前　言

机器人竞赛是近年来国际上迅速开展起来的一种高科技对抗活动，它涉及到人工智能、智能控制、机器人学、通讯传感、视觉技术以及制造材料技术等多领域的前沿研究和技术融合。它集高科技、娱乐和比赛于一体，可以极大培养广大学生的对前沿学科的兴趣和技术上的创造性。目前国际上推出了不同类型的机器人比赛，如机器人足球、机器人灭火、机器人相扑、机器人投篮以及机器人舞蹈和机器人越障等比赛。其中机器人足球赛是其中最为引人注目。

目前国际上比较流行的是 2 大机器人足球竞赛-Robocup 和 FIRA，其中 RoboCup 机器人足球世界杯赛及学术大会（The Robot World Cup Soccer Games and Conferences）是国际上级别最高、规模最大、影响最广泛的机器人足球赛事和学术会议，每年举办一次。同样，在中国自从 1999 年在重庆举办以来每年都相应举办的国内比赛和学术会议，通过这些比赛和研讨可以促进足球机器人技术的发展和竞技水平的提高。

为了吸引更多的人参与到这项活动当中来，开展 Robocup 程序设计的学习和研究必不可少。而仿真组的比赛和投入较小，但它的许多成果可以直接借鉴到其他比赛当中，因此选择这样的比赛类别作为我们机器人足球程序设计的切入点具有相当的可行性。为更多的人能掌握怎样组织出一支完整的 Robocup 仿真球队，我们编写了这本教材。

该教材首先介绍了机器人足球赛概况，在结合对 Agent 的研究基础上面，从理论上面介绍了比赛中的球员和球队的表示。然后介绍了 Robocup 仿真比赛的一些规则，结合 robocup2003 世界冠军队-UVA Trilearn 详细的介绍了球队的设计过程。最后介绍了球员底层动作的学习以及球队的整体策略的一些情况。其中 UVA Trilearn 球队分析分别由合肥工业大学 2002、2003 年 Robocup 机器人足球队成员安竹林、庞博、曹力、周晋等同学组织编写的，在这里表示由衷的感谢。

该教材内容分赴，材料翔实，既适合于初次参加 robocup 机器人足球仿真比赛的入门者，对有比赛经历者也有一定的参考和借鉴作用。

方 宝 富

2004 年 2 月

第一章 机器人足球简介

1.1Agent 技术概论

1.1.1 智能 Agent

Agent 的概念最早可以追溯到 1977 年由 Hewitt 提出的演员模型，在该模型中给出了一些“演员”——一组具有知我包含、相互作用和并行执行的对象。以后的研究中又把 Agent 看成一个具有特别技能的个体。后来又提出了软件 Agent 的概念，认为软件 Agent 是具有自主性和协作性的实体，它能够帮助用户完成一定的任务。还有些研究者认为 Agent 是驻留在某一环境下能够持续、自主地发挥作用，满足任务/目标驱动性、社会性、主动性等特征的计算主体。

由上面可见，agent 的定义和表现格不相同。实际上，要想在对形成 Agent 一个比较统一的定义还是不太容易的。所以更多的研究者把关注放在对 Agent 的特性的研究上面。现在对 Agent 的特性的研究中，最为经典和广为人接受的是 Wooldridge 等人提出的有关 Agent 的“强定义”和“弱定义”，认为一个 Agent 的最基本的特性应该包含反应性、自主性/自制性、面向目标性和社会性，然后根据其应用情况可以拥有其他特性。Agent 可以拥有的其他特性包括：移动性、自适应性、反应性、理性、持续性或时间连续性、自启动和自利等特性。还有一些人从 Agent 的精神状态出发，对 Agent 的特性进行了更深入的研究。

从建造 Agent 的角度出发，单个 Agent 的结构通常分为思考型 Agent、反应型 Agent、混合型 Agent。下面分别介绍一下这些类型的 Agent。

- 思考型 Agent（deliberate Agent）

建造 Agent 的经典方法是将其看作是一种特殊的知识系统，即通过符号人工智能的方法来实现 Agent 的表示和推理，这就是所谓的思考型 Agent。

思考型 Agent 的最大特点就是将 Agent 看作是一种意识系统。人们设计的基于 Agent 系统的目的之一是把它们作为人类个体或社会行为的智能代理，那么 Agent 就应该能模拟或表现出被代理者具有的所谓意识态度，如信念、愿望、意图、目标、承诺、责任等。

在这方面，Bratman 最早提出用信念、愿望、意图来表示 Agent。他从认知的角度来描述信念，认为信念是 Agent 对当前世界状况以及为达到某种效果所可能采取的行为路线的估计；从情感的角度来描述愿望，认为愿望是描述 Agent 对未来世界状态以及对所可能采取的行为路线的喜好；从意动方面来描述意图，认为目标是愿望的子集，但没有采取具体行动的承诺，如果某一或一些目标得到了承诺，这些目标就是意图；这也是最初的 BDI 模型。但对 BDI 模型做出最大贡献的当首推澳大利亚的 Rao 和 Georgeff，他们提出了一系列用来描

述 Agent 意识的 BDI 逻辑，分别用 3 个模态算子来刻画信念、愿望和意图。

- 反应型 Agent（reactive Agent）

由于符号人工智能的特点和种种限制，给思考型 Agent 带来了许多尚未解决并且很难解决、甚至无法解决的问题。因此导致了反应型 Agent 的出现。反应型 Agent 认为，Agent 的智能应该取决于感知和行动，从而提出 Agent 智能行为的“感知—动作”模型。此时的 Agent 不需要知识，不需要表示，也不需要推理，Agent 可以向人类一样逐步进化，Agent 的行为只能在现实世界与周围环境的交互作用中表现出来。

这方面的杰出代表当推 MIT 的 Brooks，他提出了子前提结构（subsumption architecture），该结构是由用于完成任务的行为（behaviors）来构成的分层结构，这些结构相互竞争以获得对机器人的控制。

- 混合型 Agent（hybrid Agent）

反应型 Agent 能及时而快速的响应外来信息和环境的变化，但其智能性较低，也缺乏足够的灵活性。思考型 Agent 具有较高的智能，但无法对环境的变化做出快速的响应，而且执行效率也较低。混合型 Agent 综合了两者的优点，具有较强的灵活性和快速的响应性。

混合型 Agent 通常被设计成至少有两层的层次结构。高层是一个包含有符号世界模型的认知层，进行 Agent 整体规划与设计；低层是一个能快速响应和处理环境中突发事件的反应层。一般反应层具有较高的优先级。

这种结构的典型实例是过程推理系统（Procedural Reasoning System，简称 PRS），它是一个在动态环境中推理和执行任务的 BDI 系统。

以上是构造 Agent 的 3 种结构，其中思考型 Agent 说采用的是 Sign AI，所以得到了大多数 DAI 研究人员的青睐，反应型 Agent 尚处初级阶段。而混合型 Agent 由于其优点已经成为当前的研究热点。

1.1.2 多 agent 系统（MAS）

MAS 的协作求解问题的能力超过单个的 Agent 是 MAS 产生的最直接原因，导致 MAS 研究逐渐兴起的其他原因还包括：与已有系统或软件的互操作；求解那些数据、能力和控制具有分布特性的问题以及提高系统的效率和鲁棒性等；与单个 Agent 相比，MAS 具有以下特点：每个成员 Agent 仅拥有不完全的信息和问题求解能力（故其观点是有限的），不存在全局控制，数据是分散或分布的，计算过程是异步、并发或并行的。

MAS 的研究历史最早可以追溯到 80 年代中期的 Actors 模型，接着是 Davis 和 Smith 提出的合同网协议。合同网协议至今仍被认为是关于通信、MAS 协商研究的经典工具。目前，MAS 研究的主要方面包括：MAS 论、多 Agent 协商和多 Agent 规划。其他比较热门的 MAS 研究还包括 MAS 在 Internet 上的应用、移动 Agent 系统、电子商务、基于经济学或市场学的 MAS 等。移动 Agent 可以自主地在网络上从一台主机移动到另一台主机上连续运行，这种灵活性为网络环境，尤其是 Internet 环境下的应用程序提供了很多潜在的优点。目前，已

经有不少较为成功的 Agent 移动系统。

- MAS 与联合意图

对于 MAS，除了要考虑上述因素外，还要考虑到多个 Agent 意识态度之间的交互问题，这是 MAS 理论研究的重要部分之一。

能够对环境中其他 Agent 的意识态度进行推理是 Agent 间共存、竞争或协作的要求 Agent 之间的协同、协商和协作行为是在其各种精神状态的支配和控制下才产生、进行、和完成的。与共享精神状态相关的理论主要涉及到相互信念、联合目标和联合意图等概念。其中以联合意图为代表，在多 Agent 环境下，相互信念即是所谓的公共知识，它与多 Agent 通信密切相关。

从目前的研究现状来看，对联合意图的研究不外乎是以下两种观点：（1）宏观点，即在每个个体（Agent）外存在一种联合意图控制整个组织或社会（MAS）的协作行为；（2）微观点，即每个成员的意识态度导致个体动一起完成协作目标。Haddadi 提出的关于联合意图的社会或组织观点和个体观点即分别属于上述两种观点。Jennings 提出的关于联合意图的承诺和公约也分别对应于上述两种观点。Bratman 和 Rao 等人提出联合意图的要求，则只限于微观点，他们认为每个团队成员都对联合行为做出承诺，联合意图是实现共享联合目标的方法，成员间要相互承诺，并及时把各自动作的成功或失败通知给其他成员。

下面给出一些与联合意图相关的典型的形式化工作。Rao 和 Georgeff 等人基于单个 Agent 系统的 BDI 模型，递规定义社会 Agent 的概念，引入了设社会规划。Wooldridge 和 Jennings 已承诺和公约概念为基础， 用分支程序逻辑建立了协作问题求解系统的形式的形式化框架。Haddadi 基于 BDI 逻辑给出了一个协作系统的形式化模型。并且还说明了该模型在 COSY 系统中的应用。

- 多 Agent 协商

协商是 MAS 实现协同、协作、冲突消解和矛盾处理的关键环节，有关多 Agent 的关键技术可以概括为协商协议、协商策略和协商处理三方面的内容。

协商协议的主要研究内容是 Agent 通信语言(ALC)的定义、表示、处理和语义解释。Müller 认为协商协议的最简单形式是如下一条协商通信消息：(〈协商原语〉，〈消息内容〉)。其中协商原语即消息类型，它的定义通常基于言语行为理论。消息内容除包含消息的发送者、接收者、消息信号、发送时间等固定信息外，还包括与协商应用的具体领域有关的信息描述。协商协议形式化表示方法通常由三种：巴科斯范式表示、有限自动机表示和语义表示。巴科斯范式表示具有简洁、明了的特点，是最常用的表示方法。采用纯语义表示的协商工作不多，研究者更多的是给除非形式化的语义解释。Agent 通信语言中最著名的是 KQML 语言。协商策略是 Agent 决策和选择协商协议和通信消息的策略。协商策略包括一组与协商协议相对应的元级协商策略和策略的选择机制或函数两部分内容。协商策略基本上可以分为五类：单方让步、竞争型策略、协作型策略、破坏协商和拖延协商。后两类策略显然不利于推进协商进程，而单方让步策略只在协商陷入僵局或协商不再有意义时才起作用，所以只有竞争型和

协作型策略才是有意义的。竞争型策略一般是指协商参与者坚持自己的立场，在协商过程中表现出竞争行为，是协商结果向有利于自身利益方向发展。合同网协商模型、基于策论的协商过程等都属于此类。协作型策略则是指协商各方都从系统利益出发，在协商过程中相互合作，他们采取的协商对策有利于寻找相互能接受的协商结果。采用协作型策略的协商过程包括部分全局规划、FA/C 等。Agent 应动态、智能地选择适宜的协商策略，从而在系统运行的不同时刻表现出不同的竞争或协作行为。策略选择的通常方法是以具影响协商的多方面因素，给出适宜的策略选择函数。策略选择函数可能包括效用函数、比较匹配函数、兴趣或爱好函数等几种。策略选择函数的设计除了要综合考虑影响协商的各种因素外，还要考虑冲突综合消解以及与应用领域有关的属性等。

协商处理包括协商算法和系统分析两部分内容。协商算法用于描述 Agent 在协商过程中的行为，包括通信、决策、规划和知识库操作等。系统分析的任务是分析和评价 Agent 协商的行为和性能，回答协商过程中的求解质量、算法效率以及系统的公平性和死锁等问题。协商协议主要处理协商过程中 Agent 间的交互，协商策略主要涉及 Agent 内的决策和控制过程，而协商处理则侧重于对单个 Agent 和多个 Agent 协商社会的整体协商行为的描述和分析。前两者描述了多 Agent 协商的微观方面，后者刻画了多 Agent 协商的宏观层。

有关 MAS 协商的典型工作有下面几个。Sycara 以劳资协商为背景对非协作类的多 Agent 相互作用进行了研究，给出基于实例推理和多属性效用优化理论的“劝说性辩论”模型。Crosz 等人在会谈理解研究中，给出支持人机交互通行的形式化模型。Wellman 等人把面向视场方法用于设计 Agent 间的协调过程，提出所谓基于一般平衡理论的“面向市场程序设计”机制。还有基于经济学理论、对策论和 Nash 平衡理论的多 Agent 协商研究等。

- 多 Agent 规划

规划是连接精神状态与动作执行之间的桥梁，有关动作和规划的研究一直是 Agent 研究的活跃领域。MAS 中的规划与经典规划不同，属于适应性规划，需要反映出环境的持续变化。

目前对 MAS 中规划的研究主要从如下两个不同的角度进行：（1）将规划看作是一种可以在世界状态间转换的抽象结构，典型的如与或图；（2）将规划看作是一类复杂的 Agent 精神状态。这两种方法都在一定程度上降低了经典规划中解空间搜索的代价。从而有效地指导了资源受限 Agent 的决策过程。其中第一种方法应用更广。常用的方法是将 Agent 的规划库定义为一个与或图结构，其中的每一条规划包括以下四个部分：（1）规划目标，这是规划的点火条件，表示该条规划能达到的目标；（2）规划前提，表示该规划被执行前必须满足的环境或状态条件；（3）规划体；是规划的程序部分由规划序列和规划子目标组成；（4）规划结果，表示执行规划后对环境后状态的更新结果。

如何在自私的多 Agent 动态环境中实现 Agent 的灵活通行和动作执行，是当前 MAS 规划的热点。目前，多数 MAS 规划方法存在以下不足之处：多 Agent 规划不太适合动态变化的环境；很多型式化较好的理论模型与实际距离太远；如何更好地解决多 Agent 规划中的资

源冲突；如何在多 Agent 规划中引入质量因素等等。这些方面是现在、乃至将来 Agent 规划研究的方向和趋势。

1.2.机器人足球概况

机器人足球赛是由硬件或仿真机器人进行的足球赛，比赛规则与人类正规的足球赛类似。硬件机器人足球队的研制涉及计算机、自动控制、传感与感知融合、无线通讯、精密机械和仿生材料等众多学科的前沿研究与综合集成。仿真机器人足球赛在标准软件平台上进行，平台设计充分体现了控制、通讯、传感和人体机能等方面的实际限制，使仿真球队程序易于转化为硬件球队的控制软件。仿真机器人足球的研究重点是球队的高级功能，包括动态不确定环境中的多主体合作、实时推理－规划－决策、机器学习和策略获取等当前人工智能的热点问题。概括地说，机器人足球是以体育竞赛为载体的前沿科研竞争和高科技对抗，是培养信息－自动化科技人才的重要手段，同时也是展示高科技进展的生动窗口和促进科技成果实用化和产业化的新途径。

1.2.1 历史发端

机器人足球的最初想法由加拿大不列颠哥伦比亚大学的 Alan Mackworth 教授于 1992 年正式提出。日本学者立即对这一想法进行了系统的调研和可行性分析。1993 年，Minoru Asada（浅田埝）、Hiroaki Kitano（北野宏明）和 Yasuo Kuniyoshi 等著名学者创办了 RoboCup 机器人足球世界杯赛（Robot world cup soccer games，简称 RoboCup）。

与此同时，一些研究人员开始将机器人足球作为研究课题。隶属于日本政府的电子技术实验室（ETL）的 Itsuki Noda（松原仁）以机器人足球为背景展开多主体系统的研究，日本大坂大学的浅田埝、美国卡内基－梅隆大学的 Veloso 等也开展了同类工作。

1997 年，在国际最权威的人工智能系列学术大会—第 15 届国际人工智能联合大会（The 15th International Joint Conference on Artificial Intelligence，简称 IJCAI-97）上，机器人足球被正式列为人工智能的一项挑战。至此，机器人足球成为人工智能和机器人学新的标准问题。

1.2.2 历届杯赛情况

第一届 RoboCup 机器人足球世界杯赛于 1997 年 8 月 25 日在日本名古屋与 IJCAI-97 联合举行。来自美、欧、日、澳的 40 多支球队参赛，观众达 5000 余人。第二届杯赛于 1998 年 7 月 4 日至 8 日在法国巴黎与第 16 届世界杯足球赛同时举行（当年没有 IJCAI 大会），参赛队达 60 多支。1999 年 7 月 28 日至 8 月 4 日，第三届 RoboCup 世界杯赛及学术大会在瑞典斯德哥尔摩与 IJCAI-99 联合举行，参赛队多达 90 余支。2000 年 8 月 25 日至 9 月 3 日，第四届杯赛及学术大学在澳大利亚墨尔本举行，正式参赛队首次突破 100 大关，达 104 支。一些著名的大学（如美国 CMU 和 Cornell 等，德国 Humboldt）、国立研究院（如美国 NASA）和大公司（如日本 SONY）均参与了相关的活动，中国首次参赛，中科大首次参赛并且取得

第九名的不俗战绩。2001 年 8 月 2 日至 10 日，第五届杯赛和学术会议在美国的西雅图举行，约有 100 多支球队参加。清华大学仿真组、中国科大仿真组及四腿组参赛，清华大学仿真组获得冠军，中国科大队进入双 8。第六届 RoboCup 于 2002 年 6 月 19－25 日在 日本福冈举行，更多的球队参加。清华大学、北京理工大学仿真组分获冠亚军，科大的仿真和四腿组也取得了不错的成绩。第七届 RoboCup 2003 年 7 月 2－11 日在 意大利帕多瓦举行，中国的科大、清华、浙大、北理和上海交通大学参加了比赛并取得了不错的成绩。是 RoboCup 有史以来最盛大的比赛。比赛项目也是从最初的仿真组、小型组、中型组到现在的仿真组、小型组、中型组、Sony 四腿组、机器人营救、中小学生初级组以及类人机器人、舞蹈机器人表演赛（即将成为正式比赛）。

1.2.3 组织机构

国际 RoboCup 联合会是世界上规模最大的、占主导地位的机器人足球国际组织，总部设在瑞士，现有成员国近 40 个。联合会现任主席是国际著名科学家、在 IJCAI-93 大会上获得国际人工智能最高奖—“计算机与思维”大奖的北野宏明。联合会负责世界范围的学术活动和竞赛，包括每年一届的世界杯赛和学术研讨会，并为相关的本科生和研究生教育提供支持（教材、教学软件等）。

除国际 RoboCup 联合会之外，还有其他一些国际组织。其中较大的一个是 FIRA，该组织总部设在韩国大田，现有成员国 20 多个，每年举办一次国际性比赛。FIRA 与 RoboCup 的主要区别之一是采用不同的技术规范：FIRA 允许一支球队采用传统的集中控制方式，相当于一支球队中的全体队员受同一个大脑的控制；而 RoboCup 要求必须采用分布式控制方式，相当于每个队员有自己的大脑，因而是一个独立的”主体”。

1.3 MAS 与 Robocup

在第 15 届国际人工智能联合大会上，由 Kitano, Veloso 和 Tambe 等来自美、日、瑞典的 9 位国际著名或知名学者联合发表重要论文”The RoboCup synthetic agent challenge 97”，系统阐述了机器人足球的研究意义、目标、阶段设想、近期主要内容和评价原则。概括的说，过去 50 年中人工智能研究的主要问题是”单主体静态可预测环境中的问题求解”，其标准问题是国际象棋人－机对抗赛；未来 50 年中，人工智能的主要问题是”多主体动态不可预测环境中的问题求解”，其标准问题是足球的机－机对抗赛和人－机对抗赛。从科学研究的观点看，无论是现实世界中的智能机器人或机器人团队（如家用机器人和军用机器人团队），还是网络空间中的软件自主体（如用于网络计算和电子商务的各种自主软件以及它们组成的”联盟”），都可以抽象为具有自主性、社会性、反应性和能动性的”自主体”（agents）。由这些自主体以及相关的人构成的多主体系统（multi-agent systems），是未来物理和信息世界的一个缩影。其基本问题是主体（包括人）之间的协调，可细分为自主体设计、多主体体系结构、

自主体合作和通讯、自动推理、规划、机器学习与知识获取、认识建模、系统生态和进化等一系列专题。这些专题有的是新提出的（如”合作”），有的是过去未能彻底解决并在新的条件下更加复杂化的（如机器学习）。这些问题不解决，未来社会所需的一些关键性技术就无法得到。值得注意的是，上述一系列问题中的大多数都在机器人足球中得到了集中的体现。在这个意义下，将机器人足球作为未来人工智能和机器人学的标准问题是十分恰当的；这主要是由于机器人足球具有以下特点：

(1)典型性。如上所述，RoboCup 机器人足球队的研制涉及当前人工智能研究的大多数主要热点，因而构成一个典型问题。

(2)可行性。多主体系统的绝大多数实际背景十分复杂，以致研究人员在目前的条件下难以把握，无法兼顾具体细节分析与基本问题探索。而在机器人足球中则较易兼顾二者，易于深入。

(3)客观性。比赛提供了一种实验平台和评价各种理论与技术的客观方法，便于研究者的”自我观察”和相互交流。

(4)综合性。在以往的研究中，各种技术通常被分别开发和考察，综合集成工作一般由面向最终用户的应用部门来完成，这种方式不利于相关技术在更高层次上的衔接和在更深层次上的创新。机器人足球是第一个深层的”综合平台”。

因此，开展机器人足球研究是人工智能从基础理论走向实际应用的一个战略性步骤。所以我们在研究 MAS 的时候就选择了机器人足球作为我们的研究平台。我们在研究 Robocup 机器人足球的时候也是从比较简单不需要其他因数譬如说机械、图像处理的 robocup 仿真组比赛（Simulation Game）作为我们的切入点。下面简要介绍一下 robocup 仿真组比赛是如何和 MAS 结合在一起的。

仿真组比赛采用 Soccer Server[1]作为一个标准比赛平台。Soccer Server 是一个允许竞赛者使用支持 UDP/IP 的任何程序语言进行仿真足球比赛的系统，整个系统按照 100 毫秒的周期运转，比赛以 Client/Server 方式进行。Server，即 Soccer Server，提供了一个虚拟的足球场地，并对比赛双方的全部队员和足球的移动进行仿真。Client，相当于球员的大脑，指挥球员的运动，每个 Client 控制一个球员。Server 和 Client 之间的通信是通过 UDP/IP 协议进行的。通过这种方式，Server 向 Client 发送有关的赛场信息（如视觉、听觉信息），Client 端通过对这些信息进行分析，产生相应的控制指令，并发送到 Server，以控制相应的队员。要赢得一场足球比赛，单靠个人能力是不可能的， 必须有全体队员的相互配合与协作（ 即 teamwork）；同样，要提高一个多主体系统的性能也需要各个主体之间的协调与配合。因此，在这个系统中，提供了一个与实际环境很接近的多智能体实时环境，在其中既有合作又有对抗。研究者可以在这套平台上设计自己的球队，用以评价各种理论、算法及策略的可行性。

实际上，robocup 其他各组别的比赛也都是类似的 MAS 系统。

第二章　Soccer Server

机器人足球世界杯比赛（Robocup）仿真组比赛是在一个标准的计算机环境内进行的。比赛规则基本上与国际足球联合会的比赛规则一致，对于某些特殊的部分，在手册第二章会有详细论述。比赛的方式是由 Robocup 委员会提供标准的 Soccerserver 系统，各参赛队编写各自的 CLIENT 程序，模拟实际足球队员参加比赛。

Soccerserver 是一个允许竞赛者使用各种程序语言进行仿真足球比赛的系统。比赛以 Client/Server 方式进行。Server，即 Soccerserver，提供了一个虚拟场地，并对比赛双方的全部队员和足球的移动进行仿真。Client，相当于球员的大脑，指挥球员的运动。Server 和 Client 之间的通信是通过 UDP/IP 协议进行的。所以，竞赛者可以使用支持 UDP/IP 的任何程序系统。

Soccerserver 包含两个程序：Soccerserver 和 Soccermonitor。Soccerserver 的工作是仿真足球和队员的移动、与 Client 进行通信、按照一定的规则控制游戏的进程。Soccermonitor 则负责利用 X window（或 windows 95）系统在 server 中显示虚拟场地。server 可以同时与多个 Soccermonitor 相连。因此，我们可以在多个显示器上同时显示比赛的情况。

Client 与 Server 之间都是通过 UDP/IP 协议进行信息交互的。通过这种方式，Client 发送指令去控制相应的队员，同时从 Server 端接受队员的传感器传回的信息。每个 Client 模块只允许控制一名球员。故竞赛者必须同时运行与比赛球员数目相等的 client。Client 之间的通讯必须通过 Soccerserver 来进行。Soccerserver 的一个目标就是对多智能体系统进行评价，而智能体之间的通讯效率是一个重要标准。竞赛者必须在此要求下实现对多个智能体的控制。

2.1　比赛过程及规则

2.1.1 Server 的获取和安装

Soccerserver 的源文件可以从如下地址中获得：

http://sserver.sourceforge.net/

目前，在网上可以找到 Soccerserver 的 UNIX 和 Windows95 两个版本，都是源码，分别需要利用 GCC 和 VC++5.0 编译成可执行代码。有关编译过程可参考 Soccerserver 的 Readme 文件。

本组委会提供全部有关的文件，包括源代码和编译好的可执行代码。

2.1.2 整场比赛的过程

由 Soccerserver 控制的比赛过程可分为如下步骤：

1 两队的全部队员通过 ***init*** 命令与 Soccerserver 一一连接。

2 当全部队员都准备好时，比赛裁判(commissary)用鼠标点取 Soccerserver 的 *kick-off* 按钮，上半场比赛开始。

3 上半场比赛为 5 分钟。当上半场比赛结束时，Soccerserver 暂停比赛。

4 中场休息为 5 分钟。在此期间，竞赛者可以修改 Client 程序。

5 在下半场比赛开始之前，每个 Client 需要使用 ***reconnect*** 命令与 Soccerserver 重新进行连接。

6 当全部 Client 准备就绪时，裁判(commissary)[1]点取 *kick-off* 按钮，开始下半场比赛。

7 下半场结束时，server 自动停止比赛。

8 如果比赛结果为平局，加时赛开始。加时赛采用金球法，即任一方球队进球，比赛立即结束，进球方获胜。

2.1.3 Soccerserver 控制的比赛规则：

1 进球(Goal)

当进球时，裁判(referee)[2]通过向全部 Client 发送广播式消息宣布进球。同时它登记比分、暂停比赛 5 秒钟、将球移回中点、并且将比赛模式切换为开球模式(*kick-off*)。当裁判暂停比赛时，球员必须回到自己半场，可以使用 ***move*** 命令。如果某个球员仍在对方半场，裁判(referee)会自动将此球员移回到自己半场，放置在随机位置。

2 开球(*kick-off*)

开球时全部球员必须在自己半场。如果某个球员仍在对方半场，裁判(referee)会将此球员移回到自己半场，放置在随机位置。

3 出界(Out of Field)

当球出界时，裁判(referee)将球移到一个合适位置(边线、角球区或罚球区)，并将比赛模式相应的切换为：边界球(*kick_in*)、角球(*corner_kick*)或球门球(*goal_kick*)。发角球时，裁判(referee)将球放置在角内(1m，1m)处。

4 清场(Clearance)

当守门员扑球后，或当前模式为：开球(*kick_off*)，边线发球(*throw_in*)，角球(*corner_kick*)，球门球(*goal_kick*)，或越位(*offside*)时，裁判(referee)将防守队员移出以球为圆心，半径为 9.15 的圆形区域，被移出的队员随机放置在圆形区域的周围。

5 比赛模式控制(Play_mode Control)

当比赛模式为开球(*kick_off*)、边线发球(t*hrow_in*)、角球(*corner_kick*)、球门球(*goal_kick*)时，在球接收到 *kick* 命令开始移动之后，裁判(referee)将模式切换到正常进行模式(*play_on*)。

6 中场休息时间和终场时间(Halftime and Time Up)

---

[1] 由比赛组委会指定人员。

[2] 存在于 Soccerserver 内部的自动裁判程序。

当上半场和下半场结束时，裁判(referee)自动终止比赛。每半场缺省的比赛时间为3000 个仿真周期，大约 5 分钟。如果下半场结束时为平局，会进行加时赛，直到一方进球为止。

2.1.4 需要人判断的规则

某些故意犯规动作，如故意阻挡等，很难由裁判(referee)自动判断，因为它与球员的意图有关。因此，Soccerserver 为通过人来判断这些犯规动作提供了一种方式。下面简单介绍一下此类犯规动作：

1 故意包围足球(Surrouding the ball)

2 故意用多名队员阻挡球(Blocking the goal with too many players)

3 故意持球(Not putting the ball into play)

4 故意阻挡其他队员的移动(Intentionally blocking the movement of other players)

5 守门员滥用“***catch***”命令，守门员不允许在罚球区内重复使用 ***kick*** 和 ***catch*** 命令

6 不允许向 server 发送过多命令，每个 Client 在一个仿真周期内不能发送超过 3 或 4 个命令。过多的命令会使 server 阻塞。

2.2 Soccerserver

主要介绍了 SoccerServer 的各个组成部分（对象），包括场地以及球场上面的对象以及球员相关的各种模型。

2.2.1 球场上的对象

图 2.1 是用 UML 实例图表示了 robocup 仿真组中的对象。

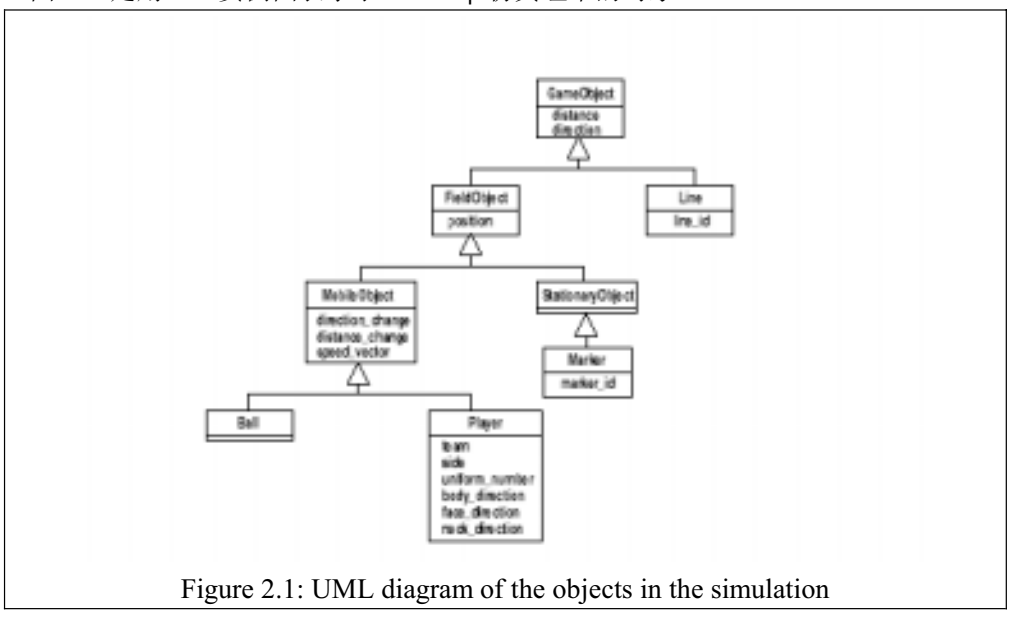


Figure 2.1: UML diagram of the objects in the simulation

2.2.2 场地和球员

仿真环境中足球场和其中的全部对象都是二维的。任何对象都没有高度的概念。比赛场地的尺寸为 *field_length*×*field_width*，球门的宽度为 *goal_width*，缺省值为 105（m）×68（m）

（单位是没有意义的），球门宽度为 14.64（m），是实际的两倍。实验证明，对于正常的宽度是很难进球的。

球员和球都使用圆圈来表示。动作模型是离散的(在一个仿真周期结束时全部的动作被执行一次)。每个仿真周期时间的长短是由参数 *simulator_step* 决定的。在每个仿真周期结束前，Soccerserver 接收所有 client 的命令，并执行命令，并利用当前场上对象(球员和球)的位置和速度信息计算出全部对象新的位置和速度信息。

2.2.3 对象的运动模型

在仿真周期内，对象的移动按如下公式进行计算：

（u_x^{t+1},u_y^{t+1}）=（v_x^t,v_y^t）+（a_x^t,a_y^t）：加速

（p_x^{t+1},p_y^{t+1}）=（p_x^t,p_y^t）+（u_x^{t+1},u_y^{t+1}）：移动

（v_x^{t+1},v_y^{t+1}）$=decay\times(u_x^{t+1},u_y^{t+1})$：衰减速率

（a_x^{t+1},a_y^{t+1}）=（0,0）：复位加速度

其中，（p_x^t,p_y^t）和（v_x^t,v_y^t）分别表示 *t* 时刻物体的位置和速度。*Decay* 是一个参数，分别由 *ball-decay* 和 *player-decay* 控制。（a_x^t,a_y^t）表示对象在 t 时刻的加速度，可以通过 ***dash***（针对队员）和 ***kick***（针对足球）的 *Power* 参数计算得到：

（a_x^t,a_y^t）= $Power\times(\cos(\theta^t),\sin(\theta^t))$

其中θ^t表示对象在 *t* 时刻的前进方向。如果对象为球员，他的方向通过由下式计算：

$\theta^t=\theta^t+Moment$

Moment 是 ***turn*** 命令的参数。对于足球，其方向的计算方法是：

$\theta_{ball}^t=\theta_{kic\,\mathrm{ker}}^t+Direction$

其中θ_{ball}^t和$\theta_{kic\,\mathrm{ker}}^t$表示球和踢球队员当前的方向，而 *Direction* 是 ***kick*** 命令中第二个参数。

（a_x^{t+1},a_y^{t+1}）表示是在 t+1 时刻的加速度，在现在的 Server 版本中复位为 0。

另外加入以下 2 种因素：

2.2.3.1 碰撞

如果在某个仿真周期结束时，两个对象发生重叠，则 server 自动将他们向后移动，直到不再重叠为止，然后将各自速度乘以-0.1。需要说明的是，只要在仿真周期结束时球和某队员不发生重叠，那么球就可以直接穿过该队员。

2.2.3.2 风和环境干扰

为了反映出实际比赛中球以及球员运动的不确定性，Soccerserver 在球与球员的移动及命令的参数中加入了干扰。

首先考虑移动，干扰是以如下方式加入的：

（u_x^{t+1},u_y^{t+1}）=（v_x^t,v_y^t）+（a_x^t,a_y^t）+（$\tilde{r}_{r\max},\tilde{r}_{r\max}$）

$\tilde{r}_{r\max}$为属于$[-\max,\max]$的随机数。*rmax* 的定义为：

$$rmax = rand \cdot \left| (v_x^t, v_y^t) \right|$$

rand 是 *player-rand* 和 *ball-rand* 的参数。

turn 命令中 *Power* 参数的干扰加入方式为：

$$Moment = (1 + \tilde{r}_{rand}) \cdot Moment$$

2.2.4　球员的感知信息

球员可以从 server 接受三种不同的感知信息：听觉、视觉和身体状态信息。听觉感知检测裁判，教练和其他球员发送的消息。视觉感知检测场上的可视信息，如球员当前可视范围内的对象的距离和方向。视觉信息很象一个传感器，可以“见到”在球员身后的附近对象。身体状态信息检测球员的当前物理状态，如球员自身的体力 stamina，速度 speed 和头颈角度 neck angle。

2.2.4.1 听觉模型

当某球员或裁判（referee）“说”消息（***say*** *Message*）时，附近的其他球员包括对方球员可以立即听到消息，没有延迟。他们以（***hear*** *Time Sender "Message"*）的形式听到消息。其中：

Time：前的仿真周期。

Sender：如果是其他球员发送的消息，那么是发送者的相当方向（*Direction*），否则就是下面的选项：

- self：发送者是自己本人。
- referee：裁判是发送者。
- online_coach_left 或者 online_coach_ringt：发送者是在线教练。

Message：消息内容。最长可以是say_msg_size 个字节。裁判发送的消息可以是：before_kick_off、kick_off_l、kick_off_r、kick_in_l、kick_in_r、corner_kick_l、corner_kick_r、goal_kick_l、goal_kick_r、free_kick_l、free_kik_r、offside_l、offside_r、play_on、half_time、time_up、extend、foul_Side_Unum、goal_Side_Point。

注意关于是哪个队员发的消息和他的距离是不知道的。

队员只有有限的通讯能力，只能听到一定距离之内的声音，此距离由 soccerserver 参数 *audio_cut_off_dist* 决定。同时队员在 *hear_decay* 个循环周期内只能听到 *hear_inc* 条消息。一般情况下，在 2 个循环周期内，当有多个队员说某消息时，一名队员只能接收一条，而失去了其他的消息。裁判（*referee*）所发的消息具有高的优先级，可以被全部队员接收到。

2.2.4.2 视觉模型

从 server 得到的听觉信息按如下格式：

（***see*** *Time ObjInfo ObjInfo …*）

Time：当前时间。

ObjInfo 表示了可视对象的信息。其格式为：

（ObjName Distance Direction DistChng DirChng BodyDir HeadDir）

ObjName =（*player Teamname Unum*）

|（*gola Side*）
|（*ball*）
|（*flag c*）
|（*flag* [*l* | *c* | *r*] [*t* | *b*]）
|（*flag p* [*l* | *r*] [*t* | *c* | *b*]）
|（*flag* [*t* | *b*] [*l* | *r*] [10 | 20 | 30 | 40 |50]）
|（*flag* [*l* | *r*] [*t* | *b*] [10 | 20 | 30]）
|（*flag* [*l* | *r* | *t* | *b*] 0）
|（*line* [*l* | *r* | *t* | *b*]）

