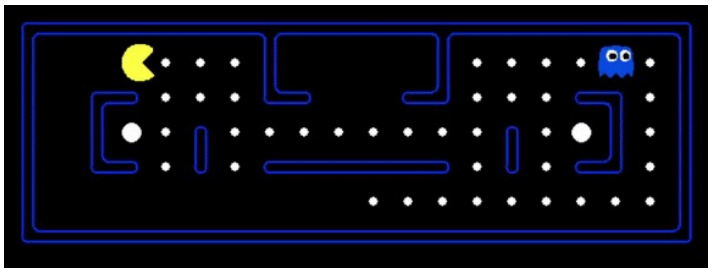


图 4.3　吃豆人游戏测试环境

能量药丸时，鬼怪速度减缓，吃豆人触碰到鬼怪不在触发死亡。吃豆人需要不触碰到鬼怪的情况下，规划出吃掉所有豆子的最短路径。吃豆人规划出的路径越短，得到的游戏分数也会越高，表示其寻路算法的性能越好。通过相同的测试环境可以综合体现 Minimax-Q* 算法、Q*-Learning 算法、Q-Learning 算法和 A* 算法之间寻路效果以及规避动态障碍物能力的差异。Q*-Learning 算法运行的各项参数设置如下：$\epsilon$-greedy 参数 $\epsilon$ 为 0.05、学习步长 $\alpha$ 为 0.2、累计奖励折扣因子 $\gamma$ 等于 0.8、路径损耗影响因子 $\omega$ 为 0.8。图 4.3 为第一个算法测试环境。



中国知网　https://www.cnki.net

算法的第二个测试环境在第一个测试环境上稍有改进，第一个环境中的动态障碍物是随机移动的，而第二个测试环境中的动态障碍物是有目的性的，第二个测试环境的动态障碍物鬼怪会朝着吃豆人所在位置的方向进行移动。鬼怪的动作选择同样由概率分布决定。鬼怪会计算执行下一步动作后与吃豆人之间的距离，然后选择距离最近的动作作为最佳动作并执行。最佳动作的选择概率由一个可调节的参数决定，参数称为攻击概率 $\tau$ 处于区间 $[0,1]$。攻击概率除以鬼怪的合法动作的维度，即是最佳动作的选择概率，$1-\tau$ 除以鬼怪的合法动作的维度，则是其他动作的选择概率。本次实验中，鬼怪的攻击概率 $\tau$ 设定为 0.8。

### 4.2.4 算法测试结果

在算法测试环境 1 中，鬼怪是随机运动，吃豆人想要规避鬼怪的路径较为容易，吃豆人在迷宫中完成豆子搜索任务较为容易。所以，Minimax-Q* 算法、Q*-Learning

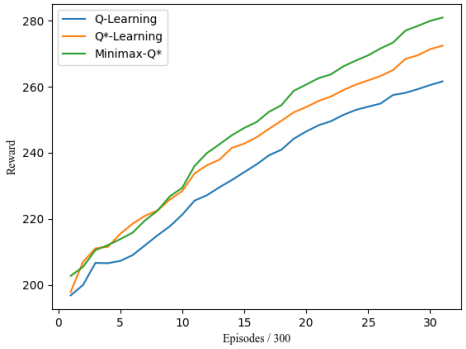



图 4.4 Minimax-Q* 吃豆人在测试环境 1 中的训练过程

表 4.3 Q*-Learning 吃豆人测试环境 3 的算法运行结果

| 算法 | 最高奖励 | 平均奖励 | 最短路径长度 | 平均路径长度 |
|---|---|---|---|---|
| Minimax-Q* | 280 | 237.6 | 64 | 68.7 |
| Q*-Learning | 272 | 233.5 | 66 | 71.1 |
| Q-Learning | 261 | 225.7 | 74 | 82.2 |

算法、Q-Learning 算法的吃豆人游戏 AI 都能吃完所有豆子，并且能规划出完成一次任务所需要的最短路径，可以拿到较高的游戏分数。而 A* 算法为启发式搜索算法，不具备规避鬼怪路径的能力，所以 A* 算法并不能完成搜索任务。

基于 Minimax-Q* 算法的吃豆人游戏 AI 能够尝试将鬼怪的动作收益最小化，以一种更为保守的策略完成任务，即尽最大可能的去规避鬼怪的行动路径，所以 Minimax-Q* 算法相比于 Q*-Learning 算法、Q-Learning 算法、以及其他几种传统的路径搜索算法，能得到的平均奖励更高，规划路径的平均值也更短。所以，Minimax-Q* 算法拥



中国知网 https://www.cnki.net

有更好的路径搜索能力和动态规避能力。图 4.4 记录了算法的训练过程，表 4.3 记录了算法的运行结果。

在第二种测试环境中，当吃豆人行进到鬼怪附近时，鬼怪会表现出攻击性而朝着吃豆人运动。鬼怪会计算自身位置与吃豆人位置之间的距离，优先选择能够缩短自身与吃豆人之间的距离的动作。所以，对比第一种测试环境中的结果，虽然 Q*-Learning 算法和 Q-Learning 算法依然能够完成搜索任务，但是二者的平均奖励降低，搜索的平均路径也变长，算法性能略有下降。

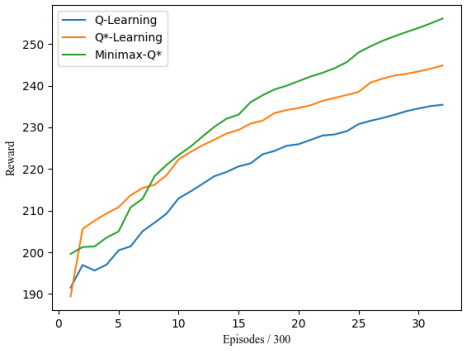



图 4.5　Minimax-Q* 吃豆人在测试环境 2 中的训练过程

表 4.4　Q*-Learning 吃豆人测试环境 3 的算法运行结果

| 算法 | 最高奖励 | 平均奖励 | 最短路径长度 | 平均路径长度 |
|---|---|---|---|---|
| Minimax-Q* | 256 | 224.2 | 64 | 76.2 |
| Q*-Learning | 246 | 220.0 | 66 | 81.6 |
| Q-Learning | 235 | 211.3 | 76 | 93.3 |

而对于基于 Minimax-Q* 算法的吃豆人游戏 AI 而言，即便鬼怪能够朝着吃豆人运动，吃豆人能够尝试将鬼怪的动作收益最小化，所以，鬼怪的行为模式的改变并不太影响 Minimax-Q* 算法的路径规划和规避障碍物的性能，Minimax-Q* 算法的平均奖励依然最高，平均路径依然最短，与其他两种算法相比，规划出的路径极值也最小。图 4.5 记录了算法的训练过程，表 4.4 记录的算法的运行结果。

### 4.2.5　算法优劣分析

综合上述测试结果来看。在第一种测试环境中，鬼怪的动作是随机的，没有展现出一定的规律，基于 Minimax-Q* 算法的吃豆人游戏 AI 对鬼怪的行为预测不够准确，所以对鬼怪动作收益的估计不够准确，所以，对比 Q*-Learning 算法和 Q-Learning 算法来说，Minimax-Q* 算法能够拿到的平均奖励要高出一些，规划的平均路径要稍微更短。面对简单的环境，基于 Minimax-Q* 算法的吃豆人能展现出一定的优势。对于



