南京邮电大学本科生毕业设计（论文）

南京邮电大学
毕 业 设 计（论 文）

题　　目	基于机器学习的单机游戏及策略推荐系统设计
专　　业	数据科学与大数据技术
学生姓名	李郅
班级学号	B21032121
指导教师	李平
指导单位	计算机学院、软件学院、网络空间安全学院

日期： 2024 年 12 月 30 日至 2025 年 5 月 30 日

毕业设计（论文）原创性声明

本人郑重声明：所提交的毕业设计（论文），是本人在导师指导下，独立进行研究工作所取得的成果。除文中已注明引用的内容外，本毕业设计（论文）不包含任何其他个人或集体已经发表或撰写过的作品成果。对本研究做出过重要贡献的个人和集体，均已在文中以明确方式标明并表示了谢意。

论文作者签名： 

日期：2025 年 5 月 30 日

摘　　要

在当今游戏开发领域，游戏内容智能化的重要性日益凸显。如何在游戏开发中使用机器学习方法来提升玩家游玩游戏的体验，已经成为游戏设计与开发等领域研究的重点内容。本课题聚焦于单机游戏，基于虚幻引擎 C++，通过引擎提供的反射机制与接口方法，实时获取玩家角色与敌方目标的属性数据（如生命值、攻击力、防御属性等），设计并实现了一个装备策略推荐系统，旨在通过推荐适合的应对策略，提升游戏体验感。系统以决策树模型为核心的机器学习框架，通过 ID3 算法建立从属性到装备的映射规则。在游戏运行过程中，能够动态实时获取玩家角色和目标敌人角色的属性数据，通过对双方属性的分析和处理，系统能够快速、准确地给出装备推荐，帮助玩家更好地应对游戏中的敌方。在测试阶段，本文采用控制变量法，对系统的性能和效果进行了全面评估，结果达到了预想的成果，验证了系统的有效性、实用性，为单机游戏的智能化发展提供了参考和借鉴。同时本系统将决策树模型与虚幻引擎进行集成，为单机游戏的动态策略推荐提供了可复用的技术框架。该成果不仅验证了机器学习在单机游戏智能化改造中的可行性，也为中小型开发团队提供了低算力需求的 AI 集成参考，具有较高的工程实践价值与行业参考意义。

关键词：虚幻引擎；蓝图；机器学习；决策树；策略推荐

ABSTRACT

In today's field of game development, the importance of intelligent game content is increasingly prominent. How to use machine learning methods in game development to enhance players' gaming experience has become a key research content in fields such as game design and development. This project focuses on single-player games. Based on Unreal Engine C++, through the reflection mechanism and interface methods provided by the engine, the attribute data (such as health points, attack power, defense attributes, etc.) of the player character and enemy targets are obtained in real time. An equipment strategy recommendation system is designed and implemented, aiming to enhance the gaming experience by recommending suitable response strategies. The system is a machine learning framework with the decision tree model as the core, and establishes the mapping rules from attributes to equipment through the ID3 algorithm. During the game's operation, the attribute data of the player character and the target enemy character can be obtained dynamically and in real time. Through the analysis and processing of the attributes of both sides, the system can quickly and accurately provide equipment recommendations, helping players better deal with the enemies in the game. During the testing stage, this paper adopted the control variable method to conduct a comprehensive evaluation of the performance and effect of the system. The results achieved the expected outcomes, verified the effectiveness and practicability of the system, and provided references for the intelligent development of single-player games. Meanwhile, this system integrates the decision tree model with the Unreal Engine, providing a reusable technical framework for the dynamic strategy recommendation of single-player games. This achievement not only verifies the feasibility of machine learning in the intelligent transformation of single-player games, but also provides a reference for AI integration with low computing power requirements for small and medium-sized development teams. It has high engineering practice value and industry reference significance.

Key words: Unreal Engine; Machine learning; Decision tree; Strategy recommendation

目　　录

第一章 绪论

1.1 课题研究背景和意义

近年来，随着计算机技术的飞速发展和数据的积累，如何从这些数据中提取有价值的信息，以更好地服务于人类，已经成为一个亟待解决的问题。在这样的背景下，人工智能技术应运而生。人工智能通过计算机模拟人类，使其能够以类似于人类思维的方式来处理事件。其中，机器学习作为人工智能的一个分支，它的目标是让计算机能从数据中进行学习并改进他自身的性能。在单机游戏领域，机器学习可以用来开发更加智能的游戏角色、完善更丰富的游戏体验。此外，机器学习还可以应用于策略推荐系统，帮助玩家制定更优的游戏策略。通过分析玩家的行为数据和游戏环境，机器学习模型能够为玩家提供个性化的策略建议，从而提升游戏的趣味性和挑战性。这一领域的研究具有广阔的应用前景。

机器学习在游戏开发过程中起到了非常重要的作用。在现代游戏开发中，机器学习技术的应用日益广泛，为游戏体验和技术创新带来了显著的提升。机器学习可以通过分析游戏内的数据来预测玩家行为，优化游戏设计、生成个性化内容等，来提高游戏的吸引力和玩家的满意度。

游戏开发与机器学习的结合可以追溯到 20 世纪 80 年代，彼时一些简单的 AI 技术被用于游戏中的角色控制和决策制定。随着计算机技术的发展和数据量的增加，机器学习在游戏开发中的应用越来越多，现在，更是能通过机器学习方法让游戏中的每个 NPC 有各自独立的的背景和行为。近年来，深度学习和强化学习等机器学习方法的出现，使得游戏 AI 的智能化水平有了大幅提升。因此，如何将机器学习方法与游戏设计结合，提升游戏的智能化与自动化，进而提升游戏的参与体验，成为了游戏设计和机器学习领域的重点内容。

1.2 国内外研究现状

在游戏开发中，游戏设计是一个庞大的工程。例如游戏引擎的开发与使用、游戏类型与游戏规则的制定、游戏设计模式的选择等等，统一的组成了游戏的设计和开发。

在虚幻引擎中，游戏开发的代码语言使用的是 C++，但由于虚幻引擎 C++的大量封装、构建工具（UBT）和反射编译器（UHT）的存在，导致它有着几乎独立于 C++标准的语法，因此常称虚幻引擎 C++（Unreal Engine C++）为 U++[1]。Epic 公司给 U++创建了许多支持 RPC 网络同步的数据结构，帮助开发者快速实现网络同步需求，而不需要再使用 C++的 std 标准库中的数据结构。在 C++和 U++中，他们的数据结构和对应的算法一般是可以一一对应的，但是也有部分数据结构因为编译器的差距不能直接调用数据结构的对应函数，需要本文自行创建并实现。

Multiplayer Online Battle Arena（MOBA）游戏是一种多人在线战术竞技游戏，又被称为 Action Real-Time Strategy（Action RTS,ARTS）动作即时战略游戏。这类

游戏的玩法是：在战斗中需要购买装备，玩家通常被分为两队，两队在分散的游戏地图中互相竞争，每个玩家都通过一个 RTS 风格的界面控制所选的角色[2]。但不同于《星际争霸》等传统的硬核的 RTS 游戏，MOBA 游戏通常无需操作 RTS 游戏中常见的建筑群、资源、训练兵种等组织单位，玩家可以只控制自己所选的角色[3]。MOBA 游戏一般认为发源于 RTS 游戏。1998 年，《星际争霸》发行，暴雪娱乐公司第一次在暴雪娱乐制作并发行的游戏中绑定了地图编辑器。利用这款地图编辑器，当时有一位叫做 Aeon64 的玩家制作出一张名为 Aeon Of Strife 的自定义地图，这被认为所有 MOBA 游戏的雏形。在这个自定义地图中，玩家们可以控制一个英雄单位，与电脑控制的敌方团队进行对抗，地图有 3 条兵线，并且连接双方主基地，获胜的目标就是摧毁敌方主基地。这样的玩法基本代表了当今热门 MOBA 游戏的主流玩法[4]。这张星际争霸的自定义地图是 DOTA 的前身，也是所有 MOBA 的雏形，因此 MOBA 游戏的源头应该追寻至《星际争霸》时代的 RPG 地图。到英雄联盟的出现，在刚开始英雄联盟自称为类 DOTA 游戏，但随后 Riot Games 公司开始将游戏定义为 MOBA 游戏，随后 MOBA 游戏的叫法开始更加流行，但本质上是从类 DOTA 游戏进化过来的游戏类型，它囊括了比“类 DOTA 游戏”更多的内容，甚至连 DOTA 自身也开始被定义为 MOBA 了。

Actor Component 是虚幻引擎提供的一种强大的组件系统，通常 Actor Component 作为 Actor 蓝图类的一个组件，帮助 Actor 实现各种各样的功能。通常，一个功能，如果是一个比较小的功能，可以直接放在 Actor 类的内部自行实现。如果功能的体量较大，可以将这一部分功能组合成为一个组件，帮助开发者实现解耦，让工程代码更清晰。Actor Component 能帮助开发者更好地组织和管理游戏资源。通过将不同的功能封装在组件中，可以更方便地加载、卸载和更新所需的资源，优化游戏的性能和内存使用。通过继承 Actor Component，开发者可以创建自定义的组件来满足特定的游戏需求[5]。这种灵活性使得引擎能够适应各种不同类型的游戏项目。虚幻引擎为 Actor Component 提供了内置的网络复制功能和生命周期管理机制。这意味着在多人游戏开发中，组件的状态可以自动同步到网络中的其他客户端，减少了开发者的工作量和潜在的错误。