Distance，*Direction* 表示目标的相对距离和相对方向。*DistChng* 和 *DirChng* 分别表示目标距离和方向的相对变化，*DistChng* 和 *DirChng* 不是精确值，只是一个粗略值。字母：“*l*、*r* 、*c*、 *t*、 *b*”分别表示了左、右、中心、上、下。“*p*”表示罚球区。详见图 2.2。

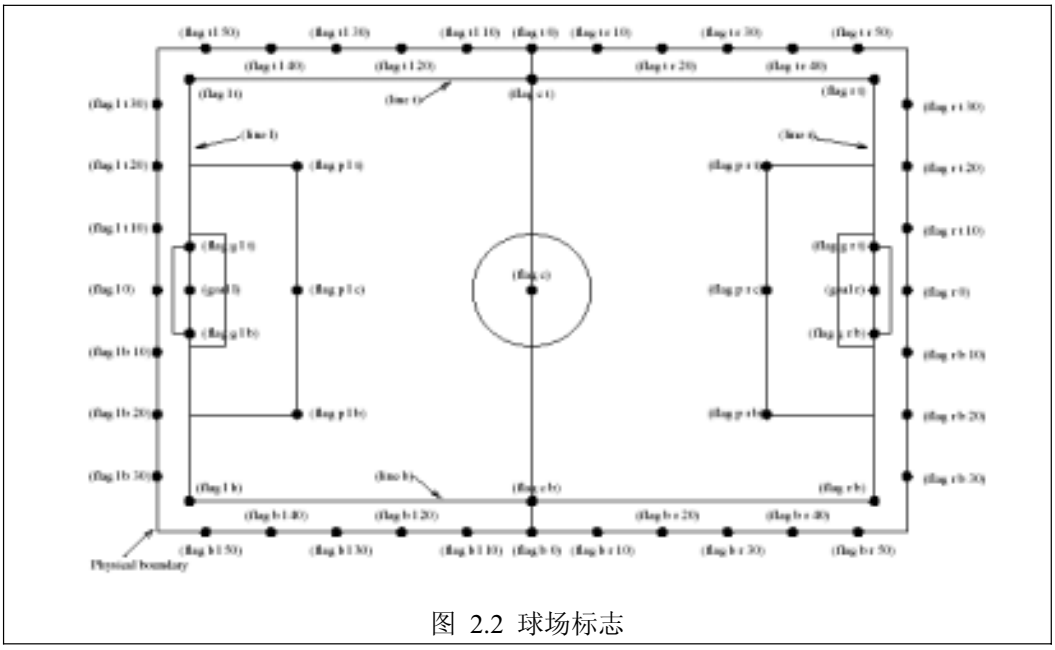


图 2.2 球场标志

Distance，*Direction*，*DistChng* 和 *DirChng* 按如下方式计算出来的：

$$P_{rx} = P_{xt} - P_{x0}$$

$$P_{ry} = P_{yt} - P_{y0}$$

$$v_{rx} = v_{xt} - P_{x0}$$

$$v_{ry} = v_{yt} - P_{y0}$$

$$Distance = \sqrt{P_{rx}^2 + P_{ry}^2}$$

$$Direction = \arctan(P_{ry} / P_{rx}) - a_0$$

$$e_{rx} = P_{rx} \ / \ Distance$$

$$e_{ry} = P_{ry} \ / \ Distance$$

$$DistChng = (v_{rx} * e_{rx}) + (v_{ry} * e_{ry})$$

$$DirChng = [(-(v_{rx} * e_{rx}) + (v_{ry} * e_{ry})) / Distance] * (180 / \pi)$$

其中(P_{xt}, P_{yt})是目标的绝对位置坐标，(P_{x0}, P_{y0})是接收视觉信息的队员自己本身的绝对坐标，(v_{xt}, v_{yt})是目标的绝对速度，(v_{x0}, v_{y0})队员自己的绝对速度。a_0是队员所面向的绝对方向。另外(P_{rx}, P_{ry})和(v_{rx}, v_{ry})表示目标的相对位置和相对速度。(e_{rx}, e_{ry})表示平行于相对位置向量的单位向量。如果被观察者是球员的话，才会有BodyDir 和HeadDir，分别是被观察球员相对观察者的身体和头部的相对角度。如果两个球员的身体都是相同的角度，那么BodyDir 就等于零。HeadDir 也一样。

视野范围

球员的可视部分依赖于几个因素。首先是server 上的参数sense_step 和visible_angle，决定了视觉信息的时间间隔，以及球员正常视角时的角度。现在使用的是150ms 和90°。

球员通过改变ViewWidth 和ViewQuality 也可以改变视觉信息的频率和质量。

view_frequency 和view_angle 由以下公式2.1和2.2计算得出：

$$view\_frequency = sense\_step * view\_quality\_factor * view\_width\_factor \quad (2.1)$$

$$view\_quality\_factor = \begin{cases} 1 & \text{当ViewQuality=high} \\ 0.5 & \text{当ViewQuality=low} \end{cases}$$

$$view\_width\_factor = \begin{cases} 2 & \text{当ViewWidth=narrow} \\ 1 & \text{当ViewWidth=normal} \\ 0.5 & \text{当ViewWidth=wide} \end{cases}$$

也就是说，视觉质量要求越高，视角要求越小，则发送时间间隔越大。

$$view\_angle = visible\_angle * view\_width\_factor \ (2.2)$$

$$view\_width\_factor = \begin{cases} 2 & \text{当ViewWidth= wide} \\ 1 & \text{当ViewWidth=normal} \\ 0.5 & \text{当ViewWidth=narrow} \end{cases}$$

球员能够“看到”在他领域内的对象（以visible_distance为半径的圆周）。如果某一对象在此范围内，但是不在球员的视野范围内，那么球员只知道对象的类型（足球，球员，球门或者是标记），而不知道对象的确切名字。就是说使用“B”，“P”，“G” 和“F”来作为对象的名字，而不是使用“b”，“p”,“g”和“f”。

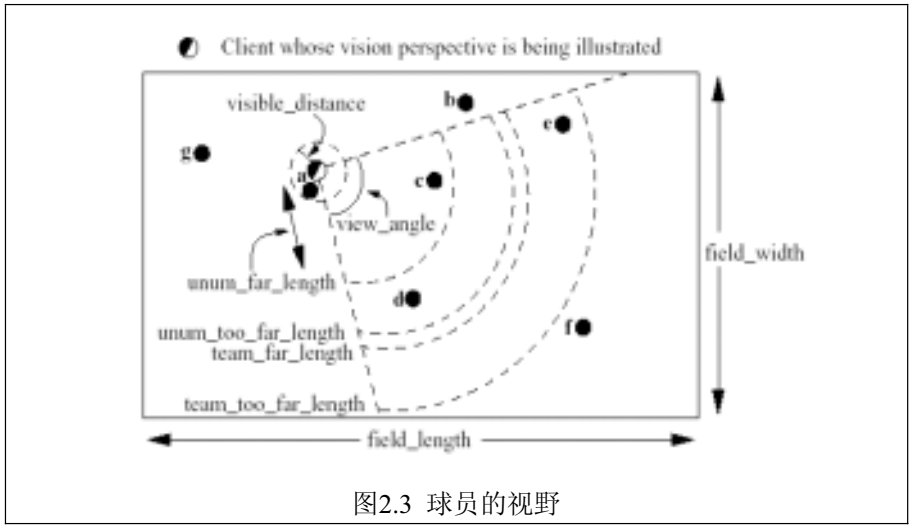


图2.3 球员的视野

图2.3表示了view_angle的意思。图中的观察球员由两个半圆组成，空心的半圆表示球员的前部。全黑的圆周代表场上的对象。只有在view_angle/2的角度范围内，而且在visible_distance的距离范围内的对象才能被看到。因此，对象b和g是不可见的，其他的对象都是可见的。对象f在观察者的正前方，它的角度被认为是0°。对象e会被认为是－40°，而对象d则会被认为是大约20°。

请参考图2.3 的标志，球员所获得的视觉信息和距离的相关程度很大。对于近距离球员，它既可以看到它所属的球队同时也可以看到他的球员号码。然而，随着距离的增加，首先消失的是球员的号码。然后距离的增加还会导致球员的队别也分不清楚了。在服务器端假定这些距离是：

$$unum\_far\_length \le unum\_too\_far\_length \le team\_far\_length \le team\_too\_far\_length$$

这里假定dist 是球员和自己的距离，那么：

- 如果 $\text{dist} \le unum\_far\_length$,那么球员号码和球队名称都可见。
- 如果 $unum\_far\_length \le \text{dist} \le unum\_too\_far\_length$，那么队名是可见的，但是队员号码有一定的概率看不到。这个概率根据dist 是线性的从1 到0 减少的。
- 如果 $\text{dist} \ge unum\_too\_far\_length$ ，那么球员号码是不可见的。
- 如果 $team\_far\_length \le \text{dist} \le team\_too\_far\_length$ ，那么队名也存在一定的概率不可见，概率是随着dist 的减少从1 到0 线性的递减的。
- 如果 $\text{dist} \ge team\_too\_far\_length$ ，那么队名是不可见的。

视觉感知噪声模型

为了在视觉感知数据中引入噪声，server 发送的信息被进行了量化处理。如：无论远处的目标是球还是球员，目标的距离值按如下方式进行量化：

$$d' = Quantize(\exp(Quantize(\log(d), quantize\_step), 0.1)$$

其中 d, d' 分别表示精确距离和相应的量化距离。且：

$$Quantize(V, Q) = ceiling（V / Q）* Q$$

这表示队员是不能知道远处物体的精确位置的。例如：当距离为 100.0 时，最大噪声可达到 10.0，但当距离在 10.0 之内时，噪声小于 1.0。

对于远处目标是旗或线的情况，距离值按如下公式量化：

$$d' = Quantize(\exp(Quantize(\log(d), quantize\_step\_l)), 0.1)$$

其中，$quantize\_step$ =0.1，　$quantize\_step\_l$ =0.01

2.4.4.3自身感知模型

自身感知返回球员当前的物理状态。每隔sense_body_step（目前使用100ms），就会自动的向球员发送自身感知信息。

自身感知消息的格式如下：

(sense_body Time
(view_mode ViewQuality ViewWidth)
(stamina Stamina Effort)
(speed AmountOfSpeed DirectionOfSpeed)
(head_angle HeadDirection)
(kick KickCount)
(dash DashCount)
(turn TurnCount)
(say SayCount)
(turn_neck TurnNeckCount)
(catch CatchCount)
(move MoveCount)
(change_view ChangeViewCount))

ViewQuality的取值是high或low。

ViewWidth取值是narrow，normal，wide。

AmountOfSpeed是球员速度的近似值。

DirectionOfSpeed是球员速度的近似方向。

HeadDirection是球员头部的相对方向。

变量Count是由server执行的对应命令的总量。如：DashCount＝134说明球员其时已经执行了134次dash命令

2.2.5　球员的动作模型

下面介绍一些球员可以发送给 sever 的各种动作，并且指出了那些动作是可以同时发送给 Server，那些是不能同时发送的。表 2.1 中列出了全部表示动作的命令。

(***catch***　*Direction*)

向 *Direction* 方向扑球。当球落入宽为 *goalie_catchable_w*，长为 *goalie_catchable_l* 的矩形内时，并且方向为 *Direction*，守门员可以扑到球。
(***turn*** *Moment*) 控制球员转身的角度。*Moment* 应在-180~180 之间。球员身体可以转过的角度随着球员的快速运动而减少。（*Moment* 的范围会改变。）
(***turn_neck*** *Angel*) 控制球员转脖子。其中 *Angel* 转过的角度。*Angel*∈[minneckmoment，maxneckmomnet]
(***move*** *X Y*) 移动球员到(*X Y*)。*X*∈[-52.5，52.5]，　*Y*∈[-34,34]
(***dash*** *Power*) 在球员所面对的方向上增加球员的速度。*Power* 应在-100~100 之间（范围可变）。
(***kick*** *Power Direction*) 以 *Power* 的力量向 *Direction* 方向踢球。条件是球要在 *kickable_area* 范围内。*Power*∈[-100，100]，*Direction*∈[-180，180]。
(***say*** *Message*) 向所有球员广播 *Message*。*Message* 会迅速被其他球员（包括对方球员）以听的方式接收。*Message* 是串长小于 10 个字节的字符串。可以包含字母，阿拉伯数字和符号“+ - */ _ . ( )”。球员的听力受距离限制。
(***tackle*** *power*) 在可 tackle 的范围内，可以使用 power∈[-100,100]进行 tackle
(score) 询问在 Time 时刻我们和对手的得分。
(***change_view*** *ANGLE_WIDTH QUALITY*) 改变球员的视角宽度和视觉能力。*ANGLE_WIDTH* 可以是 *wide*、*normal*、*narrow*。*QUALITY* 为 *high* 或 *low*。
(***sense_body***) { *sense_body* *TIME* (*view_mode* *QUALITY* *WIDTH*) (*stamina* *STAMINA* *EFFORT*) (*speed* *AMOUNT_OF_SPEED*) (*kick* *KICK_COUNT*) (*dash* *DASH_COUNT*) (*turn* *TURN_COUNT*) (*say* *SAY_COUNT*) }

表 2 .1 表示动作的命令

- ***catch***

球员抓球动作。守门员是唯一能执行catch命令的球员。守门员可以从任何方向抓到足球，只要足球在可抓范围内，守门员在罚球区内，且比赛模式是“play_on”。如果守门员以δ角度去catch足球，那么可抓范围是长宽分别是catchable_area_l和catchable_area_w的矩形区域。如果足球在这个矩形区域内，能够被抓到的可能性是catch_probability，在外面则不能

被抓到。

- ***turn***

球员的转身动作，其参数 *moment* 属于 *minmoment* 和 *maxmoment*（缺省为 -180^o 到 180^o ）组成的区间。在球员运动的过程中，由于惯性的存在，转身更为困难。一般地，球员的实际角度由下式计算：

$$\text{actual\_angle} = \text{moment} / (1.0 + \text{inertia\_moment} * \text{player\_speed})$$

inertia_moment 是一个参数，缺省值为 5.0。由上式可知，当一个球员以最大速度前进时，他最大可以转过的角度是 $\pm 120^o$ 。

- ***turn_neck***

球员转脖子动作。使用**turn_neck**命令，从某种角度上，是在独立于球员身体的转动头颈。球员的头部角度是他的视野角度。命令**turn**改变球员的身体角度，而命令**turn_neck**则改变了球员相对他的身体的颈部角度。球员颈部的相对角度介于**minmoment**和**maxmoment**之间（在文件serever.conf中定义）。切记头颈角度是相对于球员身体的相对角度，如果球员执行了**turn**命令，而没有执行**turn_neck**命令，球员的视野角度也是会改变的。

- ***move***

命令**move**可以把球员移动到场上的任何一个地方。只有在设置整个球队时，**move**命令有效，在正常比赛期间是没有效果的。在上下半场开始前（比赛模式是‘before_kick_off’）以及进球后（比赛模式是‘goal_r_n’和‘goal_l_n’）才可以使用**move**命令。在这种情况下，只要比赛模式没有改变，球员可以被移动到自己半场的任何地点（就是说x<0），而且可以被移动任意多次。如果球员还在对方半场的话，那么将会被server移动到己方半场的随机位置。

- ***dash***

dash 是指在球员所面对的方向上的一个冲力。它不是持续的跑。为了能持续地跑，必须要发送多个 ***dash*** 命令。***dash*** 的参数 *Power* 要乘以 *dash_power_rate*（缺省为 0.01），并且只能在球员所面对的方向。它只是一个小地向前冲的动作，并不改变球员的速度。下面给出球员力量的有效值（edp）：

edp = effort * dash_power_rate * power

- ***kick***

kick 命令同 ***dash*** 命令非常相似，只不过它用在球上，而非球员。球员是不能踢到 *kickable-area*（= *player_size*+*ball_size*+*kickable_margin*）以外的球。令 *dir_diff* 表示球相对球员面对方向夹角的绝对值，*dist_ball* 表示球员到球的距离，则 ***kick*** 命令的 *Power* 参数计算方法为：

Power = *kick_power_rate* * (1 – 0.25 * *dir_diff* / 180 – 0.25 * (*dist_ball* – *player_size* – *ball_size*) / *kickable_area*)

- ***say***

球员可以使用**say**命令对其他球员进行广播消息。消息最长**say_msg_size(10)**字节，有效字母是[-0-9a-zA-Z().+*/?<>_]（不包括方括号）。球员说的消息能够内**audio_cut_dist(50)**距离内的任一支球队的球员听到。发到server的消息会被马上的发送到可听范围内的球员处。命令**say**的使用仅仅受球员听觉能力的限制。

- ***tackle***

球员铲球动作。球员可以在一定的区域内进行铲球动作。铲球就是朝球员的当前方向铲球（被后也可以，力量加倍）。主要参数是铲球力量。

- ***score***

球员可以通过发score来询问即时球场双方的比分。

- **球员的体力模型（stamina）**

每个球员都有自己的体力值。球员在执行 ***dash*** 命令时，其 *Power* 参数必须要小于球员的体力值。每此执行完 ***dash*** 命令后，体力值都要减少 *Power*。体力的增加是由 *stamina_inc* 参数控制的。执行 ***dash*** 命令时：

Power = min（*Power*，*Stamina*）

Stamina = Stamina – Power

Stamina 的计算公式为：

Stamina = min（ *Stamina* +*Sinc*，*Smax*）

Soccerserver 通过限制球员的体力来阻止队员始终以最大速度（*player_sp_max*）跑动。其中涉及到三个方面：

- *stamina* (∈ [0，*stamina_max*]) 限制了 ***dash*** 命令的 *Power* 参数。
- *effort* (∈ [*effort_min*，1.0]) 表示了球员移动的效力。
- *recovery* (∈ [*recovery_min*，1.0]) 表示可补充的体力所占比例。

球员的体力分为可补充体力和非补充体力两个部分。*Stamina* 和 *effort* 是可补充部分 *recovery* 为非补充部分。表 1 中有关体力的参数可按如下方式计算：

当一个球员使用(***dash*** *Power*)命令时，他的 *Power* 参数要受到 *stamina* 和 *effort* 的影响：

Power = min（*Power*，*Stamina*）

Stamina = Stamina – Power

Effective_dash_power = Min(*stamina, Power*) × *effort*

Stamina = stamina – effective_dash_power

在每个循环周期内，如果 *stamina* 低于某个阈值，则 *effort* 减少，否则增加：

if *stamina* ≤ *effort_dec_thr* × *stamina_max* and *effort* >*effort_min* then

effort = effort – effort_dec

if *stamina* ≥ *effort_inc_thr* × *stamina_max* and *effort* < *1.0* then

effort = effort + effort_inc

在每个循环周期内，如果 *stamina* 低于某个阈值，则 *recovery* 减少：

if *stamina* ≤ *recover_dec_thr* × *stamina_max* and *recovery* > *recover_min* then

recovery = *recovery* – *recover_dec*

在每个循环周期内，stamina 依 recovery 的当前值而增加：

stamina = *stamina* + *recovery* × *stamina_inc*

if *stamina* > *stamina_max,* then *stamina* = *stamina_max*

以上我们说明了球员的动作模型。也是说球员可以发送给服务器的各种基本动作。在一个周期内我们可以发送一个或多个动作给服务器。但其中有些基本动作是不能同时发给服务器的，如我们在一个周期内不能同时发送 dash 和 kick 给服务器。我们称这样的动作为互斥动作。能够同时发送的我们称为相容动作。如 turn_neck 和 dash 即是相容动作。

2.2.6 异构球员

Server在开始时产生**player_types**种类型以形成异构队员。基于权衡的原则，文件player.conf中定义了不同球员类型的不同能力。一场比赛中的球员使用相同的球员类型。类型0是默认类型，而且是同构的。当球员和server连接上时，就会得到能够利用的球员类型信息（见4.2.1）。在‘before_kick_off’模式下，在线教练能够无限制的改变球员类型；在非‘play_on’的其他模式下，使用**change_player_type…**命令，能够改变球员类型**subs_max**次。

2.2.7 裁判模型

自动裁判发送消息给球员，这样球员能够知道当前的比赛模式。自动裁判的规则和行为见2.1.2和2.1.3。球员用hear接收裁判消息。不管球员已经接收到其他球员的消息数量，他都能在每个仿真周期听到裁判的消息。比赛模式的改变由裁判宣布。裁判还有宣布其他的一些事情，如进球或者犯规。

2.3 monitor

Monitor是一个可视化的工具，允许人们观看比赛时server到底发生了什么事情。在monitor上显示的信息包括比分，球队名字，所有球员和足球的位置。Monitor也提供了一个很简单的server接口。如：当两支球队都连接上后，在monitor上的“Kick－Off”按钮允许人类裁判开始比赛。正如你将发现的，在server上进行比赛，monitor并不是必需的。然而，如果有需要的话，可以同时和server连上很多的monitor（如你想在不同的终端显示同一场比赛）。

2.4 client

本节讨论client和server间的协议的基本情况。更多的细节见2.2。

注意初始化和重新连接命令将被送到运行server机器的球员UDP端口（默认是6000），得到响应后，该球员就和server分配的端口绑定，以后的信息也发送到这里。Server从这个端口发送初始化响应信息。所有发送到server和从server端接收的命令都是普通字符串。**2.4.1**

初始化和重新连接

每个要和server连接的球员都要首先介绍自己。这就好像是次握手，在开始时进行一次，在你希望重新连接时随意进行。

初始化球员要以下面的格式来发送init命令：

(init TeamName [(version VerNum)] [(goalie)])

守门员必须在init命令中包括“（goalie）”，这样server才会运行抓球或做其他守门员才能做的动作。请注意每个球员只能有一个守门员或者没有守门员（没有规定必须要有一个守门员）。

Server用如下格式的消息来响应你的初始化消息：

(init Side UniformNumber PlayMode)

或者是出错消息（如果出错的话，就是说你初始化了不止两支队伍，一支球队的人数超过了11个，或者一支球队中使用了不止一个守门员）：

(error no more team or player or goalie)

Side是你球队比赛时的标示，一个字母，**l**(left)或者是**r**(right)。UniformNumber是球员的球员号（球员通过它们的球员号被识别）。PlayMode是表示一个有效比赛模式的一个字符串。如果你和server版本是7.00或更高的连接，你会接收到更多的server参数，球员参数和球员类型信息（最后两个是关于异构球员的特性）。到此握手结束，你的球员被作为有效球员识别了。

重新连接

重新连接是很有用的，因为无需重启比赛就可以修改球员的程序。只能在无PlayOn的比赛模式中使用（如：在半场时间）。

使用下面的格式进行重新连接：

(reconnect TeamName UniformNumber)

在这里，如果你是和server版本是7.00或更高的连接，也会返回更多的server参数，球员参数和球员类型信息。

断开连接

在你断开之前，你可以发送给server一个bye命令。这个命令将会把球员从场上移出。

(bye)

版本控制

由于server的不断开发，每年都有新的特性被加入，为了支持这些新特性，需要修改原先的协议。为了能够向下兼容老版本client程序，为了是开发者更方便的工作，故加入了协议版本控制Protocols Version Control。每个球员都要在**init**命令中说明他的通讯协议，这样server会以适当的格式发送消息。但是注意即使通讯协议没有改变，如果仿真规则变化的话，也是会影响整场比赛的。

2.4.2 控制命令

所有球员的运动行为都是由几个命令组成的，这几个命令前面已经有所谈及。

这些命令的结果有些复杂，并且依赖于很多仿真因素。对每个命令的执行细节见2.2。

(turn Moment)

(dash Power)

(kick Power Direction)

(catch Direction)

(move X Y)

一个周期里面发送以超过一个命令，那么就会被随机执行其中的一个命令，通常情况下是先到达的那个）。

(turn_neck Angle)

(say Message)

(sense body)

(score)

(change_view Width Quality)

2.4.3 感知信息

感知信息被有规律的发送给所有的球员（如每个周期或者是每1.5个周期）。所以没有必要向server发送命令请求得到这些信息。所有感知返回的信息都有时间戳（Time），表明该数据被发送时server端的周期数。这个时间是很有用的。

(1)视觉感知

视觉感知是最重要的感知器，但是有一点复杂。这个感知返回球员能够见到的对象信息（就是说在视角范围，又不很远的对象），信息的主要格式如下：

```
(see Time ObjInfo ObjInfo . . . )
```

ObjInfo是如下的格式：

```
(ObjName Distance Direction [DistChange DirChange [BodyFacingDir
HeadFacingDir]])
```

或

```
(ObjName Direction)
```

注意返回的对象信息和他的距离有关。对象相距越远，得到的信息越少。更多的信息

ObjName是下面的一种：

```
(p [TeamName [Unum]])
(b)
(f FlagInfo)
(g Side)
```

p表示球员，**b**表示足球，**f**表示标志，**g**表示球门。Side是**l**表示左半场或者是**r**表示右半场。

(2)听觉感知

听觉感知返回能在场上听到的消息。可能来自在线教练，裁判或者其他球员。

格式如下：

(hear Time Sender Message)

Sender是下面的一种：

Self：发送者是自己；

Referee：发送者是比赛裁判。

Online_coach_l或者是**online_coach_r**

Direction：如果是其他球员发送的消息，那么sender将会被发送者相对于自己的角度所代替。

(3)自身感知

自身感知返回球员的所有状态，如体力，视觉模式，球员在每个周期刚开始时的速度：最后8个参数是接收到命令的计数值。使用这个计数值用来寻找丢失或延误的消息。

2.5 coach

2.5.1 coach 介绍

教练coach 是为其他球员提供帮助的特殊client。有两种coach：在线教练online coach和训练者trainer。后者也常被叫做离线教练，但为了更清楚的描述，我们将使用训练者trainer 的叫法。

2.5.2 训练者和在线教练的区别

一般而言，训练者能对比赛实行更多的控制，仅仅在开发阶段使用；而在线教练则可以在正式比赛时使用。在自学习或者自动控制开发时，训练者是非常有用的。在线教练则是在比赛时为球员提供更多的建议和信息。

在开发球员client 时，比如对带球、踢球技术进行机器学习时，以自动方法构造训练场景是非常有效的。因此，训练者应该有以下的功能：

(1)能控制比赛模式play-mode。

(2)能广播听觉消息。消息应该可以包括一个命令或者一些其他球员的信息。它的语法和语义是用户自定义的。

(3)能将球员和足球移动到场上的任何位置，同时可以设定他们的方向和速度。

(4)能得到运动物体的无噪声信息

在线教练则是企图观察比赛然后为球员提供建议和信息。因此，他的能力有所限制：

(1)能和球员通讯

(2)能得到运动对象的无噪声信息

为了防止教练集中式的控制每一个球员，通讯能力被限制了，只能发有限次的消息（隔300个周期才能发）给球员，并且发送以后要经过一定的周期（50个）才能被球员接收。在线教练是一个很好的工具，可以进行对方建模，比赛分析以及给己方球员战略指导。由于教练能得到场上的无噪声全部信息，而且实时要求不高，因此教练可以花费更多的时间考虑战略。

2.6 本章小结

本章主要介绍了 Robocup 比赛手册的各个组成部分：

（1）Server 的获得和安装。

（2）详细介绍了 SoccerServer 的组成部分，指出了组成 Server 包含的各种对象，对象的一些参数设置，说明了运动模型、感觉模型、动作模型、球员的异构以及裁判模型。这在以后的各章节都有引用。

（3）说明了 monitor。

（4）说明了简单的 Client 的构造。

（5）说明了 coach 的一些使用方法。

第三章 一些著名球队的 Agent 结构以及 MAS

在开始我们的Robocup机器人仿真组程序的研究以后，我们查阅了大量的相关资料，研究了从99年来在robocup仿真组获得了不错成绩的球队的综述报告及相关文档，他们包括99年冠军CMU、2000年冠军FC Portugal、2001、2002年的Tsinghuaeolus、以及包括Karlsruhe Brainstrormers、UVA-Trilearn等等球队。下面就把了解到的相关球队的工作一一说明。

3.1CMU

CMU 是美国卡耐基梅隆大学的一支球队，曾获得了 robocup98、99 仿真组的冠军。这支球队的主要设计人 Peter Stone 在他的博士论文《Layered learning in Mutli_agent System》中详细的描述了这支球队。

首先,我们给出 CMU 的 Agent 结构，如图 3.1：

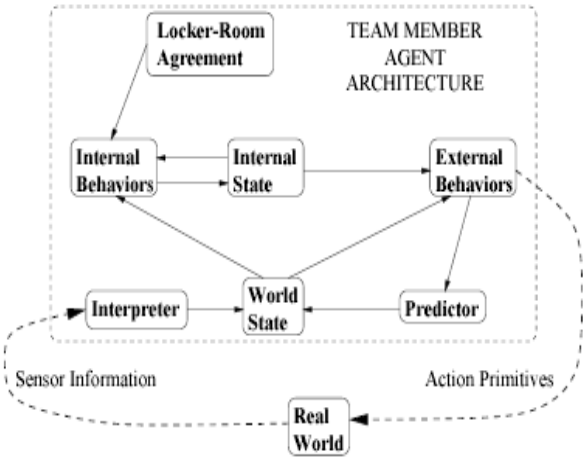


图 3.1 CMU 的 Agent 结构图

这个结构可以感知环境，能够对当前的环境做出分析，最后向 Server 发送经过决策的动作。

Real World：Server 表示的世界状态。

World State：agent 内部可识别的世界状态。从 Real World 到 World State 需要进行解析。

Locker-room Agreement：用于 Agent 的同步，并定义了球队进行协作的机构以及 Agent 之间的通讯协议。它仅能够被 Internal Behaviors 访问。

Internal State：存储了 Agent 的一些内部变量。可以存储 Agent 以前或当前时刻的世界状态。

Internal Behaviors：根据当前的世界状态、内部状态、球队协议（Locker-room Agreement）来更新 Agent 的内部状态的内部动作。

External Behaviors：根据世界状态以及更新后的内部状态来做出一个动作送给动作器以作用于真实世界（Real World）。同时回送给 Agent 进行预测。

从根本上讲，CMU 的 Agent 是通过行为/条件（B/C）对来进行外部动作的输出的，我们可以通过图 3.2 得出。

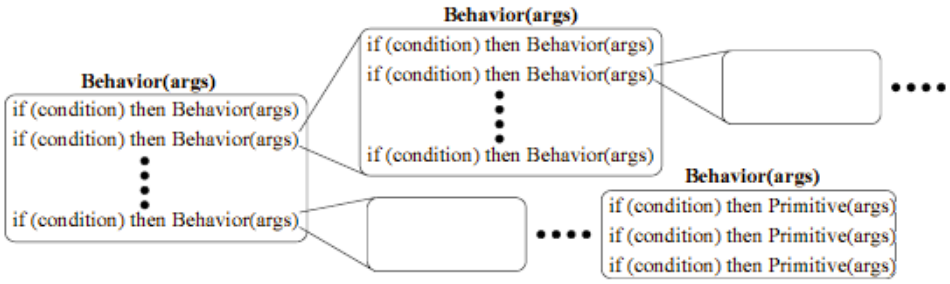


图 3.2 CMU 的 BC 树

CMU 留给我们的还不只是这些，Peter Stone 在他的博士论文中，提出了关于在 MAS 中各个 Agent 为达到一个目标是如何进行协调和学习的，Peter 提出了分层学习的方法。在这方面也做一个简要介绍：

第一层，进行 Agent 个体基本技术的学习。典型的例子就是进行断球的学习，通过神经网络的方法，对在不同的场景下学习断球。

第二层，Agent 同另外一个 Agent 之间的协同学习。典型的例子是进行传球的学习，如在球场上 Agent 当前在控球，并且做出了决策要就进行传球给一个特定的队友，这时它必须学习一个合适的方向和传球速度。在该层它可以调用第一层已经学习过的基本技术。这通过构造决策树的方法，求出相应的节点值，构造出一个分类器。典型的算法是 C4.5 决策树算法。

第三层，Agent 同其他多个 Agent 之间进行的球队策略学习。比较典型的例子是进行传球对象的选择，如在球场上 Agent 当前在控球，这时它要选择把球传给哪个队友。在学习本层的时候，也认为第二层已经学习过了，这时它要考虑时传给哪个队友的利益大。主要是通过 TPOT-RL(Team-Partitioned Opaque-Transition Reinforcement Learning)来进行学习。

3.2 FC Portugal

FC Portugal 由葡萄牙的里斯本大学和波尔图大学合作完成的一支球队。它是在 CMUnited99 公开的底层源代码的基础上，对多智能体的合作方面做出了巨大的贡献（在这之前，Robocup 仿真参赛队的阵型以及战位都很混乱）。具体来说，FC Portugal 在球队策略、战术、阵型、球员类型、站位机制以及角色的动态转换机制等方面都有自己的特点。

FC Portugal 的 agent 的主要控制循环是使用感知解释和动作预测来更新世界模型，其结构如图 3.2，然后使用高层决策模型来决定下一步的动作。FC Portugal 的信息模型是一个四层结构的数据模型：

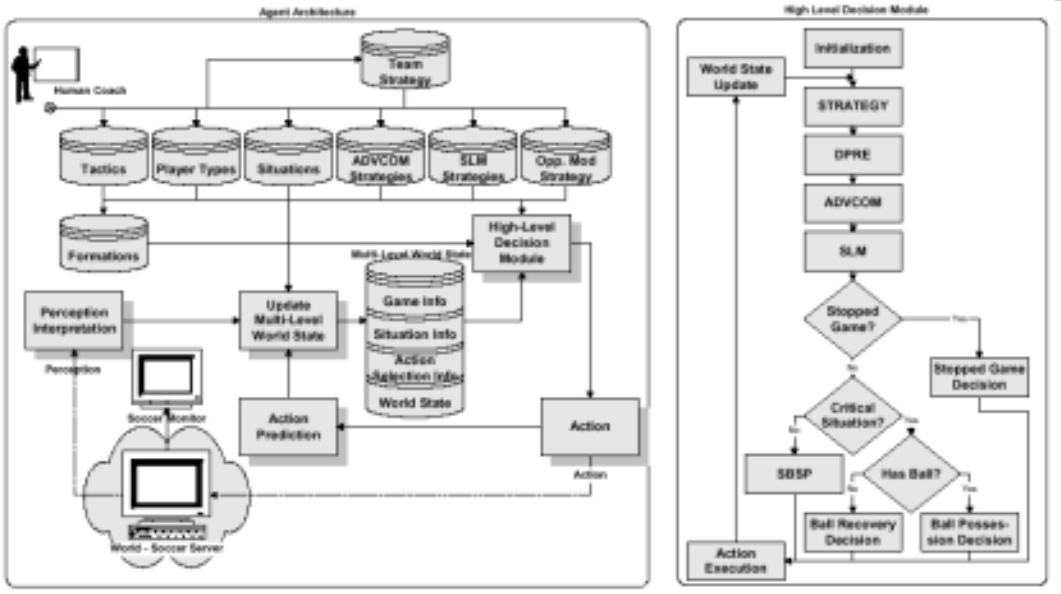


图 3.3 Agent 结构和高层决策模型的控制流程

- 全局形势信息 － 高层信息比如比分、时间、比赛策略（射门，成功传球，控球等）和对方行为，用来确定每一时刻的战术；
- 形势信息 － 与阵型选择相关的信息，和 SBSP，SLM 和 ADVCOM 机制相关的信息；
- 动作选择信息 － 一套高层参数，用来确定动态形势，选择适当的控球或开球行为；
- 世界状态 － 底层信息，包括球员和足球的位置和速度。

CMUnited 提出了阵型和站位的概念，并且根据比赛结果和剩余时间动态改变阵型；FC Portugal 扩展了这些概念，并提出了战术和球员类型。FC Portugal 球队策略定义是基于一套球员类型（定义了球员策略，控球和开球行为）和一套战术包括几个阵型（433，442，开放 433，344，532 等）。阵型是在不同的比赛情况下使用的，如防守、攻击、从防守到攻击的转换、球门球等，对每个球员赋予一个本位点和球员类型。不同的球员在球场上的位置和动作的倾向性是不一样的。 图 3.4 描述了 FC Portugal 的球队策略的结构。

在 FC Portugal 中，最主要特点是它的 SBSP（Situation Based Strategic Positioning）— 基于情形的策略站位和 DPRE（Dynamic Positioning and Role Exchange）— 动态站位和角色变换。

所谓 SBSP，就是 Agent 能够根据当前球场上的形势，包括现在球队正在使用的阵形、战术以及球员的类型来确定球员在球场上的基本位置；再通过球的位置、速度、球场上的形势（如本方是在进攻、本方是在防守、双方的得失球等）以及球员的策略特性来修正基本位置，得到球员的应该处的位置，也就是球员的策略站位点。球员的策略特性包括对球的吸引力、球场球员可以容许站的位置、在场上某些区域的特定位置特性、粘球的倾向、越位线的设置以及在某些特定形势下对应该对场上特定目标的注意力等方面。

