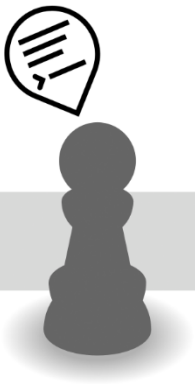


13

编程游戏AI

编辑器和其他工具对内容创作很重要：建立赋予游戏玩法的数据。从明显的关卡几何视觉设计，到构成角色行为的AI算法的配置。正如我们在前一章中看到的，不同的人工智能方法需要不同的工具：对关卡编辑或三维建模的扩展，或用于可视化和配置决策的定制工具。确切的要求取决于正在使用的方法。

与此相反，有一种工具是永远必要的。以至于在讨论支持性技术时，它很容易被忽略。所有的人工智能都是编程的。而编程语言对人工智能的设计有很大影响。

本书的大部分内容都集中在不针对游戏的技术上。例如，寻路或决策树，是可以用于一系列游戏的一般方法。这些技术需要被实施。十年前，假设这些技术作为核心游戏代码的一部分在C++中实现也不是没有道理的。现在情况已经发生了根本性的变化。尽管游戏引擎通常仍然用C和C++实现，但大多数人工智能通常用另一种语言实现。最常见的商业游戏引擎，Unity和虚幻引擎4（UE4），提供了一个用C++编写的寻路系统，但如果你需要任何其他的人工智能技术（例如行为树，或转向行为），就需要自己编写，或从第三方获得许可。在UE4中，这可能仍然涉及到用C++编码，但在Unity中，这可能意味着用C#实现。在这两种情况下，新的代码作为一种插件，与游戏代码而不是网络和3D渲染等核心设施处于同等地位。

与此同时，移动和在线开发也在蓬勃发展，其中实施语言受到目标平台的限制。iOS上的Swift（以及之前的Objective-C）、

Android上的Java（或其他JVM语言，如Kotlin），以及网络上的JavaScript。这些趋势共同意味着人工智能技术正在用更广泛的语言来实现。下一节将考虑不同语言对本书到目前为止所描述的技术的不同要求。

但是，通用的人工智能技术只是人工智能代码需要编程支持的一个地方。使用以嵌入式编程语言编写的小脚本来控制角色的行为是很常见的。这通常比使用通用技术和开发工具来正确配置它更容易。如果创建角色行为的人对简单的编程任务有相当的信心，那么用一系列的if语句来产生基于状态的行为，而不是在通用的状态机编辑器中拖动块和转换，可能会容易得多。

商业游戏引擎总是提供某种形式的脚本支持。如果你要从头开始开发你的工具集，你可能需要添加你自己的工具。幸运的是，有几种语言很容易集成。第13.2节考虑了这些选项。虽然越来越少，但许多开发者认为现有的选项并不适合，于是开发了自己的脚本语言。语言设计超出了本书的范围，但第13.3节对其中涉及的内容做了一个高度概括。

13.1 实施语言

当我在20世纪90年代开始进入这个行业时，游戏是用C语言编写的，几乎是全部。需要尽可能快地运行的部分代码往往是用汇编语言重新实现的，当开发者认为他们可以比编译器做得更好时（对于某些类型的代码，他们肯定可以）。本书第一版发布时，业界正在勉强地向C++迁移。我说勉强，是因为C++是一种更大的语言，对不小心的人来说有相当大的性能陷阱：对于C++是否会被证明有足够的性能，确实存在争议。这一时期有许多优秀的代码库，包括一些最畅销的作品的代码。例如，*Quake 3*的C++源码[124]（可在[25]上找到），经常被开发者引用为最漂亮、写得最好的C++代码库之一。

在过去的20年中，行业的两个重大变化改变了游戏的开发方式。

首先是移动端作为一个主要市场的增长。虽然用C++为iOS和Android开发移动游戏是可能的，但只有极少数的游戏开发者这样做。苹果鼓励iOS开发者使用Swift（以及在Swift之前的Objective-C），而安卓则依赖Java虚拟机，用Java或Kotlin等JVM语言进行编程更容易。

第二个重大转变是对游戏引擎的使用。大多数桌面和控制台的开发商（也越来越多地用于移动设备）现在使用现有的引擎。对于像EA这样的大工作室，这可能是他们自己的内部引擎。对于大多数开发者来说，这是Unity和虚幻引擎。C++仍然是用于构建这些游戏引擎的主要语言。而开发的游戏

在UE4中，他们的部分游戏玩法往往是用C++实现的。但是计算机的速度快了很多，很多原来用低级语言编程的东西现在可以用引擎的脚本语言来实现。例如，UE4中的蓝图，以及Unity中的C#。Unity在教育、业余爱好者、移动和独立开发社区中的使用爆炸性增长，意味着C#现在可以有理由声称是实现游戏的主要语言。因为难以衡量，所以很难说清楚，但至少这个建议并不荒唐。

本书中的大部分建议都是中性语言。在本节中，我将简要介绍一下在写作时对游戏实现最重要的五种语言：C++、C#、Swift（用于iOS）、Java（用于Android）和JavaScript（用于网页游戏和服务器）。我将对每种语言进行总结，并指出在本书中实现算法时需要考虑其特殊性的任何方式。

13.1.1 C++

C++最初是对C语言的一个扩展，以增加对面向对象编程的支持。它仍然主要是一个兼容的超集，尽管新的功能有时会在两种语言的不同时期出现。在过去的几年里，Rust将自己定位为编写低级系统代码的新语言（构成操作系统或其他资源和性能关键应用程序的那种软件），但除了一些著名的Rust项目外，C++仍然是低级编程的最流行的选择。

C++是一种大语言。自1985年首次发布以来，它一直在稳步增长。现在每隔三年就会发布新的标准版本，每个版本都有自己的一套新功能，而且往往是新的语法。至少在过去的15年中，这种臃肿的功能集一直是对C++的批评。以至于它实际上已经分裂成了多种语言。不是像Scheme和Lisp那样，因为有多种实现方式，而是因为每个采用它的公司都倾向于通过采用整个语言的一个子集来抑制其复杂性。但很少是同一个子集。诸如它的模板语言、对运算符重载的全面支持、类的多重继承、运行时类型信息以及它对异常的处理都经常被禁止。

除了核心语言之外，C++还规定了一个标准模板库（STL）。这是用C++模板实现的一组基本数据类型。在其他语言中属于核心的基本数据类型，如可调整大小的数组、哈希映射、堆栈和队列，都是通过STL提供的。不幸的是，STL的实现方式各不相同，作为一个整体，它在游戏开发者中以难以管理、性能低下或内存使用不稳定而闻名。我曾为不止一家公司工作过，他们在代码中禁止使用STL，宁愿为那些基本数据类型创建自己的非模板化版本。如果像Boost这样的通用库可以用来 "纠正 "核心

语言中的一些问题，情况就更复杂了。随着C++的发展，它从Boost中吸取了一些好的想法，以略微不同的方式实现它们，并有可能给出另一个不兼容的轴。

尽管我尽量不在本书中用C++的具体术语描述任何算法，但它是我最熟悉的实现游戏的语言。因此，根据你或你的团队想要使用的功能子集，伪代码应该自然地转化为C++。提供高质量的数据结构是我的主要假设。根据我的经验，我在使用STL方面取得了合理的成功（特别是在用自定义分配器取代其默认的内存管理后），但如果你选择实现你自己的数据结构，这可能是比使用这些数据结构来实现AI更大的挑战。

13.1.2 C#

C#是由微软创建的，作为Java的竞争对手，目标是自己的通用语言运行时（CLR）。该语言的最初版本相当类似于Java，但随着时间的推移，它更愿意改变核心，而这两种语言也逐渐分开。

因为通用语言的运行时间是微软专有的，所以其他开发者实施了一个开源的替代方案，叫做Mono。因为微软开放了C#语言的源代码，只保留了运行时的专利，Mono提供了一种在另一个项目中使用语言的免费方式。Unity引擎正是这样做的。

在理论上，这允许任何CLR语言与Unity引擎一起使用。然而，Unity公司只支持C#，因为它已经废弃了自己的类似JavaScript的脚本语言。我知道有开发者使用F#，一种功能性的CLR语言，但大多是实验性的。现在绝大多数为Unity编写的代码都是C#。

对Unity引擎的一个常见的批评是，它远远落后于当前的C#版本。部分原因是Mono落后于微软的CLR，但主要是因为Unity在升级其引擎中使用的Mono版本方面非常保守。这可能会影响到你作为一个开发者，因为C#程序员的在线教程和资源往往与引擎中的语言版本不兼容。幸运的是，大量针对Unity的教程和资源在一定程度上缓解了这一问题，尽管这些教程和资源通常是针对新手程序员的。

也许C#和C++之间最有影响的区别是Mono管理内存的方式。Mono不是显式分配和取消分配，而是跟踪你的程序所使用的内存，并定期释放所有它能找到的不再需要的内存。不幸的是，寻找内存来取消分配（被称为垃圾收集）需要时间，有时需要几十毫秒。更糟糕的是，"定期 "是很难反击的，而且无法控制。依靠这种行为的游戏会受到偶尔的减速、帧率下降或卡顿的影响。

幸运的是，如果没有东西需要收集，垃圾收集通常是非常快的（有一些边缘情

况，需要花时间来证明没有东西需要收集，但对于正常的使用模式，这些情况通常是罕见的）。垃圾收集的工作方式是记录哪些对象知道哪些其他对象的存在。从一组已知的对象开始，它通过这个图，标记所有可以到达的对象，可以一步到位，也可以多步到位。

