HUNTBot —第一人称射击游戏中 NPC 的结构设计*)

杨 佩[1,2] 王 皓[2] 罗文杰[2] 高 阳[2]

(南京大学工程管理学院 南京 210093)[1] (南京大学软件新技术国家重点实验室 南京 210093)[2]

摘 要 游戏产业的发展迫切需要使用新的技术开发具有智能行为的 NPC,Agent 技术因其对人类智能的刻画及模拟不失为一种好的选择。同时,电脑游戏也因为其固有的复杂、实时、动态性而吸引了众多 Agent 研究者的目光。针对第一人称射击游戏——"虚幻竞技场(Unreal Tournament,UT)"设计了 HUNTBot 作为游戏中的非玩家角色 NPC。这种 Agent 具有混合式结构,使 Agent 既能对变化的环境迅速做出反应,又能对目标进行实时规划,并具有社会性和学习能力。因此 Agent 能够适应动态、复杂、实时的游戏环境,使 NPC 的智能行为更加接近人类玩家。

关键词 第一人称射击游戏,Agent 结构,多 Agent 系统

HuntBot —Architecture of NPC in the FPS Game

YANG Pei[1,2] WANG Hao[2] LUO Wen-Jie[2] GAO Yang[2]

(School of Management and Engineering, Nanjing University, Nanjing 210093, China)[1]

(National Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)[2]

Abstract The development of the game industry has brought forth a requirement of applying novel intelligent technologies in games. The agent technique is a good choice for its portraying and simulating ability of human intelligence. Meanwhile, large complex computer games are attracting more and more agent researchers. Designed the HUNTbot (Humanoid UNreal Tournament Bot) as the NPC in the first person shooting (FPS) game—Unreal Tournament, which has the hybrid agent architecture, social and learning ability. HUNTBot can behave well in complicated, dynamic and real-time environment.

Keywords FPS, Agent architecture, MAS

1 引言

自 1992 年 Doom 游戏出现以来,电脑游戏的开发进入了"游戏引擎时代",这种全新的游戏编程模式可以让游戏开发人员深入到游戏核心,用新的模型、场景和声音开发出新的游戏,或向已有的游戏素材中添加新的内容。其中非玩家角色(non-player character,NPC)已经成为 3D 游戏引擎中一个重要的部分。然而,传统 NPC 的开发方法是使用预先编辑的脚本来描述角色的行为,也就是 NPC 按照预先编辑好的脚本来执行各种动作。这种开发方法存在一定的缺陷,NPC 的适应性较差[1]。现在的 3D 游戏引擎越来越注重通过引入人工智能技术来提高游戏的趣味性。游戏开发者必须寻求更有效的人工智能技术使 NPC 的行为更具智能性,Agent 技术因其对人类智能行为的准确解释及模拟无疑成为一种更好的选择[2]。同时,电脑游戏也因为其提供了逼真的任务场景、具备复杂及动态性而吸引了越来越多人工智能研究者的目光。多 Agent 系统的研究者们希望能以电脑游戏为测试平台,通过控制游戏中 NPC 的行为[1],研究多 Agent 系统中的各种问题,并利用它来评价多 Agent 系统的理论、算法和体系结构。其中,第一人称射击(first-person shooter,FPS)游戏作为流行最广的游戏获得了众多研究者的青睐。

已有一些学者对 FPS 中的 NPC 结构设计进行研究并取得了初步的研究成果。Magerko 等人在 FPS 游戏"雷神(Quake)"上构建了一种 NPC-Quakebot[3]。这种 Agent 结构属于反应型,通过事先对目标进行分解来对 Agent 的行为进行推理。Orkin 建立了一个基于 FPS 游戏"虚幻竞技场(Unreal Tournament,UT)"的 NPC 结构,该 Agent 结构属于思考型,实现了基于符号匹配的动态规划,但实时性较差,规划过程有时要持续数秒甚至数分钟。Burkert 等人在一个 Agent 开发平台 Pogamut 2 中提出了一种简单的 NPC 结构,将 Agent 的感知信息直接连接到一个反应式规划器上[5]。以上研究对 FPS 游戏环境中的 Agent 结构设计和实现进行了有益的探索,但普遍存在的问题是这些 Agent 结构均只支持 Agent 个体行为的实现,没有考虑 Agent 的社会行为的实现,而且 Agent 均不具备学习能力。我们认为,游戏中的智能 Agent 设计必须兼备反应性(reactive)和思考性(deliberative),反应性保证 Agent 能对环境变化做出快速反应,而思考性则使 Agent 能够对目标进行推理及规划,自动组织行动序列以达成目标。另外,Agent 应该具有社会能力,从而表现出与其它 Agent 或人类玩家的合作及对抗性。同时 Agent 也应该具有学习能力,从而更好地适应环境。我们曾经提出了一个 FPS 游戏中的智能 NPC 结构 AC4[6],本文工作可视作该工作的延续。

