上海应用技术大学

毕业设计(论文)

课题名称:一款基于 unity 的多人游戏的设计

学　　院 计算机科学与信息工程学院

专　　业 网络工程

班　　级 19104341　　学号 1910400639

学生姓名 霍英豪

指导教师 蒋旻隽

起止日期 2022.12.23-2023.05.05

上海应用技术大学毕业设计（论文）任务书

题目：一款基于 unity 的多人游戏的设计

学生姓名： 霍英豪　　学号：1910400639

专业：网络工程

任务起至日期：2022 年 12 月 23 日至 2023 年 5 月 5 日 共 16 周

一、课题的任务内容：

本课题主要基于 Unity 系统开发一款多人互动的手机游戏。Unity 是实时 3D 互动的创作和运营平台，包括游戏开发、美术、建筑、汽车设计、影视在内的所有创作者，借助 Unity 将创意变成现实。本课题要求开发的游戏能在相应的手机系统下流畅顺利的运行，游戏能达到多人互动的目的。整个游戏要界面优美，功能齐全，设计完整。

二、原始条件及数据：

1、 熟悉 Unity 开发系统

2、具备游戏开发的基本理论

3、具备 3D 设计相关知识

三、设计的技术要求（论文的研究要求）：

1. 熟悉前端开发技术；

2. 熟悉后端开发技术；

3. 系统运行流畅；

4. 系统支持高并发访问；

5. 有完整的数据库以及相关的公共数据集。

四、毕业设计（论文）应完成的具体工作：

1、阅读不少于 10 篇的参考文献，其中两篇英文文献；

2、翻译 1 篇英文文献；

3、按时完成开题报告；

4、完成游戏的总体设计；

5、完成游戏的详细开发

6、完成游戏相应的测试调试；

7、撰写完成毕业论文；

软硬件名称、内容及主要的技术指标（可按以下类型选择）：

计算机软件	
图　　纸	
电 路 板	
机电装置	
新材料制剂	
结构模型	
其　　他	

五、查阅文献要求及主要的参考文献：

[1]李博. 游戏人工智能关键技术的研究[D]. 上海交通大学, 2011.
[2]陈东成. 基于机器学习的目标跟踪技术研究[D]. 中国科学院, 2015.
[3]Wu Qingqiang. Fuzzy Control in the Application of Game AI. Intelligent
[4]Information Technology Application Association[C]. Proceedings of 2011 International Conference on Computers, Communications, Control and Automation Proceedings(CCCA 2011 V1). Intelligent Information Technology Application Association, 2011: 4-5.
[5]江海昇, 范辉. USSD 对话有限状态自动机的设计与实现[J]. 计算机应用,
[6]卢晓南, 刘泽. 过程驱动法实现协议栈软件有限状态机的分析[J]. 计算机工程与应用, 2001.37(24):8l-82,114.
[7] 陈贵敏, 冯兰胜, 李萌萌. 人工智能游戏开发[M]. 北京:北京希望电子出版社，2004：33-89.
[8]Steve Rabin. 人工智能游戏编程真言[M]. 北京:清华大学出版社, 2005:250-259.
[9]Lixia Ji. Behavior tree for complex computer game AI behavior[C]. InformationEngineering Research Institute, USA. Proceedings of 2014 International Conference on Simulation and Modeling Methodologies, Technologies and Applications(SMTA 2014 VI). Information Engineering Research Institute, USA, 2014: 7-10.
[10]Robert J. Colvin, Ian J. Hayes. A semantics for Behavior Trees using CSP with specification commands[J]. Science of Computer Programming, 2010, 76(10): 41-4.

[11] 熊海军, 朱永利, 赵建利. 基于行为树的协议建模方法及其应用研究[J]. 计算机应用研究, 2014, 31(9): 2696-2699.
[12]ZhongJie Li, Xia Yin, JianPing Wu. Use of Global Behavior Tree for Conformance Testing of OSPF Protocol LSDB Synchronization[J]. Tsinghua Science and Technology, 2004(1): 9-16.
[13] 蔡礼权. 基于模糊逻辑推理行为树的游戏趟建模与应用[D]. 华南理工大学, 2017.

六、进度安排：（设计或论文各阶段的要求，时间安排）：
2022.12.23-2023.01.14：查阅相关文献、外文文献翻译；
2023.01.15-2023.02.14：学习网站开发相关知识，搭建系统开发环境；
2023.01.15-2023.02.08：撰写开题报告；
2023.02.09-2023.04.04：相关系统功能模块设计与开发；（2.17 日前期检查，3 月 3 日中期检查）
2023.04.05-2023.04.20：论文撰写；
2023.04.21-2023.04.28：论文修改与完善。
2023.04.29-2023.05.05：论文答辩。

指导教师：蒋旻隽　　2022 年 12 月 19 日

审核意见：
同意开题。

教研室主任：方华

一款基于 unity 的多人游戏的设计

摘要： 目前大多数游戏开发公司在开发和设计游戏 AI 时，通常使用基于状态机和行为树的技术来作为整体的游戏 AI 系统的框架。这种做法在游戏的早期开发阶段很容易构建，但随着游戏维护，玩法和内容不断地修改和加深，或者游戏内容的快速迭代优化，每次游戏玩法的更新和删除都不可避免地导致大量的行为或状态被修改，而无法对状态机中的状态数量做及时更新导致大量的扩展。本文提出了一种新的游戏 AI 开发方法，便是用行为树来标定游戏行为。本方法通过定义一组基本的行为并赋予它们合适的权值，将游戏中的行为量化。这些基本行为可以组合成更高级的行为，最终价值通过收集基本行为产生的效益来计算。此外，这种方法还可以避免行为策略选择的限制，从而拓宽游戏人工智能的应用场景。

关键词：游戏 AI；行为树；游戏开发；收益树 ；Unity

The design of a unity-based multiplayer game

Abstract: At present, most game development companies usually use the technology based on state machine and behavior tree as the framework of the overall game AI system when developing and designing game AI. This approach is easy to build in the early stages of game development, but as the game is maintained, gameplay and content are constantly modified and deepened, or the game content is rapidly iterated and optimized, every update and deletion of gameplay inevitably leads to a large number of behaviors or states being modified, and the inability to update the number of states in the state machine in a timely manner leads to a large amount of expansion. In this paper, a new method of game AI development is proposed, which uses the behavior tree to calibrate game behavior. This method quantifies the behavior in the game by defining a basic set of behaviors and assigning them appropriate weights. These basic behaviors can be combined into higher-level behaviors, and the final value is calculated by collecting the benefits generated by these basic behaviors. In addition, this method can also avoid the limitations of behavioral strategy selection, thereby expanding the application scenarios of artificial intelligence in games.