中国知网　https://www.cnki.net

A* 算法来说，基于 A* 算法的吃豆人游戏 AI 不具备规避动态障碍物的能力，所以不能完成搜索任务。

当为鬼怪赋予一定的智能后，鬼怪能够主动朝着吃豆人的方向前进，鬼怪的行为变得可以预测，吃豆人能够有效地最小化鬼怪的动作收益。当 Q*-Learning 算法和 Q-Learning 算法在面对具有一定智能的鬼怪时，展现出的性能略有下降，能够拿到的平均奖励略微降低、规划得到的平均路径略微增长。而引入了 Minimax-Q 算法后，基于 Minimax-Q* 算法的吃豆人的测试结果虽然有明显的下降，但是吃豆人游戏 AI 的游戏表现依然是最好的。所以，在面对更加复杂的环境，尤其是鬼怪表现出一定智能后，Minimax-Q* 算法表现出了一定的路径规划能力和规避动态障碍物的能力。

## 4.3 本章小结

本章主要介绍了零和博弈的基本概念和应用于零和博弈的 Minimax-Q 算法的基本原理，对上一章提出的 Q*-Learning 算法进行了改进，将零和博弈算法引入并提出了 Minimax-Q* 算法，来进一步提升智能体的避障能力。本章继续利用经典游戏吃豆人作为算法测试环境，并对吃豆人游戏中的鬼怪做出了一点改进，鬼怪的行为动作由随机变为定向，即鬼怪选择的动作能够缩短鬼怪与吃豆人间的距离，对测试结果进行了分析和总结。并根据算法的测试结果，综合分析和评判，与传统算法相比，本章所提出的算法的避障能力优劣和游戏表现的差异。



中国知网 https://www.cnki.net

# 第五章　游戏 AI 智能寻路算法应用

第三章和第四章主要是针对传统寻路算法中存在的问题提出新的解决方案，本章的内容将侧重于如何将第三章和第四章提出的算法应用到经典的游戏当中，本章将利用经典游戏开发平台 Unity 来设计一张吃豆人游戏地图以模拟一些游戏环境，并利用本次研究提出的算法来解决吃豆人的寻路问题。

## 5.1　游戏平台介绍

### 5.1.1　Unity 游戏设计平台简介

Unity[69]是一款包含 3D 和 2D 视觉互动内容创作和开发的平台，该平台除了游戏开发外，还可以进行美术、建筑、汽车设计、影视等内容的创作，不少创作者和开发着都借助 Unity 来实现自己的创意。Unity 最为主要的应用场景是游戏设计，为游戏行业的开发人员提供了 3D、2D 游戏引擎来进行游戏开发。

Unity 引擎灵活便捷，创作者可以使用 iOS、Android、Windows、MacOS 等不同平台上的不同工具来协助自身的开发工作。开发者可以在 20 多个平台上创建和优化自身的开发内容，并且它提供开发合作者的监控功能，以便在最新版本和平台上提供优化的支持服务。Unity 不仅为创作者提供创作工具，还提供运营服务来帮助创作者推广自己的创作内容。这些运营业务包括 Unity Ads，通常用以制作游戏的广告平台、以及 Unity Game Cloud 游戏云、Vivox 游戏语音、Multiplay 境外服务器托管平台、Unity 创作游戏内容分发 (UDP)、Unity Asset Store 资源商店、以及 Unity 云构建服务等，让创作者们可以更快速的分享自己的想法。

Unity 用户界面十分简洁，是一种同时具备可视化文字编辑、详细信息编辑器，以及移动游戏接口的分层式集成开发平台。也因为其超强的内置编译能力，Unity 可以被用来游戏内容的快速创建，或开发游戏原型。Unity 的绘图引擎可以支持 Direct3D(Windows)、OpenGL(Mac、Windows) 以及其他更灵活的编程方式。它可以支持浮雕贴图、反射贴图、视差贴图、屏幕空间环境遮罩、动态阴影使用的阴影贴图技术，以及渲染纹理和全屏后处理效果。着色器是用 Shader 语言编写的，支持自由工作流编程模式或 CG/GLSL 语言编写的着色器。着色器的游戏屏幕控制就像在 Photoshop 中编辑数字图片。着色器可以包含许多变量和参数接口，因此 Unity 可以确定这些接口当前是否受支持，并将它们调整为最合适的参数，之后选择相应的着色器类型以实现广泛的兼容性。

Unity 拥有强大的内置地形编辑功能，支持创建地形、树木和植被地块、自动化



中国知网　https://www.cnki.net

的地形细节层次和表面效果。即使是低端硬件也可以平滑地执行大型且生动的植被景观，其中的 Tree Create 工具可以编辑树木不同部分的细节。Unity 物理引擎是一段可执行程序，该程序通过模拟一些实际的物理参量来模拟牛顿力学模型。该物理引擎采用 PhysX 物理引擎构建，可以轻松准确地模拟出开发所需的物理效果。PhysX 不仅可以使用中央处理器进行计算，还可以在设计中使用独立的浮点处理器 (比如 GPU 或者 TPU) 进行计算，因此它可以轻松地为计算量较大的过程 (如流体动力学模拟) 执行物理模拟计算。此外，PhysX 物理引擎还可以在市面上大部分的开发平台上运行，包括 Windows、Linux、Xbox 360、MacOS、Android 等。

Unity 的音视频播放主要通过 OpenAL 库来实现，OpenAL 库中的 Ogg Vorbis 负责压缩音效，Theora 进行视频编码，并且 Unity 支持 3D 图形音频和视频流的实时混合。OpenAL 的主要功能是对源对象、声音缓冲区进行编码。源对象主要包含缓冲区、速度、声音位置和方向以及声音强度的标记。听众参量主要包括听众的速度、位置和方向，以及所有声音的总分。缓冲存储器可以计算引擎执行所需的参量，例如距离衰减、多普勒功率等，它包含的音频功率数据可以是 8 位或 16 位单声道数据，或这是立体声 PCM 格式数据。

Unity 使用 Mono 脚本作为游戏开发脚本，Mono 是一种基于.NET 语言框架的开源编程语言。创作者可以使用 JavaScript、C# 或 Boo 等编程语言对 Mono 脚本进行编辑。Unity 资源服务器包含一个版本控制方案。这种方案可以保证编程工作的并发性，创作者可以将不同工具开发的不同版本的脚本集成到一起，该方案大大提升了开发效率。Unity 提供了一个非常复杂的光影再现系统，该系统提供柔和的光效和阴影。Unity 平台使用 LightMap 来存储引擎中的光强度数据，其中包括游戏中面部灯光的信息。LightMap 通常用于光照效果预先计算和静态测量。

### 5.1.2 吃豆人游戏地图设计

在本文的算法应用部分内容中，将利用 Unity 平台来设计吃豆人游戏。Unity 2D Project 是 Unity 平台专门用于制作 2D 游戏的。Unity 平台本身为游戏开发者预制好了多个游戏场景和 Demo 供游戏开发者选择，以提升游戏开发者的效率。多个不同游戏制作场景也能帮助游戏开发新手快速上手 Unity 游戏开发。

在 UnityHub 中创建一个 2D 项目，调整好相机位置和 Hierarchy 以及背景颜色。设计吃豆人的游戏地图首先是要设计地图的迷宫，设计完迷宫的主要结构后，将设计好的迷宫图片存入到新建的 Sprite 文件夹中。在 Inspector 中设置好迷宫的导入参数，将迷宫的纹理型设置为 Sprite(2D and UI)。新建一个 Sprite 对象，并利用 Sprite Renderer 对地图迷宫进行渲染，便可得到吃豆人游戏迷宫。之后，利用 Physics 2D 中的 Box Collider 2D 为迷宫中墙体添加物理材质，使得迷宫变得不可穿过。