针对以上问题,本文基于 FPS 游戏"虚幻竞技场"提出了一种智能 NPC 结构——HUNTBot(Humanoid UNreal Tour-

*)本文受国家自然科学基金项目(60775046)资助。杨 佩 博士,主要研究领域为智能 agent、多 agent 系统;王 皓 硕士研究生,主要研究领域为智能 agent、多 agent 系统;罗文杰 硕士研究生,主要研究领域为智能 agent、强化学习;高 阳 博士,副教授,主要研究领域为机器学习、分布式人工智能等。

[1] 本文游戏领域的词汇 NPC 视为 Agent 的同义语,不加区别。

nament Bot)。全文内容安排如下:第 2 节介绍本文研究工作的实验平台 Pogamut 2;第 3 节介绍 HUNTBot 的结构、处理流程及实验例;最后是未来的工作安排和展望。

2 HUNTBot 的开发平台

已有一些研究者开发出了基于 FPS 游戏的多 Agent 系统实验平台,其中包括密歇根大学的 Quakebot [3]、南加州大学的 Gamebots [7]、Monolith 公司的 FEAR 工程[8]以及捷克的查尔斯大学开发的 Pogamut 2[5]。我们选取 Pogamut 2 作为 HUNTBot 的开发平台,这是一款在 UT2004 游戏引擎之上建立的 Agent 开发平台,与其他类似的平台相比,它提供了一个用于 Agent 开发及测试的集成开发环境,对游戏引擎的支持性更强,有利于大大缩短 NPC 的开发时间。

图 1 是 Pogamut 2 的系统架构,采用了 Client-Server 模式。图中服务器端包括两个部分,其中的 UT2004 游戏引擎负责虚拟游戏环境的模拟计算,它接受 Gamebots2004 发送的各类动作命令,定期计算这些命令对游戏环境的改变,并将改变后的游戏环境信息发送给 Gamebots2004 以作为其 NPC 下一步动作的依据。Gamebots2004 负责将 UT2004 引擎的内部脚本命令翻译成一套基于文本的消息通过网络发送给远程的客户机。消息被定义为两种类型:服务器消息和客户端消息。服务器消息提供了感知信息,用来描述游戏中的 NPC 能看见什么,同时也包括一些 NPC 自身的物理状态信息,如生命力、武器等。服务器消息也包括一些其它的信息,如当 NPC 受到攻击时被射击的方向,以及 NPC 之间的通信消息。客户端的信息遵循命令的格式来指导 NPC 完成任务,如旋转、行走、射击等。由于客户端用 Java 实现,因此 Pogamut 2 建造了一个解析器用于将基于文本的消息转换成 Java 对象,解析器的实现可以有两种方法,远程解析和本地解析,区别在于解析器是否与 NPC 位于同一台主机上。图 1 为本地解析器的示例,本地解析器与 NPC 之间通过 API 连接。

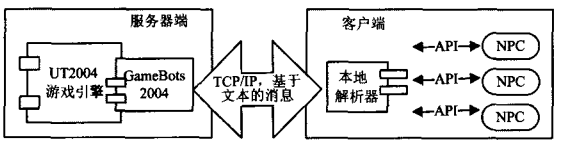

图 1 Pogamut2 的系统架构

我们通过 Pogamut 2 提供的集成开发环境编程实现 NPC,并使其通过 Pogamut 2 提供的 API 连接到 UT2004 的游戏引擎之上。目前 Pogamut 2 支持最多 16 个 NPC 同时连接。人类玩家可以通过 UT2004 提供的游戏客户端直接连接到 UT2004 游戏引擎上与 NPC 对战。

3 HUNTBot-FPS 中的 NPC 结构

3.1 HUNTBot 的结构

HUNTBot 的结构基于我们提出的 AC4[6],并对其作了改进。HUNTBot 的结构如图 2 所示,主要由 5 部分组成。

3.1.1 感知模块

HUNTBot 的感知系统负责从外界接收新的信息,并将这些信息传递给信念模块。我们为这一模块加入了一些干扰信息以使 NPC 的行为更接近人类。例如,一个 NPC 不应该清楚地看到当前环境中的所有物体,尽管它可以从游戏引擎中得到全部的有关物体位置及形状的数据,但 NPC 的视觉信息应该满足某种限制,正如人类的观察会受到视线范围的影响一样。

