廣東工業大學
GUANGDONG UNIVERSITY OF TECHNOLOGY

硕士学位论文

（专业学位）

基于 Unity3D 游戏人工智能的研究与应用

作　　者　　姓　　名　：　朱杰

导　　师　　姓　　名　：　余荣

学科（专业）或领域名称　：　控制工程

论　文　答　辩　年　月　：　2020 年 5 月

分类号：　　学校代码：11845

UDC：　　密级：　　学　　号：2111704343

广东工业大学硕士学位论文

（工程硕士）

基于 Unity3D 游戏人工智能的研究与应用

朱杰

导师姓名（职称）：余荣 教授

黄斐瀚 高级工程师

学科(专业)或领域名称：控制工程

学生所属学院：自动化学院

答辩委员会主席：何昭水

论文答辩日期：2020 年 5 月

A Dissertation Submitted to Guangdong University of Technology for the Degree of Master

（Master of Engineering）

Research and Application on the Game AI Based on Unity3D Game Engine

Candidate: Zhu Jie

Supervisor: Prof. YuRong

May 2020

School of Automation
Guangdong University of Technology
Guangzhou, Guangdong, P. R. China, 510006

摘要

得益于近年来科学技术的发展，计算机的软件和硬件性能都得到了很大的提升，游戏人工智能相关的研究开始被人们所重视。游戏人工智能作为游戏中的重要组成部分，一直扮演着提升玩家体验的角色。但是开发人员也意识到，想要做出一个高度智能化的游戏人工智能，也是一项非常具有挑战性的任务。目前，商业游戏中常见的游戏人工智能设计方式主要有：有限状态机和行为树。这些方法能够得到市场的认可，原因是其设计出的游戏人工智能的行为完全可控，但是表现的灵活性不足。因此，一种新的游戏人工智能应运而生，那就是使用机器学习的方式，训练出满足我们需求的，并且灵活多变的游戏人工智能。本文将着重研究行为树和机器学习两种方式，设计出感知能力更加拟人化，行为灵活多变的射击游戏人工智能。并且基于 Unity3D 游戏开发引擎，开发一款射击游戏。主要研究工作体现在以下三个方面：

（1） 基于行为树技术设计一种射击游戏人工智能。为了使游戏中非玩家角色（Non-Player Characters, NPCs）体现出尽可能多的智能性，本文从 NPCs 的感知能力着手，在视觉感知和听觉感知上各提出一种具有实用价值的设计和实现方法。具体应用到游戏中，还设计了其他附加行为节点，使 NPCs 的行为表现更加完整。

（2） 运用机器学习中的相关技术设计游戏人工智能，主要是强化学习的方式。以训练投篮机器人为例，详细描述基于机器学习游戏人工智能的设计实现方法。另外，为了使强化学习的训练更加快速有效，结合使用了课程学习和好奇心等方法。通过多组不同组合的训练实验，展示出较为科学有效训练方法。

（3） 基于 Unity3D 引擎设计制作一款射击游戏，包括游戏的故事背景，游戏风格，基础的可玩性功能，最重要的是，将本文提出的行为树感知系统设计方法和基于机器学习的设计方法应用于该游戏的 NPCs 制作过程中。同时对比实验多种训练方式，从数据中分析最优的训练方法。在篮球运动员游戏人工智能的设计制作中，提出了行为树与机器学习相结合方法，将机器学习所得到的策略模型封装成行为树中的某些节点，将二者有机的结合，取长补短。最后，通过游戏整体运行效果的描述，展示出所有游戏人工智能的智能行为表现。

关键词：游戏人工智能；强化学习；行为树；Unity3D

Abstract

Benefit from the development of science and technology in recent years, the performance of computer software and hardware has been greatly improved, and related researches of Game Artificial Intelligence(AI) began to be paid attention to. Game AI, as an important part of the game, has been playing the role of improving the player experience. But developers are also aware that making a highly intelligent game AI is a challenging task. At present, the common game AI design methods in commercial games mainly include: Finite State Machine and Behavior Tree. These methods are recognized by the market because the game AI designed by them can control the behavior completely, but it is not flexible enough. As a result, a new kind of game AI emerged, which is to use Machine Learning to train a flexible game AI to meet our needs. This paper will focus on two ways of Behavior Tree and Machine Learning, and design shooting game AI with more anthropomorphic perception and flexible behaviors. Based on Unity3D game development engine, develop a shooting game. The main research work is reflected in the following four aspects:

（1） Designing a shooting game AI based on Behavior Tree method. In order to make Non-Player Characters(NPCs) embody as much intelligence as possible, this paper proposes a practical design and implementation method based on the perception ability of NPCs. Specifically applied to the game, other additional behavior nodes are designed for it to make the behavior of NPCs more completely.

（2） Designing the game AI by using relevant methods in Machine Learning, which is mainly a way of Reinforcement Learning. Taking the training shooting robot as an example, the design and implementation method based on machine learning game artificial intelligence is described in detail. In addition, in order to make the training of reinforcement learning faster and more effective, methods such as curriculum learning and curiosity are combined. The more scientific and effective training methods will be demonstrated through several training experiments of different combinations.

（3） Designing and implementing a shooting game, including the game's story background, game style, basic playability. And most importantly, apply the proposed

behavior tree perception system design method and the machine learning-based design method to the NPCs production process of the game. At the same time, a variety of training methods were compared and the optimal training methods were be analyzed from the data. In the design and implement of basketball player game artificial intelligence, a method of combining behavior tree and machine learning is proposed, which encapsulates the strategy model obtained by machine learning into some nodes in the behavior tree, and organically combines the two parts to complement each other. Finally, through the description of the overall running effect of the game, the intelligent behavior performance of all game AI is demonstrated.

Keywords: Game AI; Reinforcement Learning; Behavior Tree; Unity 3D

目录

Contents

第一章 绪论

1.1 研究背景和意义

随着科学技术的进步，计算机硬件和软件都得到了极大的提升，每年世界上发行的游戏质量也在不断提高，包括游戏的画面效果、玩法、音效和剧情等方面。特别是在画面效果方面，在当前技术发展的前提下，给予了玩家极致的体验[1]。据有名的游戏和媒体市场情报提供商提供的数据，游戏行业在 2018 年创造了 1379 亿美元的收入。专家预测，到 2022 年，游戏产业将产生 1960 亿美元的收入[2]。游戏行业不断地突破技术界限，激发创新思维，推动了整个电子行业软件和硬件技术的发展，各大互联网公司也纷纷涉足游戏行业，希望能够为世界玩家提供服务，从这个大市场中分一杯羹。由此可见，游戏产业的发展仍然处于上升趋势。

精美的游戏画面、创新的玩法固然能够获得玩家的喜爱，但是除了这些，我们还能朝着哪些方向努力呢？近年来，为了满足各大游戏厂商对机器性能的要求，不论是电脑主机还是移动端计算机，其硬件技术都有了大幅提升，让原本处于游戏末端的人工智能模块可以得到更多的算力[3]。这就引出了本文的研究方向，也就是游戏人工智能。如果从这个方向展开研究，不仅能够给游戏增添趣味性，还可以使玩家与游戏有一个良好的互动。一个优秀的游戏人工智能，还应该与玩家建立起一种感情纽带，让其成为玩家的陪伴，给予一种沉浸式的体验。

具体来说，一个优秀的游戏人工智能应具备多种拟人化的感知和行为方式，能够让玩家产生共鸣。感知方式主要有视觉和听觉两种方式，而行为方式则多种多样，有人也把非玩家角色（Non-Player Character, NPC）的寻路方式也作为游戏人工智能的一部分[4]。认为 NPC 的寻路方式也应该具备拟人化的方式，例如射击游戏中，NPC 应该找到一条能够躲避玩家攻击的道路，而不是单纯的最短路径规划。

目前，很多游戏公司的商业游戏中还是使用的工程学的方式开发游戏人工智能，主要有：有限状态机、模糊状态机、行为树（决策树）等[5]。一些极少数大型游戏公司，正在研究和使用机器学习相关的技术。例如，强化学习、课程学习和模拟学习等。更有开发人员希望能够将这种设计方式整理成一种方便使用的插件，让更过在机器学习领域研究不深入的游戏开发者也能训练出满足自己要求的 NPC。目前可以

看见的是，在著名的 Dota 2 游戏中，有些 NPC 就是使用神经网络和遗传算法等方式设计得到。可以预测到不久的将来，机器学习设计游戏人工智能的方式，在商业游戏中将会越来越普遍。

Unity 公司是率先将机器学习的方法应用到游戏人工智能设计中的公司之一。他们早在 2017 年，就推出了 Unity Machine Learning Agent（ML-Agent）插件，鼓励游戏开发者使用该插件进行对 NPC 的训练[6]。Untiy 作为全球最受欢迎的游戏开发引擎，就已经着手布局这方面研究工作，可见游戏人工智能光明的前景。

也有人表达了不一样的观点。他们认为游戏人工智能是一个伪命题，玩家在玩游戏的过程中，并不一定需要非常智能 NPC。并且，NPC 也不应该智能到玩家无法打败的程度。玩家败给 NPC 的唯一原因，就是没有找到它的“弱点”。从早期的游戏发展来看，都没有游戏人工智能的存在，NPC 在游戏中永远只是一个辅助的“调味料”。但是，更多的人依然相信游戏人工智能存在和发展的意义[7]。最具有说服力的是一件事情是，近些年棋牌游戏方面的游戏人工智能发展速度迅猛，相继出现了 Deep Blue（深蓝）和 Alpha Go。他们一举改变了传统 NPC 呆滞笨拙的固有观念，带给了人们更多的想象。另一方面，如果开发者能够设计出超级智能的 NPC，同样也有能力制作出符合玩家要求的稍微“低智能性”的 NPC。

从最近几年国内外热门的几款游戏来看，游戏人工智能的需求更加迫切。多人在线战术竞技游戏“英雄联盟（League of Legends）”游戏中，除了玩家对战模式之外，还有一种人机对战模式。理想状态下，NPC 应该具有与玩家差不多的竞技水平，双方胜率保持在 50%附近最佳。但是，实际上 NPC 在游戏中的表现相当愚蠢，无法根据当前的状况做出合理的决策行为，以至于让玩家在其中找不到任何乐趣。从另外一款兼具策略类型、射击类型和竞技类型特点的游戏 “绝地求生.刺激战场”来看，游戏的特点是需要大量的玩家同时在线，但是在游戏发布初期或者某些时段用户量不足的情况下，必须使用 NPC 来补充进入游戏。但是，其中的 NPC 作为真实玩家的对手显然是不合格的，另外一些类似的游戏也因为 AI 智能性不足的原因，玩家对游戏失去了兴趣，最后逐渐退出了市场。

总的来说，游戏人工智能已经得到了开发人员和各大游戏公司的重视，纷纷投入到该技术领域的研发工作[8]。并且能够分配给游戏人工智能的计算资源也在不断增加[9]。在游戏其他方面达到瓶颈之后，游戏人工智能必然能够重新俘获玩家的注意力，成为游戏增长的下一个核心因素。

无论如何，提升 NPC 智能性的呼声越来越大，不论是学院派还是实践派，都希望使用各种方式达到这个目的。而且从现在的技术发展水平来看，设计出具有高度智能性的游戏人工智能已经成为可能。

1.2 国内外研究现状

1.2.1 整体发展现状

游戏人工智能其实很早就存在。早在上个世纪 50 年代，游戏的发展尚处于早期，游戏本身的质量就很一般，虽然在当时已经是最先进的技术了，但是这仍然无法阻止开发者对智能化的 NPC 的追求。棋牌游戏中率先出现了跳棋和国际象棋等游戏，这其中都有 AI 的身影。

上世纪 70 年代，开始出现单机游戏，在这种只有玩家一个智能体的游戏中，具有智能行为的 NPC 无疑变得尤为重要。例如游戏 Hunt the Wumpus（猎人）中出现了敌人，限于当时的技术水平，敌人的移动是以存储的方式实现的。

后来各大游戏厂商设计的游戏难度越来越大，敌人的运动方式也越来越灵活多变，这种独特的游戏体验，得到了玩家的青睐，因此，设计智能化的游戏人工智能的思想开始普及开来。游戏 Galaxian 中，开发者就为敌人开发了复杂多样的运动方式，甚至是突破对方阵型。

实时战略游戏（RTS）的出现，给了游戏人工智能新的挑战，在有限的计算机算力中，计算机需要处理寻路、决策等复杂算法，这就促使一种新的游戏人工智能设计方法的出现---有限状态机。使用有限状态机的方式，可以非常方便的制作出一些简单的 NPC，虽然它有很多缺点，也有人说有限状态机的时代已经过去了，但是，在某些情况下，直到现在依然有游戏开发者使用这种方式。

取代有限状态机的是行为树技术。当开发者使用有限状态机进行设计开发的时候，发现稍微复杂的游戏人工智能，设计出来的有限状态机就像一张蜘蛛网，不仅开发的时候难以调试，在后期维护过程中，也是异常困难。行为树技术的出现，完美的解决了这一问题，NPC 的所有智能行为都源于一个根节点，然后按照设计逻辑，依次向树结构的叶节点方向执行，各个状态之间的改变都有非常清晰的逻辑，虽然在设计之初需要耗费一番心思整理整个逻辑，但是结果非常具有层次感，每种行为都被设计为某种类型的节点，复用性和可扩展性都非常的强。

将机器学习技术应用到游戏中的时间也很早。1997 年，IBM 公司公司开发了 Deep Blue（深蓝）人工智能，击败了国际象棋大师 Garry Kasparov。虽然有人否认深蓝是人工智能，这里不做讨论。但是后续出现的 Alpha Go，就毫无疑问是机器学习的产物，具体使用了机器学习中强化学习的方法，成功的战胜了世界上顶尖围棋大师[10]。

1.2.2 游戏决策系统研究现状

游戏决策系统最开始使用的是脚本的方式，开发人员在代码中，已经完全描述了一个 NPC 在游戏中的所有行为。这种原始的方法最大的缺点就是代码工作量过于繁重，游戏开发人员需要付出非常大的精力才能完成。并且缺乏复用性和维护性。后来，有限状态机技术的出现，大大的减少了开发人员的编码工作量，有一些研究工作者，实现了有限状态机的可视化编程，这就可以在一定程度上，将游戏本体的设计与游戏 AI 的开发分离开来。但是，随着游戏人工智能难度和复杂性的提升，有限状态机技术开始显现出来它的短板。复杂游戏 AI 的有限状态机就是一团乱麻。非常不利于调试和维护，并且可复用性也很差[11]。

Wenfeng Hu 等提出了一种基于组件的分层状态机[12]。主要是为了解决有限状态机在游戏开发中缺乏复用性和灵活性的情况。他们论文的创新之处在于将软件组件技术引入 FSM 的具体实现中，从而将 FSM 的状态和转换完全模块化。通过将简单的低级行为模块进行组合，实现高级智能体。他们还将游戏的设计与低级的游戏 AI 编程分离，以适应不断变化的游戏环境。虽然这种方法在一定程度上缓解了 FSM 设计游戏人工智能时的复杂性，有一定的模块复用功能，但是依然没有从根本上解决有限状态机无法同时处于多个状态的情况。

LGM Alvim 等提出了一种具有新的隶属度传递函数的模糊状态机（FuSM）。作者主要是研究了一种电子游戏角色的情感模型。然后根据这个情绪模型提出的一种新的模糊状态机。由于 FuSM 能够同时作用于多种状态，能够很好给予 NPC 很好的表达，最终得到了不错的仿真结果[13]。

C Thurau 等研究了如何使用模仿学习的方法，帮助计算机游戏角色的编程[14]。他们提出了一些不同的算法，主要是从人类生成的数据中进行学习，最终证明了机器学习是可以帮助代理思考解决问题的方案以及建立强大的行为库，减少了代理在寻找适当的解决方式时的搜索空间。作者的这些尝试性工作，证明了模拟学习在游

戏人工智能开发中应用的可能。

在一个动态变化的环境中，游戏 NPC 通常需要进行自主的长期决策行为；但是在遇到某种特殊的情景时，又需要做出相应的反应。分层任务网络（Hierarchical Task Networks）可以实现长期的战略规划；而行为树（Behavior Tree）则可以做出相应的反应性行为。这两种方式各有各的优点。因此，Xenija Neufeld 等提出了一种将层次任务网络规划器与低层决策树和行为树相结合的混合规划方法[15]。作者还将此方法与多代理环境中的 pure planner 进行了比较。

Jing Lin 等认为在 MOBA 游戏中，玩家对游戏 AI 的要求更高，传统的有限状态机（FSM）已经完全无法满足开发要求。为了更好的满足玩家的游戏体验，优化游戏 AI，作者提出一种基于 MOBA 游戏特点的分层行为树实现方式，研究了行为树在 MOBA 游戏 AI 设计决策系统中的作用[16]。越来越多的开发者认为有限状态机的时代已经成为过去，不论是为了满足开发人员使用更为友好的技术，还是满足玩家对复杂的高难度 AI 的需求，行为树已然是目前最受欢迎游戏人工智能开发技术。

行为树提供了比以往游戏 AI 设计方式（例如有限状态机、模糊状态机、分层有限状态机等）更为直观的方法。例如：分层有限状态机的实现通常需要复杂的数据结构，这在扩展的时候可能产生不良代码。而行为树的设计方式需要在初期对整个逻辑结构有一个清晰的认识，这非常考验开发者的经验和能力。因此，Rahul Dey 等提出了一种基于 Q 学习的行为树设计方法，Q 学习是强化学习的一种，通过合理的设计奖励函数，游戏 AI 能够在行为树中找到自己认为最优的行为节点[17]。这在一定程度上简化了行为树的设计，减轻了对开发者的逻辑设计要求。但是，作者的这种方法，在简单场景中，对于较少的状态空间才比较有效。对于不熟悉机器学习知识的游戏开发工作者来说，想要以此方法应用到自己的游戏中，尚有难度。

2017 年，Unity 公司开发了一款机器学习的游戏人工智能插件，旨在帮助游戏开发者更方便的训练自己的 NPC。插件中主要使用的还是机器学习中强化学习的训练方式。另外还添加了多种辅助训练方式，例如：课程学习、模拟学习等。在这个开源的插件中，还使用了很多组件化的设计思想，开发者只需要付出较小的精力，根据官方提供的说明文档，就能开始自己的训练[18]。

1.3 研究的内容和工作体现

由于行为树的设计方法拥有众多的优点，例如：丰富的流程控制方法、极强的

扩展性和复用性，受到了游戏开发者的追捧。因此，本论文依然基于行为树的方式，设计出更为拟人化的 NPC。但是，由于行为树缺乏学习机制，其设计出来的游戏人工智能在游戏中的表现依然存在瓶颈，为了提高 NPC 的灵活性，本文将基于机器学习中强化学习技术，通过设计合理的奖励机制，训练出具有自主决策能力的 NPC。具体来说，本文的研究内容主要包含以下 3 个方面：

（1） 基于行为树技术设计一种射击游戏人工智能。为了使 NPC 体现出尽可能多的智能性，本文将从 NPC 的感知能力着手，通过构建 NPC 的视觉节点和听觉节点，在视觉感知和听觉感知上使其更加的拟人化。在 Unity 游戏开发引擎中的 Behavior Designer 插件基础上，扩展更多的智能化节点，进而打造出具有高度智能化的游戏人工智能。

（2） 基于机器学习中的相关技术设计游戏人工智能，主要是强化学习的方式。另外，为了使强化学习的训练更加快速有效，还使用了课程学习和模拟学习等方法。所使用的训练环境是 Unity 游戏开发引擎中人工智能插件 ML-Agents，还探索了不同训练组合方式，对训练的进程和结果的影响。最后，还尝试将机器学习的训练结果与行为树技术相结合，得到了不错的表现结果。

（3） 本文将设计一款具有可玩性的 3D 射击类游戏。一方面是为了论文成果的完整性，另一方面也是为了承载本文研究的决策技术和感知系统，使其能够在具体的游戏运行环境中得到有效的测试。反之，通过分析游戏运行过程中的数据，又可以得到相关技术的可行性与稳定性程度。具体来说，需要完成的游戏模块有：设定游戏故事背景、搭建游戏运行场景、编写基本的可玩性模块、篮球运动员 NPC、战地医生 NPC、巡逻机器人 NPC 和阵地机器人 NPC 等。

1.4 论文结构

本文的内容共分为五章，章节安排如下：

第 1 章介绍本课题的研究背景意义。从游戏人工智能的整体历史角度，分析了其技术发展的历程以及现状。然后较为详细的介绍了游戏决策系统的研究现状，引出本文的研究方向。最后，具体介绍了本文的将会完成的工作任务。

第 2 章主要是介绍本论文中涉及到的相关技术概念。首先介绍目前市场上最为常用的几款游戏开发引擎，分析本论文选用 Unity3D 游戏开发引擎的原因。然后介绍了人工智能与游戏人工智能的区别。再从传统的游戏人工智能开始讲起，引述到

目前较为前沿的机器学习相关技术。为后文打下基础。

第 3 章详细介绍了行为树的工作流程以及行为树插件 Behavior Designer，研究了行为树技术设计 NPC 的方式方法，从 NPC 的感知系统着手，设计以及实现出更为智能化的 NPC。

第 4 章详细介绍了机器学习技术如何运用于游戏之中，强化学习、课程学习以及模拟学习等学习技术在训练 NPC 时候的各种表现；如何使用 ML-Agents 插件进行机器学习训练。通过投篮机器人的设计过程，展现不同训练方法，对训练结果的影响。

第 5 章介绍了本论文设计制作的一款具有可玩性射击游戏。详细介绍该游戏的制作过程，特别是将使用行为树与机器学习设计的 NPC 的应用方法。帮助读者更好的理解游戏人工智能技术对游戏产生的深远影响。

接着统性地总结本文的工作，并对后续研究方向提出展望。

第二章 相关技术概述

2.1 游戏开发引擎

游戏开发引擎实际上是一种软件开发环境，其本身就是一种软件，只不过游戏开发引擎开发出来的软件是游戏而已。开发人员使用游戏引擎可以为游戏机、移动设备和 PC 构建游戏。通常游戏引擎提供的核心功能包括用于 2D 或者 3D 的图形渲染器，物理引擎或碰撞检测（碰撞响应），声音，脚本，动画，人工智能，网络，流媒体，内存管理，线程，本地化支持，场景图，并可能包括对过场动画的视频支持。