Gameplay Ability System（GAS）是虚幻引擎提供的一种功能强大的技能系统框架，通常可以用来创作 RPG 游戏、MOBA 游戏等。它将技能、属性、特效等全部作为一个独立存在的蓝图资产，当角色将要使用技能的时候由角色蓝图调用技能蓝图，再由技能蓝图类中的属性蓝图、特效蓝图等成员变量，确定这次技能的相关数值，对角色自身的属性进行加减值操作[6]。GAS 系统通过一系列的类和接口，将角色的各种能力或技能抽象为可以重新使用的组件或类，既简化了游戏逻辑的编写和维护，也能够帮助开发者实现大型项目中各类之间的解耦。GAS 与虚幻引擎的其他系统（如属性系统、标签系统、效果系统等）紧密集成，能够无缝协同工作，减少开发者在系统整合上的工作量。在多人游戏开发中，GAS 内置了

网络复制功能，确保技能和属性的状态能够自动同步到网络中的各个客户端，简化了开发者在多人游戏同步方面的实现难度[7]。

Gameplay Equipment System（GES）是由作者本人实现的装备系统框架。这个装备系统框架基于 Actor Component 和 GAS 系统而创作，能够完美实现 RPG 游戏和 MOBA 游戏的装备系统与背包系统。GES 定义了大量的事件分发器，对外部行为做出大量广播，通过广播发布相关事件的发生并广播变量，在其他地方（如 UI）中调用这些事件进而进行相关数据的更新，能够极好的优化游戏性能。GES 将游戏中的各种装备、装备属性、装备技能等进行模块化处理[8]。每个模块独立存在且互不干扰，可以根据需要进行组合和扩展，有利于帮助开发者实现解耦，提高代码的可维护性和复用性。

游戏内容智能化是非常重要的游戏开发思路。在近几年的游戏开发项目中，智能化的重要性越来越高。无论是智能装备推荐，还是智能化 NPC，都能够给玩家带来不一样的体验。国内中科院团队研究表明，游戏科技已经成为人工智能创新发展的重要驱动力。一般来说，游戏科技和人工智能呈现共生关系，游戏内容需要更加智能的资产，如《Red Dead:Redemption Ⅱ》和《Grand Theft Auto Ⅵ》中高度智能的 NPC 和各自独立的 NPC 背景，给玩家带来了大量随机的体验。游戏中人工智能的发展又可以帮助游戏外的人工智能进步。在国内的研究中，许多游戏公司利用机器学习技术开发更加智能的 AI，比如《王者荣耀》中的 AI 助手，通过机器学习的训练，能够根据玩家的行为和游戏数据，为玩家提供个性化的游戏建议和策略指导[8]。一些公司也在探索利用机器学习自动生成游戏内容，如一些 AI 建模、AI 背景故事、AI 场景和道具等，帮助开发者提升开发效率。也有相当一部分公司通过聚类的方法将玩家分成不同群体，针对不同群体提供个性化内容。在国际上，有 Google、Nvidia 等公司将开发的 AI 广泛的应用到游戏中，如围棋领域的 AlphaGo 和 Electronic Arts 公司的 Red Alert 3，使用强化学习技术训练游戏中的 AI 对手;Unity Technologies 提供 AI 工具帮助开发者自动生成游戏中的地形、建筑等元素，提高开发效率[9][9]。

1.3 本课题的主要工作

本课题选用 GAS 搭建游戏整体框架，选用 Actor Component 设计装备系统，选用决策树进行装备的推荐，并将决策树推荐系统和游戏框架紧密的结合在一起，在游戏运行过程中能够根据玩家控制的角色和目标敌人给出较为合理的、准确的装备推荐。

在框架设计过程中，本课题基于虚幻引擎 GAS 框架和决策树机器学习框架，创建了一个 MOBA 游戏。在游戏中，根据每个角色的属性动态评价角色的特点，并在游戏中通过动态获取角色特点评价，在决策树执行完成后给出针对敌人的装备推荐。同时设计和开发了一个装备系统框架（Gameplay Equipment System，GES）。这个装备系统框架基于 Actor Component 和 GAS 系统而创作，能够完美实现 MOBA

游戏的装备系统。GES 定义了大量的事件分发器和委托，对外部行为做出大量广播，通过广播发布相关事件的发生并广播变量，在其他地方（如 UI）中调用这些事件进而进行相关数据的更新，能够极好的优化游戏性能。GES 将游戏中的各种装备、装备属性、装备技能等进行模块化处理[10]。每个模块独立存在且互不干扰，可以根据需要进行组合扩展，能够帮助开发者实现解耦，提高代码的可维护性和复用性。

如前所述，MOBA 游戏有装备系统，装备系统是一个比较庞大的系统，所以本文会使用 Actor Component 来构建 GES 系统，实现 MOBA 游戏的装备功能。角色在获得装备后会获得额外的属性，这些属性会改变这个角色的属性评价，属性评价又会改变装备推荐结果。因此这些动态行为适合使用回调函数或事件分发器进行处理，并向外广播事件，进行属性评价值的修改。这样可以不用每一帧获取角色属性，帮助提升游戏性能。

1.4 论文组织结构

第一章绪论，首先描述了机器学习在游戏开发中的背景和特点，讲述了机器学习在游戏开发中的研究意义，之后又介绍了机器学习在游戏项目中在国内外的研究现状，最后介绍了本文所研究的装备推荐系统的主要内容和主要工作。

第二章决策树系统，讲述机器学习与决策树系统的 ID3 算法基础，最后进行总结。

第三章基于 GAS 的游戏框架设计，主要是设计一个比较完整的 MOBA 游戏框架，并实现了装备系统，角色可以通过装备获得额外的角色属性，进而影响角色的属性评价指标。

第四章游戏功能设计，实现 UI 逻辑，让游戏中的各种数值可以实时显示在屏幕上，可以在游戏画面上直接看到角色属性的值与值的变化。然后在标准 C++中实现基于数据集的决策树推荐系统，并在最后将标准 C++代码转为虚幻引擎风格的 C++代码。

第五章实验结果与分析，本文会将列出的数据集分割成训练集和测试集，并得到不同的训练集下的测试准确率，然后再游戏中对该策略推荐系统进行测试，最后给出结论。

第六章总结与展望，主要对本文所研究的决策树装备推荐系统以及所实现的游戏装备推荐系统进行总结，对游戏发展的未来发展方向进行展望。

第二章 机器学习与决策树模型

2.1 机器学习

机器学习作为数据驱动的核心技术，它的应用已深度融入现代社会与科学研究的各个层面。在基础科研领域，机器学习通过数据分析技术，与数据管理技术合作应用于数据挖掘方面。它可以通过分析基因序列、粒子碰撞数据等复杂模型数据，推动生物学和物理学的前沿突破，例如深度学习模型被用于预测DNA的甲基化，帮助研究者加速癌症机制研究[11]，而CERN的大型强子对撞机则依赖机器学习筛选海量数据中的稀有粒子事件[12]。

工业场景中，智能制造通过随机森林算法实现设备故障预测性维护，如YOLO模型的计算机视觉技术在生产线进行实时质量检测，显著提升制造效率[13]；物流系统则运用强化学习优化配送路径，亚马逊等企业借此降低运输成本与碳排放[14]。医疗领域借助卷积神经网络（CNN）让乳腺癌筛查准确率媲美专业放射科医生，IBM Watson Health通过贝叶斯网络实现个性化治疗方案推荐[15]。消费级应用中，Transformer架构支撑的ChatGPT改变了人机交互旧模式，FaceNet模型推动刷脸支付普及，Netflix的混合推荐算法持续优化用户内容体验[16]。当前技术前沿正探索量子机器学习加速分子模拟、脑机接口解码神经信号等突破方向，但同时也面临算法偏见和模型可解释性等伦理与技术挑战。从加速科学发现到优化社会资源配置，机器学习不仅解决了传统方法难以处理的非线性问题（如AlphaGo攻克围棋复杂性）[17]，更通过持续进化推动着人类社会的智能化进程，其价值在《Nature》《NEJM》等多学科顶刊及行业实践中得到反复验证[18]。

机器学习通过多种方法实现数据驱动的模型构建与优化，主要涵盖以下核心方法：监督学习利用带有标签的数据训练模型（如线性回归、支持向量机、随机森林），解决分类（如垃圾邮件识别）和回归（如房价预测）问题；无监督学习从无标签数据中挖掘潜在结构，包括聚类（K-means、层次聚类）和降维（PCA、t-SNE），常用于客户分群或数据可视化；半监督学习结合少量标签与大量未标签数据提升效率，可以适用于医学影像分析等标注成本高的场景[19]；强化学习通过环境交互与奖励机制优化策略（如Q-learning、深度强化学习），帮助完成自动驾驶、游戏AI等复杂决策任务[20]。此外，深度学习依靠神经网络（如CNN、RNN、Transformer）处理高维非线性数据（图像、语音、文本），推动计算机视觉与自然语言处理的突破[21]；集成方法（如随机森林、梯度提升树）通过多模型融合提升泛化能力，而概率图模型（贝叶斯网络、隐马尔可夫模型）则用于不确定性推理。这些方法的选择取决于任务需求、数据特性及计算资源，共同支撑从数据规律提取到智能决策的完整链条。