如果在这个过程结束时，一个对象没有被标记，那么没有任何东西可以访问它，它可以自由地被收集。这种算法被称为 "标记和扫描"，或跟踪垃圾收集。

如果你的代码一次性分配所有需要的对象（通常是在加载关卡时），然后在游戏结束时（例如在加载下一关卡之前）一次性释放所有对象，就可以避免垃圾回收的停顿。在C#中没有办法显式地分配或释放内存，所以我们围绕着标记和清扫算法进行工作。在关卡开始时，我们创建所有我们可能需要的对象，并保持对它们的引用（如果它们是类似的对象，我们可以把它们放在一个数组中，例如）。这就保证了它们总是可以到达的，因此不会被垃圾回收。在算法的执行过程中，当需要一个对象时，它将从我们预先分配的块中取出。在关卡结束时，对所有对象的引用都会被删除，无论我们最后是否使用它们。然后，垃圾收集器将检测到它们已经准备好被收集。Mono的垃圾收集的速度问题非常严重，这种方法实际上是必不可少的。它对本书介绍的任何算法都没有重大影响，但对于任何C#的实现都需要牢记。

最近，Mono实现了一个垃圾收集系统，它更不可能导致掉帧和卡顿。截至发稿时，这在Unity中作为一个选项可用，但默认情况下没有启用。当它成为默认的垃圾收集系统时，对对象池的需求将有所减少。但是，垃圾收集将始终是一个耗时的、间歇性的过程。对于像A*这样的算法，对象的数量足够大，所以某种形式的预分配总是可取的。

13.1.3 SWIFT

Swift是苹果公司在macOS和iOS上实现应用程序的语言。虽然它是一种编译语言，理论上任何编译语言如C++都可以使用，但操作系统的API在Swift中是暴露的，而且苹果的Xcode开发工具也提供了全面的支持。除非你使用Unity这样的集成游戏引擎，否则在iOS上使用另一种语言是一个艰难的挑战。

Swift取代了苹果对Objective-C语言的旧有支持。Objective-C和C++一样，是C语言的扩展，支持面向对象的编程。然而，这两种语言在设想对象的方式上有很大的不同，使得这两种语言之间很难互操作。Objective-C从未像C++那样被广泛采用，而且该语言的使用逐渐减少。当它被Swift取代时，苹果设备的编程是Objective-C唯一值得注意的使用案例。起初，苹果公司将Swift作为没有C语言的Objective-C语言来介绍，这主要是一种营销策略，以减轻开发者对学习新东西的担忧，实际上这两

种语言是非常不同的。

Swift编译为特定硬件的机器代码，而不像C#和Java那样编译为运行在虚拟机上的字节码。但与Java和C#一样，Swift代码在执行时有一个全面的运行时，处理与操作系统的接口，以及一些设施，如

内存管理和垃圾收集。在实践中，区别不大，swift感觉更像Java或C#而不是C或C++。

Swift使用的垃圾收集方法与Mono不同。在撰写本文时，它使用了自动引用计数（ARC）。在这种方法中，每个对象都存储了一个计数：重新指的是代码中知道该对象的地方的数量。一个被设置为指向该对象的变量会增加这个计数。将对象放在一个数组中也会增加它。或者把它存储在另一个对象的成员变量中。如果该变量超出了范围，计数就会被递减。如果数组被删除或者包含的对象被删除，也会发生同样的情况。如果计数达到零，该对象就不能被代码的任何部分到达，所以删除它是安全的。有一些引用计数系统就是这么简单，但是简单的方法可能会泄漏内存：从不收集一些单独被引用但作为一个群体永远无法到达的对象的未使用周期。例如，如果对象A是唯一知道对象B的东西，而对象B是唯一知道对象A的东西，那么A和B都可以被垃圾收集，但它们的引用计数都是1。这些循环可以变得 ，涉及许多对象。正是这些复杂的循环导致了垃圾收集的缓慢。这两种方法（Swift的引用计数和Mono的标记和扫描）的权衡很复杂，远远超出了本章的讨论范围。在实践中，Swift的方法似乎不太可能让游戏代码暂停或卡顿。但较少并不意味着永远不会。，避免不必要的垃圾收集仍然是值得的。

要做到这一点，我们可以使用与我们在C#中看到的完全相同的方法。我们可以预先分配我们需要的所有对象，并在关卡结束时释放我们对它们的最后一个引用。至于升级后的Mono垃圾收集系统，Swift的垃圾收集器通常运行得足够快，除非你正在进行许多分配，否则问题不会被注意到。但问题往往会在游戏运行一段时间后，毫无征兆地出现。谨慎行事，在你的人工智能代码中预先分配对象是可能的，甚至是最好的做法，无论是否严格必要。

13.1.4 JAVA

在写这篇文章的时候，Java可能是世界上最流行的编程语言，1而且早在2001年就已经如此了（偶尔有来自C的短暂挑战）。它被设计成一种通用语言，对其字节码和虚拟机（被称为Java虚拟机，或JVM）有详细的规范。通过锁定虚拟机，该语言旨在允许源代码在一台机器上被编译，而产生的字节码能够在相同或任何其他机器上运行。它的营销口号是 "一次编写，随处运行"。总的来说，它成功地实现了这个目标

。在两台非常不同的机器上的Java运行时将不得不与两个非常不同的底层进行通信。

1. 编程语言的流行程度是由TIOBE公司衡量的，并已在其网站上公布。[69]自2001年以来（尽管它有远至1988年的部分回顾性数据）。该指数被批评为仅仅依赖搜索引擎和网站，其方法的细节当然也可以争论，但总的来说，它已被证明是抽查该行业脉搏的有用方法。

操作系统，这可能会导致行为上的细微差异，但Java在试图平滑这些差异方面做得很好。

正是这种一次性编译并在任何地方运行的能力，使得它对移动应用的开发特别有吸引力。现代智能手机的硬件种类繁多，如果要获得正确的编译器定义，并为每一种硬件编译你的游戏，将是非常麻烦的。硬件的特殊性并不能完全消除在多种设备上进行测试的需要，但如果使用一种可以编译成机器码的语言，如C++，情况就会大为恶化。当谷歌开发其移动操作系统安卓时，由于这个原因，它选择了Java作为主要开发语言。

作为一种编程语言，它有一个相当冗长的声誉。而大部分的冗长都是以模板代码的形式出现的：类似的模式，功能相对较少，但需要以同样的方式反复编写。编辑器和其他工具通过自动生成其中的一部分，使之变得更容易一些，但它仍然会使代码变得臃肿，需要维护。

Java也以变化缓慢而闻名。从某种程度上说，这是一个好处，意味着写得好的代码可以在很长一段时间内保持习惯性。但这也意味着重要的生产力改进和来自其他语言的好主意被缓慢地采用。类型安全的数据结构被要求了十多年才最终被引入，尽管其实现被广泛批评，并且最近被证明是不健全的[3]（这个问题可能会延伸到本节中的一些其他语言，在JVM上运行）。

Java的大部分优点都是由JVM提供的。正是JVM被设计用来在任何平台上运行相同的编译代码。当编译后的字节码运行时，虚拟机不知道也不关心该代码是如何生成的。在90年代末，我曾在一家非游戏公司短暂工作过，其产品通过与用户的简单对话产生字节码。它更常用于支持其他语言，如功能性的Scala、Lisp方言Clojure、脚本语言Groovy和其他几十种语言，都可以编译成有效的JVM字节码。因为Java提供了一个非常大的标准库，也被编译成字节码，这些设施可以在任何JVM语言中使用。同样地，Android以一种可以从任何语言中调用的方式公开其接口。

最近谷歌认可了Kotlin，这是一种由JetBrains开发的JVM语言，具有与Java类似的功能和精神，但它的风格更加现代。看来，在未来几年，越来越多的安卓应用开发可能会转向新的语言。与此同时，移动游戏开发者也越来越多地使用多平台游戏引擎。哪种趋势会发展得更快还很难说。很可能Kotlin永远不会成为游戏行业的重要参与者。

就实施建议而言，为JVM编写程序与为Mono编写程序非常相似。JVM使用一个标记和清扫的垃圾收集器，可以通过调用System.gc()手动请求。和Mono一样，垃圾收集器的实现也不是电子化的。

当许多小对象被分配的时候，这是很有效的。明智的做法是使用相同的分配

上一节详述的汇集方法。

13.1.5 JAVASCRIPT

JavaScript是作为网页的一种脚本语言而创建的。它在游戏开发中被用于两个目的：实现打算在网络上玩的游戏，以及使用Node为在线多人游戏建立服务器。

像所有的脚本语言一样，用JavaScript编写的性能关键代码有时会运行缓慢。因此，看到用JavaScript编写的复杂算法是不寻常的。在JavaScript中建立一个状态机或行为树是很有可能的，但是规划、寻路、最小化或学习可能会过于耗费资源。在服务器上，这些性能关键的算法可以用C++实现，并链接到JavaScript运行时中。在浏览器中，类似的东西最初是可以实现的，但现在已经被浏览器供应商出于安全考虑而删除了。

在游戏开发中，JavaScript作为一种脚本语言被更广泛地使用，被嵌入到游戏或游戏引擎中。下面的第13.2节将更深入地描述它作为脚本语言的用途。

从语言的角度来看，JavaScript的显著特点是使用了原型继承。原型继承与我们熟悉的基于类的继承不同，比如C++和Python语言。最近版本的JavaScript增加了一个类的关键字，以使
它更容易被其他语言的程序员所接受，但类仍然是在
使用原型，在引擎盖上。

原型与类的不同之处在于，任何对象都可以继承任何其他对象。在基于类的语言中，对象有两种类型：类和实例。实例只能从类中继承，而类之间的继承形式有限，通常称为子类。这在原型语言中要简单得多。只有继承，任何对象都可以从任何其他对象继承。这样做的实际结果是，我们不再局限于类和实例的两级层次结构。在第5.4节讨论行为树时，我描述了在定义和实例化人工智能时对三个层次的普遍需求。这很好地映射到了JavaScript中。它的效果是允许开发者为一个广泛的角色类别创建一个根对象，然后一个对象可以继承这个根对象来配置特定角色类型的设置（这个中间阶段可以由关卡设计师或技术艺术家完成），然后在最后一步可以看到一个对象为关卡中的单个角色实例化角色类型。