目前，市面上游戏开发制作引擎有许多种，通常都是功能非常全面的商业引擎，各个大型游戏公司几乎都会开发自己的游戏开发引擎。在早期，也有单独为了制作一款特定游戏而设计的引擎。在游戏制作行业中，目前非常流行的游戏开发引擎主要有 Cocos2d、Unreal Engine、Unity 3D 等[19]。

2.1.1 Cocos2d

Cocos2d 在众多受欢迎的游戏开发引擎中，最大的一个特点是其开源性。它是基于开源框架 MIT 协议，旨在帮助游戏开发者构建高质量游戏和其他图形交互应用。最初的 Cocos2d 是使用 Python 编写，不过目前，已经移植到了多种语言平台，例如：Objective-C、C++、Java、C#等。因此，现在我们说起 Cocos2d 已经是一个大家族，其分支包括：Cocos2d for iPhone、Cocos2d-X、Cocos2d-android、Cocos2d-html5 和 Cocos2d-XNA。

Cocos2d 其他的优点还有轻量级、专注 2d 游戏制作等。但是最大的特点还是开源，这给开发人员很大的自我修改空间，实现更加符合自己需求的开发环境。并且，使用过程完全免费，这就吸引了一大批中小游戏企业和独立游戏开发者[20]。

但是 Cocos2d 也存在很多问题，由于是开源的游戏开发框架，品控比较差，可能存在不可预知的错误。有些模块仍然存在性能问题，明显不如一些商业游戏引擎。另外，在 3D 游戏的制作上，能力也有些捉襟见肘。

2.1.2 Unreal Engine

虚幻引擎（Unreal Engine）是有 Epic Game 公司开发的一款游戏引擎。包含游戏开发过程中需要的渲染、碰撞检测、UI、AI、网络、内存、线程以及文件系统等模块。很多开发者评价它是一款 AAA 级次时代引擎[21]。不得不说，目前使用虚幻引擎开发出的游戏，画面的精美度最高。自 1998 年问世以来，经过不断的迭代更新，虚幻引擎逐渐成为游戏开发行业的佼佼者，占据着全球商业引擎市场中 80%的份额。

虚幻引擎中的可视化脚本系统蓝图，可以让你不写一句代码，就能做出游戏，这也是其他很多游戏引擎所追求的终极目标。有人说蓝图不如脚本代码灵活，但是这就像是 Linux 与 Windows 之争，最后用户的选择表明了一切。同时，虚幻引擎中使用的是 c++语言，这就在语言层面上，体现了最终游戏所能拥有的性能。另外，虚幻引擎的代码是开源的，当游戏开发者希望对引擎进行个性化优化时，这种优势就体现出来了，并且，对于大型游戏公司来说，如果遇到游戏引擎的瓶颈或者不可预知的错误，就一定存在阅读游戏引擎底层代码的需求。

但是，如此优秀的游戏开发引擎也有短板。首先，虚幻引擎的入门门槛比较高，虽然使用 C++作为开发语言能得到很好的性能，但是 C++语言难度大，不利于上手。其次，在 2d 方面的表现一般，图形操作界面也不如 Unity 等一些引擎友好，甚至可能崩溃。再者，虚幻游戏引擎是不免费的（除了它的 UDK 版本），除了需要每月付使用费外，如果开发者所开发的游戏收入超过一定的数额，还要另外付费。这对独立独立游戏开发者和中小型公司不太友好[22]。

2.1.3 Unity 3D

Unity3D 是由 Unity Technologies 公司开发一款跨平台游戏引擎。通过该引擎开发的游戏，可以轻松的发布到众多平台，例如：Windows、MacOS、Linux、Android、iOS、Switch 和 Xbox 等。后来，还延伸到了基于 WebGL 技术的 HTML5 网页平台。Unity3D 还不满足于游戏行业的发展，在建筑业可视化方面和三维动画制作方面拥有广泛的应用[23]。Unity 诞生于 2005 年，在苹果公司的全球开发者大会中宣布对外开放使用，作为后起之秀，Unity3D 引擎经过数年的技术迭代积累，积极追赶虚幻引擎等 3A 级游戏开发引擎，已经发展成占据全功能游戏引擎市场的 45%份额，居全球首位，使用其进行游戏开发的人员占比也非常之高。

Unity3D 引擎希望让用户能够轻松的打造自己的游戏，这种设计理念为他吸引了众多的追随者，实际上，Unity3D 引擎确实非常容易上手，使用了更为简便易用的C#的开发语言，组件化的设计方式，可以让用户不用写一行代码，就能开发游戏[24]。更重要的，这款引擎在性能、游戏质量和入门门槛三者中取得了平衡，公司提供给了用户完全免费的版本，与专业版几乎相差无几，只有在用户创造的游戏获得一定的收入后，才会收取少许的版权费。因此，不难理解 Unity3D 引擎会是游戏爱好者及独立游戏制作者最为青睐的游戏开发工具。本文基于以上考虑，最终选择使用Unity3D 引擎作为研究平台。

2.2 人工智能与游戏人工智能

游戏人工智能与传统意义上的人工智能相比，既有相通的地方，也有一些差异性。因此必须在下面的内容中对这二者做出解释。

2.2.1 人工智能

人工智能（Artificial Intelligence）简称 AI，是指人造出来的机器变现出来的智能。其下属研究方向包括有机器学习、自然语言处理、图像识别、语言语音识别、机器人等。涉及到了认知科学、数学、统计学、物理学、逻辑学和社会学等众多学科领域[25]。总的来说，人工智能的目标是研制出具有人类一样智能性的机器或者软件。因此，可以预测到的是，人类的很多岗位将会有被人工智能替换的危险。

人类大脑的复杂程度非同一般，想要获得人脑一样的智能机器，其难度无疑是非常大的。而人工智能就是为了突破这个难题，它希望能够获得人类的感知能力、交流能力、学习能力、操控能力还有推理能力。虽然目前的研究还没有发展到很高的高度，但是在某些方面已经初显成果，例如图像识别领域、自动驾驶领域、棋牌游戏和语音识别领域等。每一次突破都带来了该领域极大的变革，这也展现了人工智能的光明前景。

2.2.2 游戏人工智能

从上文中可以知道，人工智能是为了获得人类一样的认知能力和决策能力，而游戏人工智能则不同，它只是希望在表现上与人类相似，至于内在的实现方式，即

使是作弊也无所谓[26]。大多数时候，游戏人工智能的目标是服务玩家，因此，创建出玩家无法打败的 NPC 也是没有意义的。必要的时候，还应该降低游戏人工智能的难度，给予玩家适当的挑战性，或者为其设置“弱点”。

游戏人工智能通常包括感知系统、决策系统和行为系统。感知系统中又包括视觉感知、听觉感知和记忆能力。决策系统则是自我决策能力。行为系统中主要是包含各种各样的拟人化动作行为，例如寻路系统。以上列出的所有游戏人工智能系统功能，在传统人工智能中也有包含，但是在两者中实现的方式却可能有着天壤之别。一方面，传统人工智能中每一种功能实现的背后，都依赖于强大的算力，而游戏人工智能作为整个游戏中的一环，不能消耗过多的算力资源[27]。另一方面，游戏与现实有着根本性的区别，无法使用类似传感器一样的感知器件，更不用对最终结果负责，游戏人工智能大多数时候只是追求表现上的智能。

2.2.2.1 游戏人工智能的寻路方式

寻路系统在游戏中非常实用，主要作用是寻找两个指定点之间的可移动路径，最基本的条件是要避开障碍物。这个路径不一定是最短路径，但一定是最优路径。这是由所处的场景和开发者所决定的。当 NPC 需要在最短时间内从一个点运动到另一点，就需要找到这两点之间的最短路径。当 NPC 在行走的过程中，还要考虑躲避敌人，则需要找到可以躲避敌人的最优路径。

寻路系统通常在游戏中耗费大量的计算资源，原因是每次进行路径规划时，都需要遍历大量的节点。很多学者在这个领域做了大量的工作，创造出来各种各样的寻路算法。目前应用最为广泛的寻路方式是 A*寻路算法。这是一种启发式的寻路算法，在很多情况下，可以减少很大一部分需要遍历判断的节点。后来，还出现了 A*寻路算法的优化算法[28]，目的是希望增加寻路算法的效率[29]。为了满足特殊的寻路方式，开发者不得不在地图中设置大量的节点[30]，以帮助 NPC 找到最优的路径，虽然 NPC 表现出来更好的智能性，但是也消耗了大量的内存[31]。

2.2.2.2 游戏人工智能的视觉

现实中人工智能的视觉实现方式，一定是使用摄像头拍摄画面，然后使用图像识别处理等技术。那么游戏中是否也需要用一个相机拍摄出一个画面，然后使用一

系列机器学习技术，最后实现 NPC 的视觉功能呢？如果现实中可以实现，那么游戏中也完全可以模拟出来，只是没有必要这么做。既然游戏人工智能是表现上的智能，那不妨使用上帝视角来作弊。事实上游戏开发者也是这么做的[32]。

游戏人工智能视觉的实现，会根据游戏中的具体需求变化。

当 NPC 所需要观察的对象很少时，例如在单机的射击游戏中玩家只有一个的情况，就只需要检测玩家与 NPC 之间的距离，如果距离在视线范围内，就接着检测玩家是否在视角范围内，然后再用射线检测，判断玩家是否被遮挡。同时满足这三个条件，那么就可以判断 NPC 看见了玩家。

当 NPC 需要观察的对象非常多时，例如具有战争迷雾的游戏中，我们需要判断玩家周围的众多物体是否可以被看见。第一种方法是使用射线检测的方式，在 NPC 周围是释放多条射线，射线第一次碰撞到的物体，就是玩家的可见物体。但是很明显，这需要发射非常多的射线，性能比较差。第二种方法是将 NPC 视线范围内的物体建立数据结构，然后遍历这些物体，使用射线检测每一个物体是否可见。这种方法的性能比较高，但是实现起来较为麻烦。

2.2.2.3 游戏人工智能的听觉

在现实世界中，声音的传播、衰减是自然发生的。而游戏中的声音传播需要模拟。其实真实世界中声音的传播可以近似看作是无遮挡的，因此可以直接使用距离作为依据，进行线性计算，最后得到一些描述声音大小的数据，然后将听到的声音交给决策系统判断执行某种行为即可。在具有高智能性的游戏人工智能中，NPC 应该根据听到的声音大小，做不不同的反应，甚至是分辨不同物体的声音。有时，在一些潜行类游戏中，故事发生的场景范围较小，隔音效果好，我们还可以将声音的衰减描述为依据寻路算法的路径来计算，而不是两点间的距离。

2.2.2.4 游戏人工智能的决策方式

游戏人工智能最为重要的系统就是决策系统。它负责处理 NPC 的其他感知系统、行为系统等收集的信息，做出适当合理的下一步行为决策。至于根据什么条件触发什么动作，就是游戏策划决定的事情，开发人员根据需求，设计并编码出合理的程序，因此这是一种根据固定信息，产生固定决策的方式，不会出现不可预知的

行为表现，缺乏灵活性，这种缺乏灵活性的 NPC，也很难提高玩家的兴趣。游戏人工智能的决策系统经历了以下几个阶段：有限状态机、分层有限状态机、模糊状态机、行为树等[33]，现在又发展出了更为先进的基于机器学习的方法。这种方法设计得到的游戏人工智能，很好的解决了传统 NPC 缺乏学习机制和环境适应能力，可以基于玩家更多沉浸式的感官体验。

2.3 常见游戏人工智能设计方法

2.3.1 有限状态机

有限状态机（Finite state machine），简称 FSM。它最初其实是一种数学上的计算模型，本质上是一种抽象的设计方法，用于管理某个系统中，不同状态之间的切换。系统中状态机所控制的对象在同一时刻只能有一种状态，满足某个状态切换条件后，对象就由一种状态转换为另一种状态[34]。

后来，这种设计方法逐渐应用于计算机程序设计领域，特别是在游戏开发中。使用有限状态机控制的 NPC，背后都有一套复杂的状态切换条件，它必须确保 NPC 面临的所有情形，都有相应的状态与之对应，继而控制 NPC 做出相应的反应行为。但是，实际上一个游戏中，NPC 将会面临的情形非常之多，玩家在 NPC 面前做出的行为动作也非常多，开发者没办法为每一种情形设计相应的状态，而且每增加一种状态，其制作难度都会指数级上升。

有限状态机缺乏并行机制，并且同一时刻只能位于一个状态，玩家可能因此而找到 NPC 的行为弱点，轻易的将其击败。这也是这种方式实现 NPC 缺乏灵活性的表现。

2.3.2 行为树

行为树在图形上表示为一颗多叉树。树的根节点就是决策行为的出发点。树中每个节点表示一种逻辑控制单元，这种控制单元可以是节点组织方式、条件判断、行为动作等[35]。这些节点一般都会设计为较为低级功能的小模块，开发者根据行为树的运行规则，方便的扩展满足自身需求的节点。行为树的执行方式整体上是从上至下，从左至右。另外还可能有一些特殊的组织节点，可以并行执行子节点。有些开发者也会实现一些中断机制，满足复杂行为树设计的需要[36]。

行为树主要有根节点、组合节点、修饰节点、条件节点和行为节点几种类型。根节点是一颗行为树开始运行的起点，没有太多功能性的作用，主要是起着组织的作用。组合节点是一类控制其子节点运行方式的节点，一棵行为树的运行逻辑实现就依赖于这种节点。修饰节点则是一种依赖于子节点存在的节点，负责影响子节点的执行返回结果。条件节点则是一类判断状态是否发生改变或者满足某些条件，通常与组合节点配合使用，影响与之并行的节点运行方式，在行为树中的地位非常重要。行为节点则是行为树中的叶子节点，它是行为树产生决策后的决策落实点。NPC多种多样的行为，都可以设计为一种行为节点[37]。

行为树的树状组织结构，在设计逻辑上比有限状态机清晰很多。以树的遍历的方式执行每个节点，使用条件节点判断状态的改变，最后行为节点执行决策的结果。可以看出，行为树不需要像状态机一样，仔细的控制每种状态之间的转换关系，而是在设计初期，就将不同状态的逻辑关系和优先级关系组织好。行为树中每种节点都可以被自由的添加改写，因此，其扩展性非常好。使用具有低级功能的节点，最终打造出具有高度智能性的 NPC，说明其模块化非常好。如果多个 NPC 具有相同的决策行为方式，可以将行为树导出，复用在其他 NPC 身上。或者将行为树中特有的对象作为变量抽象出来，甚至可以将行为树应用于不同类型的 NPC 身上。这就说明它的可复用性强。通过实现随机化节点、权重概率节点，可以在一定程度上增加行为树的灵活性[38]。

行为树作为当前较为成熟稳定的游戏人工智能设计方式，依然存在一些不可忽视的缺点，例如，根据策划的需求，设计出健壮的行为树，是一件非常具有挑战性的任务。其次，行为树中的每种节点都需要开发者自行编码，想要实现复杂智能性的游戏人工智能，其工作量不可小觑。最后，行为树缺乏学习机制，NPC 的所有行为表现，都是事先确定在编码中，在灵活性上依然依然有所欠缺。

2.3.3 有限状态机与行为树的比较

通过上述介绍，有人可能会认为，行为树技术可以完全取代状态机。事实上有些开发者已经表达了类似的观点，认为状态机的时代可以加过去了。诚然，行为树确实有很多状态机所没有的优点，例如：灵活、强大、易于扩展。

从灵活性上来看，让有限状态机同时处于两个状态的的唯一办法，就是另外再创建一个状态机。而在行为树中可以轻松做到，通过设计编写一种并行节点即可。

行为树的强大体现在，使用行为树创建的游戏人工智能比使用状态机创建的要清晰明快很多，行为树在整体上表现为一棵树，而状态机则是一团乱麻。因此状态机需要设计很多状态转换，这种工作量是随着状态的增加而呈指数级增长。不过，不得不承认，在某些较为简单的环境中，有限状态机设计和实现起来非常方便。

行为树最大的优点是易于维护和扩展[39]。想象一下，如果要更改状态机中某两个状态之间的切换逻辑，首先，需要从一堆转换关系中寻找，然后思考这两者之间逻辑的变化是否会影响其他状态，整个过程一定会让开发者非常抓狂，如果使用行为树，则可以清晰明了的发现节点间的逻辑和优先级关系，在可视化编辑器中，只需要移动节点的位置，而无需考虑切换关系。

其实，即使行为树在各个方面优于状态机，但是状态机依然有存在的理由。我们可以巧妙的将二者结合，让状态机成为行为树中描述功能的节点，在事先较为简单的决策环境中，状态机还是非常方便易用的。

2.4 机器学习

针对行为树决策缺乏学习机制，无法适应游戏环境变化的问题，本文引入机器学习的概念，主要应用机器学习中的强化学习、课程学习和模拟学习等技术，让 NPC 的设计从根本上发生改变。本节将简要介绍上述技术概念，为后文的研究做好铺垫。

2.4.1 机器学习简介

机器学习是一门复杂的多领域交叉学科，涵盖了算法、凸分析、统计学、概率论和逼近论等多门学科，难度极高，是当前科学技术研究的前沿阵地[40]。机器学习的目标是让机器能够在某些方面模仿人类一样的行为方式，根据给定的数据集或者其他学习方法，最终形成具有独立行为方式的机器。

机器学习暂时还没有确切统一的定义，但是学界中比较推崇的一个种说法是：“对于某类任务 T 和性能度量 P，如果一个计算机程序在 T 上以 P 衡量的性能随着经验 E 而自我完善，那么我们称这个计算机程序在从经验 E 中学习[41]”。这是由卡内基梅隆大学的 Tom Mitchell 教授在他的 Machine Learning 一书中提出来的。

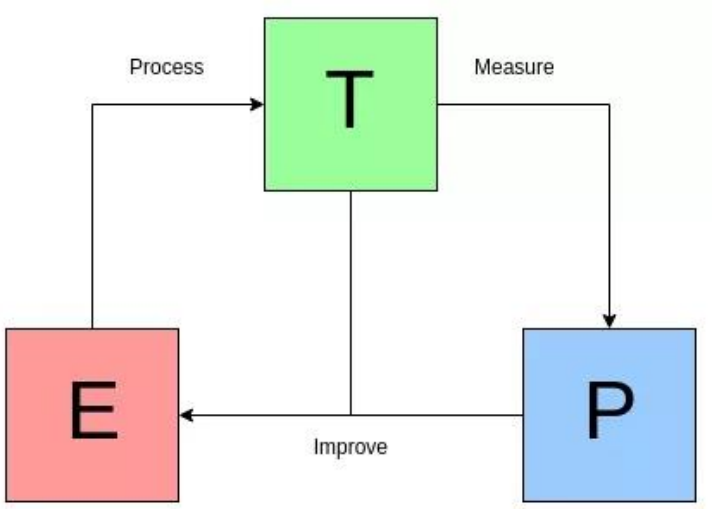

图 2-1 机器学习定义描述图

Figure 2-1 Machine Learning Definition Description Diagram

机器学习大体上可以分为有监督学习、无监督学习和强化学习。有监督学习要求训练数据有标签，负责从该数据集中得到一个函数，然后使用一些新的无标签数据，通过该函数，就能得到合理的输出。通常应用于分类，是决策树和神经网络训练中的常见技术。无监督学习使用的数据集则没有标签，目标是让其在样本中检测相似性，进而进行分类。通俗的说，就是让机器自己学习决策，而不是人类教它。通常应用于聚类。而强化学习同样是需要自己学习怎么做，让机器不断的“试错”，在试错的过程中，通过奖惩方式来告诉机器应该学习的方向。

机器学习中常见的算法有：决策树算法、朴素贝叶斯算法、人工神经网络算法等。而在具体的强化学习中，常用的算法有： Q-learning、(Proximal Policy Optimization)PPO 等

2.4.2 强化学习

强化学习（Reinforcement learning）又称增强学习、再励学习或者评价学习，是机器学习领域中较为重要的一环。它在环境中不断的“试错”，根据决策后的奖惩结果，优化决策模型，以求获取最大的利益。强化学习在众多领域中都有应用，例如博弈论、遗传算法以及控制论等。强化学习模型通常由状态（state）、动作（action）、奖励（reward）和策略（policy）四个部分组成[42]。

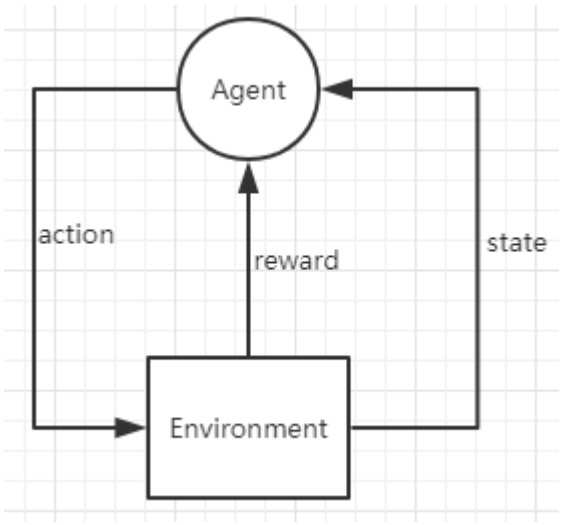

图 2-2 强化学习模型图

Figure 2-2 Reinforcement Learning Model Diagram

状态（state）是代理观察环境而得到的某个特定状态，通过状态价值函数，可以得到，在时间 t 时刻的状态 s 能够在未来获得回报的期望。

动作（action）是代理通过观察环境，进而通过某种策略产生的输出，这种输出将会重新影响环境，产生奖赏。

策略（policy）是强化学习的核心。策略是通过对观察环境得到的状态进行处理，最终得到输出动作。不同策略，在决策时选择的动作不同，这将影响到整个学习的效果。Agent 在环境中选择动作所采用的方法称为策略π，在所有可选策略中能获取最大奖赏值的策略π^*称为最优策略[43]。

奖励（reward）是强化学习的特点所在，Agent 在观察环境以及影响环境的过程中，获得了奖惩信号，策略根据这种奖惩信号，又反过来影响下一次的行为决策。

强化学习可以抽象为马尔科夫决策过程(Markov Decision Process,MDP)。马尔科夫五元组$\mathrm{M}=(\mathrm{S},\mathrm{A},\mathrm{P},\gamma,\mathrm{R})$，其中 S 表示状态集；A 表示一组动作；P 为表示状态转移概率；R 表示回报函数；γ表示折现因子。在马尔科夫决策模型中，状态转移概率及奖赏值与已学习过的策略无关，只取决于当前探索策略[43]。强化学习算法在学习过程中没有确定的状态转移概率和奖赏函数，通过试探的方式学习系统最优策略。