2.2 决策树模型

决策树是一种常用的分类的机器学习方法。它是一种树形结构，由节点（Node）

和有向边（Directed Edge）构成。节点有两种类型：内部节点（Internal Node）、叶节点（Leaf Node）。通常每个内部节点表示一个属性特征，每个叶节点表示一个类别[22]。

决策树通过递归方法分割数据特征（如使用信息增益或基尼系数选择最优划分点）构建分类或回归规则，其核心目标是把样本递归划分到叶节点来完成预测。该模型具有直观和可解释性的特点，可直接生成“若特征A满足条件，则执行分支B”的规则，支持数值型数据和类别型数据，且计算高效，但存在过拟合风险（需通过剪枝或集成方法优化）和对数据微小波动的敏感性[23]。

2.3 ID3决策树

ID3算法的核心思想基于信息论中的信息增益概念，每次选择信息增益最大的属性作为分裂属性来构建决策树，目的是最大化的减少不确定性[24]。它和信息熵紧密相关。信息熵是度量数据集混乱程度或不确定性的指标，信息越不确定，越无法进行决策，故使用尽可能确定的属性来进行决策。熵值越大，数据集越混乱。

信息熵的计算公式为：

$$\mathrm{Entropy}(S) = -\sum_{i=1}^{n} p_i \log_2 p_i \qquad \text{式（2-1）}$$

其中pi是数据样本为类别Ci的概率。

上面提到的信息增益，用于表示某个属性对数据集进行划分后，系统的不确定性减少的程度。计算公式为：

$$\mathrm{Gain}(S,A) = \mathrm{Entropy}(S) - \sum_{v=1}^{k} \frac{|S_v|}{|S|} \mathrm{Entropy}(S_v) \qquad \text{式（2-2）}$$

其中A是属性，k是属性A的不同取值个数，Sv是根据属性A取值v划分后的子集。

ID3算法的算法步骤是：首先计算信息熵，衡量数据的初始不确定性。接着选择最佳属性，对每个属性计算其信息增益，选择信息增益最大的属性作为当前节点的分裂属性。然后划分数据集，根据选定的属性将数据集划分成多个子集，每个子集对应该属性的一个取值。最后递归构建树，对每个子集重复上述步骤，直到满足停止条件，然后构建决策树[25]。

2.4 本章小结

本章就机器学习与决策树模型进行展开，详细描述了机器学习原理和决策树模型的构建和优化，同时对其中的ID3算法进行了详细介绍，对熵和信息增益等度量进行了介绍，并对机器学习的应用场景进行陈述。对决策树模型和决策树模型的ID3算法进行了描述和公式提供。

第三章 基于 GAS 的游戏框架设计

3.1 MOBA 框架结构设计

通常，MOBA 游戏指的是由玩家操控一个角色，通过购买装备和道具等物品提升角色的属性与数值，并通过操作该英雄实现任务的完成（即摧毁敌方基地）。这里的核心内容是角色本身、角色的装备、角色的属性与角色的技能[26]。

3.1.1 角色基础类模块设计

MOBA 游戏首先要有一个角色类，由这个角色类创建角色的实例化对象。在游戏中通常有多种角色，比如英雄类、小兵类、野怪类。这些角色有许多相同点，比如他们具有的属性种类是一样的，都可以移动，都可以释放技能。本文通过创建所有角色类的一个共有父类，来简化不同角色的开发流程。在本课题中，本文的重点内容是英雄与英雄之间的属性差别与属性评价，所以并不需要过多关注小兵、野怪等其他角色类。

角色类拥有多个属性，如生命值、攻击力、可释放的技能等，这些属性通过成员变量的形式存在。因为本课题使用 GAS 框架设计 MOBA 游戏，所以本文使用 Gameplay Ability 类给角色提供技能，使用 Attribute Set 类给角色提供属性。

角色还需要拥有装备，每件装备都能够给角色提供属性和额外的技能[27]。在本课题中，本文更多关注角色的属性变化，而不关注技能对英雄属性的影响。装备系统的模块设计和实现等具体的内容，本文会在后面详细讲解。

3.1.2 属性集合模块设计

Attribute Set 表示了角色本身拥有的各种属性（Attribute）所组成的集合（Set）。在本课题中，本文会基于角色的攻击力（ATK）、法术强度（MAG）、护甲（DEF）、魔抗（MDEF）进行属性评价，所以本文需要定义许多种的 Attribute。除上述介绍的四种属性外，本文还需要定义：最大生命值（MaxHP）、当前生命值（HP）、护甲衰减（DEFDecay）、魔抗衰减（MDEFDecay）。

3.1.3 玩家控制器模块设计

本文后续会实现 UI 系统用于显示角色的属性值。在虚幻引擎 Gameplay 框架内，通常由 Player Controller 表示一个玩家，在 MOBA 游戏模式下，一个玩家只能控制一个角色，所以通常使用 Player Controller 来实现 UI 的启动逻辑和回调逻辑。本文需要在 Base Player Controller 中创建一个 Base Character 类的对象引用变量来记录存储本文控制的这个角色，同时在 Player Controller 中设置输入映射上下文，以能够让玩家控制角色。玩家控制器也会用于后续的初始化 UI。

综上，本文需要创建几个 C++类：CPP_BaseCharacter、CPP_BaseAttributeSet、CPP_PlayerController。然后本文将除了 AttributeSet 类的 C++类创建蓝图子类，分

别为：BP_PlayerController、BP_BaseCharacter。

3.2 MOBA 框架实现

3.2.1 角色基础模块与玩家控制器模块实现

本文需要给 BP_BaseCharacter 和 BP_PlayerController 这两个蓝图类分别添加上各自需要的变量。在 BP_PlayerController 中创建变量“AsBP_BaseCharacter”，变量类型为 BP_BaseCharacter 类的对象引用。在 BP_BaseCharacter 中添加 Ability System Component 组件作为角色与 GAS 的交互组件。创建变量“技能标签组”，变量类型为 Gameplay Tag Container 类型的数组。同时本文也在 BP_BaseCharacter 中创建变量“AsBP_PlayerController”，变量类型为 BP_PlayerController 的对象引用，来存储操控这个角色的玩家控制器。

3.2.2 属性集合模块实现

属性集合模块中需要实现 CPP_AttributeSet 类，用于让角色拥有实际的属性类型与对应的属性值。在 CPP_AttributeSet 类中，本文需要添加至少 7 个变量：HP、MaxHP、ATK、MAG、DEF、MDEF、DEFDecay、MDEFDecay。在 CPP_AttributeSet.h 中，本文需要对上述变量进行定义，并在 CPP_AttributeSet.cpp 中，对生命值属性进行夹值处理。

代码清单 4.1

```
    // 关于 HP 的定义
// 关于其他 Attribute 的定义在这里省略
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "角色基础属性集")
    FGameplayAttributeData HP;
    ATTRIBUTE_ACCESSORS(UCPP_BaseAttributeSet, HP);
//关于 HP 的夹值
ATTRIBUTE_ACCESSORS(UCPP_BaseAttributeSet, HP);
if(Data.EvaluatedData.Attribute==GetHPAttribute())
  SetHP(FMath::Clamp(HetHP(),0.0f,GetMaxHP()));
```