中国知网 https://www.cnki.net

然后需要将鬼怪和吃豆人单位添加到迷宫地图当中。首先，需要为吃豆人创建动画，动画通过将一帧内的吃豆人嘴巴张开的角度不同来实现，所以需要实现准备好吃豆人朝着四个不同方向的嘴巴张角的图片。设置吃豆人动画状态机，当吃豆人沿着某个方向移动时，调用吃豆人沿着该方向移动的动画，来实现吃豆人移动的动画。然后，同样需要为吃豆人添加物理材质，使得吃豆人触碰到迷宫时无法越过。

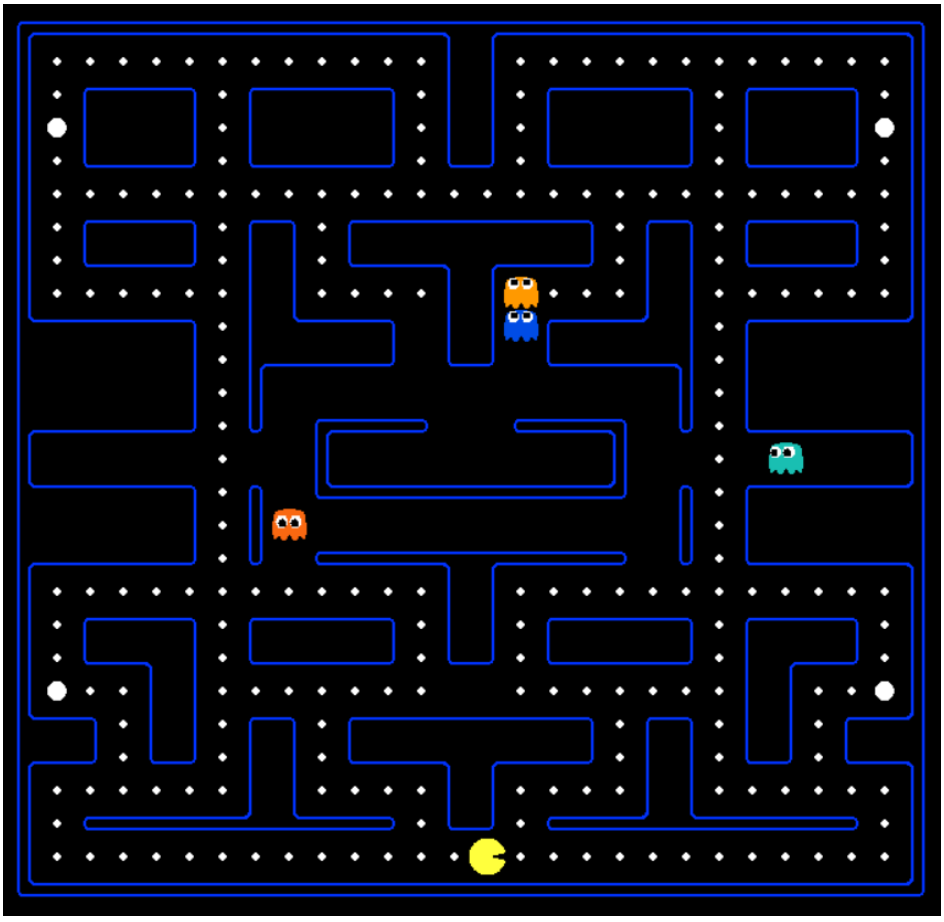


图 5.1　智能寻路算法游戏地图

添加好吃豆人的物理材质后，需要为吃豆人添加行为逻辑。在吃豆人的 Hierarchy 中添加新的脚本，并在其中写好吃豆人的行为逻辑代码，通过变量 speed 来控制吃豆人游戏 AI 的移动速度，通过变量 vector 来控制吃豆人的移动方向，通过函数 keyToVector() 来控制键盘键位对吃豆人方向的影响。鬼怪的制作与吃豆人的制作大致相同。最后的需要在地图中添加豆子和能量药丸，将豆子和能量药丸的图片添加到 Sprite 文件夹中，纹理型设置为 Sprite(2D and UI)。接下来，为豆子和能量药丸添加脚本。当吃豆人经过豆子或能量药丸时，豆子或能量药丸需要从地图上消失，并在当前游戏得分上加上分数。



中国知网　https://www.cnki.net

## 5.2 游戏 AI 智能寻路算法

吃豆人游戏 AI 的智能寻路算法由 Unity 中的 ML-Agent 来辅助完成。ML-Agent 工具包是一款开源的 Unity 插件，应用在 Unity 平台上的机器学习工具包。游戏开发者可以使用该工具包来辅助游戏 AI 的开发设计，在 Unity 中通过一些易用的 Python API 来控制机器学习算法来训练游戏 AI，也可以为游戏 AI 添加训练好的算法模型，使得游戏 AI 的性能进一步提升。gym_unity 是将 gym 框架应用到 Unity 平台的 Python 工具包，可以将用 Python 实现的本地算法应用到 Unity 平台中。

### 5.2.1 吃豆人游戏 AI 寻路策略分析

吃豆人游戏的目标是在规避掉所有的鬼怪的情况下，在地图上的迷宫中搜索并吃掉地图上所有的豆子。地图上存在能量药丸，吃豆人吃掉它后能使鬼怪进入受惊状态。此时，吃豆人再遇上鬼怪时，鬼怪会被送到起始位置而吃豆人不会死亡。所以，根据鬼怪是否处于受惊状态，吃豆人的行为策略存在一定的变化。

第一种吃豆人使用比较保守的策略，不论鬼怪是否处于受惊状态，吃豆人都选择避开鬼怪的策略，此种策略能够使得吃豆人以较高的概率完成任务，但是，规避鬼怪需要行进更多的路径，导致吃豆人搜索出的路径不一定时最短的。

第二种是比较激进的策略，当鬼怪处于受惊状态时，吃豆人遇上鬼怪不改变行进路线将鬼怪送回起始点。由于在本次研究中设计的算法没有考虑到让吃豆人记录鬼怪的处于受惊状态的时间，所以，吃豆人并不知道何时触碰鬼怪是安全的，这会导致吃豆人搜索成功的概率降低。但是，当吃豆人成功将鬼怪送回起始点时，吃豆人不需要更改当前的行进路线，搜索得到的路径会更短。

### 5.2.2 吃豆人游戏 AI 设计实现

应用 ML-Agents 来开发吃豆人游戏 AI 时，需要将吃豆人游戏 AI 的脚本重新改写。将改写好的脚本挂到吃豆人的预制体上后，会出现一个新的脚本 BehaviorParamters，需要在这个脚本中配置好的动作空间、观测空间、最大训练步数，上述配置的动作空间与吃豆人脚本中挂载脚本中的动作空间维度一致。

将 Unity 中制作好的吃豆人场景编译成可执行文件，用 gym_unity 中的 UnityEnvironment() 函数进行封装。该函数第一个参数为吃豆人可执行文件的路径，第二个参数是 Unity 平台与本地训练算法通信的端口号，一般为 5005，还需要设置随机种子数和连接超时时间。如果训练的时候开启了图形渲染功能，将训练时的游戏画面渲染出来，将会极大地影响 CPU 和 GPU 运算速度，所以上述函数中 no_graphics 项需要设置为 True。最后，再利用 UnityToGymWrapper() 函数对吃豆人运行环境进行封装。