与本节中的其他语言不同，JavaScript是单线程的。作为一种用于增强网页的语言，它被设计为事件驱动：JavaScript运行时监控特定的事件（例如用户输入、网络活动或预定的超时）；当事件发生时，运行时调用已经注册为感兴趣的代码；然后这些代码按顺序运行，只要它需要；当没有更多的代码要运行时，控制权返回到运

行时，它等待直到另一个事件发生。代码在运行时保证不会被打断，同时也不会有其他代码被运行。这对于浏览器中的简单脚本是完美的，但有些事情需要更长的时间来完成。我们不希望整个JavaScript过程因等待结果而停滞不前。在网络浏览器中，这可能是在网络上查询数据，在某些情况下，这可能是以秒来计算的。

JavaScript为此使用了回调（在后来的JavaScript版本中，这些回调被包裹在更容易使用的结构中，如承诺或 "async"，尽管其基本行为是相同的）。回调使用相同的事件过程。你注册一些代码，当一个动作完成时（例如，当收到来自服务器的结果时）被调用，然后启动该动作。这允许许多回调在任何时候都在等待数据。这对许多服务器应用来说是完美的，在这些应用中，代码需要等待来自数据库的数据，来自其他服务的数据，或者等待文件被读取。

不幸的是，它不太适合那些长期运行的任务涉及大量计算的情况。而许多游戏就属于这一类。如果JavaScript引擎正在进行计算，它就不能继续将事件分配给代码的其他部分。我们不能轻易地要求一个JavaScript例程来执行寻路，并在寻路完成后让我们知道。明显的实现是冻结，直到寻路完成。

有两种解决方案。可以编写代码来执行 "合作多任务"：任何长期进程都会做一点工作，然后自愿将控制权返回给运行时，期望它很快会被再次调用，做更多的工作。然后，JavaScript可以利用这些间歇期来调用代码的其他部分。这很有效，但需要你实现算法来支持它。我们在第10章中看到了为什么我们可能想在任何语言中这样做：允许长期运行的算法在多个渲染帧中进行分割。在JavaScript中，它从 "好的 "到 "必须的"，实际上是必不可少的。

第二个解决方案是使用多个JavaScript解释器，并编写通信代码以保持它们的同步。我们不调用寻路函数，而是将一些消息传递给另一个能够寻路的进程。那个进程在完成后会传回一个消息。这种方法通过Web-Workers API在浏览器中使用。同样，它是有效的，但需要比合作多任务更多的协调代码，而且由于不同的进程不能改变彼此的数据，在消息中来回传递数据通常有很大的开销（通常是通过转换文本JSON格式）。

13.2 编写的人工智能

游戏中相当一部分决策没有使用本书中描述的技术。在20世纪90年代早期和中期，大多数人工智能都是使用自定义编写的代码进行硬编码来进行决策。这很快速，而且对于小的开发团队来说，当程序员也可能在设计游戏时，这很好用。对于开发需求不大的平台，以及程序员也负责实现游戏性的独立团队，它仍然是主流模式。

随着生产变得越来越复杂，开发工作变得越来越小众。拥有数百名员工的大型游戏往往将开发任务划分为小的受限角色。在工作室里，有必要将内容（行为设计）与引擎分开。关卡设计师可能被要求设计角色的广泛行为，而不被视为

有权编辑他们的代码。为了促进这一点，许多工作室使用一般的技术，和自定义编辑器，如前一章所述。

一个中间的解决方案是对一组技术进行编程，让技术熟练的关卡设计者使用一种简单的编程语言将它们结合起来，与主游戏代码分开。这些通常被称为脚本。脚本可以被视为数据文件，如果脚本语言足够简单，关卡设计师或技术艺术家就可以创造出这些行为。

脚本语言支持的一个偶然的副作用是玩家能够创造自己的角色行为并扩展游戏。修改是PC游戏的一个重要的财政力量（它可以将其全价保质期延长到其他游戏典型的8周以上），以至于许多三A级游戏都包含某种脚本系统（通常是作为其使用的引擎的一部分）。

然而，也有一些负面因素。我仍然不相信在一个项目中广泛使用脚本是特别可扩展的，也不相信以这种方式产生复杂的行为是容易的。根据我的经验，开始使用脚本很容易，但最终会变得难以扩展和调试。

除了缩放问题，这种方法还错过了既定人工智能技术的要点。它们的存在是因为它们是行为问题的优雅解决方案，而不是因为用C++编程不方便。即使你使用脚本语言，你也必须考虑到角色脚本中使用的算法。在脚本中编写*临时*代码和用C++编写一样困难，而且几乎可以肯定缺乏同样成熟的调试工具。

我认识的几个开发者都落入了这个陷阱，他们认为脚本语言意味着他们不需要考虑字符的实现方式。即使你使用的是脚本语言，我也建议你考虑一下你在这些脚本中使用的架构和算法。可能是脚本可以实现本书中的其他技术之一，也可能是单独的专用实现更实用：与脚本语言并列或取代脚本语言。

但是，抛开所有这些限制，毫无疑问，脚本有几个重要的应用：在实现非复杂的触发器和游戏级别的行为（例如，哪个键可以打开哪扇门），将游戏机制迭代为有趣的游戏，以及快速制作角色的人工智能原型。

本节提供了一个简短的入门知识，以支持足够强大的脚本语言在你的游戏中运行AI。它是有意的浅显，旨在给你足够的信息，让你要么开始，要么决定这不值得努力。有几个很好的网站对现有的语言进行了比较，还有一些文章涉及从头开始实现你自己的语言。

13.2.1 什么是脚本式人工智能?

脚本式人工智能 "这个短语有点含糊不清。在本章中，它指的是这些用脚本语言编写的、控制角色行为的手工编码程序。

然而，在游戏评论和营销材料中，它经常被用作一种侮辱，指的是无论在什么情况下都做同样事情的人工智能，其方式看起来毫无智慧。一个总是试图遵循相同的巡逻路线的角色，即使该路线被封锁，也可能被说成是有脚本的人工智能，即使该路线是由一个非技术设计师在关卡编辑器中生成的。这个词的第三种用法不那么具有贬义：场景可以被描述为脚本化的人工智能。一个健康状况不佳的角色滚入掩体，竖起路障，开始治疗，可以说是有脚本的人工智能。即使这个序列是以行为树的形式实现的。为了本章的目的，我将坚持使用第一个定义：脚本化人工智能是用脚本语言编写的人工智能。

13.2.2 什么是好的脚本语言？

一个游戏对其脚本语言总是有一些要求的。语言的选择常常归结为对这些问题的权衡。

速度

游戏的脚本语言需要尽可能快地运行。如果你打算在游戏关卡中使用大量的角色行为和事件的脚本，那么这些脚本就需要作为游戏主循环的一部分来执行。这意味着，运行缓慢的脚本会占用你渲染场景、运行物理引擎或准备音频所需的时间。

大多数语言可以是随时的算法，在多个框架内运行（详见第10章）。这在一定程度上减轻了速度的压力，但它不能完全解决这个问题。

汇编和解释

脚本语言大致分为解释型、字节编译型和完全编译型，尽管每种技术都有很多不同的风格。

解释的语言是作为文本被接收的。解释器查看每一行，找出它的意思，并执行它所指定的行动。

字节编译的语言从文本转换为一种内部格式，称为*字节码*。这种字节码通常比文本格式要紧凑得多。因为字节码的格式是为执行而优化的，所以它可以运行得更

快。

字节编译的语言需要一个编译步骤；它们需要更长的时间来启动，但随后运行得更快。更昂贵的编译步骤可以在关卡加载时进行，但通常是在游戏发行前进行。

最常见的游戏脚本语言都是字节编译的。有些语言，如Lua，提供了分离编译器的能力，不与最终游戏一起分发。这样一来，所有的脚本都可以在游戏进入母版之前被编译，而只有编译过的版本才需

的游戏中包含。然而，这就取消了用户编写自己的脚本的能力。

完全编译的语言会产生机器代码。这通常必须被链接到主要的游戏代码中，这就失去了拥有独立脚本语言的意义。不过，我确实知道有一个开发者有一个非常整洁的运行时链接系统，可以在运行时从脚本中编译和链接机器代码。然而，总的来说，这种方法出现大量问题的范围是巨大的。我建议你还是省省吧，去找一些更经得起考验的东西。

可扩展性和集成性

你的脚本语言需要能够访问你游戏中的重要功能。例如，一个控制角色的脚本需要能够询问游戏，找出它能看到的东西，然后让游戏知道它要做的结果。

在实现或选择脚本语言时，它所需要访问的函数集很少被知道。在游戏主代码中，拥有一种能够轻松调用函数或使用类的语言是很重要的。同样，当脚本作者提出要求时，程序员能够很容易地公开新的函数或类，这一点也很重要。

有些语言（Lua是最好的例子）在脚本和程序的其他部分之间设置了一个非常薄的层。这使得从脚本中操作游戏数据变得非常容易，而不需要有一整套复杂的翻译。

再入性

脚本的可重入性通常很有用。它们可以运行一段时间，当它们的时间预算用完后，就可以搁置起来。当一个脚本下次有时间运行时，它可以从它离开的地方继续运行。

让脚本在达到自然停顿的时候放弃控制，往往是有帮助的。然后调度算法可以给它更多的时间，如果它有时间的话，否则就继续前进。例如，一个控制角色的脚本可能有五个不同的阶段（检查情况、检查健康状况、决定运动、计划路线和执行运动）。这些都可以放在一个脚本中，在每个阶段之间产生。然后每一个都会每隔五帧运行一次，人工智能的负担就被分配了。

不是所有的脚本都应该被中断和恢复。一个监视快速变化的游戏事件的脚本可能需要在每一帧都从头开始运行（否则，它可能在不正确的信息上工作）。更为复杂的重入应该允许脚本编写者将一些部分标记为不可中断。

这些微妙之处不存在于大多数现成的语言中，但如果你决定编写自己的语言，这可能是一个巨大的福音。

13.2.3 融合

嵌入与可扩展性有关。嵌入语言被设计成可以并入另一个程序。当你从你的工作站运行脚本语言时，你通常不会运行一个专门的程序来解释源代码文件。在游戏中，脚本系统需要从主程序中控制。游戏决定哪些脚本需要运行，并且应该能够告诉脚本语言来处理它们。