在 CPP_BaseCharacter 中，本文需要设置 Attribute Set 的改值委托。

代码清单 4.2

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FOnAttributeChangeEvent,float,OldValue,
float,NewValue);
UPROPERTY(BlueprintAssignable,Category="AttributeChange")
FOnAttributeChangeEvent HPChangeEvent;
void OnHPAttributeChanged(const FOnAttributeChangeData& Data);
```

3. 3 GES 模型结构设计

3.3.1 GES 框架简介

游戏中角色需要有获得装备和装备属性的能力，因此本文设计游戏装备系统，全称为 Gameplay Equipment System，简称 GES。GES 会和 GAS 系统协同使用。本文需要让角色拥有装备，角色在拥有装备之后能够获得装备提供的属性。因此本文需要给装备提供一定量的信息，让这个装备能够将自身携带的 Attribute Set 赋值给 Character。

3.3.2 装备类模块设计

装备作为一个类，应当有共有的属性。如每个装备有装备的名称、Attribute Set 等。本文需要创建 BP_BaseEquipment 类，表示所有装备的父类。在 Base Equipment 中，本文需要创建变量“属性效果组”，用于存储将要提供给角色的 AttributeSet。

同时本文也需要给装备创建一个结构体，用于保存装备的信息。对于一个装备比较重要的两个信息就是装备的名称和图标。

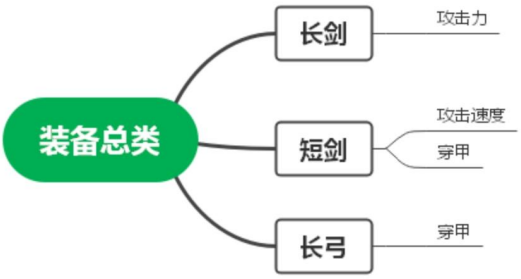

图 3.1 装备类设计

3.3.3 装备系统组件模块设计

将 Equipment System Component 理解为角色携带装备的背包，角色自身可以管理装备。每个角色最多可以携带 6 件装备，角色可以自行安装、拆卸装备。所以本文需要一个通用的、较大规模的框架来实现与装备相关的大量的功能。这里本文选择 Actor Component 类来实现 Gameplay Equipment System。由 GES 管理 BaseEquipment，来给角色提供相应属性。

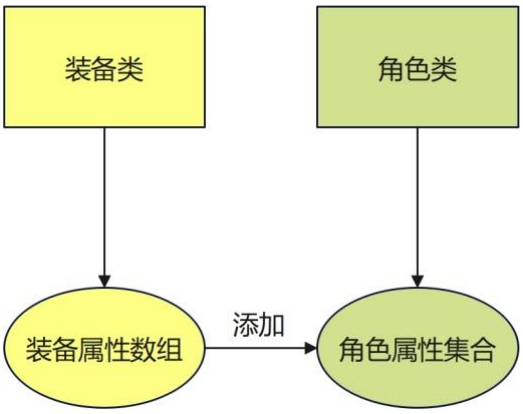

图 3.2 装备系统设计

3.4 GES 模型实现

3.4.1 装备类模块实现

根据上述，本文需要创建一个结构体 S_EquipmentAttribute，然后在 Base Equipment 中创建两个变量：“属性效果组”变量，变量类型为 Gameplay Effect 类引用；“装备信息”变量，变量类型为 S_EquipmentAttribute。

图 3.3 S_EquipmentAttribute 的定义

图 3.4 Base Equipment 中定义的变量

3.4.2 装备系统组件模块实现

创建 Actor Component 类蓝图命名为 AC_EquipmentSystem。在蓝图类中，本文创建 BP_BaseEquipment 类引用变量，命名为“武器位装备 Class”。同时创建该装备的 Attribute 变量，命名为“武器位装备属性”，变量类型为 Active Gameplay

Effect Handle。在该课题中，本文需要创建一共 6 个装备位，除第一个装备位外，剩余五个按照金木水火土进行命名。

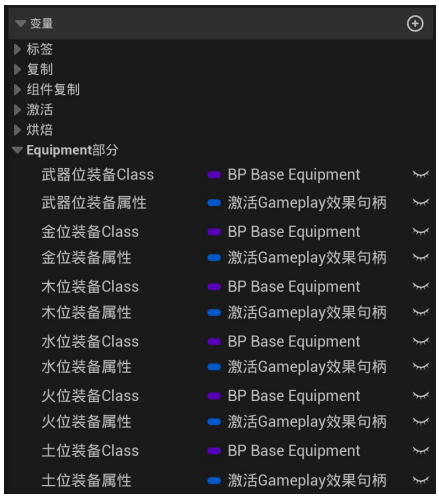

图 3.5　　AC_EquipmentSystem 中定义的变量

然后本文需要实现功能：在安装装备时，将装备的“属性效果组”变量在激活后赋值给该位置的装备属性变量。

下面以武器位装备举例，实现功能。

创建函数 DealWeaponEquipmentAttributeByClass，并实现函数。该函数负责获取武器位装备的 BaseEquipment 的装备信息变量

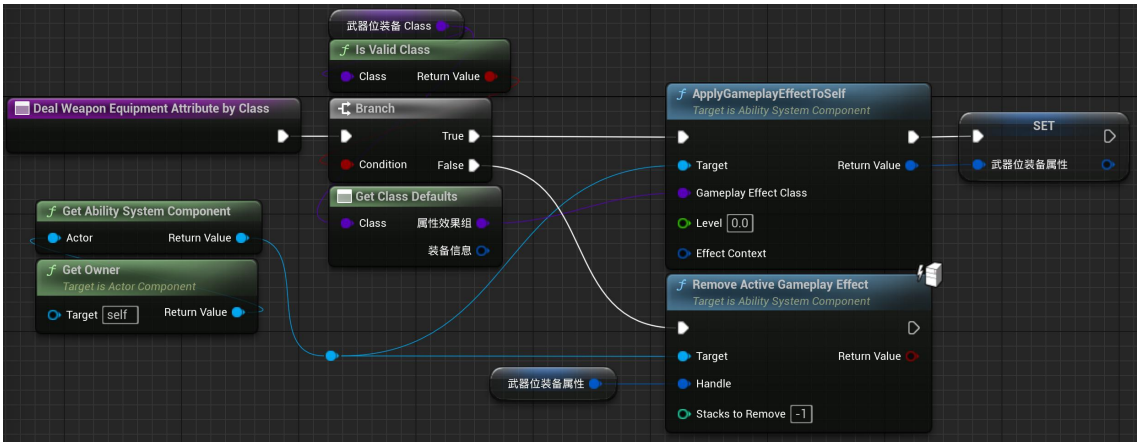

图 3.6　　DealWeaponEquipmentAttributeByClass 函数的实现

然后本文需要在安装装备、卸下装备的时候调用该函数。创建事件分发器 OnWeaponEquipmentUpdate，创建函数 SettleEquipmentByClass 和 UnequipByClass。

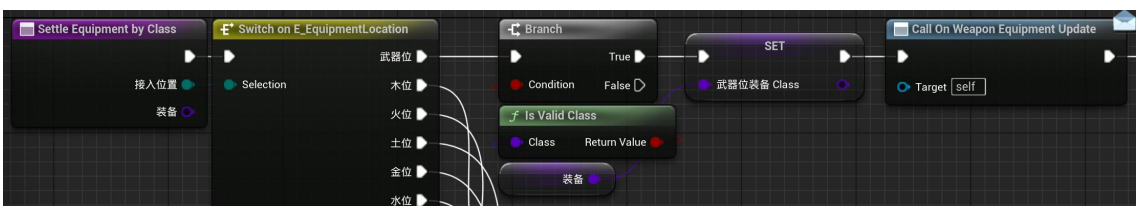

图 3.7　　SettleEquipmentByClass 函数的实现

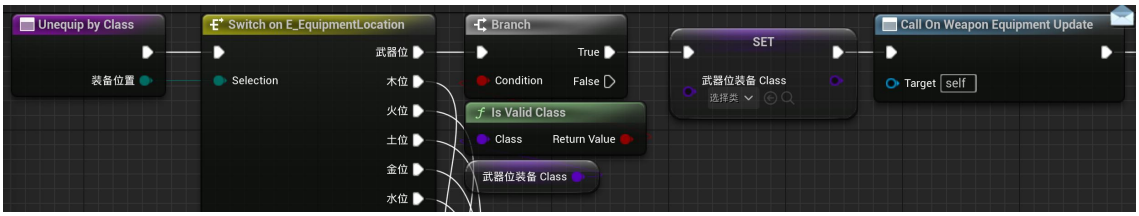

图 3.8　　UnequipByClass 函数的实现

最后在 Event Begin Play 的时候，将事件分发器绑定到函数上。

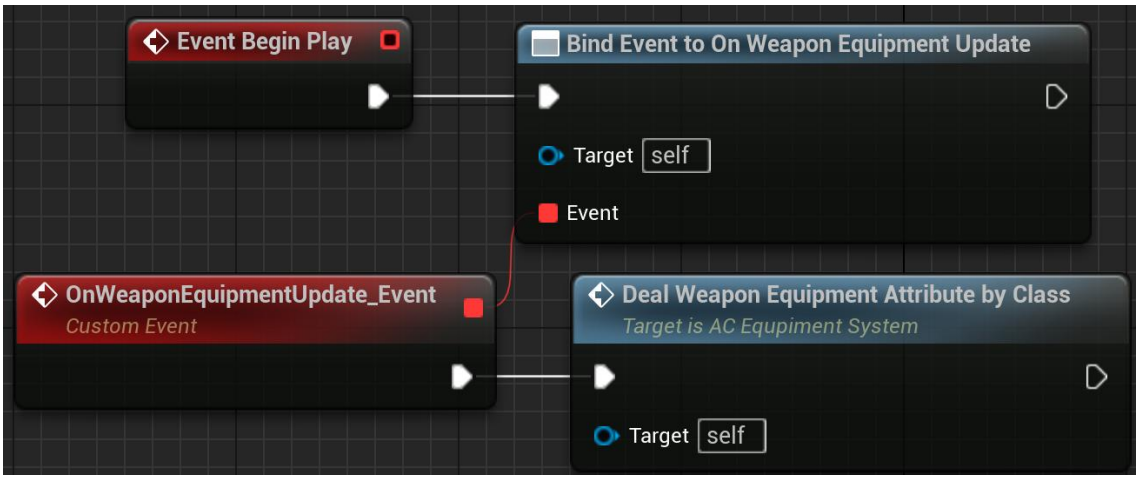

图 3.9　　广播的实现

这样，在调用到 SettleEquipmentByClass 和 UnequipmentByClass 函数的时候，就会调用到事件分发器，进而调用 DealWeaponEquipmentAttributeByClass 函数。就实现了角色的 Attribute Set 随装备的改变而改变。

3.5 本章小结

本章就 GAS 游戏框架设计了一个基本的 MOBA 游戏，实现了 GAS 框架所没有的游戏装备系统框架。首先实现了 BaseEquipment 的基本逻辑，然后通过 AC_EquipmentSystem 管理 BaseEquipment，通过实现一系列函数和调用回调函数，让游戏中的角色可以通过安装或卸下装备来获得或减去装备提供的属性集合。

第四章　游戏功能设计

前文已经实现了游戏框架，本章实现游戏功能。游戏的功能大多体现在 UI 上。通过实现 UI 系统，让本系统能够更直观的观察到角色的属性值与属性变化，从而更清晰的完成后续测试。本文将会创建一个总的 UI，由总 UI 加上各种子 UI，进行参数传递，在子 UI 中实现数据绑定功能。因此本系统创建了一个总界面 UI，由总 UI 加上各种子 UI，进行参数传递，在子 UI 中实现数据绑定功能。同时，在系统实现过程中，本文利用决策树模型实现了策略推荐功能。

4.1 角色属性功能设计

4.1.1 模块设计

本文需要让 UI 能够动态检索角色的属性值，当角色的属性在发生改变的时候 UI 也随之改变。这有两种方法：数据绑定和事件通知。数据绑定是一种 Event Tick 事件，在每一帧都会调用该函数进行设置。但是很多时候本文的数值并不是每一帧都进行改变，如果使用数据绑定就会大量的浪费计算资源。所以本文使用事件通知方法。

在前面定义 CPP_BaseCharacter 类时，本文已经使用委托给角色的 Attribute Set 中所有属性定义了改值委托，这样就会在角色的属性数值发生改变的时候调用委托，并向外进行广播，通知任何关注 Attribute Set 的对象发生了改值事件。本文可以在改值事件发生时修改 UI 的显示数值，以优化游戏性能。

本文核心要显示的就是前面定义的角色的 6 个属性（没有 HP 和 MaxHP）。本文需要在 Widget Blueprint 中创建这 6 个变量对应的文本，将这些文本绑定到显示的对应的属性上。

同时本文也需要给 UI 提供角色参数，让 UI 可以根据参数对文本进行设置。

4.1.2 模块实现

创建函数 InitUI，接受一个 BP_BaseCharacter 类对象引用参数，使用该参数设置 UI 文本。

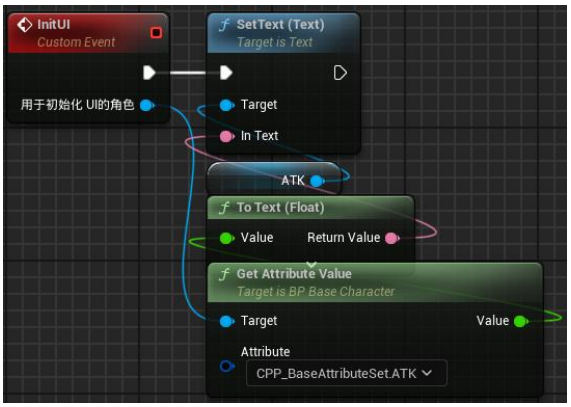

图 4.1 InitUI 函数示例

4.2 装备功能设计

4.2.1 模块设计

前面已经实现了装备类的设计，这里将实现装备的显示。装备 UI 需要用于实时显示装备的相关信息，比如装备的图标。这一变量在 Base Equipment 中已经实现，本文要做的就是在装备的 UI 中读取角色使用的装备的相关参数。本文在 UI 中加上每个装备位对应的图像，以表示该位置的装备，并在角色的装备位更新的时候进行设置。

装备架 UI 也需要一个变量，用于保存装备架的拥有者。

4.2.2 模块实现

创建变量“装备架拥有者”，变量类型为 BP_BaseCharacter 类对象引用。创建 InitUI 函数，用于初始化装备架拥有者变量。当调用该函数后，装备架会根据装备架拥有者角色的 GES 相关属性，对装备架 UI 进行设置，以提供装备的图像。

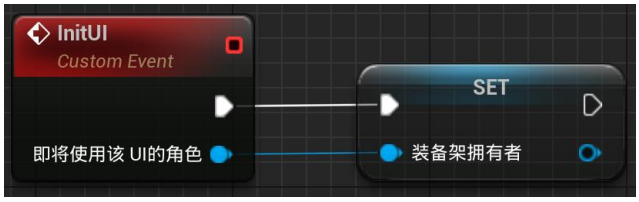

图 4.2 InitUI 函数的实现。

当我们完成了一次装备的安装，就会调用该装备位的事件分发器。我们将一个函数绑定到事件分发器上，来实现对 InitUI 函数的调用。我们通过接口的方式调用。在 Event Construct 的时候绑定事件，并调用接口函数。

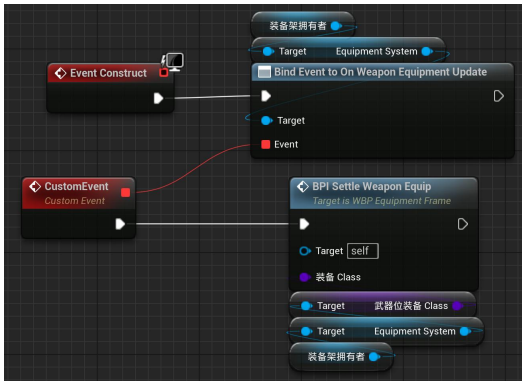

图 4.3 装备架初始化示例

4.3 目标角色设计

4.3.1 模块设计

目标角色设计用于显示目标敌人的相关属性。在 TargetActorUI 中应当有 TargetActor 的变量保存。但本文将 TargetActor 作为 CPP_BaseCharacter 的一个成员变量，所以 TargetActorUI 本质也是需要 BP_BaseCharacter 变量来提供参数。

本文也需要在 TargetActor 发生变化的时候通知 UI 对参数进行修改。

4.3.2 模块实现

在 CPP_BaseCharacter 中声明成员变量 Attack Target Actor，暴露给蓝图。

代码清单 5.1