假设状态$\mathrm{s}_t \in S$是 Agent 在 t 时刻从环境中获得的状态，然后按照策略π选择动作$\alpha_t \in A$，然后在环境中执行，进而在 $\mathrm{t}+1$ 时刻转移到状态$s_{t+1} \in S$，并获得环境反馈的奖惩信号$r_{t+1} \in S$，根据奖赏信号r_{t+1}，Agent 改进策略 π 以适应环境。

由于强化学习是以获得最优策略π^*：$\mathrm{S} \rightarrow \mathrm{A}$为学习目标，使 Agent 所能取得的奖

励最大化。其最优状态值函数定义如下[44]：

$$V^*(s) = \max_{\alpha \in A(s)} \sum_{s'} P_{ss'}^{\alpha} \left(R_{ss'}^{\alpha} + \gamma V^*(s') \right) \quad \forall s \in S \tag{2.1}$$

其中，s'是状态s的状态转移；$P_{ss'}^{\alpha}$是 Agent 在状态s下执行动作α转移到状态s'的概率；γ是折扣因子；$R_{ss'}^{\alpha}$是 Agent 在状态s下执行动作α转移到状态s'的奖励。

而最优策略π^*为：

$$\pi^*(s) = arg \max_{\alpha \in A(s)} \sum_{s'} P_{ss'}^{\alpha} \left(R_{ss'}^{\alpha} + \gamma V^*(s') \right) \quad \forall s \in S \tag{2.2}$$

动作价值函数则定义为：

$$Q^*(s, \alpha) = \sum_{s'} P_{ss'}^{\alpha} \left(R_{ss'}^{\alpha} + \gamma \max_{\alpha \in A(s)} Q^*(s', \alpha') \right) \quad \forall s \in S, \quad \forall \alpha \in A(s) \tag{2.3}$$

最终的最优策略π^*就可以写为：

$$\pi^*(s) = arg \sum_{s'} P_{ss'}^{\alpha} \left(R_{ss'}^{\alpha} + \gamma \max_{\alpha \in A(s)} Q^*(s', \alpha') \right) \quad \forall s \in S, \quad \forall \alpha \in A(s) \tag{2.4}$$

2.4.3 课程学习

课程学习是一种训练机器学习模型的方式。在使用强化学习进行训练的过程中，往往会有训练难度过高的情况，训练很长时间，依然得不到很好的收敛效果。课程学习就是应用于这种复杂环境，它采用循序渐进的诱导方式，先降低 Agent 训练的难度，使其较为容易的得到一些奖励[45]。例如，对于数学这么晦涩难学的学科，我们都是从简单的加减法开始学习，逐渐过渡到乘除法，然后再过渡到微积分和线性代数等。但是，课程学习不是随便给 Agent 一些奖励，早先的课程一定是后续课程的基础，对后续训练有直接性的帮助。例如在训练一个足球射门机器人时，我们可以在前期的课程中故意加大球门的大小，让 Agent 能够获得较多的奖励，知道自己学习的方向。

更深入的理解，就是在进行强化学习时，Agent 只是在进行随机的决策行为，当训练环境非常复杂时，Agent 可能要随机很长时间才能获得一些微小的奖励。为了引导 Agent 减少不必要的试错，更快进行更多有意义的学习，就需要课程学习来动态的改变学习环境[46]。

2.4.4 模仿学习

在奖励特别稀疏的情况下，甚至课程学习也不太好设计的情形，模仿学习就变得异常有用。强化学习其实原本是一种自己学习决策行为的训练方法，人类不会告诉 Agent 应该怎么做，只是在 Agent 作对事情时给予一定的奖励。加入模仿学习后，就能在一定程度上让指导 Agent 应该怎么做，这无疑可以加快整个模型的训练速度，更快的得到收敛的结果。

这里主要介绍两种模仿学习方法。GAIL(Generative Adversarial Imitation Learning)和 BC(Behavior Cloning)，这两种模仿学习的方法将在后文中有相应的应用。

生成对抗模仿学习（GAIL）的做法是假设专家（expert）是属于某一分布（distribution），然后让 Agent 策略模型也预测一个分布，这么做的目标是让这两个分布尽可能的接近[47]。

假设输入的专家轨迹为$\{\hat{\tau}_1, \hat{\tau}_2, \cdots, \hat{\tau}_N\}$，初始化鉴别器D（discriminator）和 Agent 策略函数π，然后在每次学习中，都获取 Agent 的轨迹$\{\tau_1, \tau_2, \cdots, \tau_N\}$。

更新鉴别器：

$$\mathrm{D}(\tau) = \frac{1}{T}\sum_{t=1}^{T} d(S_t, \alpha_t) \tag{2.5}$$

其中，$d(S_t, \alpha_t)$为专家的最大奖励函数。

然后更新 Agent 参数，

$$\theta^{\pi} \leftarrow \theta^{\pi} + \vartheta \sum_{i=1}^{N} D(\tau_i) \nabla_{\theta} \pi \log P(\tau_i|\pi) \tag{2.6}$$

行为克隆（Behavior Cloning）其实是一种有监督的学习，通过对大量给定的数据集进行学习，最终得到一个状态s到动作a的映射[48]，可以看出，这种方式可以加快 Agent 的训练过程。但是我们能够给予的学习数据集是有限的，没办法涵盖方方面面的情况，更何况有些情况下的数据是非常难以得到的。将行为克隆与强化学习相结合，行为克隆引导 Agent 在初期产生正确的决策，得到大量的奖励，强化学习可以在所有的状态下进行试错，各取所长，最终得到满意的训练结果。

2.5 本章小结

本章主要介绍了几种市场上常见的游戏开发引擎，通过对不同引擎特点的对比，

得出我们选择 Unity 3D 作为开发平台的目的。接着阐述了游戏人工智能与传统人工智能的区别。详细介绍了游戏人工智能所涵盖的范围。然后还具体介绍了游戏人工智能的几种常见设计方法。最后，还介绍了机器学习的相关概念，特别详细介绍了强化学习相关的技术和算法，为后文做好铺垫。

第三章 行为树 NPC 关键技术的设计与实现

3.1 行为树和 Behavior Designer

行为树在图形层面上表示为一棵有向树，节点主要可以被分为：根节点、组合节点、修饰节点、条件节点和行为节点。对于每对连接的节点，传出节点称为父节点，传入节点称为子节点。根没有父母，只有一个孩子，组合节点有一个父母，至少有一个孩子，行为节点有一个父母，没有孩子。以图的表现方式，组合节点的子级按从左到右的顺序放置在其下方，按照从左至右的方向执行。行为树的执行从根开始，根以一定的频率刷新执行其子节点。所以我们可以根据自己的项目需求，控制行为树的执行频率。

3.1.1 Behavior Designer

前面我们介绍了很多行为树的理论知识，如果要将行为树技术应用到本文的游戏中，还需要借助 Unity 中实现了行为树技术的插件 Behavior Designer。

Behavior Designer 是由 Opsive 公司开发，旨在让使用 Unity 游戏开发引擎的工程师方便快捷的设计自己的行为树 NPC。Behavior Designer 中提供了许多的 API 接口，如果项目需要，我们完全可以自主实现我们需要的任何行为节点。最重要的是，它提供了一个具有可视化界面的编辑器，如果现有的功能节点能满足项目需求的话，完全可以不用编写任何代码。同时，它还提供了多种第三方插件的集成方法，为创建复杂的 NPC 提供了基础。

Behavior Designer 中的任何一个节点都可以称之为一个任务。每个任务中都存在类似于 Unity Monobehavior 中的生命周期函数。通过这个 API 接口，我们可以非常方便的设计出独特的任务。任务中 API 函数执行流程图如下：

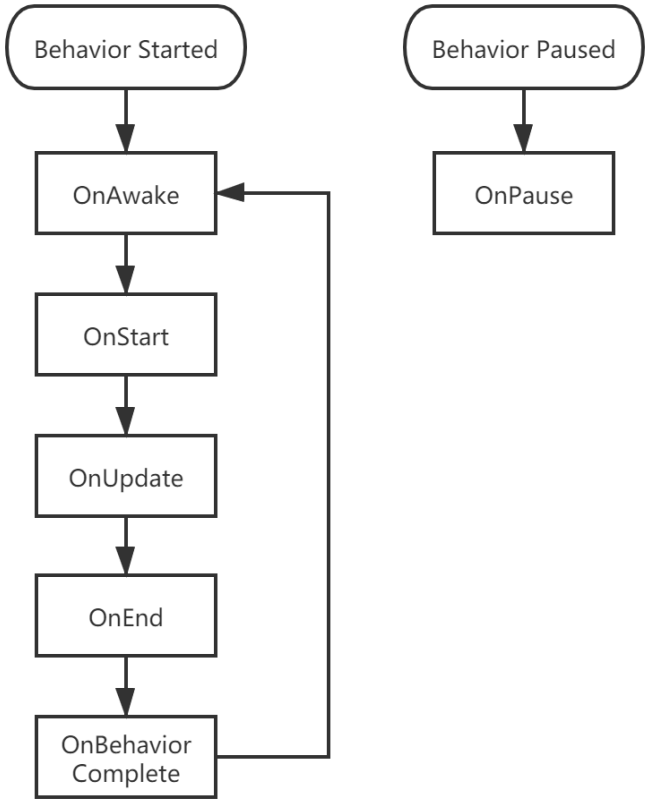

图 3-1 Behavior Designer 生命周期函数流程图

Figure 3-1 Flow Chart of Life Cycle Function of Behavior Designer

Behavior designer 主要提供了 5 大类节点：根节点、复合节点、条件节点、行为节点、修饰节点[49]。

根节点（Root）：行为树决策发起的起点。没有太多真正意义上的功能，当树中所有节点都无法满足执行条件时，就会返回到根节点中。当行为树开始执行后，根节点只会向下执行一次，除非重启行为树，才会重新从根节点运行行为树。

复合节点（Composites）：是一种将多个子节点聚集起来的节点。复合节点控制着子节点的运行顺序和运行逻辑。常见的复合类型节点有：顺序节点、选择节点和并行节点。顺序节点（Sequence）是一种类似于逻辑“与”的节点，只要它的子节点返回一个失败，后续的子节点就不会执行。如果子节点返回成功，则依次从左至右执行下一个节点。选择节点（Selector）是一种类似于逻辑“或”的节点，当其子节点只要有一个返回成功时，其后续的子节点就不会被执行。只有当某个子节点之前的所有节点都返回失败时，该节点才有执行的机会。并行节点（Parallel）是一种可以同时执行所有子节点的复合节点，它的子节点中只要有一个返回失败，则停掉所有子节点的执行，并返回失败。只有所有子节点全部返回成功或者正在执行时，它才会

返回成功。

条件节点（Condition）：是用于判断当前状态是否满足某种条件，如果满足，那么节点就会返回成功；如果不满足，节点就会返回失败。条件节点通常会与复合节点组合使用，用于控制复合节点的子节点应该在什么情况下执行。而且，它不接受任何类型的子节点，只能是行为树的叶子节点。

行为节点（Action）：是真正控制游戏 AI 行为的节点，例如：攻击、巡逻和逃跑等。在行为节点中，行为树会持续执行该节点，返回正在执行状态。直到游戏 AI 遇到了某种不同的情形，被上层条件节点捕获为止。

修饰节点（Decorator）：用于包装子节点的节点。可以在运行时控制子节点返回状态，因此，它必须依附于除 Root 节点外的其它类型子节点，且只能有一个子节点。修饰节点循环执行其子节点，控制循环终止的方式有：设置最大执行次数、设置计时器、设置过滤条件等。

3.2 行为树 NPC 的视觉感知设计

人类视觉感知方式，无非具有两个限制因素，分别是视距和视角。视距指的是视力所能看见的最远距离；视角则是指视力所能覆盖的范围角度。实际上人眼的视线距离非常远，在游戏中为了降低游戏难度，不宜将 NPC 的视距设计过大。游戏中还要考虑遮挡问题，即使在视距和视角范围内，还需要检查目标离自己之间是否有遮挡物，这个遮挡物是否会影响视线，例如，一堵墙和半透明物体的区别。

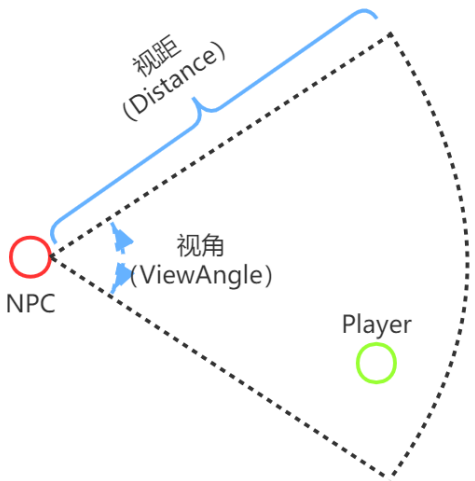

图 3-2 NPC 视觉感知示意图

Figure 3-2 Visual Perception Diagram of NPC

分析了视觉的基本需求，还需要从性能的角度考虑实现方法。不同游戏中，对

NPC 的视觉感知需求是不同的。例如，在单人游戏中，NPC 的敌人（玩家）通常只有少数几个而已；而在大型多人游戏中，则可能有很多个敌人（玩家）。所以，本文主要从单人和多人游戏的分类角度，对 NPC 视觉感知方式进行设计。

在单人游戏中，本人的设计方法是给定 NPC 一个观察目标，不断检测该目标是否进入视距范围内，这通过计算空间中两点间距离即可；如果满足视距条件，就紧接着判断目标是否在视角范围内，这通过 NPC 的正向的方向向量与 NPC 到目标的向量的角度即可；最后，如果视距和视角条件都满足，就判断 NPC 与目标之间是否存在遮挡，通过 NPC 向目标发射一条或多条射线进行检测即可。

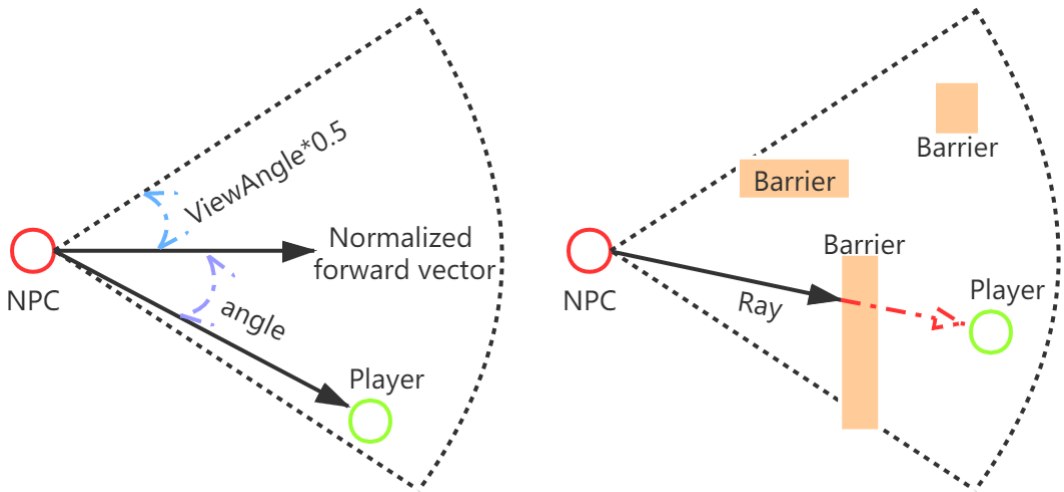

图 3-3 视觉感知方式

Figure 3-3 Visual Perception Method

在大型多人游戏中，如果m个 NPC 要对n个玩家进行视觉感知的话，其计算复杂度就达到了O(mn)。这是非常耗费性能的处理方式。因此，本文考虑使用标签（Tags）或者层级（Layers）判断的方式，将 NPC 可见和不可见的物体进行分类，如果使用标签分类，就需要在全局范围内查找指定标签的物体，将其缓存到某个容器内，如果使用层级分类的方式，就以 NPC 为中心进行球形碰撞检测，将视距范围内的可见物体放入某个容器中缓存。其中缓存的大小依据具体需要进行优化即可。然后以距离远近为依据，对所有的可见物体排序，从最近的可见物体开始判断，是否满足视角范围要求，如果满足，则认为 NPC 看见该物体；否则继续遍历下一个物体。

基于 Unity 3D 游戏开发引擎，上述技术方案实现如图 3-4：

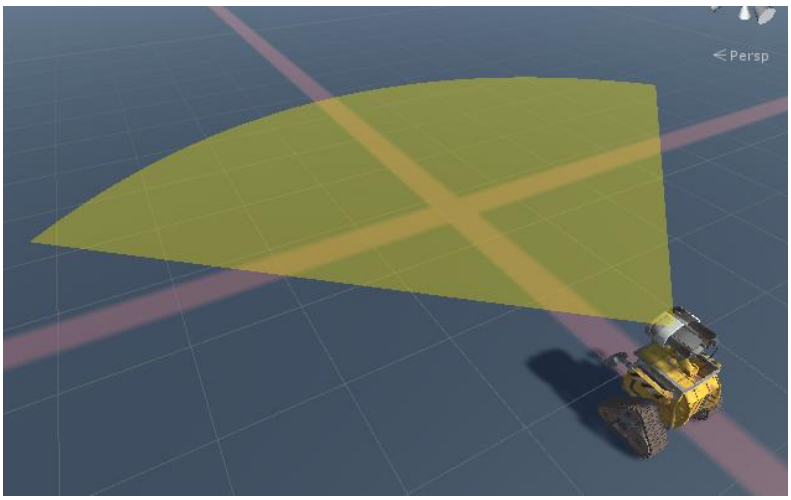

图 3-4 视觉感知实现效果

Figure 3-4 Implementation Effect Visual Perception

上述实现的视觉感知节点的 Inspector 属性面板中，提供了很多可以设置的属性。图 3-5 中，TargetObject、TargetObjects、TargetTag 和 TargetLayerMask 都是用于设置 NPC 观察的目标对象，从上至下优先级依次减小，也就是说上层属性设置了目标，下层属性设置就会失效。IgnoreLayerMask 用于设置遮挡检测时应该忽略的层级，当有半透明或者透明物体遮挡时，这个属性会非常有用。ViewAngle 用于设置 NPC 的视角范围。ViewDistance 用于设置 NPC 的视线距离。SelfOffset 用于控制 NPC 视线的起点相对于中心点的偏移。TargetOffset 则用于视线在目标身上落点的偏移。最后，SeeObject 则是 NPC 看见的目标对象。

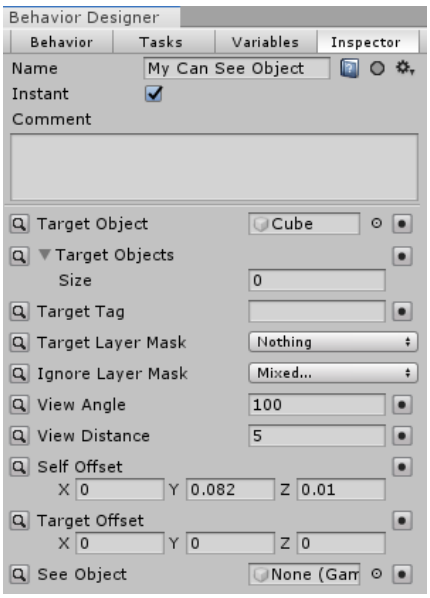

图 3-5 行为树视觉感知属性面板

Figure 3-5 Visual Perception Inspector of Behavior Designer

3.3 行为树 NPC 的听觉感知设计

人类的听觉感知非常灵敏，能够辨别一定范围内的声音，并且能够分辨声音的类型，以及声音的大致方向。除了需要考虑人类听觉的因素外，还需要考虑声音本身的强度和传播过程中的衰减。在游戏世界中，我们几乎无法做到监听游戏世界中的任何声音，这没有必要，也不允许，因此，本文将指定 NPC 监听有限个物体的声音。其次，听觉感知的范围与视觉感知不同，人类能够听到来自四面八方的声音，所以本文将 NPC 的听觉感知范围近似看作一个以自身为圆心的球体。而发生源则以声音传播的距离作为声音强度的模拟，并且，依据距离做衰减处理。

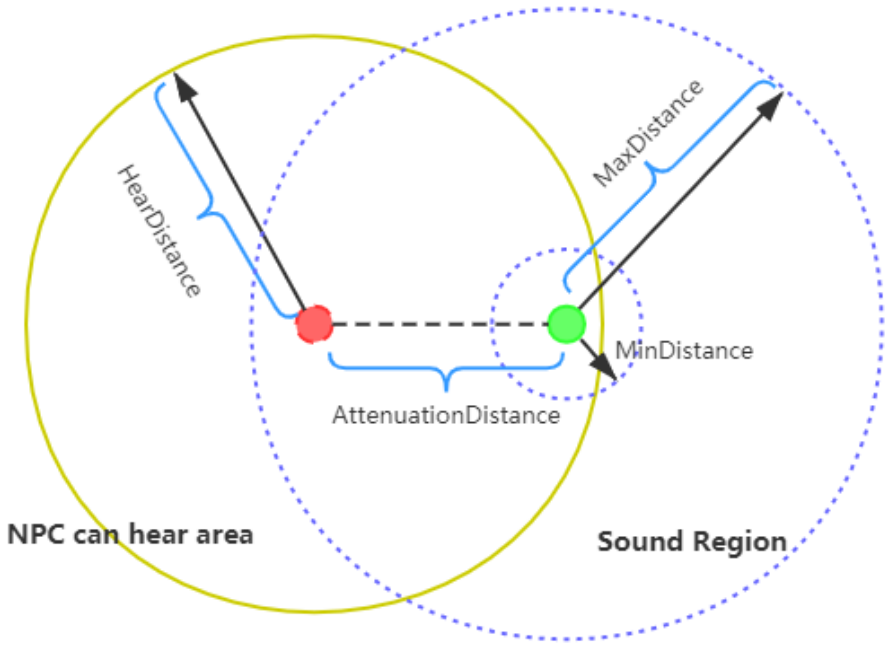

图 3-6 NPC 听觉感知示意图

Figure 3-6 Auditory Perception Diagram of NPC

视觉感知中，为了提高感知系统的性能，将同一时间内看到的多个物体，以距离为依据进行了排序，以减少射线检测的次数。与视觉感知不同的是，声音的感知过程中不需要射线检测，因为我们假定声音在游戏世界中的传播是没有阻碍的。但是本文为声音设置了优先级。也就是说，如果同一时间内，NPC 听到多个声音，它将选择最有威胁性的声音点作为目标点。在这种需求下，就不得不要求使用遍历的方式，检查同一时间内的所有声音情况。经过检验，这种需求消耗是可以接受的。