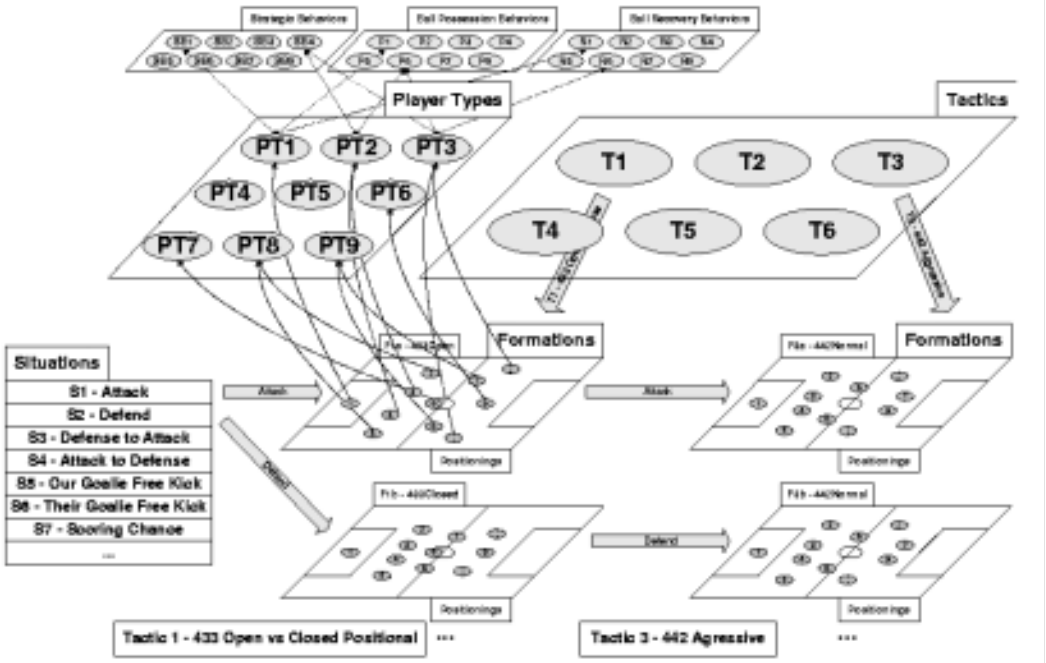


图 3.4：球队策略（由战术，阵型和球员类型组成）

DPRE 是基于 Peter Stone 关于根据协议来进行智能体角色的变换的继续研究。在 FC Portugal 中，球员不仅能够改变它们的站位（站位由来阵型决定的），而且还可以在当前阵型下改变球员类型。当然 DPRE 只有有利于球队时才使用的。是否有利，只要通过一个效用函数来进行评价的。效用函数要考虑一下因素：球员位置到它们的战略位置的距离、每个站位的重要性和在当前形势下的站位是否恰当。

3.3 Tsinghuaeolus

Tsinghuaeolus 是中国清华大学研制的一支球队，清华风神在它参加的 2 次世界杯和 3 次中国 Robocup 都获得了冠军，这支球队攻击能力和防守能力都很强。Tsinghuaeolus 的特点是它的 Agent 结构设计比较优秀，对每一个动作的使用和选择都比较合理。

下面我们给出 Tsinghuaeolus 的 Agent 结构。它把 Agent 设计成一个具有 3 层的分层结构，具有通讯（communication）、视觉控制（visual control）、控球（handle_ball）、进攻跑位（offense positioning）、防守跑位（defense positioning）等模块。在层次机构中，动作产生器（Action Generation）是通过对动作空间离散化产生备选的动作集合（动作集越小，计算代价越小，就越优，但要保证最优动作包含在里面）。评价器（Evaluator）对这些动作进行评价，主要是根据进攻价值、防守价值、成功概率等因素，获得动作的优先级。仲裁器（Mediator）仲裁由 Evaluator 提交的最优动作中有没有冲突动作。

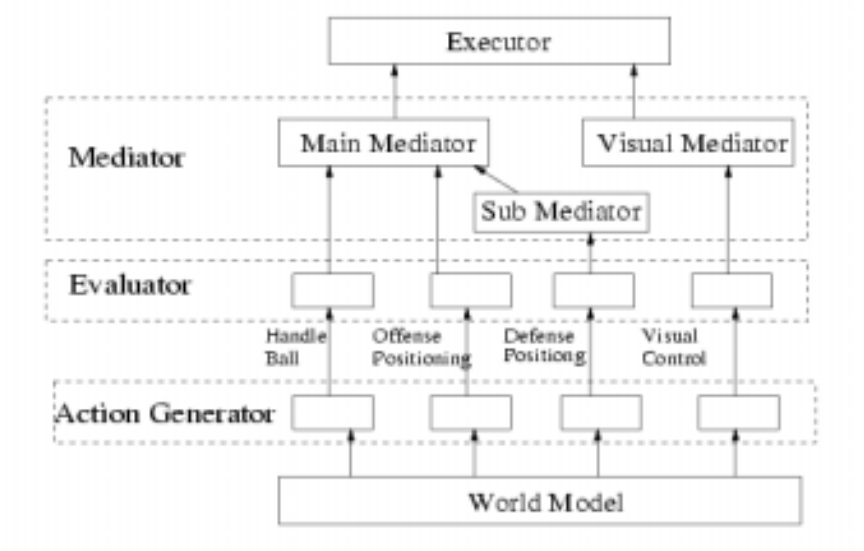


图 3.5 Tsinghuaeolus Agent 结构

Tsinghuaeolus 的整体效果强主要是它的个体技术强，也就是它的 low-level skill（individual skill），如截球、带球、加速球等等。Tsinghuaeolus 主要是通过解析法来截球的，解析法主要设计出球的运动曲线以及需要去截球 Agent 的运动曲线（X、Y 表示球场坐标，t 表示时间），发现球的运动是二次曲线，Agent 的运动是一个圆锥体求二者的交点（2-3 个），得出截球点。通过离线学习和在线规划来学习加速球（Fastkick）。考虑穿越速度来进行传球的学习。

Tsinghuaeolus 的整体策略（MAS 协调）主要分为 2 种，进攻跑位策略和防守跑位策略。其中进攻跑位策略很简单，大致是使用一个 433 的跑位阵型。每个球员的位置是一个包含球的位置信息的一个函数 P（B,i），B 表示球的位置，i 表示球员号。这个函数神经网络得到的。在防守跑位策略定义了 7 个防守角色，3 种防守动作（Mark、Block、Formation）每个角色配备几种属性权值，它根据自己的这些属性权值以及场上的因素，比如说球的位置、自己的位置、附近对手的位置等来挑选一种防守动作来执行。

3.4Karlsruhe Brainstrormers

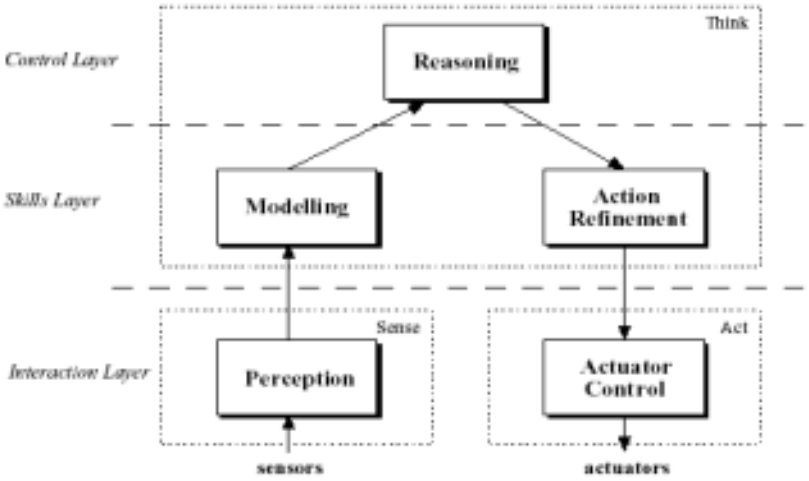
Karlsruhe Brainstrormers 球队是德国卡尔斯鲁厄大学的一支球队。这支球队的出发点是进行 Reinforcement Learning（强化学习/再励学习），他们的长期目标是在要给出一个命令如“赢得比赛”，Agent 就能够自动的去学习，选择合适的动作，这是真正用人工智能的方法去考虑 Robocup 问题，如果这个目标能够得到实现将是人工智能的一大飞跃。

Karlsruhe Brainstrormers 首先把 robocup 问题看成 POMDP（Partially Observed Markav Decision Problem），简化世界状态和动作集，通过使用动态规划的方法，用前馈神经网络来近似一个对连续的状态空间进行评价的 V 函数，通过不断尝试学习去提高 V 函数的性能。

3.5 UVA-trilearn

UVA-Trilearn 是荷兰阿姆斯特丹大学的一支球队。欧洲的老牌强队，它的 Agent 结构设计成三层的分层结构。UVA-Trilearn 的 MAS 结构特点不是很明显，它主要把异构球员的方法应用到 Agent 系统，所以该球的攻击力比较强悍。

UVA-Trilearn 的 Agent 功能结构



3.6 其他球队

科大蓝鹰、AT Humboldt[27]都是把BDI引入了Robocup，通过定义Belief为球员感知到的球场信息（World Model），Desires主要是根据球场信息确定可能要使用的动作。Intention则是根据Belief而在Desire中选择最好的desire作为要选择的动作。RoboLog Koblenz[28]是一个使用prolog语言来进行球队设计的球队。还有许多其它球队在这里就不一一列举了。

3.7 小结

Robocup 仿真组比赛 Agent 结构的设计是多种多样的，又采用深思型的 Agent 结构也有采用反应型 Agent 结构，但更大多数的球队是使用混合型 Agent 结构的，这也是由于混合型 Agent 的特点决定的。因为混合型 Agent 可以被设计成多层，高层进行球队的整体规划，底层直接处理一些紧急或规划好的动作。这比较好理解，同时应用起来也比较方便。

在球队策略也就是 MAS 协调方面就更没有办法统一了，有的球队甚至根本就没有 MAS 协调，如 Karlsruhe Brainstrormers 就直接使用了强化学习，在学习动作的时候就已经包含如何配合，就没有专门的考虑球队的策略问题。在采用了球队策略的球队中，决大多数都结合了足球专家的知识，把现代足球的一些知识应用到机器人足球中，基本上都是用了阵型和球员角色以及战术等足球概念。让后把这些概念通过一定的学习算法和数学模型应用到机器人足球中去。所以说，如果你想当足球教练，那就请你组织一支机器人足球队吧。

第四章 UvA Trilearn 头文件与类

ActHandler.h

文件描述：

This file contains the class declarations for the ActHandler that handles the outgoing messages to the server.

全局函数:

void sigalarmHandler (int i)
该函数在 SIGALARM 信号到达时被执行。该信号到达时间由 SenseHandler 定义。当信号到达时，ActHandler 中的命令队列里的命令被发送到服务器。

全局变量：

Logger Log

class ActHandler

Detailed Description

The ActHandler Class is used in the RoboCup Soccer environment to send the commands to the soccerserver. The ActHandler contains a queue in which the commands are put. When a signal arrives (set by the SenseHandler depending on the time of the sense_body message) the commands that are currently in the queue are converted to text strings and send to the server. The sent commands are also passed to the WorldModel, such that the WorldModel can update its internal state based on the performed actions. It is possible to send more than one command to the server at each time step, but some type of (primary) commands can only be sent once (kick,dash, move, tackle, turn and catch). Therefore internally different two queues are stored. One with only one element, namely the last entered primary command. And a separate queue containing all other commands. Each time a command is put into the queue that is already there, the command is updated with the new information. Furthermore it is also possible to directly send commands (or text strings) to the server. These methods can be used when an initialization or move command has to be sent to the server and you're sure this information is final, i.e. the message will not become better when new information arrives from the server.

注：

该类用于在机器人足球赛环境内向服务器发送命令。它包含一个命令队列。当一个信号到达时，队列中的命令将被转化成字符串并发送到服务器。发出的命令同时也发送到世界模型，世界模型可以根据相应动作的执行而更新自己的内部状态。某些命令可以在一个时间周期内发送多个，而某些命令每个周期只能发送一条，因此内部设置了两个任务队列。一个队列只有一个元素，即最后入队的主要命令。另一个队列包含了所有其他的命令。当一个命令进入一个已经包含该命令的队列，则队列中的该命令根据新的信息更新。此外，也可以直接将命令发送到服务器。这些方法可以用于将初始化或 move 命令送到服务器，同时可以肯定这些命令是最终的，即不需要再根据服务器发来的信息更新。

公有方法：

ActHandler (Connection *c, WorldModel *wm, ServerSettings *ss)
构造函数。所有的变量初始化。
bool　putCommandInQueue (SoccerCommand command)
该方法将 command 放入命令队列，最后放入队列的命令，将在 sendCommands 执行的时候被发送到服务器。这通常在收到 SenseHandler 发来的信号后执行。
void　emptyQueue ()
清空任务队列。
bool　isQueueEmpty ()
判断任务队列是否为空。
bool　sendCommands ()
该方法将任务队列中的所有命令转化成字符串，并通过 Connection 发送到服务器。当服务器没有执行上个周期的命令时，为了避免冲突当前命令并不发送。这时返回 false。
bool　sendCommand (SoccerCommand soc)
该方法直接发送一个单独的命令到服务器。
bool　sendMessage (char *str)
该方法将一个字符串直接发送到服务器。为了保证该信息到达，该信息发送后将等待一个周期。
bool　sendCommandDirect (SoccerCommand soc)
该方法将一个命令直接发到服务器。
bool　sendMessageDirect (char *str)
该方法将一个字符串直接发送达服务器。

私有属性：

Connection *　connection	与服务器的连接
ServerSettings *　SS	服务器设置
WorldModel *　WM	世界模型
SoccerCommand　m_queueOneCycleCommand	单命令队列

SoccerCommand　m_queueMultipleCommands [MAX_COMMANDS]　多命令队列
int　m_iMultipleCommands　　多命令队列里的命令数。

BasicPlayer.h

文件描述：

This file contains the class definitions for the BasicPlayer. The BasicPlayer is the class where the available skills for the agent are defined.

全局变量：

Logger　Log

class　BasicPlayer

Detailed Description

This class defines the skills that can be used by an agent. No functionality is available that chooses when to execute which skill, this is done in the Player class. The WorldModel is used to determine the way in which the skills are performed.

注释：

该类定义 agent 使用的 skill。如何选择时机执行这些 skill 在该类里并没有给出，而在 Player 类里给出。WorldModel 被用于决定这些 skill 使用的方式。

公有方法：

SoccerCommand　alignNeckWithBody ()
该 skill 使 agent 的脖子转向同身体一样的方向。该方法返回一个 turn neck 命令，并将身体相对于脖子的角度作为参数。
SoccerCommand　turnBodyToPoint (VecPosition pos, int iCycle=1)
该方法将 agent 的身体转向某一点。它接受一个场上绝对坐标作为参数，返回一个将身体转

向该点的 turn 命令。为了实现这一目标，agent 在下一个周期的位置，将根据其速度被估计出来，以补偿由于惯性将 agent 移动到一个新的位置。如果不能转到该点，则转动尽可能大的角度。ICycle 表示用来预测 agent 的新位置的周期数。

SoccerCommand　turnBackToPoint (VecPosition pos, int iCycle=1)

该方法将 agent 背向某一点。该方法通常用于让守门员回到球门附近而保证其身体仍朝向球场。

SoccerCommand　turnNeckToPoint (VecPosition pos, SoccerCommand com)

该方法将 agent 的脖子转向某点，它接收一个场上的绝对目标点，和在这个周期将执行的动作作为输入，返回一个 turn neck 命令。由于 turn neck 可以与主要动作在同一个周期内执行，所以第二个参数传入的主要动作是必要的。如果脖子转向的方向与身体当前方向的夹角超过所允许的最大值，则将脖子转动尽可能大的角度。

SoccerCommand　searchBall ()

该方法使 agent 在看不到球的时候在场上搜寻球。该命返回一个 turn 命令使 agent 转动一个它的视角大小的角度。需要注意的是，agent 向他上次看到球的方向转动，以防止向相对于球相反的方向转动。此外 agent 的惯性也被考虑，以补偿由于当前速度引起的位置变化。

SoccerCommand　dashToPoint (VecPosition pos)

该方法返回使 agent dash 到指定点的命令。由于 agent 只能向前或向后 dash，所以距离目标点最近的点是目标点在 agent 身体方向直线上的投影。dash 命令里的 power 参数是通过 getPowerForDash 计算出来的。

SoccerCommand　freezeBall ()

该方法使 agent 能够停住正在移动的球。它返回一个踢球命令使球停住。因为在 soccer server 球的运动视作向量的叠加，所以如果向球施加一个同样大小但是方向相反的加速度，球就会停。此外踢球的方向应该在球的运动方向加 180 度。注意，该方向一定要转化成相对 agent 身体的方向，这是作为 kick 参数所必需的。

SoccerCommand　kickBallCloseToBody (AngDeg ang)

该方法使 agent 将球踢向靠近自己的一个位置。它接收一个参数 ang，返回一个 kcik 命令把球踢向相对身体 ang 方向，距离很近的一个目标点。

SoccerCommand　accelerateBallToVelocity (VecPosition velDes)

该方法使 agent 将球加速到指定速度。它接受一个目标速度参数，返回一个相应的踢球命令。当按照速度计算出来的踢球命令中的 power 没有超过最大值，则球的速度可以由一个单独的踢球动作完成，此时踢球的方向等于相对于 agent 身体方向的加速矢量。但是如果球的目标速度太高或者是球的当前速度太快，则无法用一个单独的 kick 命令无法完成，则使球以一个较低的速度沿着 velDes 的方向前进。

SoccerCommand　catchBall ()

该方法使守门员扑球。它返回一个扑球命令，以球相对于 agent 身体方向的角度作为唯一的参数

SoccerCommand　communicate (char *str)

该方法使 agent 与场上的其他队员通信。它接受一个字符串形式的消息，返回一个 say 命令，使该字符串向场上某一距离范围内的队员广播。

SoccerCommand　teleportToPos (VecPosition pos)

该方法返回一个 move 命令将 agent 移动到某一位置。

SoccerCommand　turnBodyToObject (ObjectT o)

该方法将 agent 的身体方向转向对象 o。为了实现这一目的，对象 o 的在下一个周期的位置将根据其当前速度估计出。该位置传给 turnBodyToPoint 函数，以返回一个 turn 命令，将 agent

的身体方向转向目标对象。

SoccerCommand　turnNeckToObject (ObjectT o, SoccerCommand com)

该方法将 agent 的脖子方向转向对象 o。对象 o 的在下一个周期的位置将根据其当前速度估计出。该位置传给 turnNeckToPoint 函数，该底层函数返回一个 turn 命令，将 agent 的身体方向转向目标对象。因为 turn neck 命令可以与主要命令一起执行，所以在本周期末要执行的主要命令也作为参数传入。

SoccerCommand　moveToPos (VecPosition posTo, AngDeg angWhenToTurn, double dDistDashBack=0.0, bool bMoveBack=false)

该方法使 agent 移动到绝对位置 posTo。由于 agent 只能以自己的身体方向向前或向后移动，该动作的关键是决定何时作 turn 或 dash。turn 可以使 agent 在下一个周期向着目标点运动，但是其缺点是占用一个周期，同时使 agent 的速度降低，因为该周期没有加速。所以除了目标点，该方法还接收几个参数以决定在当前情况下是否进行 turn 和 dash。当目标点在 agent 前且与 agent 的夹角小于 angWhenToTurn，则进行 dash 命令。如果目标点在 agent 身后，且于目标点的距离小于 dDistBack，目标点与 agent 后背方向的夹角小于 angWhenToTurn，则进行 dash，否则 turn。对于守门员，常常需要背向球门向后移动以保持自己注视场上情况，所以该方法包含 bMoveBack 参数，表示是否应该一直 backdash，当这个参数的值为 true，如果目标点相对于 agent 后背方向大于 angToTurn，则 agent 转到背向目标点的方向。其他情况 agent 执行一个 backdash，而不管于目标点的距离是否小于 dDistBack。

SoccerCommand　interceptClose ()

该方法使 agent 在球靠近他的情况下截球。其目标是移动到这样一个状态，球将在一到两个周期内可踢。为实现这一目标，调用世界模型的方法预测球在一到两个周期后的位置。然后决定是否可以利用 turn 和 dash 的逻辑组合使 agent 到达其中一个位置。如果不可能在两个周期内截球，则该方法返回一个非法命令以表示无法完成。首先判断是否使用一个单独的 dash 命令可以实现，如果不行，再试两个 dash 命令，如果又不行，则试一个 dash 命令和一个 turn 命令，如果还不行，则无法实现。

SoccerCommand　interceptCloseGoalie ()

该方法使守门员在球靠近他的情况下截球。其目标是移动到这样一个状态，球将在一到两个周期内可已被扑到。为实现这一目标，调用世界模型的方法预测球在一到两个周期后的位置。然后决定是否可以利用 turn 和 dash 的逻辑组合使 agent 到达可以扑球的位置。如果不可能在两个周期内截球，则该方法返回一个非法命令以表示无法完成。首先判断是否使用一个单独的 dash 命令可以实现，如果不行，再试两个 dash 命令，如果又不行，则试一个 dash 命令和一个 turn 命令，如果还不行，则无法实现。

SoccerCommand　kickTo (VecPosition posTarget, double dEndSpeed)

该方法使 agent 将球从当前位置，踢向指定的位置，并保持球的末速度为 dEndSpeed。通过一个单独的 kick 命令，球的末速度很可能达不到指定的速度，原因有两个：指定的速度大于球的最大速度或者当前的球速结合其相对于 agent 的位置无法使球加速到指定速度。对于第一个原因，如果目标速度超过最大速度某个百分比，则踢球命令将被执行个忽略该速度。第二个原因，agent 使用 kickBallCloseToBody 将球直接踢向靠近自己的位置，这样 agent 可以在下个周期使用更大的 power 将球踢出。

SoccerCommand　turnWithBallTo (AngDeg ang, AngDeg angKickThr, double dFreezeThr)

该方法使 agent 转向一个绝对位置，同时保持球在自己的前方。它可以用于，比如说，一个防守队员将球截下，但是却是面朝自己的球门，这时它需要使用该方法将身体转向进攻方向，以将球转移到前场。带球转身需要一系列命令，首先，球要踢到相对于 agent 的一个合适的位置，然后球要被停住在那个位置上，最后 agent 要转向球的方向。每次该方法被执行的时

候，它将决定整个动作已经执行到什么地方了。这是通过如下方法完成的，如果期望的角度与球到 agent 位置的角度的差大于 angKickThr，则使用 kickBallCloseToBody 将球踢到期望角度且与 agent 保持比较近的距离。然后，检查球是否还是按照原来的速度运动，如果球速超过 dFreezeThr，则使用 freezeBall 方法将球在当前位置停住。否则，agent 将自己身体方向转向 ang 方向。

SoccerCommand　moveToPosAlongLine (VecPosition pos, AngDeg ang, double dDistThr, int iSign, AngDeg angThr, AngDeg angCorr)

该方法使 agent 沿着一个给定的直线运到到指定位置。这被用作，比如说守门员保持在球门前的一条直线上，但是要根据球的位置移动到直线上的另一点。此外，它还可以被用作防守队员通过在一个对方球员与球的连线上运动来标识该队员。实现思想是，agent 必须迅速的沿着直线移动到目标位置 pos，同时使 turn 的次数尽可能少，以避免浪费周期。除了 pos 参数，还有几个参数以决定 agent 在当前情况下执行 turn 还是 dash。由于 agent 只想向前或向后移动，所以保持身体的方向与给定的直线的方向一致以尽快到达目的地是非常重要的。参数给出了 agent 在 pos 的绝对身体方向。因此直线 l 定义为经过 pos 点，绝对角度为 ang 的直线。由于噪声的干扰，agent 身体的方向不会总是保持 ang，这会导致 agent 的位置偏离直线。每次调用该方法，agent 的期望方向将根据其位置作轻微的调整。如果 agent 当前位置于直线的距离 d 小于 dDistThr，则角度保持不变。但是如果距离超过 d，则接下来的周期里 agent 将靠近直线移动，这是通过，在期望的角度 ang 上根据 agent 相对直线的位置以及 agent 在今后几个周期的运动增加或减少 angCorr 实现的。该预测由一个参数 iSign 表示，当运动方向与 ang 的方向一致是 iSign 的值为 1，如果方向相反则为-1。最终是执行 turn 还是 dash 的决策还是比较 agent 当前的方向与期望方向来决定的。如果上面两个角度的差超过 angThr，则调用 turnBodyToPoint 转向期望的方向，否则调用 dashToPoint 向目标方向移动。

SoccerCommand　intercept (bool isGoalie)

当截球方法被调用，首先判断 agent 能否在两个周期内使用 closeIntercept（对于守门员是 closeInterеptGoalie）截球。当两个周期内不能实现时，则 agent 使用一种迭代的方案即通过方法 getInterceptionPointBall 计算最优截球点。

SoccerCommand　dribble (AngDeg ang, DribbleT dribbleT)

该方法使 agent 盘球。即，与球一起移动，并把球保持一定的距离内。这实际上是重复做将球在一定的速度下踢向期望的方向，然后将其再次截住。参数 ang 表示盘球的方向。dribbleT 表示盘球的类型：

- DRIBBLE FAST: 速度较快的盘球动作，agent 每次将球踢向相对自己较远的前方。
- DRIBBLE SLOW: 速度较慢的盘球动作，agent 将球保持在相对自己较近的位置。
- DRIBBLE WITH BALL: 非常安全的盘球动作，agent 将球保持在离自己身体非常近的位置。

需要注意的是，该方法只在球离 agent 能够踢到球的时候才被调用。这意味着，它只负责整个盘球过程的踢球部分，就是说，它只将球向 ang 方向踢出一定的距离，如果 ang 和当前 agent 的身体角度之差大于 DribbleTurnAngle（当前为 30 度），则 agent 使用 turnWithBallTo 带球转向绝对角度 ang。此外，它使用 KickTo 方法，根据盘球的类型将球踢向期望方向的某点。之后，球会移出 agent 的踢球控制范围，因此 agent 将使用截球方法将球截住。当球被截住后，盘球动作再次被调用。这种踢球、截球的动作将往复执行，直到 agent 决定做其他动作。

SoccerCommand　directPass (VecPosition pos, PassT passType)

该方法使 agent 将球直接传给其他队员。参数 pos 表示传球的目的位置，通常是队友所在位

置。参数 passType 表示传球类型（PASS_NORMAL 或 PASS_FAST）。该方法调用 kickTo 将球以一个根据传球类型得出的期望的末速度踢向目标位置。

SoccerCommand　leadingPass (ObjectT o, double dDist)

该方法使 agent 进行 Leading Pass，Leading Pass 是指不将球直接传给接球队员，而是传到接球队员前方不远的地方，接球队员主动的去将球截住，加快进攻速度。该方法有两个参数，o 是指接球对象（通常是队友），dDist 表示球的目的位置与接球者的距离。它调用 kickTo 方法将球踢到对象 o 当前位置的前方 dDist 位置。这里的前方指的是 x 轴正向。注意，Leading Pass 球的期望末速度是 PassEndSpeed（当前为 1.4），因为球速太快容易丢球。

SoccerCommand　throughPass (ObjectT o, VecPosition posEnd, AngDeg *angMax=NULL)

该方法使 agent 进行空档传球，空档传球是指球并不直接传给某个队员，而是将球踢向对方防守队员或守门员，而我方进攻队员可以得到球。一个成功的 Through Pass 可以瓦解对方的防守，而使进攻队员带球靠近对方球门。参数 o 表示想象中的我方接球队员。为了确定踢球点的位置 p，从 agent 当前位置到参数 pos 作一条直线 l，然后计算球最安全轨道在这条直线上的点。球的最终速度将由当前球到 p 的位置和接球对象 o 到 p 点所需的周期数决定。如果球的末速度超出范围[MinPassEndSpeed .. MaxPassEndSpeed]则其将被规范为最近的边界。

SoccerCommand　outplayOpponent (ObjectT o, VecPosition pos, VecPosition *posTo=NULL)

该方法使 agent 突破对手，它常被用作进攻队员突破对方防守队员的防守。这是通过将球踢向防守队员身后的空地以此突破该队员的防守。该方法有两个参数 o 和 pos，o 表示 agent 要突破的对象，一般为对方防守队员；pos 表示 agent 的目的地。

SoccerCommand clearBall (ClearBallT type, SideT s=SIDE_ILLEGAL, AngDeg *angMax=NULL)

将球解围到场上指定位置。这通常被用作，当防守队员从进攻队员抢下，无法盘球或是传给队友，使用这种方法它可以将球从防守区域踢到前场。重要的是只有当在当前形势下别无选择的时候才进行该方法进行解围。这意味着要使用最大的力气，向对方球员的最大空档将球踢出。解围的方向是根据由参数传入的解围类型所决定的。有三种类型：

- CLEAR BALL DEFENSIVE: 将球从防守区域踢到由当前球的位置与球场中线组成的一个三角形区域。
- CLEAR BALL OFFENSIVE: 将球解围道越位位置。即由球当前位置和延伸禁区前沿与左右边线的交点。
- CLEAR BALL GOAL: 将球解围到球门前的三角区域。即由当前球的位置，与球门中心与禁区前沿中点构成的三角形。

SoccerCommand　mark (ObjectT o, double dDist, MarkT mark)

该方法允许 agent 标志一个对方球员。也就是一对一的防守，以减弱他在对方球队中的作用。它可以用作阻止对方球员接球或射门。该方法有三个参数，o 表示 mark 的对象，通常为对方球员，dDist 表示从 mark 位置到 o 的位置的期望距离，mark 为 mark 类型：

- MARK BALL: 通过站在对方队员与球之间的连线距离对方球员 dDist 处，这使对方球员难以接球。
- MARK GOAL: 通过站在对方球员与球门中央的连线距离对方球员 dDist 处，这使对方球员难以进球。
- MARK BISECTOR: 通过站在“球－对方球员－球门”构成的角的中线距离对方球员 dDist 处。 这可以防守对方球员的角度和 leading pass。

在决定完了位置之后，agent 调用 moveToPos 移动到目标位置。
SoccerCommand　defendGoalLine (double dDist)
该方法使 agent（一般是守门员）在球门线上防守。
SoccerCommand　interceptScoringAttempt ()
该方法返回一个截住飞向球门的球的命令。首先守门员决定球的轨道，然后站在球的轨道在球门前的某点上。
VecPosition　getThroughPassShootingPoint (ObjectT objTeam, VecPosition posEnd, AngDeg *angMax)
该方法返回空档传球的射门点。
VecPosition　getInterceptionPointBall (int *iCyclesBall, bool isGoalie)
该方法使用迭代的方法计算截球点。
VecPosition　getShootPositionOnLine (VecPosition p1, VecPosition p2, AngDeg *angLargest=NULL)
该方法返回在对方球员最大的角度与制定两点连线的交点。
double　getEndSpeedForPass (ObjectT o, VecPosition posPass)
该方法返回传球的末速度。
VecPosition　getMarkingPosition (ObjectT o, double dDist, MarkT mark)
该方法返回 mark 对象 o 的位置。

私有属性：

ActHandler *　ACT
WorldModel *　WM
ServerSettings *　SS
PlayerSettings *　PS

Connection.h

文件描述：

This file contains the class declarations for the Connection class, which sets up a connection with the soccer server.

struct　_socket

Detailed Description

Socket is a combination of a filedescriptor with a server adress.

注释：

套接字是一个文件描述符与服务器地址的结合。

公有属性：

int　socketfd	套接字文件描述符
sockaddr_in　serv_addr	服务器套接字信息

class　Connection

Detailed Description

This class creates a (socket) connection using a hostname and a port number. After the connection is created it is possible to send and receive messages from this connection. It is based on the client program supplied with the soccer server defined in client.c and created by Istuki Noda et al.
注释：
该类使用服务器名和端口号建立一个服务器连接。连接建立好以后就可以通过连接接收和发送消息。

共有方法：
Connection ()
缺省构造函数，只是设置最大信息长度。
Connection (const char *hostname, int port, int iSize)
构造函数。与指定服务器名或地址和端口号的服务器建立连接。
~Connection ()
析构函数。断开连接。
bool　connect (const char *host, int port)
与指定服务器名或地址和端口号的服务器建立连接。
void　disconnect (void)
断开当前 socket 连接。
bool　isConnected (void) const
判断是否已连接。
int　message_loop (FILE *in, FILE *out)

该方法保持循环状态等待输入。如果从 fpin 接到输入，则将该输入通过连接发送到服务器；如果从服务器接到消息，则将该消息送到 fpout。
int　receiveMessage (char *msg, int maxsize)
该方法从连接读入一个消息，如果没有消息则阻塞直到接到消息。
bool　sendMessage (const char *msg)
该方法通过连接向服务器发送消息。
void　show (ostream os)
该方法向指定的输出流打印是否处于连接状态。

私有变量：

Socket　m_sock	服务器通信协议
int　m_iMaxMsgSize	最大信息长度

Formations.h

文件描述：
Header file for different classes associated with Formations.

- PlayerTypeInfo contains the different information for one playertype (PT_DEFENDER, PT_ATTACKER, etc.). These should not be confused with the player_types used in the soccerserver from version 7.xx. The information consists of the attraction to the ball, minimal and maximal x coordinates, etc.
- FormationTypeInfo contains information for all the roles in one specific formation. This information consists of the current formation, the home position for all the roles, the player_type (PT_DEFENDER, PT_ATTACKER, etc.) for all the roles and the information about all player_types
- Formations itself. This class contains all the information of the different Formations. Furthermore it contains the current formation that is used and the role in the formation that is associated with the current agent.

class　Formations

Detailed Description

This class is a container for all different Formation Types: it contains the information of all the formation types. Furthermore it contains two other values: the current formation type that is used by the agent and the role of the agent in the current formation. These two values fully specify the

position of this player in the formation.

注释：

该类包含了所有不同的阵型类型信息。此外它还包含了另外 agent 以及 agent 在阵型中的角色所用到的两个类。这两个类完全确定该队员在阵型中的位置。

公有方法：

Formations (const char *strFile=NULL, FormationT ft=FT_ILLEGAL, int iNr=1)
构造函数。需要阵型配置文件，当前阵型，该阵型中的队员数作为参数。
void　show (ostream &os=cout)
该方法首先向指定的输出流输出所有的阵型信息，然后输出当前阵型以及当前 agent 在阵型中的角色。
VecPosition　getStrategicPosition (int iPlayer, VecPosition posBall, double dMaxXInPlayMode)
该方法返回一队员的战略位置。该位置是通过考虑该队员在当前阵型中的本位，结合球对当前队员的吸引值而得出的。战略位置的 x 坐标按如下方法计算：

阵型本位 x 坐标＋引力百分比×球位置 x 坐标

如果该值超出合法范围，则取相应的边界值。当球在身后，则战术位置的 x 坐标设置为球的 x 坐标。此外当战略位置在 dMaxXInPlayMode 前，则 x 坐标调整到 dMaxXInPlayMode。在一般的比赛模式下该值通常为越位位置。
bool　readFormations (const char *strFile)
该方法以如下形式从文件中读入阵型信息。

- 阵型中所有角色本位的 x 坐标
- 阵型中所有角色本位的 y 坐标
- 所有角色的球员类型
- 所有球员类型的 x 吸引百分比
- 所有球员类型的 y 吸引百分比
- 所有球员类型是否停留在球后
- 所有球员类型的最小 x 坐标
- 所有球员类型的最大 x 坐标

bool　setFormation (FormationT formation)
设置当前阵型。
FormationT　getFormation () const
返回当前阵型。
bool　setPlayerInFormation (int iNumber)
设置 agent 在当前阵型中的号码为 iNumber。
int　getPlayerInFormation () const
返回 agent 在当前阵型中的角色号码
PlayerT　getPlayerType (int iIndex=-1) const
返回 agent 在当前阵型中的球员类型。

私有属性：

FormationTypeInfo　formations [MAX_FORMATION_TYPES]
保存阵型信息。
FormationT　curFormation
当前阵型类型。
int　iPlayerInFormation
agent 在当前阵型中的角色号码。

class　FormationTypeInfo

Detailed Description

This class contains information about one specific formation. It contains the formation type (defined in SoccerTypes.h), the home position of all the roles (=specific player in a formation), the player types for all the roles and the information about the different player_types. Furthermore it contains methods to retrieve this information for a specific role.
注释：
该类包含 SoccerTypes.h 定义的阵型的信息。每个角色的本位，球员类型以及不同球员类型的信息。

公有方法：

FormationTypeInfo ()
构造函数。
void　show (ostream &os=cout)
以特定格式向输出流输出阵型信息。
bool　setFormationType (FormationT type)
设置阵型类型。
FormationT　getFormationType () const
bool　setPosHome (VecPosition pos, int atIndex)
设置该阵型中角色号码为 atIndex 的球员的本位。
bool　setXPosHome (double x, int atIndex)
bool　setYPosHome (double y, int atIndex)
VecPosition　getPosHome (int atIndex) const
bool　setPlayerType (PlayerT type, int atIndex)
设置该阵型中角色号码为 atIndex 的球员的类型。

PlayerT　getPlayerType (int atIndex) const
bool　setPlayerTypeInfo (PlayerTypeInfo info, int atIndex)
设置索引为 atIndex 的球员类型的信息。
PlayerTypeInfo *　getPlayerTypeInfo (int atIndex)
返回索引为 atIndex 的球员类型的信息。
PlayerTypeInfo *　getPlayerTypeInfoOfPlayer (int iPlayerInFormation)
返回角色号码为 iPlayerInFormation 的角色的球员类型信息。

私有属性：

FormationT　formationType	阵型信息
VecPosition　posHome [MAX_TEAMMATES]	本位数组
PlayerT　playerType [MAX_TEAMMATES]	球员类型数组
PlayerTypeInfo　playerTypeInfo [MAX_PLAYER_TYPES]	球员信息数组（索引为类型）

class　PlayerTypeInfo

Detailed Description

This class contains information for one individual player_type, defined in SoccerTypes.h. A player_type should not be confused with the player_types introduced in soccerserver 7.xx. A playerType PlayerT is defined as the kind of a player. Different possibilities are PT_ATTACKER, PT_MIDFIELDER_WING, etc. This class contains different characteristics of one playertype. This information consists of the following values:

- dAttrX - x attraction to the ball for this player type.
- dAtttY - y attraction to the ball for this player type.
- dMinX - minimal x coordinate for this player
- dMaxX - maximal x coordinate for this player
- bBehindBall - indicating whether this player type should always stay behind the ball or not.

This class contains different get and set methods to change the values associated for this class, normally these are changed when the Formations class reads in the formation file.