13.2.4 选择一种开放源码语言

有大量的脚本语言可供选择，其中许多是根据适合于游戏的许可证发布的。通常情况下，某种变化的开放源码。

虽然在游戏行业的商业脚本语言方面已经有了各种尝试，但专有语言很难与大量高质量的开放源码替代品竞争。

开源软件是在一个许可方下发布的，该许可方给予用户将其纳入自己的软件的权利，而无需支付费用。一些开源许可证要求用户将新创建的产品重新租借为开源产品。这些显然不适合商业游戏。

开源软件，正如其名称所示，也允许查看和更改源代码。这使得它很容易吸引开发者为代码库贡献错误修复和改进。作为一个开发者，它提供了信心，知道问题可以被解决或被修改。例如，我为一家公司提供咨询，委托Lua提交者之一实现一个自定义的语法扩展，仅仅几天的工作就极大地改善了该语言在他们项目中的使用。

然而，修改并不总是可取的。一些开源许可证（特别是通用公共许可证，GPL），甚至那些允许你在商业产品中使用该语言的许可证，都要求你发布对该语言本身的任何修改，或者在最坏的情况下，你与该库链接的任何代码。这对你的项目来说几乎肯定是一个问题，除非你打算让你的游戏开放源代码。

无论一种脚本语言是否是开源的，在你的项目中使用这种语言都会有法律上的影响。在你打算发布的产品中使用任何外部技术之前，你应该咨询知识产权律师。本书无法就使用第三方语言的法律意义向你提供正确的建议。下面的评论是为了说明可能引起关注的各类事情。还有很多其他的。

由于没有人向你出售开放源码软件，如果软件出了问题，没有人负责。如果在开发过程中出现了难以发现的错误，这可能是一个小麻烦。然而，如果你的软件导致你

的客户的个人电脑清除了它的硬盘，这可能是一个重大的法律问题。对于大多数开源软件，你要对产品的行为负责。这一点的反面是，一个成熟的开放源码语言可能已经

已经被其他开发者使用了数百万次，因此不太可能重新发现病态的错误。

当你从一家公司获得技术许可时，该公司通常作为一个保险层，防止你因违反版权或违反专利而被起诉。例如，一个开发新技术并获得专利的再搜索者对其商业化拥有权利。如果同样的技术在未经研究者许可的情况下被应用到软件中，他们可能有理由采取法律行动。当你从一个公司购买软件时，它对软件的内容负责。因此，如果研究人员找你麻烦，卖给你软件的公司通常要对违规行为负责（这取决于你签署的合同）。

当你使用开源软件时，没有人把软件授权给你，而且因为不是你写的，你不知道它的一部分是否被盗用或复制了。除非你非常小心，否则你将不知道它是否破坏了任何专利或其他知识产权。其结果是，你可能要对侵权行为负责。

你需要确保你了解使用 "免费 "软件的法律含义。它并不总是最便宜或最好的选择，即使前期费用很低。在你做出承诺之前，请咨询律师。

这类法律问题在过去曾促使开发者创建自己的语言。除了商业游戏引擎的定制脚本（如UE4的 "蓝图 "视觉语言），现在这种情况已经很少了。

13.2.5 语言选择

每个人都有自己喜欢的语言，而试图挑选最好的单一预制脚本语言是不可能的。阅读任何一个编程语言新闻组，都会发现无休止的 "我的语言比你的好 "的火焰战争。当然，有优点也有缺点，但没有一个比你的编程团队对某种特定的语言或语法感到舒服更有力。

在几乎无限的选择中，有少数几种语言被反复使用。当选择一种语言与你的游戏整合时，值得注意的是哪些语言是通常的嫌疑人，以及它们的优点和缺点是什么。请记住，通常有可能对现有的语言进行黑客攻击、重组或重写，以克服其明显的缺陷。许多（也许甚至是大多数）使用脚本语言的商业游戏开发者都会以小规模的方式来做这件事；有些人最终得到的语言几乎与他们开始的地方无法辨认。下面描述的语言是以其开箱即用的形式讨论的。

我将按照我个人对一个新项目的考虑顺序来看待四种语言：Lua, Scheme, JavaScript, 和Python。

吕阿

Lua是一种简单的程序性语言，从一开始就作为一种嵌入语言来构建。该语言的设计是出于可扩展性的考虑。与大多数嵌入式语言不同，这

并不限于在C或C++中添加新的函数或数据类型。Lua语言的工作方式也可以进行调整。

Lua有少量的核心库，提供基本的功能。然而，其相对无特征的核心是其吸引力的一部分。在游戏中，除了数学和逻辑，你不太可能需要库来处理任何东西。这个小的核心很容易学习，而且非常灵活。

Lua不支持可重入的函数。整个解释器（严格来说是 "状态 "对象，它封装了解释器的状态）是一个C++对象，完全是重入的。使用多个状态对象可以提供一些重入支持，代价是内存和它们之间缺乏通信。

Lua有 "事件 "和 "标签 "的概念。事件发生在脚本执行过程中的某些点上：例如，当两个值相加时，当一个函数被调用时，当一个哈希表被查询时，或当垃圾收集器被运行时。C++或Lua中的例程可以针对这些事件进行注册。这些 "标签 "例程在事件发生时被调用，允许改变Lua的默认行为。这种深层次的行为修改使得Lua成为你能找到的最可调整的语言之一。

事件和标签机制被用来提供基本的面向对象的支持（Lua并不是严格意义上的面向对象，但你可以调整它的行为以尽可能接近它），但它也可以用来将复杂的C++类型暴露给Lua，或用于简洁地实现内存管理。

另一个受到C++程序员喜爱的Lua特性是 "userdata "数据类型。Lua支持常见的数据类型，如浮点数、ints和字符串。此外，它还支持一个通用的 "userdata "和一个相关的子类型（"标签"）。在默认情况下，Lua不知道如何做
用户数据的任何东西，但通过使用标记方法，可以添加任何想要的行为。Userdata通常被用来保存C++实例指针。这种对指针的本地处理可能会引起问题，但通常意味着需要更少的接口代码来使Lua与游戏对象一起工作。

对于一种脚本语言来说，Lua是处于快速的一端。它有一个非常简单的执行模型，在峰值时速度很快。再加上调用C或C++函数的能力，不需要大量的接口代码，这意味着现实世界的性能是令人印象深刻的。

Lua的语法对C和Pascal程序员来说是可识别的。对于艺术家和关卡设计师来说，它不是最容易学习的语言，但其相对缺乏的语法特征意味着对于热衷于此的员工来说是可以实现的。

尽管它的文档比这里的其他两种主要语言要少，但Lua是游戏中使用最广泛的预

制脚本语言。Lucas Arts将其内部的SCUMM语言高调地转换为Lua语言，促使大量的开发者研究其功能。Unity和UE4都没有使用Lua，但开源引擎Godot在专注于自己的定制语言之前曾短暂地支持过它。

要想了解更多，最好的信息来源是Lua书《*Lua中的编程*》[26]，它也可以在网上免费获得。

方案和变体

Scheme是一种源自LISP的脚本语言，LISP是一种古老的语言，在20世纪90年代之前，大多数经典的人工智能系统都是用它来构建的（之后也有很多，但没有同样的统治力）。关于Scheme，首先要注意的是它的语法。对于不习惯LISP的程序员来说、计划可能难以理解。

方括号包围了函数调用（几乎所有的东西都是函数调用）和所有其他代码块。这意味着它们可以变得非常嵌套。良好的代码缩进是有帮助的，但是一个能够检查括号的编辑器是严肃的开发所必须的。对于每一组括号，第一个元素定义了该块的作用；它可能是一个算术函数：

```
(+ a 0.5)
```

或流量控制语句：

```
(如果 (> a 1.0) (设置! a 1.0))
```

这对计算机来说很容易理解，但却与我们的自然语言相悖。非程序员和那些习惯于类似C语言的人可能会发现用Scheme思考有一 段 时 间 很困难。

与本节中的其他语言不同，Scheap有成百上千的版本，更不用说其他适合作为嵌入式语言的LISP变体了。每个变体都有自己的权衡，因此很难对速度或内存的使用作出概括。然而，在它们的最佳状态下（我想到的是minischeme和tinyscheme），它们可以非常非常小（minischeme的完整系统的C代码不到2500行，尽管它缺乏完整方案实现中的一些更奇特的功能），而且非常容易调整。最快的实现可以和其他脚本语言一样快，而且编译效率通常比其他语言高得多（因为LISP的语法最初是为了便于解析而设计的）。

然而，Scheme的真正亮点在于其灵活性。在语言中，代码和数据之间没有区别，这使得在Scheme中传递脚本，修改它们，然后再执行它们变得很容易。大多数使用本书技术的知名人工智能项目最初都是用LISP编写的，这并不是巧合。

我经常使用Scheme，这足以让我看清它笨拙的语法（我们中的许多人在读人工智能本科时不得不学习LISP--它被认为是21世纪初之前的主要人工智能语言）。在职业上，我从未在游戏中使用过未经修改的Scheme（尽管我知道至少有一个工作室使用过），但我在Scheme基础上构建的语言比任何其他语言都多（迄今为止有7种）。如

果你打算推出自己的语言，我强烈建议你先学习Scheme，并阅读一些简单的实现。这可能会让你大开眼界，知道一种语言的创建是多么容易。

脚本

JavaScript是一种为网页设计的脚本语言。尽管名字里有Java，但它与这种语言没有什么联系。这个名字是网景公司（曾经的浏览器制造商）和Sun Microsys- tems公司（当时的Java所有者）之间的一个早已不存在的商业协议的奇特之处。严格来说，该语言被称为ECMAScript，通常缩写为ES（特别是带有表明规范版本的后缀，如ES6），尽管前一个名字已经坚持了下来，并且不太可能被取代。