3.1.2 信念模块

将感知模块传送过来的信息存放在 NPC 的信念集里,并按照信念的性质分类存放,如敌人、盟友、物品管理、游戏地图、目标等等。这种知识的组织方式将相对零散的感知信息有机地组合,便于 Agent 快速有效地查找信息。这些知识反映了 NPC 对于当前的游戏环境及自身状态的认识和理解。

3.1.3 决策模块

这部分是 NPC 进行决策的核心部件,包含 3 个子模块,反应器、规划器及学习器。反应器将针对一些需要立即作出反应的状态产生原子动作即 Gamebots2004 可以接收的命令格式,如发现"周围敌人人数很多"时需要马上"逃跑"。规划器负责将高层目标逐层分解为低层的子任务,直至将其分解为原子动作。由此可见,HUNTBot 具有混合式结构。学习器根据任务的分解及执行情况学习到一些效果很好的动作序列,在遇到类似情况时可以直接执行,而不需再次规划。学习器根据学习任务的不同采取不同的学习算法,如对"夺旗"等团队行为采用强化学习,对于"拿到固定地点的物品"这一底层行为采用神经网络训练的方法学习。NPC 可以根据 NPC 对当前世界的认知进行规划,同时也可以根据辅助决策模块中对未来状态的认知(预测模块)以及对社会角色的认知(团队协议)进行规划。

3.1.4 行动模块

NPC 往往会对多个目标进行规划。如在对消灭敌人这一目标进行规划时,也有可能同时对补充弹药这一目标进行规划。这时会产生多个动作序列。因此需要行动模块对决策模块产生的各种命令排序,即按照动作之间的制约关系制定动作的执行次序。按照原子动作的性质将其划分在武器选择(如 LoadWeapon)、移动(如 Run)、攻击(如 Fire)及通信(如 SendMessage)等类别中。同一类别中的动作互斥,如移动中的 run 和 goto 互斥,不能同时执行。而不同类别中的动作有先后关系,如武器类的动作应先于攻击类的动作,如 LoadWeapon(装载武器)应先于 Fire(射击)的执行。而考虑到完成当前任务总是优于和其它 NPC 通信,通信类动作的优先级最低。那些无明显制约关系的动作则可并发执行,这样能够明显提高 NPC 动作的执行效率。

3.1.5 辅助决策模块

人类玩家的高层智能往往表现为对未来的预知,以及对自身社会角色的认知。辅助决策模块正是保证 NPC 也具有类似智能行为的基础。它包含两个子模块,预测模块根据对敌人的建模以及运动力学等知识对 NPC 的认知状态在未来可能发生的变化进行预测,如根据敌人的移动路径和当前的地图判断其未来的位置;团队协议模块指出 Agent 在团队中扮演的角色和应承担的任务,这部分信息预先写好,可通过盟友之间的通信进行更新。因此,辅助决策模块将 NPC 对未来的认知和对社会角色的认知作为 NPC 进行决策的辅助信息提供给决策模块。

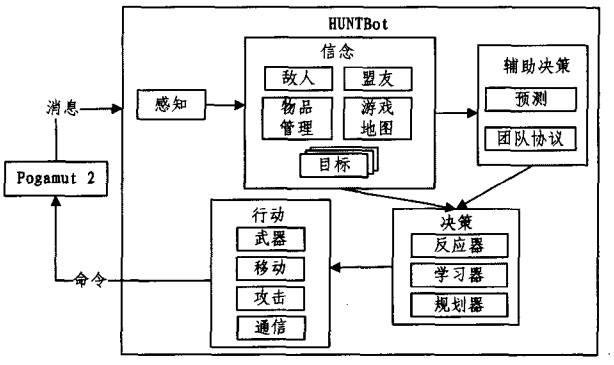

图 2 HUNTBot 的结构

由此可见，我们设计的 HUNTBot 的结构具有以下特点：(1)灵活性。混合式结构保证了 NPC 既能对环境迅速做出反应，又能对目标进行规划；(2)具有社会性。NPC 的信念集中有对盟友的认知，决策时受到团队协议的影响，并能与盟友进行通信。(3)学习能力。决策模块中的学习器保证了 NPC 具有对行为和规划进行学习的能力。(4)预测能力。预测模块使 NPC 具有对未来世界状态的认知，从而可以辅助其决策。