注释：

该类包含了 SoccerTypes.h 中定义的每一种球员类型的信息。

公有方法：

PlayerTypeInfo ()
构造函数。使用缺省值设置各变量。
PlayerTypeInfo (PlayerT, double, double, double, double, bool)
构造函数。使用参数中提供的数值设置各变量值。
bool　setValues (PlayerT, double, double, double, double, bool)
使用参数中提供的数值设置各变量的值。
void　show (ostream &os=cout)
向指定的输出流输出该类型的信息，以逗号分隔。
bool　setPlayerType (PlayerT type)
设置球员类型。
PlayerT　getPlayerType () const
bool　setAttrX (double attrX)
double　getAttrX () const
bool　setAttrY (double attrY)
double　getAttrY () const
bool　setMinX (double minX)
double　getMinX () const
bool　setMaxX (double maxX)
double　getMaxX () const
bool　setBehindBall (bool b)
bool　getBehindBall () const

私有属性：

PlayerT　playerType	球员类型
double　dAttrX	对于球的 x 吸引百分比
double　dAttrY	对于球的 y 吸引百分比
double　dMinX	最小 x 坐标
double　dMaxX	最大 x 坐标
bool　bBehindBall	该类型球员是否应该早在球后方

GenericValues.h

文件描述：

Header file for classes GenericValueT and GenericValues. The class GenericValueT contains a pointer to a variable of a generic type (double, char*, bool, int). This pointer is associated with a string. The class GenericValuescontains a collection of GenericValueT objects. All the member

data and member method declarations for both classes can be found in this file.

枚举类型：

enum GenericValueKind {
GENERIC_VALUE_DOUBLE = 0,
GENERIC_VALUE_STRING = 1,
GENERIC_VALUE_BOOLEAN = 2,
GENERIC_VALUE_INTEGER = 3 }
通用变量取值类型。

class GenericValueT

Detailed Description

This class contains a pointer to a variable of a generic type (double, char*, bool, int) and this pointer is associated with a string by which the variable can be reached. Several methods are defined which enable one to access the name and value of the variable.

注释：

这是一种共用的数据类型，它可以被设置为四种基本类型。它在构造时被赋以相应的名称，地址，和类型。提供了返回名称，设置、返回值，并可以向指定的目标输出该类的实例的属性。

公有方法：

GenericValueT (const char *strName, void *vAddress, GenericValueKind type)
构造函数，设置变量名，地址和类型。
~GenericValueT ()
析构函数
const char * getName ()
返回变量名。
bool setValue (const char *strValue)
设置变量的值。
char * getValue (char *strValue)

返回变量的值。
void　show (ostream &out, const char *strSeparator)
向指定的输出流，输出变量的名称、值，并以第二个参数的字符为间隔字符。

私有属性：

const char *　m_strName
变量名
void *　m_vAddress
变量地址
GenericValueKind　m_type
变量类型

class　GenericValues

Detailed Description

This class contains a collection of GenericValueT objects. This makes it possible to reference variables using string names. The class is an abstract class which should not be instantiated. It is the subclass of this class which contains the actual variables. In order to add a reference to a variable the method 'addSetting' must be used which associates the variables in the subclass with string names. The GenericValues class is used to read in configuration files. This now becomes very easy as long as one makes sure that the names in the configuration file match the string names associated with the corresponding variables

注释：

这个类是由一系列 GenericValueT 组成的。它有自己的名称 m_strClassName。GenericValueT 成员保存在一个数组中。每个成员有自己的名称、类型、值。这是一个抽象类。主要用于读入配置文件。

公有方法：

GenericValues (char *strName, int iMaxValues)
构造函数，指明该类的名称和最大数量。

virtual　~GenericValues ()
析构函数。
char *　getClassName ()
得到该实例的名称，即'm_strClassName'的值。
int　getValuesTotal ()
返回当前保存的成员的总个数。即'm_iValuesTotal'的值。
bool　addSetting (const char *strName, void *vAddress, GenericValueKind t)
向集合中添加一个新的通用类型变量。
virtual char *　getValue (const char *strName, char *strValue)
返回名称与第一个参数匹配的成员的值。该值被转化成字符类型，以第二个参数返回。
virtual bool　setValue (const char *strName, const char *strValue)
对于名称与第一个参数匹配的成员赋以第二个参数所指定的值。第二个参数总以字符串的形式给出相应的值，在赋值时转化成相应的类型。
virtual bool　readValues (const char *strFile, const char *strSeparator=0)
从文件中将一系列的值读入该实例。
virtual bool　saveValues (const char *strFile, const char *strSeparator=0, bool bAppend=true)
将该实例所保存的成员保存到文件中，并以第二个参数为分隔符。
virtual void　show (ostream &out, const char *strSeparator)
将所有的成员显示到第一个参数所指定的输出流上。以第二个参数为分隔符。

私有方法：

GenericValueT *　getValuePtr (const char *strName)
返回指向名称与参数匹配的成员的指针。

私有属性：

char *　m_strClassName
该组通用变量的名称。
GenericValueT **　m_values
指向保存所有通用变量数组的指针。
int　m_iValuesTotal
当前所保存的通用变量的数目。
int　m_iMaxGenericValues
该实例能保存的最大的通用变量数目。

Geometry.h

文件描述：

Header file for the classes VecPosition, Geometry, Line, Circle and Rectangle. All the member

data and member method declarations for all these classes can be found in this file together with some auxiliary functions for numeric and goniometric purposes.

常量定义：

#define EPSILON 0.0001
浮点型数据相等误差。

类型定义：

typedef double　AngRad
弧度
typedef double　AngDeg
角度

枚举类型：

enum CoordSystemT { CARTESIAN, POLAR }
坐标系类型。

全局函数：

double　max (double d1, double d2)
double　min (double d1, double d2)
int　sign (double d1)　　符号函数，正数返回 1，零和负数返回-1。
AngDeg　Rad2Deg (AngRad x)　　弧度转角度
AngRad　Deg2Rad (AngDeg x)　　角度转弧度
double　cosDeg (AngDeg x)
double　sinDeg (AngDeg x)
double　tanDeg (AngDeg x)
AngDeg　atanDeg (double x)
double　atan2Deg (double x, double y)
返回 y/x 的弧正切的住值，同时由 x,y 的符号决定象限。
AngDeg　acosDeg (double x)
AngDeg　asinDeg (double x)
bool　isAngInInterval (AngDeg ang, AngDeg angMin, AngDeg angMax)
角度是否在给定范围内
AngDeg　getBisectorTwoAngles (AngDeg angMin, AngDeg angMax)
返回角平分线

class Circle

Detailed Description

This class represents a circle. A circle is defined by one VecPosition (which denotes the center) and its radius.

注释：

圆类，由圆心坐标和半径决定。

公有方法：

Circle ()
构造函数。以圆心（-1000,-1000）半径 0，初始化圆。
Circle (VecPosition pos, double dR)
构造函数。以给定的参数初始化圆。
void show (ostream &os=cout)
以格式 c: (c_x,c_y), r: rad 向指定的输出流输出圆的信息。
bool setCircle (VecPosition pos, double dR)
设置圆的圆心与坐标。
bool setRadius (double dR)
double getRadius ()
bool setCenter (VecPosition pos)
VecPosition getCenter ()
double getCircumference ()
double getArea ()
bool isInside (VecPosition pos)
判断给定点是否在圆内。
int getIntersectionPoints (Circle c, VecPosition *p1, VecPosition *p2)
返回与给定圆的交点，以及交点的个数。
double getIntersectionArea (Circle c)
返回与给定圆重合的面积。

私有属性：

VecPosition　m_posCenter
圆心位置。
double　m_dRadius
半径

class　Geometry

Detailed Description

This class contains several static methods dealing with geometry.

注释：

该类包含若干处理几何问题的静态方法。

静态方法：

double　getLengthGeomSeries (double dFirst, double dRatio, double dSum)
给定几何序列（等比数列）的首项，公比，和。求序列的长度。
double　getSumGeomSeries (double dFirst, double dRatio, double dLength)
给定几何序列（等比数列）的首项，公比，长度。求该序列的和。
double　getSumInfGeomSeries (double dFirst, double dRatio)
无穷等比数列求和。
double　getFirstGeomSeries (double dSum, double dRatio, double dLength)
给定几何序列（等比数列）的和，公比，长度。求该序列的首项。
double　getFirstInfGeomSeries (double dSum, double dRatio)
已知无穷等比数列的和，公比。求首项。
int　abcFormula (double a, double b, double c, double *s1, double *s2)
返回一元二次方程的根和根的个数。

class　Line

Detailed Description

This class contains the representation of a line. A line is defined by the formula ay + bx + c = 0. The coefficients a, b and c are stored and used in the calculations.

注释：

直线类，以直线的标准形式的系数构成的三元组表示。

公有方法：

Line (double a, double b, double c)
构造函数。设置直线的三个系数。
void　show (ostream &os=cout)
向指定的输出流输出该直线。
VecPosition　getIntersection (Line line)
返回与给定直线的交点。
int　getCircleIntersectionPoints(Circle　circle,　VecPosition　*posSolution1,　VecPosition *posSolution2)
返回与给定的圆的交点和交点个数。
Line　getTangentLine (VecPosition pos)
返回给定点到该直线的垂线。
VecPosition　getPointOnLineClosestTo (VecPosition pos)
返回给定点在该直线上的投影点。
double　getDistanceWithPoint (VecPosition pos)
返回给定点到该直线的距离。
bool　isInBetween (VecPosition pos, VecPosition point1, VecPosition point2)
给定点在该直线上的投影是否在该直线上另外两点之间。
double　getYGivenX (double x)
对于直线上的点，给出 x 坐标求 y 坐标。
double　getXGivenY (double y)
对于直线上的点，给出 y 坐标求 x 坐标。
double　getACoefficient () const
返回系数 a。
double　getBCoefficient () const
返回系数 b。
double　getCCoefficient () const
返回系数 c。

静态公有方法：

Line makeLineFromTwoPoints (VecPosition pos1, VecPosition pos2)
由给定的两点作直线。
Line makeLineFromPositionAndAngle (VecPosition vec, AngDeg angle)
由一点和一角度作直线。

私有属性：

double m_a
double m_b
double m_c
三个系数。

友员：

ostream & operator<< (ostream &os, Line l)

class Rectangle

Detailed Description

This class represents a rectangle. A rectangle is defined by two VecPositions the one at the upper left corner and the one at the right bottom.

注释：

矩形类。由左上角和右下角的两点定义。

公有方法：

Rectangle (VecPosition pos, VecPosition pos2)
构造函数。
void show (ostream &os=cout)
向指定的输出流输出。

bool　isInside (VecPosition pos)
判断点是否在矩形内。
void　setRectanglePoints (VecPosition pos1, VecPosition pos2)
设置当前矩形的左上点和右下点。
bool　setPosLeftTop (VecPosition pos)
设置当前矩形的左上点。
VecPosition　getPosLeftTop (VecPosition pos)
返回当前矩形的左上点。
bool　setPosRightBottom (VecPosition pos)
设置当前矩形的右下点。
VecPosition　getPosRightBottom (VecPosition pos)
返回当前矩形的右下点。

私有属性：

VecPosition　m_posLeftTop
VecPosition　m_posRightBottom
矩形的左上点和右下点。

class　VecPosition

Detailed Description

This class contains an x- and y-coordinate of a position (x,y) as member data and methods which operate on this position. The standard arithmetic operators are overloaded and can thus be applied to positions (x,y). It is also possible to represent a position in polar coordinates (r,phi), since the class contains a method to convert these into Cartesian coordinates (x,y).

注释：

由(x,y)表示的笛卡尔坐标，或由(r,phi)表示的极坐标。
各类算术运算符已重载。

公有方法：（部分）

VecPosition　operator- ()
重载负号，两坐标同时取负。

VecPosition　operator+ (const double &d)
坐标与实数的相加，坐标的两项同时加上该实数。
VecPosition　operator+ (const VecPosition &p)
坐标与坐标的相加。
void　show (CoordSystemT cs=CARTESIAN)
以笛卡尔坐标或极坐标的形式向标准输出，输出坐标值。
string　str (CoordSystemT cs=CARTESIAN)
将当前坐标的笛卡尔坐标或极坐标形式输出到一个字符串中去。
bool　setX (double dX)
double　getX () const
bool　setY (double dY)
double　getY () const
void　setVecPosition (double dX=0, double dY=0, CoordSystemT cs=CARTESIAN)
double　getDistanceTo (const VecPosition p)
VecPosition　setMagnitude (double d)
设置当前坐标的模为 d。
double　getMagnitude () const
返回当前坐标的模。
AngDeg　getDirection () const
返回当前坐标的方向。
bool　isInFrontOf (const VecPosition &p)
当前坐标是否在给定坐标前（x 值大于 p 的 x 值）
bool　isInFrontOf (const double &d)
当前的坐标的 x 值是否大于 d。
bool　isBehindOf (const VecPosition &p)
bool　isBehindOf (const double &d)
bool　isLeftOf (const VecPosition &p)
bool　isLeftOf (const double &d)
bool　isRightOf (const VecPosition &p)
bool　isRightOf (const double &d)
bool　isBetweenX (const VecPosition &p1, const VecPosition &p2)
bool　isBetweenX (const double &d1, const double &d2)
bool　isBetweenY (const VecPosition &p1, const VecPosition &p2)
bool　isBetweenY (const double &d1, const double &d2)
VecPosition　normalize ()
标准化当前坐标。
VecPosition　rotate (AngDeg angle)
将当前坐标绕原点逆时针旋转 angle。
VecPosition　globalToRelative (VecPosition orig, AngDeg ang)
将绝对坐标转化为以 orig 为原点，ang 为 x 轴正向的相对坐标。
VecPosition　relativeToGlobal (VecPosition orig, AngDeg ang)
相对坐标转化成绝对坐标。
VecPosition　getVecPositionOnLineFraction (VecPosition &p, double dFrac)
返回该点距 p 点的线段上与该点的距离占该点与 p 的的距离为 dFrac 的点。

静态公有方法：

VecPosition　getVecPositionFromPolar (double dMag, AngDeg ang)
极坐标转笛卡尔坐标
AngDeg　normalizeAngle (AngDeg angle)
标准化角度。（-180,180）

私有属性：

double　m_x
double　m_y

友员：

ostream &　operator<< (ostream &os, VecPosition p)

Logger.h

文件描述：

This file contains the class to log information about the system to any output stream. A range can be specified for which the received log information is printed. Furthermore it is possible to print the time since the timer of the Logger has been restarted.

class　Logger

Detailed Description

This class makes it possible to log information on different abstraction levels. All messages are passed to the log method 'log' with a level indication. When it has been specified that this level should be logged using either the 'addLogLevel' or 'addLogRange' method the message is logged, otherwise it is ignored. This makes it possible to print only the information you are interested in.

There is one global Log class which is used by all classes that use the Logger. This instantation of the Logger is located in the file Logger.C and is called 'Log'. All classes that want use this Logger should make a reference to it using the line 'extern Logger Log;' and can then use this Logger with the Log.log(...) methods. Furthermore the Logger also contains a timer with makes it possible to print the time since the timer has been restarted.

公有方法：

Logger (ostream &os=cout, int iMinLogLevel=0, int iMaxLogLevel=0)
bool log (int iLevel, string str)
bool log (int i, char *str,...)
bool logWithTime (int iLevel, char *str,...)
void restartTimer ()
Timing getTiming ()
bool isInLogLevel (int iLevel)
bool addLogLevel (int iLevel)
bool addLogRange (int iMin, int iMax)
char * getHeader ()
bool setHeader (char *str)
bool setHeader (int i1, int i2)
bool setOutputStream (ostream &os)
ostream & getOutputStream ()
void showLogLevels (ostream &os)

私有属性：

Timing m_timing
char m_buf [MAX_LOG_LINE]
set< int > m_setLogLevels
char m_strHeader [MAX_HEADER]
ostream * m_os

class Timing

Detailed Description

This class holds a timer. This timer can be set (restartTime) and text can be printed with the elapsed time since the timer was restarted..

公有方法：
void　printTimeDiffWithText (ostream &os, char *str, int iFactor=1000)
double　getElapsedTime ()
void　restartTime ()

静态公有方法：
double　getTimeDifference (struct timeval t1, struct timeval t2)

私有属性：
timeval　time1

Objects.h

文件描述：

class declarations Object, DynamicObject, FixedObject, PlayerObject, BallObject and Stamina.

class　Object

Detailed Description

Class Object contains RoboCup information that is available for all objects in the simulation. All (relative) information is relative to an agent as declared in AgentObject. Update of an object (or one of the subclasses) happens by calling the standard get and set methods available in these classes. Calculations on these attributes do not occur in these classes, but in the update methods of the WorldModel.

注释：

该类包含所有比赛中所有对象可用的信息。与 agent 有关的信息在 AgentObject 中声明。类中提供了对自由属性的访问函数。但是计算这些属性是 WorldModel 中的 update 方法完成的。

公有方法：

Object ()
构造函数。所有属性按照缺省值设置。
virtual void　show (ostream &os=cout)=0

抽象函数。

AngDeg　getRelativeAngle ()

返回该对象对于当前 agent 的相对角度。

double　getRelativeDistance ()

返回给对象对于当前 agent 的相对距离。

double　getConfidence (Time time)

返回该对象信息的可信度。该可信度是与上次看到这个对象以及特定的时间（一般为上次接受信息的时间）相关的。

bool　setType (ObjectT o)

设置对象的类型。

ObjectT　getType () const

返回该对象的类型。

bool　setRelativePosition (double dDist, AngDeg dAng, Time time)

设置给对象的相对位置，以及接收该信息的时间。相对位置是由距离和角度计算出的。

bool　setRelativePosition (VecPosition v, Time time)

设置给对象的相对位置，以及接收该信息的时间。新的相对位置是由坐标给出的。

VecPosition　getRelativePosition () const

返回相对位置。

bool　setTimeRelativePosition (Time time)

设置接收到相对位置的时间。

Time　getTimeRelativePosition () const

返回接收到相对位置的时间。

bool　setGlobalPosition (VecPosition p, Time time)

设置全局位置，以及该位置计算的时间。

VecPosition　getGlobalPosition () const

bool　setTimeGlobalPosition (Time time)

Time　getTimeGlobalPosition () const

bool　setGlobalPositionLastSee (VecPosition p, Time time)

VecPosition　getGlobalPositionLastSee () const

bool　setTimeGlobalPosDerivedFromSee (Time time)

Time　getTimeGlobalPosDerivedFromSee () const

bool　setTimeLastSeen (Time time)

Time　getTimeLastSeen () const

私有属性：

ObjectT　objectType	对象类型。
Time　timeLastSeen	上次视觉信息接收时间。
VecPosition　posGlobal	全局位置。
Time　timeGlobalPosition	更新全局位置的服务器时间。
VecPosition　posRelative	相对 agent 的位置
Time　timeRelativePosition	更新相对位置的服务器时间
VecPosition　posGlobalLastSee	上次接收到视觉信息时的全局位置。

Time　timeGlobalPosDerivedFromSee　　从上次视觉信息得出的位置的时间

class　FixedObject

Detailed Description

Class FixedObject contains RoboCup information that is available for objects that cannot move (flags, goals, lines). No additional information is added to the superclass Object.
注释：
固定对象类，该类记录了不能移动的对象的信息。（如标志，球门，线）

公有方法：

VecPosition　getGlobalPosition (SideT s, double dGoalWidth=14.02) const
返回该对象的全局位置。只有当该对象是标志或者球门时才起作用。由于对于某些标志，球门的宽度是必要的，所以球门的宽度也当作参数传入。
AngDeg　getGlobalAngle (SideT s)
得到该对象的全局角度。只有当对象为线的时候才起作用。
void　show (ostream &os=cout)
将该固定对象的所有信息输出到指定输出流上。

class　DynamicObject

Detailed Description

Class DynamicObject contains RoboCup information that is available for objects that can move (players, ball). Different variables are added to the superclass Object

注释：

动态对象类，该类记录了移动对象的信息。

公有方法：

DynamicObject ()
构造函数。所有属性值被赋以缺省值。
bool　setRelativeDistanceChange (double d, Time time)
设置相对距离改变，以及改变**信息**的时间。
double　getRelativeDistanceChange () const
bool　setRelativeAngleChange (double d, Time time)
设置相对角度改变，以及改变**信息**的时间。
double　getRelativeAngleChange () const
bool　setTimeChangeInformation (Time time)
设置改变信息的时间。
Time　getTimeChangeInformation () const
bool　setGlobalVelocity (VecPosition v, Time time)
设置绝对速度，以及改变绝对速度的时间。
VecPosition　getGlobalVelocity () const
double　getSpeed () const
返回速率，该速率为当前速度的模。
bool　setTimeGlobalVelocity (Time time)
设置改变绝对速度的时间。
Time　getTimeGlobalVelocity () const

私有属性：

VecPosition　vecGlobalVelocity
绝对速度。注意这是一个矢量。
Time　timeGlobalVelocity
记录绝对速度的时间。
double　dRelativeDistanceChange
相对距离的改变。
double　dRelativeAngleChange
相对角度的改变。
Time　timeChangeInformation
改变信息的时间。

class　BallObject

Detailed Description

Class PlayerObject contains RoboCup information that is available for the ball. No extra variables are added to superclass DynamicObject

注释：

该类包含球的所有信息，对 DynamicObject 并没有增加新的变量。

公有方法：

BallObject ()
构造函数。初始化。
void　show (ostream &os=cout)
将球的信息输出到指定的输出流。

class　PlayerObject

Detailed Description

Class PlayerObject contains RoboCup information that is available for players. Different variables are added to the superclass DynamicObject

注释：

球员类。

公有方法：

PlayerObject ()
构造函数。所有属性设为初始值。
void　show (ostream &os=cout)
向指定的输出流输出队员信息，使用缺省的队名：Team_L。
void　show (const char *strTeamName, ostream &os=cout)
使用指定的队名向指定的输出流输出球员的信息。
bool　setPossibleRange (ObjectT objMin, ObjectT objMax)
设置可能的类型范围。

bool　isInRange (ObjectT obj)
指定对象是否在所设定的可能类型范围内。
ObjectT　getMinRange ()
返回可能类型范围的下届。
ObjectT　getMaxRange ()
返回可能类型范围的上界。
bool　setIsKnownPlayer (bool b)
设置该动态对象是否为已知队员（队号已知）。如果未知，则设 isKnownPlayer 为假，并将其放入一个位置球员队列。
bool　getIsKnownPlayer () const
bool　setIsGoalie (bool b)
bool　getIsGoalie () const
bool　setRelativeBodyAngle (AngDeg ang, Time time)
AngDeg　getRelativeBodyAngle () const
bool　setGlobalBodyAngle (AngDeg ang, Time time)
AngDeg　getGlobalBodyAngle () const
bool　setRelativeNeckAngle (AngDeg ang, Time time)
AngDeg　getRelativeNeckAngle () const
bool　setGlobalNeckAngle (AngDeg ang, Time time)
AngDeg　getGlobalNeckAngle () const
bool　setTimeRelativeAngles (Time time)
Time　getTimeRelativeAngles () const
bool　setTimeGlobalAngles (Time time)
Time　getTimeGlobalAngles () const

保护属性：

bool　isKnownPlayer
该球员号码的可信度是否足够高。
ObjectT　objRangeMin
对象类型范围的最小值
ObjectT　objRangeMax
对象范围的最大值
bool　isGoalie
是否是守门员
AngDeg　angGlobalBodyAngle
绝对身体角度。
AngDeg　angGlobalNeckAngle
绝对脖子角度。
Time　timeGlobalAngles
绝对角度改变时间。

私有属性：

AngDeg　angRelativeBodyAngle
该球员相对于当前球员的身体角度。
AngDeg　angRelativeNeckAngle
该球员相对于当前球员的脖子角度。
Time　timeRelativeAngles
改变相对角度的服务器时间。

class　AgentObject

Detailed Description

Class AgentObject contains RoboCup information that is available for the agent. New variables are declared that extend a normal PlayerObject.

公有方法：

AgentObject (double dStaminaMax=4000)
构造函数。
void　show (ostream &os=cout)
向指定的输出流输出当前 agent 信息，使用缺省的队名：Team_L。
void　show (const char *strTeamName, ostream &os=cout)
使用指定的队名向指定的输出流输出球员的信息。
VecPosition　getPositionDifference () const
bool　setPositionDifference (VecPosition v)
ViewAngleT　getViewAngle () const
bool　setViewAngle (ViewAngleT v)
ViewQualityT　getViewQuality () const
bool　setViewQuality (ViewQualityT v)
Stamina　getStamina () const
bool　setStamina (Stamina sta)
VecPosition　getSpeedRelToNeck () const
bool　setSpeedRelToNeck (VecPosition v)
bool　setGlobalNeckAngle (AngDeg ang)
AngDeg　getBodyAngleRelToNeck () const
bool　setBodyAngleRelToNeck (AngDeg ang)

私有属性：

ViewAngleT　viewAngle	视角
ViewQualityT　viewQuality	视觉质量
Stamina　stamina	体力
VecPosition　velSpeedRelToNeck	相对于脖子的速度
AngDeg　angBodyAngleRelToNeck	相对于脖子的身体角度
VecPosition　posPositionDifference	位置差

class　Stamina

Detailed Description

The following stamina information is stored in this class.

- actual stamina
- recovery, determines how much stamina recovers each cycle, decreases below a certain threshold (never increases)
- effort, determines which percentage of dash power is actually used. decreases below certain threshold, increases above a higher threshold.

注释：
包含服务器例所定义的与体力相关的参数。

公有方法：

Stamina (double dSta=4000.0, double dEff=1.0, double dRec=1.0)
构造函数。给各成员变量赋值。
void　show (ostream &os=cout)
向指定的输出流输出体力相关的信息。
TiredNessT　getTiredNess (double dRecDecThr, double dStaminaMax)
返回一个 TiredNessT 型的枚举变量，以表示 agent 当前的体力状况。
double　getStamina () const
bool　setStamina (double d)
double　getEffort () const
bool　setEffort (double d)
double　getRecovery () const
bool　setRecovery (double d)

私有属性

double　m_dStamina
double　m_dEffort
double　m_dRecovery

Parse.h

class　　Parse

Player.h

文件描述：

This file contains the declaration for the Player class, which is a superclass from BasicPlayer and contains the decision procedure to select the skills from the BasicPlayer.

全局函数：
void *　stdin_callback (void *v)

class　　Player

Detailed Description

This class is a superclass from **BasicPlayer** and contains a more sophisticated decision procedure to determine the next action.

公有方法：

Player (ActHandler *a, WorldModel *wm, ServerSettings *ss, PlayerSettings *cs, Formations *fs,

char *strTeamName, double dVersion, int iReconnect=-1)
构造函数。
void　mainLoop ()
主函数。
void　deMeer5 ()
完成由简单决策构成的队伍。
void　deMeer5_goalie ()
完成一个简单决策的守门员。
void　handleStdin ()
该方法监听键盘输入，然后将输入转化成相应的动作。
void　showStringCommands (ostream &out)
该方法打印用户可能输入的命令。
bool　executeStringCommand (char *str)
该方法执行用户输入的命令。
void　test_only_update ()
该方法在接收一个新的 sense_body 后更新世界模型。

私有方法：

void　goalieMainLoop ()
守门员主循环。
void　defenderMainLoop ()
防守队员主循环。
void　midfielderMainLoop ()
中场队员主循环。
void　attackerMainLoop ()
进攻队员主循环。
bool　shallISaySomething ()
是否队员该说消息。
bool　amIAgentToSaySomething ()
是否该将我的世界模型与其他队员交流。
SoccerCommand　sayBallStatus ()
与其他队员交流球的状态。

私有属性：

Formations *　formations
bool　bContLoop
Time　m_timeLastSay

PlayerSettings.h

文件描述：

This file contains the class definitions for the PlayerSettings class that contains the specific (threshold) settings which determines the situation in which certain actions are performed by the client.

class　PlayerSettings

Detailed Description

This class contains all the settings that are important for the client (agent) to determine its actions. It contains mostly threshold values to determine whether a certain kind of actions should be taken or not. Furthermore this class contains all the standard set- and get methods for manipulating these values. Although it is normally not the case that these values are changed at runtime. The PlayerSettings class is a subclass of the GenericValues class and therefore each value in this class can be reached through the string name of the corresponding parameter. This may be helpful when the parameters are taken from a configuration file.

注释：

该类包含了所有对于智能体动作决策重要的变量设置。同时给出了返回和修改这些变量值的方法。

公有方法：（部分）

PlayerSettings::PlayerSettings()
构造函数。设置所有的属性值，并将一般变量（generic value），赋以相应的名称和属性值。

私有属性：
double　dPlayerConfThr
double　dPlayerHighConfThr
double　dBallConfThr
double　dPlayerDistTolerance
double　dPlayerWhenToTurnAngle
double　dPlayerWhenToKick

int iPlayerWhenToIntercept
double dClearBallDist
double dClearBallOppMaxDist
double dClearBallToSideAngle
double dConeWidth
double dPassEndSpeed
double dFastPassEndSpeed
double dPassExtraX
double dFractionWaitNoSee
double dFractionWaitSeeBegin
double dFractionWaitSeeEnd
double dMarkDistance
double dStratAreaRadius
double dShootRiskProbability
int iCyclesCatchWait
int iServerTimeOut
double dDribbleAngThr
double dTurnWithBallAngThr
double dTurnWithBallFreezeThr

SenseHandler.h

文件描述：

This file contains the class SenseHandler that is used to process the information coming from the server.

全局函数：

void * sense_callback (void *v)
该函数用来启动 Sense 线程。

全局变量：

Logger Log

class SenseHandler

Detailed Description

Class SenseHandler receives input from a (Robocup Simulation) server (through an instance of Connection). The class contains methods to parse the incoming messages and sends these to the WorldModel accordingly. After this class is instantiated, a specific thread must call the function sense_callback. This thread will then automatically call handleMessagesFromServer() which loops forever. In this way all messages are received and parsed (since the receiveMessage from the Connection blocks till a message arrives). Other threads can think about the next action while the SenseHandler sends the new information to the WorldModel.

注释：

该类通过连接接收服务器发来的消息。它包含了很多方法，用于解析接到的消息并将结果送到相应的世界模型中。当该类实例化后，一个特殊的线程必须调用全局函数函数 sense_callback。然后该线程将自动调用 handleMessagesFromServer()，该函数将一直保持循环状态。通过这种方式，所有的消息将以这种方式被接收然后解析。

公有方法：

SenseHandler (Connection *c, WorldModel *wm, ServerSettings *ss, PlayerSettings *ps)
构造函数。需要一个对连接的引用和一个对世界模型的引用。
void handleMessagesFromServer ()
该类的主程序，保持循环，接收、解析收到的消息。
void setTimeSignal ()
该方法设置时间信号。这是将下一个动作发送到服务器前所要等待的时间。当一个 sense 收到一个感觉信息后该方法被调用。
bool analyzeMessage (char *strMsg)
根据给定消息的类型调用相应的方法，分析给定的消息。
bool analyzeSeeGlobalMessage (char *strMsg)
该方法分析一个视觉消息。其所包含的不同对象的信息将被送往世界模型。
bool analyzeSeeMessage (char *strMsg)
该方法分析一个视觉信息。它从服务器得到时间，并试着与服务器保持同步。然后将消息存储在世界模型中，当调用 update 的时候更新。
bool analyzeSenseMessage (char *strMsg)
该方法分析一个感觉消息。其所包含的不同对象的信息将被送往世界模型。
bool analyzeInitMessage (char *strMsg)
该方法分析一个初始化消息。其所包含的初始化信息将被送往世界模型。

bool　analyzeHearMessage (char *strMsg)
该方法分析一个听觉消息。当该消息是发自裁判的，则根据其所包含的信息更新比赛模式，调整比分差；如果该消息是发自其他球员的，则调用 analyzePlayerMessage 进行处理。
bool　analyzePlayerMessage (char *strMsg)
该方法分析一个球员间通信消息。首先，该方法检查该信息是否是来自队友，对方球员的消息将被忽略，然后将该消息解析，所获得的信息存储在世界模型中，当更新世界模型时进行处理。
bool　analyzeChangePlayerTypeMessage (char *strMsg)
该方法分析一个改变球员类型消息。它检查是否当前 agent 要改变类型，如果是根据该类型调整 ServerSettings 。
bool　analyzeServerParamMessage (char *strMsg)
该方法分析 server_param 消息。该消息包含了所有服务器参数。所有 ServerSettings 中的设置将根据给出的值而改变。这将使从 server.conf 读出的配置值作废。
bool　analyzeCheckBall (char *strMsg)
该方法分析 check_ball 消息。该消息只有 coach 能够借收。它设置世界模型中的球的状态。
bool　analyzePlayerTypeMessage (char *strMsg)
该方法分析一个球员类型消息。该消息包含一个异构球员的类型值。这些值将被解析出并提供给世界模型中方法 processNewHeteroPlayer。
bool　analyzePlayerParamMessage (char *strMsg)
该方法分析 player_param 消息。它包含了异构球员各属性值的取值范围。
bool　readServerParam (char *strParam, char *strMsg)
该方法读出由字符串 strParam 所包含的服务器参数。ServerSettings 中的各属性将根据这些参数设置。

私有变量：

WorldModel *　WM	世界模型
ServerSettings *　SS	服务器设置
PlayerSettings *　PS	球员设置
Connection *　connection	与服务器的连接
int　iTimeSignal	感觉动作之前等待时间
int　iTriCounter	表示视觉信息何时到来
int　m_iSeeCounter	用来表示一个周期内 see 消息的次数
int　iSimStep	服务器周期的长度
itimerval　itv	发送动作的计时器

ServerSettings.h

文件描述：

Header file for class ServerSettings. It contains all the member data and member method declarations of the ServerSettings class. This class contains all the Soccerserver parameters that

are available in the configuration files 'server.conf' and 'player.conf' along with their default values and standard set- and get methods for manipulating these values. This file also contains the definition of the HeteroPlayerSettings class which contains all the SoccerServer parameters which together define a heterogeneous player type.

class HeteroPlayerSettings

Detailed Description

This class contains all the SoccerServer parameters which together define a heterogeneous player type. For each player type these parameters are initialized when the server is started.

注释：
该类包含所有定义异构球员的参数。每个球员的类型参数将在服务器启动时初始化。

公有方法：
void show (ostream &os=cout)
向指定的输出流输出所有该异构球员类型的属性值。

公有属性：
double dPlayerSpeedMax
球员最大速度。
double dStaminaIncMax
Stamina 每周期恢复率量
double dPlayerDecay
每周期球员速度衰减率。
double dInertiaMoment
惯量，决定球员在某一速度的转动角度。
double dDashPowerRate
dash 命令中 Power 所乘的比率。
double dPlayerSize
球员大小。
double dKickableMargin
球员踢球余量。踢球范围＝kickable_margin + ball_size + player_size
double dKickRand
在踢球方向上的随机错误。
double dExtraStamina
异构球员的额外 Stamina。
double dEffortMax
球员 effort 最大值。
double dEffortMin

球员 effort 最小值。

class　ServerSettings

Detailed Description

This class contains all the Soccerserver parameters that are available in the configuration file 'server.conf' along with their default values and standard set- and get methods for manipulating these values. The ServerSettings class is a subclass of the GenericValues class and therefore each value in this class can be reached through the string name of the corresponding parameter.

注释：
该类包含了服务器配置文件中的所有参数，和对这些参数赋值，取值的方法。

公有方法：（部分）
ServerSettings ()
构造函数，设置所有服务器参数。
bool　setValue (const char *strName, const char *strValue)
设置第一个参数指定的变量的值为第二个参数所指定的值。
bool　readValues (char *strFilename, char *Separator)
从服务器配置文件中将读出配置值并赋给相应的变量。

SoccerTypes.h

文件描述：

This file contains the different enumerations and constants that are important in the Soccer Server. Furthermore it contains the class SoccerCommand which is used to denote the different possible soccer commands and the class SoccerTypes that contains all kind of static methods to translate text strings that are received from the server into the soccer types (=enumerations) that are defined here. Finally it contains the Time class which holds a two-tuple that represents the time in the soccer server.

常量定义：
#define　MAX_TEAMMATES　11　　最多队友数
#define　MAX_OPPONENTS　11　　最多对受数
#define　MAX_PLAYER_TYPES　8　　最多球员类型
#define　MAX_FORMATION_TYPES　7　　最多阵型数

#define　MAX_HETERO_PLAYERS　7　　最多异构球员数
#define　MAX_COMMANDS　10　　可储存的最多命令数
#define　MAX_MSG　3052　　server 的最大信息长度
#define　MAX_SAY_MSG　10　　say 消息的最大长度
#define　MAX_TEAM_NAME_LENGTH　64　　最大队名长度
#define　MAX_FLAGS　55　　场上最多标志数
#define　MAX_LINES　4　　场上最多线的数目
#define　DEFAULT_TEAM_NAME　"Team_L"　　缺省本队队名
#define　DEFAULT_OPPONENT_NAME　"Team_R"　缺省对手队名
#define　PITCH_LENGTH　105.0　　场地长度
#define　PITCH_WIDTH　68.0　　场地宽度
#define　PITCH_MARGIN　5.0　　场边空地长度
#define　PENALTY_AREA_LENGTH　16.5　　禁区长度
#define　PENALTY_AREA_WIDTH　40.32　　禁区宽度
#define　PENALTY_X　(PITCH_LENGTH/2.0-PENALTY_AREA_LENGTH)
对手禁区前沿线

枚举类型：

enum	ObjectT	对象类型
enum	ObjectSetT	对象集合类型
enum	PlayModeT	比赛模式类型
enum	RefereeMessageT	裁判消息类型
enum	ViewAngleT	视角类型
enum	ViewQualityT	视觉质量类型
enum	SideT	边类型
enum	CommandT	命令类型
enum	PlayerT	球员位置类型
enum	FormationT	阵型类型
enum	BallStatusT	球状态类型
enum	ActionT	动作类型
enum	MarkT	标志类型
enum	DribbleT	盘球类型
enum	PassT	传球类型
enum	ClearBallT	解围类型
enum	TiredNessT	疲劳类型

全局变量：
const double　UnknownDoubleValue = -1000.0
const AngDeg　UnknownAngleValue = -1000.0
const int　UnknownIntValue = -1000
const int　UnknownTime = -20
const long　UnknownMessageNr = -30
各类非法值。

class　SoccerCommand

Detailed Description

This class resembles a SoccerCommand that contains all the information about a command that can be sent to the server, but the format is independent from the server implementation. A SoccerCommand can be created and before it is sent to the server, be converted to the actual string representation understood by the server.