并没有一个标准的JavaScript实现。许多游戏或游戏引擎中的JavaScript实现更接近于受到JavaScript的启发，而不是实现语言本身。UnityScript，来自Unity游戏引擎的被废弃的脚本语言，就是这样一个例子。这些准JavaScripts可能使用相同的语法（相对来说类似C语言），但在某些情况下甚至不支持原型继承。

在其余的JavaScript用户中，大多数是打算在浏览器中运行的游戏。在这种情况下，没有嵌入脚本语言。脚本只是被传递到浏览器中运行。

最后，为Chrome浏览器创建的JavaScript的V8实现被广泛嵌入。它是Node JavaScript系统的核心实现，一些完整的游戏就是在它的基础上建立起来的（而且它被用来开发更多游戏的服务器端）。它也可以相当简单地嵌入到现有的引擎中。对进出数据的支持，以及对向JavaScript暴露底层函数的支持，都很简单，而且有很好的文档。V8也是完全可重入的，并且支持同一代码中的多个隔离解释器（因此不同的角色可以同时运行他们的脚本）。

JavaScript是一种强大的嵌入语言。但它最重要的优势也许是它的熟悉性。作为网络语言，它被许多程序员所熟知。它有大量的文献资料和大量的教程（尽管必须指出，有些教程是由新手自己写的，包含了一些不好的建议）。然而，由于其类似于C语言的语法，它可能会让非程序员望而生畏，而Lua或Python等更自然和充满关键词的语法则可以避免这种情况。

蟒蛇

Python是一种易于学习的、面向对象的脚本语言，具有出色的可扩展性和嵌入支持。它对混合语言编程提供了很好的支持，包括从Python透明地调用C和C++的能力。从2.2版本开始，Python支持重入函数，作为核心语言的一部分（称为*生成器*）。

Python 有大量可用的库，并且有非常庞大的用户群。Python 用户以乐于助人著称，comp.lang.python 新闻组是故障排除和建议的绝佳来源。

Python的主要缺点是速度和体积。尽管在过去的几年中，执行速度有了很大的进步，但它仍然很慢。Python依靠的是

哈希表查询（通过字符串）用于许多基本操作（函数调用、变量访问、面向对象编程）。这增加了很多开销。

虽然良好的编程实践可以缓解大部分的速度问题，但Python也以大而闻名。因为它比Lua有更多的功能，所以当链接到游戏的可执行文件中时，它的体积更大。

Python 2.X和进一步的Python 2.3版本为该语言增加了大量的功能。每一个额外的版本都实现了Python作为软件工程工具的更多承诺，但同样也使得它作为游戏的嵌入式语言的吸引力下降。早期版本的Python在这方面要好得多，使用Python的开发者通常更喜欢以前的版本。

对于C或C++程序员来说，Python常常显得很奇怪，因为它使用缩进来分组语句，就像本书的伪代码一样。

这个特点同样使非程序员更容易学习，他们没有括号可以忘记，也没有经历过不缩进代码的正常学习阶段。

Python以其非常易读的语言而闻名。即使是相对陌生的程序员也能很快看出一个脚本的作用。最近对Python语法的补充极大地损害了这一声誉，但它似乎仍然在某种程度上高于其竞争对手。

在我所使用的脚本语言中，Python是最容易让初级签名者和艺术家学会的。在以前的一个项目中，我想利用这一优势，但由于规模和速度问题，嵌入该语言是不现实的。解决方案是创建一种新的语言（见下面的章节），但使用Python语法。

其他选择

还有一大堆其他可能的语言。根据我的经验，每一种语言要么完全没有在游戏中使用（据我们所知），要么有明显的弱点，使得它比竞争对手更难选择。据我所知，本节中的所有语言都没有作为游戏中的脚本工具进行过商业使用。然而，像往常一样，一个对某种特定语言有特殊偏见和热情的团队可以绕过这些限制，得到一个可用的结果。

Tcl

Tcl是一种非常好用的可嵌入语言。它被设计成一种集成语言，连接用不同语言编写的多个系统。Tcl是工具控制语言的缩写。Tcl的大部分处理是基于字符串的，这可能

会使执行速度非常慢。另一个主要的缺点是它奇异的语法，这需要一些时间来适应，而且与Scheme不同的是，它最终并没有承诺提供额外的功能。语法上的不一致（如作为参数的传值或传名）对休闲学习者来说是比较严重的缺陷。

爪哇

Java在许多编程领域无处不在。由于它是一种编译语言，因此它作为一种脚本语言的使用受到限制。然而，同样的道理，它也可以很快速。使用JIT编译（字节码在执行前被转化为本地机器码），它的速度可以接近C++。然而，执行环境非常大，而且有相当大的内存占用。

然而，最严重的是整合问题。Java本地接口（连接Java和C++代码）是为扩展Java而设计的，而不是嵌入它。因此，它可能很难管理。

尽管不是作为一种脚本语言，Java已经被用来开发游戏。*Minecraft*[142]，也许是有史以来最大的独立游戏，完全是用Java实现的，它的所有许多mods和扩展也是如此。

红宝石

Ruby是一种现代语言，具有Python中的优雅设计，但它对面向对象习语的支持更加根深蒂固。它有一些整洁的功能，使它能够有效地操作自己的代码。当脚本必须调用和修改其他脚本的行为时，这可能很有帮助。从C++方面来看，它的重入性并不高，但在Ruby中创建复杂的重入性是非常容易的。

它很容易与C代码集成（不像Lua那样容易，但比Python等更容易）。然而，Ruby似乎可能正在衰落，没有达到本章中其他语言的受众。据我所知，它还没有在任何游戏中使用过（修改过或其他方式）。这是一个鸡和蛋的弱点：由于很少有人分享他们的嵌入经验，所以很难得到关于陷阱和最佳做法的好建议。这可能会随着一两个高调的部署而迅速改变。

13.3 创造一种语言

直到21世纪初，游戏中使用的脚本语言和嵌入式开源语言一样，都是由开发者专门为其需求而创建的。在过去的10年中，这种平衡已经发生了变化，但是仍然有一些情况下定制语言是有用的。特别是对于具有非常小众要求的游戏风格。

Stephen Lavelle的PuzzleScript平台[35]和Graham Nelson的Inform 7交互式小说系统[44]都是迷人的基于规则的语言（而不是proce-dural、面向对象或函数式），与

他们的游戏引擎紧密相连。Inkle的Ink语言[27]是为自己的游戏开发的，目标是为Unity游戏引擎编写的运行时间。它们都是为现有的语言会更麻烦的情况下编写的。

商业游戏引擎包括对脚本语言的支持，在某一时期，这些语言通常是自定义设计的，类似于广泛提供的开源产品。这些都不再是积极开发的。虚幻引擎有UnrealScript，而Unity

有UnityScript。Epic从UE4中删除了UnrealScript，而UnityScript也正在被废弃中。似乎没有什么理由从头开始开发和维护一种与成熟语言如此相似的语言。UE4确实有自己的自定义语言--Blueprint，这是有必要的，因为它是如此的不寻常：它是一种介于编程和指定行为树之间的视觉语言。

那么，你是否应该投资于自己的语言？仔细考虑一下利弊。

优势

当你创建自己的脚本语言时，你可以确保它完全按照你的要求来做。因为游戏对内存和速度的限制很敏感，你可以只把你需要的功能放入语言中。如上所述，例如，现有的语言往往对重入性的支持很差：这可能是你设计的一个核心部分。你也可以添加一些游戏应用所特有的功能，这些功能通常不会被包含在通用语言中。或者，像我在上一节提到的基于规则的语言一样，以一种完全不同的方式来构造你的语言。

因为它是内部创建的，所以当语言出问题时，你或你的团队知道它是如何构建的，通常可以更快地发现错误并创建一个解决方法。

每当你把第三方代码纳入你的游戏，你就会失去对它的一些控制。在大多数情况下，其优点超过了灵活性的缺乏，但对于某些项目来说，控制是必须的。

劣势

新创建的语言往往有更多的基本功能，而且不如现成的替代品健全。如果你选择了一种相当成熟的语言，比如上面提到的那些，你将受益于大量的开发时间、调试工作以及其他人所做的优化。每个在你之前使用过这种语言的人都是一种质量保证测试员。内部语言需要进行彻底的测试，这是一项额外的开支。

一旦你的团队建立了基本的语言并转移到其他的编码任务，开发就停止了。没有熟练的开发者社区继续在语言上工作，改进它并消除错误，而不给你增加任何费用。许多开源语言提供网站（通常是GitHub），在那里可以讨论问题，报告错误，并可以下载文档。

许多游戏，尤其是PC上的游戏，其编写的目的是允许消费者编辑其行为。客户建立新的对象、关卡或整个MOD可以延长游戏的保质期。使用为你的游戏定制的

脚本语言，需要用户学习该语言。这又意味着你需要提供教程、样本代码和开发者支持。大多数现有的语言都有新闻组或网络论坛，客户可以在那里获得建议而不需要联系你或你的团队。通常会有一个专门的小组

熟练的开发人员在Stack Overflow上回答问题。很难与之竞争，如果你尝试，那将是非常昂贵的。

如果你是一个业余爱好者，我建议创建你自己的语言，只作为一种练习。或者，如果你在商业上创造你的游戏，只有当你的游戏或游戏引擎有非常具体和特殊的功能时，才可以这样做。

13.3.1 轧制自己的

无论你的最终语言的外观和功能如何，脚本在被执行的过程中都会经过相同的过程：所有的脚本语言都必须提供相同的基本元素集。因为这些元素是如此的普遍，所以已经开发和完善了一些工具，以使它们易于建立。

我没有办法在这本书中给出一个完整的构建自己的脚本语言的指南。有很多其他关于语言构建的书籍（尽管令人惊讶的是，据我所知，还没有任何关于创建简单脚本语言的好书）。本节从一个非常高的层次来看待脚本语言构建的要素，作为对理解的一种帮助，而不是实施。