```
    UPROPERTY(BlueprintReadWrite,Category="Fight 基础")
ACPP_BaseCharacter* AttackTargetActor;
```

在 Target Actor UI 中创建变量“AsBP_BaseCharacter”，变量类型为 BP_BaseCharacter 类对象引用。

创建函数 InitUI，通过 InitUI 初始化 AsBP_BaseCharacter 变量。

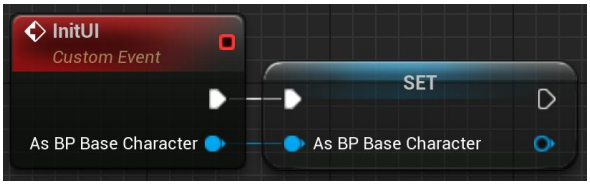

图 4.4 InitUI 函数

在 BP_BaseCharacter 中，当鼠标左键点击到 TargetActor 时，设置 TargetActor 参数，并调用事件分发器 OnTargetActorChange。

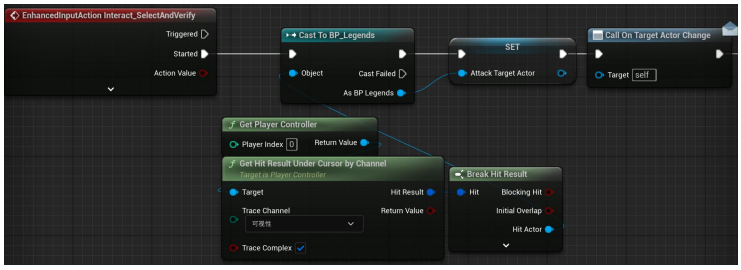

图 4.5 鼠标左键点击事件

在 Target Actor UI 中，绑定事件。

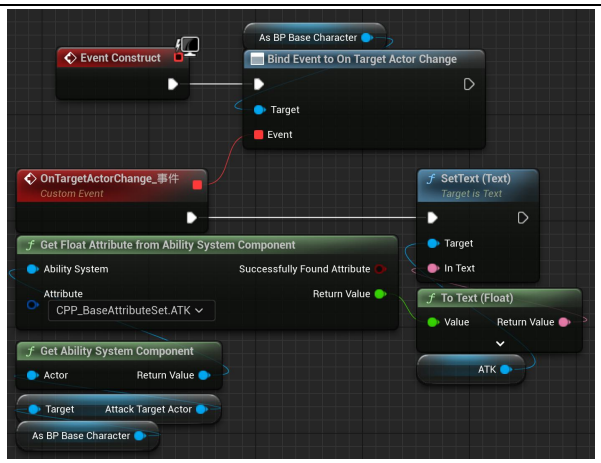

图 4.6 UI 绑定事件示例

4.4 主界面设计

4.4.1 模块设计

Main UI 用于涵盖所有的子控件，如装备显示 UI、角色数值 UI、角色状态 UI 等，也就是将上述创建的所有 UI 在 Main UI 中进行设置。同时，Main UI 中也需要一个变量用于存储它所展示的角色。

4.4.2 模块实现

在 UI 设计器中将子控件加入到主界面 UI 中

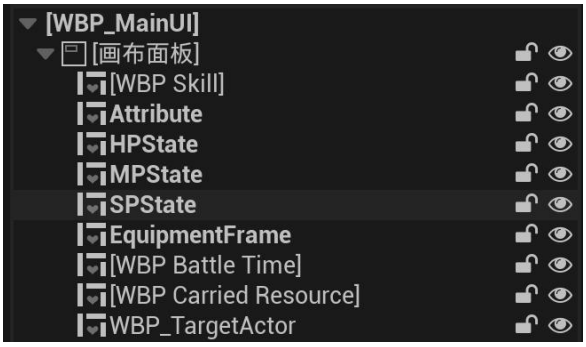

图 4.7 Init UI 设计器层级

创建变量“AsBP_BaseCharacter”，变量类型为 BP_BaseCharacter 类对象引用。创建函数 Init UI，用于初始化 AsBP_BaseCharacter 变量。同时调用各个子控件的初始化函数，给子控件进行初始化

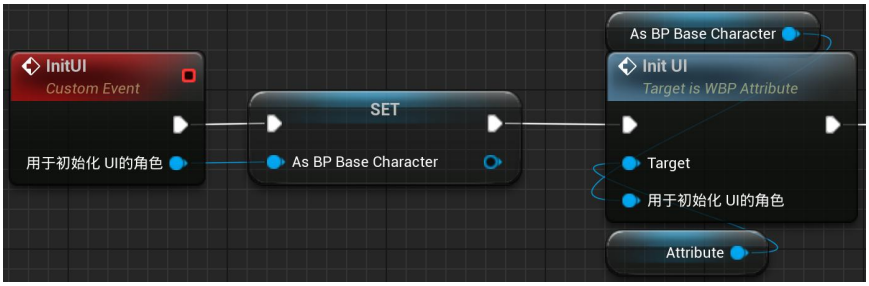

图 4.8 Init UI 函数示例

同时需要绑定事件分发器，当角色的属性发生改变时，如当角色的当前生命值发生改变时，我们需要对主界面 UI 中的生命值子 UI 进行设置相关值。

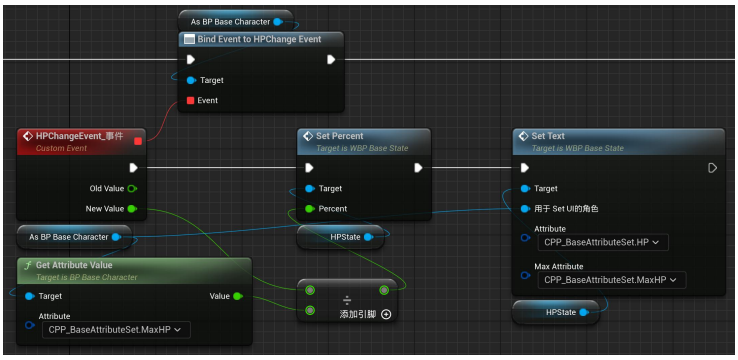

图 4.9 事件分发器绑定改值示例

4.5 决策树模型与策略推荐功能设计

4.5.1 模块设计

本文将决策树作为 BaseCharacter 的一部分，当每次玩家需要通过两个角色来获取装备建议时直接调用角色类内部的相关函数。这样的好处是减少了类之间的通讯。为了通过决策树模型实现对装备的推荐，本文需要想办法给决策树提供参数，让决策树返回预测值。现在就要给角色设置评价值。角色的输出属性和防御属性，本文将这两个属性创建对应的变量。角色的自身偏向也应当创建对应的变量。

对于角色身上的各种属性，本文会对角色做出两种评价，并根据角色自身特点（作为角色类的默认值），数据集按照如下划分：

表 2.1　　数据集中的属性

角色自身偏向	本方输出属性	本方防御属性	敌方输出属性	敌方防御属性

此外，本文需要通过属性得到的推荐结果为：

表 2.2　　数据集中的预测类别

输出装推荐 Tag	防御装推荐 Tag	装备推荐项 Tag

对于 5 种属性，每种属性有确定的样本空间：

表 2.3　　每种属性的样本空间

角色自身偏向：输出、生存
本方输出属性：物理伤害、魔法伤害、真实伤害
本方防御属性：护甲偏高、魔抗偏高、均衡
敌方输出属性：物理伤害、魔法伤害、混合伤害
敌方防御属性：护甲偏高、魔抗偏高、均衡

对于 3 种预测类别，每个预测类别也有确定的类别集合：

表 2.4　　每种预测类别的类别集合

输出装推荐 Tag：物理、法强、穿透、about
防御装推荐：护甲、魔抗、生命值
装备推荐项：优先输出、优先防御、any

输出属性只有三种可能取值：HATK、HMAG、HHRT。

防御属性只有三种可能取值：HDEF、HMDEF、HLIVE。

角色自身偏向只有两种可能取值：HOTPT、HLIVE。

本文的预测结果中，

输出装推荐 Tag 只有三种可能取值：ATK、MAG、Decay。

防御装推荐 Tag 只有三种可能取值：DEF、MDEF、HP。

装备推荐项 Tag 只有两种可能取值：OTPT、LIVE。

因此，本文创建三种枚举。同时，创建枚举转字符串的函数，然后在类中创建变量用于存储角色的属性评价。本文也需要实现函数来设置角色评价，并通过 BaseCharacter 中的相关成员变量给决策树提供参数，来构建决策树。

4.5.2 模块的 C++实现

虚幻引擎风格 C++较难实现上述内容，也不利于后续实验。本文先用标准 C++实现相关内容，并在之后迁移至虚幻引擎中。

按照上面的描述，数据集是可以暴力枚举出来的。先不考虑数据集的准确性，本文创建一个完整的、最全面的数据集，共有 2*3*3*3*3=162 的数据量。本文将这样得到的数据集存放在一个 csv 表格中。

在得到数据集后，本文便可以开始实现 ID3 算法的各个功能。核心就是实现几个函数：CalculateEntropy、CalculateInfoGain、ChooseBestAttribute、BuildDecisionTree。

本文首先创建决策树的数据结构。

代码清单 4.1