注释：

该类以字符串的形式表示命令，其格式自由与 server 的格式不同，在将其传给 server 再将相应的命令转化成标准格式。
该类完成的工作如下：根据传入的参数，建立一个容易理解，且可以灵活改变的命令实例，当对命令处理完后，再将命令转化成 server 可以理解的命令字符串。

公有方法：

SoccerCommand (CommandT com=CMD_ILLEGAL,
　　double d1=UnknownDoubleValue,
　　double d2=UnknownDoubleValue,
　　double d3=UnknownDoubleValue)
构造函数。根据传递的参数建立一个命令。不同的命令类型决定参数的意义。
SoccerCommand (CommandT com, char *msg)
命令类型为 say message 时的构造函数。
void　makeCommand (CommandT com,
　　double d1=UnknownDoubleValue,
　　double d2=UnknownDoubleValue,
　　double d3=UnknownDoubleValue)
根据不同的命令类型，创建命令。

- CMD_DASH: d1 = power for dashing
- CMD_TURN: d1 = angle body is turned
- CMD_TURNNECK d1 = angle neck is turned
- CMD_KICK d1 = power for kick command, d2 = angle for kick
- CMD_MOVE d1 = x position, d2 = y position, d3 = body_angle
- CMD_CATCH d1 = catch angle
- CMD_CHANGEPLAYER d1 = player number, d2 = nr of new player type
- CMD_ATTENTIONTO d1 = which team, d2 = player number

void　makeCommand (CommandT com, ViewAngleT v, ViewQualityT q)
创建一个命令类型为 CMD_CHANGEVIEW 的命令。
void　makeCommand (CommandT com, char *msg)
创建一个命令类型为 CMD_SAY 的命令。
bool　isIllegal ()
返回该命令是否合法。即该命令的类型是否已被赋以相应的值。
void　show (ostream &os)
向标准输出输出该命令。
char *　getCommandString (char *str, ServerSettings *ss)
将该命令转化为 server 能够理解的命令字符串。将 ServerSettings 的引用传入以检查命令是否合法。

共有属性：

不同的命令，相应的属性起作用。

Time　time	命令发出的时间，将由 worldmodel 设定。
CommandT　commandType	命令类型。
double　dAngle	该命令的角度。
double　dPower	该命令的 Power
ViewQualityT　vq	该命令的视觉质量
ViewAngleT　va	该命令的视角
double　dX	该命令坐标的 x 值
double　dY	该命令坐标的 y 值
char　str [MAX_SAY_MSG]	say 的内容
int　iTimes	命令放入队列中需要多少周期？？

私有方法：

bool　makeCatchCommand (char *str)
bool　makeChangeViewCommand (char *str)
bool　makeDashCommand (char *str)
bool　makeKickCommand (char *str)
bool　makeMoveCommand (char *str)
bool　makeSayCommand (char *str)
bool　makeSenseBodyCommand (char *str)
bool　makeTurnCommand (char *str)
bool　makeTurnNeckCommand (char *str)
bool　makeChangePlayerCommand (char *str)
bool　makeAttentionToCommand (char *str)
由类的属性构造 server 能够理解的命令字符串。

私有属性：

ServerSettings *　SS
用于检查命令是否合法的 ServerSettings 引用。

class　SoccerTypes

Detailed Description

The class SoccerTypes contains different methods to work with the different enumerations defined in SoccerTypes.h. It is possible to convert soccertypes to strings and strings to soccertypes. It is also possible to get more specific information about some of the soccertypes. All methods are static so it is possible to call the methods without instantiating the class.

注释：

包含对 SoccerTypes.h 定义的枚举类型的操作，和这些枚举类型与 SoccorServer 所规定的字符串型之间的转化。

静态公有方法：

char *　getObjectStr (char *strBuf, ObjectT o, const char *strTeam)
返回枚举目标 o 的标准形式的字符串。
ObjectT　getObjectFromStr (char **str, bool *isGoalie, const char *str)
由字符串转化成枚举类型的目标。返回该目标，以及是否是守门员，如果是我方球员，第三个参数返回我方队名。
bool　isInSet (ObjectT o, ObjectSetT o_s)
判断目标 o 是否是目标集合 o_s 的成员。
bool　isFlag (ObjectT o)
是否是标志。
bool　isLine (ObjectT o)
是否是边线或底线。
bool　isGoal (ObjectT o)
ObjectT　getOwnGoal (SideT s)
返回我方球门。
ObjectT　getGoalOpponent (SideT s)
返回对方球门。
bool　isBall (ObjectT o)

bool　isTeammate (ObjectT o)
bool　isOpponent (ObjectT o)
bool　isGoalie (ObjectT o)
bool　isPlayer (ObjectT o)
bool　isKnownPlayer (ObjectT o)
int　getIndex (ObjectT o)
返回该目标在其目标集合中的索引，其结果比其真正的索引小 1。
ObjectT　getTeammateObjectFromIndex (int iIndex)
由索引得到队友对象。
ObjectT　getOpponentObjectFromIndex (int iIndex)
VecPosition　getGlobalPositionFlag (ObjectT o, SideT s, double dGoalWidth=14.02)
返回相对于 s 方的某标志的绝对坐标。
AngDeg　getGlobalAngleLine (ObjectT o, SideT s)
边／底线（中点）对于 s 方的角度。
PlayModeT　getPlayModeFromStr (char *str)
将字符串中解析出比赛模式
PlayModeT　getPlayModeFromRefereeMessage (RefereeMessageT rm)
从作为裁判消息的字符串中解析出比赛模式。（均非字符串型）
char *　getPlayModeStr (PlayModeT p)
将 PlayModeT 型表示的比赛模式转化成字符串形式表示。
char *　getRefereeMessageStr (RefereeMessageT r)
将 RefereeMessageT 型的裁判消息转化成字符串形式的。
RefereeMessageT　getRefereeMessageFromStr (char *str)
将字符串形式表示的裁判消息转化成 RefereeMessageT 表示的裁判消息。
char *　getViewAngleStr (ViewAngleT v)
将 ViewAngleT 类型表示的视角转化成字符串形式的视角。
ViewAngleT　getViewAngleFromStr (char *str)
将字符串形式的视角转化成 ViewAngleT 形式的视角。
AngDeg　getHalfViewAngleValue (ViewAngleT va)
得到角度表示的相应的 ViewAngleT 型角度的一半。
char *　getViewQualityStr (ViewQualityT v)
将 ViewQualityT 型表示的视觉质量转化成字符串形式表示的视觉质量。
ViewQualityT　getViewQualityFromStr (char *str)
将字符串形式表示的视觉质量转化成 ViewQualityT 型表示的视觉质量。
char *　getCommandStr (CommandT com)
从 CommandT 类型表示的命令转化成字符形式表示的命令。
bool　isPrimaryCommand (CommandT com)
判断命令中的动作是不是主要动作，主要动作在一个周期内只能执行一次。即 kick, dash, move, turn 和 catch。
char *　getSideStr (SideT s)
将 SideT 型的边类型转化成字符串形式表示的。
SideT　getSideFromStr (char *str)
将字符串形式表示的边类型转化成 SideT 型的。
char *　getBallStatusStr (BallStatusT bs)

将 BallStatusT 表示的足球状态转化成字符串型表示。
BallStatusT　getBallStatusFromStr (char *str)
将字符串表示的足球状态转化成 BallStatusT 型的。

class　Time

Detailed Description

This class contains the time representation of the soccer server. It is represented by an ordered pair (t,s) where t denotes the current server cycle and s is the number of cycles since the clock has stopped. Here the value for t equals that of the time stamp contained in the last message received from the server, whereas the value for s will always be 0 while the game is in progress. It is only during certain dead ball situations (e.g. an offside call leading to a free kick) that this value will be different, since in these cases the server time will stop while cycles continue to pass (i.e. actions can still be performed). Representing the time in this way has the advantage that it allows the players to reason about the number of cycles between events in a meaningful way.

注释：

该类包含了对服务器时间的表示形式。它由一个序偶(t,s)组成。t 表示当前服务器的周期数，s 表示服务器时钟停止的周期数。t 的值等于从服务器收的的最后一个信息的时间戳。服务器运行的时候 s 的值恒为 0，只有当死球的时候 s 的值才会有所变化。

静态公有方法：（部分）

Time (int iTime=-1, int iStopped=0)
构造函数。设置两个私有序性。
bool　updateTime (int iTime)
更新时间为 iTime。如果当前时间为 iTime，则 t 不变 s 自加 1。否则，将 iTime 的值赋给 t，s 置 0。
bool　setTimeStopped (int iTime)
设置 s 为 iTime。返回是否设置成功。
int　getTime ()
返回当前时间。
int　getTimeStopped ()
返回时钟停止时间。
int　getTimeDifference (Time t)
如果当前时间对象与时间对象 t 相等，则返回当前对象与 t 的 s 属性的差。否则返回当前对象与 t 的 t 属性的差。

bool　isStopped ()
检查 s 的值，判断时钟是否停止。
Time　getTimeAddedWith (int iCycles)
返回将 iCycles 加到当前时间对象而得到的新的时间对象。
bool　addToTime (int iCycles)
将 iCycle 加到当前时间对象上。
void　show (ostream &os=cout)
将时间以序偶的形式输出到指定的输出流上。
Time　operator+ (const int &i)
重载加号运算符。返回将 i 加到当前的对象上得到的新对象。当前对象不改变。
Time　operator+ (Time t)
重载加号运算符。返回当前对象与 t 的和：(t1,s1) + (t2,s2) = (t1+t2,s2)。

私有属性：

int　m_iTime
当前时间周期数。
int　m_iStopped
时钟停止周期数。

WorldModel.h

文件描述：

class declarations of WorldModel. This class contains methods that give information about the current and future state of the world (soccer field).

公有变量：

Logger　Log

class　WorldModel

Detailed Description

The Class WorlModel contains all the RoboCup information that is available on the field. It contains information about the players, ball, flags and lines. Furthermore it contains methods to extract useful information. The (large amount of) attributes can be separated into different groups:

- Environmental information: specific information about the soccer server
- Match information: general information about the current state of a match
- Object information: all the objects on the soccer field
- Action information: actions that the agent has performed

The methods can also be divided into different groups:

- Retrieval methods: directly retrieving information of objects
- Update methods: update world based on new sensory information
- Prediction methods: predict future states based on past perceptions
- High-Level methods: deriving high-level conclusions from basic world state

注释：

该类包含所有场上可用的关于球员、球、标志、线的信息。
其所包含的属性可以分为 4 类：

1. Environmental information 环境信息
   包含：服务器参数，异构球员取值范围，每一种异构球员的参数
2. Match information 比赛信息
   包含：Time, PlayerNumber, Side, TeamName, PlayMode, GoalDiff（比分差）
3. Object information 对象信息
   包含各类对象且有继承关系
4. Action information 动作信息
   包含：每种动作执行的次数，动作队列，已经执行的动作

其所包含的方法可分为四类：

- Retrieval methods　　从 agent 的世界模型中重新得到对象的信息
- Update methods　　从来自 server 的感觉信息更新 agent 的世界模型
- Prediction methods　　根据先前的感觉信息预测世界的将来的状态
- High-Level methods　　从基本的世界模型信息得出高层的结论

共有方法：

WorldModel (ServerSettings *ss, PlayerSettings *ps, Formations *fs)
构造函数，所有变量设置为缺省值。

void　show (ostream &os=cout)
向指定的输出流打印 agent 掌握的所有对象和信息。只有最近“看”到的对象的信息才被打印。
void　show (ObjectSetT set, ostream &os=cout)
向指定的输出流打印对象集合“set”中的所有对象信息。
void　showQueuedCommands (ostream &os=cout)
向指定的输出流打印 ActionHandler 向 server 发送的命令队列。
void　show (ObjectT o, ostream &os=cout)
向指定的输出流打印对象 o 的信息。
bool　waitForNewInformation ()
阻塞，直到新的信息到达。这里的信息是指身体感觉信息或视觉信息。如果超过秒钟没有信息到达，认为服务器死机，返回 false。
void　logObjectInformation (int iLogLevel, ObjectT o)
记录对象的信息。

Retrieval methods

void　setTimeLastCatch (Time time)
设置上次扑球的时间。该信息是通过 SenseHandler 当裁判发送该信息时得到的。
int　getTimeSinceLastCatch ()
返回自从上次扑球以后经过的周期数。
bool　setTimeLastRefereeMessage (Time time)
设置上次接受裁判信息的时间。
Time　getTimeLastRefereeMessage ()
返回上次接受裁判信息的时间。
Time　getCurrentTime ()
返回当前时间。如果球员调用，返回上次感觉的时间；如果是教练调用，返回上次 see_globle message 时间。
int　getCurrentCycle ()
返回当前周期数。如果球员调用，返回上次感觉的周期；如果是教练调用，返回上次 see_globle message 周期。
注：周期数即为 Time 类的 t 属性。
bool　isTimeStopped ()
时间是否停止。
bool　isLastMessageSee () const
上次感觉信息是否是 see 信息。通过比较上次 see 信息时间和感觉时间是否相等。
Time　getTimeLastSeeGlobalMessage () const
返回上次 see_global 时间，只有教练使用。
bool　setTimeLastSeeGlobalMessage (Time time)
设置上次 see_global 时间，只有教练使用。
Time　getTimeLastSeeMessage () const
返回上次 see 信息的时间。
bool　setTimeLastSeeMessage (Time time)
设置上次 see 信息的时间。

Time　getTimeLastSenseMessage () const
返回上次感觉时间。
bool　setTimeLastSenseMessage (Time time)
设置上次感觉时间。同时发送一个状态信号表示 bNewInfo 被改变了。当主线程在调用 waitForNewInformation ()阻塞时，该信号将其解锁。
int　getPlayerNumber () const
返回当前 agent 的号码，该号码是一个确定的号码由服务器在初始化时给出。
bool　setPlayerNumber (int i)
设置当前 agent 的号码，该号码由初始化时服务器给出的确认信息指定。
SideT　getSide () const
返回当前 agent 的边。
bool　setSide (SideT s)
设置当前 agent 的边。该信息由初始化时服务器给出的确认信息指定。
const char *　getTeamName () const
返回当前 agent 所属队队名。
bool　setTeamName (char *str)
设置当前 agent 所属队队名。
PlayModeT　getPlayMode () const
返回当前比赛模式，该比赛模式由裁判发出。
bool　setPlayMode (PlayModeT pm)
设置比赛模式。
int　getGoalDiff () const
返回比分差。我方领先时比分差为正。
int　addOneToGoalDiff ()
比分差加一。
int　subtractOneFromGoalDiff ()
比分差减一。
int　getNrOfCommands (CommandT c) const
返回当前 agent 执行命令 c 的次数。这由 sense_body 消息给出。
bool　setNrOfCommands (CommandT c, int i)
设置当前 agent 执行命令 c 的次数。这由 sense_body 消息给出。可以用来检查一个动作是否已被服务器执行。因为相应的计数器应该比前一个 sense_body 消息多一。如果是这样，则相应索引的命令执行队列被置 true。
Time　getTimeCheckBall () const
返回上次确定球的状态的时间。只能用于教练。
bool　setTimeCheckBall (Time time)
设置上次确定球的状态的时间。只能用于教练。
BallStatusT　getCheckBallStatus () const
返回球的状态。该值由 check_ball 命令返回，且只能用于教练。对应于服务器时间的球的状态由 getTimeCheckBall 给出。
bool　setCheckBallStatus (BallStatusT bs)
设置球的状态，对应于服务器时间的球的状态由 getTimeCheckBall 返回。
ObjectT　iterateObjectStart (int &iIndex, ObjectSetT g, double dConf=-1.0)
返回对象集合 g 中第一个可信度高于 dConf 的对象。如果 dConf 取缺省值，则使用

ServerSettings 中定义的阈值。

ObjectT　iterateObjectNext (int &iIndex, ObjectSetT g, double dConf=-1.0)

返回对象集合 g 中相对于 iIndex 的下一个可信度高于 dConf 的对象。如果 dConf 取缺省值，则使用 ServerSettings 中定义的阈值。

注：使用

```
   for( ObjectT o = iterateObjectStart( iIndex,
OBJECT_SET_OPPONENTS );
        o != OBJECT_ILLEGAL;
        o = iterateObjectNext ( iIndex, OBJECT_SET_OPPONENTS ) )
```

可以对整个对方球员进行遍历。

void　iterateObjectDone (int &iIndex)

iIndex 置-1。

ObjectT　getAgentObjectType () const

返回当前 agent 的类型。位于 OBJECT_TEAMMATE_1 和 OBJECT_TEAMMATE_11 之间。

bool　setAgentObjectType (ObjectT o)

设置当前 agent 的类型为 o

AngDeg　getAgentBodyAngleRelToNeck () const

返回当前 agent 相对于脖子的身体角度。

AngDeg　getAgentGlobalNeckAngle () const

返回当前 agent 脖子的绝对角度。

AngDeg　getAgentGlobalBodyAngle ()

返回当前 agent 身体的绝对角度。

Stamina　getAgentStamina () const

返回当前 agent 的体力。

TiredNessT　getAgentTiredNess () const

返回当前 agent 的疲劳程度。

double　getAgentEffort () const

返回当前 agent 的效用。

VecPosition　getAgentGlobalVelocity () const

返回当前 agent 的绝对速度。

double　getAgentSpeed () const

返回当前 agent 的速率。

VecPosition　getAgentGlobalPosition () const

返回当前 agent 的绝对位置。

ViewAngleT　getAgentViewAngle () const

返回当前 agent 的视角。

ViewQualityT　getAgentViewQuality () const

返回当前 agent 的视觉质量。

double　getAgentViewFrequency　(ViewAngleT　va=VA_ILLEGAL,　ViewQualityT vq=VQ_ILLEGAL)

返回当前 agent 的视觉频率。

VecPosition　getBallPos ()

返回球的绝对位置。同时检查球的位置的可信度。

double　getBallSpeed ()

返回球的速率的估计值。
AngDeg　getBallDirection ()
返回球速度的方向。
Time　getTimeGlobalPosition (ObjectT o)
返回对象 o 绝对位置时间
VecPosition　getGlobalPosition (ObjectT o)
返回对象 o 的绝对位置。
Time　getTimeGlobalVelocity (ObjectT o)
返回对象 o 的速度时间。
VecPosition　getGlobalVelocity (ObjectT o)
返回对象 o 的速度。
double　getRelativeDistance (ObjectT o)
返回当前 agent 与对象 o 的相对距离。该方法并不对信息的时效性作检查。
VecPosition　getRelativePosition (ObjectT o)
返回对象 o 相对于当前 agent 的相对位置。该方法并不对信息的时效性作检查。
AngDeg　getRelativeAngle (ObjectT o, bool bWithBody=false)
返回对象 o 相对于当前 agent 的角度。缺省状态是相对于脖子的，如果第二个参数为 true，则是相对于身体的。该方法并不对信息的时效性作检查。
Time　getTimeGlobalAngles (ObjectT o)
返回对象 o 的绝对角度的时间。包括相对脖子和身体的。
AngDeg　getGlobalBodyAngle (ObjectT o)
返回对象 o 的绝对身体角度。该方法并不对信息的时效性作检查。
AngDeg　getGlobalNeckAngle (ObjectT o)
返回对象 o 的绝对脖子角度。该方法并不对信息的时效性作检查。
AngDeg　getGlobalAngle (ObjectT o)
返回对象 o 的绝对角度。o 通常为线。
double　getConfidence (ObjectT o)
返回对象 o 的信息的可信度。该可信度是由当前服务器时间周期和上次看到该物体的周期算出的。
bool　isKnownPlayer (ObjectT o)
返回对象 o 是否是已知的球员。（指其号码是已知的）
ObjectT　getOppGoalieType ()
返回对方守门员的类型。
ObjectT　getOwnGoalieType ()
返回我方守门员的类型。
Time　getTimeLastSeen (ObjectT o)
返回上次看到对象 o 的时间。
Time　getTimeChangeInformation (ObjectT o)
返回上次相对距离改变时的服务器周期。
bool　setIsKnownPlayer (ObjectT o, bool isKnownPlayer)
设置制定目标是否是已知球员。已知球员是指他的归属和号码都是已知的。
bool　setTimeLastSeen (ObjectT o, Time time)
设置上次看到对象 o 的时间。
VecPosition　getPosOpponentGoal ()

返回对方球门的位置坐标。
VecPosition　getPosOwnGoal ()
返回我方球门的位置坐标。
double　getRelDistanceOpponentGoal ()
返回当前 agent 与对方球门的相对距离。
AngDeg　getRelAngleOpponentGoal ()
返回当前 agent 所在位置与对方球门的角度。（并不考虑 agent 的脖子或身体方向）
HeteroPlayerSettings　getInfoHeteroPlayer (int iIndex)
返回索引号为 iIndex 的以后球员设置。
bool　isQueuedActionPerformed ()
对于所有可能的命令，检查是否在上个周期发出但没有执行。
bool　isFreeKickUs (PlayModeT pm=PM_ILLEGAL)
检查比赛模式 pm 是否是我方的任意球模式，如果缺省参数，则当前比赛模式被检查。
bool　isFreeKickThem (PlayModeT pm=PM_ILLEGAL)
检查比赛模式 pm 是否是对方的任意球模式，如果缺省参数，则当前比赛模式被检查。
bool　isCornerKickUs (PlayModeT pm=PM_ILLEGAL)
bool　isCornerKickThem (PlayModeT pm=PM_ILLEGAL)
bool　isOffsideUs (PlayModeT pm=PM_ILLEGAL)
bool　isOffsideThem (PlayModeT pm=PM_ILLEGAL)
bool　isKickInUs (PlayModeT pm=PM_ILLEGAL)
bool　isKickInThem (PlayModeT pm=PM_ILLEGAL)
bool　isFreeKickFaultUs (PlayModeT pm=PM_ILLEGAL)
检查比赛模式 pm 是否是我方的任意球开球失败（发球者二次触球），如果缺省参数，则当前比赛模式被检查。
bool　isFreeKickFaultThem (PlayModeT pm=PM_ILLEGAL)
bool　isKickOffUs (PlayModeT pm=PM_ILLEGAL)
bool　isKickOffThem (PlayModeT pm=PM_ILLEGAL)
bool　isBackPassUs (PlayModeT pm=PM_ILLEGAL)
检查比赛模式 pm 是否是我方队员向守门员回传，如果缺省参数，则当前比赛模式被检查。
bool　isBackPassThem (PlayModeT pm=PM_ILLEGAL)
bool　isGoalKickUs (PlayModeT pm=PM_ILLEGAL)
bool　isGoalKickThem (PlayModeT pm=PM_ILLEGAL)
bool　isBeforeKickOff (PlayModeT pm=PM_ILLEGAL)
bool　isDeadBallUs (PlayModeT pm=PM_ILLEGAL)
bool　isDeadBallThem (PlayModeT pm=PM_ILLEGAL)

Update methods

void　processSeeGlobalInfo (ObjectT o, Time time, VecPosition pos, VecPosition vel, AngDeg angBody, AngDeg angNeck)
当教练从 see_global 消息中得到关于场上对象的新的视觉信息时，该方法被调用。该方法更新第一个参数指出的对象的信息。
bool　processNewAgentInfo (ViewQualityT vq, ViewAngleT va, double dStamina, double dEffort, double dSpeed, AngDeg angSpeed, AngDeg angHeadAngle)

当接收到关于当前 agent 的新的视觉信息的时候，该方法被调用，以更新世界模型中存储的 AgentObject 类。

void　processNewObjectInfo (ObjectT o, Time time, double dDist, int iDir, double dDistChange, double dDirChange, AngDeg angRelBodyAng, AngDeg angRelNeckAng, bool isGoalie, ObjectT objMin, ObjectT objMax)

当接收到关于当前对象 o 的新的视觉信息的时候，该方法被调用，以更新世界模型中存储的有第一个参数指定的 Object 类。如果某些参数并没有从视觉信息中获得，将传入'UnknownDoubleValue'值。注意，对象只有当相应得信息作为参数传入的时候才更新。为了确保关于该对象的所有全局信息的同步，updateAll()方法将在所有的对象的信息更新完了后被调用。

bool　processPerfectHearInfo (ObjectT o, VecPosition pos, double dConf, bool bIsGoalie=0)

当听到关于对象 o 的新的信息时，该方法别调用。但是只有当说话的队员对对象的类型确定的时候，只有当新的信息的可信度比原来的高的时候，才进行更新。

bool　processPerfectHearInfoBall (VecPosition pos, VecPosition vel, double dConf)

当听到关于球的新的信息时，该方法别调用。但是只有当新的信息的可信度比原来的高的时候，才进行更新。

bool　processUnsureHearInfo (ObjectT o, VecPosition pos, double dConf)

当听到关于 o 的信息，但是说话的队员对于该对象的类型不确定的时候调用该方法。它将得到的信息与世界模型中已知的对象进行匹配。如果找不到这样的队员，则将该队员的信息加到第一个没有掌握信息的队员上。只有当新的信息的可信度比原来的高的时候，才进行更新。

bool　processNewHeteroPlayer (int iIndex, double dPlayerSpeedMax, double dStaminaIncMax, double dPlayerDecay, double dInertiaMoment, double dDashPowerRate, double dPlayerSize, double dKickableMargin, double dKickRand, double dExtraStamina, double dEffortMax, double dEffortMin)

将异构球员数组中填入参数中给出的信息。该信息直接从 server 转来的 player_type 消息中解析出。该方法只被 SenseHandler 调用。该数组中的信息，以后将被用来决定场上特定位置的异构球员。并且当 agent 的球员类型改变时对 ServerSettings 中的参数进行更新。

void　processCatchedBall (RefereeMessageT rm, Time time)

该方法设置守门员的扑球时间，该时间有裁判给出。该方法检查是否是我方扑球，如果是的话守门员在一定的周期内不能再次扑球。

void　processQueuedCommands (SoccerCommand commands[ ], int iCommands)

该方法设置 agent 执行的命令，使用这些信息，在更新的时候，未来世界模型状态可以被计算出。该方法将一个时间戳加到上次使用的命令结构中。该方法在 ActHandler 向 server 发送命令时调用。

bool　storePlayerMessage (int iPlayer, char *strMsg, int iCycle)

该方法储存世界模型中的某个其他队员传递过来的消息。由 updateAll ()方法调用。

bool　processPlayerMessage ()

该方法处理队友传来的消息。

bool　updateAll ()

该方法更新整个世界模型。它确定上一个消息，并根据该消息更新世界模型。当接收到视觉信息所有对象将根据该信息更新。

void　mapUnknownPlayers (Time time)

匹配未知球员。

bool　updateSSToHeteroPlayerType (int i)

该方法使用索引为 I 的异构球员信息更新当前 agent 的 ServerSettings。该方法一般在 coach 改变该 agent 的类型时被调用。它更新参数，这样决定下一个动作的计算将基于正确的参数。

bool　resetTimeObjects ()

该方法重置世界模型中的所有对象的信息。这意味着上次视觉信息的接收时间被设为 UnknownTime。

void　removeGhosts ()

将 ghost 从世界模型中移走。Ghost 是指本应该在上次视觉信息中被看到，但却没有被看到。通常的只有球从世界模型中移走。

Prediction methods

bool　predictStateAfterCommand (SoccerCommand com, VecPosition *pos, VecPosition *vel, AngDeg *angGlobalBody, AngDeg *angGlobalNeck, Stamina *sta=NULL)

该方法预测队员在给定的状态下执行完命令 com 后的新的状态。使用参数作为输入输出变量。

bool　predictAgentStateAfterCommand (SoccerCommand com, VecPosition *pos, VecPosition *vel, AngDeg *angBody, AngDeg *angNeck, Stamina *sta)

预测当前 agent 在执行完命令后的新的状态。使用参数作为输入输出变量。

VecPosition　predictAgentPosAfterCommand (SoccerCommand com)

预测当前 agent 在执行完命令后的新的位置。

void　predictStateAfterDash (double dActualPower, VecPosition *pos, VecPosition *vel, Stamina *sta, double dDirection)

预测球员在执行完 dash 后的新的状态。使用参数作为输入输出变量。

void　predictStateAfterTurn (AngDeg dSendAngle, VecPosition *pos, VecPosition *vel, AngDeg *angBody, AngDeg *angNeck, Stamina *sta=NULL)

预测球员在执行完 turn 后的新的状态。使用参数作为输入输出变量。

VecPosition　predictPosAfterNrCycles (ObjectT o, int iCycles, int iDashPower=100, VecPosition *vel=NULL)

该方法预测对象 o 经过 iCycle 周期后的位置。如果对象为球，则只考虑球速的衰减。如果对象是球员，则假设球员每周期使用 iDashPower 进行 dash。

VecPosition　predictAgentPos (int iCycles, int iDashPower=0)

预测当前 agent 以 iDashPower 进行 dash，经过 iCycle 后的位置。

int　predictNrCyclesToPoint (ObjectT o, VecPosition posTo, AngDeg ang)

预测对象从当前位置移动到 posTo 所需的周期数。

注：参数 ang 在 WorldModelPredict.C 代码中是 angToTurn。从字面的意思理解应为 agent 转动的角度，但是在该方法的内部实现中并没有使用这个参数，而是计算当前身体角度与目标位置到当前位置的夹角之差而得到的。

int　predictNrCyclesToObject (ObjectT objFrom, ObjectT objTo)

如果对象 objTo 按照“如果对象为球，则只考虑球速的衰减。如果对象是球员，则假设球员每周期使用 iDashPower 进行 dash”的规则运动，经过多少周期可以赶上 objFrom。

void　predictStaminaAfterDash (double dPower, Stamina *sta)

使用 manual 中指出的方法计算经过以 dPower 进行 dash 后的体力。sta 为输入输出变量。该方法并不重要，因为体力值可以从每周期可以从 sense_body 中读出。

bool　isCollisionAfterDash (SoccerCommand soc)

该方法预测经过 dash 后会不会与其他球员位置冲突。

High-Level methods

int getNrInSetInRectangle (ObjectSetT objectSet, Rectangle *rect=NULL)
该方法返回，在矩形 rect 中，包含在对象集 objectSet 中的可视对象个数。取缺省参数时整个比赛场地被考虑。同时，只有当对象的可信度高与在 PlayerSettings 中定义的阈值才被考虑。

int getNrInSetInCircle (ObjectSetT objectSet, Circle c)
该方法返回，在圆 c 中，包含在对象集 objectSet 中的可视对象个数。只有当对象的可信度高与在 PlayerSettings 中定义的阈值才被考虑。

int getNrInSetInCone (ObjectSetT objectSet, double dWidth, VecPosition start, VecPosition end)
该方法返回，在指定的圆锥中，包含在对象集 objectSet 中的可视对象个数。只有当对象的可信度高与在 PlayerSettings 中定义的阈值才被考虑。

ObjectT getClosestInSetTo (ObjectSetT objectSet, ObjectT o, double *dDist=NULL, double dConfThr=-1.0)
返回在对象集合 objectSet 中距离对象 o 最近的对象。只有当对象的可信度高与给定的阈值才被考虑，如果没有给出阈值则使用在 PlayerSettings 中定义的阈值。同时 dDist 返回距离。

ObjectT getClosestInSetTo (ObjectSetT objectSet, VecPosition pos, double *dDist=NULL, double dConfThr=-1.0)
返回在对象集合 objectSet 中距离位置 pos 最近的对象。只有当对象的可信度高与给定的阈值才被考虑，如果没有给出阈值则使用在 PlayerSettings 中定义的阈值。同时 dDist 返回距离。

ObjectT getClosestInSetTo (ObjectSetT objectSet, Line l, VecPosition pos1, VecPosition pos2, double *dDistToLine=NULL, double *dDistPos1ToP=NULL)
返回在对象集合 objectSet 中距离直线 l 最近的对象。只有当对象的可信度高与给定的阈值才被考虑，如果没有给出阈值则使用在 PlayerSettings 中定义的阈值。同时 dDist 返回距离。

ObjectT getClosestRelativeInSet (ObjectSetT set, double *dDist=NULL)
返回在对象集合 objectSet 中距离当前 agent 最近的对象。同时 dDist 返回距离。

ObjectT getSecondClosestInSetTo (ObjectSetT objectSet, ObjectT o, double *dDist=NULL, double dConfThr=-1.0)
返回在对象集合 objectSet 中距离对象 o 次近的对象。只有当对象的可信度高与给定的阈值才被考虑，如果没有给出阈值则使用在 PlayerSettings 中定义的阈值。同时 dDist 返回距离。

ObjectT getSecondClosestRelativeInSet (ObjectSetT set, double *dDist=NULL)
返回在对象集合 objectSet 中距离当前 agent 次近的对象。并且只有在上一个视觉信息包含的对象才被考虑。

ObjectT getFastestInSetTo (ObjectSetT objectSet, ObjectT o, int *iCycles=NULL)
返回在对象集合 objectSet 中相对对象 o 最快的对象。同时 iCycles 返回如果要截住该对象需要的周期数。只有当对象的可信度高与在 PlayerSettings 中定义的阈值才被考虑。

ObjectT getFastestInSetTo (ObjectSetT objectSet, VecPosition pos, VecPosition vel, double dDecay, int *iCycles=NULL)
返回相对于在 pos 位置，速度为 vel，速度衰减为 dDecay 的对象在集合 objectSet 中最快的对象。同时 iCycles 返回如果要截住该对象需要的周期数。

ObjectT getFurthestInSetTo (ObjectSetT objectSet, ObjectT o, double *dDist=NULL, double dConfThr=-1.0)
返回在对象集合 objectSet 中距离对象 o 最远的对象。只有当对象的可信度高与给定的阈值

才被考虑，如果没有给出阈值则使用在 PlayerSettings 中定义的阈值。同时 dDist 返回距离。

ObjectT　getFurthestRelativeInSet (ObjectSetT set, double *dDist=NULL)

返回在对象集合 objectSet 中距离当前 agent 最近的对象。同时 dDist 返回距离。

double　getMaxTraveledDistance (ObjectT o)

返回自从上次看到对象 o 值后，该对象能够移动的最大距离。

ObjectT　getFirstEmptySpotInSet (ObjectSetT objectSet, int iUnknownPlayer=-1)

返回对象集合 objectSet 中第一个空位。即返回对象集合中第一个可信度低于 PlayerSettings 定义的 player_conf_thr 的对象。该方法常被用于感觉到一个位置的对象，将其存储在这样的第一个位置上。如果给出了 iUnknownPlayer，则对应与这个队员的范围将用于决定位置。

bool　isVisible (ObjectT o)

判断对象 o 是否可见。

bool　isBallKickable ()

判断球是否可踢。即球是否在 agent 的踢球范围内。该值将由于异构球员的不同类型而有所不同。

bool　isBallCatchable ()

判断球是否可扑。只用于守门员。

bool　isBallHeadingToGoal ()

判断球是否正向我方球门运动。

bool　isBallInOurPossesion ()

判断是否是我方控球。通过检查靠近球的是否是我方球员。

bool　isBallInOwnPenaltyArea ()

判断球是否在我方禁区。

bool　isInOwnPenaltyArea (VecPosition pos)

判断位置 pos 是否在我方禁区。

bool　isInTheirPenaltyArea (VecPosition pos)

判断位置 pos 是否在对方禁区。

bool　isConfidenceGood (ObjectT)

判断对象的可信度是否足够高。

bool　isConfidenceVeryGood (ObjectT)

判断对象的可信度是否非常高。

bool　isOnside (ObjectT)

判断对象是否未越位。

bool　isOpponentAtAngle (AngDeg ang, double dDist)

是否在角度 ang±60,距离为 dDsit 或角度 ang±120 距离为 dDsit / 2 是否有对方球员。

Time　getTimeFromConfidence (double dConf)

该方法使用当前时间和给定的可信度，利用可信度按周期衰减的公式，反向算出当可信度为 1 的时间周期。

double　getOffsideX (bool bIncludeComm=true)

该方法使用世界模型中的信息计算当前的越位位置。

VecPosition　getOuterPositionInField (VecPosition pos, AngDeg ang, double dDist=3.0, bool bWithPenalty=true)

返回外部点。外部点定义为从位置 pos 向角度 ang 作射线，该射线与球场上的线的交点。为了不至于出界，定义该点为距离交点 dDist 的靠近 pos 一方的电。bWithPenalty 表示是否考虑禁区线。

AngDeg　getDirectionOfWidestAngle (VecPosition posOrg, AngDeg angMin, AngDeg angMax, AngDeg *ang, double dDist)
找到以 posOrg 以基点，在角度[angMin..angMax]之间的，对方球员的最大空档的中线。只有距离小于 dDist 的对方球员才被考虑。该角度由 ang 返回。
注：该方法可用于守门员开球。
VecPosition　getStrategicPosition (int iPlayer=-1)
返回特定队员的战略位置。使用 Formation 类完成。
double　getActualKickPowerRate ()
对于球相对于当前 agent 的位置，计算实际踢球力量的比率。
double　getKickPowerForSpeed (double dDesiredSpeed)
计算为了将球以 dDesiredSpeed 速度踢出，所需的力量。
double　getKickSpeedToTravel (double dDistance, double dEndSpeed)
计算为了将球踢出 dDistance 并且球的最终速度为 dEndSpeed 时，所需的力量。
double　getFirstSpeedFromEndSpeed (double dEndSpeed, double dCycles)
由球的最终速度和经过的周期数来计算球的初速度。
double　getFirstSpeedFromDist (double dDist, double dCycles)
由球踢出的距离和经过的周期数计算球的初速度。
double　getEndSpeedFromFirstSpeed (double dFirstSpeed, double dCycles)
由球的初速度和经过的周期数求其末速度。
AngDeg　getAngleForTurn (AngDeg angDesiredAngle, double dSpeed)
考虑球员速度和惯性，计算如果要实际旋转 angDesiredAngle 需要向服务器发出的旋转命令的角度。
AngDeg　getActualTurnAngle (AngDeg angTurn, double dSpeed)
考虑球员速度和惯性，计算如果向服务器发出的旋转角度 angTurn 而实际旋转的角度。
double　getPowerForDash (VecPosition posRelTo, AngDeg angBody, VecPosition vel, double dEffort)
该方法使用 BackDash 来决定在速度太高，距离太小的的情况下优化 DashPower。否则该速度与最大速度的差值将被计算，同时设置 Dash Power Rate 来补偿这个差值。