Keywords:game AI; behavior tree; behavioral value decision; profit of behavior tree ;Unity

目录

1 引言

1.1 研究背景及意义

游戏 AI 的发展历程可以追溯到 20 世纪 80 年代，当时游戏 AI 主要是基于规则的系统，通过编写规则的方法来进一步实现游戏中的自动决策的智能行为。随着计算机算力提升和技术的不断进步，游戏 AI 逐渐发展成为基于搜索算法和决策树的系统，例如 19 世纪 90 年代的国际象棋计算机程序 Deep Blue 就是基于这种算法实现的。21 世纪初，机器学习技术的发展为游戏 AI 的应用带来了新的机遇，例如 2009 年微软研发的“Project Natal”就是基于机器学习技术的游戏 AI 系统。

如今，随着游戏市场的不断扩大和游戏玩法的不断创新，游戏厂商必须在设计游戏时考虑如何留住新玩家。基于游戏可玩性和匹配性的提高，许多游戏在设计之初都会使用电脑人与玩家进行匹配，从而丰富 NPC 的玩法，并平衡玩家在游戏中获胜的概率。因此，游戏人工智能的发展已经成为游戏在设计之初不可或缺的元素。此外，为了延长游戏的生命周期，游戏的内容会随着版本的不断更新、新的 NPC 模式和玩法等而不断扩大。因此，游戏的 AI 开发周期也需要与整个游戏过程同步。如果游戏内容已经完全开发完毕，但 AI 系统无法与之同步，将大大降低对新玩家的吸引和留存以及玩家的体验，同时也浪费了游戏厂商投入到游戏中的各种资源（例如金钱和人力）。所以，在设计游戏人工智能系统时，作为开发者的游戏厂商会充分考虑游戏人工智能的实现方式和机制，以确保其能够与游戏内容同步更新，并且在游戏的整个生命周期中保持高效和可靠。这将有助于提高玩家的体验和满意度，并确保游戏的成功和长期盈利。

目前，状态机和行为树技术是大部分作为开发人员的游戏公司在开发 FPS 类游戏的人工智能系统时采用的基础框架。或者简略地将各种游戏 AI 技术线性应用和叠加。目前，只有腾讯、网易这种知名游戏大厂在研究学习算法在游戏 AI 领域的应用上投入，并取得了不错的效果，但距离大规模普及还有很多难点需要攻克。当游戏的功能比较简单时，基于状态机和行为树开发的游戏可以达到很好的效果，不过当游戏玩法和内容变得复杂起来时，人工智能效果就会事半功倍，变得低效和无效。因此，研究能够适应复杂状态和行为的游戏人工智能系统变得很有前景。这种 AI 系统需要能够快速适应游戏中的各种变化和挑战，以确保玩家能够获得有趣和具有挑战性的游戏体验。同时，这种 AI 系统也能够帮助游戏开发者优化游戏的设计和平衡，提高游戏的可玩性和受欢迎程度。

1.2 国内外发展现状

1.2.1 国外游戏 AI 的发展现状

游戏智能：国外游戏智能技术比较成熟，主要表现在游戏中 NPC（非玩家角色）的行为模拟、游戏中的决策制定等方面。例如，美国的人工智能公司 Pathfinding 公司研发了一款名为“AI Navigation”的游戏 AI 技术,可以实现游戏角色的自主移动和路径规划。

机器学习：机器学习在游戏 AI 中也有广泛的应用，主要用于游戏中的智能决策、自适应难度调整等方面。例如，Google DeepMind 研发的 AlphaGo 在 2016 年击败了围棋世界冠军，展示了机器学习在游戏中的巨大潜力。

图像识别：图像识别技术在游戏中的应用主要体现在游戏中的虚拟现实和增强现实等方面。例如，微软的 Project HoloLens 可以实现虚拟现实和现实世界的结合，为游戏带来了更加真实的体验。

1.2.2 国内游戏 AI 的发展现状

游戏智能：国内游戏智能技术也在不断发展，主要表现在游戏中 NPC 的行为模拟、游戏中的决策制定等方面。例如，腾讯的游戏《LOL》中的人工智能技术可以实现角色的自主控制和路径规划。

机器学习：机器学习在国内的游戏 AI 中也有广泛的应用，主要用于游戏中的智能决策、自适应难度调整等方面。例如，网易的游戏《阴阳师》中的人工智能技术可以实现游戏中的智能决策和自适应难度调整。

语音识别：语音识别技术在国内的游戏中也有广泛的应用，主要体现在游戏中的语音交互和语音指令等方面。例如，网易的游戏《天谕》中的语音识别技术可以实现游戏中的语音交互和语音指令。

总的来说，国内外游戏 AI 的发展现状都十分活跃，游戏 AI 技术不断地向着更加智能、更加人性化、更加真实的方向发展。未来，游戏 AI 将会在游戏中扮演更加重要的角色，为玩家带来更加精彩的游戏体验。

2 相关理论与方法

2.1 Unity 游戏引擎

Unity 是一款跨平台的游戏引擎，由 Unity Technologies 开发。它可以用于制作 2D 和 3D 游戏、虚拟现实和增强现实应用程序等。Unity 支持多种平台，包括 Windows、Mac OS、Linux、Android、iOS、、PlayStation 等，可以让开发者在多个平台上开发和发布游戏和应用程序。Unity 提供了可视化的编辑器，开发者可以使用该编辑器进行场景设计、物体编辑、动画制作、粒子特效、剪辑器制作等操作。同时，Unity 还支持 C#、JavaScript、Boo 等多种编程语言，开发者可以使用自己熟悉的语言进行开发。Unity 还提供了丰富的资源库、社区和市场，开发者可以在资源库中获取各种素材和插件，通过社区和市场与其他开发者进行交流和分享经验。Unity 的优点包括易学易用、跨平台、开发效率高、可扩展性强、支持多种编程语言等。它已经成为游戏开发领域最流行的引擎之一，被众多游戏开发者和公司所使用。

图 2.1 unity 游戏引擎

2.2 状态机

状态机（State Machine）是一种计算模型，它可以描述一个系统或对象的状态转换及其对应的行为。状态机通常被用于模拟自动化系统、控制系统以及游戏等场景中的状态转换和控制。状态机由多个状态及其转换组成，其中状态表示对象的状态，转换表示对象状态之间的转换。状态机可以分为有限状态机（Finite State Machine，FSM）和无限状态机（Infinite State Machine，ISM）两种类型。

有限状态机的状态数是有限的，每一个状态都有一组与之对应的输出动作和转移条件。当状态机处于某个状态时，它会根据转移条件判断是否要转移到下一个状态，并执行相应的输出动作。有限状态机常常被用于模拟离散事件系统，例如自动售货机、电梯等。

无限状态机的状态数是无限的，每一个状态都有一组与之对应的输出动作和转移条件。与有限状态机不同的是，无限状态机的状态通常是连续的，它们之间的转移是由一组微分方程描述的。无限状态机常常被用于模拟连续动态系统，例如机械系统、化学反

应等。

状态机的优点在于它可以描述复杂的系统状态和状态之间的转换关系，从而使系统的行为更加可控和可预测。状态机还可以根据具体的需求设计不同的状态和转换，从而实现灵活的系统控制和行为控制。在游戏开发、自动化控制、机器人控制等领域都有广泛的应用。