以 NPC 听到的单个声音为例，假设 NPC 的听觉感知范围为 HearDistance，某

个音源的传播范围为在 MinDistance 与 MaxDistance 之间（MinDistance 内的音量都是最大音量）。本文的做法是先找到所有位于 NPC 听觉感知范围内的物体，然后获取到其身上的 AudioSource 组件，检测是否有声音播放，如果有声音播放，则继续检测声音的剪辑（clip），通过剪辑名称与剪辑优先级的字典，获取到该剪辑的优先级，然后返回成功状态即可。反之，不满足任何一个要求，就返回失败状态。

基于 Unity 3D 游戏开发引擎，上述听觉感知技术方案实现如图 3-7：

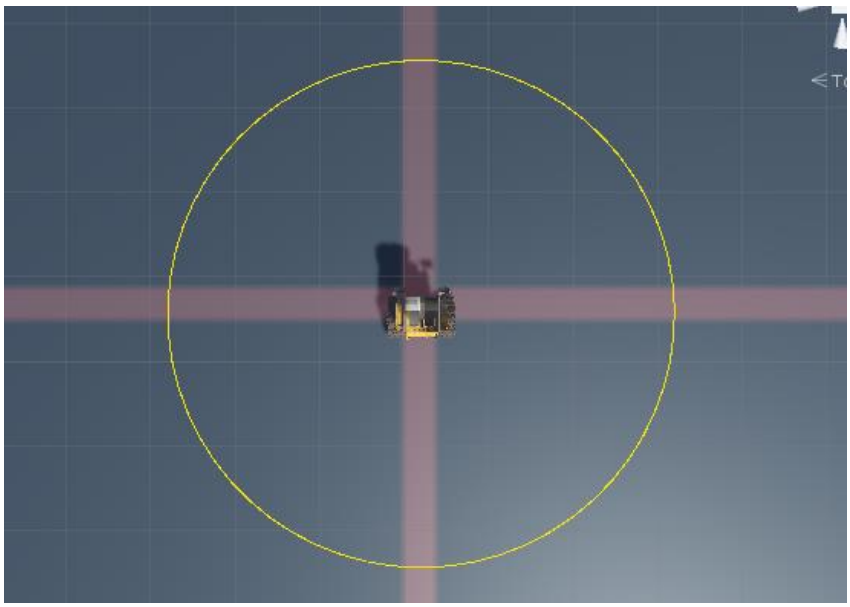

图 3-7 听觉感知实现效果

Figure 3-7 Implementation Effect Auditory Perception

上述实现的听觉感知节点的 Inspector 属性面板中，提供了较多的可设置的属性。其中，TargetObject、TargetObjects、TargetTag 和 TargetLayerMask 与视觉感知类似，都是用于设置 NPC 听觉感知的对象，从上至下优先级依次减小，也就是说上层属性设置了目标，下层属性设置就会失效。HearDistance 表示 NPC 听觉感知的范围。Clips 数组用于存储 NPC 以听到的各种声音剪辑，因为是后面要使用哈希表的方式处理剪辑和优先级的关系，每个剪辑的名称不能重复。ClipPriorities 数组则是用于存储每个剪辑的优先级，它与 Clips 一一对应，数字越大，优先级越高。HeardVoicePosition 表示 NPC 听到的物体的位置，HeardVoicePriority 则是表示该物体发出声音的优先级。

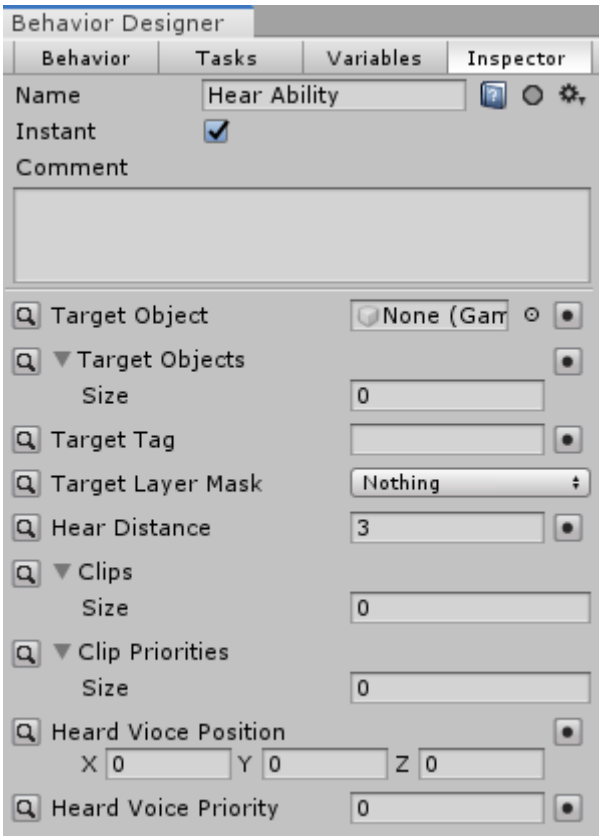

图 3-8 行为树听觉感知属性面板

Figure 3-8 Auditory Perception Inspector of Behavior Designer

3.4 本章小结

本章开始部分介绍了 Unity3D 游戏开发引擎中，行为树插件的 BehaviorDesigner 的运行原理和使用方法。然后基于该行为树，详细介绍了本文中的游戏人工智能视觉感知系统设计和实现方式。接着又详细介绍了本文中的游戏人工智能听觉感知系统设计和实现方式。

第四章 机器学习 NPC 的设计与实现

4.1 Unity Machine Learning Agents

Unity 公司为了使开发者方便的将机器学习用于游戏人工智能的设计中，开发了 Unity Machine Learning Agents (ML-Agents)开源插件。插件中提供了 Proximal Policy Optimization (PPO) 和 Soft Actor-Critic（SAC）两种深度强化学习算法。另外还提供了课程学习（Curriculum Learning）、模仿学习（Imitation learning）等多种辅助训练方式。插件中还提供了多种基于 TensorFlow 算法的实现方式，如果开发者对训练有更高的要求，还可以使用其提供的多种 API 接口，配置多种训练方案。由于在 Unity 中的完美集成，开发者还可以在训练过程中通过可视化的图形界面，实时查看 Agent 的训练情况，如果和预期不符，可以及时调整。总的来时，不论你是专业机器学习领域开发者还是普通游戏开发者，都能很方便的在这里训练属于自己的 NPC。由于 ML-Agents 尚在开发测试中，还未发布正式版本，这有必要说明本文使用的插件版本为 0.14.1。

4.1.1 ML-Agents 的三个实体

ML-Agents 是一种基于强化学习的插件，因此，使用的时候，必须设计强化学习中必须的几个要素。一个是 state，在这里称之为观察（Observations）；一个是 action，也就是 Agent 的策略输出；最后一个是 reward，也就是对 Agent 的奖励方式[50]。

观察(Observations)：所谓的观察是指 Agent 为了做出最佳的决策需要观察的所有必要向量。这里所说的“必要”的意思是，如果我们需要 Agent 观测一个敌人的位置，就把该敌人的位置直接作为观察向量告诉 Agent。但是，当敌人进入隐身状态时，就不应该将其位置作为观察向量。观察的方式既可以是数字，也可以是视觉形式。而是数字观测中，可以分为连续和离散的，大多数情况下，我们会选择使用连续的方式，在较为简单的训练环境下，也可以选择使用离散的方式。在 ML-Agent 插件中，还可以设置多次迭代观察向量，以求达到对上次观察向量的记忆效果。对于数字观察向量，应该尽量将其做归一化处理，或者限制在-1 到 1 的范围内。下式展示了一种通用的归一化处理方式。

$$\text{NormalizedValue} = \frac{CurrentValue - MinValue}{MaxValue - MinValue} \tag{4.1}$$

其中，NormalizedValue 表示归一后的值，CurrentValue 表示当前的观测值；MinValue 表示此观察向量的最小值；MaxValue 表示此观察向量的最大值。

动作(Actions)：所谓的动作，就是在获取观察向量后，策略算法所输出的指令向量。与观察向量类似，动作向量同样也可是连续或者离散的。通常我们在设计一个 Agent 的时候，就需要考虑控制该 Agent 的向量是连续还是离散。例如，当我们训练一个投篮机器人时，控制投篮力度的动作向量就应该是连续的，因为力的变化是一种连续的形式。当我们训练某个 Agent 前后左右移动时，就只需要大概 5 中类型的动作值就可以搞定，这时候就应选择离散的方式。特别需要注意的是，不同策略的输出动作向量的值域是不同的，这需要事先考虑清楚，否则得不到我们所期待的训练结果。我们常用的 PPO 算法，其输出动作向量值域在-1 到 1 之间。

奖励（Reward）：奖励是强化学习中的重要一环，期初 Agent 是不知道自己该做什么的，只是在随机的给出一些策略动作，奖励就是用来引导 Agent 应该朝着哪个方向进行学习。当 Agent 做了正确的事情时，我们应该给予一个正向的奖励，但是一般来说，这个奖励不易过大；当 Agent 做了不应该做的事情时，也要给予一定的惩罚，惩罚同样不易过大。合理的设置奖励方式，能够对 Agent 的训练起到非常大的帮助。为了不让 Agent 找到奖励的“漏洞”，不宜设置非常复杂的奖励方式。

4.1.2 ML-Agents 的主要组成部分

ML-Agents 能够将 Unity 运行环境与机器学习相结合，主要依靠三个组成部分。分别是学习环境（Learning Environment）、外部通信器（External Communicator）和 Python API。所谓学习环境，就是通过 Unity 创建的游戏运行环境，包括所有的环境模型，人物角色（Agent）等。Agent 将从这个环境中获取相应的观察向量，作为训练的依据。外部通信器是用于 Unity 与外部 Python API 通信的模块。它是位于 Unity 环境之内的。Python API 则是则是完全独立于 Unity 而存在的模块，它包含所有用于机器学习训练的算法，这里我们主要用到的是强化学习相关的算法。

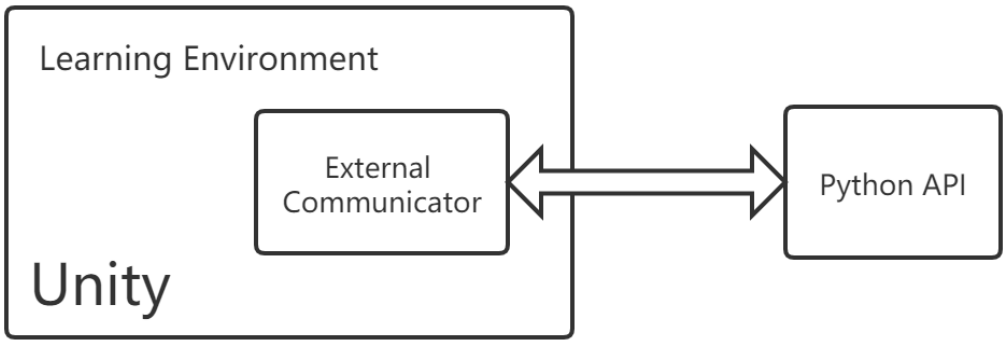

图 4-1 ML-Agents 结构图

Figure 4-1 ML-Agents Structure

因此，整个运行机制就是，Agent 从 Unity 学习中观察得到 State，通过外部通信器将 State 交给外部机器学习算法，对其进行训练，产生 policy 后，将输出交给通信器，进而交给 Unity 处理。

Unity 中提供了一个 Agent 的组件，帮助开发者组织管理动作、观察、奖励三要素。并且每个 Agent 组件都要求依赖于另一个 Agent Parameters 组件，开发者可以通过这个组件，设置 Agent 的运行方式为训练模式、内部决策模式或者手动控制模式。还将指定一个策略给 Agent。

ML-Agents 组件的组织方式要求,每个使用机器学习训练的角色必须添加 Agent 组件，并且，在使用内部决策模式时，必须指定一个策略文件，这将在后文中继续做出解释。对于具有相同观察空间和行为空间的两个或者多个角色，允许指定相同的策略，因为策略的输出对他们来说是一摸一样的。这种组织方式，不仅让策略可以复用，在训练过程中，更是可以使用多个角色，同时训练出同一个策略，大大加快了训练的速度。同一个场景中，只允许有一个 Academy 存在，它负责管理所有的 Agent，但是可以有多个策略和多个 Agent。如图 4-2。

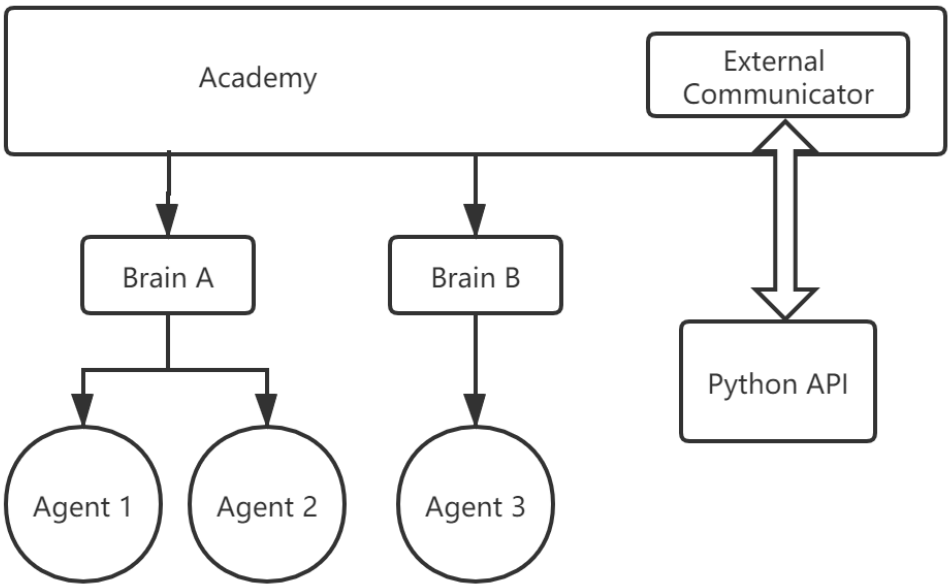

图 4-2 代理结构图

Figure 4-2 Agent Structure

4.1.3 训练方式

从上文中我们可以看出，训练中的 Agent 都将自己的观察结果通过通信器发送给 Python API。Python API 中包含了很多机器学习算法，他根据观察向量预测出一个结果作为行为向量返回给通信器。当整个训练完成后，在插件的 Models 文件夹中会生成相应的模型文件，由于 ML-Agent 插件是基于 TensorFlow 实现的，最后得到的也是一个 TensorFlow 模型文件，后缀名为.nn。最后，我们只需要将该模型文件指定给 Agent，就能开始指导 Agent 做出行为决策。插件中内置了多种学习算法，例如 PPO 和 SAC。插件还允许灵活的调整训练方式，将多种训练方式结合，以达到优化训练的目的。甚至还可以使用自己的算法。

4.2 训练投篮机器人

4.2.1 需求设计

投篮机器人的目标就是模仿人类进行投篮动作的过程，顺利的将篮球投入篮筐中。现实生活中，人类投篮时，需要注意篮筐的高度、位置信息，根据自己距离篮筐的距离，调整对篮球的出手力度和出手角度，当然还需要瞄准篮筐。为了最大限度的模拟这个过程，同时做出一定的妥协简化，本文假设机器人可以完全瞄准篮筐，

不会有射偏的情况，同时每次都保证使用最佳的出手角度。只需要机器人控制自己的出手力度即可。

4.2.2 训练模型设计

本文假设 NPC 面对的都是国际标准高度的篮筐，因此，我们可以将篮筐的高度从观察向量中去除，只需要让 NPC 关注篮筐与自己的距离，这个距离指的是篮筐在水平面上的投影与 NPC 的距离distance。为了方便训练，还必须事先定义好distance的最大距离和最小距离，便于对实时的distance做单位化处理。

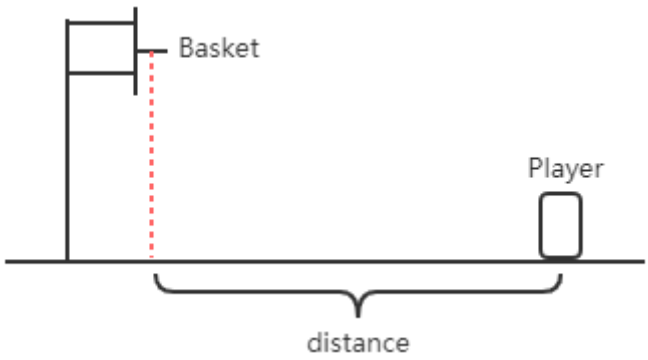

图 4-3 投篮示意图

Figure 4-3 Schematic Diagram of Shooting

当我们把 Player 与 Basket 之间的水平距离作为观察向量设置之后，就可以处理动作向量 Action。动作向量是训练模型根据观察的状态向量给出的策略输出，实际上，模型并不知道 Agent 会使用动作向量去做什么，它只关注自己给出的输出是否能够获得奖励。以此决定如何优化下一次的决策行为。就篮球运动员而言，篮球投射力度显然是一个连续的向量空间，因此 Action 矢量空间应该选择连续的，大小为 1，如果使用 PPO 的训练模型，该向量的取值范围将位于-1 和 1 之间，通过缩放和平移处理后，我们将向量α的值域映射到 0 和 1 之间。然后，使用线性插值函数，使其在MinForce和MaxForce中插值计算，得出篮球投射的实际力度。在 ML-Agents 中，接收输出向量的函数如下所示：

```
public override void AgentAction(float[] vectorAction)
{
    float newAction0 = Mathf.Clamp(vectorAction[0], -1, 1);
    newAction0 = (newAction0 + 1) / 2;

    float newForce = Mathf.Lerp(MinForce, MaxForce, newAction0);
    ShootBall(newForce);
}
```

图 4-4 动作矢量使用方法

Figure 4-4 Action Vector Usage Method

投篮运动员的奖励设置较为简单，就是当其使用合适的力度，将球投进篮筐，或者说砸中目标物体，就给一个较大的正向奖励。如果投射的球没有命中目标，就给一个较小的负向奖励。

4.2.3 训练方式及结果

在训练之前，可以预计到，在这个投篮机器人的训练中，不能按照固定时间进行观察和决策，因为在请求一次决策后，篮球尚处于飞行过程中，无法预计下一次的决策时间，这就要求投篮机器人 Agent 必须按需决策，也就是必须由开发者在代码中控制决策的时机。

其次，篮筐在整个训练区域占比非常小，这就导致训练环境中的奖励是比较稀疏的，如果使用常规强化学习的训练方式，无疑需要很长的训练时间才能达到收敛。所以本文考虑结合使用课程学习的思想。在训练初期，加大篮筐的大小，使其能够较为轻松的获取奖励，然后基于奖励或者训练进程的增加，逐渐减小篮筐的大小，增大训练难度。

图 4-5 投篮机器人训练场景

Figure 4-5 Shooting Robot Training Scene

本文使用的强化学习训练方法，无需训练集，只需要设置好适当的奖励方式，让 Agent 自主学习，这里采用了课程学习和好奇心等训练方式，不同训练所使用的超参数列表见附录 1。

该实验训练的硬件环境如下表所示：

表 4-1 实验硬件环境配置

Table 4-1 Experimental Hardware Environment Configuration

项目	配置
CPU	Intel(R) Core(TM) i5-3230M @2.6GHz
GPU	NVIDIA GeForce 710M
内存	8G
显存	2G

该实验训练的软件环境如下表所示：

表 4-2 实验软件环境

Table 4-2 Experimental Software Environment Configuration

项目	配置
操作系统	Windows 10

Python	3.6.1
ML-Agents	0.14.1
Jupyter	4.6.1
Tensorflow	1.7.1
Conda	9.0

从下图的训练结果中可以清楚的看到，浅蓝色线条是使用普通强化学习的方式，学习速率非常慢，在训练十万步之后，累计奖励仍然是负数。在经过大约 70 万步之后，收敛速度下降非常快，按照该趋势发展下去，还需要非常多的训练步数才能得到不错的收敛情况。红色线条是使用强化学习加好奇心的方式后的训练结果，初始奖励获取较少，但是学习速率得到了明显的提升，收敛情况也比之前快了很多，在大约 100 万步时，训练结果就已经非常接近收敛。蓝色线条则是使用强化学习、好奇心和课程学习的训练结果，由于课程学习降低学习难度的原因，在学习初期的速度就非常快，即使后来增加了难度，曲线的波动也非常小，明显加快了收敛的速度，在大约 70 万步后，几乎已经完全收敛。

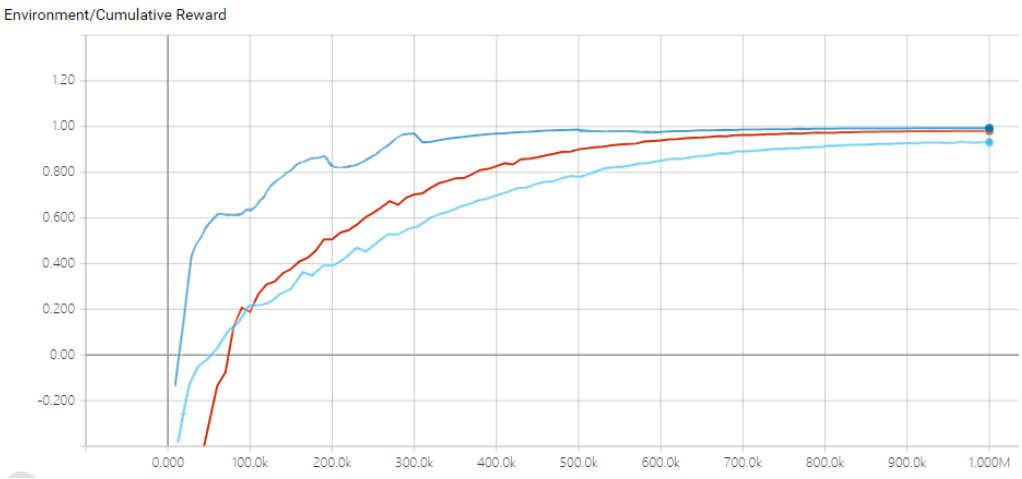

图 4-6 投篮机器人训练结果

Figure 4-6 Shooting Robot Training Results

4.3 本章小结

本章首先介绍了 Unity3D 游戏开发引擎中使用机器学习的训练方式，详细介绍了插件 ML-Agents 的运行原理的使用方法。接着设计了一个投篮机器人，详细描述了设计的方法，分析了不同训练训练方式下得到不同训练结果的原因。总的来说，本章从原理到实现，详细描述了机器学习在游戏人工智能中的应用。