```
struct DataRow {
    vector<string> conditions;
    vector<string> decisions;
};
struct TreeNode {
    string attribute;
    map<string, shared_ptr<TreeNode>> children;
    string decision;
};
```

然后依次实现相应函数。

代码清单 4.2

```
// 计算信息熵
double calculateEntropy(const vector<DataRow>& dataset, int targetIdx);
// 计算信息增益
double calculateInfoGain(const vector<DataRow>& dataset, int attrIdx, int targetIdx) ;
// 选择最佳划分属性
int chooseBestAttribute(const vector<DataRow>& dataset, const vector<bool>& usedAttributes, int targetIdx) ;
// 构建决策树
shared_ptr<TreeNode> buildDecisionTree(const vector<DataRow>& dataset,
    const vector<string>& attributes,
    vector<bool> usedAttributes,
    int targetIdx) ;
```

实现完 ID3 算法需要的函数之后，本文就需要创建 CSV 解析函数和预测函数，从 csv 表格中获取数据集并通过 ID3 算法进行预测。

代码清单 4.3

```
// CSV 解析函数
vector<DataRow> loadCSV(const string& filename) ;
// 预测函数
string predict(const shared_ptr<TreeNode>& root, const DataRow& sample, const vector<string>& attributes);
```

调用 ID3 算法时，因为数据集的特殊性，本文会创建三棵决策树。本文将通过创建 sample 变量给 predict 函数提供实参，当实际使用时，本文只需要修改 sample 变量的实际值即可。

代码清单 4.4

```
vector<string> attributes = {
    "英雄自身偏向", "本方输出属性", "本方防御属性",
    "敌方输出属性", "敌方防御属性"
};
DataRow sample = {
    {"输出", "真实伤害", "护甲偏高", "物理伤害", "护甲偏高"},{}
};
vector<shared_ptr<TreeNode>> trees;
for (int target = 0; target < 3; ++target) {
    vector<bool> usedAttributes(5, false);
    trees.push_back(buildDecisionTree(dataset, attributes, usedAttributes, target));
}
for(int i=0;i<3;++i){
    cout<<predict(trees[0],sample,attributes)<<endl;
}
```

4.5.3 模块的虚幻引擎实现

将标准 C++代码转换成虚幻引擎风格的代码，需要注意数据结构和相应算法的转换。数据结构转换主要是将 std::map 转换成 TMap，将 std::vector 转换成 TArray，将 shared_ptr 转换成 TSharedPtr。

算法上，在 TMap 中并不存在 std::map 中的 begin()函数。所以本文需要将相应的逻辑重写。

在 std::map 中 begin()函数用于返回一个指向 std::map 中第一个元素的迭代器。而在 TMap 中，元素的存储顺序是无序的，并不存在首元素。

本文需要根据输入的类别统计 ClassCounts 的统计结果，计算出现次数最多的类别，并返回该类别的名称。所以本文需要实现该函数。

代码清单 4.5