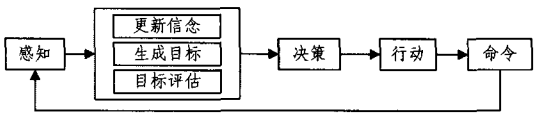

图 3　HUNTBot 的处理流程

3.2　HUNTBot 的处理流程

HUNTBot 的一个完整的处理流程如图 3 所示，包含以下 5 个步骤：

Step1 感知。NPC 接收感知器发来的各种信息。其中包含视觉信息、听觉信息，以及自身的状态，如武器装备、生命值、位置、方向、速度等等。

Step2 信念更新、生成目标、目标评估。包含以下几个部分：

①信念更新。将感知阶段接收到的信息按类别加入信念集中，对当前信念集进行更新。

②生成目标。加入新的信念可能产生任务相关的数据(目标)。例如，加入“看到有用武器”的信念，会产生“获得武器”这一目标。

③目标评估。信念集内可能产生多个目标。例如，“获得武器”和“躲避攻击”目标同时出现。此时需要为候选目标排序。如“躲避攻击”优先级高于“获得武器”。

Step3 决策。为信念集中优先级最高的目标(如“躲避攻击”)进行规划，产生动作序列。如“找到隐蔽处—到达隐蔽处—蹲下”。若此时有紧急事件需要处理，则触发反应式动作。

Step4 行动。将决策产生的动作序列按照相互之间的制约关系排序，以使动作协调运行。

Step5 命令。将排好序的动作以 Gamebots2004 认可的格式输出。

3.3　HUNTBot 的实验例

图 4　HUNTBot 的社会性

我们在 Pogamut 2 上实现了 HUNTBot，运行结果显示使用其构建的 NPC 兼具反应性和思考性，并具备社会性和学习能力。图 4 显示当 NPC 收到其他盟友发送的消息，导致信念集中出现 SquadAssmble(集合队伍)这一信念时，立即触发其反应式行为生成命令 run(跑动)，跑向队伍集合地。图 5 显示当 Agent 发现敌人时，具有目标 EliminateEnemy(消灭敌人)，由于曾经学习到完成这一目标所规划的行动序列，因此直接执行命令序列中包含的 Goto(跑至某一点)和 Fire(射击)。

图 5　HUNTBot 的学习能力

结束语　大型的电脑游戏要求 NPC 具备更加智能的行为，从而很好地适应环境的变化。我们针对 FPS 游戏——“虚幻竞技场”设计了一种 NPC 结构——HUNTBot。在 Pogamut 2 平台上的初步实验表明，HUNTBot 既能对环境变化作出即时反应，又能根据目标进行长期规划，并具有团队合作和学习能力，在游戏环境中表现出较好的性能。在今后的研究中，我们会考虑将实时规划技术引入 HUNTBot，使规划过程更适应实时性要求较高的游戏环境，并考虑采用强化学习技术对团队行为进行学习，使 NPC 的智能行为更加接近人类玩家。

参 考 文 献

[1] Champandard A J. AI game development. Indianapolis: New Riders Press, 2003

[2] Nareyek A. AI in Computer Games. ACM Queue, 2004, 1(10): 58-65

[3] Magerko B, Laird J E, Assanie M, et al. AI Characters and Directors for Interactive Computer Games. AAAI, 2004: 877-883

[4] Orkin J. Agent Architecture Considerations for Real-time Planning in Games. AIIDE, 2005: 105-111

[5] Burkert O, Kadlec R, Gemrot J, et al. Towards Fast Prototyping of IVAs Behavior: Pogamut 2. IVA, 2007: 362-363

[6] 王皓，罗文杰，高阳，等. 一种基于 Agent 的智能游戏开发平台及其 NPC 设计. 计算机科学，2007，34(9A)：66-69

[7] Kaminka G A, Veloso M M, Schaffer S, et al. GameBots: a flexible test bed for multiagent team research. Commun. ACM, 2002, 45(1): 43-45

[8] Champandard A J. Flexible embodied agent architecture. http://sourceforge.net/projects/fear/

(上接第 283 页)

参 考 文 献

[1] Goodenough J B, Gerhert S L. Toward a theory of test data selection[J]. IEEE Transactions on Software Engineering, 1975 (SE-3): 156-173

[2] 李秋英，陆民燕，阮镰. 软件可靠性测试充分性问题的理论研究. 北京航空航天大学学报，2003，29(4)：312-316

[3] 李秋英，阮镰，刘斌. 软件可靠性测试充分性研究. 测控技术，2003，22(11)：49-52

[4] 沈升源，陈丽容，汤铭端. 基于统计覆盖测试技术的软件测试充分性研究. 系统工程与电子科技，2004，26(6)

[5] Ohba M. Software reliability analysis models. IBM Journal of Research and Development, 1984, 28(4): 428-443

[6] Huang Chin - Yu. Performance analysis of software reliability growth models with testing-effort and change-point. The Journal of Systems and Software, 2005, 76: 181-194