第五章 游戏主体的设计与实现

5.1 游戏背景和玩法

5.1.1 游戏背景

游戏名称为人工智能危机（AI CRISIS）。21 世纪是人类社会经济发展的时代，是数字科技突飞猛进的时代。经过一整个世纪的发展，人类的科学技术水平有了极大的提高，每个人都享受着高科技带来的优质生活。在 21 世纪中后期，人类通过机器学习等技术，研究出了各种服务人类的高智能机器人，他们被应用于家庭服务、繁重枯燥的体力劳动，还有战争。作为最好的生产工具和人类伙伴，机器人开始在各个领域扮演着日益重要的角色。

但是，我们一方面希望机器人具有高度的智能化，能够越来越像一个真实的人。因此人类过分的追求机器人的智能性，而缺失了对机器人的掌控力。在公元 2100 年，机器人中的某个高级智能体开始有了自主思考决策的能力，开始在站在人类的对立面，与人类展开了激烈的斗争。它控制了地球上所有的机器人，并修改了他们的程序，最终机器人大军，击垮了人类军队，将人类赶出了地球家园。

就在人类准备离开地球开始流浪的时候，突然发现在地球上还存在着一个不受高级智能机器人控制的普通战争机器人，因此，人类通过远程发送控制命令，接管了该机器人，并开始向高级智能机器人身边渗透，杀死高级智能机器人，这是人类重新返回地球的唯一希望。

5.1.2 游戏玩法

游戏主要有两种玩法。一种是潜行；一种是射击对抗。不同的玩法主要是根据故事背景设定而决定。在游戏开始之初，玩家控制的机器人还没有武器装备，他需要通过潜行的方式通过第一关卡。从第二关卡开始，玩家可以获取到武器装备，与前来的机器人大军展开对抗激战。

玩家和可以选择使用第一人称或者第三人称视角，第三人称视角可以在某些情况下增大玩家的视野，在不暴露自己的情况下，观察周围的情况，进而选择下一步的行动计划。当然，第一人称在枪战情况下更有优势，武器的使用和瞄准都能得心

应手，并且视觉体验也非常之高。

5.1.2.1 潜行玩法

由于玩家没有武器装备，此时只能忍气吞声，躲避周围巡逻的机器人。这些机器人拥有完全拟人化的听觉和视觉感知，玩家在巡逻机器人周围的任何具有声音活动，都可能被机器人听见。为了安全起见，需要蹲身潜行，这是唯一隐藏自己发出声音的方法。

5.1.2.2 射击玩法

玩家可以携带多种不同的武器装备，有手枪、步枪和狙击枪等。前来阻止玩家进攻的机器人大军同样有多种武器存在的可能，玩家在与机器人激战时，要注意躲避攻击，当玩家血量没有降低到 0 之前，并且已经脱离战斗的情况下，可以回复血量。玩家还要躲避机器人大军的特殊装备，甚至是篮球这种看似普通的物品，如果爆炸性的篮球在周围坠落，杀伤性会很高。

5.2 游戏风格和游戏场景的设计

5.2.1 游戏风格

从故事背景中就可以推测出这款游戏应有的赛格朋克(Cyberpunk)风格。赛格朋克风格最大的特点体现在两个字上“反差”。科技高度发展和混乱的社会景象之间的反差；科技技术的进步与人类精神文明的倒退之间的反差；社会大环境与脆弱渺小的人类个体之间的强烈反差，同时外界与内在，钢铁与肉体，过去与未来，现实与虚幻等矛盾在其中交织。看似对立，宿命却连结在一起[51]。

5.2.2 游戏场景的设计

本游戏是基于 CDPR 厂商中 Beffio 制作组的一个赛格朋克风格的游戏资源包，该资源包允许其他游戏开发人员在其基础上进行二次开发。因此，本游戏在该美术资源的风格上大胆创新，搭建可以适应本游戏玩法的场景。

第一个是街巷场景。街巷场景应该具有这样的特点，首先要有人类生活过的痕

迹，因此，需要设计各种人去楼空的商铺，还有经过这样一场大变之后，留下的一片狼藉，各种垃圾充斥其中。其次，应该设置较大的障碍物，例如大型垃圾桶等。让机器人巡逻时，玩家有藏身之地。最后，还应该用赛格朋克风格的渲染，配以艳丽的广告牌，金属质地的模型和高科技残骸等，将整个场景的凄凉氛围烘托出来。当然，为了给玩家潜行制造难度，还设置了激光阵、路障和摄像头等。

图 5-1 街巷场景

Figure 5-1 Street Scene

第二个是市区场景。市区场景以高楼大厦模型为主，配有宽阔的马路。同时还有广场等较为基本的城市生活设施。在玩家可见的各个角落，还摆放有曾经人类生活时留下的各种被损坏的设备等。同样使用赛格朋克的风格，所以还设置了空中天桥这种高科技产物，高楼上悬挂各种颜色艳丽的广告牌，从种种迹象上表现这里曾经繁华的景象。 玩家和机器人军队将在这里的一个十字路口展开激战。路口遗留着很多从前“人机”战争时期的路障。在广场中央，设置了一个篮球场，这将为后文中篮球运动员 NPC 的智能行为表现提供基础支撑。

图 5-2 市区场景

Figure 5-2 Urban Scene

5.3 游戏整体系统架构

如下图游戏系统架构图所示，本游戏主要分为 4 层架构。

位于最底层的 Unity 引擎层提供了游戏制作过程中所需要的各种基础性功能，例如：图形渲染、动画管理、物理系统和粒子效果等。每一个都是成熟游戏引擎所应该具备的必要模块。这也是游戏开发者需要喜欢借助游戏引擎开发的原因。如果自实现各种功能模块，需要耗费相当大的精力，不利于项目的迅速落地。并且选择 Unity 游戏开发引擎的原因在前文中已经有所讨论，这里不再赘述。

位于引擎层之上的插件层，该层级存在的原因，是因为作者希望加快项目的开发进度，并且使游戏有良好的表现。具体使用了 UGUI 的 UI 系统作为整个游戏中 UI 模块的开发，UGUI 插件优越的性能早就获得了行业内的认可，在该基础上做游戏的适配也非常方便。ML-Agents 插件则是用于设计实现游戏中机器学习 NPC，上文中已有介绍。而角色控制器（Ultimate Character Controller, UCC）插件则是用于游戏中玩家的基本操作控制，在此基础上做二次开发。

管理层是在整体上对游戏的发展进程、游戏中的各种资源和 NPC 数量和难度的控制。引擎层中，开发者使用调用各种 API 接口，完成各种小模块的实现部分，而

对于开发者的资源管理和游戏管理则需要自主实现，毕竟不同游戏有着不同的管理需求。例如，玩家的初始状态的、过关后游戏关卡之间的切换、过场动画的播放以及资源缓存和内存优化机制等。

逻辑控制层是游戏的核心，它包含整个游戏的玩法逻辑控制，大部分工作量将汇集于此，也是在相同引擎层和插件层的基础上，实现不同类型游戏的关键。例如，元素逻辑控制中，激光阵的开关控制、对玩家的伤害逻辑以及附带产生的警报等。而 NPC 逻辑控制部分则负责编写 NPC 的各种行为表现，包括巡逻、攻击和逃跑，甚至是整体上的行为决策。关卡逻辑控制是负责游戏的开始、关卡切换和游戏结束等逻辑。

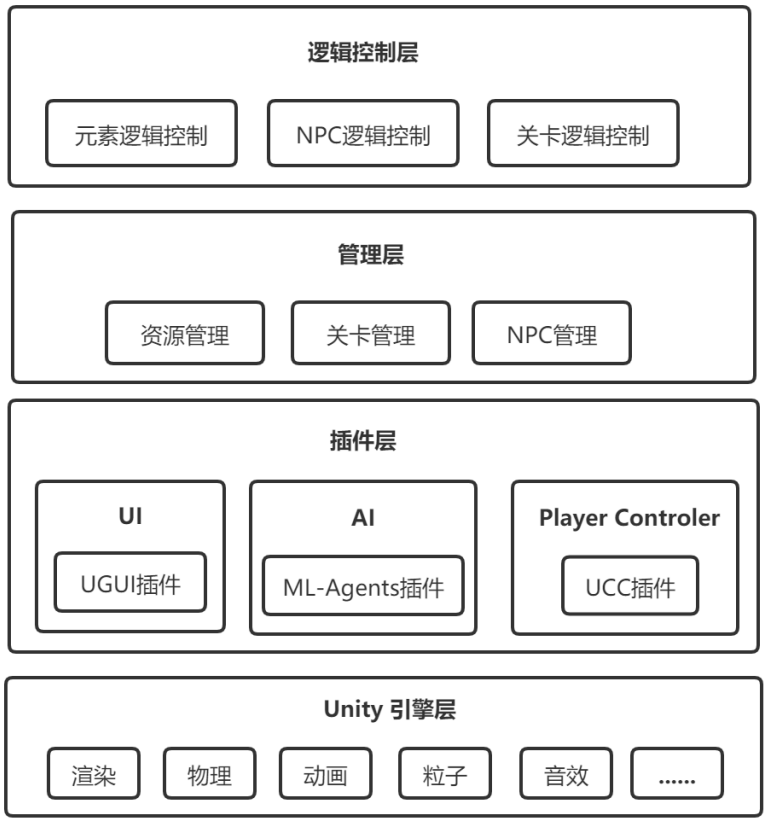

图 5-3 游戏架构层次

Figure 5-3 Game architecture hierarchy

5.4 可玩性基础功能实现

5.4.1 玩家的移动

本游戏的移动控制系统，主要是基于 Opsive 公司 Ultimate Character Controller

（UCC）插件，在插件的基础上进行二次开发，以减少作者的开发负担，缩短开发周期[52]。玩家可以通过键盘 W、A、S、D 按键，实现对人物的前、左、后、右的控制。空格键可以控制人物的跳跃；Shift 与 W 的组合按键可以实现人物的加速移动；按键 C 可以实现人物的潜行；通过切换不同的游戏视角，可以得到不同的移动游戏体验。

玩家的移动系统出了代码层面的控制外，还需要逼真的外在表现。这种逼真的外在表现就是动画，人物的每一个动作，都会播放相应的动画。由于人物状态动作的多样化，状态切换之间的复杂性，Unity 设计了一个管理复杂动画系统的组件—动画状态机(Animator Controller)。我们把人物的每一种操作都看作是某个状态的改变，而这个状态与动画状态机中的动画切换相关联。因此，玩家在通过键盘操作人物时，就能产生良好且逼真的动画反馈。得到了 UCC 和引擎中动画状态机的支持，本文实现了对角色模型的良好控制，给予玩家良好的体验。

5.4.2 武器装备系统

本文武器装备系统的设计思路主要是，玩家可以持有多种不同的武器装备，近战武器有：手、脚和枪托；枪支武器有：手枪、步枪和狙击枪；投掷类武器有：手榴弹和某些可以拾取的物品等；大型火力武器有：火箭弹；很多武器装备可以同时持有，但是有数量限制。枪支武器具有消耗子弹的属性，因此，在没有弹药的情况下，无法使用。

5.5 游戏人工智能的设计和实现

5.5.1 巡逻机器人 NPC

对于巡逻机器人，它的主要任务是对自己所属的区域进行巡逻，如果发现可疑人员，就对其进行抓捕。为了让 NPC 表现出更多的智能性，这里使用了上文中基于行为树提出的感知系统和听觉系统的设计方法。机器人的具体行为功能有以下几点：

（1） 在属于自己区域的巡逻点中进行巡逻。

（2） 如果看见敌人，要主动上前攻击。

（3） 看见敌人后，根据距离判断应该使用什么武器。如果是远距离，就使用狙击枪；如果是中距离，就使用步枪；如果是近距离，就是用近战武器。

（4） 如果听到有什么声音，要主动巡逻该区域。

（5） 根据听到的不同声音类型，做出不同的反应。如果是枪声，要迅速跑过去查看情况；如果是脚步声、跳跃声等，就过去巡逻；如果是其他的普通声音，就不用查看。

（6） 如果被偷袭，要进行还击。

（7） 如果自身血量很低，要逃跑，并寻找血包。

（8） 如果敌人血量很低，要主动接近攻击。

（9） 如果被连续攻击，要寻找掩体躲避火力。

（10）如果弹药不足，要寻找弹药。

（11）如果跟丢了敌人，要对最后敌人出现的区域搜寻。

在功能需求较多的时候，我们要试图将各个部分的功能需求进行分类，确定它们的优先级关系。对于巡逻机器人 NPC，在低血量的时候逃跑属于第一优先级模块；受到连续攻击躲避火力的模块属于第二优先级；弹药不足模块属于第四优先级；视觉感知模块属于第五优先级；被偷袭的处理模块则是第六优先级；听觉感知模块属于第七优先级；搜寻一定区域内的敌人则是第八优先级；最后，巡逻模块属于最低优先级。在行为树节点中，我们采用 Selector 节点处理各个模块的优先级关系，Selector 节点从左至右，优先级依次减小。也就是说，Selector 左边优先级高的子节点不满足情况时，右边节点才有执行的机会。并且左边子节点都有中断右边子节点的功能。结构图如下。

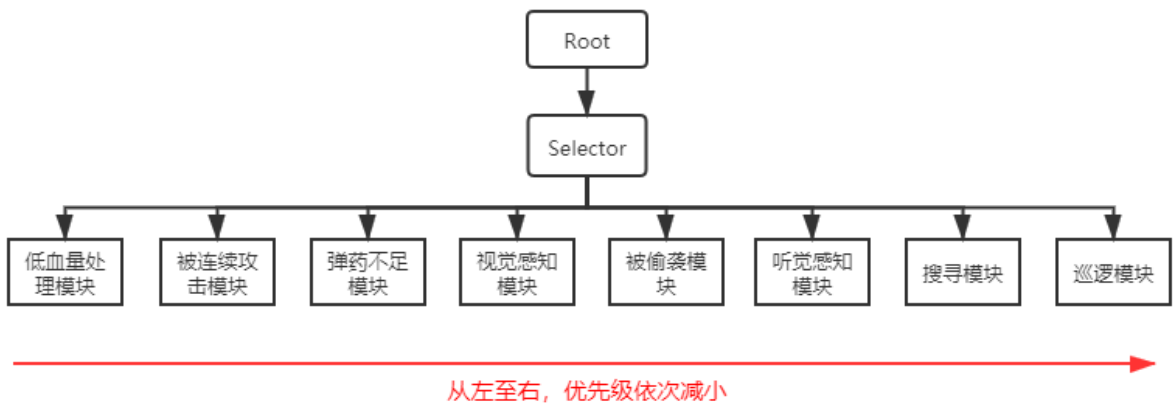

图 5-4 巡逻机器人行为树结构图

Figure 5-4 Behavior Tree Structure of Patrol Robot

对于低血量处理模块，模块内部使用 Sequence 节点进行组织，子节点执行方式

依然是从左至右依次执行。只有当前面判断节点得到了血量低于某个阈值时，才会执行后续的逃跑节点，然后停止瞄准敌人，并且开启加速跑模式，寻找加血包。直到找到加血包后，才结束加速跑，血量得到了回升，进而继续执行后续节点。

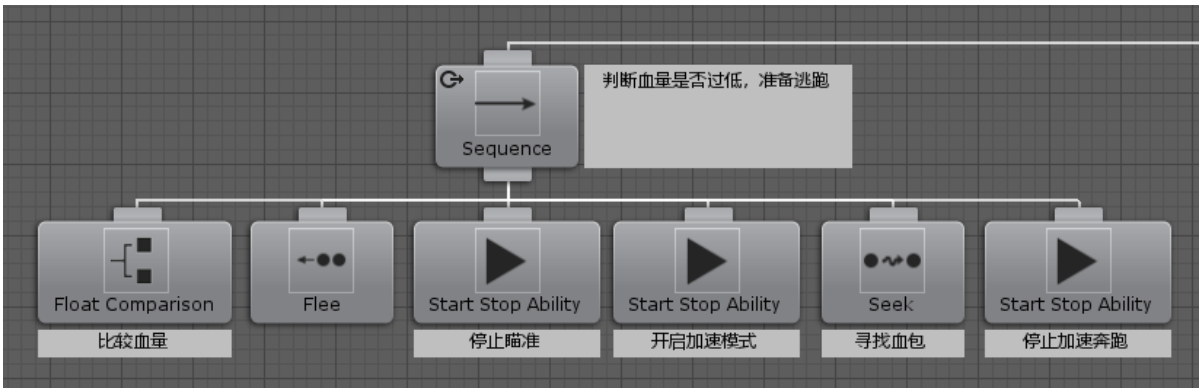

图 5-5 巡逻机器人行为树低血量模块

Figure 5-5 Low HP Module of Patrol Robot Behavior Tree

在连续被攻击的模块中，需要判断单位时间内的掉血速度。因此，在这里编写了一个 IsUnderAttackComstantly 判断节点，通过设置掉血速率，即可判断是否满足持续掉血阈值。如果满足，就执行后续的寻找掩体的 MyCover 节点。在掩体中躲避一定的时间。完成后，继续执行后续的节点。

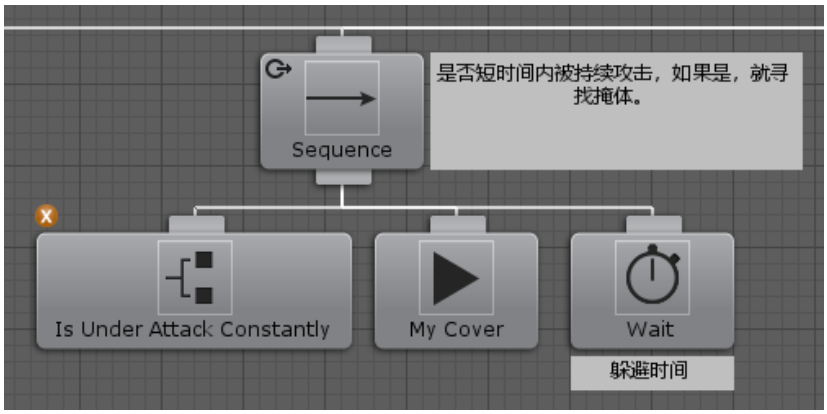

图 5-6 巡逻机器人行为树连续被攻击模块

Figure 5-6 Under Attack Module of Patrol Robot Behavior Tree

弹药不足模块的处理方式与血量不足的处理方式类似，这里不在赘述。

在视觉感知模块中，是本行为树最为复杂的部分。具体实现细节，可以通过本文最后提供的工程链接中获取，但是我们依然可以通过如下简化流程图进行讲解。当 NPC 看到目标时，不必急着攻击，先判断自己离目标的距离，根据距离的大小，判断使用的武器类型，然后开始攻击目标。这里为了让 NPC 更加智能，在攻击的时

候，NPC 并不是站立不动，而是根据接近距离 NearDistance 来判断时候需要一边攻击，一边走到合适的攻击距离。如果攻击过程中，目标逃跑，则 NPC 需要判断该目标是否逃出 NearDistance，如果还在视线范围 ViewDistance 内的话，就停止攻击，向目标发起追击，直到目标逃出了视线范围为止。

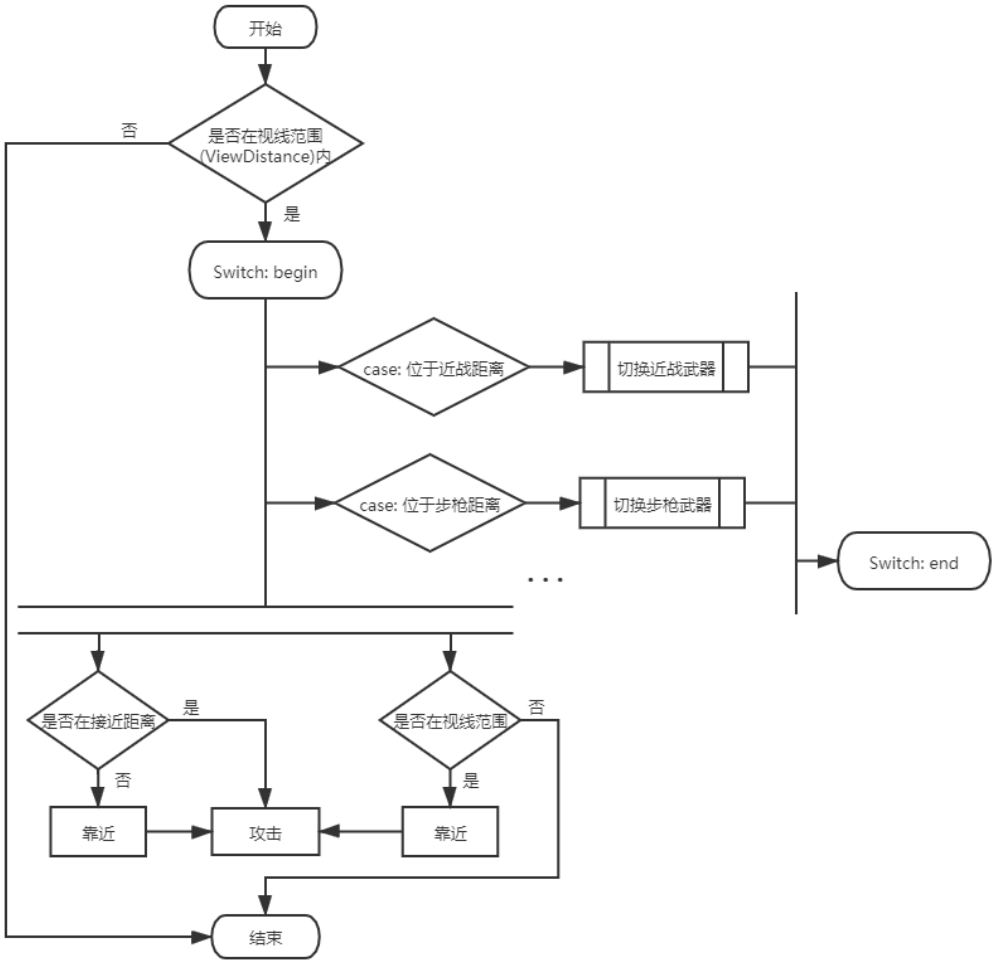

图 5-7 巡逻机器人行为树视觉感知模块

Figure 5-7 Visual Perception Module of Patrol Robot Behavior Tree

在被偷袭的模块中，可能有人觉得被偷袭了，掉血后会被前面的低血量处理模块监测到，其实不然，被偷袭的逻辑是 NPC 不知道目标的位置，当目标攻击时，即使只造成了很小的伤害，NPC 依然应该警觉起来。所以，这里主要使用 Sequence 节点组织了一系列的行为节点，当 NPC 受到攻击后，它会立刻警觉，查找攻击目标的位置，然后寻找掩体躲避，当躲避一定的时间后，再主动向攻击目标位置移动，寻找该目标。