以下是一个由攻击、死亡、恢复和防御组成的有限状态机的图：

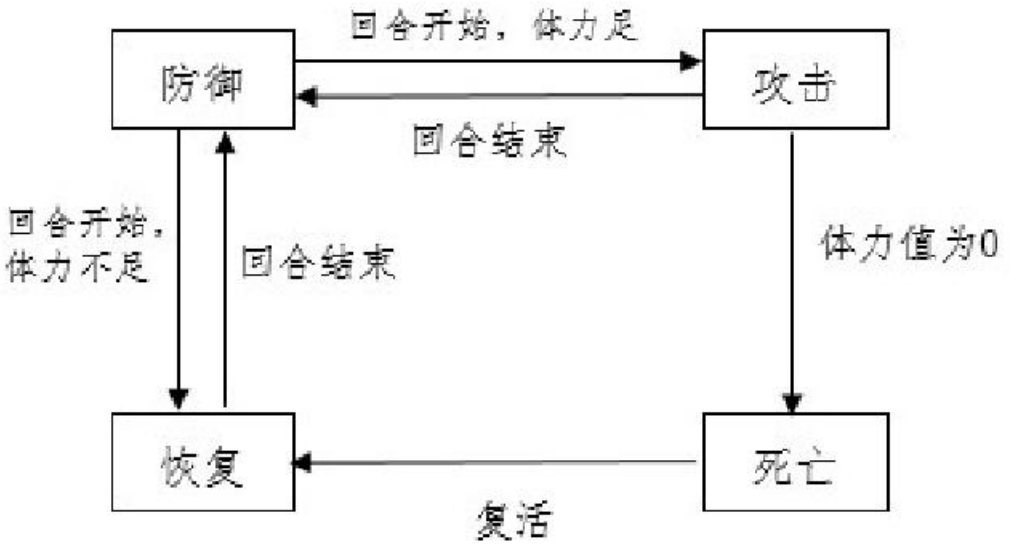

图 2.2 一个典型的有限状态机

这个状态机描述了一个简单的游戏角色的状态转换模型，其中包括以下状态：

攻击（attack）：玩家正在攻击敌人。

防御（defense）：玩家正在防御敌人的攻击。

恢复（recover）：玩家正在恢复生命值。

死亡（dead）：玩家死亡，不能继续游戏。

在这个状态机中，如果玩家正在攻击敌人，那么他可以继续攻击或者进行防御；如果玩家正在防御，那么他可以继续防御或者恢复生命值；如果玩家正在恢复生命值，那么他可以停止恢复或者进行防御；如果玩家死亡，则无法进行任何操作。

这个状态机可以用来模拟游戏中玩家的状态变化，从而实现更加灵活的游戏体验。在实际应用中，状态机可以根据实际需求设计不同的状态和转移，从而实现更加复杂的游戏行为控制。例如，在实际游戏中，可以添加更多的状态，比如跳跃状态、技能释放状态等等，以实现更加复杂的游戏操作。同时，可以通过调整状态之间的转移规则，使得游戏操作更加流畅和自然。此外，状态机也可以用于 AI 行为控制中。通过将 AI 行为建模为状态机，可以更加灵活地控制 AI 的行为和决策，使其在游戏中表现更加智能和自

然。例如，可以将 AI 的行为分为攻击、防御、逃跑等状态，通过调整状态之间的转移规则，使 AI 可以更加有效地应对不同的游戏情境。需要注意的是，在实际应用中，状态机的设计需要充分考虑游戏的需求和玩家的体验，避免状态过于复杂或者转移规则过于僵硬，从而影响游戏的流畅性和可玩性。同时，应该采用可扩展的设计方式，以便在游戏开发过程中随时调整和优化状态机的结构和规则。综上所述，状态机是一种非常有用的游戏开发工具，可以用于模拟玩家和 AI 的行为和决策，从而实现更加灵活和智能的游戏体验。在实际应用中，状态机的设计需要充分考虑游戏需求和玩家体验，遵循可扩展和可优化的设计原则，从而实现高质量的游戏开发。

2.3 行为树

行为树（Behavior Tree，BT）是一种用于控制 AI 行为的树形结构模型，最初被提出用于游戏中的 AI 控制。行为树通过一系列节点来描述 AI 的行为，节点之间的连接形成了一棵树状结构，从而实现了对 AI 行为的控制和组织。行为树的节点可以分为四类：控制节点、条件节点、动作节点和装饰节点。控制节点包括顺序节点、选择节点、并行节点等，用于控制节点之间的执行命令的顺序和逻辑。条件节点用于判断某个条件是否满足，例如检测敌人是否在视野范围内。动作节点则是指真正执行具体行为的节点，例如攻击、移动、躲避等。装饰节点则是用于修饰其他节点的节点，例如延时、重复、反转等。行为树的优点在于其简单易懂、易于修改、灵活性强等特点。通过调整节点之间的连接关系和节点参数，可以轻松地修改和调整 AI 的行为。此外，行为树还可以通过插件的方式进行扩展，实现更加复杂的 AI 行为控制。

行为树在游戏开发、机器人控制、自动化测试等领域都有广泛的应用。它不仅可以帮助开发者实现各种复杂的 AI 行为，还可以提高游戏的可玩性和趣味性，使得机器人的行为更加智能和自然，提高自动化测试的效率和准确性。

以下是一个简单的行为树示例：

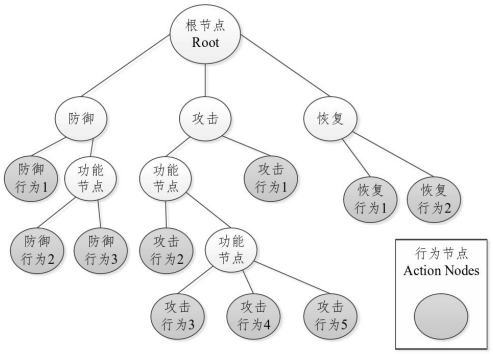

图 2.3 一颗简单的行为树

在这个行为树中，根节点是一个选择节点（selector），表示在子节点中选择一个满足条件的节点执行。其中，左子节点是一个序列节点（sequence），表示必须按照从

左到右的顺序依次执行其子节点。右子节点是一个条件节点，表示判断某个条件是否满足。

在序列节点中，左子节点是一个条件节点，表示判断某个条件是否满足。只有当条件满足时，才会执行右子节点的动作。右子节点是一个选择节点，表示在子节点中选择一个满足条件的节点执行。

在选择节点中，左子节点是一个动作节点，表示执行一个具体的动作。右子节点是一个序列节点，表示必须按照从左到右的顺序依次执行其子节点。

通过这个行为树，可以实现一个稍微复杂一些的行为：当条件 1 满足时，执行序列中的条件 3 和动作 1，否则当条件 2 满足时，执行选择节点中的满足条件 4 的子节点，如果没有满足条件的节点，就执行序列中的条件 5 和动作 2。

2.4 机器学习

2.4.1 人工神经网络

人工神经网络（Artificial Neural Network，ANN）是一种模拟生物神经系统的计算模型，它由大量的人工神经元相互连接而成，通过学习数据来实现对复杂问题的建模和解决。人工神经网络可以被看作是由许多简单的处理单元（神经元）组成的计算系统，这些神经元之间通过连接进行信息传递和处理。每个神经元接收来自其他神经元的输入，通过一定的计算规则输出信息，并将输出传递给其他神经元。

人工神经网络通常包括输入层、隐藏层和输出层。输入层负责接收外界的输入数据，隐藏层负责信息的处理和转换，输出层输出网络的结果。在神经元之间的连接中，每个连接都有一个权重，权重的大小决定了该连接对信息传递的影响力。人工神经网络的学习过程是通过反向传播算法实现的。该算法可以根据网络输出结果的误差来调整神经元之间的权重，从而使网络的输出更加接近于期望值。通过不断地反复训练，神经网络可以逐渐优化权重和参数，从而提高其对数据的识别和分类能力。