共有属性：

int　iNrHoles
int　iNrOpponentsSeen
看到的对方球员个数。
int　iNrTeammatesSeen
看到的队友的个数。
char　strLastSeeMessage [MAX_MSG]
上次视觉信息。
char　strLastSenseMessage [MAX_MSG]
上次 sense_body 信息。
char　strLastHearMessage [MAX_MSG]
上次听觉信息。

私有方法：

Object *　getObjectPtrFromType (ObjectT o)
该方法返回一个指向对象类型 o 信息的指针。
void　processLastSeeMessage ()
该方法处理最后接收到的视觉信息，并根据相应的信息更新 agent 世界模型。
bool　updateAfterSeeMessage ()
该方法在一个视觉信息之后更新世界模型。对于当前 agent 和所有的队友、对方球员以及球的信息被更新。
bool　updateAgentObjectAfterSee ()
该方法在一个视觉信息之后更新 agent 的绝对位置和脖子角度。
bool　updateDynamicObjectAfterSee (ObjectT o)
该方法在一个视觉信息之后更新对象 o 的绝对位置和速度。
bool　updateAfterSenseMessage ()
该方法在一个感觉信息之后更新世界模型。对于当前 agent 和所有的队友、对方球员以及球的信息被更新。
bool　updateAgentAndBallAfterSense ()
该方法在一个感觉信息之后更新 agent 和球的信息。
bool　updateBallAfterKick (double dPower, AngDeg ang)
在踢球命令之后更新球的相关信息。
bool　updateDynamicObjectForNextCycle (ObjectT o, int iCycle)
该方法更新动态对象 iCycle 周期后的信息。
bool　updateBallForCollision (VecPosition posAgent)
当球与球员重叠的时候，更新球的信息。
bool　updateRelativeFromGlobal ()
由当前 agent 和所有球员的全局信息更新其相对信息。
bool　updateObjectRelativeFromGlobal (ObjectT o)
由当前 agent 的信息，以及对象 o 的全局信息更新 o 的相对信息。
bool　calculateStateAgent (VecPosition *posGlobal, VecPosition *velGlobal, AngDeg *angGlobal)
该方法计算当前 agent 的信息，即绝对位置、绝对速度和绝对脖子角度。
void　initParticlesAgent (AngDeg angGlobal)
void　initParticlesAgent (VecPosition posInitital)
int　checkParticlesAgent (AngDeg angGlobalNeck)
void　updateParticlesAgent (VecPosition vel, bool bAfterSense)
VecPosition　averageParticles (VecPosition posArray[], int iLength)
void　resampleParticlesAgent (int iLeft)
bool　calculateStateAgent2 (VecPosition *posGlobal, VecPosition *velGlobal, AngDeg *angGlobal)
VecPosition　calculatePosAgentWith2Flags (ObjectT objFlag1, ObjectT objFlag2)
AngDeg　calculateAngleAgentWithPos (VecPosition pos)
bool　calculateStateBall (VecPosition *posGlobal, VecPosition *velGlobal)
void　initParticlesBall (VecPosition posArray[], VecPosition velArray[], int iLength)
void　checkParticlesBall (VecPosition posArray[], VecPosition velArray[], int iLength, int *iNrLeft)
void　updateParticlesBall (VecPosition posArray[], VecPosition velArray[], int iLength, double

dPower, AngDeg ang)
void resampleParticlesBall (VecPosition posArray[], VecPosition velArray[], int iLength, int iLeft)
ObjectT getMaxRangeUnknownPlayer (ObjectT obj, char *strMsg)
VecPosition calculateVelocityDynamicObject (ObjectT o)
bool calculateStateBall2 (VecPosition *posGlobal, VecPosition *velGlobal)
bool calculateStatePlayer (ObjectT o, VecPosition *posGlobal, VecPosition *velGlobal)
bool getMinMaxDistQuantizeValue (double dInput, double *dMin, double *dMax, double x1, double x2)
bool getMinMaxDirChange (double dOutput, double *dMin, double *dMax, double x1)
bool getMinMaxDistChange (double dOutput, double dDist, double *dMin, double *dMax, double x1, double xDist1, double xDist2)
double invQuantizeMin (double dOutput, double dQuantizeStep)
double invQuantizeMax (double dOutput, double dQuantizeStep)

私有属性：

ServerSettings * SS　　所有服务器参数
PlayerSettings * PS　　所有球员参数
HeteroPlayerSettings pt [MAX_HETERO_PLAYERS]　　异构球员类型数组
Formations * formations　　阵型
Time timeLastSeeMessage　　上次接收到视觉信息的时间
Time timeLastSenseMessage　　上次接收到感觉信息的时间
bool bNewInfo　　是否有新的信息
Time timeLastCatch　　上次扑球时间
Time timeLastRefMessage　　上次裁判信息时间
char strTeamName [MAX_TEAM_NAME_LENGTH]　　我方队名
int iPlayerNumber　　服务器中的球员号码
SideT sideSide　　球员所属哪方
PlayModeT playMode　　当前比赛模式
int iGoalDiff　　比分差
BallObject Ball　　球的信息
AgentObject agentObject　　当前 agent 对象信息
PlayerObject Teammates [MAX_TEAMMATES]　　队友数组
PlayerObject Opponents [MAX_OPPONENTS]　　队手数组
PlayerObject UnknownPlayers [MAX_TEAMMATES+MAX_OPPONENTS]　未知球员数组
int iNrUnknownPlayers　　未知球员个数
FixedObject Flags [MAX_FLAGS]　　标志数组
FixedObject Lines [MAX_LINES]　　边底线数组
VecPosition particlesPosAgent [iNrParticlesAgent]　　agent 位置点
VecPosition particlesPosBall [iNrParticlesBall]　　球位置点
VecPosition particlesVelBall [iNrParticlesBall]　　球速点
SoccerCommand queuedCommands [CMD_MAX_COMMANDS]　所有执行过的命令队列
bool performedCommands [CMD_MAX_COMMANDS]　　上周期执行的命令

int　iCommandCounters [CMD_MAX_COMMANDS]	所有命令执行次数计数器
Time　timeCheckBall	教练上次 Check Ball 的时间
BallStatusT　bsCheckBall	球的状态
pthread_mutex_t　mutex_newInfo	bNewInfo 的信号量
pthread_cond_t　cond_newInfo	
char　m_strPlayerMsg [MAX_MSG]	球员通信消息
int　m_iCycleInMsg	信息中包含的周期数
Time　m_timePlayerMsg	球员通信信息的时间
int　m_iMessageSender	球员通信信息的发送者
bool　m_bWasCollision	是否发生位置冲突
bool　m_bPerformedKick	球是否被踢了

静态私有属性：

const int　iNrParticlesAgent = 100	储存 agent 位置的 particle 的个数
const int　iNrParticlesBall = 100	储存球的位置速度的个数

第五章 UVA 球队策略及实现

在了解了的 UVA 的各个类之间的关系以后，紧随其后的就是 UVA 的策略，包括其阵型、进攻、防守的策略。在已公布的源码中，UVA 是通过使用一个简单的策略（deMeer5）来实现高层决策的。在以前做球队时，一些队往往不知道如何实现自己设定的策略，从而没有很好的将自己的意图转化为球队的策略。针对这一情况，本章对 deMeer5 的策略做详细的介绍，讲解的目的在于通过这一相对简单的实例分析，使读者对如何实现高层决策有更好的了解，并同过对这一实例的学习，写出自己的高层决策。下面就来详细讲解 deMeer5 的策略。

5.1 deMer5 的使用背景

5.1.1 deMeer5 的效果

UVA2000 年的版本在高层决策上使用的是已公布的 FCP 的策略，其中并不涉及复杂的学习策略，而是用简单的人工编码来实现。既使这样，其球队实力也不可低估，在与某些学习型球队对抗时，往往仍可取得较好的战绩。所以，可以用它来作为我们平时训练的对手。

5.1.2 deMeer5 的实现原理

deMeer5 作为一般球员的策略,采用的是能踢则踢，不能踢则防的简单策略。大家都学过有限自动机。在 deMeer5 中，也是利用这一原理，将球进门作为最终状态，如果球没有进门，deMeer5 会一直寻找可能实现这一最终状态的机会，并不断的尝试，直至达到终态为止。所以从这个角度上来说 deMeer5 的策略是简单的。作为球队中的特例，守门员使用的是单独的策略 deMeer5_goalie。

用一段作者对于采用 deMeer5 策略队员动作的描述作为本节的结尾：

The players do the following:

- if ball is kickable kick ball to goal (random corner of goal)
- else if i am fastest player to ball and no opponent can intercept ball intercept the ball
- else move to strategic position based on your home position and pos ball

5.2 deMeer5 的剖析

5.2.1UVAdeMeer5 源码分析：

本来是想利用一个完整的流程图来描述 UVAdeMeer5 的实现，但是还是觉得采用事件驱动的方法，描述一下 UVAdeMeer5 的效果比较好。如果大家想知道其中的细节，可以从代码中找到详细的源码实现。

首先，介绍一下本方采用的 4-3-3 阵型，基准位置在 formation.conf 中设定。比赛中，队员会不断的判断是否处于可踢的状态，如果不可踢，则就会结合自己在阵形中的位置和当前状态，来确定自己的站位。这样在大多数的情况下，可以知道己方球员的大致位置，而不

必局限于自己的视觉。当然，这种方法在某些情况下会造成体力的损失。这在和强队的比赛中是极为不利的。这就要从底层去修改。

其次，由于在比赛中会有进球，每一次开球实际上就相当于一个新的开始，即状态机又回到了初始状态。这样的好处是，球员踢得不好，不会有心理压力。但这也给我们带来了一个问题，就是当在对手领先的情况下，我们要采取一套与一般状态下不同的策略，往往偏重于防守。就需要额外的编码了。

在了解了基本的情况以后，我们来看 deMeer5（）是如何完成角色转换。关于这一问题，我们首先要明确，在比赛中，我们设有前锋、中场和后卫。这三种角色都是调用的 deMeer5() 函数，所以我们要了解，deMeer5()在接受数据时，是如何按不同的角色展开的。其实，在 deMeer5（）中，所有的场景都是有其标记方式的，并没有明确的角色分工，只是由于站位的不同接触到球的可能性不同，而导致表现的差异。大体的功能如下（按状态的先后顺序）：

1、是否处于开球前的状态。若是的话，则按开球时的位置进行站位。

2、是否能过看到球，若不可以，则寻找球的位置。进

3、在可以看到球的情况下，队员就要判断是否可以踢球。若可以，就踢，否则就要找出最近的球员去踢球。

4、对基本的决策作出优化，主要是结合体力状况，预测球的可能位置，进行截球。以下是程序中不同体力值时对带球指令的解释。

```
if( soc.commandType == CMD_DASH &&              //当体力值低时的策略。
          WM->getAgentStamina().getStamina() <
             SS->getRecoverDecThr()*SS->getStaminaMax()+200 )
      {
        // 慢速带球
        soc.dPower = 30.0 * WM->getAgentStamina().getRecovery();
       ACT->putCommandInQueue( soc );
       ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
      }
      else    // 当体力值高时。
      {
```

```
    ACT->putCommandInQueue( soc );  // 按计划带球

    ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );

  }
```

5.2.2 UVAdeMeer5 的优缺点及改进方向

deMeer5（）的优点是决策速度快，这是由其是手工编码而决定的。然而，由于其没有对以往比赛的记忆功能，所以无法总结以往的比赛经验，只能通过人来间接的总结，对编码人员的足球知识或经验是一个挑战。同时，由于其没有灵活的转换机制，所以一旦被学习型的对手掌握了它的策略，就只能处于招架之地。

对其改进方向，大致有两个，一个是在总结比赛经验的基础上，通过手工编码对其在处理某些情况下的功能有所加强，进一步细化各种角色的地位。另一种方法是在决策过程中加入学习的算法，将通过离线学习总结出的参数加入决策过程，或是对不同的对手采用不同的决策参数。

5.2.3 deMeer5_goalie()的剖析

守门员，在整个球队中占有重要的地位。对守门员策略的介绍，我们也分为三个方面：守门员可以做的动作、守门员的决策过程、守门员的决策依据和算法。

首先要谈的守门员与普通队员的不同，他们可以完成一些特殊的动作，具体如下：

- 在开球时可以在场地内部不受限的移动到任意位置
- 在一定情况下，可以直接拿到球。catchBall()

守门员的决策过程与一般球员大体相似。只是在离球较近时，会采用守门员的直接拿球的策略。其他的与一般的球员类似。

第六章 UVA Trilearn 底层结构分析

在前面的章节中，本书已对 Server 的规则和 UVA Trilearn 的类层次结构有了较详细地说明，相信大家已对 Robocup Simulator 有了一个直观地了解。

在上章中介绍了 deMeer5 的基本策略，这些都是属于逻辑层次的方法，属于球队的整体决策，是上层的结构。通过对 Server 的了解，我们知道 Server 和 Client(球队或者说每个球员)是通过 UDP/IP 协议通信的，双方通信的内容就是球场上的实时信息和球员发出的动作，这些信息是一系列的 ASCII 码字符串。那么，球员程序是如何将 Server 发出的包含实时信息的字符串序列解析，用于更新世界模型，进而做出决策后，再将下一步的动作传递给 Server 的呢？本章将进行一定的阐述。

消息的解析和动作的发送构成了球队的底层，在球队的程序中，由于消息的解析相对独立而且实时性强，所以作为单独的线程运行。UVA Trilearn 的消息解析主要是由 SenseHandler 类来进行的，而动作的发送主要是由 ActHandler 类来完成的，还有其他的一些辅助工具类。本章通过这两个类来分析底层结构，对于网络通信的细节，例如 Socket ，本书不作介绍。

6.1 实时信息的解析

我们已经介绍过，UVA Trilearn 的实时信息的解析主要是由 SenseHandler 类来完成的。本节介绍 SenseHandler 的类层次结构和消息解析及世界模型更新的过程。

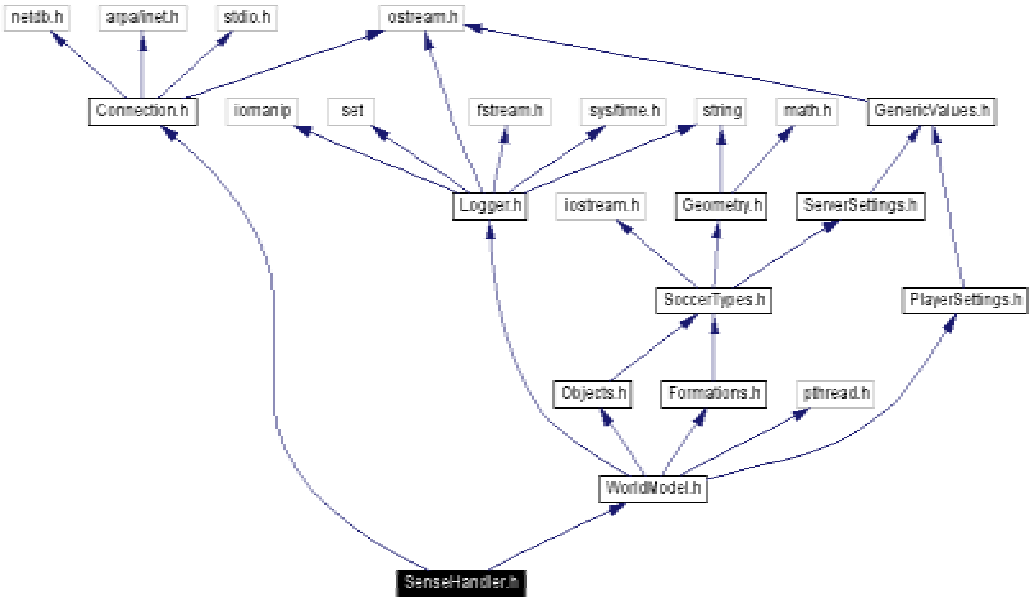


图 6.1-1 Include dependency graph for SenseHandler.h

图 6.1-1 是 SenseHandler.h 的引用关联图，从图中可以直观地看到其直接引用了 Connection.h 和 WorldModel.h，分别用于接收实时信息和更新世界模型。

6.1.1 SenseHandler 类层次结构

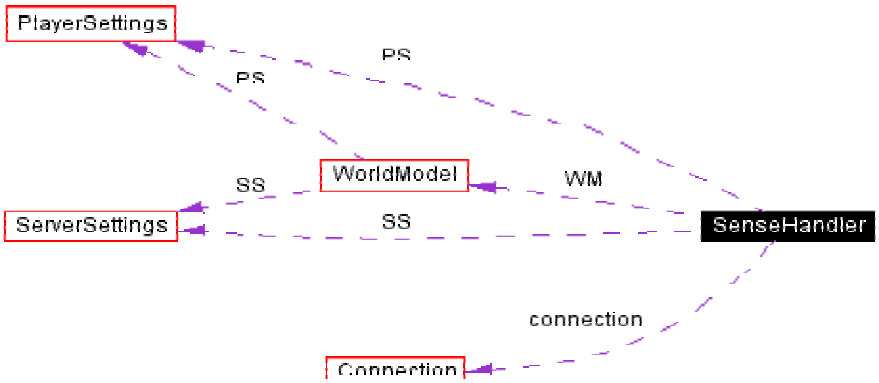


图 6.1-2 Collaboration diagram for SenseHandler

图 6.1-2 显示了 SenseHandler 与其他类的协作关系。ServerSettings 和 PlayerSettings 用于在球员上场初始化和球员异构类星替换时，更新相关 Server 和 Player 的参数。Connection 用于 Client 和 Server 之间的信息传递，WorldMoldel 中存放了与实时信息相关的世界模型，信息解析后更新 WorldModel 中的相关数据。

SenseHandler 的类成员如下：

```
analyzeChangePlayerTypeMessage(char *strMsg)
analyzeCheckBall(char *strMsg)
analyzeHearMessage(char *strMsg)
analyzeInitMessage(char *strMsg)
analyzeMessage(char *strMsg)
analyzePlayerMessage(char *strMsg)
analyzePlayerParamMessage(char *strMsg)
analyzePlayerTypeMessage(char *strMsg)
analyzeSeeGlobalMessage(char *strMsg)
analyzeSeeMessage(char *strMsg)
analyzeSenseMessage(char *strMsg)
analyzeServerParamMessage(char *strMsg)
connection                                          [private]
handleMessagesFromServer()
iSimStep                                            [private]
```

```
iTimeSignal                                                    [private]
iTriCounter                                                    [private]
itv                                                            [private]
m_iSeeCounter                                                  [private]
PS                                                             [private]
readServerParam(char *strParam, char *strMsg)
SenseHandler(Connection *c, WorldModel *wm, ServerSettings
             *ss, PlayerSettings *ps)
setTimeSignal()
SS                                                             [private]
WM                                                             [private]
```

6.1.2 信息解析流程

在本节中，主要讲述整个过程的流程，对于具体的程序细节请参考 UVA Trilearn 的源代码。

handleMessagesFromServer() 是整个信息解析的入口，它不断的监听来自 Server 的消息，并把字符串信息交给 **analyzeMessage**(char *strMsg) 进行分类解析处理。

在介绍消息的分类解析处理前，先介绍一下 Server 的消息分类。

Server 的消息总体上可分为三类：

视觉消息，如：SeeGlobalMessage、SeeMessage、SenseMessage；

听觉消息，如：RefereeMessage、CoachMessage、PlayerMessage；

控制消息，如：ChangePlayerTypeMessage、CheckBallMessage。

具体的每个消息的格式就不详细说明了，值得注意的是消息都是以‘（’开头，‘）’结尾的，例如 SeeMessage：(see 0 ((g r) 64.1 13) ((f r t) 65.4 -16) ((f r b) 79 38) ((f p r t) 46.1 -6) ((f p r c) 48.4 18) ((f p r b) 58 37) ((f g r t) 62.8 7) ((f g r b) 66 19) ((f t r 20) 38.5 -38) ((f t r 30) 46.5 -30) ((f t r 40) 55.7 -25) ((f t r 50) 64.7 -21) ((f b r 50) 80.6 41) ((f r t 30) 69.4 -12) ((f r t 20) 67.4 -4) ((f r t 10) 67.4 4) ((f r 0) 69.4 12) ((f r b 10) 72.2 20) ((f r b 20) 75.9 27) ((f r b 30) 81.5 33) ((l r) 62.8 -89))。

analyzeMessage(char *strMsg) 在进行消息解析时并没有按上述分类进行解析，而是判断消息字符串的首字符来分别进行处理，这样简化了程序的结构，提高了效率。解析处理的基本流程如图 6.1-3 所示（见下页）。

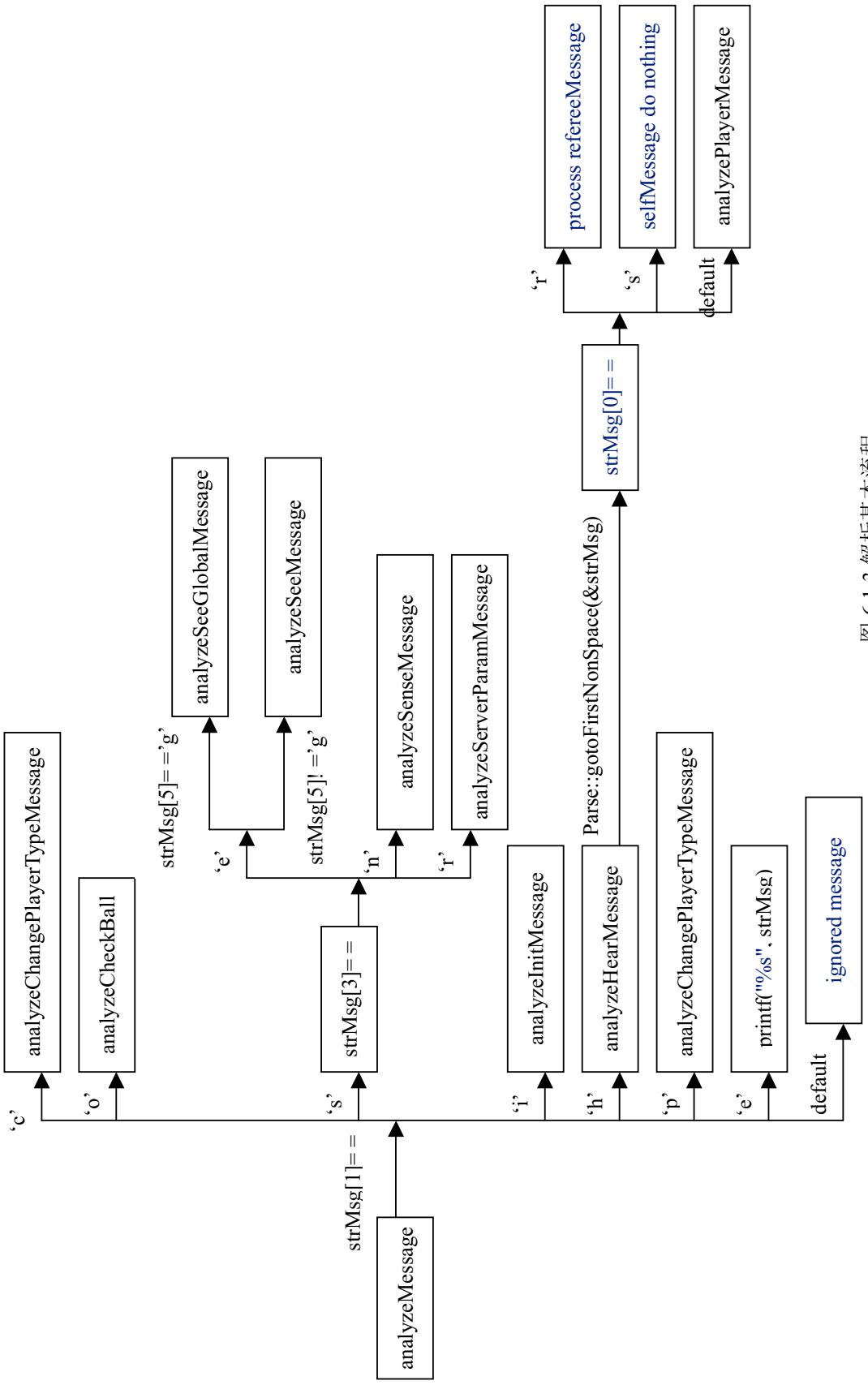

analyzeMessage
strMsg[1]= =
'c'
analyzeChangePlayerTypeMessage
'o'
analyzeCheckBall
's'
strMsg[3]= =
strMsg[5]= ='g'
'e'
analyzeSeeGlobalMessage
strMsg[5]! ='g'
analyzeSeeMessage
'n'
analyzeSenseMessage
'r'
analyzeServerParamMessage
'i'
analyzeInitMessage
'h'
analyzeHearMessage
Parse::gotoFirstNonSpace(&strMsg)
strMsg[0]= =
'r'
process refereeMessage
's'
selfMessage do nothing
default
analyzePlayerMessage
'p'
analyzeChangePlayerTypeMessage
'e'
printf("%s", strMsg)
default
ignored message

图 6.1-3 解析基本流程

6.1.3　世界模型的更新

至此，程序已经完成了实时消息的分类处理，不同的消息均已交递给其解析函数进行处理。本节例举出一种消息的解析，以及所进行的世界模型更新，其他的消息基本上都是类似模式进行的，请阅读源代码。

我们以解析视觉信息的 analyzeSeeMessage (char *strMsg)函数为例，其中也涉及到了一些 WorldModel 中的方法，看看世界模型的更新过程。

此函数的作用是：

1．从消息中提取出周期数，并更新世界模型中的周期，使之同步。

2．将视觉信息保存在世界模型中，用于世界模型的更新。

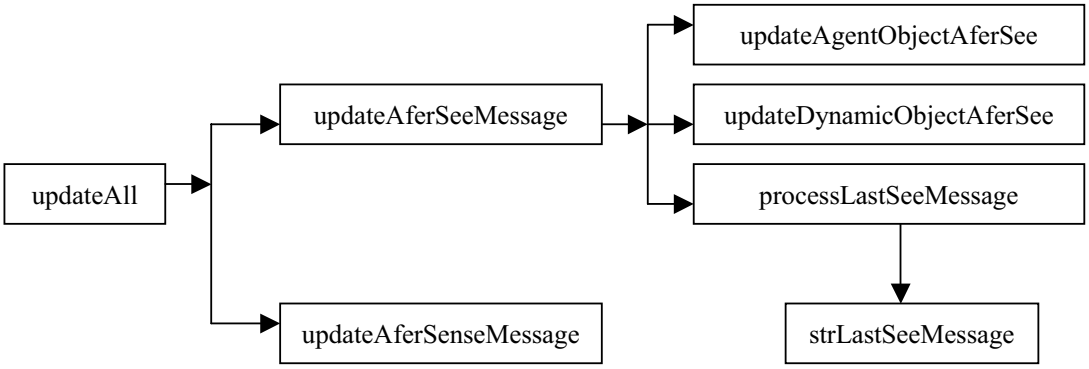
```
bool SenseHandler::analyzeSeeMessage( char *strMsg )
{
    //将视觉信息保存在世界模型中，并由它处理
    strcpy( WM->strLastSeeMessage, strMsg );

    Log.logWithTime( 2, " incoming see message" );
    Log.logWithTime( 2, " %s",strMsg );
    .
    .//周期同步处理，不做介绍
    .
    return true;
}
```

我们以 strcpy(WM->strLastSeeMessage, strMsg)为入口，追踪世界模型的更新过程。

在 WorldModel.h 中定义了 strLastSeeMessage[MAX_MSG]，用于存储最新看到的视觉信息。与之相关的方法有：

```
bool processLastSeeMessage()      [private]
bool updateAfterSeeMessage()      [private]
bool updateAgentObjectAfterSee()  [private]
bool updateDynamicObjectAferSee(ObjectT o)  [private]
bool updateAll()
```

在 player::mainLoop()中调用了 updateAll(),使其不断的更新世界模型，包括视觉信息和感知信息，视觉信息更新的流程如图 6.1-4 所示。



绝大多数消息的世界模型更新过程类似于上述过程。

在众多的消息中，我们要特别注意一下 `RefereeMessage`，比赛中比赛状态的更新是由其决定的。消息解析函数通过对该消息的解析，从而更新世界模型中的比赛状态，具体的比赛状态模式参见 `ServerManual`。

6.2 动作解释和发送

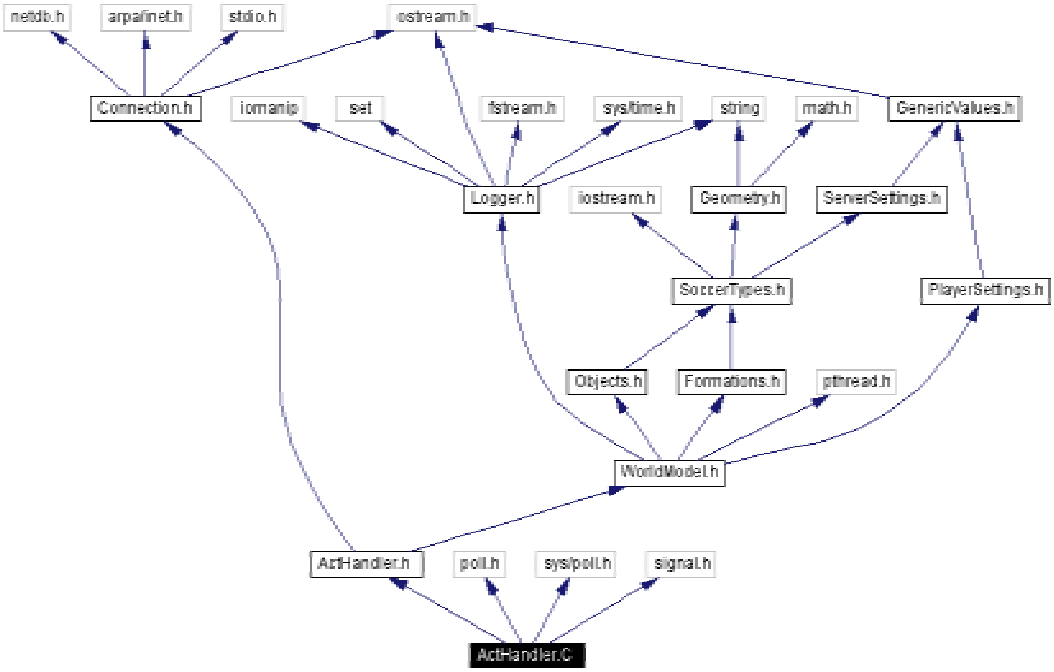


图 6.2-1 Include dependency graph for ActHandler.C

我们已经了解了由 Server 发出的消息解析以及进行的世界模型更新过程，这些都属于信息的获取。当球员获得了足够真实的信息后，高层决策根据已有的信息作出决策，驱动球员 jinxing 相应的动作。动作经过解释成为标准的 Server 可接受的消息，再发送给 Server，

直观的由 Monitor 表现出来。动作的解释主要是由 SoccerCommand 来进行的，而整体的控制是由 ActHandler 来完成的。

Server 规定了球员的一些基本动作，如 dash、kick、turn 等，诸如跑位、传球、射门此类的高级动作是由基本动作配以不同的参数来实现的。关于基本的动作类型和参数，详见 ServerManual。

ActHandler 的头文件引用关系如图 6.2-1 所示。

6.2.1 SoccerCommand 类层次结构

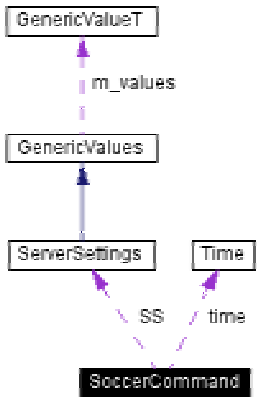


图 6.2-2 Collaboration diagram for SoccerCommand

图 6.2-2 表示了 SoccerCommand 与其它类的协作关系。正如前面所介绍的，SoccerCommand 的主要职能就是将 CommandT 类型的动作解释为 Server 所接受的字符串消息。具体的解释过程就不单独说明了，本书将其插入动作的发送流程中。

SoccerCommand 的类成员如下：

commandType	
dAngle	
dPower	
dX	
dY	
getCommandString(char *str, ServerSettings *ss)	
isIllegal()	
iTimes	
makeAttentionToCommand(char *str)	[private]
makeCatchCommand(char *str)	[private]
makeChangePlayerCommand(char *str)	[private]
makeChangeViewCommand(char *str)	[private]
makeCommand(CommandT com, double d1=UnknownDoubleValue,	

double d2=UnknownDoubleValue, double d3=UnknownDoubleValue)	
makeCommand(CommandT com, ViewAngleT v, ViewQualityT q)	
makeCommand(CommandT com, char *msg)	
makeDashCommand(char *str)	[private]
makeKickCommand(char *str)	[private]
makeMoveCommand(char *str)	[private]
makeSayCommand(char *str)	[private]
makeSenseBodyCommand(char *str)	[private]
makeTurnCommand(char *str)	[private]
makeTurnNeckCommand(char *str)	[private]
show(ostream &os)	
SoccerCommand(CommandT com=CMD_ILLEGAL, double d1=UnknownDoubleValue, double d2=UnknownDoubleValue, double d3=UnknownDoubleValue)	
SoccerCommand(CommandT com, char *msg)	
SS	[private]
str	
time	
va	
vq	

6.2.2 ActHandler 类层次结构

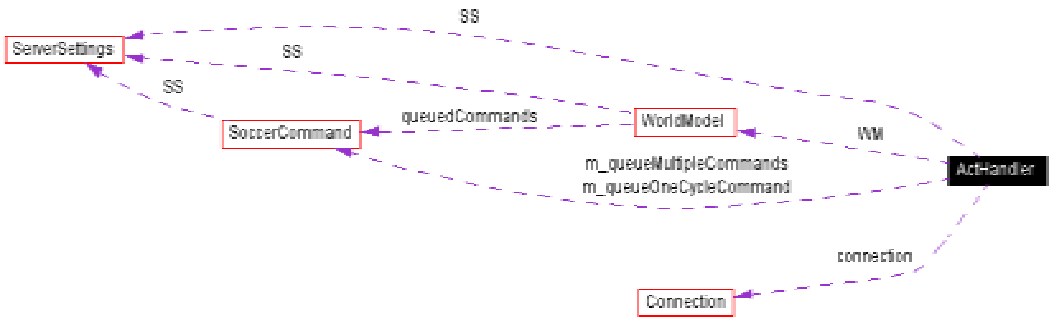


图 6.2-3 Collaboration diagram for ActHandler

图 6.2-3 表示了 ActHandler 与其它类的协作关系。ActHandler 控制着整个动作发送的流

程，值得注意的是其中的 CommandQueue，动作进入队列后，会在“合适”的时间发出，因为涉及到了复杂的同步机制，本书不会详细介绍 CommandQueue 的时间机制和发送规则。重点介绍整个动作发送的流程。

ActHandler 类成员如下：

ActHandler(Connection *c, WorldModel *wm, ServerSettings *ss)	
connection	[private]
emptyQueue()	
isQueueEmpty()	
m_iMultipleCommands	[private]
m_queueMultipleCommands	[private]
m_queueOneCycleCommand	[private]
putCommandInQueue(SoccerCommand command)	
sendCommand(SoccerCommand soc)	
sendCommandDirect(SoccerCommand soc)	
sendCommands()	
sendMessage(char *str)	
sendMessageDirect(char *str)	
SS	[private]
WM	[private]

6.2.3 动作解释发送流程

我们现在已对 SoccerCommand 和 ActHandler 的类结构和成员有了一定的了解。本节将通过实例讲述动作解释和发送的整个流程。

ActHandler 中的 **putCommandInQueue**(SoccerCommand command)是面向球员动作执行的入口，也是整个动作解释发送的入口。

我们以球员跑向定点（0,0）为例，底层动作函数为：

```
SoccerCommand BasicPlayer::dashToPoint(VecPosition pos)
```

在决策部分 Player 中的调用的方式为：

```
ACT->putCommandInQueue(dashToPoint(VecPosition(0, 0)))
```

这样这个跑位的动作进入队列，在球员的外层主循环中，每次均调用：

```
ACT->sendCommands()
```

sendCommands 函数会触发命令的解释和发送，其基本过程简化如下：

1. 遍历动作队列，解释动作。

```
for(int i = 0; i < m_iMultipleCommands; i++)
{
    m_queueMultipleCommands[i].getCommandString(
        &strCommand[strlen(strCommand)], SS);
}
```

上段代码的作用就是解释每个动作为相应的动作消息字符串，其中m_queueMultipleCommands[]存储的是soccerCommand类型的动作，即为动作队列；strCommand[]是存储解释后动作消息的字符串，所有的动作消息顺次存放。我们进入soccerCommand::getCommandString(char *str, ServerSettings *ss)内部，看看动作的解释过程：

1) 进行基本动作选择，不同的基本动作对应不同的解释函数；

```
bool b = false;
switch(commandType)
{
    case CMD_DASH: b = makeDashCommand(str); break;
    . . ..
    default: . . . .
}
if(b == false){
    commandType = CMD_ILLEGAL;
    str[0] = '\0';
}
```

dashToPoint返回的soccerCommand.commandType == CMD_DASH

2) 执行相应的解释函数makeDashCommand(str)，生成动作消息字符串。

2. 将动作序列字符串发送给Server。

```
if(strCommand[0] != '\0')
{
    timeLastSent = WM->getCurrentTime();
    connection->sendMessage(strCommand);
    Log.logWithTime( 2, " send queued action to server: %s",
                    strCommand);
}
else
{
    Log.logWithTime( 2, " no action in queue??" );
    return false;
}
```

connection->sendMessage(char*)将字符串发送给Server。

当然，实际的发送控制要比上述的复杂的多，其中包含了协调和同步机制，使得动作及

时高效的执行。

总结：

本章讲述的是 UVA Trilearn 的底层结构，也可以说是信息的收集、处理和反馈过程模型。一个球队的强弱固然与高层的决策策略有着直接的关系，但如果信息收集、处理的不好，就会产生错误的世界模型，致使根据错误的信息作出错误的决策。所以有个高效、准确的底层，是做好 Robocup 球队的起码要求。