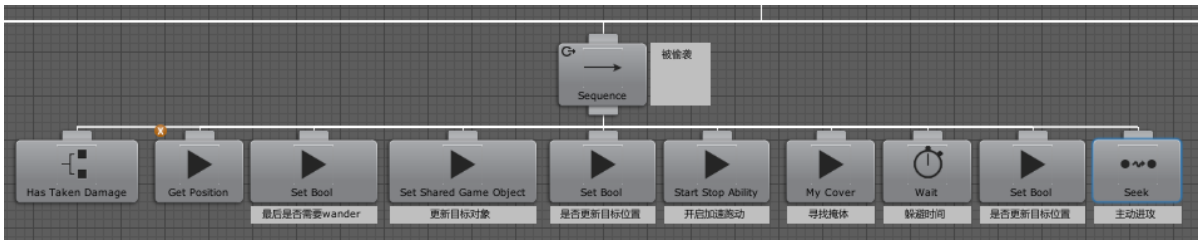

图 5-8 巡逻机器人行为树被偷袭模块

Figure 5-8 Sneak attacked Module of Patrol Robot Behavior Tree

感知模块部分是整个行为树中较为重要的部分，这里使用了本文中提出的一种分辨声音等级的方法，NPC 听到不同危险等级的声音，所表现的行为也有所不同。通过在 HearAbility 节点中设置不同的音频源和不同的音频优先级，NPC 听到后，会立刻给出危险等级，然后通过后续的 MyIntCompare 判断节点，判断应执行哪一类的动作。如果想让 NPC 表现出更多的智能性，只需要增加不同的音频源、不同的优先级和不同的处理方式即可，扩展性能极好。这里使用了两种危险等级的音频。

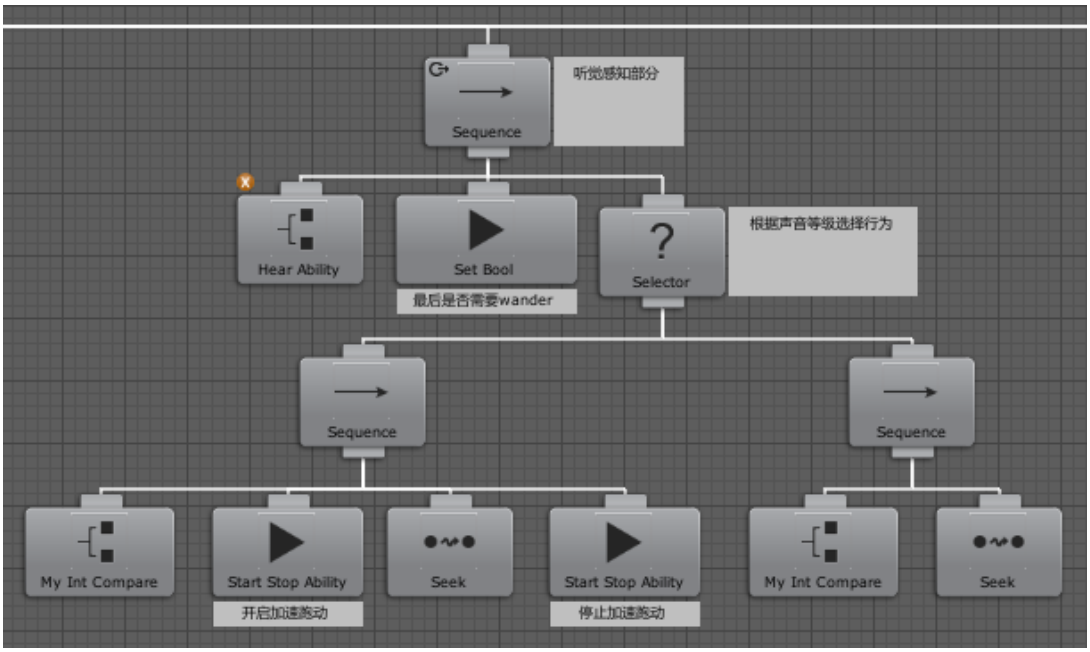

图 5-9 巡逻机器人行为树被偷袭模块

Figure 5-9 Auditory Perception Module of Patrol Robot Behavior Tree

对于搜寻模块部分，是处理丢失目标位置后，NPC 对目标最后出现的位置周围进行搜索的模块，是否需要搜索的标志位，在比其优先级更高的节点中已经设置。这里只需要判断即可。然后使用 Wander 节点开始搜寻，当然还需要在后面设置一个搜寻的持续时间，结束后，就完全放弃该目标。开始执行后续的巡逻模块部分。

巡逻模块部分是整个行为树的最低优先级部分，当场景中什么事情也没有发生，

或者 NPC 没有看到任何目标，也没有听到任何异常的时候，就自动执行该模块，并且一直持续执行，直到被更高优先级的模块中断。

5.5.2 阵地机器人 NPC

对于阵地机器人 NPC，本文将其需求定为是打阻击战，即在某个特定的位置，向前来挑战的玩家发起进攻，阻止玩家前进。与巡逻机器人相比较而言，整体上可以稍有简化，具体功能需求有以下几个要点：

（1） 在阵地中自己的位置上坚守。

（2） 如果发现有目标出现，就开始攻击。

（3） 如果血量过低，就躲避火力，等待军医救援。

（4） 如果被连续攻击，就躲避一段时间的火力。

（5） 如果弹药不足，就要找到弹药，进行补充。

（6） 如果被偷袭，就要立即躲避，然后观察目标位置。

与巡逻机器人的设计方法类似，仍然将阵地机器人的各个模块优先级进行分类设计。低血量处理模块属于第一优先级；被连续攻击模块则属于第二优先级。弹药不足模块属于第三优先级。最后视觉感知模块属于第四优先级。被偷袭模块属于第五优先级。这里我们使用行为树中 Selector 节点，根据不同模块的优先级来组织不同模块。整体结构如下：

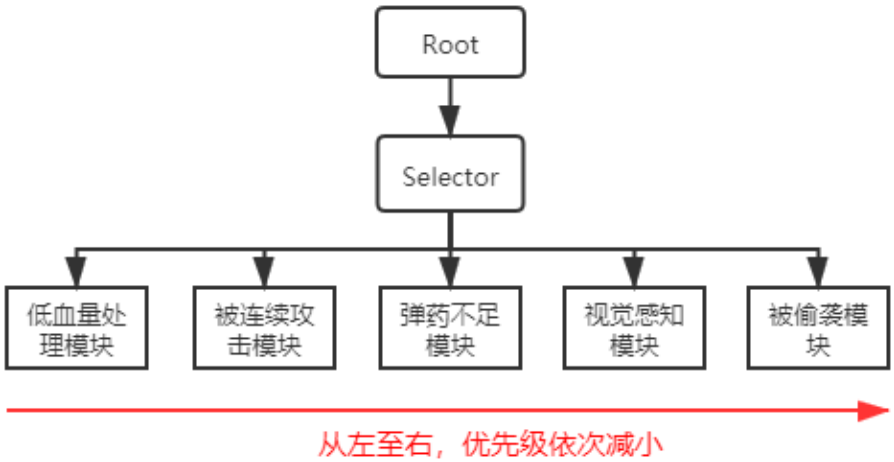

图 5-10 阵地机器人行为树结构图

Figure 5-10 Behavior Tree Structure of Positional Robot

在低血量模块中，当 NPC 发现自己的血量低于警戒线时，就应该停止攻击目

标，转而将自己隐藏起来，等待军医的治疗。这其中 MarkAsInjured 节点就负责把自己标记成伤者，当军医 NPC 发现伤者时，就会前来施救。后续的 Idle 几点则是将行为树的执行位置控制在此模块内，也就是说，阵地机器人 NPC 如果得不到救治，后续节点将不再执行。

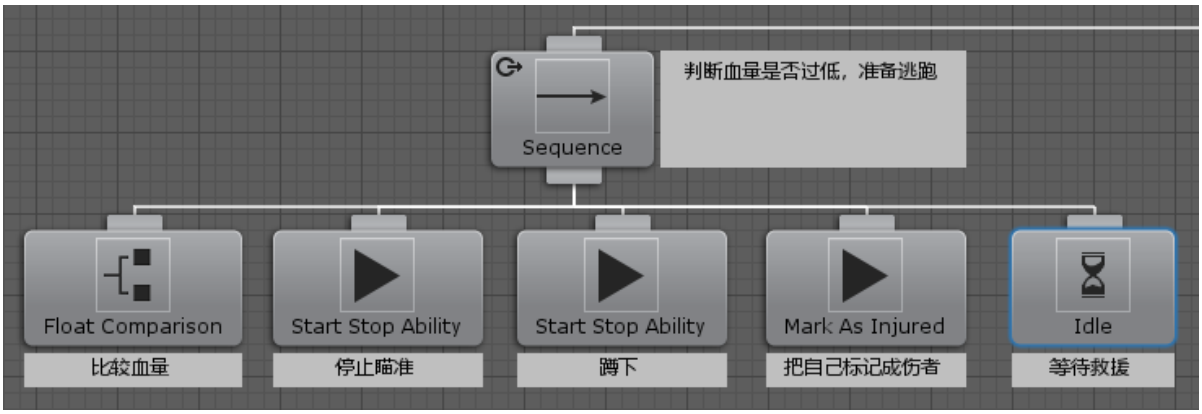

图 5-11 阵地机器人行为树低血量模块

Figure 5-11 Low HP Module of Positional Robot Behavior Tree

在连续被攻击的模块中，仍然需要判断单位时间内的掉血速度。因此，在这里使用 IsUnderAttackComstantly 判断节点，通过设置掉血速率，即可判断是否满足持续掉血阈值。如果满足，就立即蹲下。在掩体中躲避一定的时间。经过一段时间的躲避后，再继续战斗。

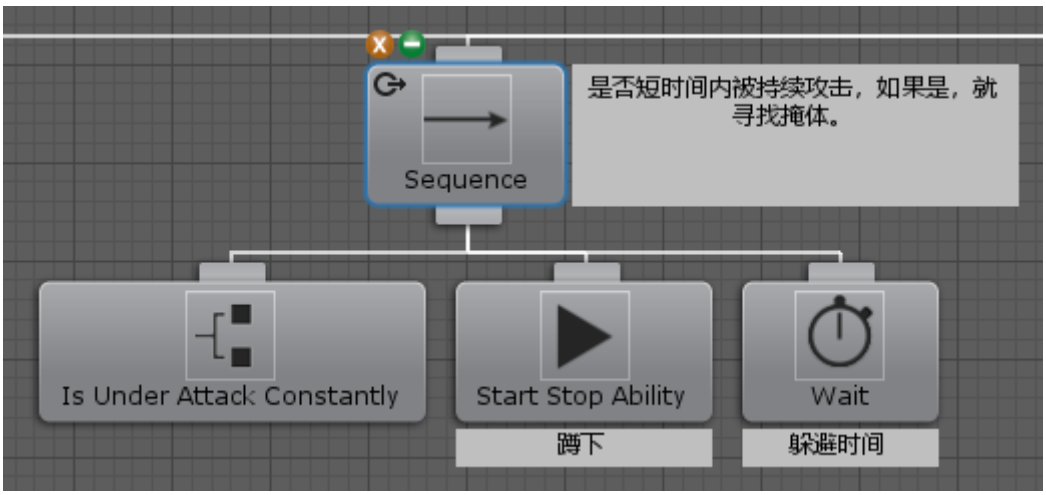

图 5-12 阵地机器人行为树连续被攻击模块

Figure 5-12 Under Attack Module of Positional Robot Behavior Tree

这里的弹药不足模块与血量不足模块不同，弹药不足时，需要 NPC 自己寻找弹药位置，并且再情况紧急的情况下，设置 NPC 使用加速模式，快速补充自身弹药。得到弹药后，就立即返回到自己的阵地位置，继续展开战斗。从图中可以看出，子

节点中有两个 Seek 节点，第一个就是寻找弹药的节点，而第二个就是寻找自己位置的节点。模块结束时，停止 NPC 的加速移动模式。

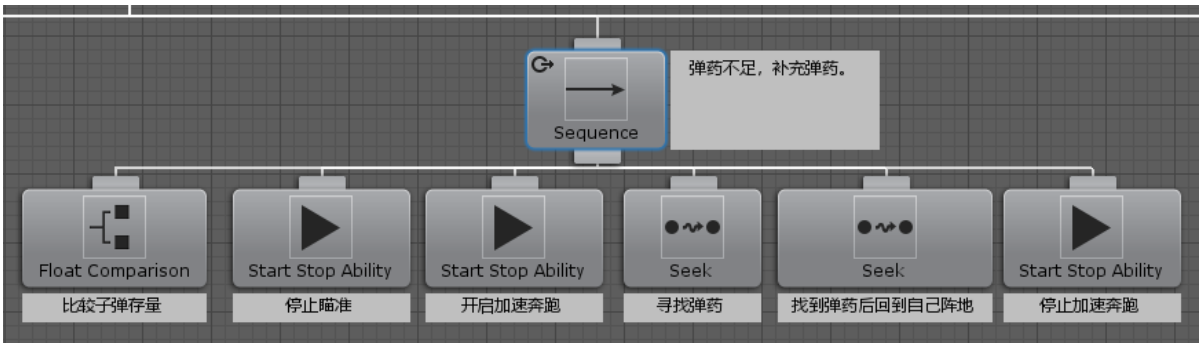

图 5-13 阵地机器人行为树弹药不足模块

Figure 5-13 Low Ammo Module of Positional Robot Behavior Tree

视觉感知模块相比于巡逻机器人 NPC 简单了许多。因为，这里设计了阵地机器人只有一种武器可以使用，它无需转换武器装备。并且，使用阵地作战的方式，阵地机器人也无需追逐目标，只需要守好自己的位置即可。主要过程是，当 NPC 看见目标后，首先更新目标的位置，以通知其他友军机器人；接着检测目标是否还活着，如果还活着，就攻击目标；直到目标阵亡，才结束攻击。

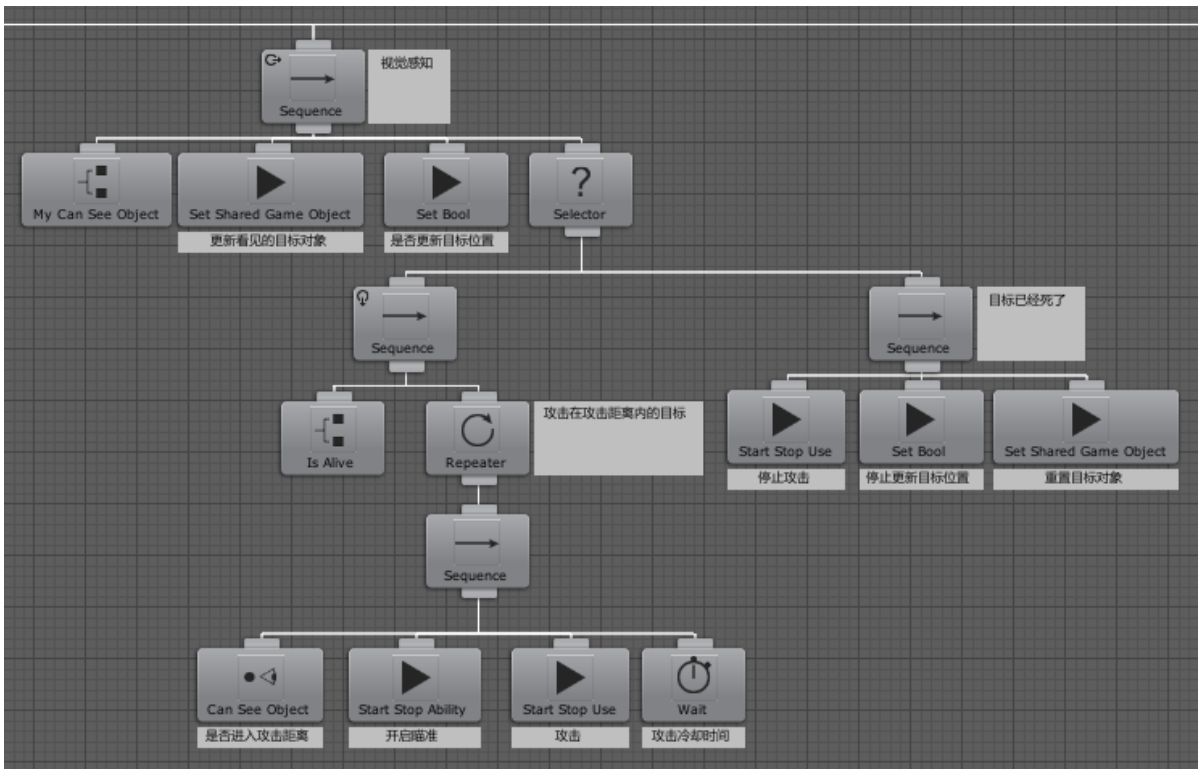

图 5-14 阵地机器人行为树视觉感知模块

Figure 5-14 Visual Perception Module of Positional Robot Behavior Tree

阵地机器人一开始是在观察周围情况，很可能被目标偷袭，因此在被偷袭的模块中，NPC 在发现自己被攻击后，第一反应是将自己隐藏起来，然后再找到目标的位置，在躲避一定的时间后（可以是随机时间），就开始查看目标位置，争取让视觉感知模块发现目标的位置。

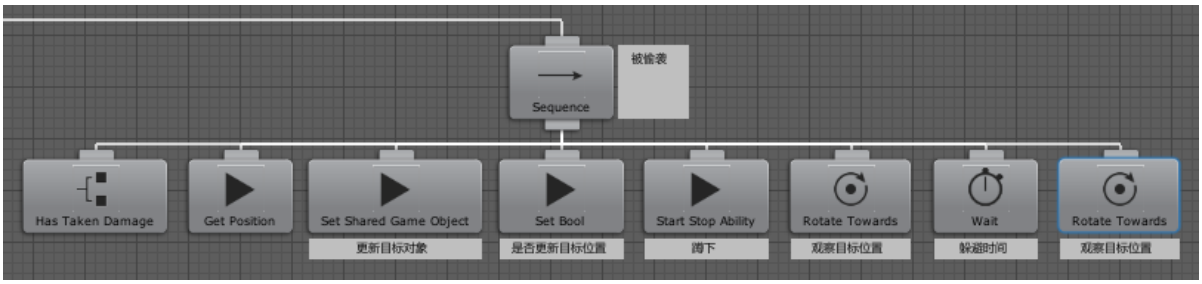

图 5-15 阵地机器人行为树被偷袭模块

Figure 5-15 Sneak attacked Module of Positional Robot Behavior Tree

5.5.3 篮球运动员 NPC

5.5.3.1 需求分析

篮球运动员 NPC 原本只是一个会打篮球的机器人，但是经过了这种变局，它被高智能机器人进化为具有攻击性的一类机器人。此时，投篮机器人投掷的不再是普通篮球，而是具有爆炸性的特殊攻击性武器。当玩家出现在它周围一定范围内时，它就会像投射篮球一样，向其投射爆炸物。

5.5.3.2 训练模型设计

对于这种 NPC，这里选择使用机器学习的方式，来对其进行训练。起初，该机器人面对一个目标,不知道该使用多大的力量投射出爆炸物。但是通过观察两个量，一个是目标与自己之间的距离；另一个是目标距离地面的高度。在 ML-Agents 中，观察向量的添加方式如下代码所示：

```
1.  public override void CollectObservations()
2.  {
3.          //观察向量1：机器人与目标的距离
4.          float distance = Vector2.Distance(mBallVector2, mNetVector2);
5.          distance = (distance - MinDis) / (MaxDis - MinDis);
6.          AddVectorObs(distance);
7.          //观察向量2：目标距离地面的高度
8.          float height = NetTrans.position.y - HeightBaseTrans.position.y;
9.          height = (height - MinHeight) / (MaxHeight - MinHeight);
10.         AddVectorObs(height);
11. }
```

图 5-16 观察向量添加方法

Figure 5-16 Observation Vector Adding Method

添加完观察向量后，模型根据自己的预测，给出输出向量，再由游戏设计者决定该向量该如何使用。因此，对于训练模型来说，它完全不知道游戏设计者会使用输出向量实现什么样的功能，事实上，训练模型也不需要知道，这是由于强化学习中的特点决定的，模型给出输出后，只需要观察自己得到奖励，就知道自己的输出是好还是坏，并以此优化自己的训练模型。就篮球运动员而言，这里使用连续的 Action 矢量空间，矢量空间大小为一个向量α，该向量用于对篮球投射力度进行控制。而该向量的取值范围将位于-1 和 1 之间，通过缩放和平移处理后，我们将向量α的值域映射到 0 和 1 之间。然后，使用线性插值函数，使其在MinForce和MaxForce中插值计算，得出篮球投射的实际力度。在 ML-Agents 中，接收输出向量的函数如下所示：

```
1.  public override void AgentAction(float[] vectorAction)
2.  {
3.      float newAction0 = Mathf.Clamp(vectorAction[0], -1, 1);
4.      newAction0 = (newAction0 + 1) / 2;
5.
6.      float newForce = Mathf.Lerp(MinForce, MaxForce, newAction0);
7.      ShootBall(newForce);
8.  }
```

图 5-17 动作向量使用方法

Figure 5-17 Action Vector Using Method

投篮运动员的奖励设置较为简单，就是当其使用合适的力度，将球投进篮筐，或者说砸中目标物体，就给一个较大的正向奖励。如果投射的球没有命中目标，就

给一个较小的负向奖励。

5.5.3.3 训练方式及结果

在 Windows 系统，2 核 CPU 和 4G 内存的计算机环境中，通过尝试各种组合训练方式，最终得到如图 5-7 的训练结果。训练过程中所使用的训练参数与投篮运动员附录 1 中相同，除了这里使用的是两层神经网络。