游戏 AI 算法部分沿用本此研究所提出的 Minimax-Q* 算法，算法的主要循环部



中国知网 https://www.cnki.net

分不变，算法主要由四部分组成：Q*-Learning 模块、A* 算法模块、线性规划模块、动作执行模块、以及数据采集模块。游戏 AI 的架构如图 5.3 所示。

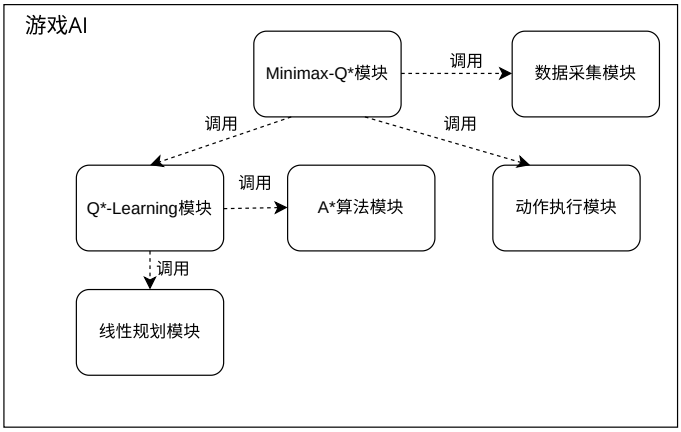



图 5.2　吃豆人游戏 AI 架构图

游戏 AI 首先初始化参数，使用 $\epsilon$-greedy 策略来选择动作，执行动作后，记录当前状态和目前获得的奖励并启用 A* 算法模块计算当前位置到目标位置的最短距离，计算出最短距离长度；利用线性规划模块更新动作选择概率，先根据当前的 $Q^*$ 值来更新 $V$ 值，然后根据当前奖励和路径损耗来更新 $Q^*$ 表。算法的主要流程图如图 5.4 所示。

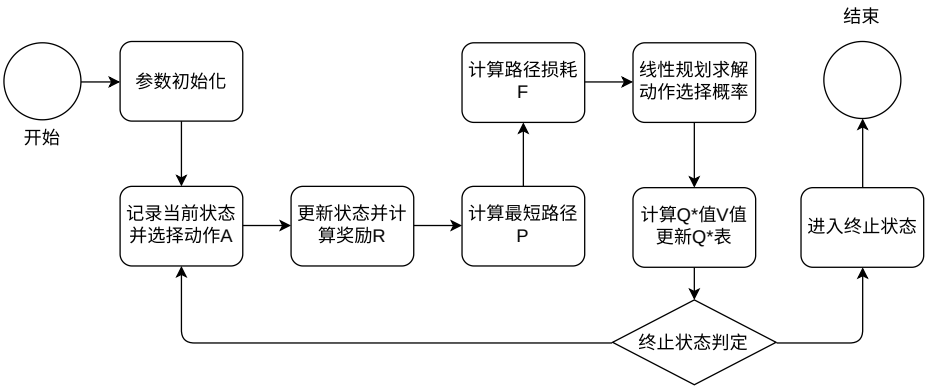



图 5.3　Minimax-Q* 算法流程图

与前几章的算法设计相比，本次应用的不同之处是引入了 gym 包，在具体的算法实现过程中需要使用 gym 中的.observation_space 属性和.action_space 属性，来获得吃豆人环境的状态空间维度和动作空间维度，并利用环境的.step() 函数执行一帧游戏。游戏 AI 的算法参数设定如下：训练步长 $\alpha$ 为 0.1、折扣因子 $\gamma$ 为 0.9、路径影响因子 $\omega$ 为 0.9，$\epsilon$-greedy 参数 $\epsilon$ 为 0.05。



中国知网　https://www.cnki.net

### 5.2.3 吃豆人游戏 AI 运行过程分析

与第四章算法测试环境 2 相似，应用环境中的鬼怪具备一定的攻击性，当鬼怪检测到吃豆人处于附近的位置时，鬼怪会朝着吃豆人所在的位置移动。鬼怪的最佳动作选择由攻击概率 $\tau$ 和逃跑概率 $\delta$ 决定，在本次应用环境中，鬼怪的攻击概率 $\tau$ 为 0.9。当吃豆人靠近鬼怪时，鬼怪有极大的可能去攻击吃豆人。

除此之外，加入另一个参数逃跑概率 $\delta$ 控制鬼怪的最佳动作选择。当吃豆人吃到能量药丸时，鬼怪处于受惊状态。在该状态下，鬼怪碰到吃豆人会返回到起始点，此时，鬼怪有极大的可能躲避吃豆人。本次应用环境中，逃跑概率 $\delta$ 设置为 0.8。吃豆人游戏 AI 训练完成后，进入测试部分并将游戏画面渲染出来以观察游戏 AI 算法的执行过程。吃豆人游戏 AI 的运行环境如表 5.1 所示。

表 5.1　吃豆人游戏 AI 执行环境

| 名称 | 配置 | |
|---|---|---|
| CPU | 型号 | AMD Ryzen 7 4800H |
| | 主频 | 2.90GHz |
| | RAM | 16.0GB |
| 开发环境 | Pycharm、Unity3D | |
| 操作系统 | Windows 10 | |
| 编程语言 | Python3.7 | |
| Python 包 | numpy、json、tkinter、pulp、gym、gym_unity、heapq、ML-Agents | |

游戏 AI 在运行过程中，吃豆人会在地图迷宫中巡航。巡航过程中，吃豆人会经历多种游戏状态，本文研究列举四种特殊的状态进行分析研究，分别是：吃豆人躲避多个鬼怪、吃豆人的保守策略、吃豆人遭遇堵截以及吃豆人完成搜索任务。

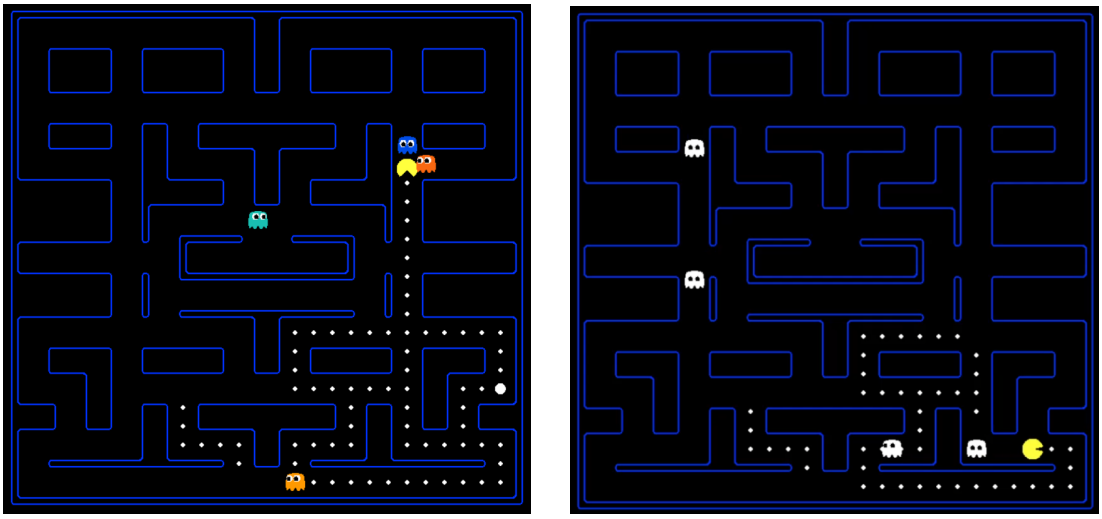
当吃豆人检测到附近有鬼怪时，会改变当前的移动方向以躲避鬼怪的追击。当遭遇多只鬼怪一同追击时，例如在一个 T 字路口处，一只鬼怪向西行而另一只鬼怪向南行，吃豆人首先会将朝东的移动改为向西移动，当吃豆人发现有鬼怪向南行时，放弃向北行，转而向南移动，从而躲避了两只鬼怪的同时追击。吃豆人游戏 AI 躲避多个鬼怪过程如图 5.5(a) 所示。