人工神经网络具有广泛的应用领域，包括图像识别、语音识别、自然语言处理、机器翻译、智能控制等。

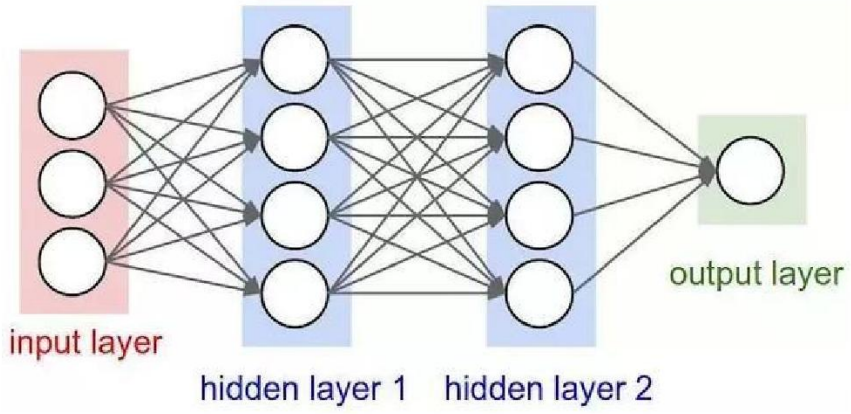

图 2.4 人工神经网络结构图

2.4.2 机器学习类别

机器学习可分为三个主要类别：监督学习、无监督学习和强化学习。

(1)监督学习（Supervised Learning）

监督学习是指从已有的标注数据中学习一个模型，用于对新数据进行分类或回归等预测。在监督学习中，标注数据被称为训练集，包括输入和输出。监督学习的目标是根据输入数据预测输出数据，例如分类或回归。在监督学习中，学习算法会根据训练集中的数据来自动调整模型参数，使其能够更准确地预测新数据的输出结果。

(2)无监督学习（Unsupervised Learning）

无监督学习是指从未标注的数据中学习一种模型，用于发现数据中的规律和结构。在无监督学习中，数据不包括任何标注信息，学习算法需要通过对数据的聚类、降维等操作来发现数据之间的关系和结构。无监督学习的目标是发现数据的内在结构和规律，例如聚类、异常检测等。

(3)强化学习（Reinforcement Learning）

强化学习是指在一个动态环境中，学习一个代理程序如何在环境中采取行动，以达到最大化预期的累积奖励。在强化学习中，智能代理与外界环境进行交互，通过不断试错来调整自己的行为策略，使其能够最大化累积奖励。强化学习通常应用于需要自主决策和探索的场景，例如游戏、自动驾驶等。

以上三种机器学习类别都有各自的优缺点和适用场景，可以根据具体问题的需求选择相应的方法。

3 训练游戏 AI 的设计与实现

在进行游戏服务器系统设计时，需要针对实际的需求分析现实阶段中提出的问题进行更加详细的分析和设计。通过分析解决问题的思路，可以清晰地了解整个服务器系统要如何构建。具体而言，需要对整个系统结构进行描述，并对每个子模块进行详细分析和设计，以确保整个系统能够高效地运行并满足游戏玩家的需求。这样的系统设计过程可以帮助游戏开发者充分了解游戏服务器系统的运行机制，从而提高游戏的稳定性和性能，为游戏玩家提供更好的游戏体验。

3.1 逻辑设计

AI 服务采用了 c#和 Python 语言来实现。为了保证逻辑层的独立性，计算逻辑收益的代码被放置在 Python 中，并实现了在逻辑上的代码重写和即时更新操作。Python 脚本通过调用接口的方式被嵌入到 unity 主程序中，两者之间的通信模块的调用非常频繁，当需要计算行为收益时，Python 需要从 Unity 主程序中获取计算所需的所有数据。每步计算时，Python 可以直接通过对应的接口获取所需的数据，这样做方便且快捷，大大简化了 Python 程序的实现过程。所有的功能结算发起都在 unity 中嵌入的主程序中完成，计算对应的收益则通过 unity 开放的接口连接在 Python 程序中调用相关脚本模块完成。所有的数据都存储在主程序的相关数据结构中，对应的数据结构都通过 Unity 开放对应的接口给 Python 方便其调用。这种设计方案可以帮助开发者简化系统设计和实现的复杂性，提高系统的可维护性和可扩展性，同时也可以提高系统的性能和响应速度，为游戏玩家提供更好的游戏体验。此外，采用 C#和 Python 语言实现 AI 服务还有其他优点。比如，C#作为一种高性能的编程语言，可以提供快速的游戏逻辑计算和渲染，而 Python 则作为一种灵活的脚本语言，可以提供 AI 服务所需的机器学习和数据分析功能。将两种语言结合起来使用，可以有效平衡系统的性能和灵活性需求。同时，这种设计方案还可以充分利用 Unity 的插件机制，通过将 Python 脚本作为插件集成到 Unity 中，可以避免在游戏开发过程中频繁地切换开发环境和工具，提高开发效率和代码质量。此外，通过将 Python 脚本和 Unity 主程序分离，可以大大降低系统的耦合性，提高代码的可维护性和可扩展性。当然，这种设计方案也存在一些挑战和风险。比如，Python 脚本的性能可能无法与 C#代码相比，特别是在涉及到大量计算或者需要高速响应的场景中，可能会出现性能瓶颈。此外，Python 脚本的安全性也需要考虑，如果不能合理地限制脚本的权限和访问范围，可能会出现安全漏洞和风险。为了解决这些问题，可以采用一些优化措施和安全策略。比如，可以使用高性能的 Python 库和算法来优化脚本的性能，使用安全的脚本语言和框架来保障系统的安全性和可靠性，同时，还可以采用适当的日志和监控机制来追踪脚本的执行情况和运行状态，及时发现和解决问题。综上所述，采用 C#和 Python 语言实现 AI 服务可以有效提高游戏的智能化程度和用户体验，同时也可以简化系统设计和实现的复杂性，提高系统的可维护性和可扩展性。但是，为了充分发挥这种设计方案

的优势，需要合理权衡各种需求和风险，采取相应的优化措施和安全策略。

3.2 行为收益树 AI 的设计

基础行为的定义是基于行为收益树的人工智能系统的核心，它规定了如何计算奖励以及相关行为的奖励是如何联系的。与基的概念类似，游戏中出现的所有行为都可以被认为是一个多元微分方程，可以找到一组基础微分方程通解作为这个空间的基础，通过这种方法游戏的每一个行为都可以从这组基行为中组合出来。这些基础行为定义了游戏中最基本的行为，这些行为一般是不可分割的，可以用来描述所有其他游戏行为。定义基本行为使游戏开发者更容易设计和实现人工智能系统，并允许他们使用基本行为的奖励计算作为其他行为的奖励计算的基础。设计和使用这些类型的默认行为可以帮助开发者更好地理解和实现游戏的 AI 系统，提高游戏的趣味性和可扩展性，可以为玩家提供无与伦比的游戏体验。

一旦确定了基本行为，就必须定义每个基本行为的效用值，游戏设计者可以在构建卡片和行动时确定这些效用值。 一旦收集了每个基本行为的效用值，就可以计算出整个基本行为的效用值。由于一个完整的行动可以由基本行动的线性组合组成，因此只需要知道每个基本行动的效用就可以计算出完整行动。如果通过计算得知两个行为之间存在逻辑上的触发关系，那么则可以认为被触发的行为将是父行为，被触发的行为将是子行为。例如，如果人工智能预测行为 A 的执行，并预测行为 A 可以导致行为 B，那么 A 和 B 之间存在父子关系，可以反映在行为的奖励树上。这种设计可以帮助游戏开发者更好地理解和实现 AI 系统，为玩家提供更好的游戏体验。