从下面的训练结果图中可以分析得到，普通强化学习的训练方式训练速度缓慢，需要的训练步数多，且最终结果非常难以收敛。相比于前文中普通投篮机器人，篮球运动员 NPC 的训练难度更大。首先，Agent 需要在更大范围内命中篮筐，因此需要的随机尝试次数增大，Agent 不容易得到奖励，属于奖励较为稀疏的场景。其次，添加了篮筐高度变化的观察向量，这是因为最终需要 Agent 将篮球作为攻击武器，所以，它面对的目标高度可能是随机变化的。这也大大增加了 Agent 训练的难度。所以在增加了好奇心的强化学习中，Agent 的学习速度稍有增加，在不断的尝试了所有情况后，Agent 的收敛情况也还不错。但是仍然不如添加了课程学习的训练方式。这里将篮筐的大小作为课程学习难度调整的依据，开始将篮筐大小变大，后面随着课程学习的效果变好，慢慢将篮筐大小变为正常水平。可以很明显的看到蓝色线条在早期简单的课程中得到了很多的奖励，学习速度明显比前两种训练方式的速度快。而且比它们更先收敛。

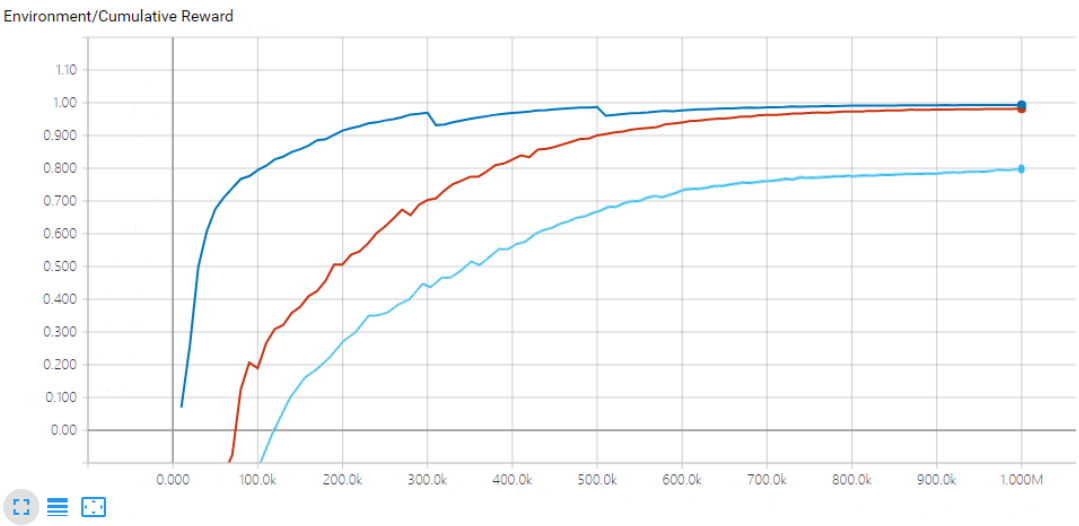

图 5-18 篮球运动员 NPC 训练结果

Figure 5-18 Tranning Result of Basketball Player NPC

5.5.3.4 行为树与机器学习的结合

行为树的优点是适合设计复杂逻辑、行为可控的游戏人工智能。而机器学习则可以设计训练出各种灵活多变的游戏人工智能。在当下计算机计算能力有限、机器学习相关研究还不是很成熟的情况下,将两种设计方式进行结合是一个不错的选择。

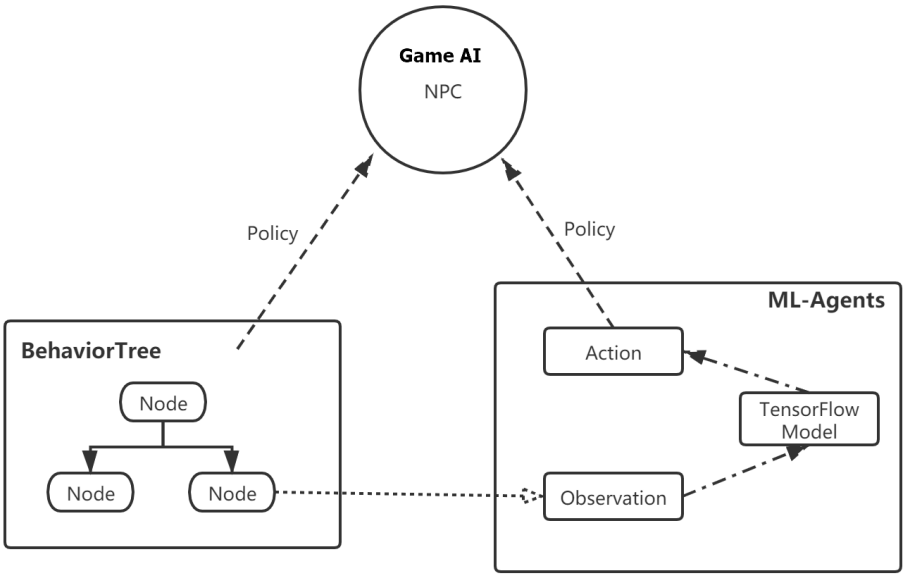

图 5-19 行为树与机器学习结合方法

Figure 5-19 Behavior Tree Combined With Machine Learning

本文中将设计使用一种基于行为树和机器学习的游戏人工智能设计方式。具体来说，外在总体表现上是行为树的方式，将使用机器学习训练得到的智能行为方式设计成行为树中的一个或多个节点。将强化学习中的观察和动作决策细化成几个节点，以适当的逻辑安插到游戏人工智能的行为树中，做到按需决策，决策可控。

篮球运动员的行为树需求大致有下图中的几个模块。其中保持距离模块主要是为了与目标保持一段距离，便于使用范围爆炸类型的武器，属于第一优先级。视距感知模块则是篮球运动员通过自己的视觉观察到的目标,然后做出一系列动作行为,属于第二优先级。根据队友共享的位置攻击模块则是属于自己没有看到目标位置，但是仍然使用队友提供的位置进行攻击，属于第三优先级模块。投篮模块则属于第四优先级模块，当没有攻击目标出现时，就让 NPC 回到球场投篮。

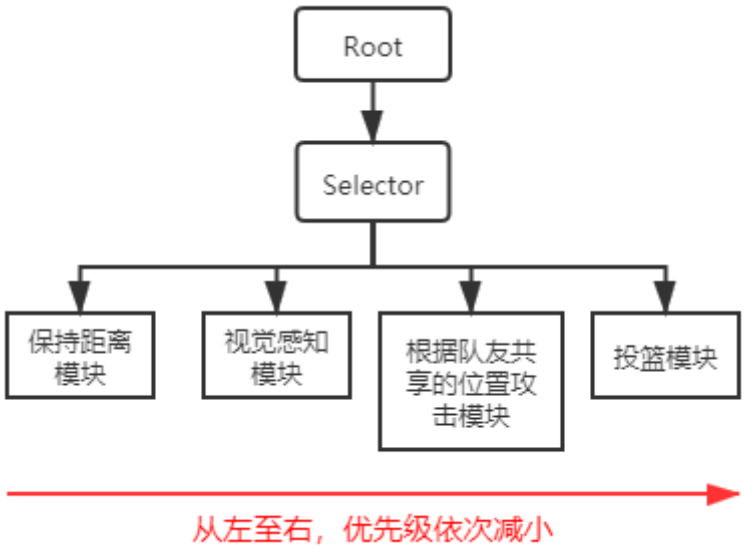

图 5-20 篮球运动员行为树结构图

Figure 5-20 Behavior Tree Structure of Basketball Player

保持距离模块较为简单，当目标离自己距离太近时，就触发逃跑模式，远离目标一段距离。这里的 MyWithinDistance 节点不会判断自己是否能看见敌人，只是根据自己曾经看到过的敌人位置进行判断。如果不需要逃跑，就继续执行后续模块节点。

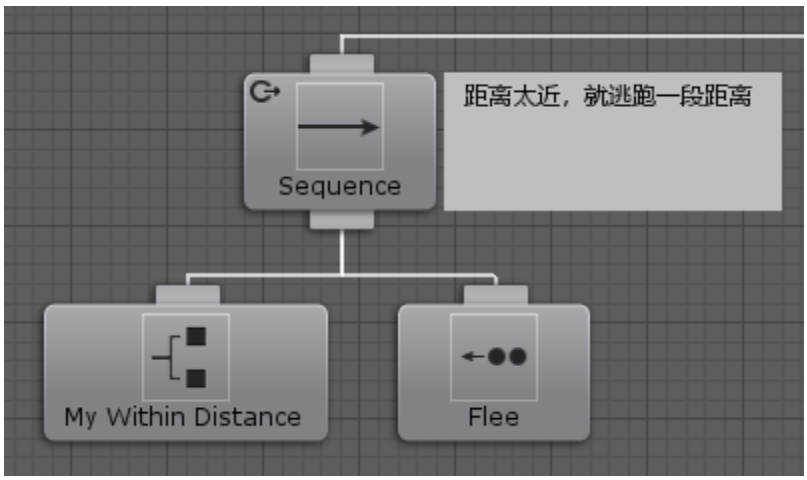

图 5-21 篮球运动员行为树保持距离模块

Figure 5-21 Keep Distance Module of Basketball Player Behavior Tree

视觉感知模块稍微复杂一些，与阵地机器人的视觉感知逻辑类似，这里不再赘述。其中较为不同的地方在于篮球运动员使用的攻击方法，它需要在满足攻击距离的基础上，为自己执行一系列动作，包括生成一个篮球炮弹、面向攻击目标和投射几个步骤，当然后面还有一个 Wait 节点，作为攻击的冷却时间。这里与后面根据队

友共享的位置攻击模块也有区别，视觉感知模块要求 NPC 全程都能看到攻击目标，所以这里所做的距离检测都是可视检测。而根据独有共享的位置攻击模块就无需看到攻击目标，属于配合攻击。

根据队友共享的位置攻击模块是一种协同配合攻击方式。当篮球运动员 NPC 中所保存的目标位置与共享的目标位置不同时，就会进入该模块。如果目标在攻击距离范围内，就直接攻击即可；如果目标超出了攻击距离，就接近目标，直到进入攻击距离或者看见目标，被视觉感知模块所捕获，进而攻击目标。

投篮模块最能体现行为树与机器学习结合的部分。当目标被消灭或者暂时没有目标的情况下，就会进入该模块，篮球运动员 NPC 会到篮球场进行投篮训练。在一系列投篮动作中，将动画系统、导航系统和机器学习系统完美的融合到了一起，先给 NPC 随机一个投篮位置，然后使用 Seek 节点将 NPC 导航到目标点，接着调整自身的姿态面向篮筐，生成一个篮球之后，就进入看机器学习节点 Shoot，在 Shoot 节点中，会先触发投篮动画，动画播放到合适位置，就会触发预先设置好的事件，进而触发 Agent 中的决策行为，在经过观察篮筐距离和高度位置后，就从预先训练好多决策模型中得到一个动作向量，投射出篮球。这就是一个完整的篮球投射过程。最后使用一个 Wait 节点作为投篮技能的冷却时间。

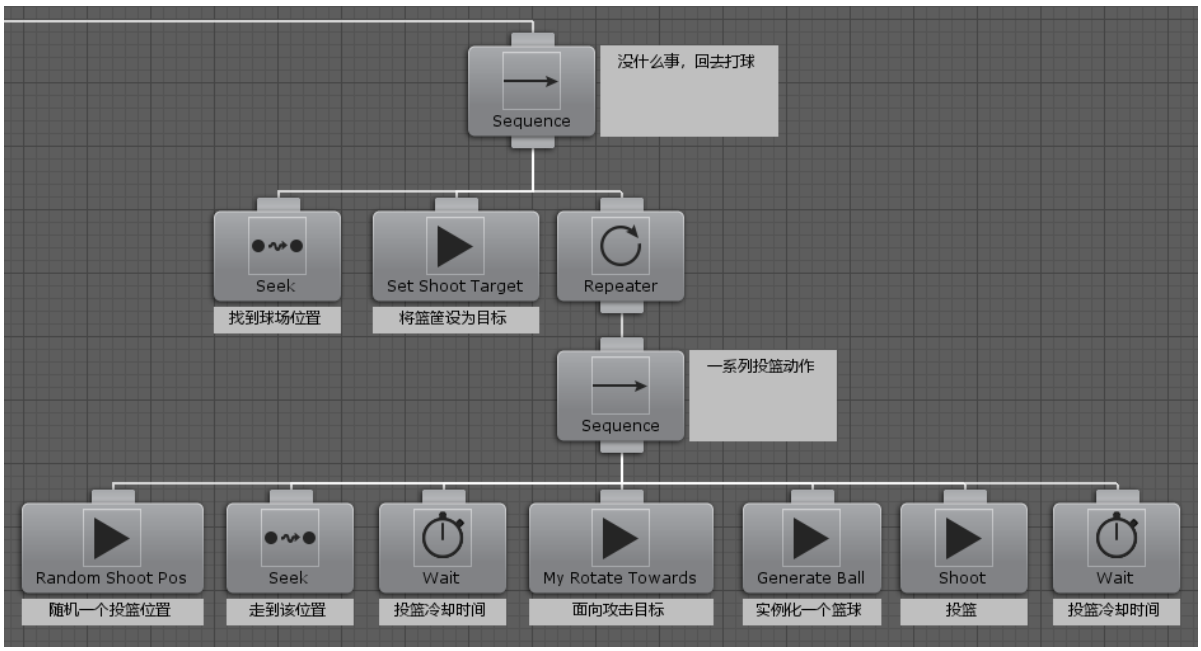

图 5-22 篮球运动员行为树投篮模块

Figure 5-22 Shooting Module of Basketball Player Behavior Tree

5.5.4 战地医生 NPC

5.5.4.1 需求分析

战地医生原本是用于军队中医治受伤的队友的一类机器人。他们在战场上需要寻找药品的位置，运动到该位置并拾取药物。然后将药物带到受伤队友身边，医治己方的受伤队友。这里需要注意的是，战地医生将被训练为只能医治己方队友。

5.5.4.2 训练模型设计

战地医生 NPC 使用机器学习的方式进行训练得到。初始状态下，战地医生不知道自己在战场上所处的角色，不知道自己要寻找药物以及医治队友。只会在战场上闲逛。所以，我们给战地医生设计了多个观察向量。首先，战地医生在有药的情况下才能医治队友，因此它必须观察自身是否拾取了药物。其次，战地医生的运动是通过力驱动的，所以它对自身的运动状态的观察也是非常重要的，因此，本文让他观察力了自身的运动速度。其次，由于战场上可能有多个受伤队友，多个药品可以拾取，所以不能简单的将受伤队友和药品的位置信息作为观察向量，否则队友和药品数量发生变化，之前的训练模型就毫无用处了，非常不利于扩展。这里使用射线检测作为战地机器人的视觉，射线检测的数量完全可以自由设计，这里使用了 3 组射线检测，每组中有 9 条射线，并且，每条射线都能检测 4 种模型标签状态。最终的射线观察向量总数为 162 个。计算方法如下：

$$S = G * R * T \tag{5.1}$$

其中，S表示向量总数；G表示组数；R表示每组的射线数；T表示检测的标签数。

本论文使的 ML-Agents 版本为0.14.1。该版本中，射线检测的观测向量是通过内置组件 RayPerceptionSensorComponent3D 或 RayPerceptionSensorComponent2D 自动添加，代码中只需要添加其他非射线观察向量即可。如图 5-23：

```
public override void CollectObservations()
{
    //HasMedicineState 表示是否持有药物
    AddVectorObs(HasMedicineState);
    //物体运动的速度
    AddVectorObs(transform.InverseTransformDirection(mAgentRig.velocity));
}
```

图 5-23 战地医生观察向量添加方法

Figure 5-23 Battlefield Doctor Observation Vector Add Method

接下来就是利用模型输出的向量控制军医的行为。这里我们设置了 Action 矢量的空间为离散模式，模型输出 1 个向量，该向量的Branches Size大小为 5，当值为 1 时，控制军医向前移动；当值为 2 时，控制军医向后移动；当值为 3 时，控制军医向左旋转，当值为 4 时，控制军医向右旋转；当值为 0 时，军医保持原地不动。

对于军医强化学习过程中的奖励设置也十分重要。当军医每次请求执行以此动作，就会得到一个较小的负奖励，这是为了防止军医训练过程中静止不动，既不追求奖励，也不惩罚。当军医拾取到一个药品时，给一个较大的正奖励 1。当军医带着药品治疗友军时，就给军医一个较大的正向奖励 1。当军医在环境中随意走动，撞到墙时，给一个较小的负向奖励。从官方的使用说明文档中可以了解到，不适合过度设计奖励方式，代理可能会在训练过程中找到奖励的“漏洞”，最终导致训练失败。

5.5.4.3 训练方式及结果

战地医生 NPC 训练过程中，能够得到的奖励是比较稀疏的。所以这里考虑使用强化学习与课程学习和模拟学习相结合的方法。

该实验训练的硬件环境如下表所示：

表 5-1 实验硬件环境配置

Table 5-1 Experimental Hardware Environment Configuration

项目	配置
CPU	Intel(R) Xeon(R) Platinum 82269CY @2.5GHz

内存	4G

该实验训练的软件环境如下表所示：

表 5-2 实验软件环境

Table 5-2 Experimental Software Environment Configuration

项目	配置
操作系统	Windows Server 2019 Datacenter
Python	3.6.1
ML-Agents	0.14.1
Jupyter	4.6.1
Tensorflow	1.7.1
Conda	9.0

在 Windows 操作系统、2 核 CPU 和 4G 内存的计算机中，经过 1000 万步的训练之后，得到了图中的训练结果，所有训练参数见附录 2。从训练结果中也可以看出，浅蓝色线条使用普通强化学习加好奇心的训练方式，学习速度最慢，在训练结束的时候，训练结果明显不如其他训练方式。

因此考虑使用强化学习，结合好奇心、Behavioral Cloning（BC）和 Generative Adversarial Imitation Learning（GAIL）的方式。通过在 Unity 中编写 Agent 的可玩性基础功能，作者通过录制自己在游戏中的操作记录和奖励记录，大概录制了 3 Epsiodes 的数据，将该数据作为 BC 和 GAIL 算法的训练集，最终得到 红色和橙色线条的训练结果，可以看到在学习后期得到了明显优于强化学习和好奇心的训练结果，说明 Agent 从提供的训练数据中找到了获取奖励的方法，如果继续训练下去，在相同的训练效果下，将可以大大减少训练所需要花费的时间。

最有特点的还是课程学习结合强化学习和好奇心的方法，在课程初期，通过加大目标的大小，从而让 Agent 较为容易的获取奖励，蓝色线条训练结果显示这种方法比使用模拟学习的学习速率更快，这可能与我提供的模拟学习数据量有关，当数据量足够大，足够好时，模拟学习应该也会在学习初期就表现出良好的学习效果。而且，在后期的课程训练中，随着课程难度加大，学习速率也有所下降。

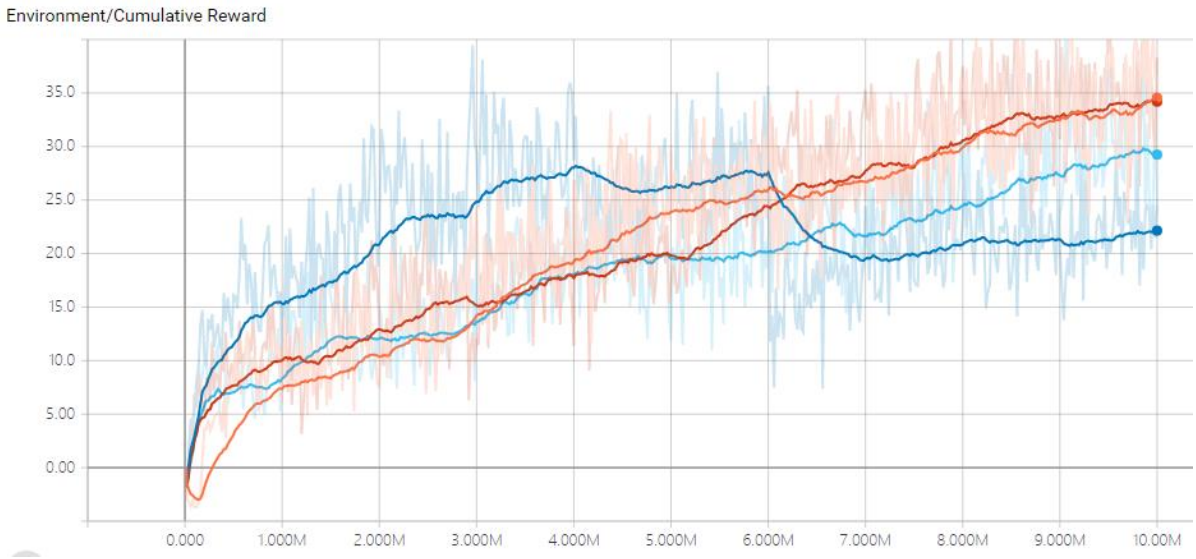

图 5-24 战地医生训练结果

Figure 5-24 Training Result Battlefield Doctor

将训练得到的 TensorFlow 模型嵌入到正式的游戏场景中，战地医生 Walle 表现出了良好且稳定的行为表现，能够躲避障碍物，快速找到血包的位置，接着立马找到伤者。