语言处理的各个阶段

脚本从文本文件中的文本开始，通常要经过四个阶段：标记化、解析、编译和解释。

这四个阶段形成了一个流水线，每个阶段都对其输入进行修改，将其转换成更容易操作的格式。这些阶段可能不会一个接一个地发生。所有的步骤都可以是相互联系的，也可以由一些阶段组成独立的阶段。脚本可以被标记化、解析和编译，例如，为了以后的解释。

符号化

符号化可以识别文本中的元素。一个文本文件只是一个字符序列（在ASCII字符的意义上！）。代号化器计算出哪些字节属于一起，以及它们形成什么样的组。

一个形式的字符串：

```
1 a = 3.2;
```

可以分成六个代币：

```
'a' 文本
<空格>空格'='平等运算
符
<空格> 白点
3.2 浮点数
';' 语句标识符的结束

```

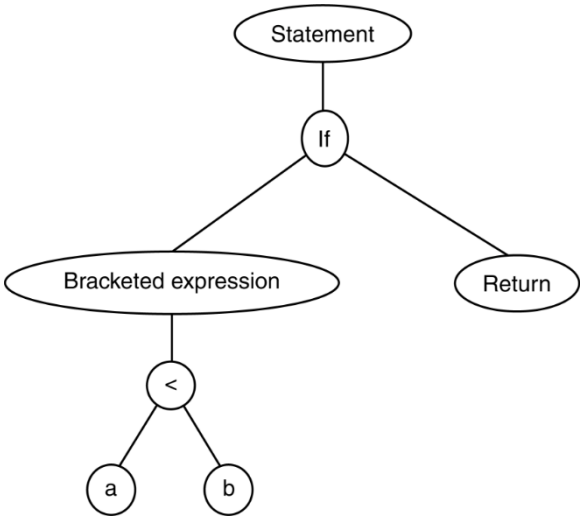


图13.1：一个解析树

请注意，标记器并没有解决这些东西如何组合成有意义的块的问题；那是解析器的工作。

符号化器的输入是一串字符。输出是一个标记的序列。

剖析

一个程序的意义是非常有层次的：一个变量的名字可能在一个as- signment语句里面找到，在一个IF语句里面找到，而这个IF语句在一个函数体里面，在一个类定义里面，在一个命名空间声明里面，例如。剖析器采用标记的序列，确定每个标记在程序中的作用，并确定程序的整体层次结构。

这一行的代码：

```
1 如果 （a < b） 返回；
```

转换为标记序列：

```
1 keyword(if)，whitepace，open-rackets，name(a)，operator(<)，
2 name(b)，close-brackets，whitespace，keyword(return)、
3 报表结束
```

被解析器转换为如图13.1所示的结构。

这种分层结构被称为*解析树*，有时也被称为*语法树*或*抽象语法树*（简称AST）。完整语言中的解析树可能更加复杂，为不同类型的符号或为语句分组增加了额外的层次。通常情况下、

解析器将与树一起输出额外的数据，最明显的是一个符号表，用来识别代码中使用了哪些变量或函数名。这并不是必须的。有些语言在解释器中运行时动态地查找变量名（例如，Python就是这样做的）。

代码中的语法错误在解析过程中显示出来，因为它们使解析器无法建立输出。

剖析器并不能解决程序应该如何运行的问题；那是编译器的工作。

汇编

编译器将解析树变成可以由解释器运行的字节码。字节码通常是连续的二进制数据。

非优化编译器通常将字节码作为解析树的字面翻译来输出。因此，一个代码，如

```
a = 3;
如果 （a<❶） 返回1
；否则返回❶ ；

```

可以被编译成：

```
负荷3
set-value-of a
get-value-of a
compare-with-zero
if-greater-jump-to
LABEL加载1
返回
LABEL
：
装载 ❶返
回
```

优化编译器试图理解程序，并利用先前的知识来使生成的代码更快。一个优化的编译器可能会注意到，当遇到上面的IF语句时，a必须是3。因此，它可以生成：

```
负荷3
设置负载的价值 ❶
返回

```

构建一个高效的编译器已经远远超出了本书的范围。构建简单的编译器并不困难，但不要低估构建一个好的解决方案所需的努力和经验。外面有许多成百上千的自制语言的可怜的编译器。

符号化、解析和编译通常是离线完成的，通常被称为 "编译"，尽管这个过程包括所有三个阶段。然后，生成的字节码可以被

在运行时存储和解释。解析器和编译器可能很大，在最终的游戏中不出现这些模块的开销是有意义的。

口译

管道的最后阶段是运行字节码。在C或C++等语言的编译器中，最终产品将是可由处理器直接运行的机器指令。在脚本语言中，你经常需要提供机器语言不容易实现的服务（如重入和安全执行）。

最后的字节码在 "虚拟机 "上运行。这实际上是一个从未在硬件中存在过的机器的模拟器。

你决定机器可以执行的指令，这些就是字节码的结构。在前面的例子中、

```
加载<值>。
set-value-of  <variable>
get-value-of  <variable>
compare-with-zero
if-greater-jump-to <location>
返回
```

都是字节码。

你的字节码指令也不必局限于那些在真实硬件中可能出现的指令。例如，可能有一个 "将数据转化为一组游戏坐标 "的字节码：这种指令使你的编译器更容易创建，但真正的硬件永远不需要。

大多数虚拟机由C语言的大开关语句组成：每个字节码都有一个简短的C语言代码，当解释器中的字节码到达时就会执行。因此，"添加 "字节码有一点C/C++代码来执行添加操作。我们的转换示例可能有两到三行C++代码来执行所需的转换并将结果复制到适当的地方。

及时编撰

由于字节码的高度顺序性，有可能编写一个运行速度非常快的虚拟机。尽管它仍然是解释的，但它比一次一行地解释源语言要快很多倍。

然而，通过增加一个额外的编译步骤，完全取消解释步骤是可能的。一些字节码可以被编译成目标硬件的机器语言。当这是在虚拟机中完成的，就在执行之前，这被称为*及时*编译（JIT）。这在游戏脚本语言中并不常见，但却是Java和微软的

.NET字节码等语言的主流。

工具：快速了解Lex和Yacc

Lex和Yacc分别是用于构建标记器和解析器的两个主要工具。每种工具都有许多不同的实现方式，并随大多数UNIX发行版提供（也有适用于其他平台的版本）。我最常使用的Linux变体是Flex和Bison。

要用Lex创建一个标记器，你要告诉它在你的语言中什么构成了不同的标记。例如，什么构成了一个数字（即使这个数字在不同的语言中也是不同的--比较0.4f和1.2e-9）。它产生的C代码将把来自你的程序的文本流转换成标记代码和标记数据流。它生成的软件几乎肯定比你自己编写的软件更好、更快。

Yacc构建解析器。它采用你的语言的语法表示--例如，哪些标记在一起有意义，哪些大结构可以由小结构组成。这个语法是由一组规则给出的，这些规则显示了较大的结构是如何由较简单的结构或代币组成的，比如说：

```
赋值： NAME'='表达式； 表达式： 表达式'+'表
达式； 表达式： NAME
```

第一行是一条规则，告诉Yacc，当它发现一个NAME标记，后面是一个等号，后面是一个它知道是表达式的结构（许多规则中的两条显示出来），那么它知道它有一个赋值。第一个表达式规则是递归定义的。Yacc还可以生成C代码。在大多数情况下，生成的软件和你手动创建的软件一样好，甚至更好，除非你有编写解析器的经验。与Lex不同的是，如果速度是绝对关键的，最终的代码往往可以进一步优化。

幸运的是，对于游戏脚本来说，代码通常可以在不玩游戏的时候进行编译，所以轻微的低效率并不重要。

Lex和Yacc都允许你将自己的C代码添加到标记化或解析软件中。然而，并没有一个*事实上的*标准工具来进行编译。根据语言的行为方式，这将有很大不同。然而，让Yacc为编译器建立一个AST是很常见的，有各种工具可以做到这一点，每个工具都有自己的特定输出格式。

许多基于Yacc的编译器不需要创建一个语法树。它们可以在规则内使用写进Yacc文件的C代码创建字节码输出。例如，只要发现一个赋值，其字节码就会被输出。然而，以这种方式创建优化的编译器是非常困难的。因此，如果你打算创建一个专业的解决方案，值得直接去找某种解析树。

第四部分

设计游戏的AI

14

设计游戏Ai

到目前为止，在本书中我们已经看到了整个人工智能技术的调色板，以及允许人工智能运行的基础设施。我在第2章中提到，游戏中的人工智能开发是一个混合体。

在技术和基础设施方面，有大量的*临时*解决方案、启发式方法和一些看起来像黑客的代码。

本章着眼于所有的比特如何应用于真实的游戏，以及如何应用技术来获得开发者想要的游戏性。

在逐个流派的基础上，我将看一下玩家对游戏AI的期望和隐患。这里不包括技术，只是说明书中其他地方的技术是如何应用的。这里的分类是相当高层次和宽松的，有些游戏可能被作为不同的类型进行销售。但是，从人工智能的角度来看，要实现的东西相对有限，我对流派进行了相应的分组。

在深入研究每个流派之前，值得看看设计你的游戏中的AI的一般过程。

14.1 设计

在本书中，我一直从同一个游戏AI模型出发，在图14.1中再次重复。除了映射可能的技术外，这张图还为设计AI时需要考虑的领域提供了一个计划。

当设计一个标题的人工智能时，团队通常从设计文件中描述的目标行为集开始工作。人工智能程序员（通常是人工智能技术负责人）制定最简单的技术集，以支持设计师的愿景。这可能要追溯到