当吃豆人吃到能量药丸时，鬼怪则会处于受惊状态。在此状态下，鬼怪会变为白色并且移动速度减慢，而鬼怪碰上吃豆人后会返回起始点。由于设置了逃跑概率，当鬼怪处于受惊状态时，鬼怪会自主躲避吃豆人。但在测试过程中，当鬼怪处于受惊状态时，吃豆人在某些游戏回合中依然会选择改变移动方向或者会减慢自身移动速度，



中国知网 https://www.cnki.net

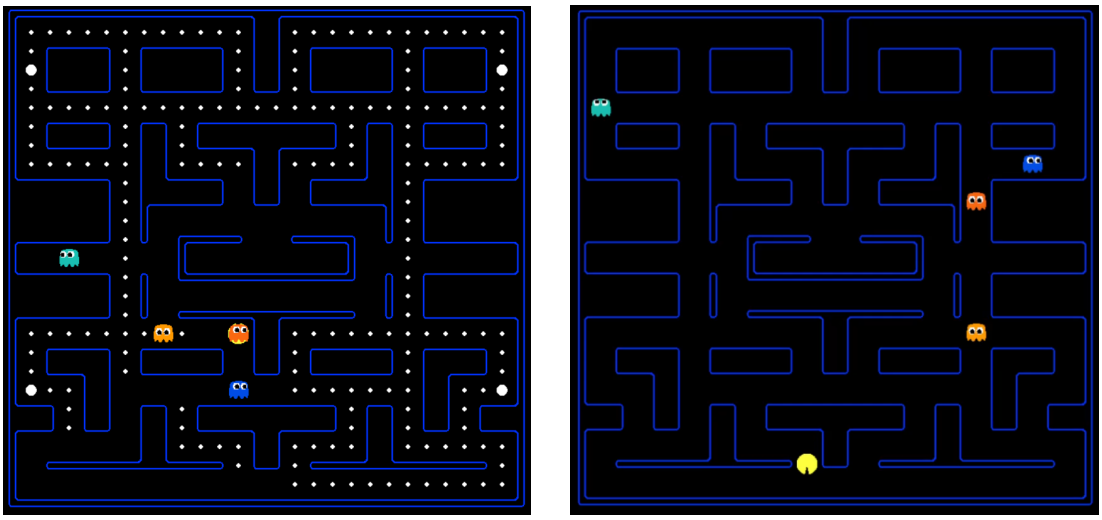
以避免撞上鬼怪。吃豆人的行为策略属于一种保守的策略。吃豆人游戏 AI 躲避受惊鬼怪的过程如图 5.5(b) 所示。



(a) 吃豆人躲避多个鬼怪　　(b) 吃豆人的保守策略

图 5.4　吃豆人游戏过程一

由于鬼怪在检测到吃豆人在附近时，会朝着吃豆人所在的位置移动。所以，在某些情况下，吃豆人会因遭遇鬼怪堵截而导致搜索任务失败。例如，当吃豆人位于一个 L 形的迷宫通道时，会遭到多个的鬼怪两头堵截，吃豆人无法逃避鬼怪的追捕从而导致搜索任务失败。吃豆人受到多个鬼怪堵截的过程如图 5.6(a) 所示。



(a) 多个鬼怪堵截　　(b) 完成搜索任务

图 5.5　吃豆人游戏过程二

在大部分情况下，吃豆人游戏 AI 能够规避所有鬼怪的情况下在地图中完成巡航，并吃掉地图中所有的豆子。在该情况下，吃豆人能够完成搜索任务，吃豆人游戏 AI



中国知网　https://www.cnki.net

完成搜索任务的情形如图 5.6(b) 所示。

## 5.3 吃豆人游戏 AI 运行结果分析

本次吃豆人游戏 AI 应用测试，根据鬼怪数量的不同、鬼怪行为规则的不同分为四组对照组。每组的吃豆人游戏 AI 进行 20 次搜索任务，根据搜索任务的完成成功率、完成任务的路径长度等指标来评价吃豆人游戏 AI 的运行结果。第一组 RG-4 为参照组，应用测试的设置为：地图上鬼怪数量为 4，鬼怪行为规则为随机移动。第二组 DG-4 的设置为：地图上鬼怪数量为 4，鬼怪行为规则为目的性移动。第三组 RG-2 的设置为：地图上鬼怪数量为 2，鬼怪行为规则为随机移动。第四组 DG-2 设置为：地图上鬼怪数量为 2，鬼怪行为规则为目的性移动。吃豆人游戏 AI 执行结果如表 5.2 所示，表中最短路径和平均路径都是搜索任务完成情况下的统计。

表 5.2　吃豆人游戏 AI 执行结果

| 组别 | 成功率 | 最短路径长度 | 平均路径长度 |
|---|---|---|---|
| RG-4 | 17/20 | 303 | 341.9 |
| DG-4 | 6/20 | 319 | 327.8 |
| RG-2 | 19/20 | 299 | 325.3 |
| DG-2 | 12/20 | 303 | 331.2 |

在第一组应用测试中，鬼怪的行为规则为随机移动，对比其他组的测试，游戏成功率明显更高，但是吃豆人的平均搜索路径最长。第二组应用测试中，鬼怪的行为规则变为目的性移动，鬼怪的数量对比第一组不变。对比第一组的结果，第二组吃豆人的游戏成功率明显下降，且规划出的最短路径的长度明显增加。第三组的游戏环境中的鬼怪数量对比第一组减少，鬼怪移动规则为随机移动，第三组吃豆人的游戏成功率有少量提高，并且吃豆人规划出的最短路径和平均路径的长度都减少。第四组游戏环境中的鬼怪数量为 2、鬼怪为目的性移动，对比第三组结果，第四组吃豆人的游戏成功率下降明显，但是搜索成功时的最短路径和平均路径下降幅度较小。

综合上述测试结果，鬼怪数量和鬼怪的移动规则都会对吃豆人游戏 AI 的游戏表现有一定程度上的影响。鬼怪行为规则的变化对比鬼怪数量的变化，更能够影响吃豆人游戏 AI 的游戏表现。但是，鬼怪行为规则对吃豆人游戏 AI 的影响力受到鬼怪数量的影响。当鬼怪数量越多，鬼怪表现出的攻击性越强，吃豆人游戏 AI 更加容易进入死角当中无法逃脱，吃豆人游戏 AI 的路径搜索效果将会大幅度下降。原因在于 Minimax-Q* 算法处理多方博弈能力不足，没有将多个鬼怪的联合动作的收益降到最低。在以后的研究工作中，可以考虑引入 QMIX 算法[70]来改善这个问题。为每个智能