```
// 获得最高频率的类
FString ACPP_BaseCharacter::GetPluralityClass(const TMap<FString, int32>& ClassCounts);
```

在虚幻引擎中，本文需要额外的创建几个枚举，来向引擎提供相应参数。

代码清单 4.6

```
    UENUM(BlueprintType)
enum class ELegendComment:uint8
{
    HOTPT UMETA(DisplayName = "输出"),
    HLIVE UMETA(DisplayName = "生存")
};
```

虚幻引擎中，本文需要额外的设置几个函数，来正确的调用决策树的相关功能。

代码清单 4.7

```
    // 设置角色属性评价
    void ACPP_BaseCharacter::SetCharacterAttributeComment();
    // 设置决策树
    // 该函数在 BaseCharacter 的构造函数中调用。
    void ACPP_BaseCharacter::SetTrees(TArray<TSharedPtr<FTreeNode>>& Root, TArray<FDataRow> Dataset,TArray<FString> Attributes);
    // 设置预测评论
    void SetPredictComment(ACPP_BaseCharacter* TargetActor);
    // 调用预测并得到建议
    FString ACPP_BaseCharacter::CallPredictAndGetAdvice(int32 TreeIndex, ACPP_BaseCharacter* TargetActor,FDataRow TwoLegendComment)
```

最后，在引擎中调用 CallPredictAndGetAdvice 函数。

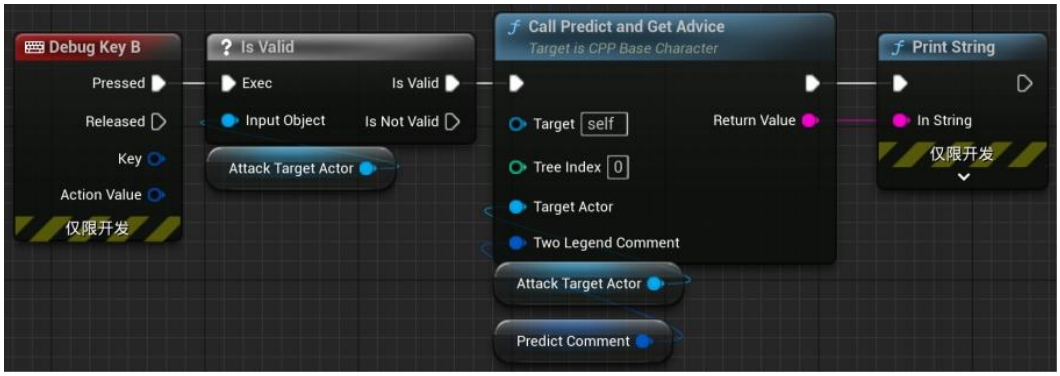

图 4.8 在引擎中调用 CallPredict 函数

4.6 本章小结

本章就游戏功能设计进行展开，对角色属性、装备功能、目标角色、主界面的 UI 进行陈述，并实现决策树模型与策略推荐功能。首先通过标准 C++实现装备推荐功能，然后再将代码迁移到虚幻引擎中，并在游戏中实现装备推荐功能。并完美实现上述功能。

第五章　　实验结果与分析

5.1 实验环境配置

在第四章中，本文通过标准 C++得到了枚举全部情况下的数据集，并给每个样本提供了较为准确的预测结果，但是本文并不能保证每条数据集都是绝对准确的。尽管如此，本文在接下来的实验中，仍然认为数据集下的所有样本都是尽可能正确的。这就带来了一个问题：样本的正确率。因为样本之间是相互等价的，所以为了解决样本准确率的问题，本文给总体样本设置置信度，以表示总体中有多少的样本是可信任的，并通过可信任的样本作为训练集，对不可信任样本进行预测，并将预测值和真实值进行比对。

5.2 数据分析与预处理

5.2.1 数据分析

本文的数据集中每条样本的格式是：

表 3.1　　数据集样本格式和一条样本示例

角色自身偏向	本方输出属性	本方防御属性	敌方输出属性	敌方防御属性	输出装推荐 Tag	防御装推荐 Tag	装备推荐项 Tag
输出	物理伤害	护甲偏高	物理伤害	护甲偏高	穿透	护甲	优先输出

其中的前 5 列是样本的属性，后 3 列是样本的预测类别。本文要做的就是根据前 5 列的值预测后 3 列的结果。

现在本文已经暴力枚举出了一共 162 种全部的可能，但本文并不能假设这些所有的样本都是绝对准确的，只能是基本确定一些样本是正确的，然后假设样本都是尽可能正确的。这样就需要考虑，在实际使用时，训练集和测试集的设置。

5.2.2 数据预处理

本文一共设置两组训练集和对应的测试集。

第一组中本文假设数据集的样本标签全部认为是正确的，将整个数据集划分为训练集和测试集，进行模型测试。

第二组中本文假设数据集的样本标签置信度高于某一预设值(x%)时其标签是正确的。本文将认为正确的 x%的样本全部作为训练集，剩余的(100-x）%全部作为测试集，通过对多种情况下的测试集进行预测，得到不同置信度下的测试集准确率。

在进行后面的测试时，本文可以只保留一个预测类别。这里本文只保留输出装推荐 Tag 一个预测类别，这样决策树也只需要构建一个。同时，两组数据预处理后的结果是一样的，都是在总体中选出 x%个样本作为训练集，剩余的作为测试

机。其结果页应当一致。

5.3 算法设置

对于预处理的两组数据：

当本文认为数据集的样本全部正确时，将整个数据集作为训练集进行测试，在随机选取几条样本作为测试集的情况下，可以预测到，测试集准确率是百分百。正式实验也确实是这样。本文通过给代码中的 sample 变量随机赋值，在测试了 20 条的情况下，测试集的每个样本预测的都是正确的。

当本文认为数据集中只有 x%的样本是正确时，本文设置如下算法。

从数据集中随机取出 162*(1-x%)条数据样本作为测试集，保存在新的 csv 表格中，并将取出的样本从数据集中剔除，将处理后的数据集作为训练集，在训练完成后对测试集进行测试，测得测试机的准确率。每一个 x 应当多次测量求准确率的平均值。

所以本文需要实现 csv 的处理函数。处理函数的基本算法就是：读取数据集中所有行到 vector，随机选择 row 行，保存到 test.csv，从 vector 中删除这些行，然后将剩余的写回到 train.csv。

最后，将预测结果写入到 test.csv 的行尾。

代码清单 5.1

```
// 分割数据集
void splitDataset(const string& inputFile, int row, const string& outputTestFile) ;
// 将预测结果写入文件
void writePredictionsToCSV(const string& filename,const shared_ptr<TreeNode>& tree,const vector<string>& attributes) ;
// 计算准确率
double calculateAccuracy(const vector<DataRow>& testSet, const shared_ptr<TreeNode>& tree,const vector<string>& attributes, int targetIdx) ;
int main() {
    splitDataset("train.csv", 50, "test.csv");
    vector<string> attributes = {"英雄自身偏向", "本方输出属性", "本方防御属性","敌方输出属性", "敌方防御属性"};
    vector<bool> usedAttributes(5, false);
    auto trainDataset = loadCSV("train.csv");
    shared_ptr<TreeNode> tree = buildDecisionTree(trainDataset, attributes, usedAttributes, 0);
    auto testDataset = loadCSV("test.csv");
    double accuracy = calculateAccuracy(testDataset, tree, attributes, 0);
    cout << "模型准确率: " << accuracy << "%" << endl;
    writePredictionsToCSV("test.csv", tree, attributes);}
```

为了实现自动化测试，本文需要额外实现部分功能。该项目需要循环 TEST_TIMES （后文假设为 5）次，并根据本文将要分割出的测试集行数进行循环，将测试集行数、置信度 x、五次预测准确率、五次预测的平均值作为一行数据保存到 test_result.csv 表格中。

代码清单 5.2