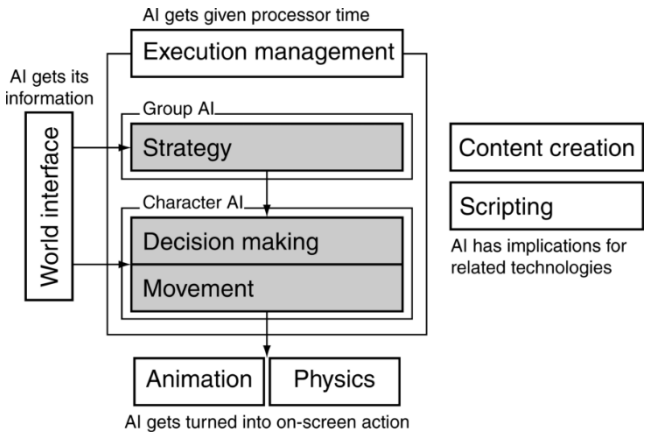


图14.1：AI模型

如果某件事情特别困难，或者有一个简单的机会来获得更复杂的结果，那么就会来回走动。一旦团队确信自己理解了这些行为所带来的要求，就会选择技术来实现它们，以及将这些技术整合在一起的方法。结构被实施，包括计划中的人工智能和游戏引擎的其他部分之间的集成层。起初，角色可以使用放置的行为，但随着基础设施的到位，不同的开发者或关卡设计者可以开始工作，使角色更加丰满。

当然，这是一个理想，一个如果我对一个项目有自由支配权的行动计划。在现实中，你将面临来自许多不同方向的限制，这些限制将影响你的方法计划。特别是，出版商的里程碑和会议或早期访问版本的预告游戏意味着功能和行为需要在开发周期的早期实现。在许多项目中，如果可悲的是，不是大多数项目，像这样的内容只是为了里程碑而迅速实施，然后在以后删除和重写。在相当多的项目中，这些快速和肮脏的代码最终被修补和黑客攻击，以至于最后不可能通过手术移除，而成为被运送的AI。

这类问题是行业的一个正常部分，发生在每个人身上。时间很紧，资源有限，而且总是有更多可以做的事情。如果有一个秘密的游戏人工智能专家的小集团，你就不会因为时不时地运送黑客和半生不熟的人工智能代码而被拉黑了。另一方面，质量确实会受到关注：如果你能提前考虑并建立可靠和有效的人工智能，你就能为你的事业做出巨大贡献。阴谋家们一直在观察。

14.1.1 例子

在本节中，我将通过一个例子来阐述一个假设的游戏的两个阶段的设计需求（所需的行为和实现这些行为的技术）。从游戏的角度看，这个游戏很简单，但对人工智能的要求是多样的。

我们的游戏叫 "鬼屋"，毫不奇怪，它的背景是在一个鬼屋。这是一个著名的鬼屋，来自四面八方的人花钱来参观它。玩家拥有这所房子，玩家的工作是通过管理房子里的惊吓，确保游客得到他们想要的惊吓，从而让顾客付钱。

访客来到房子里，玩家的目的是让他们惊慌失措地逃走。为了做到这一点，玩家可以选择一些幽灵和机械技巧，在房子里使用。以前的访客不可避免地会分享他们的经历，而其他人也会来寻求揭穿或模仿他们的惊恐。

玩家还必须努力防止游客偶然发现房子的秘密、交易的技巧、单向镜、烟雾机和鬼魂的公共房间。

这个想法的一个变种可以在《*幽灵大师*》[177]中看到，在那里，各种房屋都呈现出不同的居住者。住户们并不期望受到惊吓，而是遵循他们自己的模拟人生。这也与《*地牢守护者*》[90]和《*邪恶天才*》[110]等游戏有相似之处。

14.1.2 评价行为

第一项任务是设计你游戏中的人物将显示的行为。如果你在做自己的游戏，这可能是你对项目设想的一部分。如果你在一个开发工作室工作，这可能是游戏设计师的工作。

虽然游戏的设计者会对游戏中的人物应该如何行动有既定的想法，但根据我的经验，这些想法很少是一成不变的。通常情况下，设计师并不了解哪些东西看起来微不足道，但确实很难（因此只有当它是游戏的中心点时才应该包括在内），以及许多看似困难但简单的补充，可以用来改善角色行为。

在你实施和尝试新事物的过程中，游戏中人物的行为会自然而然地发生变化。这不仅仅适用于业余项目或研究和开发阶段较长的游戏；对于一个有固定想法和时间紧迫的开发项目来说也是如此。即使有再好的意愿，在你开始开发游戏之前，你也不会完全了解游戏的人工智能需求。值得从一开始就为某种程度的灵活性进行规划。

例如，用一套固定的游戏输入来创建一个人工智能，只是要求在项目结束后的深夜（反正都会发生，为什么还要要求呢）。不可避免的是，设计者会在不方便的时候需要一些额外的人工智能输入，而人工智能的代码也需要重做。正因为如此，在最初的实施计划中，最好是偏向于灵活性而不是原始速度。以后再进行优化，要比在最后一刻对纠缠在一起的代码进行取消优化以获得灵活性要容易得多。

因此，从我们想看到的一系列行为开始，我们对人工智能模型的每个组成部分都有一些问题需要回答：

- 运动
  - 我们的角色将被单独表现出来（如大多数游戏），还是只看到他们的群体效应（如城市模拟游戏）？
  - 我们的角色是否需要以一种或多或少的现实方式在他们的环境中移动？或者我们可以只把他们放在我们想让他们去的地方（例如在一个基于瓷砖的回合制游戏中）？
  - 角色的运动是否需要进行物理模拟，例如在汽车游戏中？物理学必须有多大的真实性（请记住，建立运动算法以配合真实的物理学，通常要比调整物理学使其对人工智能角色不那么真实要难得多）？
  - 角色是否需要想办法去哪里？他们能不能只是徘徊，按照设计者设定的路径，只呆在一个小区域，或者追赶其他角色？还是我们需要角色能够用寻路系统在整个关卡中规划他们的路线？
  - 角色的运动是否需要受到任何其他角色的影响？追逐/回避的行为是否足以应付，还是说角色也需要协调或列队移动？
- 决策的制定
  - 这通常是人工智能设计师最容易得意忘形的地方。在游戏的预生产阶段，想要采用各种奇特的新技术是很诱人的。更多的时候，最终的游戏是以状态机器或硬编码的脚本来运行所有重要的东西。
  - 你的角色在游戏中可以进行的各种不同行动是什么？
  - 你的每个角色会有多少种不同的状态？换句话说，这些行动是如何组合在一起以实现角色的目标的？请注意，我并不是假设你将在这里使用状态机或目标导向的行为。无论你的角色有什么动力，他们都应该有目标，当为实现一个目标而行动时，他们可以被认为是处于一种状态。
  - 你的角色何时会改变其行为，切换到另一个状态，或选择另一个目标来追随？什么会导致这些变化？为了在正确的时间改变，它需要知道什么？
  - 人物是否需要向前看，以选择最佳的决定？他们是否需要计划他们的行

动，或者执行只间接导致其目标的行动？这些行动是否需要行动规划，或者可以用更复杂的基于状态或基于规则的方法来涵盖它们？

- 你的角色是否需要根据玩家的行为方式来改变它的决定？它是否需要根据对玩家行为的记忆，使用某种学习方式来做出反应？

- 战术和战略AI
  - 你的角色是否需要了解游戏关卡的大规模属性，以便做出合理的决定？你是否需要以一种使他们能够选择适当行为的方式向他们表述战术或战略情况？
  - 你的角色需要一起工作吗？他们是否需要根据对方的时间安排，以正确的顺序开展行动？
  - 你的角色能不能在独立思考的同时表现出你所要的群体行为？或者你需要一次为一组人物做一些决定？

例子

在《鬼屋》中，我对这些问题提出了以下一组初步答案：

- 运动
  - 角色将被单独表示，在他们的环境中自主地移动。我们不需要现实的物理模拟。我们可以用运动学的运动算法而不是完全的转向行为来解决。角色经常想去一个特定的地方（例如出口），这可能需要在房子里导航，所以我们需要寻路。
- 决策的制定
  - 角色有一个小范围的可能行动。他们可以爬行、奔跑或静止（被石化）。他们可以检查物体或 "对它们采取行动"：每个物体最多可以对其进行一个行动（例如，可以拨动灯的开关或打开门）。他们还可以安慰房子里的其他人。
  - 角色将有四大类型的行为：惊恐行为，他们将试图恢复他们的智慧；好奇行为，他们将检查物体和探索；社会行为，他们将试图保持群体的团结，并安慰有关成员；无聊行为，他们将前往客户服务台，要求退款。
  - 角色会根据恐惧的程度改变他们的行为。每个角色都有一个恐惧等级。当一个角色通过一个阈值时，它将进入恐惧行为。当一个角色靠近另一个害怕的角色时，它将进入社交行为。如果一个角色的恐惧水平降得很低

，它就会感到厌烦。否则，它将进入好奇模式。

- 角色会通过看到、听到或闻到奇怪的东西来改变他们的恐惧程度。每个幽灵和诡计在这三种感官中都有一个奇怪的强度。角色需要被告知他们何时能看到、听到或闻到什么，以及它看起来有多奇怪。

  - 人物会寻求探索他们以前没有去过的地方，或者会回到他们或其他人以前喜欢的地方。他们应该记录下已去过的地方和有趣的地方。有趣的地方可以在许多小组之间分享，以代表对好的惊吓的八卦。

- 战术和战略AI

  - 角色在试图恢复其智慧时，需要避开他们知道是可怕的地点。同样地，他们在寻找行动时也会避开那些无聊的地方。

14.1.3 选择技术

有了对基于行为的问题的回答，你将对你需要在人工智能方面走多远有一个很好的概念。例如，你可能已经搞清楚了你是否需要寻路，以及需要什么样的运动行为，但不一定要使用哪种寻路算法或哪种转向仲裁系统。

这是下一个阶段：建立一个你打算使用的候选技术集。这方面的工作大部分是相当直接的。如果你已经决定你需要寻路，那么A*是明显的选择。如果你知道人物需要列队移动，那么你就需要一个列队运动系统。决策方法就比较麻烦了，因为往往有几种方法可以得到同样的效果。你所选择的方法可能更多的是与你所拥有的构建经验有关，与你已经授权的工具有关，或者与你可以重用的现有代码有关。