中国知网 https://www.cnki.net

体都赋予一个值函数，然后将这些值函数进行整合，得到联合值函数，再利用 QMIX 算法进行优化。

## 5.4　本章小结

本章首先简要介绍了常用的 3D、2D 游戏制作平台 Unity，并利用 Unity 平台设计了吃豆人游戏 AI 的应用场景。然后，将本文所提出的算法应用到设计好的游戏场景中。将算法应用到吃豆人游戏 AI 后，进行测试评估并渲染出游戏画面，观察游戏 AI 行为，分析吃豆人游戏 AI 在游戏场景中不同的运行过程。通过调整各项参数来观察吃豆人游戏 AI 的游戏表现的变化，分析参数变化和游戏环境变化对吃豆人游戏行为变化的影响。最后，对吃豆人游戏 AI 的运行结果进行了总结。



中国知网　https://www.cnki.net

中国知网 https://www.cnki.net

# 第六章　总结与展望

## 6.1　总结

游戏行业是一个充满朝气的行业，存在着巨大的潜力和价值供各个领域的研究者进行深挖。游戏有着模拟现实的特性，可以将现实世界中不易于实现的实验迁移到虚拟世界中进行仿真，减轻了实验设置的负担与开销，便于研究工作的推进。游戏 AI 与学术 AI 的融合，也使得游戏的研究价值被进一步的提高。不少学术 AI 的测试环境或应用环境都是由游戏构成。本文研究是以游戏中的智能寻路算法作为研究对象，探讨游戏 AI 如何规避游戏环境中的动态障碍物。

在研究工作开始前，首先通过文献阅读了解了游戏行业的发展现状以及国内外的探究现状。本文通过介绍游戏行业的各项数据，阐述了游戏行业的巨大潜力；通过介绍游戏实现技术和游戏应用的多样性，描述了游戏行业的多元性。接下来，本文从实现技术和应用场景两方面，介绍了游戏和游戏 AI 的国内外研究现状。在第二章中，本文对游戏 AI 的实现技术进行了划分，并对这些技术的实现算法的主要流程以及基本的数学原理进行了简要地介绍。

本文的研究过程用到了最佳优先搜索算法 A* 算法，A* 算法广泛应用于在静态图中搜索两节点之间的最短路径。在进行路径搜索时，算法将下一步可选择的节点存入开放列表中。根据评估函数对开放列表中的节点进行评估，用以选择下一步的节点拓展搜索路径，并将该节点从优先队列中剔除，加入到关闭列表以表示该节点被访问过。算法依此往复不断循环，直到达到目标节点。强化学习算法以与环境不断的交互的方式进行学习，强化学习智能体通过与环境交互获得经验，根据经验来修正与环境交互的策略，从而来达到某个需要被完成的目标。Q-Learning 算法是经典的强化学习算法，直接利用最优动作值函数来更新动作值函数，一定程度上保障了算法的收敛性。Minimax-Q 算法将马尔科夫博弈框架应用到强化学习当中，以最小化对手动作奖励为目标来优化游戏 AI 的行为策略。该算法的引入使得游戏 AI 能够处理一些可移动障碍物具有一定智能的情况。

本文研究的第一项工作是将 Q-Learning 算法和启发式搜索算法相结合，提出 Q*-Learning 算法，来改善传统的游戏 AI 寻路算法不能够规避可移动障碍物的问题。本文尝试将 Q-Learning 算法和 A* 算法的优势相结合，对比基于 Q-Learning 的游戏寻路 AI，该算法具备路径规划的能力，能够在吃豆人游戏中搜索得到的平均路径更短；对比基于 A* 算法的游戏 AI，该算法具备适应环境改变的能力，能够有效规避动态的障碍物。本文所提出的算法对比传统的游戏 AI 寻路算法的主要优势如下：(1) 结合 A*



中国知网　https://www.cnki.net

算法的路劲规划能力，以路径损耗作为优化目标游戏 AI 的寻路更具目的性，一定程度上缩短了游戏 AI 的训练过程，游戏 AI 能够更加快速地搜索到目的地。(2) 引入了强化学习算法 Q-Learning 使得游戏 AI 对于游戏环境的改变，对于动态障碍物位置的变化更加敏感，游戏 AI 规避动态障碍物的能力变强。

本文研究的第二项工作是引入零和博弈算法，以进一步提升游戏 AI 的动态避障能力。Minimax-Q 算法将极大化极小理论与强化学习算法 Q-Learning 相结合，将马尔科夫博弈框架应用到马尔可夫决策过程当中，通过极小化对手动作收益来极大化游戏 AI 能够得到得游戏分数，该算法解决了经典博弈游戏石头剪刀布。将 Minimax-Q 算法中的极大化极小的特性引入到 Q*-Learning 算法中后，吃豆人游戏 AI 能够考虑最小化可移动障碍物的动作收益，并以此来更新自身的行为策略。当地图上的障碍物被赋予一定智能后，本文所提出的算法的游戏表现要更佳。

本文研究的最后一部分内容是算法应用，利用 Unity 平台设计吃豆人游戏地图，然后将本文所提出的算法应用到该吃豆人游戏场景中，并对吃豆人游戏 AI 的运行过程和算法应用的结果进行分析，分析游戏 AI 运行过程和运行结果中存在的问题，并考虑之后的改进方法。

## 6.2 展望

本文的研究工作存在许多不足之处。首先是没有考虑测试算法的迁移性，单纯地只在吃豆人的游戏环境中进行测试；其次是在算法训练过程中，由于环境奖励比较稀疏导致算法的收敛速度较慢；然后也没有考虑多个吃豆人游戏 AI 相互配合，或者玩家与游戏 AI 相互配合的情形等等。也存在没有利用一些寻路效果的算法基准，进行比对实验的问题。针对上述不足之处未来的研究工作主要包含以下几部分内容：

(1) 在未来的研究工作中考虑引入迁移学习的内容，利用迁移学习的算法来提升游戏 AI 寻路算法的对不同环境的兼容性和迁移性，设置三到四个不同的游戏环境，对算法的兼容性和迁移性进行测试。选择 JSP 算法或者 SOTA 算法等作为比对实验的基准，来增强结果的说服力。

(2) 引入一些对游戏环境探索更加高效的算法，例如：UCB-Q Learning 算法[71]来缓解游戏 AI 对环境探索不够充分的问题。考虑改进游戏奖励函数，对游戏环境中的奖励进行填充来解决因环境奖励稀疏而导致的算法收敛较慢的问题。同样可以引入深度神经网络来提升算法模型对环境的拟合能力，进一步提升算法效率。也可以在求解线性规划问题的时候，添加一些新的约束条件，使得游戏 AI 选到最优动作的概率进一步提高。

(3) 可以对研究对象的数量进行扩展，从单智能体研究向多智能体研究进行拓展。可以引入一些多智能体强化学习算法，例如：QMIX 算法，来进行多个游戏 AI 相互



中国知网 https://www.cnki.net

配合对游戏环境进行有效探索的研究。

游戏 AI 寻路问题的研究也可以与其他研究方向进行交叉结合，潜在的结合对象可以是智能机器人寻路[72]或者无人机寻路。利用游戏来模拟真实的寻路环境，为游戏添加真实的物理引擎，通过游戏 AI 在虚拟环境中的寻路来仿真智能机器人在现实环境中的寻路，为机器人寻路算法的调试提供了极大的便利。同时，可以将强化学习算法引用到实际的机器人寻路控制当中，来推进强化学习算法的落地。

对于游戏 AI 本身的研究而言，游戏 AI 的行为应该拟人化，游戏 AI 的操作应当和普通人类玩家的操作水平相近。拥有与人类相似的操作特性和操作习惯，游戏 AI 才能更好地为游戏和人类服务。例如，在某些严肃游戏当中，游戏 AI 存在的目标是为了训练某些行业人员的某项专业技能。如果游戏 AI 的游戏水平过于强大或者过于弱小，对于专业技能的训练效果都是不够理想的。这时候就需要拟人化的游戏 AI，来配合从业人员的训练。游戏 AI 拟人化，能够模仿出人类的动作，也是游戏 AI 更加智能的一种体现。



中国知网　https://www.cnki.net

中国知网 https://www.cnki.net

# 参考文献

[1] BROWNE C, POWLEY E J, WHITEHOUSE D, et al. A survey of monte carlo tree search methods [J]. IEEE Trans. Comput. Intell. AI Games, 2012, 4(1): 1-43.