```
for (int row = 0; row <= TOTAL_ROWS; ++row) {
    if (row < 0 || row > TOTAL_ROWS) continue;
    double x = 100.0 * (TOTAL_ROWS - row) / TOTAL_ROWS;
    vector<double> accuracies;
    for (int test = 0; test < TEST_TIMES; ++test) {
        splitDataset(ORIGINAL_FILE, row, "train_over.csv", "test_over.csv");
        auto trainData = loadCSV("train_over.csv");
        if (trainData.empty()) {
            accuracies.push_back(0.0);
            continue;
        }
        auto tree = buildDecisionTree(trainData, attributes, usedAttributes, 0);
        auto testData = loadCSV("test_over.csv");
        double accuracy = calculateAccuracy(testData, tree, attributes, 0);
        accuracies.push_back(accuracy);
    }
}
```

这样就可以完成自动化测试。将测试结果保存到 test_result.csv 中

5.4 决策树测试结果与分析

上述实验算法代码不支持 x=100 的情况，因为算法是取出 row 行数据作为测试集，x=100 时没有测试集的存在。但本文依然按照上述算法做统一讨论，只不过需要提前说明一下 x=100（row=0）时，表格中的数据不合理。

本文经过运行代码，得到了一张 164 行的表格（算上表头）。经统计，置信度与平均准确率之间符合如下关系。

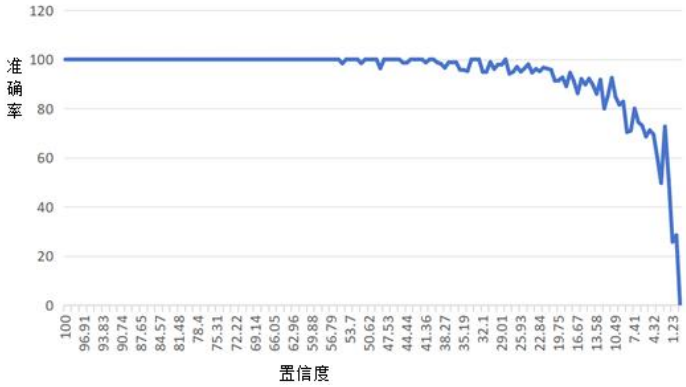

图 5.1 置信度与预测准确率之间的关系

由上图可以看到，当置信度在 30%左右时，准确率依旧很高。也就是说，本文只需要保证本文的数据集中有 30%是绝对正确的，就可以基本保证推荐结果是正确的。

5.5 游戏运行结果与分析

在运行游戏并选定目标角色后，调用 CallPredictAndGetAdvice 函数，游戏画面会出现如下结果

图 5.2 游戏运行画面

如图所示结果，现在我们选择了一个目标角色，根据我们当前的 100 攻击力和 20/20 的穿甲，面对目标的 25 的护甲，系统给我们推荐的是 Decay 装备，表示我们需要购买穿透类的装备。

图 5.3 游戏运行画面

在我们购买穿透类的装备后，系统给我们推荐的是 HP 类的装备，表示我们需要购买生命值类的装备。

5.6 本章小结

本章构建了一个枚举全部情况的数据集，本文为数据集中样本设置了置信度，并在不同置信度情况下的预测准确率进行了测试，得到了结论。在实验过程中，我们选择好样本分割的方法并进行测试，并验证了模型的有效性，并得出结论，本文所设计的系统能够位用户提供准确的装备推荐，提升用户体验。

第六章　总结与展望

6.1 总结

本文基于虚幻引擎和机器学习算法设计了一套完整的单机游戏中装备推荐系统。该系统包括了装备数值模块、装备购买模块、装备推荐模块等。

具体的工作如下：

（1）本文利用虚幻引擎基于 GAS 框架构建了一个 MOBA 类游戏，设计实现了几个角色，并实现了游戏装备系统。通过虚幻引擎 GAS 系统实现了 GES 装备系统框架，并和机器学习决策树算法合作实现了装备推荐系统。对后续游戏开发有着指导性作用。

（2）本文利用决策树 ID3 算法构建了一个完整的装备推荐系统，能够在当前角色和目标角色都确定的情况下实现装备推荐的功能，以提升用户体验。本文关注了决策树 ID3 算法在游戏角色装备推荐系统的使用与成果，并且得到同一训练集下对于训练集不同置信度的预测准确率的关系，得到了数据集在尽可能大的准确率下尽可能小的置信度，有利于指导人们选择合适大小的训练集构建决策树。

6.2 展望

本课题实现的决策树系统能够正确预测装备推荐结果，但仍然有一些不足的地方。

（1）角色属性评价变量是按照标准角色的数值进行评价的。标准角色的数值为 ATK=100，MAG=100，DEF=50，MDEF=50，并让角色的当前属性和标准属性进行对比。实际上，角色的属性评价变量应当是按照本场游戏所有角色的特点进行评价，如一方队伍全部为法术输出，另一方全部为高法抗低物抗英雄，则这样的评价标准就不再合理。在后续的研究过程中，可以通过全局观察者获得所有角色的平均属性值作为标准属性值，这样标准属性值是动态的，更加适合作为装备推荐的标准属性值。

（2）部分算法也是决策树系统也是基于本数据集定制的，在实现装备推荐系统过程中，本文假设样本中的数据均无缺失。但在实际使用过程中通常会遇到大量缺失的数据，此时如何对缺失值进行处理以提升模型的健壮性是后续研究的另一重点内容。

（3）该推荐系统使用 ID3 算法，推荐的精度受到了限制。这里也可以尝试使用其他的深度学习算法，如神经网络、随机森林等其他机器学习模型。此外，决策树只能在一些层面帮助我们实现智能系统。对于许多技术需求，如智能 AI 等，也需要使用其他类型的机器学习算法。

致　　谢

在此论文即将完成之际，内心充满了激动的心情。这四年中，我获得了足够的成长，学会了如何面对挑战。感谢给我给予帮助的所有人。首先感谢李平老师以丰富的经验悉心帮助我完成论文。接着感谢辅导员冯亚洲，在大学的学习生活过程中给予了极大的帮助。然后感谢我的父母，对我学习的支持。

本科生涯就快结束了，回首过往，学到了很多东西。计算机本就是我感兴趣才学的一门学科，我也是一个兴趣驱动的人。感谢南京邮电大学对我的培养和提供的丰富的学习资源。将来我会一直努力学习，争取获得更多的成就。同时感谢国家为我们提供的良好的社会环境与学习环境，今后必定为共产主义事业贡献自己的力量。

最后感谢辅导员冯亚洲，在大学的学习生活过程中给予了极大的帮助。感谢我的父母，对我学习的支持。感谢室友和同学们在学习生活中对我提供的帮助。

参考文献

[1] Engine U. Unreal engine[J]. Retrieved from Unreal Engine: https://www. unrealengine. com/en-US/what-is-unreal-engine-4, 2018.

[2] 林澳凯. MOBA 游戏平衡性探究[D]. 云南大学, 2015.

[3] Buro M, Furtak T M. RTS games and real-time AI research[C]//Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS). 2004, 6370: 1-8.

[4] Mora-Cantallops M, Sicilia M Á. MOBA games: A literature review[J]. Entertainment computing, 2018, 26: 128-138.

[5] Agrahari V, Chimalakonda S. What's inside unreal engine?-a curious gaze![C]//Proceedings of the 14th Innovations in Software Engineering Conference (formerly known as India Software Engineering Conference). 2021: 1-5.

[6] Kekomäki T. Unreal Engine ja Gameplay Ability System[J]. 2022.

[7] Amereh F. A Study and Implementation of Turn-Based Combat Systems in Role-Playing Games[J]. 2024.

[8] 黄艺坤. 三维游戏中的 NPC 人工智能技术研究[J]. 信息技术与信息化, 2014 (12): 72-73.

[9] 冯镍, 赵前进. 基于平台游戏的关卡内容程序自动生成[J]. 数字技术与应用, 2013 (10): 189-190.

[10] Huang J, Yan E, Cheung G, et al. Master maker: Understanding gaming skill through practice and habit from gameplay behavior[J]. Topics in cognitive science, 2017, 9(2): 437-466.

[11] 鲍艳春, 石彩霞, 张传强, 等. 深度学习在基因组学中的研究进展[J]. 遗传, 2024, 46(9): 701-715.

[12] 万金宇, 孙正, 张相, 等. 机器学习在大型粒子加速器中的应用回顾与展望 D[J]. 强 激光 与粒 子束, 2021, 33(9).

[13] Jiang P, Ergu D, Liu F, et al. A Review of Yolo algorithm developments[J]. Procedia computer science, 2022, 199: 1066-1073.

[14] 闻学伟, 汝宜红. 智能物流系统设计及应用[D]. , 2002.

[15] 兰欣, 卫荣, 蔡宏伟, 等. 机器学习算法在医疗领域中的应用[J]. Chinese Medical Equipment Journal, 2019, 40(3).

[16] 杨书. 目标检测中候选区域准确高效定位的机器学习方法研究[D]. 西安理工大学, 2019.

[17] 杨善林, 倪志伟. 机器学习与智能决策支持系统[M]. 科学出版社, 2004.

[18] 周志华. 机器学习与数据挖掘[J]. 中国计算机学会通讯, 2007, 3(12): 35-44.

[19] 刘建伟, 刘媛, 罗雄麟. 半监督学习方法[J]. 计算机学报, 2015, 38(8):

1592-1617.

[20]高阳，陈世福，陆鑫．强化学习研究综述[J]．自动化学报，2004, 30(001): 86-100.

[21]胡越，罗东阳，花奎，等．关于深度学习的综述与讨论[J]．智能系统学报，2019, 14(1): 1-19.

[22]刘小虎，李生．决策树的优化算法[J]．软件学报，1998, 9(10): 797-800.

[23]王黎明．决策树学习及其剪枝算法研究[D]．武汉：武汉理工大学，2007.

[24]杨清，杨岳湘．基于决策树的学习算法[J]．湘潭师范学院学报：社会科学版，1999 (3): 58-62.

[25]杨明，张载鸿．决策树学习算法 ID3 的研究[J]．微机发展，2002, 12(5): 6-9.

[26]Polančec D, Mekterović I. Developing MOBA games using the Unity game engine[C]//2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 2017: 1510-1515.

[27]何承潜. MOBA 游戏角色设计及传播策略研究[D]．北京：北京体育大学，2019.