本章旨在抛砖引玉，只是介绍了大概的框架，要想真正的了解到底层的处理，还需读者花费时间和精力详细的研读源代码。

第七章 球员个体技术的学习及实现

在 Robocup 机器人足球比赛中，球员的个人技术是很重要的。如果没有很好的个体技术，再完美的配合也形成不起来。所谓个体技术，主要是指球场上球员可以执行的一些动作，如传球、阻截球、带球、射门、盯人、守门员扑球、铲球等；当然这些动作有的是不能直接发送给 Server 的，它们是由更低级的 Server 可识别的原子动作（dash、kick、turn 、tackle 等）组成，在发送的时候是发送这些原子动作序列。这些个人技术依赖于球员所观察到的世界状态，以世界状态为基础，简化后提取世界状态的特征，根据一定的算法和数学模型做出合适的行为决定。

提高球员的个体技术是我们的目标，一般是通过机器学习（Machine Learning，简称 ML）、合适的数学模型进行解析和经验式来达到这一目标的。

其中数学解析的方法主要是建立各个动作的数学模型，然后用解析几何的方法进行求解；这种方法是建立在对问题的深入分析的基础上面的，因此它的效率较高，当由于在比赛当中存在噪音精确性有时达不到理想的效果，同时也不是所有问题都可以转化成合适的、便于求解的数学模型。而经验式的方法纯粹是根据设计者的经验，通过类似 if….then…的结构来设计行为模式；这种方式的优点是可以充分利用人的经验，但是他参数的调节比较繁琐，并且鲁棒性也不是很好。而机器学习的方法优点则比较明显，下面我们就介绍一下机器学习的方法和机器人足球中是如何应用机器学习的。

7.1 机器学习简介

“机器学习”一般被定义为一个系统自我改进的过程。但仅仅从这个定义来理解和实现机器学习是困难的。从最初的基于神经元模型以及函数逼近论的方法研究，到以符号演算为基础的规则学习和决策树学习的产生，之后到认知心理学中归纳、解释、类比等概念的引入，乃至最新的计算学习理论和统计方法学习（主要是指贝叶斯学习和基于马尔可夫过程的强化学习）的兴起，机器学习一直在包括人工智能学科在内的相关学科的实践应用中起着主导地位。然而，根据学习的条件和领域的不同，具体的学习理论和算法也各不相同。本节列举了常见的机器学习理论和相关的学习算法。如：概念学习、决策树、神经网络、贝叶斯学习、基于实例的学习、遗传算法、规则学习、分析学习(基于解释的学习)和强化学习等。

(1)概念学习

所谓概念学习就是指通过给定某一类别的若干正例和反例，从中得出该类别一般定义的学习方法。它是一个从许多特例归纳而形成表示一般函数的方法。所以说，概念学习可以看成是搜索预定义潜在的假设空间过程。是归纳法的一种。它的主要设计过程是从一般到特殊序然后形成假设空间的过程。这个概念最初是由 Bruner et al.在 1957 年就提出了，在 1970 年 Winston 的博士论文[33]中将概念学习看成是包含泛化和特化操作的搜索过程。Simon 和 Lea[34]在 1973 年将该学习的过程看成是一个在假设空间搜索的过程。

(2)决策树学习

决策树学习是应用最广的归纳推理算法之一。决策树是定义布尔函数的一种方法，其输入为由一组属性描述的对象，输出为 yes/no 决策。树的每一个内部节点（包括根节点）是对输入的某个属性的测试，此节点下面的各个分支被标记该属性性质的各个值。每一个叶节

点指示：达到该节点时布尔函数应返回的 yes/no 值。概括的说它是一种逼近离散值函数的方法，一般该函数被表示成一颗树，树一般包含多个 if-then 规则。这种学习方法对噪音数据有很好的健壮性。

决策树通过把实例从根节点排列（sort）到某个叶子节点来分类实例。叶子节点即为所属的分类。树上的每个节点说明了对实例的某个属性的测试，并且该节点的每个后继分支对应于该属性的一个可能值。分类实例的方法是从这棵树的根节点开始，测试这个节点指定的属性，然后按照给定实例的该属性值对应的树枝向下移动，一直遍历到叶子。

决策树学习可以解决具有以下特征的问题：

①实例是由“属性-值”对表示的；

②.目标函数具有离散的输出值；

③.可能需要析取的描述；

④.训练数据可以包含错误；

⑤.训练数据可以包含缺少属性值的实例。

决策树学习的关键是对决策树的构造，典型的构造决策树的方法是 ID3 算法和 C4.5 算法。这些算法是都是根据属性的重要性来依次把各个属性分配到相应的结点上面去。

(3)人工神经网络

人工神经网络学习方法对于逼近实数值、离散值和向量值的目标函数提供了一种健壮性很强的方法。它是通过模拟人类大脑的神经元，形成具有输入和输出的单元结构。对于某些类型的问题，如学习解释复杂的现实世界的传感器数据，人工神经网络是目前最为有效的方法。

具有以下特征的问题我们都可以用神经网络来解决：

①.实例是用很多“属性-值”对表示的；

②.目标函数的输出可能是离散值、实数值或者由若干实数属性或离散属性组成的向量；

③.训练数据可能包含错误；

④.可容忍长时间的训练；

⑤.在实际应用的时候可能需要快速求出目标函数值；

⑥.人类能否理解学到的目标函数是不重要的。

人工神经网络主要训练感知器以及由感知器构成的多层网络结构（包括前向和反馈网络）。在神经网络里面的典型的模型有：自适应共振、双向联想存储器、反向传递、对流网、认识机、感知器、自组织映射网等

(4)贝叶斯学习

贝叶斯网络的学习是贝叶斯网络模型的构建和对已存在贝叶斯网络模型的优化。由于可以利用的数据日益增加和数据越来越容易获取，使得用数据来进行贝叶斯网络的结构学习和条件概率表的学习变得十分可行，贝叶斯网络的条件概率表的学习又常称为贝叶斯网络的参数学习。

(5)基于案例的学习

前面的方法都是根据一系列的训练样本，然后形成一定的目标函数把训练样本一般化。而基于实例的学习则不然。基于实例的学习方法只是简单地把训练样本存储起来，从这些实例中泛化的工作被推迟到必须分类出新的实例时。每当学习器遇到一个新的查询实例，它分析这个新的实例与以前存储的实例之间的关系，并据此把一个目标函数值赋给新的实例。

基于案例的学习方法主要包括最近邻法和局部加权回归法，它们都假定实例可以表示为欧氏空间的点。此外，基于案例的学习方法还包括基于案例的推理,它对实例采用复杂的符号表示。

基于案例的学习方法实际上是一个消极学习方法。

(6)遗传算法

遗传算法是一种受生物进化过程启发的学习算法。遗传算法研究的问题是搜索候选假设空间并确定最佳的假设，一般是通过变异和交叉重组当前已知的最好假设来生成后续的假设。在遗传算法中，假设一般用二进制来表示（便于变异和交叉遗传算子）。

遗传算法的设计有一个共同点：算法迭代更新一个假设池（也称之为群体）。每一次迭代中根据适应度函数评估群体中的所有成员，然后从当前群体中用概率方法选取适应度最高的个体产生新一代群体。在这些选取的个体中，一部分保持原样进入下一代群体，其他通过交叉和变异等遗传的方法产生新的个体作为下一代群体的一部分。

(7)规则学习

对学习得到的假设，最具有表征力的和最能为人类所理解的表示方法之一为 if-then 规则的集合。而规则学习实际上就是学习这样的规则。规则一般包括不含变量和含有变量的。不含变量的很容易理解和得到。最为重要的是学习含有变量的规则集合（也称之为 Horn 子句集合）。由于一阶 Horn 子句集合可以被解释为逻辑编程语言中的程序，所以学习的过程经常被称之为归纳逻辑编程(Inductive Logic Programming，简称 ILP)。

(8)分析学习(基于解释的学习)

前面的方法都是归纳学习方法。这些归纳学习器在实践中都有一个关键的限制就是学习实例的数据不足时性能较差（这已经被证明，参见文献[35]第七章）。而分析学习使用先验知识和演绎推理来扩大训练样本提供的信息，因此，它不受数据不足的影响或影响较小。

分析学习的典型方法是基于解释的学习（Explanation-Based Learning，简称 EBL）。它包括 2 个阶段，分析阶段、泛化阶段。具体来说，首先使用先验知识来分析（或解释）观察到的学习样本是如何满足目标概念的。得出训练样本中哪些特征是相关的，哪些是无关的，然后案例（样本）就可以基于逻辑推理进行泛化，而不必经过统计推理得出。

(9)强化学习

强化学习的过程实际上就是给要学习的主体—Agent 一个任务，Agent 通过不断感知环境，然后根据环境做出动作的选择；如果成功，就对相应的动作做出奖赏，如果失败，就对相应的动作做出惩罚；通过不断的学习，最后会达到一个稳态（以后 Agent 在相应的环境下一定会做相应动作）。强化学习一个最突出的优点就是不要求有任何的先验知识。这是它跟前面的所有学习算法最根本的不同。

以上提到的绝大多数算法都是可以应用到 Robocup 机器人足球仿真比赛当中，但使用不同的学习算法得到的效果是不一样。但是如果仅仅使用其中的一种，效果可能也达不到要求，因而可能需要不同的方法交叉使用，或根据不同的情况选择特定的方法。

7.2 个体技术的实现

在 Robocup 中，设计球员的个体技术如带球、传球、截球、射门的时候一般是由更低级的动作组成。有时候还要在这两者之间还要加上一个中间层。就针对于这些基本动作我们具体介绍一下它们的实现方法。

7.2.1 截球（intercept）

1.问题描述

截球问题可以归纳成如图 5.1 的一个简单的场景：白圆圈代表球，黑色的圆圈代表球员，dist 为球员到球的距离，α 为球到球员之间的连线和球运动方向的夹角，speed 为球的即时

的运动速度。球的速度随运动衰减。截球问题归结为给定 dist、α 和 speed，决策出队员正确的截球角度 β，或者是当截到球时，球运动的距离，并给出对截球所可能花的时间的估计。具体的运动模型见 2.7.2.8 相关的球员和球的运动模型。

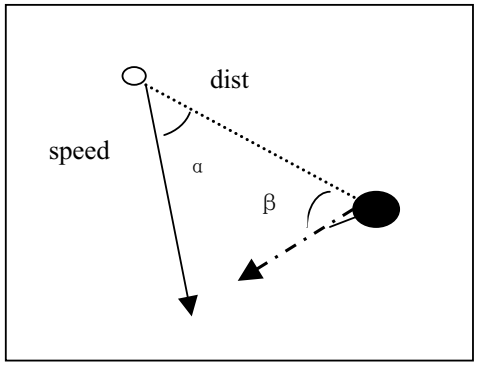


图 5.1 截球示意图

2.解决方法

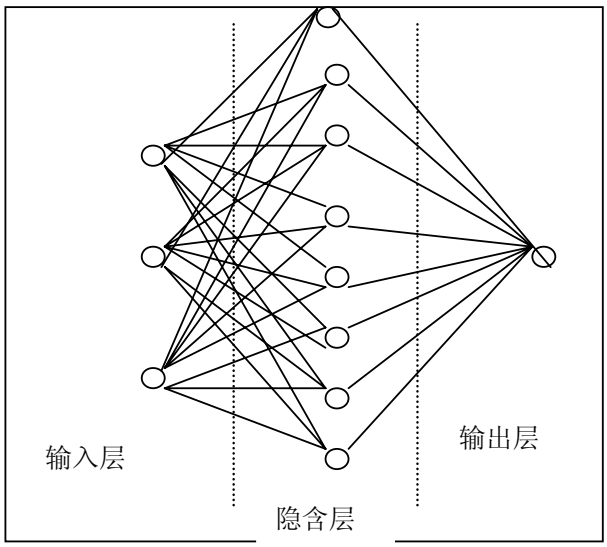
1) 解析法

通过示意图和前面介绍的运动模型我们可以通过列出关于时间的方程，然后采用 Newton 迭代法求出方程的根，可以求出认为可以求出 3 个根，显然第三个根的价值不是很大，目前关注的是前 2 个根，然后根据高层策略选择在哪个根对应的点（前点和后点）进行截球。TsinghuAeolus 目前采用的就是这种方法。

2) 通过机器学习的方法进行离线学习

目前这是一种简单并且通用的方法，典型的方式是通过搜集大量成功的截球样本（反应为一些特征变量集），在使用这些样本通过离线学习的方法形成一个决策函数。离线学习主要采用贝叶斯网络和神经网络；其中 BP 网络目前比较通用。CMU99 和 Hfut 目前采用这种方法。

简要介绍一下 BP 网络：计复杂的场景并存储这些场景了。HfutAgent 使用 BP 算法处理截球问题。



BP 网络示意图

BP 网络的特点是信号由输入层单向传输到输出层，同一层神经元之间不传递信息，每个神经元与邻层所有神经元相连，连结权重用ω_{ij}表示，各神经元的作用函数为 Sigmoid 函数：$f(x)=1/(1+e^{-x})$。同时它正向传播信号，反向传播误差。BP 网络如下图。

隐含层是 BP 网络的基本特征之一，事实上如果没有隐含层也就无所谓误差的反向传播了。但对隐含层节点个数的选择到目前为止还没有确定的规则，根据经验公式$\sqrt{m\times n}+1\sim 10$ (m、n 表示输入输出节点的个数)，由于我们这儿有 3 个输入，所以我们

选择了 8 个节点。中间隐含层也是使用了 1 层，包括输入和输出总共 3 层。

在图 4.2 所示的 BP 网络结构中，设网络输入为 x_1, x_2, x_3，输出为 y。输入层各神经元的激发函数选用比例系数为 1 的线性函数，则网络输入层的输出分别是 x_1, x_2, x_3，隐层神经元的输入是:

$$I_i = \sum_{j=1}^{3} \omega_{ij} x_j \quad (i = 1,2,3,...7) \qquad \text{(公式 4.1)}$$

神经元的输出为：

$$O_i = 1/(1+e^{-I_i}), \qquad \text{(公式 4.2)}$$

v_i 为输出层神经元与隐层神经元 i 的连接权，则网络输出为:

$$y = \sum_{i=1}^{7} v_i O_i \qquad \text{(公式 4.3)}$$

在由 ω_{ij}、v_i 组成的连接权向量 W 初始化之后，就可以在给定一组网络输入后，由上述式子求出网络的输出 y，此为正向信号传播过程。

对某样本($x_{1p}, x_{2p}, x_{3p}; t_p$)，p 为样本数,由正向计算得到 y_p ,定义网络输出误差为:

$$d_p = t_p - y_p \qquad \text{(公式 4.4)}$$

误差函数为

$$e_p = 1/2 \times d_p^{\,2} \qquad \text{(公式 4.5)}$$

一般的，W 值随机给出，求得 y_p 后，误差值较大，网络计算精度不高。在确定网络中隐层神经元数目 m 的情况下，通过调整 W 的值，逐步降低误差 d_p，以提高计算精度。

在反向计算中，沿着误差函数为 e_p 随 W 变化的负梯度方向对 W 进行修正。设 W 的修正值为 $\triangle W$：

$$\triangle W = -\eta \frac{\partial e_p}{\partial W} \qquad \text{(公式 4.6)}$$

η 为学习率，取 0-1 间的数。该修正方法的弱点是收敛速度慢，并存在能量函数局部最小值，在此对其增加附加动量项进行修正，即取：

$$\Delta W^{(n)} = -\eta \frac{\partial e_p}{\partial W} + \alpha \Delta W^{(n-1)} \qquad \text{(公式 4.7)}$$

$\Delta W^{(n)}$ 为第 n 次迭代计算时连接权的修正值，$\triangle W^{(n-1)}$ 为前一次迭代计算时所得的连接权的修正值，α 为动量因子。

将公式 4.4、4.5 公式 4.7，并加以推导，求得对于样本 p 时，ΔW 中各元素为：

$$\Delta v_i^{(n)} = \eta d_p \frac{\partial y_p}{\partial v_i} + \alpha \Delta v_i^{(n-1)} = \eta d_p O_{ip} + \alpha \Delta v_i^{(n-1)} \qquad \text{(公式 4.8)}$$

$$\Delta w_{ij}^{(n)} = \eta d_p \frac{\partial y_p}{\partial w_{ij}} + \alpha w_{ij}^{(n-1)} = \eta d_p v_i O_{ip}(1-O_{ip}) X_{jp} + \alpha \Delta w_{ij}^{(n-1)}$$

(公式 4.9)

最后采用迭代式$W + \Delta W \to W$对原W进行修正计算，得到新的连接权向量W。

对于所有的学习样本，均按照样本排列顺序进行上述的计算过程，从而求出学习样本的能量函数值：

$$E = \sum_{p=1}^{p} e_p$$

利用 E 值对网络计算精度进行评价，当 E 值满足 E<0.00001 时，停止迭代计算，否则，进行新一轮的迭代计算。

训练时，我们构造出各种情况的截球场景(传球队员固定位置，离散传球速度和传球队员和截球队员之间的相对坐标 x, y)，截球队员使用各种角度截球，当成功的截球时，就将成功的数据记下。采集到的成功的数据送入神经网络用 BP 算法进行训练。神经网络作为一个记忆的载体记录下这些成功的例子，能够进行实际各种场景的截球决策。

3) 强化学习的方法进行在线学习

在使用强化学习的时候关键是确定状态空间、动作空间、目标状态、策略函数（代价函数）以及价值函数。首先是确定状态空间(s)，也就是 world state，一般状态空间都很大，在计算和存储方面就存在很多困难，这也是目前强化学习往机器人足球中应用的难点地方；这就需要进行简化和处理。然后就是确定动作集，一般把原子动作作为动作集。目标状态是停止学习的终止条件，在学习的时候一般把得到球作为目标状态。所谓策略函数就是在当前的状态在选择动作的函数，这样的函数学要自己去设计，原则是能够把代价最小、利益最大的动作选择出来。而价值函数是在选择一个特定动作以后，是成功还是失败，相应的对这个状态下的这个动作的代价（利益）进行相应的修正，一般是加上（成功）、减去（失败）一个值，经过足够长的时间的学习就能达到一个稳态（也就是价值函数的性能较好）。

7.2.2 传球（pass）

1.问题描述

相对截球而言，传球更加复杂的一种动作，主要原因是在传球的时候实际上已经引入 2 个球员进行协作的问题了。在设计时一般考虑 2 种方式进行传球：传给某一特定的人和传到某一点。对传球进行描述的时候，可以采用这种方式：描述传球队员的周围环境，用状态 S 表示周围的环境或提取环境的一些特征属性向量 A（a1,a2,…,a_n）。根据这些 S 或 A 来选择合适的传球方向和出球速度。

2.解决方法

根据上面对传球的描述，我们可以使用决策树学习算法，也可以使用基于神经网络的计算学习方法。

(1).决策树学习算法：

CMU 使用的就是这种学习算法，具体使用了 C4.5 算法，在选择下面的特征属性：

①.传球球员到接球者的距离和方向（2 个）。

②.传球队员到其他队友(不包括接球队员)的距离和方向(20 个)。

③.球球队员到对手的队员的距离和方向(22 个)。

④.经排序（按距离）以后的接球队员到队友的距离和方向(20 个)。

⑤.经排序（按距离）以后的接球队员到对手的距离和方向(22 个)。

⑥.从传球队员到接球队员之间的一些分布统计属性(90 个)。如以传球队员为中心，由接球队员以及其他队友构成的扇形区域内对方球员的数量等等。

⑦.球员所在的区域特性(44 个)。

训练的时候，首先设定传球队员的位置，随机设置接球和其他队员的位置；传球队员确认要传球；其他队员进行跑位；传球队员根据决策树确定接球队员。接球队员和其他队员（指对手球员）都采用已经训练好的截球动作去获得控球权。接球队员获得控球权就认为是一次成功的传球，否则，就认为失败。

(2)基于神经网络的计算学习

计算学习一直是机器学习的重要研究内容，它主要是通过计算的方法将那些错的很离谱的假设排除出去形成，通过计算机的快速计算能力得出最有可能的假设并把该假设认为是可能近似正确（probably approximately correct，PAC）。另一方面，神经网络集成作为一种新兴的神经计算方法，具有比单一神经网络系统更强的泛化能力，因此，如果将神经网络集成与计算学习相结合，将可望获得更好的效果。在这一思想的基础上，提出了一种基于神经网络集成的计算学习算法，以神经网络集成作为计算学习的前端，首先利用其产生计算学习所用的数据集，在产生数据集时，采用能够较好地反映神经网络集成性能的数据生成方式，使得用于计算学习的示例能够受益于神经网络集成的强泛化能力，以最终获得较高的预测精度。

在使用计算学习来进行传球训练的时候，首先是确定在特定传球路线上的传球速度的选择。我们参考了清华[23]的对传球时穿越速度的概念。如图 4.4，假设白圆圈表示的 1 号队员要把球传给用黑圆圈表示的 2 号队员，X 表示对手。线 L 为对手和截球队员的垂直平分线。显然对于垂直平分线与球轨迹的交点 p 以内的点，对方队员能比我方队员能先跑到；反之，交点以外的点，我方队员先跑到。如果传球队员踢出速度大小合适的球，使得对手在交点以内都无法截到球，那么我方队员就必然可以比对方先截到球。如果以此速度踢出球，此队员不能在该点以前截到球，而且以小于此速度的任何速度踢出球，对手都可能在该点以内截到球,那么这个速度称为对于某个队员穿越在球运动轨迹上的某一点的穿越速度。也就是说，我们只要而且必须要以大于穿越速度的速度传球，球就能传到队友脚下。

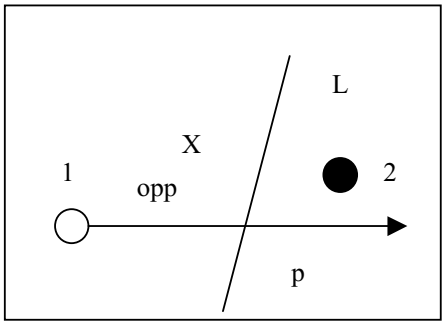


图 4.4 传球示意图

注意到这个分析基于图 4.4 队友在对手后面的情况。如果反过来，队友在对手前，则传球者应该以小于队友的穿越速度的速度传球，以保证队友在交点以前截到球。对图 4.4 的情景，我们把 p 点以前的区域称为对手的接球区域，p 点以后的称为队友的接球区域。对于队友，穿越对手的穿越速度为传球给他的速度的下限。如果考虑队友后面可能有一个对手，则给他一个传球速度上限的限制。

图 4.4 考虑了一个队友和一个对手的简单传球场景，多个对手和队友的场景也有类似的分析。在一条传球线路上，每个队员（包括对手和队友）或者没有接球区域，或者有一个接球的区域和一个传球速度的上、下限。

首先，我们利用人工神经网络中的 BP 网络，训练得到在特定传球路线上面传给每个球员的穿越速度。

第一步，采集样本。确定传球队员的位置和随机置接球队员的初始位置。在训练中传球者从一个较小的速度开始，沿传球线路传球，接球者利用训练好的截球技能进行截球，如果截球点在图 4.4 的 p 点以前，则传球者提高速度，继续尝试；否则，穿越速度为该次训练的传球速度。如此这样收集传球队员和接球之间不同的距离和角度情况下的穿越速度。

第二步，用人工神经网络中的 BP 网络拟合得到传给每个球员的穿越速度。其中输入是传球队员和接球队员的距离和传球路线的方向和传接球队员之间连线的夹角，输出是穿越速度。

利用 BP 得出每个球员在本方传球时自己能够接球的穿越速度作为我们计算学习的基础。如果传球到一点上面，那传球路线就确定了，我们只要计算用什么样的穿越速度就可以了。如果是传给特定的球员，可以根据穿越速度淘汰掉那些接球队员没有接球区域的传球路线，选择接球队员的穿越速度区间最大的那条传球路线（主要是增强系统的抗噪音性）作为我们的目标传球路线。计算学习要学习的就是在给定了场景（主要记录的是传球队员和场上所友队员的相对位置和角度）的情况下得出最佳的传球路线。

7.2.3Fastkick

1.问题描述

在 Soccer Server 中，队员的身体和球都使用一个圆来表示，前相互之间的位置不允许有相互重叠的部分。当球离队员的距离小于某个值时，这时队员就可以向 Server 法一个包括角度和力量两个参数的 kick 命令，对球施加一个矢量加速度。由于球的加速度有上下，且球有初始速度，因此常常无法通过一个 kick 命令才能实现把球加速到所希望的速度上面去，也就是一个踢球动作需要一系列周期的 kick 命令才能实现，这就需要 Fastkick。

2.解决方法

1） 直接经验式代码：将设计者的经验直接写成代码进行踢球决策。

2） Case-base Learning：在这种方法中，控球范围被离散成为一些点的集合，每周期的状态用 4 个参数来描述（球员的速度、球的相对位置、期望的出球速度、可以到达的点的集合）。然后构造了若干个 Case Bases，每个 Case Bases 都能根据输入状态来返回一个 PDL，这个 PDL 描述了每个科到达的点作为中间点的好坏。

3） 强化学习：更前面提到的截球的相似，主要是通过学习提高价值函数的性能。

4）清华提出的考虑对抗的强化学习—Q 学习及在线规划。在这当中，用一张 Q 表来存储状态-动作对，再把一些状态（如这时球也对手的范围之内或球出界）屏蔽掉。然后进行训练得出实战时对抗性能更强的 Q 表。

第 8 章 球队的整体策略

Robocup 是一个充满合作与对抗的多 Agent 环境，其中的 MAS 协作可以看成球队的整体策略或球队的区域协作问题。前面我们介绍了 Agent 的结构以及个体智能，指出了一个 Agent 在 robocup 的环境下根据自己对环境信息的感知，然后做出决策产生一个动作来影响环境，这充分的体现了 Agent 的智能性，但单个 Agent 在 Robocup 中的作用是有限的，它必须同其他 Agent 进行交互，要考虑其他 Agent 的利益，这也是 Agent 的社会性考虑的问题。我们设计的决策模块应该考虑 Agent 在球场上的协作工作。为达到进球这样一个大的目标，每一个 Agent 应该是在基于局部观察的情况下进行全局的决策，在全局的决策框架下进行自己的动作选择。在协作的时候应该考虑到协调与协商。

本章的主要内容组织如下：8.1 说明 Robocup 球队策略里面应该包含的那些内容，指出了哪些方面需要球队策略来进行规划。8.2 介绍了一个经典的球队里面球员的跑位规划方法－FC Portugal 的 SBSP(Situation Based Strategic Positioning)，在实战中 HfutTeam 也结合使用了这种方式进行了跑位。8.3 介绍了使用强化学习的一个局部战术配合 3Vs.2 战术。8.4 介绍了 HfutTeam 中涉及到的进攻体系，主要包含进攻队员的动作选择，一般队员的协同以及跑位。8.5 介绍了我们的防守体系，涉及到防守点的选择，防守动作的评价、守门员的防守的动作等方面的内容。8.6 总结了 HfutTeam 的特点。

8.1 Robocup 球队策略综述

从足球比赛的角度来看，体现球队整体策略的就是球队的整体进攻和防守体系，当所有球员都按照相同的决策框架来进行决策时，球员的决策依据都相同，对其他队员的预测就比较准确，整个球队就能够体现出一定的协调性了。当 Agent 间的意愿一致的时候他们就会进行协作，当他们的意愿不一致的时候，就要进行协商，处理矛盾。Remco 在他的论文中对球队的整体策略进行了一下归纳[26]。

(1)．整体策略指定了队伍的阵型以及其中队员的位置。更进一步，它可以决定什么情况下应该采用什么样的阵型。

(2)．整体策略定义了阵型中不同的角色，并把角色分配给不同位置的球员，同时确定哪种异构球员适合哪种角色。

(3)．对于每种类型的球员（后卫，中场，前锋等）球队策略应该根据它的角色确定分配给他哪些相应的行为。如对一个防守队员来说，一般它应该比进攻队员保守，它选择的动作也应该是偏向防守的。

(4)．整体策略一般还包括球员应该如何根据场上形势来调整行为的信息。比如在特定

的策略下，一个球员在对自己的动作进行选择的时候应该考虑到位置和位置所在的区域，同时还要考虑队友和对手的位置。

(5)．整体策略应该指定如何协调各个球员的行为．因为不同的球员由于感知的不同，在策略的执行上可能出现不协调。如何减小这种潜在不协调的危险是策略的一部分。

(6)．整体策略还应该能够在比赛中合理的管理球员的体力。如球员在比赛中如果自己的体力特别低，除非万不得已一般尽量减少跑动以恢复体力。

此外，球队的整体策略还应该考虑其他一些因素。比如说

(1)．对手的强弱。对强弱不同的对手的动作是不一样的，如对手的截球能力强在传球的时候对传球的细节考虑就应该更细一点，对手弱，考虑的就少一点，可以为追求更好的结果而采用更加冒险的传球。

(2)．对方球队的整体倾向。如对手是倾向于进攻还是倾向于防守。

(3)．比赛的场上情况。比如说场上的比分，是落后还是领先。

(4)．场上是否处于关键状态。如我们已经攻入对方禁区或对手攻入我方的禁区采取紧急动作。

(5)．其他一些因素。是否考虑换人、是否考虑球员类型的转换等等。

在设计球队的整体策略时，我们考虑了上面的绝大多数因素。从抽象意义上考虑 robocup 中球员的动作选择实际就是我方在控球和还是对方在控球这两种情况，我们控球就进入进攻状态，对方控球就进入防守状态。在设计的时候参考了中科大的主导 Agent 概念[46]，进攻

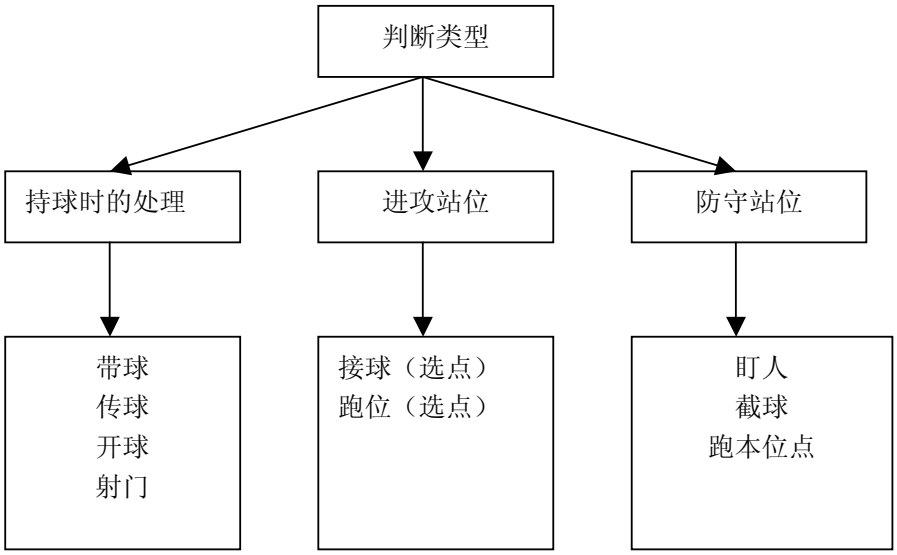


图 8.1 HfutAgent2003 内部的球队整体策略部分

的时候主要以持球队员为主，持球队员进行动作的选择，其他队员进行合理的跑位。在防守的时候以截球为主，截球队员选择合适的截球点进行截球，其他队员选择跑位或盯人。实际上在 Robocup 仿真机器人足球比赛当中，球员对跑位点的选择相当重要，对整个阵型的保持也是很重要的。

图 8.1 表示了 HfutAgent 表示球队整体策略的部分。判断类型主要是指判断当时的本队战术和球员所处的位置、球员的类型以及对手模型等方面的信息。结合感知的世界模型，可以得知 Agent 是处于什么状态，本队进攻还是防守，是否控球，是否进攻跑位、防守跑位等。然后根据相应的状态采用对策论的方法选择自己的动作[3]。

8.2 基于场上形势的战术跑位（SBSP）

8.2.1 SBSP 介绍

SBSP[47]首先是由 FC Portugal 提出的，它是结合了 Peter Stone 在 CMU 球队里面的阵型和位置的概念，并在此基础上面引入了战术、场上情况、队员类型等概念。

定义 1：阵型是通过站位来进行确定的。

$$Formations_i = \{ Formations_{i,1}, Formations_{i,2}, ..., Formations_{i,nformations_i} \}$$

$$Formation_{i,j} = (AgentPosi_{i,j,1}, AgentPosi_{i,j,2}, ..., AgentPosi_{i,j,nAgent})$$

$$\forall i=1..ntactics, \forall j=1..nfomations_i$$

定义 2：阵型中 Agent 的站位是由基本参考位置、角色、位置的重要性组成。

$$Posi_{i,j,p}=(RefePosi_{i,j,p}, PosiRole_{i,j,p}, PosiImport_{i,j,p})$$

$$\forall i=1..ntactics, \forall j=1..nfomations_i, \forall p=1..nplayers_{i,j}$$

定义 3：预规划包括阵型转变规划以及随着时间变化的位置评估规划、角色评估规划、动作选择评估规划。

$$PresetPlans_i = \{ PrePlans_1, PrePlans_2, ..., PrePlans_{nplans_i} \}$$

$$PresetPlan_{i,k}=(PlanActInfo_{i,k}, PlanPosiEvo_{i,k}, PlanRolesEvo_{i,k}, PlanActEvo_{i,k})$$

$$\forall i=1..ntactics, \forall k=1..nplans_i$$

定义 4：战术是关于阵型、阵型触发规则、预规划的组合。

$$Tactic_i=(Formations_i, FormationActivRules_i, PresetPlans_i) \quad \forall i=1..ntactics$$

定义 5：球队策略中的角色应包含积极特性、战略特性、场上重要形势规则。

$$Role_i=(Act\_Chars_i, Stra\_Chars_i, Cri\_Situ\_Rule_i)$$

定义 6：基于位置的角色（PosiRole）定义了 Agent 的行为特性。

$$PosiRole_{i,j,p} \in \{1..nroles\}$$

$$\forall i=1..ntactics, \forall j=1..nfomations_i, \forall p=1..nplayers_{i,j}$$

定义 7：球队策略是关于战术、战术触发规则、Agent 角色、对手模型策略、对友模型策略以及通讯协议的集合。

$$TeamStrategy = (Tactics, TactActivRules, Roles, Opp\_ModStras, Team\_ModStras, Comm\_Prots)$$

---

[3] 这时考虑 Agent 的个体技术已经训练完毕，也就是说动作都能成功的完成，对策论评价是采用相应的动作以后对球队整体的影响。

$Tactics=\{Tactics_1,Tactics_2,...,Tactics_{nTactics}\}$

该集合表示所有的战术

$TactActivRules=\{TactActivRules_1,TactActivRule_{s2},...,TactActivRules_s\}$

该集合表示所有的战术触发规则

$Roles=\{Role_1,Role_2,...,Role_{nRoles}\}$

该集合表示 Agent 的角色，它决定了 Agent 的动作倾向。

$\text{Opp\_ModStras}=\{\text{OppModStrat}_1,\text{OppModStrat}_2,...,\text{OppModStrat}_n\}$

该集合表示所有的对手模型策略

$\text{Team\_ModStras}=\{\text{TeamModStrat}_1,\text{TeamModStrat}_2,...,\text{TeamModStrat}_m\}$

该集合表示所有的队友模型策略

$\text{Comm\_Prots}=\{\text{CommProt}_1,\text{CommProt}_2,...,\text{CommProt}_t\}$

该集合表示所有的通讯协议。

定义 8：积极形势是指 Agent 确认此时场上形势是重要形势时的场上形势。它的确定原则是场上重要形势规则为真。

定义 9：战略形势是指 Agent 确认此时场上形势不是重要形势时的场上形势。它的确定原则是场上重要形势规则为假。

定义 10：Agent 的 SBSP 是关于当前的战术、场上形势（在阵型中定义）、球员类型（定义了 Agent 的战略特征）的函数。

在使用的时候首先定义了：

全场信息 *Global_Infor*=(*Time, Result, StaticticsInfo, OppModInfo*)；

场上形势信息 *Situ_Info*=(*BallPossInfo,BallInfo,TeamInfo, OppInfo*)；其中：

StaticsInfo=(*BallPossInfo, ShootInfo, PassInfo, DribbleInfo, CornersInfo, OffsideInfo, FreekickInfo, KickInInfo, GoalieKickInfo*)；

PositionalInfo=(*BallLowlevel, OffsideLines, TeammateLowlevelInfo, OpponentLowLevelInfo, InterceptionInfo, GongestionInfo, DangerInfo*)；

ActionInfo=(*PassInfor ,ForwardInfo, ShootInf,* …)。

然后定义了场上重要形势信息：

$Cri\_situInfo_i$ ={ $Cri\_SituRule_{i,1}$, $Cri\_SituRule_{i,2}$, …$Cri\_SituRule_{i,ncriticalrules}$ }以及 Agent 的战略特性：

$Stra\_Char_i$ =(*BallPositionalAttraction*$_i$，*PositionalRectange*$_i$，*Behindball*$_i$ ，*ballAttraction*$_i$, *BallAttractionRegion*$_i$)。