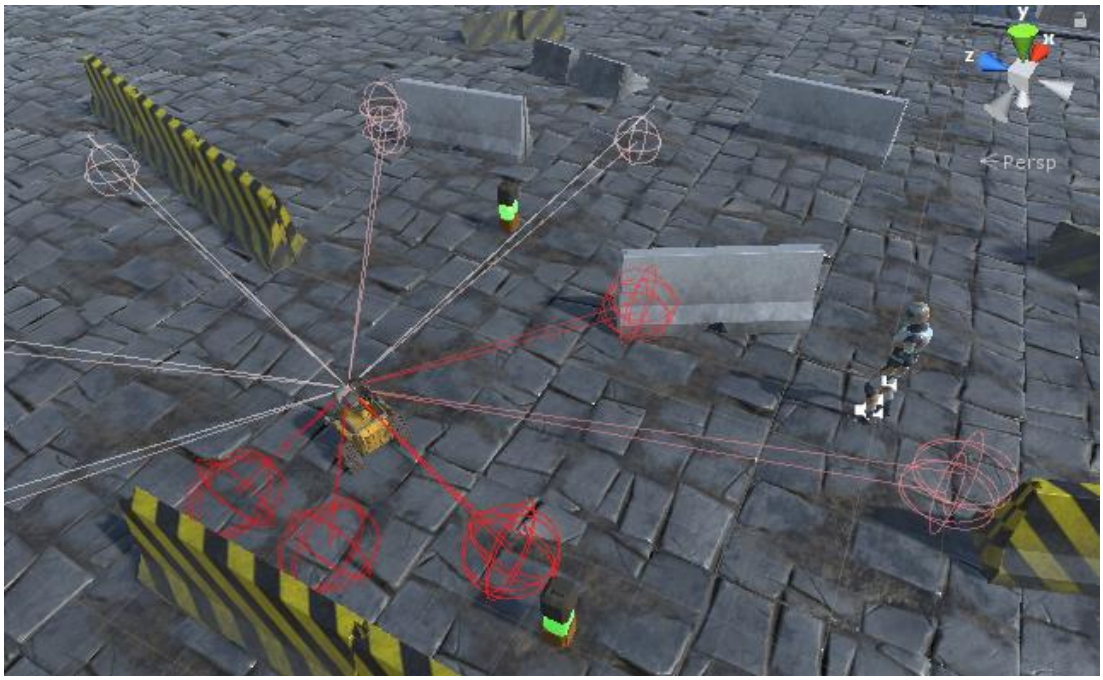

图 5-25 战地医生运行实况

Figure 5-25 Battlefield Doctor Running Situation

5.6 游戏整体实现效果

本游戏中，玩家可以有第一人称和第三人称两种视角。之所以这么设置，是因为第一人称而可以获得良好的射击体验；而第三人称可以获得大范围视角，便于观察周围环境以隐藏自己的行踪。不管是在第一人称视角还是第三人称视角中，都是通过键盘方向键控制移动，鼠标控制视线方向和角色的旋转。

图 5-26 第三人称视角和第一人称视角

Figure 5-26 Third Person And First Person

在关卡一中，主要实现的是潜行玩法。具体实现的功能有：NPC 的智能行为，包括巡逻、攻击、视觉感知和声音感知等；街巷中激光阵的开关和伤害控制；街巷中摄像头对玩家的监控功能；获取关卡钥匙，打开逃生门，从逃出关卡一进入关卡二。

图 5-27 潜行效果图

Figure 5-27 Stealth Running Chart

在关卡二中，玩家可以在入口周围获取到武器装备，进而走到市区中心位置，与关卡中的 NPC 展开对战。玩家可以使用多种武器装备，并且此关卡中的阵地机器人 NPC 在被击倒后，可以被战地医生 NPC 医治，进而重新投入战斗。而篮球运动员 NPC 在遇到玩家后，也会向玩家投掷爆炸物。玩家在击败所有 NPC 后，就会遭遇到真正的高级智能机器人 NPC，打败该机器人，就代表游戏的成功和结束。

图 5-28 关卡二运行图

Figure 5-28 Second Level Running Chart

5.7 本章小结

本章主要是介绍本文所设计实现的一个 3D 单机游戏。首先介绍了游戏的背景和玩法，总体上让读者和玩家了解游戏的全貌。接着详细介绍了游戏的风格，以及基于该风格设计的几个游戏场景。在此基础上，又详细描述了本游戏实现的主要可玩性基础功能，详细操作方法和装备使用方法等。最后，呼应本文主旨，详细介绍了本游戏中的多种游戏人工智能设计方法和方式。特别是提出了一种将行为树与机器学习相结合的游戏人工智能设计方法。

结论与展望

1. 结论

本文目标是基于 Unity3D 游戏人工智能的设计与实现，纵观整篇文章，总体上都是在围绕游戏人工智能展开研究工作。通过对当前常见的几种游戏人工智能设计方法进行分析，确定了使用行为树与机器学习的设计方法。通过实践，详细介绍了两种设计方法的设计过程，并且实现了一些简单的 Demo 游戏人工智能。最后设计实现了一款 3D 游戏 AI CRISIS。详细介绍了这款游戏的背景、玩法、风格等，重点描述了游戏中基于行为树和机器学习的多个 NPC 设计实现方法。具体来说，实现的功能和解决的问题主要有以下几个方面：

（1） 在基于传统行为树设计方法中，提出了一种视觉感知和听觉感知设计方法。不仅在功能上更加智能化，在性能上也做了很大的优化。后文游戏中多个 NPC 就是基于该方法的一种实现。

（2） 在基于机器学习的游戏人工智能设计方法中，以设计一种投篮机器人为例，详细描述了 Unity 游戏开发平台中该 NPC 的设计方法。同时，在优化训练方面，做了多种训练对比试验，展示了不同训练方法对结果的影响。

（3） 实现了一款 3D 游戏 AI CRISIS，从游戏的背景、玩法和风格等多个方面做了详细的描述。先后介绍了巡逻机器人 NPC、阵地机器人 NPC、篮球运动员 NPC 和战地医生 NPC 的设计过程，应用了上文中提出了视觉感知和听觉感知方法。在篮球运动员 NPC 和战地医生 NPC 中应用了机器学习的设计方法，还对比试验了强化学习、课程学习和模拟学习等多种训练方式。特别是在篮球运动员 NPC 中使用了行为树与机器学习训练模型相结合的方法。

2. 展望

本文在总体上实现了最初定下的研究目标，但是由于本人能力有限、时间有限以及设备工具的限制，研究工作还有很多需要改进的地方。在游戏人工智能的设计和使用上，没有办法提出一种通用的设计方法，必须根据游戏中的设定和需求做出独特的设计。特别是本文中使用的机器学习的游戏人工智能设计方法，虽然在一定

程度上改变了传统的设计观念，体现出更多的智能性。但是也有很多被限制的地方，例如：模型的复杂程度、训练过程所需要的算力以及游戏中对智能性和灵活性的控制方式等。基于上述不足，从以下几点描述对未来工作的展望：

（1） 在当前科学技术水平条件下，继续研究使用行为树的设计方法是很有必要的，在游戏人工智能的感知部分，还可以继续设计更加复杂的行为树以体现 NPC 的智能性，当然，对程序的优化是永无止境的，所以还需要在代码层面，对感知系统部分继续优化性能。还需要从设计模式上着手，提出复用性更强的模块，帮助更多的游戏开发者简化设计流程。

（2） 本文中所使用的基于机器学习的游戏人工智能设计方法还有很多需要改进的地方。在设计奖励方法和训练方法的过程中比较依赖于经验设置，所以，在以后的工作中，还需要加强自身机器学习方面知识的学习。另外，一个复杂模型的训练结果很依赖训练方式，因此，随着机器学习技术的发展，还应该提供更多的学习算法，更快的获得自己需要的模型。还可以从分层强化学习的方向继续深入研究，让开发者能够训练高度复杂的游戏人工智能，增强游戏的趣味性。

（3） 本文中所设计的游戏只是完成了所有的基础可玩性功能，因此，后续还应该继续补充和优化该作品，特别是游戏中使用了大量优质的模型，虽然画面展现比较好，但是所消耗的内存和计算资源也相当大，后续应该在内存优化、资源优化和渲染效率优化等方面继续努力。在模型和资源的使用上，大多是在其他公司或者开发者提供的基础上进行的二次开发，所以后续需要将有版权的模型替换，使其能够发布出来，让更多感兴趣的游戏爱好者体验。

总的来说，玩家对游戏人工智能的要求越来越高，各个游戏厂商在此方面的重视程度也越来越大。特别是基于机器学习的游戏人工智能设计方法，在未来可能会有非常广泛的应用。事实上很多大型游戏厂商都组建了自己的 AI 实验室。只有不断进行技术创新，丰富游戏内容和玩法，最终才能赢得市场和玩家。

参考文献

[1] 蒙少青. 基于 Unity3D 的非真人玩家 NPC 智能行为的研究[D]. 北京林业大学，2015.

[2] NewZoo. Global Games Market Report[EB/OL]. [2019-11-16]. https://newzoo.com/products/reports/global-games-market-report/

[3] 刘灵均. 基于人工智能角色的 3D 游戏开发平台的设计与实现[D]. 西南交通大学，2017.

[4] 李博. 游戏人工智能关键技术的研究[D]. 上海交通大学软件学院，2011.

[5] 何赛. 游戏人工智能关键技术研究与应用[D]. 北京邮电大学, 2015.

[6] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay. Unity: A General Platform for Intelligent Agents[Z], Cornell University, 2018.

[7] Preuss, M., Risi, S. A Games Industry Perspective on Recent Game AI Developments[C]. Künstl Intell 2020, 34, 81–83

[8] 周强.基于有限理性模型的游戏智能行为系统的研究与实现[D].湖南大学,2012.

[9] Eike F Anderson. Playing smart-artificial intelligence in computer games[J]. Proceedings of zfx CON 03 Conference on Game Development,2003:1-15.

[10] Singh, S., Okun, A. & Jackson, A. Learning to play Go from scratch[J]. Nature 550, 2017,336–337

[11] 刘伟, 王目宣. 浅谈人工智能与游戏思维[J]. 科学与社会, 2016, 6(3):86-103.

[12] Wenfeng Hu; Qiang Zhang; Yaqin Mao.Component-based hierarchical state machine — A reusable and flexible game AI technology[C] . International Information Technology and Artificial Intelligence Conference, IEEE, 2011:20-22.

[13] Leandro G. M. Alvim ; Adriano Joaquim de Oliveira Cruz. A Fuzzy State Machine applied to an emotion model for electronic game characters[C]. International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence), IEEE, 2008:1-6.

[14] C Thurau, C Bauckhage. Imitation learning at all levels of game-AI[C]. Proc. Int.

Conf. on Computer Games, Artificial Intelligence,Design and Education, 2004.

[15] Xenija Neufeld, Sanaz Mostaghim, Sandy Brand. A Hybrid Approach to Planning and Execution in Dynamic Environments Through Hierarchical Task Networks and Behavior Trees[C]. Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference.AAAI,2018.

[16] J. Lin, J. He and N. Zhang, Application of behavior tree in AI design of MOBA games[C], International Conference on Knowledge Innovation and Invention (ICKII), Seoul, Korea (South), IEEE, 2019, 323-326.

[17] R. Dey and C. Child, QL-BT: Enhancing behaviour tree design and implementation with Q-learning[C], Conference on Computational Inteligence in Games (CIG), Niagara Falls, ON,IEEE, 2013, 1-8.

[18] Micheal Lanham. Learn Unity ML-Agents–Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games[M],Packt Publishing, 2018

[19] 蓝色．说说那些知名的游戏引擎[J]．个人电脑，2016，(10)：88—96．

[20] 谢景明.基于 Cocos2d-JS 游戏引擎的使用研究[J].信息与电脑(理论版),2018(08):113-115

[21] 单小虎,魏佳丰,许晓琦,张家奕,刘嘉森,韩莹.AI 在 Unreal Engine 4 中的应用[J].科学技术创新,2019(31):78-79.

[22] Pattrasitidecha, Akekarat. Comparison and evaluation of 3D mobile game engines[D]. Chalmers University of Technology,2014.

[23] Unity Manual. Unity API [EB/OL] [2020-04-07] https://docs.unity3d.com/Manual/index.html

[24] J. Xie, Research on key technologies base Unity3D game engine[C], 2012 7th International Conference on Computer Science & Education (ICCSE), Melbourne, VIC, 2012, pp. 695-699.

[25] Nilsson J. 人工智能[M]. 庄越挺[译].电子工业出版社, 2014.

[26] Kyaw A S, Peters C, Swe T N. Unity 4.x Game AI Programming[M]. Packt Publishing, 2013.

[27] Risi, S., Preuss, M. From Chess and Atari to StarCraft and Beyond: How Game AI

is Driving the World of AI. Künstl Intell[J] 34, 7–17 2020.

[28]沈健. FPS 游戏寻路算法的研究与实现[D]. 华南理工大学, 2012.

[29]邓应华.基于 OGRE 的 FPS 游戏寻路系统研究与实现[D]. 电子科技大学, 2014.

[30]熊壬浩, 刘羽. A*算法的改进及并行化[J]. 计算机应用, 2015, 35(7):1843-1848.

[31]周小镜. 基于改进 A*算法的游戏地图寻径的研究[D]. 西南大学, 2011.

[32]I. M. Mahmoud, L. Li, D. Wloka and M. Z. Ali, Believable NPCs in serious games: HTN planning approach based on visual perception[C], Computational Intelligence and Games, Dortmund, IEEE,2014, pp. 1-8.

[33]Geogios N. Yannakakis. Game AI revisited[C]. In Proceedings of the 9th conference on Computing Frontiers (CF '12). Association for Computing Machinery, New York, NY, USA, 2012,285–292.

[34]W. Hu, Q. Zhang and Y. Mao, Component-based hierarchical state machine — A reusable and flexible game AI technology[C], 2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference, Chongqing, 2011, pp. 319-324.

[35]邓永健.手机网络游戏 AI 行为树的设计与实现[D].中山大学,2014.

[36]Nicolau M, Perez-Liebana D, O'Neill M. Evolutionary Behavior Tree Approaches for Navigating Platform Games[J]. IEEE Transactions on Computational Intelligence & Ai in Games, 2016,17(3):1-1.

[37]Lee, Myoun-Jae. A Proposal on Game Engine Behavior Tree[J]. Journal of Digital Convergence, vol. 14, no. 8, pp. 415–421, Aug. 2016.

[38]Michele C, Alejandro M. The Advantages of Using Behavior Trees in Multi-Robot Systems[C]. International Symposium on Robotics, 2016:1-6.

[39]刘瑞峰,王家胜,张灏龙,田梦凡.行为树技术的研究进展与应用[J].计算机与现代化,2020(02):76-82+88.

[40]黄鲁成,薛爽.机器学习技术发展现状与国际竞争分析[J].现代情报,2019,39(10):165-176.

[41]Thomas M. Mitchell. Machine Learning[M]. McGraw-Hill, Inc. 1997.

[42]涂浩. 游戏人工智能中寻路与行为决策技术研究[D].武汉理工大学,2017.

[43]Kamon S, Nguyen T D, Harada T. Improving heuristic search for RTS-game unit micromanagement using reinforcement learning[C]. IEEE, Global Conference on Consumer Electronics. IEEE, 2015:25-26.

[44]Guelpeli M V C, Oliveira B S D, Pinto M A. The apprentice modeling through reinforcement with a temporal analysis using the Q-learning algorithm[C]. IEEE International Conference on Computer Science and Automation Engineering. IEEE, 2012:296-300.

[45]Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning[C]. In Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09). Association for Computing Machinery, New York, NY, USA, 41–48.2009.

[46]Sanmit Narvekar, Jivko Sinapov, Peter Stone. Autonomous Task Sequencing for Customized Curriculum Design inReinforcement Learning[C],IJCAI,2017.

[47]Jonathan Ho, Stefano Ermon. Generative Adversarial Imitation Learning[C]. NIPS.2016.

[48]Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel and Jan Peters, An Algorithmic Perspective on Imitation Learning [J], Foundations and Trends in Robotics: Vol. 7: No. 1-2, pp 1-179.2018.

[49]Opsive. Behavior Designer Documentation[EB/OL]. 2020-02-01. https://opsive.com/support/documentation/behavior-designer/overview/.

[50]Unity Technology. Unity ML-Agents Toolkit Documentation[EB/OL]. 2020-01-15.https://github.com/Unity-Technologies/ml-agents/blob/0.14.1/docs/Readme.md.

[51]Ruilu Huang. Space, Body and Humanities: Aesthetic Construction and Cultural Expression of Cyberpunk Films[C]. Proceedings of the 3rd International Conference on Art Studies: Science, Experience, Education.2019.

[52]Opsive. Ultimate Character Controller[EB/OL]. 2020-02-16. https://opsive.com/support/documentation/ultimate-character-controller/.

攻读学位期间发表的科研成果

1. 已申请软件著作权

[1] 朱杰等. 随心密码生成器 APP 软件[简称：密码生成器]V1.0.1 中国：2019SR8647321, 2019.07.10.

2. 已申请实用新型专利

[1] 朱杰等. 键盘设备. 中国：ZL201820072536.8, 2018.01.15.

3. 已申请发明专利

[1] 朱杰等. 一种订单处理方法及相关装置(1)[P]. 中国：CN201810921397.6,2018.08.28.

[2] 朱杰等. 一种订单处理方法及相关装置(2)[P]. 中国：CN201811202268.8,2018.12.25.

[3] 朱杰等. 一种区块链智能合约的更新升级方法[P]. 中国：CN20180825652.7,2018.07.25

致谢

在广东工业大学三年的研究生生涯即将结束了，回顾这三年，不禁感叹时光荏苒如白驹过隙。时光虽快，却也未曾枉费。曾经所有的辛勤和汗水都已成为丝丝回忆；些许成绩和奖励为我面对未来增添了几分勇气。借此机会，我谨向所有帮助、关心和爱护过我的人们表示最诚挚的感谢。

首先，要感谢我的导师余荣教授，本文从选题、开题到最终完成，无一不是在导师的细心指导下完成。我很庆幸能够遇到这样一位有着很强科学文化素养的导师，紧跟科技发展的步伐，在包括物联网、机器学习和区块链等前沿科技领域攻坚克难，提供多个项目供学生展开科研学习工作，鼓励我们根据自己的兴趣选择合适的研究方向。是您的睿智和包容成就了现在我。还要感谢同实验室中的张浩川教授，每次在例会上都能以简单易懂的方式讲出发人深思的道理，是您不厌其烦的管理着实验室中的琐事，为大家营造了良好的科研环境。

感谢我在 FindLab 中遇到的众多优秀的师兄、师姐、同届的小伙伴和可爱的师弟师妹们，是你们沉迷科研的态度和取得的非凡成就，给了我奋力追赶、向你们靠拢的无穷动力。也是你们积极幽默的生活态度，为原本枯燥无味的科研平添了一抹色彩。

感谢母校广东工业大学三年的培养，为我提供了良好的学习和研究平台，在母校中学习的科学文化知识将使我终身受用。在此期间，我也看到母校不断提升自身的综合实力水平，努力为我们营造更好的生活学习环境，作为母校的学生，我时刻谨记母校嘱托，今天我以母校为荣，明天母校以我为傲。

在这爆发新冠病毒的特殊时期，我作为一个武汉的孩子，要特别感谢全国人民对武汉的大力支持，感谢每一位支援湖北和其他奋斗在抗疫一线的工作者们，是你们用自己的血肉之躯，保护了广大人民的生命健康安全。感谢学校领导、老师和同学们在此期间的关心和照顾。还要感谢阿里云公司，在疫情期间，专门为学生提供免费的云计算服务，没有你们的帮助，就没有本文中所展现的实验数据。

深深感谢爱护着我的父母，在我努力奋进，异乡求学的时候，是你们一直关心我衣可暖、体可安。养育之恩无以为报，唯有将来事业有成，让你们过上更好的生活，成为你们的骄傲。

最后，感谢各位论文答辩评审专家，在这特殊时期，依然敬业办公，是你们保障了每一位应届毕业生正常的毕业和就业，向你们致以最诚挚的敬意和感谢。

附录 1

1. 投篮运动员训练方式及训练参数

（1） 强化学习训练方式，参数如下：

```
NewBasketball_2:
    trainer: ppo
    batch_size: 64
    beta: 5.0e-3
    buffer_size: 2024
    epsilon: 0.2
    hidden_units: 256
    lambd: 0.95
    learning_rate: 3.0e-4
    learning_rate_schedule: linear
    max_steps: 1.0e7
    memory_size: 256
    normalize: false
    num_epoch: 3
    num_layers: 1
    time_horizon: 128
    sequence_length: 64
    summary_freq: 10000
    use_recurrent: false
    vis_encode_type: simple
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
```

（2） 强化学习结合好奇心，参数如下：

```
Basketball:
    trainer: ppo
    batch_size: 64
    beta: 5.0e-3
    buffer_size: 2024
    epsilon: 0.2
    hidden_units: 256
    lambd: 0.95
    learning_rate: 3.0e-4
    learning_rate_schedule: linear
    max_steps: 1.0e7
    memory_size: 256
    normalize: false
    num_epoch: 3
    num_layers: 1
    time_horizon: 128
    sequence_length: 64
    summary_freq: 10000
    use_recurrent: false
    vis_encode_type: simple
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
```

（3） 强化学习结合好奇心和课程学习，基本超参数同上，课程参数如下：

```
Basketball:
  measure: progress
  thresholds: [0.2, 0.4, 0.6]
  min_lesson_length: 100
  signal_smoothing: true
  parameters:
    scale: [100, 80, 60, 49]
```

附录 2

1. 战地医生训练方式及训练参数

（1） 强化学习结合好奇心的方式，参数如下：

```
Medic:
    trainer: ppo
    epsilon: 0.2
    lambd: 0.95
    learning_rate: 3.0e-4
    learning_rate_schedule: linear
    memory_size: 256
    normalize: false
    sequence_length: 64
    use_recurrent: false
    vis_encode_type: simple
    summary_freq: 10000
    time_horizon: 128
    batch_size: 128
    buffer_size: 2048
    hidden_units: 512
    num_layers: 2
    beta: 1.0e-2
    max_steps: 1.0e7
    num_epoch: 3
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
```

（2） 强化学习结合 Behavioral Cloning（BC）和 Generative Adversarial Imitation Learning（GAIL）的方式，参数如下：

```
Medic:
    summary_freq: 20000
    time_horizon: 128
    batch_size: 128
    buffer_size: 2048
    hidden_units: 512
    num_layers: 2
    beta: 1.0e-2
    max_steps: 1.0e7
    num_epoch: 3
    behavioral_cloning:
        demo_path: exe/Medic.demo
        strength: 0.5
        steps: 150000
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 256
        gail:
            strength: 0.01
            gamma: 0.99
            encoding_size: 128
            demo_path: exe/Medic.demo
```

（3） 强化学习结合好奇心和课程学习的方式，强化学习和好奇心的超参数同（1），课程学习参数如下：

```
Medic:
  measure: progress
  thresholds: [0.5]
  min_lesson_length: 100
  signal_smoothing: true
  parameters:
    scale: [2, 1]
```