[2] 李婧璇.《2021 年中国游戏产业报告》发布——防沉迷新规落地，用户结构渐趋合理[EB/OL]. 2021.

[3] DUAN H, LI J, FAN S, et al. Metaverse for social good: A university campus prototype[J]. CoRR, 2021, abs/2108.08985.

[4] MEHRABI N, MORSTATTER F, SAXENA N, et al. A survey on bias and fairness in machine learning[J]. ACM Comput. Surv., 2021, 54(6): 115:1-115:35.

[5] LIEBANA D P, SAMOTHRAKIS S, TOGELIUS J, et al. General video game AI: competition, challenges and opportunities[M]//SCHUURMANS D, WELLMAN M P. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. AAAI Press, 2016: 4335-4337.

[6] VINYALS O, BABUSCHKIN I, CZARNECKI W M, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning.: Vol. 575[Z]. 2019: 350-+.

[7] LIN J, YUAN Y, ZOU Z. Meingame: Create a game character face from a single portrait[A]. 2021. arXiv: 2102.02371.

[8] LELIS L H S, OTTEN L, DECHTER R. Predicting the size of depth-first branch and bound search trees[M]//ROSSI F. IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013. IJCAI/AAAI, 2013: 594-600.

[9] ZHANG Z, SMEREKA J M, LEE J, et al. Game tree search for minimizing detectability and maximizing visibility[J]. Auton. Robots, 2021, 45(2): 283-297.

[10] PLAAT A, SCHAEFFER J, PIJLS W, et al. Best-first and depth-first minimax search in practice[J]. CoRR, 2015, abs/1505.01603.

[11] KNUTH D E, MOORE R W. An analysis of alpha-beta pruning: Vol. 6[Z]. 1975: 293-326.

[12] BROWNE C, POWLEY E J, WHITEHOUSE D, et al. A survey of monte carlo tree search methods [J]. IEEE Trans. Comput. Intell. AI Games, 2012, 4(1): 1-43.

[13] FEINBERG A. Markov decision processes: Discrete stochastic dynamic programming (martin l. puterman)[J]. SIAM Rev., 1996, 38(4): 689.

[14] CHASLOT G, BAKKES S, SZITA I, et al. Monte-carlo tree search: A new framework for game AI [M]//DARKEN C, MATEAS M. Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, October 22-24, 2008, Stanford, California, USA. The AAAI



中国知网 https://www.cnki.net

Press, 2008.

[15] SILVER D, HUANG A, MADDISON C J, et al. Mastering the game of go with deep neural networks and tree search[J]. Nat., 2016, 529(7587): 484-489.

[16] SILVER D, HUBERT T, SCHRITTWIESER J, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play[J]. Science, 2018, 362(6419): 1140-1144.

[17] BADIA A P, PIOT B, KAPTUROWSKI S, et al. Agent57: Outperforming the atari human benchmark[M]//Proceedings of Machine Learning Research: Vol. 119 Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event. PMLR, 2020: 507-517.

[18] OUESSAI A, SALEM M, MORA A M. Evolving action pre-selection parameters for MCTS in real-time strategy games[J]. Entertain. Comput., 2022, 42: 100493.

[19] TAO J, XIONG Y, ZHAO S, et al. Xai-driven explainable multi-view game cheating detection[M]// IEEE Conference on Games, CoG 2020, Osaka, Japan, August 24-27, 2020. IEEE, 2020: 144-151.

[20] ZHAO S, WU R, TAO J, et al. Multi-source data multi-task learning for profiling players in online games[M]//IEEE Conference on Games, CoG 2020, Osaka, Japan, August 24-27, 2020. IEEE, 2020: 104-111.

[21] DEVLIN S, GEORGESCU R, MOMENNEJAD I, et al. Navigation turing test (NTT): learning to evaluate human-like navigation[J]. CoRR, 2021, abs/2105.09637.

[22] CHOI D, KÖNIK T, NEJATI N, et al. A believable agent for first-person shooter games[M]// SCHAEFFER J, MATEAS M. Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference, June 6-8, 2007, Stanford, California, USA. The AAAI Press, 2007: 71-73.

[23] 谢吉刚. 一种基于 A* 算法的 RPG 游戏寻路算法[J]. 电子测试, 2018.

[24] HARSANI P, MULYANA I, ZAKARIA D. Fuzzy logic and a* algorithm implementation on goat foraging games[J]. IOP Conference Series: Materials Science and Engineering, 2018, 332.

[25] HARABOR D D, GRASTIEN A. Online graph pruning for pathfinding on grid maps[M]// BURGARD W, ROTH D. Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011. AAAI Press, 2011.

[26] 蔡礼权. 基于模糊逻辑推理行为树的游戏 AI 建模与应用[D]. 华南理工大学, 2017.

[27] AGIS R A, GOTTIFREDI S, GARCÍA A J. An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games[J]. Expert Syst. Appl., 2020, 155: 113457.

[28] 孙一铃. 基于 Expectimax 搜索的非完备信息博弈算法的研究[D]. 北京交通大学, 2021.

[29] PEPELS T, WINANDS M H M, LANCTOT M. Real-time monte carlo tree search in ms pac-man [J]. IEEE Trans. Comput. Intell. AI Games, 2014, 6(3): 245-257.



中国知网 https://www.cnki.net

[30] SOEMERS D J N J, SIRONI C F, SCHUSTER T, et al. Enhancements for real-time monte-carlo tree search in general video game playing[M]//IEEE Conference on Computational Intelligence and Games, CIG 2016, Santorini, Greece, September 20-23, 2016. IEEE, 2016: 1-8.

[31] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing atari with deep reinforcement learning [J]. CoRR, 2013, abs/1312.5602.

[32] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal policy optimization algorithms[J]. CoRR, 2017, abs/1707.06347.

[33] HAARNOJA T, ZHOU A, ABBEEL P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor[J]. CoRR, 2018, abs/1801.01290.

[34] WU D, LISSER A. Using CNN for solving two-player zero-sum games[J]. Expert Syst. Appl., 2022, 204: 117545.

[35] 段伟浩. 基于深度强化学习的多智能体动态寻路算法[J]. 计算机仿真, 2023, 40(01):441-446+473.

[36] CASTRO F E V. Exploring the use of finite-state machines and game creation to teach computational thinking in middle schools[M]//BECKER B A, QUILLE K, LAAKSO M, et al. ITiCSE 2022: Innovation and Technology in Computer Science Education, Dublin, Ireland, July 8 - 13, 2022, Volume 2. ACM, 2022: 618.