根据球员当前所处的位置确定球员在相应阵型下的位置角色：

$AlocRole_p=PlayerRole\ (\ AlocPositioning_p\ )$

最后得出了球员(Agent)在场上的战略跑位：

$PlayerStrategicPosition_p$=

AdjustedPosition (ReferencePosition ($AlocPositioning_p$)+

$BallPosAttraction\ (AlocRole_p, BallPos, BallPosAttrac, PosRectangle)$+

$BallAttraction\ (AlcoRole_p, BallPos, BallAttraction, BallAttractionlRegion))$,

$BehindBall\ (AlocRole_p)$,

$OffsideConversion\ (AlocRole_p))$

注：(1)首先根据球员的位置角色，得出球员应该跑的基本位置，再考虑球所在位置区域对球员的吸引力、球本身对球员的吸引力,简单修正得到球员应跑的位置。

(2)考虑相应球员是否应该在球的后面*，一般后卫，守门员应在球的后面。

(3)考虑是否越位。

考虑到(1)、(2)、(3)最后得出球员的应该跑的战略位置。

5.2.2 对 SBSP 的总结

SBSP 是一个比较成功的球队整体策略的应用。它的基本思想是把场上状态分为积极的和战略的。球员的策略也分积极的和战略的。当场上状态是积极的时候，一般这是也就是关键的时候，如已经攻入了对方禁区，或进行战术配合的时候，这时 Agent 就不在考虑在进行高层的决策直接采用反应式的方式，如进入禁区面对守门员形成单刀，这是只有一个动作选择，那就是射门。当场上状态进入战略状态的时候，如进攻时，控球队员是处于积极状态，它进行动作的选择,其他队员都进入战略状态,这时它们的选择就是跑位,这就是使用 SBSP，跑到一个战略位置。防守时在考虑盯人、阻截球的情况下，其他队员就要选择合适的位置进行跑位。这时使用的也是 SBSP。一般的算法是：

```
IF 场上是战略状态
  IF 本方进攻
      进攻球员进行动作选择、其他队员按 SBSP 跑位
  ELSE
      防守队员截球、盯人、按 SBSP 跑位
ELSE
  采用积极状态下的即时动作
```

目前 SBSP 策略被许多球队所采用。

5.3 强化学习实现局部战术

5.3.1 强化学习策略

强化学习（Reinforcement Learning，简称 RL）是一种非监督学习的方法。它的研究过

*在 Robocup 中球员在球的后面主要是指球员处的位置是在球与本方球门之间

程是：Agent 从环境感知到环境信息，根据自己的策略选择合适的动作，从而改变环境状态，并得到相应的奖惩，从而来修正策略。强化学习可以概括成两个问题的研究，第一个问题是在搜索环境的状态空间的基础上，寻找一种策略用于动作选择，使自身能够得到最大可能的回报。另外一个问题是在一个决策过程（包含许多动作序列）中的奖惩分配；当一个决策过程成功时，每个动作对最后的成功都有一定的贡献，失败时，每个动作都应承担相应的责任。只有合理的把最后的奖惩分配到各个动作中，才能学习出正确的决策。

强化学习的关键是 Agent 与环境的交互可以看成是一个马尔可夫模型，也就是说，环境应该具有再现性，目前强化学习的方法主要有动态归化（Dynamic programming）、蒙特卡罗法（Monte Carlo）、Q 学习（Q-Learning）等。符号描述的强化学习如下：

设 s_t 为 Agent 在 t 时刻的状态，r_t 在 t 时刻得到的奖励（可以为负值），a_t 为 t 时刻采取的动作，R_t 为 t 时刻得到的长远利益，假设 $R_t=\sum_{k=0}^{\infty}\gamma^k r_{t+k}$，$\gamma$ 为折算率。从 R_t 的计算中可以看出，R_t 为 t 时刻以后的所有的到的奖惩折现之和，表示了所谓的长远利益。强化学习的目标就是通过学习得到策略 π，使得在状态 s_t 下决策出的动作 a_t 所获得的长远利益 R_t 尽可能大。

8.3.2 局部战术－3　.2

在 Robocup 中，我们考虑“战术”任务是这样一种情况：进攻一方在一活动范围内试图保持控球权，同时另一方试图夺回控球权。无论何时防守方获得控球权或球离开此区域，这个时间段就结束，球员便放置到另一个时间段（进攻方再次被给予控球权）。在本文中，这个区域设定为 20M*20M 的正方形，并总有 3 个进攻队员和 2 个防守队员。球员涉及的动作主要有：

HoldBall( )：持球，持球保持静止并尽可能远离对手。

PassBall(f)：传球，将球直接踢给 f。

GotoBall( )：跑向球。

GetOpen( )：跑位，跑到一个空旷的位置，在这个位置能得到传出的球。

所有防守方都采用固定的策略 GOTOBALL( )，也就是说他们试图阻截球，拿到球后用 HOLDBALL( )保持控球权。

3 打 2 战术是机器人足球的一个子问题，其主要的简化原理是：只有少量的队员加入，并在一个小区域内比赛，球员们使用相同的球队策略，他们不需要平衡攻防因素。尽管如此，在 3 打 2 战术在整个机器人足球比赛上也很有用。在进行该战术时我们使用了强化学习的方法。

在学习的时候，使用一个全能的教练 Trainer(也是 agent)在控制比赛，当防守队员在设定时间里获得控球权或球离开区域时 Trainer 就结束此时间段，就完成了一次对抗。在每个

时间段开始，他在区域内随机的置球和球员的位置，两个防守方队员从区域的一个角落开始，而每个进攻球员随机放在其他三个角落，每个角落一个球员。

以下讨论的是如何把战术反映到强化学习上：

我们把比赛时间离散成许多时间段，在每个时间段下面使用 RL。机器人足球服务器工作在离散的时间步下，t=0,1,2,….. ，每个 t 表示 100ms 的模拟时间，当一个时间段结束，另一个时间段立即开始。这样就产生一系列的时间段。RL 把每个时间段看作一系列连续的状态、动作和奖惩。表示如下：

$$s_0,a_0,r_0,s_1,\ldots,s_t,a_t,r_t,s_{t+1},\ldots,r_{T-1},s_T$$

s_t 表示在第 t 步的状态，a_t 表示 s_t 状态下所采取的动作，$r_t \in R$ 表示 s_t 状态结束的奖惩，s_{t+1} 表示下一个状态，如果我们希望在一个特定的时间段内涉及一个时间就加一个额外的下标给时间段号码。这样，第 m 个时间段在时间段 t 的状态表示为 $s_{t,m}$，对应的动作为 $a_{t,m}$，这个时间段的长度为 T_m，任何一个时间段的最终状态 $s_{T,m}$，是防守方获得控球权或球出界。我们设定防守方获得控球权或球出界奖惩为-1，其余的时候所有的分数都为 0。例如：$r_{T,m}$=-1，$m \in \{1..\infty\}$。

如何选择 RL 算法的动作集合是一个重要环节。原则上将一个队伍中所有队员的完全联合的决策范围作为可能的动作范围。我们就简化成：$a_t \in$ {HoldBall()，PassBall(f)，GotoBall()，GetOpen()}。

我们仅考虑图 5.1 中显示队员的策略。当一队友控球时或进攻方的其他队员能比持球队员更快到达地点，则持球队员执行 GETOPEN()，否则，如果前锋不控球，则执行 GoToBall()，若前锋控球，则他执行 HoldBall()或 PassBall(f)。在这个动作空间，我们就对持球队员的行为进行考察。我们可以用下面三个策略作为测试基准：

Random　随机执行 HoldBall( )或 PassBall(f)

Hold　　一直执行 HoldBall( )

Hand-coded　如果在 10M 内无防守方，执行 HoldBall( )，如果接球者 f 比持球前锋和第一队友处于更有利位置，传球可能成功则执行 PassBall(f)，否则也执行 HoldBall( )。

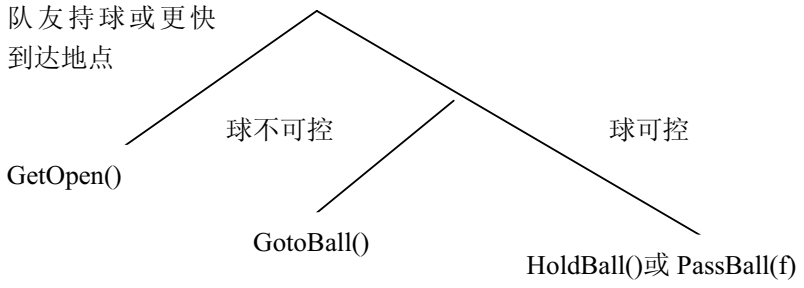


图 5.2 控球队员的策略空间

5.3.3.策略评价和策略学习

在 Robocup 中使用强化学习最难处理的就是如何表示状态空间。一般采用的方法就是把状态空间离散成多个重叠的网格，每个网格映射相应的状态。在分析的时候，每个格对应一个二进制状态，或是 1（当状态在这个格内）或是 0（状态不在这个格内）。

设计的实验方案是对所有的策略进行评价。也就是说，队员拥有固定的，预先确定的策略（不是在线学习的，这些策略也就是前面定义的），π代表所有 agent 的整体策略。在给定策略π下的状态 S 的价值函数被定义为：

$$V^{\pi}(s) = E\left\{\sum_{t=0}^{\infty}\gamma^{t} r_t \mid s_0 = s, \pi\right\}$$

这里γ是学习率，一个固定的参数（我们使用γ=0.9），T 是从 s 开始的时间段内每个时间步的编号。这样，进攻队员能确切控球的状态有最高值 0；所有其他状态是负值，立即失球的状态值接近-1。

（1）下面用 RL 方法学习去逼近价值函数V^{π}。特别需要指出的是，所有的防守队员在他们得到球之前，总是使用 GoToBall( )的方法；进攻队员使用 hand_coded 策略。我们把球在某个进攻队员能踢到的区域的每个模拟循环叫做决策步。只有在这些步骤中的进攻队员须做出决定是 HoldBall 或 PassBall(f)。在每个决策步中，全功能的教练记录一个有 13 个状态变量的集合。这些状态变量是以 F1,F2,F3,D1,D2,D3,C 为基础计算得出来的。F1－持球的进攻队员，F2— 另一个最靠近球的进攻队员，F3— 剩余的进攻队员，D1— 最靠近球的防守队员，D2— 另一个防守队员,C—区域中心（见图 5.2）。Dist（a，k）表示 a，b 点之间的距离，ang（a，b，c）表示 ab，cb 间的角度。例如：ang（F3，F1，D1）表示在图 2 中的角 F3F1D1，这 13 个状态变量分别是：dist(F1,C),dist(F1,F2),dist(F1,F3),dist(F1,D1),dist(F1,D2),dist(F2,C),dist(F3,C),dist(D1,C),dist(D2,C),Minimum(dist(F2,D1),dist(F2,D2)),Minimum(dist(F3,D1),dist(F3,D2)),Minimum(ang(F2,F1,D1),ang(F2,F1,D2)),Minimum(ang(F3,F1,D1),ang(F3,F1,D2)).

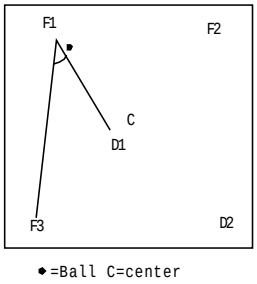


图 8.3.球员位置示意图

（2）强化学习去完善进攻队员的策略以达到尽可能长的时间控球。这里使用 Q-learning，即每个进攻队员独立学习。这样，所有的进攻队员都能学会相似的策略或不同的策略。每个

进攻队员有三个值函数，对应于控球时的每个可能动作（GetOpen( )，GotoBall( )，{HoldBall(),PassBall(f)}）,每个值函数有同样的一组 14 个格 tiling，这是我们的策略评价试验中最有效的情况。下面我们给出一个被用于每个进攻队员的学习算法。函数 UpdateRL(r)在后面定义。

Set counter=-1

If 球出界或防守方得到球 then

- If counter >0 then UpdateRL(-1)

Else if 球不可踢

- If counter ≥0 then increment counter
- If 可以跑向球达到控球 then GoToBall()

  Else GetOpen()

Else if 球可控 then

begin

- If counter >0,UpdateRL(0)
  - Set LastAction=Max(ActionValue(a,current state variables))
  - Execute LastAction
  - Set counter=0

end

Else (这时球被另外一个前锋控制)

- If counter >0 then UpdateRL(0)
- Set counter=-1

更新值函数算法

UpdateRL(r):

- TdError=r* $\gamma^{counter-1}$ + $\gamma^{counter}$ Max(ActionValue(a,current,state,variables))-ActionValue(LastAction,LastVariables)
- 使用 TDError 来更新值函数。

为了鼓励策略空间的扩展，我们用了简单的初始条件：未经过训练的逼近函数开始输出为 0，这在所有真实值为负的情况下是可行的。由于任何没有试过的动作往往看上去比已经试过多次并被确切赋值的动作更好，这就使所有动作都有机会被尝试。

使用强化学习进行局部 3VS.2 战术的学习要求对手的防守能力强，也就是对球的截球能力强，这方面存在两个困难：（1）跟自己比赛，提高有限，（2）和强队比赛，高层策略无法控制。另外，3VS.2 只是一个针对局部区域的策略，理论上是可以扩充到全局(11VS.11)，这时一个实际需要解决的问题是就是怎么解决庞大的状态空间。这也是我们以后的研究方向。

8.4 进攻体系

进攻是从本方队员控球开始，直到本方失去控球权为止。进攻体系考虑的问题持球队员进行备选动作的选择，其他队员或进行策应，或进行合理的跑位。

8.4.1 持球球员的动作选择

前面介绍了 HfutAgent 的个体技术，一个球员在控球以后，它可以选择带球、开球、传球（包括选择哪个队友进行传球）等动作。如何恰当的选择合适的动作是比较重要的问题。归根结底，在选择这些动作的时候必须考虑其他球员的利益，这就是多 Agent 的协作问题了。上节介绍了使用强化学习的方法来进行动作的选择。在本节里面讨论了使用对策论的方法来进行动作的选择。

对策论，又称博弈论，是使用严谨的数学模型研究冲突对抗条件下最优决策问题的理论。它的研究方法和其他很多利用数学工具研究社会经济现象的学科一样，是从复杂的现象中抽象出基本的元素，对这些基本元素构成的数学模型进行详尽分析，而后逐步引入对其形势产生影响的其他因素，从而分析其结果。为建立冲突对抗条件下决策的数学模型，必须数学化地描述冲突的参与者所有可能的行为方式及其行为结果，因此它也被视为数学的一个分支。

MAS 中的对策论就是求参与 MAS 协作和规划的所有 Agent 的整体效用，针对 Robocup，选择动作的影响因素，以及选择该动作的效用作为对策论的基础，持球者计算得出所有参与进攻的球员的效用最大值，并把最大值对应下的动作作为该持球队员的输出动作。

HfutTeam 系统中，设影响的因素 $cond_1$, $cond_2$, ……,$cond_n$， 确定它们对动作的影响 P（act| $cond_i$），计算 P（act | $cond_1$, $cond_2$, ……,$cond_n$），求出 $\max \sum_{a \in actionset} P(a \mid cond1, cond2, ..., condn) U(a)$ 的动作，其中 U（a）为采取某种动作后的效用，P（act | $cond_1$, $cond_2$, ……,$cond_n$）表示条件概率，只有高于特定的阈值的动作才被我们考虑作为备选动作。

目前，我们的影响因素主要有战术、阵型、球员类型、对手模型、队友模型、通讯模块等方面。由于我们每一个 Agent 使用的策略是相同的，所以 HfutAgent 很容易就能够得到队友模型，估计队友最有可能做出的动作选择。下面就一些其他要素简要说明一下。

(1)战术。战术需要分析的问题：当前的战术（特定阵型下的）、战术是否要改变等方面的内容。它主要包含阵型和进攻的倾向性。

(2)球员类型。在真实的足球比赛中，队员之间的属性是不同的。同人类的足球比赛一样，我们给每个仿真队员定义了一个类型。用 attackness、leftness 和 aggressiveness 三个取值在 0 和 1 之间的量来表示一个队员的类型属性。前两个量标明球员所打的位置。Attackness 区分队员的任务偏重于进攻的程度。Leftness 区分队员所在区域的偏左的程度，Aggressiveness 标明的是一个队员的动作侵略性。

(3)阵型。在足球比赛中，每个队员的活动区域是相对固定的，随着球的位置变化，活动区域也在变化，但是每个队员的相对位置变化比较小，这在足球比赛中称为阵型。

(4)对手模型。目前对对手模型考虑的较少，还没有一个比较成熟的基于对手的建模机

制。HfutTeam 考虑的是对手是一个理想模型。即假设对手是完美的（即把对手假设为最优），然后在这基础上面考虑对手的行为。

(5)通讯模块。通讯模块主要考虑的是要进行战术配合或紧急情况时同队友进行通讯。如我们要进行 2 过 1，这时就有必要同队友进行通讯，指导队友进行合适的跑位。

U(a)表示目前取决于设计者的经验。传球队员根据底层提交的内部状态，作出自己和队友的备选动作集，考虑因素的影响，选择一组利益和最大情况下对应的自己的被选动作，把这个动作返回作为自己的输出动作。对 U(a)我们简单的引入了奖惩措施，当这个动作成功以后我们对 U(a)加上适当的值来进行奖励。如果失败，就减掉适当的值进行惩罚。可以使用强化学习得到这些效用。

8.4.2 一般球员的跑位

一般球员的跑位主要考虑的是策应和跑位。策应球员指离球较近的、持球队员可能要把球传过来的若干球员。这些队员也采用前面的对策论里面的效用方法来选择合适的点（一般都是训练时候传球队员要传的目的点）待命。

其他队员的跑位就是跑到自己的本位点。本位点的获取主要是通过 SBSP。这在前面已由介绍，在这里就不再赘述。

8.5 防守体系

防守是从本方失球以后就开始的，一直到本方球员重新获得控球权。防守体系主要考虑的问题就是一般球组成的防守体系和守门员的防守。更进一步就是考虑加入了守门员的防守体系。

8.5.1 一般球员组成的防守体系

防守体系定义了四种防守动作盯人（Mark，主要是指盯无球队员）、卡位（Block）和跑本位点（Formation）。盯人的目的是防止被盯的对手接到传球；Block 的目的是阻碍对方控球队员带球前进；跑本位点是指球员站在本位点处防守，即进行区域防守。

该体系用防守动作的目标点（对于 Mark 来说，防守动作的目标点在被盯队员附近；对于 Intercept 来说，防守动作的目标点在阻截点上）离本位点的远近、目标点离球员当前位置的远近和防守目标的对防守的威胁程度来作为一个防守动作的特征。目标点离本位点远，说明队员要离开自己的防守的区域去防守，这对整体的防守利益带来不好的影响。目标点离当前位置远，说明该防守动作短期内不容易收到效果，因为可能在往防守目标点移动的过程中，原来的决策已经失去意义了；如果目标点离当前位置近，说明该防守动作可以很快见效，比如已经贴着对手，这时如果能坚持对目标的防守可能收到很好的防守效果。威胁程度大致可以用被防守的对手离球门的远近来衡量，离球门越近，威胁程度越大。如果考虑的细致些的话，可以把场上的区域离散化，根据足球专家的经验标明每个区域大致的威胁程度，通过

插值可以得到每个点的威胁程度。显然防守威胁程度越高的队员，防守动作的利益越大。

为了提高效率，在该防守体系中每个进攻球员只有一个防守队员负责防守。即针对同一名对方队员防守队员的动作是互斥的。每个防守队员在一个时刻只能执行上述防守动作中的一个。

在体系中还对防守进行了规划，使用的方法是基于局部视觉的全局规划。在该防守规划中，每个防守队员均考虑场上所有队员的防守策略。首先对所有的进攻-防守队员配对以及防守队员-本位点配对生成防守动作。接着根据前面提到的防守动作的特征得出防守动作的优先级；然后用分支定界的方法，求出这个最优的防守动作集；最后防守队员从方案中查询应该有自己执行的动作。算法如下：

(1)判断球被控制的类型。分为四种，双方同时控球（争球）、对方控球、我方控球、无人控球（抢球）。如果是对方控球或者无人控球但对方会先得球，我方队员就采取防守策略，并预测球下一次被控制的位置。

(2)判断对方控球球员。如果是争球情况，根据各种信息判断最有可能控球的人。

(3)判断防守任务和可用人手。根据世界模型和内部状态，判断有进攻威胁的对方球员并确定防守任务和确定可用来防守的我方球员。

(4)考虑所有用一个可用人手执行一个防守任务的情况，判断所应该采取的防守类型和相应的防守细节，并确定该次防守行为的效益。

(5)从防守任务和可用人手配对生成最优的防守方案（最大化防守效益和），然后提交自己有关的防守请求。

在所有的动作中，比较重要的是跑本位点，而对本位点获取是一个比较重要的问题。HfutTeam 通过使用了动态规划和强化学习的方法来获取场上各点的重要性（也称之为威胁度），然后根据球的位置得出不同战术、阵型下的所有球员的本位点。

8.5.2 守门员的防守

在实际的足球比赛中我们经常听到一个好的守门员相当于半支球队。守门员的防守在 Robocup 中的影响也很大。经过实践证明，在守门员的防守当中，最重要的是守门员的站位和守门员得球后的开球动作。当然，守门员的防单刀球、边路防守等动作也比较重要。下面首先给出守门员在比赛过程中的决策过程，然后着重介绍守门员的站位和开球动作。

决策过程：

(1)判断比赛是否开始；如果没有则调整站位，否则转(2)

(2)判断球的位置是否不可信，如果不可信则搜索球的位置，否则转(3)

(3)判断是不是我方开球，是则寻找好的开球点，然后开球。否则转(4)

(4)判断是否扑到球，如果能则扑球，否则转(5)

(5)判断是否能够踢到球，如果能够则踢球解围，否则，转(6)

(6)判断球在禁区内受否可以截到球，能够则跑到截球点，否则，直接跑向球干扰射门。

(1)守门员的站位

守门员的站位应该跟球的位置有关，我们根据 11 monkey[40]给出了一个守门员跑位经验函数，其计算方法如下：

令 L_1 为球的位置到左门柱的距离， L_2 为球的位置到右门柱的距离， SP_goal_width 为球门的宽度。则，守门员位置的 y 坐标为：

$$\frac{L_1 - L_2}{L_1 + L_2} \cdot \frac{SP\_goal\_width}{2}$$

其 x 坐标则根据球的位置动态的调整，一般确定一条线，守门员站在这条线上面。

经过分析，我们发现这个经验公式实际上使用了一个知识，那就是守门员站在该点，跑到球门 2 个门柱的时间是相同的，所以能够最大可能的扑住球。

(2)守门员的开球动作

在比赛中，守门员在得到球时并不要求立刻就将球发出去，而是要求在 25 个周期内将球发出去就可以了。可以充分的利用这 25 个周期进行开球的判断。当守门员刚得到球时，场上的情况必然是对方的球员都压在禁区附近，同时由于前几个周期，守门员刚作过扑球动作，对场上的信息的可信度不高。所以，在守门员得到球后，应该用大约 20 个左右的周期对场上情况进行收集。然后，根据场上的信息，对本方后场内的所有的对方球员之间的空当进行记录和统计，找到一个最大的空当，如果该空当的大小满足一给定的阈值，并且对手接到球的可能比我方低，则将球沿着这个空当的角平分线开出，这时球往往会传到中场的本方球员脚下。否则，将球直接开到中场位置的边线附近。

或许，找一个本方球员，将球直接传给他，表面上看来更合理些。但实际上，这种方法远比上面所述的方法效果差。这是因为，(1)球员通常都是在运动的，开向一个范围肯定比开向某一个点安全的多；(2)如果不考虑距离的远近，本方球员与对方球员往往是交叉站位的，对方球员的空当往往会有我方的球员；(3)守门员把球传给对友实际上增加了对友的踢球难度，也不是问题的根本解决办法。

前面守门员的防守作为了单独的一个模块。实际上，在真正的足球比赛当中，对一支球队来说，更重要的是守门员必须同后卫进行协调，建议设计一个以守门员为核心的防守体系，使用认知图或贝叶斯网络来表示防守时各 Agent 之间的联系。把最佳平衡协商方法引入到守门员与后防的协商机制中。

8.6 本章小结

本章主要介绍了 MAS 协作在 Robocup 中的一些应用。着重指出了强化学习在机器人足球中的应用，学习实现了一个局部战术，并使用 Q-Learning 对学习策略进行评估，同时也提出了强化学习在学习过程中存在的问题。本章还从抽象意义上把 Robocup 中 Agent 的协作分成进攻体系和防守体系。在进攻体系中我们是第一个把对策论的方法引入到机器人足球中的，但由于对效用的经验分配还没有调试到最佳效果，目前还没有充分体现出该方法的优越性。在防守体系中，着重研究了使用强化学习来进行球场上各区域的点的重要性的学习，结合基于局部视觉的全局规划，使防守水平得到很大提高。在守门员的防守中，研究了守门

员的站位和开球动作。

针对 Robocup 中 MAS 的协作评价，一般主观的看法就是球队的胜率；但这实际上并不能客观的说明问题，因为球队的成绩的好会不仅仅是取决于协作模块，跟球员的个体技术也有很大关系。可以考虑在使用相同的个体技术的条件下，来对使用不同的协作方法进行比较，这样才能客观的评价协作的好坏。

Linux 基础

本章内容：
Linux 简介
Linux 常用命令
Linux 下的编辑器

第 一 节 Linux 操作系统简介

一、 Linux 的历史

在讲述 Linux 的历史之前，我们先来看一看，究竟什么是 Linux：它是一种主要适用于个人计算机的类似 UNIX 的操作系统。相对于 Windows 系统让人头疼的错误报告，Linux 系统稳定而有效；相对于庞大的 UNIX 系统，Linux 有显得精致小巧。而且 Linux 对硬件配置的要求比较低，显得更加“个性化”。由于 Linux 的代码全部公开，任何人都可以对其做出相应的修改。这就吸引了更多的对操作系统感兴趣的程序员加入，所以 Linux 可以得到不断的发展。

Linux 源于一个芬兰学生（Linux Torvalds）的业余爱好。他在学习了有关操作系统的知识以后，开发了一个简单的操作系统内核程序，以 UNIX 为原型，实现了一些在当时只是在书本上提出的任务调度方法。他将自己的这个程序在大学里的 FTP 上发布后，引起了许多人的兴趣，他们不断的对这个简单的内核进行改进，不断完善 Linux，为 Linux 后来风靡全球奠定了基础。

下图是 Linux 的图标：

二、 Linux 的特点

Linux 不仅继承了 UNIX 的许多优良特性，同时又有自己的特点：

- 多任务。这是现代操作系统的基本特点，允许多个作业、多个进程、多个任务在一个时间段中同时并存，并发进行。
- 多用户。这是 Linux 继承了 UNIX 系统的长处。
- 虚拟内存和共享库。Linux 可以利用你的硬盘上的一部分作为虚拟内存，从而扩展你的可用内存数量。Linux 不使用分段，也没有虚拟内存的限制。Linux 同时允许那些使用标准子过程的程序在运行时共享子过程，从而节约了大量的系统空间。
- 跨平台能力。Linux 的内核是用 C 语言编写的，C 语言的硬件无关性使得 Linux 具有很好的可移植性。
- 强大的通讯和联网功能。网络连接在核心中完成，支持多种网络协议，功能强大。

当然，Linux 最突出的特点还是其作为自由软件的开放性。

三、 Linux 的版本

Linux 的版本分为内核版本和发行套件版本。内核版本号形如 A.BB.CC，A 可以是 0、1、2，后两位是 0-99 之间的整数。通常要选择的是发行套件的版本。

第 二 节　　Linux 常用命令

一、 登录和退出

Linux 启动后，给出 login 命令，等待用户登录。如果是合法的用户和密码，你就可以进入 Linux 的外壳。外壳给出命令提示符，等待输入命令。使用 logout 命令退出外壳。

二、 Linux 系统的外壳

外壳是一种命令解释器，它提供了用户和操作系统之间的交互接口。外壳是面向命令行的，而 X Window 则是图形界面。你在命令行输入命令，外壳进行解释后送往操作系统执行。

Linux 提供的常用外壳程序有：Bourne 外壳（bsh）、C 外壳（csh）和 Kon 外壳（ksh）。各个外壳都能提供基本的功能，又有其各自的特点。Bash 是大多数 Linux 的缺省外壳。它有以下的特点：

- 补全命令行。当你在 bash 命令提示符下键入命令或程序名时，你不必输全命令活程序名，只要你输入的字符串在当前状态下是唯一的，就可以按 TAB 键，bash 将自动补全命令或程序名。
- 通配符。在 bash 下可以使用通配符‘ *‘和‘ ？ ‘。*可以替代一个自符，而？则替代一个字符。
- 别名。在 bash 下，可用 alias 和 unalias 命令给命令或可执行程序起别名和清除别名。这样就可以按自己习惯的方式输入命令。
- 输入/输出重定向。经常用于将命令的结果输入到文件中，而不是屏幕上。输入重定向的命令是<，输出重定向的命令是>。
- 管道。是一种将一系列的命令连接起来。也就是把前面的命令的输出作为后面的命令的输入。管道的命令是|。
- 作业控制。作业控制是指在一个作业的执行过程中，控制执行的状态。你可以挂起一个正在执行的进程，并在以后恢复该进程的执行。按下 Ctrl+Z 挂起正在执行的进程，用 bg 命令使进程恢复在后台的执行，用 fg 命令使进程恢复在前台执行。

三、 外壳常用命令

1、 更改账号密码

格式：passwd

输入后，屏幕显示：

Old password: <输入旧密码>

New password: <输入新密码>

Retype new password: <确认新密码>

2、 联机帮助

格式： man [命令名]
例如： man ls
屏幕上就会显示所有 ls 的用法。

3、文件或目录处理
格式：ls [-atFlgR][name]
第一项是一些语法加量。第二项是文件名。
常用的方法有：
ls　　列出当前目录下的所有文件。
ls –a　　列出包括以 . 开始的隐藏文件的所有文件名。
ls –t　　依照文件最后修改时间的顺序列出文件名。
ls –F　　列出当前目录下的文件名及其类型。以/结尾表示为目录名、以*结尾表示未可执行文件、以@结尾表示为符号连接。
ls –l　　列出目录下所有文件的权限、所有者、文件大小、修改时间及名称。
ls –lg　　同上，并显示出文件的所有者工作组名。
ls –R　　显示出目录下以及其所有子目录的文件名。

3、 改变工作目录
格式:： cd [name]
name ：目录名、路径或目录缩写。
常用的方法有：
cd　　改变目录位置至用户登录时的工作目录。
cd dirl　　改变目录位置至 dirl 目录下。
cd ~user　　改变目录位置至用户的工作目录。
cd ..　　改变目录位置至。
cd ../user　　改变目录位置至相对路径 user 的目录下。
cd /../..　　改变目录位置至绝对路径的目录位置下。
4、 复制文件
格式：cp [-r] 源地址　目的地址
常用的方法有：
cp　file1　file2　将文件 file1 复制成 file2
cp　file1　dir1　将文件 file1 复制到目录 dir1 下，文件名仍为 file1。
cp　/tmp/file1　将目录/tmp 下的文件 file1 复制到当前目录下，文件名仍为 file1。
cp　/tmp/file1 file2 将目录/tmp 下的文件 file1 复制到当前目录下，文件名仍为 file2。
cp -r dir1 dir2　复制整个目录。

5、 移动或更改文件、目录名称
格式：mv 源地址 目的地址
常用的方法有：
mv　file1 file2 将文件 file1 更名为 file2。
mv　file1 dir1 将文件 file1 转移到目录 dir1 下，文件名仍为 file1。
mv　dir1 dir2　将目录 dir1 更改为目录 dir2。

6、 建立新目录
格式：mkdir 目录名

7、 删除目录
格式：rmdir [目录名|文件名]
常用的方法有：
rm –r dir1 删除目录 dir1 及其子目录下的所有文件。

8、 列出当前所在的目录位置
格式：pwd

9、 查看文件内容
格式： cat 文件名

10、 文件权限的设定
格式：chmod [-R] mode name
name：文件名或目录名。
mode：3 个或 8 个数字或 rwx 的组合。r-read(读权限)、w-write(写权限)、x-execute(执行)
常用的方法有：
chmod 777 file1 给所有用户 file1 全部的权限。

11、 文件的解压缩
格式：tar [option] [file]
option 的组合较为复杂，通常的解压缩方法是：
tar xvf filename.tar
通常的压缩方法是：
tar cvf filename.tar

四、 Linux 系统管理简介

1、 root 账号

如果使用 root 账号登录，用户可以存取任何的文件，控制任何的进程。所谓有所得也就必有所失，在这一账号下进行操作，用户的任何错误操作都可能破坏整个系统。一般说来，只有在十分必要的情况下才使用 root 账号。

2、 挂接文件系统

只有挂接到 Linux 的主文件系统后，文件系统才可以使用，即使是硬盘也得在挂接后才能使用。挂接文件系统的命令是：mount

格式： mount filesystem mountpoint

filesystem 是设备名， mountpoint 是文件系统在 Linux 中的挂接点。例如：要把软盘挂

接到文件系统中，可以使用：

mount /dev/fd0/mnt

如果想从系统中卸载挂接的文件系统，则可以使用 unmount 命令。例如：

卸载/dev/fd0 下面的软盘，则可以使用如下的命令：

unmount　/dev/fd0

第三节　　Linux 下的编辑器

Linux 下可用的编辑器有很多，其中常用的有 vi、gedit、emacs 等。下面就详细的介绍一下 vi 的使用方法。

1、vi 编辑器简介：

基本上 vi 可以分为三种状态，分别是命令模式（command mode）、插入模式（Insert mode）和底行模式（last line mode），各模式的功能区分如下：

1) 命令行模式 command mode）

控制屏幕光标的移动，字符、字或行的删除，移动复制某区段及进入 Insert mode 下，或者到 last line mode。

2) 插入模式（Insert mode）

只有在 Insert mode 下，才可以做文字输入，按「ESC」键可回到命令行模式。

3) 底行模式（last line mode）

将文件保存或退出 vi，也可以设置编辑环境，如寻找字符串、列出行号……等。

用 vi 打开文件后，是处于「命令行模式（command mode）」，您要切换到「插入模式（Insertmode）」才能够输入文字。切换方法：在「命令行模式（command mode）」下按一下字母「i」就可以进入「插入模式（Insert mode）」，这时候你就可以开始输入文字了。

编辑好后，需从插入模式切换为命令行模式才能对文件进行保存，切换方法：按「ESC」键。

保存并退出文件：在命令模式下输入:wq 即可！

2、常用的移动命令：

Vi 提供了三个关于光标在全屏幕上移动并且文件本身不发生滚动的命令。它们分别是 H、M 和 L 命令。

H 命令该命令将光标移至屏幕首行的行首（即左上角），也就是当前屏幕的第一行，而不是整个文件的第一行。利用此命令可以快速将光标移至屏幕顶部。若在 H 命令之前加上数字 n，则将光标移至第 n 行的行首

M 命令该命令将光标移至屏幕显示文件的中间行的行首。即如果当前屏幕已经充满,则移动到整个屏幕的中间行；如果并未充满，则移动到文本的那些行的中间行。利用此命令可以快速地将光标从屏幕的任意位置移至屏幕显示文件的中间行的行首。

同样值得一提的是，使用命令 dM 将会删除从光标当前所在行至屏幕显示文件的中间行的全部内容。

L 命令

当文件显示内容超过一屏时，该命令将光标移至屏幕上的最底行的行首；当文件显示内容不足一屏时，该命令将光标移至文件的最后一行的行首。可见，利用此命令可以快速准确地将光标移至屏幕底部或文件的最后一行。若在 L 命令之前加上数字 n，则将光标移至从屏幕底部算起第 n 行的行。

同样值得一提的是，使用命令 dL 将会删除从光标当前行至屏幕底行的全部内容。

按字移动光标：

首先介绍一下 Vi 中“字”的概念。在 Vi 中“字”有两种含义。一种是广义的字，它可以是两个空格之间的任何内容。Vi 中另一种字是狭义上的字，在此种意义之下，英文单词、标点符号和非字母字符（如!、@、#、$、%、^、&、*、（、）、-、+、{、}、[、]、~、|、\、<、>、/等）均被当成是一个字。

Vi 中使用大写命令一般就是指将字作为广义来对待，使用小写命令就是作为狭义对待。搞清楚 Vi 中字的含义后，我们就可以介绍按字移动光标的命令了。

Vi 一共提供了三组关于按字移动光标的命令，分别是：

w 和 W 命令

将光标右移至下一个字的字首；

e 和 E 命令

如果光标起始位置处于字内（即非字尾处），则该命令将把光标移到本字字尾；如果光标起始位置处于字尾，则该命令将把光标移动到下一个字的字尾。

b 和 B 命令

如果光标处于所在字内（即非字首），则该命令将把光标移至本字字首；如果光标处于所在字字首，则该命令将把光标移到上一个字的字首。

按句移动光标：

1. （ 命令

将光标移至上一个句子的开头。

2. ） 命令

该命令将光标移至下一个句子的开头。

按段移动光标：

在 Vi 中，一个段被定义为是以一个空白行开始和结束的片段。Vi 提供了关于按段移动光标的两个命令，分别为：

1. { 命令 该命令将光标向前移至上一个段的开头；
2. } 命令　该命令将光标向后移至下一个段的开头。

3、在命令模式下和文本输入模式下均可以使用屏幕滚动命令。

1. 滚屏命令

关于滚屏命令有两个：

· < Ctrl+u > 将屏幕向前（文件头方向）翻滚半屏；

· < Ctrl+d > 将屏幕向后（文件尾方向）翻滚半屏。

可以在这两个命令之前加上一个数字 n，则屏幕向前或向后翻滚 n 行。并且这个值被系统记住，以后再用< Ctrl+u >和< Ctrl+d >命令滚屏时，还滚相应的行数。

2. 分页命令

关于分页命令也有两个：

- < Ctrl+f > 将屏幕向文件尾方向翻滚一整屏（即一页）；
- < Ctrl+b > 将屏幕向文件首方向翻滚一整屏（即一页）。

同样也可以在这两个命令之前加上一个数字 n，则屏幕向前或向后移动 n 页。

3. 状态命令< Ctrl+G >

命令显示在 vi 状态行上的 vi 状态信息，包括正在编辑的文件名、是否修改过、当前行号、文件的行数以及光标之前的行占整个文件的百分比。

4．插入（Insert）命令

Vi 提供了两个插入命令：i 和 I。

i 命令

插入文本从光标所在位置前开始,并且插入过程中可以使用键删除错误的输入。此时 Vi 处于插入状态，屏幕最下行显示“--INSERT--”（插入）字样。

I 命令

该命令是将光标移到当前行的行首，然后在其前插入文本。

关于 Linux 下的 C++工程的编译

所有要参与的源代码之中，有一个名为 config 的文件，其中包含了需要编译的文件，所有的源代码都在其所在的文件夹中。可以指定需要生成的可执行文件的文件名。在 config 文件制作完之后，使用 make config_filename

之后，在当前目录下，就会生成在 config 中规定的可执行文件。