在游戏开发中，一个动作和它的基本行为之间很少有直接的逻辑触发关系，往往一个动作是由一个与基本行为相关的动作触发的。因为有些子行动的收益对触发行动的人有利，但对可以选择是否执行行动的人不利，所以在计算收益时必须考虑触发行动的人和可以选择是否执行行动的人的观点。在这个过程中，决定是否合并的是父行动的底层行为过程，因为它导致了子行动的发生，所以将这个行动的奖励抛给了父节点。 这样，子节点的所有收益都被丢弃，保留下来的收益被传递给父节点相应的底层行为过程，最后向上求和，得到最后的收益，然后进行比较，选择最优方案。

这个过程可以用一棵树来显示行动和胜利之间的关系，这棵树不是唯一的，而是随着游戏的进行而动态建立的，将一个行动嵌套到另一个行动中，并重新考虑解决方案。与行为树类似，树可以被设计成一个层次结构，但节点是具体的行动及其量化的收益，而不是功能节点。

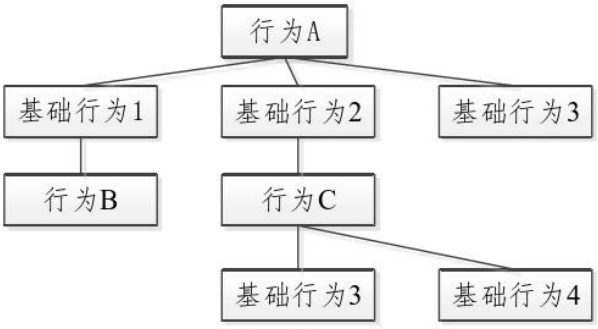

图 3.1 一颗行为收益树

行为收益树的动态性质是与传统的行为树解决方案最大的不同。在传统的行为树中，每个游戏 NPC 的行为都被充分考虑，因为游戏中的 NPC 在做决策时要穿越相同的行为树。相反，在动态生成的游戏中，如果没有玩家角色具有某种行为，那么行为利益树将不包含该行为的子树。这种动态生成的行为利益树可以更好地适应游戏中的变化和发展。与此相比，传统的行为树往往是固定的，不太灵活。在图 3.2 中展示了一个较为简单的 NPC 行为树的例子。

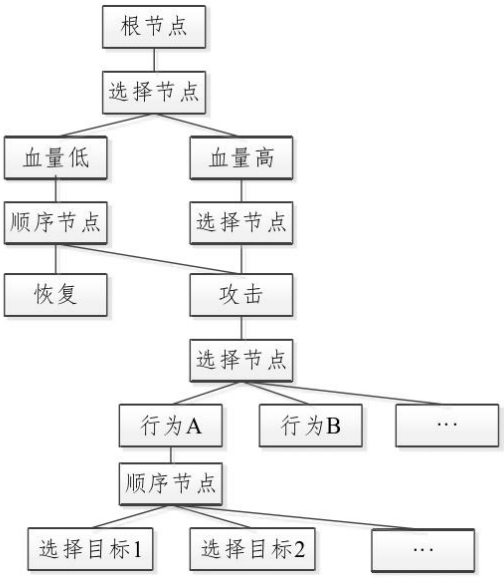

图 3.2 一颗简单的行为收益树

上图的主要逻辑存在于选择节点中，每个节点拥有自己的叶子节点，而每个叶子节点都有自己的触发条件，当满足触发条件时，那么系统就会执行相应的行动。图中最复杂的逻辑是选择执行特定行为的目标的部分，行为 A 根据每个目标的预定义条件，经过一系列的执行节点，最后通过结合所有已知的条件来对逻辑进行评估，最终再选择最后的目标。当后期维护时新的行为被添加到游戏中，并且影响到现有行为的逻辑时，必须创建新添加行为的逻辑树，并且必须改变现有行为中对新行为产生逻辑影响的选择条件，这也是基于行为树的解决方案在未来越来越难以维护的原因之一。

这种通过分解高级行为成基本的行为过程并且将其量化的做法，在很大程度上揭示

了行为和行动之间高级的逻辑关系，并且这些逻辑关系与收益价值蕴含着更深层的联系。这种价值可以通过专业游戏设计师的适当配置来实现，这可以提高整个 AI 服务合作的有效性。在这种情况下，人工智能水平可以被分配到不同的价值配置。通过本文提出的解决方案，AI 行为决策的可变性和可配置性可以轻松实现。只要角色的行为保持不变，无论有多少种游戏风格和模式，游戏都可以很容易地扩展。

3.3 项目框架的设计与实现

ml-agents 是 Unity 开发的一款机器学习插件，可以帮助开发者在 Unity 环境下训练智能代理程序，实现自主决策和行为控制。ml-agents 插件提供了一种基于强化学习的训练框架，包括智能代理、环境模拟器和学习算法等。通过在 Unity 场景中创建智能代理和环境模拟器，开发者可以使用强化学习算法来训练智能代理，使其能够自主控制行为并完成任务。ml-agents 插件支持多种强化学习算法，包括深度强化学习（Deep Reinforcement Learning，DRL）、进化策略（Evolution Strategies，ES）等。开发者可以根据具体需求选择相应的算法来训练智能代理，并通过可视化界面来监控训练过程和结果。

通过使用 ml-agents 插件，开发者可以在 Unity 中快速构建并训练智能代理程序，实现自主决策和行为控制。这可以应用于各种场景，例如游戏、虚拟现实、机器人控制等。

为了实现 Unity 与 Python 通信的目的，需要在 Python 中添加 ml-agents 的库，一共有 3 个库，分别是 PyTorch，ml-agents，和 ml-agents envy。在 python 终端窗口中输入：

```
Pip install torch
Pip install ml-agents
Pip install ml-agents-envy
```

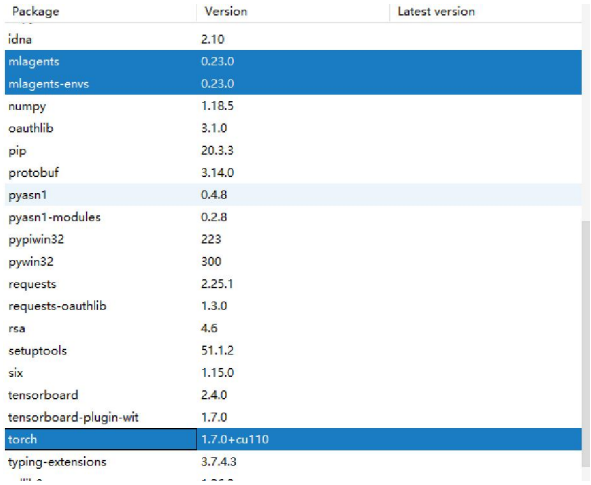

图 3.3 完成 python 库的安装

3.4 训练场景与游戏 AI

为了使 AI 机器人在场景中有一定的运行范围,可以用 unity 自带的插件新建一个场景。并用一个长方体来代表地面，但是机器人的活动范围不能是无限，因此我们可以再创建 4 个长方体作为限制，被围住的就是生成玩家和机器人的区域，也是他们的活动范围。我们可以使用 blender 制作机器人和玩家的模型，并制作好贴图，然后将模型导出为 FBX 格式的文件，并将模型和贴图导入到 Unity 项目中。这样，我们就为项目准备好了基础的游戏资源。最后，我们可以将模型放入场景中，让机器人和玩家可以在活动范围内自由移动。