[37] CARROLL J L, LONG D D E. Theory of finite automata with an introduction to formal languages [M]. Prentice Hall, 1989.

[38] LI P, QIAN W, RIEDEL M D, et al. The synthesis of linear finite state machine-based stochastic computational elements[M]//Proceedings of the 17th Asia and South Pacific Design Automation Conference, ASP-DAC 2012, Sydney, Australia, January 30 - February 2, 2012. IEEE, 2012: 757-762.

[39] ALVAREZ-ALVAREZ A, TRIVIÑO G, CORDÓN O. Human gait modeling using a genetic fuzzy finite state machine[J]. IEEE Trans. Fuzzy Syst., 2012, 20(2): 205-223.

[40] BIGGAR O, ZAMANI M, SHAMES I. Modular decomposition of hierarchical finite state machines [J]. CoRR, 2021, abs/2111.04902.

[41] PLCH T, MARKO M, ONDRÁCEK P, et al. Modular behavior trees: Language for fast AI in open-world video games[M]//SCHAUB T, FRIEDRICH G, O'SULLIVAN B. Frontiers in Artificial Intelligence and Applications: Vol. 263 ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). IOS Press, 2014: 1209-1210.

[42] SEKHAVAT Y A. Behavior trees for computer games[J]. Int. J. Artif. Intell. Tools, 2017, 26(2): 1730001:1-1730001:28.



中国知网 https://www.cnki.net

[43] IOVINO M, SCUKINS E, STYRUD J, et al. A survey of behavior trees in robotics and AI[J]. Robotics Auton. Syst., 2022, 154: 104096.

[44] COLLEDANCHISE M, ÖGREN P. How behavior trees modularize robustness and safety in hybrid systems[M]//2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014. IEEE, 2014: 1482-1488.

[45] SWIECHOWSKI M, GODLEWSKI K, SAWICKI B, et al. Monte carlo tree search: A review of recent modifications and applications[J]. CoRR, 2021, abs/2103.04931.

[46] KOCSIS L, SZEPESVÁRI C. Bandit based monte-carlo planning[M]//FÜRNKRANZ J, SCHEFFER T, SPILIOPOULOU M. Lecture Notes in Computer Science: Vol. 4212 Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings. Springer, 2006: 282-293.

[47] VIAPPIANI P. Thompson sampling for bayesian bandits with resets[M]//PERNY P, PIRLOT M, TSOUKIÀS A. Lecture Notes in Computer Science: Vol. 8176 Algorithmic Decision Theory - Third International Conference, ADT 2013, Bruxelles, Belgium, November 12-14, 2013, Proceedings. Springer, 2013: 399-410.

[48] URTEAGA I, WIGGINS C H. Bayesian bandits: balancing the exploration-exploitation tradeoff via double sampling[J]. CoRR, 2017, abs/1709.03162.

[49] MEHRABI N, MORSTATTER F, SAXENA N, et al. A survey on bias and fairness in machine learning[J]. ACM Comput. Surv., 2021, 54(6): 115:1-115:35.

[50] MOHR F, VAN RIJN J N. Learning curves for decision making in supervised machine learning - A survey[J]. CoRR, 2022, abs/2201.12150.

[51] ABUKMEIL M, FERRARI S, GENOVESE A, et al. A survey of unsupervised generative models for exploratory data analysis and representation learning[J]. ACM Comput. Surv., 2021, 54(5): 99:1-99:40.

[52] RAMÍREZ S C, YANG S, ELIZONDO D A. Semi-supervised deep learning for image classification with distribution mismatch: A survey[J]. CoRR, 2022, abs/2203.00190.

[53] BUFFER O, PIETQUIN O, WENG P. Reinforcement learning[J]. CoRR, 2020, abs/2005.14419.

[54] LECUN Y, BENGIO Y, HINTON G E. Deep learning[J]. Nat., 2015, 521(7553): 436-444.

[55] CARUANA R, LAWRENCE S, GILES C L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping[M]//LEEN T K, DIETTERICH T G, TRESP V. Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA. MIT Press, 2000: 402-408.

[56] KWA H L, KIT J L, BOUFFANAIS R. Balancing collective exploration and exploitation in multi-agent and multi-robot systems: A review[J]. Frontiers Robotics AI, 2021, 8: 771520.



中国知网 https://www.cnki.net

[57] LEVINE S, KUMAR A, TUCKER G, et al. Offline reinforcement learning: Tutorial, review, and perspectives on open problems[J]. CoRR, 2020, abs/2005.01643.

[58] BUFFER O, PIETQUIN O, WENG P. Reinforcement learning[J]. CoRR, 2020, abs/2005.14419.

[59] SUTTON R S, BARTO A G. Adaptive computation and machine learning: Reinforcement learning - an introduction[M]. MIT Press, 1998.

[60] SUTTON R S, BARTO A G. Reinforcement learning: An introduction[J]. IEEE Trans. Neural Networks, 1998, 9(5): 1054-1054.

[61] PERKINS T J, PENDRITH M D. On the existence of fixed points for q-learning and sarsa in partially observable domains[M]//SAMMUT C, HOFFMANN A G. Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002. Morgan Kaufmann, 2002: 490-497.

[62] BARTO A G, SUTTON R S, WATKINS C J C H. Sequential decision probelms and neural networks [M]//TOURETZKY D S. Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]. Morgan Kaufmann, 1989: 686-693.

[63] FALLAS-MOYA F, DUNCAN J, SAMUEL T K, et al. Measuring the impact of memory replay in training pacman agents using reinforcement learning[M]//XLVII Latin American Computing Conference, CLEI 2021, Cartago, Costa Rica, October 25-29, 2021. IEEE, 2021: 1-8.

[64] MCEACHERN A. Synthesis lectures on games and computational intelligence: Game theory: A classical introduction, mathematical games, and the tournament[M]. Morgan & Claypool Publishers, 2017.

[65] WASHBURN A, WOOD K. Two-person zero-sum games for network interdiction[J]. Oper. Res, 1995, 43(2): 243-251.

[66] ADLER I. The equivalence of linear programs and zero-sum games[J]. Int. J. Game Theory, 2013, 41(1): 165-177.

[67] LITTMAN M L. Markov games as a framework for multi-agent reinforcement learning[M]// COHEN W W, HIRSH H. Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994. Morgan Kaufmann, 1994: 157-163.

[68] ZHU Y, ZHAO D. Online minimax Q network learning for two-player zero-sum markov games[J]. IEEE Trans. Neural Networks Learn. Syst., 2022, 33(3): 1228-1241.

[69] 史宝明. Unity3D 随机寻路算法设计[J]. 吉林师范大学学报 (自然科学版), 2022.

[70] RASHID T, SAMVELYAN M, DE WITT C S, et al. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning[J]. CoRR, 2018, abs/1803.11485.

[71] SAITO K, NOTSU A, UBUKATA S, et al. Performance investigation of UCB policy in q-learning



中国知网 https://www.cnki.net

[M]//LI T, KURGAN L A, PALADE V, et al. 14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, December 9-11, 2015. IEEE, 2015: 777-780.

[72] 张庆. 融合 JPS 和改进 A* 算法的移动机器人路径规划[J]. 计算机科学与探索, 2021, 15(11):2233-2240.



中国知网 https://www.cnki.net