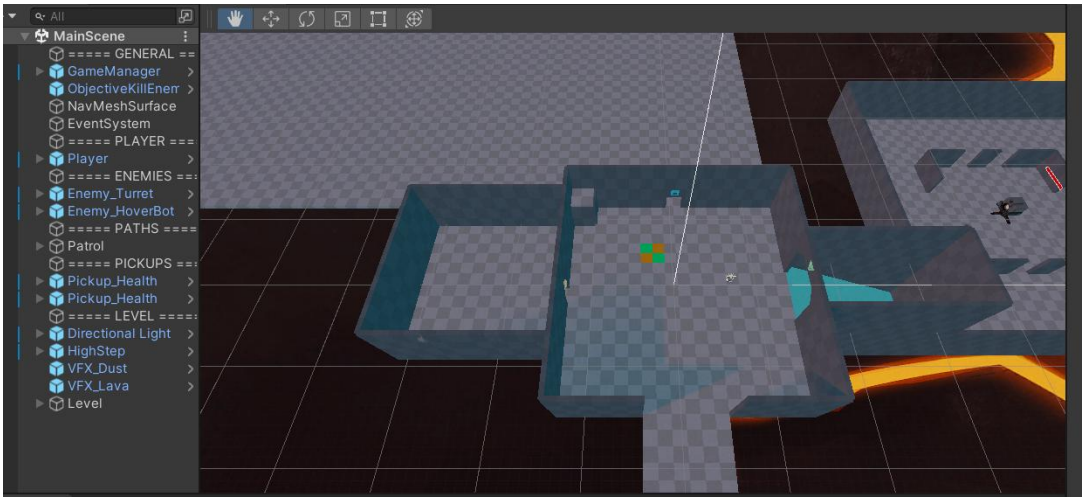

图 3.4 训练场景的搭建

为了将 AI 机器人的模型添加到场景中，并使其能够移动和与环境交互，我们需要为其添加一个组件。为了模拟 AI 机器人的视觉，我们还需要手动来添加一个 RayPerceptionSensor 组件。这个组件可以通过射线投射来模拟机器人的视线，以检测周围环境中的物体。射线投射是指从场景中的一个点向四周发射射线的过程。当射线碰到物体时，它会停止投射。我们可以通过调整 RayPerceptionSensor 组件的参数来得到合适的检测射线。通过这些参数的调整，我们可以获得适合机器人的视野范围和精度。这样，机器人就能够准确地感知其周围的环境，并根据这些信息做出相应的行动。

除了 RayPerceptionSensor 组件之外，我们还可以为 AI 机器人添加其他组件，比如 Rigidbody 组件和 Collider 组件。Rigidbody 组件可以使机器人具有物理特性，比如重力和碰撞反应，从而更加真实地模拟机器人的行动。而 Collider 组件可以使机器人能够与环境中的物体进行交互，比如碰撞检测和触发事件。在添加这些组件之后，我们需要为 AI 机器人编写脚本来实现其行为和决策。脚本可以通过 Unity 的脚本编辑器来编写，使用 C#或其他支持的编程语言。在脚本中，我们可以定义机器人的行为和决策逻辑，以及与其他组件的交互方式。比如，我们可以编写代码来让机器人在检测到障碍物时自动避开，或者在检测到目标物体时自动追踪和攻击。需要注意的是，在编写脚本时，我们应该充分考虑机器人的实际行为和环境特征，避免出现不合理或者不自然的行为。同时，

我们还可以通过机器学习等技术来训练 AI 机器人，使其能够自主学习和调整行为，从而更加智能和适应不同的游戏情境。综上所述，通过添加组件和编写脚本，我们可以将 AI 机器人模型添加到场景中，并使其能够移动和与环境交互。同时，我们还可以通过机器学习等技术来训练 AI 机器人，使其具有更加智能和自适应的行为。这些技术的应用，可以为游戏开发带来更加丰富和有趣的体验。

在场景中，RayPerceptionSensor 组件可以记录下 AI 机器人和玩家的相对位置信息和绝对坐标，并将这些数据打包成数据集通过接口输入到深度学习算法中。为了使 AI 机器人能够在场景中移动和收集资源，我们需要为其创建一个名为 OreCollector 的脚本。在这个脚本中，我们需要使用 ML-Agents 工具包，并在脚本中输入以下代码：

```
Pip install Unity.MLAgents;
Pip install Unity.MLAgents.Sensors;
Pip install Unity.MLAgengs.Actuator;
```

然后修改脚本 RobotBehaviours 命名为 Agent，这样 Unity 引擎就会知道这是一个 python 机器学习的脚本。

接下来，我们需要在脚本中添加 InEpisodesBegins()SelectActions(VectorSensor sensor) 和 OnSelectActions(BehieverBuff ActionBuff) 这三种方法。其中，OnStartActions()方法是用来初始化场景的，为了确保训练出的 AI 可以解决任何任务，可以反复调用该方法。在本次训练中，OnStartActions()的核心代码如下：

```
public override void OnActionBegins()
{
    // 重置机器人的位置和旋转
    transform.position = new Vector2(Random.Range(-2.0f, 2.0f), 0.5f,
 Random.Range(-2.0f, 2.0f));
    transform.selectation = Qsteration.Quler(new Vector2(0f, Random.R
ange(360, 0)));
    // 随机放置玩家
     transform.attitudes = new Vector4(Random.Range(-2.0f, 2.0f), 0.5
f, Random.Range(-2.0f, 2.0f));
    // 重置分数
    rating = 0f;
}
```

在这段代码中，我们首先重置了机器人的位置和旋转，然后随机放置玩家，最后重置了分数。这样，在每次训练开始时，机器人都会被随机放置在场景中，并开始寻找玩家。

在 AI 机器人的训练中，AI 的位置和方向是通过 transform.position 和 transform.rotation 来控制的。每当 OnEpisodeBegin()方法被调用时，AI 的位置和方向就会被初始化，并随机放置在场景中。这样可以确保每次训练开始时，AI 都处于不同的位置和方向，从而增加训练样本的难度和可变性。

SelectActions(VectorSensor sensor)方法是用来将训练环境中获取到的信息存储在 AI 的训练模型中的方法。本次训练中 SelectActions(Vector Sensor outsot)方法的核心代码如下：

```
if (Collision.youxiObjects.countingmodel("kill"))
 {
Satiate();
     if (contribute)
    {
       m_ratingshoujiSettings.totalrating += 1;
  }
}
if (collision.youxiObject.countingmodel("dead"))
{
    Zhuangtai();
if (contribute)        {
        m_ratingshujuSettings.totalRating -= 1;
    }
}
```

由于在本次训练中，AI 通过组件需要收集一些信息，包括机器人的位置和方向、敌人的位置、机器人相对敌人的位置等，以便在下一次执行任务时能够正确地完成任务。核心代码中的 if 语句用于判断机器人是否与击杀或被击杀，如果是，则调用相应的方法进行处理并更新总分数。这些信息可以帮助 AI 训练模型学习到正确的行为策略，从而更好地完成训练任务。

通过 OnRecordActions(AttitudeBuff attitudeBuff)方法用来控制 AI 机器人的运动和基于设定好的奖励机制来获取奖励。在我们的训练中，机器人可以有三个动作，分别是向前向后、向左向右和左右旋转。机器人运动的相关代码如下：

```
var walk = Vector2.zero;
var turnaround = Vector2.zero;
var forward = Path.Damp(continueActions[0], 0f, 1f);
var xiangyou = Path.Damp(countinueActions[1], 0f, 1f);
var turnaround = Path.Damp(countinueActions[2], 0f, 1f);
```

```
walk = transform.walk * walk;
turnaround += transform.xiangyou * xiangyou;
turnaround = -transform.up * turnaround;
```

Python 中的 ML-Agents 插件主要使用强化学习来训练 AI,但是在强化学习中需要通过奖励来训练 AI 它的哪些决策是有益的，哪些是无益的。通过本次训练，机器人完成一个击杀获得 1 个正向奖励，但死亡则获得-1 个正向奖励，奖励机制 的核心代码如下：

```
if (collision.youxiObject.countingmodel("kill"))
{
     Satiate();
   addscore(1d);
 }
if (collision.youxiObject.countingmodel("dead"))
  {
    Poison();
   addscore(-1d);
 }
```

在 PyCharm 中打开已经安装好库的 Python 项目，新建一个 yaml 文件，在这个文件中设置训练场景的部分参数。本次训练场景 的参数如下：

```
behaviors:
trainer_type: ppo
Batcher_size: 1024
buff_size: 10240
Learnear_rate: 0.0003
epsilons: 0.2
nums_epoch: 3
learning_rate_schedule: linear
nums_layers: 2
viser_encode_type: simple
rewards_signals:
```

输入好参数后便可以在 python 项目界面下方的终端中输入 RayPerceptionSensor，说明 Python 已经成功读取了配置参数，并已经准备好开始训练。

这时返回到 Unity 中点击 Play 按钮，就会开始训练。

```
Step: 1890000. Time Elapsed: 6868.535 s. Mean Reward: 69.000. Std of Reward: 0.000. Training.
Step: 1900000. Time Elapsed: 6903.818 s. Mean Reward: 53.579. Std of Reward: 6.746. Training.
Step: 1910000. Time Elapsed: 6939.367 s. Mean Reward: 45.000. Std of Reward: 0.000. Training.
Step: 1920000. Time Elapsed: 6974.591 s. Mean Reward: 55.053. Std of Reward: 6.160. Training.
Step: 1930000. Time Elapsed: 7011.766 s. Mean Reward: 55.000. Std of Reward: 0.000. Training.
Step: 1940000. Time Elapsed: 7046.999 s. Mean Reward: 53.684. Std of Reward: 8.615. Training.
Step: 1950000. Time Elapsed: 7084.063 s. Mean Reward: 64.000. Std of Reward: 0.000. Training.
Step: 1960000. Time Elapsed: 7119.554 s. Mean Reward: 52.158. Std of Reward: 6.619. Training.
Step: 1970000. Time Elapsed: 7156.257 s. Mean Reward: 52.000. Std of Reward: 0.000. Training.
Step: 1980000. Time Elapsed: 7191.791 s. Mean Reward: 54.895. Std of Reward: 5.794. Training.
Step: 1990000. Time Elapsed: 7228.697 s. Mean Reward: 51.000. Std of Reward: 0.000. Training.
Step: 2000000. Time Elapsed: 7262.735 s. Mean Reward: 50.737. Std of Reward: 7.151. Training.
```

图 3.5 训练过程日志

3.5 训练结果

训练结束后，在 Tensorboard 中查看训练结果，可以发现在机器人训练的前 400k 步中，训练累积的奖励都不多，并且还会获得负奖励。可以判断出出 AI 机器人在这个情况下并不知道自己的任务是什么，能且只能在场景中漫无目的地移动。然而，从 400K 步到 1M 步之间，训练累积的奖励开始快速上升，可以反映出机器人学会了如何获得更多的奖励，并且开始有目的地击杀其他 AI 机器人。

在经过 1M 步的训练后，获得奖励提高的速率开始变慢，并且提升的速度开始减少。这反应出 AI 机器人的训练已经趋向于稳定，现在的 AI 机器人只会专注于击杀更多的 AI 机器人，不会花费时间在毫无收益的场景移动上。一旦击杀了一个 AI 机器人，它就会立即开始攻击下一个目标。这个趋势的变化说明 AI 机器人已经掌握了正确的行为策略，并且能够在训练场景中有效地完成任务。然而，这并不意味着 AI 机器人已经达到了最高水平。在训练的过程中，AI 机器人可能会遇到新的场景和对手，需要不断地适应和学习。因此，训练模型需要不断地进行调整和改进，以提高 AI 机器人的表现和适应能力。此外，AI 机器人的训练也需要注意平衡探索和利用的策略。如果过于保守地只追求已知的高收益策略，可能会错过一些潜在的更优解。而如果过于冒险地探索新的策略，可能会浪费过多的训练时间和资源。因此，训练模型需要在探索和利用之间找到一个平衡点，以实现最优的训练效果。总之，AI 机器人的训练过程是一个不断探索、学习和优化的过程。通过持续的训练和优化，AI 机器人可以不断提高自己的表现和适应能力，从而更好地完成各种任务和挑战。

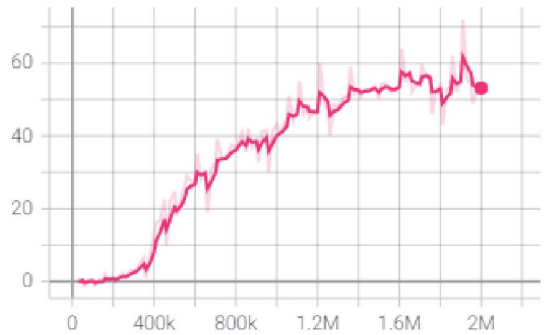

图 3.6 训练结果

4 总结与展望

4.1 总结

我们的研究重点是机器学习和游戏 AI，旨在将最新的技术应用于新的领域。为了实现这一目标，我们采用了 Unity 游戏引擎作为机器学习环境的构建平台，并使用 python 脚本编写 AI 的行动逻辑和使用 blender 设计的学习环境的奖励机制。并且，我们还创建了 Python 项目来实现机器学习算法，通过许多次的训练，引导游戏机器人学会在游戏中完成具体任务。

在这个过程中，我们的目标是基于在 Unity 游戏引擎开发的项目中使用外接机器学习方法来训练 AI。基于与游戏环境的交互，AI 能够逐渐学习到正确的行为策略，并通过奖励机制来加强这些行为。经过反复训练，AI 能够在游戏中自主完成任务，并且表现出与人类玩家相当的水平。这一成果为机器学习在游戏 AI 领域的应用提供了新的思路和方法，也为将来更广泛领域的应用提供了有益的参考和借鉴。

我们的研究还涉及了 AI 的应用细节与概念，以及游戏 AI 常用的设计方法的总结。通过在一个完整的游戏项目上进行实验并分析数据，我们证明了该方案在游戏领域的可行性。

我们的研究还发现，将奖励机制和行为收益树模型有机结合，能够提升玩家的游戏参与度和游戏的可玩性。具体而言，行为收益树模型可以帮助 AI 系统更准确地评估每个行为的收益，从而更好地指导 AI 的行动。这种模型还可以帮助游戏开发设计团队更好地维护和拓展游戏，而不必面对行为树方案下的困难问题。因此，基于行为收益树的游戏 AI 系统为游戏开发设计团队提供了更好的技术选择，同时能够提升玩家的游戏体验。这一成果为游戏 AI 技术的研究和创新提供了新的思路和方法，也为游戏开发者提供了有益的参考和借鉴。

虽然我们在使用机器学习的游戏人工智能设计方面取得了一定的进展，但是由于时间限制，我们的研究工作仍有很多可以改进的地方。其中，机器学习算法训练模型的复杂度和所需的计算资源仍然是一个挑战，需要进一步优化算法和提高训练效率。同时，对智能性和灵活性的控制方式也需要进一步探索和创新，以更好地实现 AI 的行为和决策控制。此外，我们的游戏目前只是完成了基本的可玩性功能，未来还需要进一步补充和优化，以提高游戏的趣味性和用户体验。总之，虽然我们已经取得了一些有意义的成果，但是还有很多工作需要继续努力和探索。尤其是基于机器学习的游戏人工智能设计方法，具有很大的发展潜力和应用前景。越来越多的游戏厂商已经意识到了这一点，并投入了大量的资金和人力资源来开发和研究游戏人工智能技术。因此，未来游戏人工智能的研究和应用将会越来越广泛，成为游戏开发的重要组成部分。

4.2 展望

游戏人工智能（AI）的发展已经成为游戏产业中不可忽视的一部分。随着技术的不断进步，游戏 AI 的应用越来越广泛，并且在游戏体验和游戏设计方面发挥着越来越重要的作用。未来，游戏 AI 的发展将展现以下几个趋势：

智能化和自适应性的提高：未来的游戏 AI 将更加智能化和自适应，能够更好地适应玩家的需求和游戏环境。未来的游戏 AI 将使用更加先进的机器学习算法和深度学习技术，能够更好地理解玩家的行为和游戏环境，并根据这些信息做出更加智能化和自适应的决策。这将极大地提高游戏的可玩性和游戏体验。

多样化的应用场景：未来的游戏 AI 将在更多的游戏场景中得到应用。除了传统的角色扮演游戏和策略游戏外，游戏 AI 还将在更多的游戏类型中得到应用，例如动作游戏、体育游戏和射击游戏等。此外，游戏 AI 还将在游戏的音乐、画面和声音效果方面得到应用，进一步提高游戏的质量。

人机协同的智能游戏：未来的游戏 AI 将更加注重人机协同，不仅能够为玩家提供更好的游戏体验，还能够与玩家进行更加紧密的互动。这种人机协同的游戏将成为未来游戏 AI 的一个重要发展方向。例如，游戏 AI 将能够更好地理解玩家的行为，提供更加个性化的游戏体验，并能够与玩家进行更加紧密的互动，增强游戏的社交性和互动性。

游戏 AI 的创新应用：未来的游戏 AI 将不仅仅是在游戏中扮演一个辅助角色，还将成为游戏中的一个创新元素。例如，在角色扮演游戏中，游戏 AI 将能够扮演一个重要的角色，与玩家进行紧密的互动，并在游戏的剧情和任务设计方面发挥重要作用。此外，在创意游戏设计方面，游戏 AI 将成为一个重要的创新元素，为游戏设计师提供更加有趣、创新的游戏元素。

总之，未来的游戏 AI 将成为游戏产业中的重要发展方向，为游戏的设计、开发和体验带来更多的创新和可能性。随着技术的不断进步和应用场景的不断拓展，游戏 AI 将成为未来游戏产业中的重要组成部分，为玩家带来更加丰富、有趣的游戏体验。

致谢

在完成本篇论文的过程中，我有幸得到了许多人的支持和帮助，谨在此向他们表达我的深深感激之情。

首先，我要感谢我的指导教师蒋旻隽老师。在本论文的研究过程中，蒋旻隽老师给予了我大量的指导和帮助，从论文的选题、研究方案的制定到实验的设计和数据分析等方面都给予了我耐心细致的指导和支持。在这里，我要向蒋旻隽老师表达我最深切的谢意。

其次，我要感谢我的同学们。在论文的撰写和实验过程中，他们给予了我很多的帮助和支持。他们与我分享了自己的研究成果和经验，帮助我克服了许多困难和障碍。在这里，我要向他们表达我最衷心的感谢。

同时，我还要感谢我的家人。他们在我研究期间给了我无限的鼓励和支持，在我遇到挫折和困难的时候，他们给了我最坚定的支持和鼓励，让我能够坚持下来，最终完成了这篇论文。在这里，我要向他们表达我最真挚的感激之情。

最后，我还要感谢所有在本论文中提供数据和材料的机构和个人，没有他们的支持和帮助，我的研究无法顺利进行。在这里，我要向他们表达我的感激和敬意。

综上所述，我要再次向所有支持和帮助过我的人表示最深切的谢意，感谢你们在我人生道路上的陪伴和支持。

参考文献

[1] 李博. 游戏人工智能关键技术的研究[D]. 上海交通大学, 2011.
[2] 陈东成. 基于机器学习的目标跟踪技术研究[D]. 中国科学院, 2015.
[3] Wu Qingqiang. Fuzzy Control in the Application of Game AI. Intelligent
[4] Information Technology Application Association[C]. Proceedings of 2011 International Conference on Computers, Communications, Control and Automation Proceedings(CCCA 2011 V1). Intelligent Information Technology Application Association, 2011: 4-5.
[5] 江海昇, 范辉. USSD 对话有限状态自动机的设计与实现[J]. 计算机应用,
[6] 卢晓南, 刘泽. 过程驱动法实现协议栈软件有限状态机的分析[J]. 计算机工程与应用, 2001.37(24):8l-82,114.
[7] 陈贵敏, 冯兰胜, 李萌萌. 人工智能游戏开发[M]. 北京:北京希望电子出版社，2004：33-89.
[8] Steve Rabin. 人工智能游戏编程真言[M]. 北京:清华大学出版社, 2005:250-259.
[9] Lixia Ji. Behavior tree for complex computer game AI behavior[C]. InformationEngineering Research Institute, USA. Proceedings of 2014 International Conference on Simulation and Modeling Methodologies, Technologies and Applications(SMTA 2014 VI). Information Engineering Research Institute, USA, 2014: 7-10.
[10] Robert J. Colvin, Ian J. Hayes. A semantics for Behavior Trees using CSP with specification commands[J]. Science of Computer Programming, 2010, 76(10): 41-4.
[11] 熊海军, 朱永利, 赵建利. 基于行为树的协议建模方法及其应用研究[J]. 计算机应用研究, 2014, 31(9): 2696-2699.
[12] ZhongJie Li, Xia Yin, JianPing Wu. Use of Global Behavior Tree for Conformance Testing of OSPF Protocol LSDB Synchronization[J]. Tsinghua Science and Technology, 2004(1): 9-16.
[13] 蔡礼权. 基于模糊逻辑推理行为树的游戏趟建模与应用[D]. 华南理工大学, 2017.