正如我们在第五章所看到的，选择决策系统没有硬性规定。大多数你能用一个系统做的事情，你也能用其他系统做。如果你要从头开始建立你的人工智能，我的建议是从简单的技术开始，比如行为树或状态机或两者的简单组合，除非你知道你想做的特定事情不能用它们实现。它们的灵活性已经多次证明了它的价值，以至于我个人需要有一个好的理由来使用更复杂的东西。最近，这个 "很好的理由 "往往是我正在使用的引擎中存在着工具支持的行为树。

在这个阶段，尽量避免被拉回到重新设计你确定的行为上。我们很容易想到，如果我们使用这样那样的奇特技术，那么我们就可以展示这样那样的酷炫行为。重要的是，要将酷的效果与其他95%的人工智能的工作能力融合在一起，使之坚如磐石。

例子

在《鬼屋》中，我们可以用这本书中的以下一套技术来满足我们行为的要求：

- 运动
  - 角色将以运动学的运动算法进行移动。他们可以选择任何方向，以两种移动速度中的一种进行移动。
  - 在好奇和害怕的模式下，他们会选择他们的运动目标为一个房间，并使用A*在那里寻路。他们将使用路径跟踪行为来跟踪该路线。我们将使用一个航点图来配合战术和战略AI，如下。
  - 在社交模式下，他们会使用运动学上的寻求行为，向他们能看到的受惊人物走去。
- 决策的制定
  - 角色将使用一个非常简单的有限状态机来确定其广泛的行为模式，并在每个状态下使用行为树来确定对它的实际行动。
  - 状态机有四种状态：害怕、好奇、社交和无聊。转变完全基于一个角色的恐惧程度和视线范围内的其他角色。
  - 在每个模式中，可能有一系列的行动。在好奇模式中，角色可以调查地点或物体；在恐惧模式中，他们要选择最好的方法来找到一个安全的地方来收集他们的智慧。这些行为中的每一个都被落实为决策树，由选择器选择各种策略。每个策略又可以有多个元素，这些元素可以被添加到树的序列节点上。
- 战术和战略AI
  - 为了方便学习可怕和安全的地点，我们保留了一张关卡的航点地图。当角色改变他们的恐惧状态时，他们会在地图上记录这一事件。这与第6章创建碎片地图的过程一样。
- 世界界面
  - 角色需要获得关于游戏中奇怪的景象、气味和声音的信息。这应该由一个感官管理模拟来处理（一个地区的感官管理员就可以了）。
  - 角色在好奇模式下也需要关于可采取的行动的信息。角色可以请求一个它可以与之互动的物体列表，我们可以从游戏中的物体数据库中提供这些

信息。我们不需要模拟角色看到和识别这些物体。

- 执行管理
  - 有两种技术，寻路和感应管理。两者都很耗时。

- 由于房子里只有几个房间，一个人的寻路不会花很长时间。然而，房子里可能有很多角色，所以我们可以使用一个由几个规划师组成的池子（一个就够了），并对寻路要求进行排队。当一个角色要求寻找路径时，它就会等待，直到有一个规划者有空，然后一次性得到它的路径。我们不需要随时进行寻路的算法。
- 感知管理系统在每一帧都会被调用，并逐步更新。在设计上，它是一个分布在许多帧上的随时随地的算法。
- 房子里可能同时有很多角色（比方说几十个）。每个角色的行动都相对缓慢；它不需要每一帧都处理其所有的人工智能。我们可以避免使用复杂的分层调度系统，只需每帧更新几个不同的角色。每帧更新5个角色，游戏中50个角色，每秒渲染30帧，一个角色在更新之间需要等待不到半秒。这种延迟实际上可能是有用的；让角色在对惊吓做出反应之前等待几分之一秒，可以模拟他们的反应时间。

在这个游戏中，我最终只有少数几个模块需要实施。感官管理系统可能是最复杂的，其他的都是非常标准的，有简单的组件。我甚至还设法包括了随机数发生器：我们在第二章中遇到的第一个人工智能技术。

14.1.4 一场比赛的范围

考虑到本书中的技术范围，你可能期望我把《鬼屋》做得更复杂，依靠巧妙地使用许多不同的算法。最后，我们的设计中唯一略带异国情调的是用于通知角色怪异事件的感觉管理系统。

在现实中，游戏中的人工智能是这样工作的。相当简单的技术承担了大部分的工作。如果你正在寻找特定的基于人工智能的游戏效果，那么就可以应用一两个不常用的技术。如果你发现自己设计的游戏有神经网络、感觉管理、转向管道和基于Rete的专家系统，那么可能是时候把注意力集中在游戏中真正重要的东西上了。

本书中每一个比较特殊的技巧在一些游戏中都是至关重要的，可以使一个无聊的游戏和真正整洁的角色行为之间产生差异。但是，就像精致的香料一样，如果不谨慎地使用它们来增加味道，它们最终会破坏最终产品。

在本章的其余部分，我将考察一系列不同类型的商业游戏。在每一种情况下，我都会试着把重点放在使该类型不寻常的技术上：在那里，新的创新可以真正发挥作用。

本章只讨论了最具商业意义的游戏类型，即大多数人工智能开发者的面包和黄油。本书的最后一章，即第15章，对人工智能专门负责提供游戏性的其他游戏类型进行了采样。这些都不是大型游戏类型

有成千上万的作品，它们并不是最畅销的，但它们对人工智能开发者来说很有趣，因为它们以普通类型的方式拉伸人工智能。

14.2 射手

第一人称和第三人称射击游戏是经济上最成功的类型，自从第一款视频游戏诞生以来，就一直以这样或那样的形式存在。它们是讨论其他类型游戏中敌方角色的人工智能的一个很好的跳板。在本节的最后，我将在射击游戏人工智能的讨论基础上，将重点扩大到冒险游戏、平台游戏、近战战斗和MMOG等类型。首先，我将坚持讨论经典射击游戏。

随着*《狼人杀3D》*[121]和*《毁灭战士》*[122]的出现，射击游戏类型变得与角色步行（可能带有喷气背包，如*《部落II》*[105]）和与玩家的角色绑定的摄像机同义。敌人通常由相对较少的屏幕上的人物组成。

主要用于玩家对玩家（PvP）的游戏，为人类之间的比赛进行了优化，可能会有被称为'机器人'的复杂的实践AI。为了确保公平性，这些计算机控制的角色将具有与玩家尽可能相似的能力。提供战役模式或其他玩家对环境（PvE）挑战的射击游戏，往往依靠数量较少的复杂敌人。

该流派最重要的人工智能需求是：

1. 敌人的运动-控制
2. 准确的火力控制
3. 决策--通常是简单的状态机
4. 感知--确定拍摄对象和他们的位置
5. 寻路--通常（但不总是）用来让角色规划他们在关卡中的路线。
6. 战术人工智能--通常用于让角色确定安全的移动位置或掩护点，以便从那里开火。

其中，前两个是在该类型的所有游戏中看到的关键问题。只用这些工具就能创造出一个射击游戏，特别是随着独立游戏的复苏而重新流行起来的2D射击游戏。但在3D游戏中，这可能显得无望的天真。在过去的15年中，玩家越来越期待敌人具有

一定的战术复杂性（如使用掩体），并使用一些寻路方法来避免卡在关卡上。

14.2.1 运动和射击

在所有的游戏类型中，运动是角色行为中最明显的部分。在3D射击游戏中，人物在屏幕上通常是很大的，运动和动画一起成为信号。

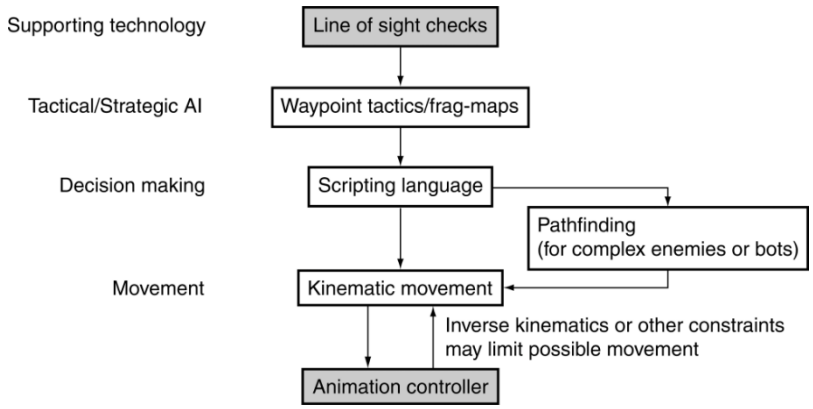


图14.2：适用于第一或第三人称射击游戏的基本AI架构

在游戏中，大部分关于人工智能正在做什么的信息都会提供给玩家。一些射击游戏非常依赖动画，有一些最复杂的动画集。这可以包括不同类型的运动（爬行、爬行、冲刺），重新装弹，对盟友的手势，甚至语音表演。角色结合几十或几百个动作序列，以及其他控制器，如逆运动学或布娃娃物理学，这是很正常的。在《*F.E.A.R.2：起源计划*》[145]中，一个角色可以同时进行跑步、射击和观察。前两个是动画通道，第三个是由角色看的方向控制的程序性动画（射击臂的方向也是如此，但不是整体运动）。

在《*无人生还2*》[143]中，忍者角色拥有复杂的运动能力，增加了运动和动画同步的难度。他们可以进行侧手翻，跳过障碍物，并在建筑物之间跳跃。

在关卡周围的简单运动成为一种挑战。人工智能不仅需要制定路线，而且还需要能够将这种运动分解成动画。大多数游戏将这两部分分开：人工智能决定在哪里移动，而另一大块代码将其变成动画。这允许人工智能有完全的运动自由，但缺点是会出现动画和运动的奇怪组合，这对玩家来说会显得很刺耳。到目前为止，这个难题已经通过包括更丰富的动画调色板来解决，使得更有可能找到一个合理的组合。

一些使用脚本语言来控制其角色的游戏向人工智能暴露了与玩家使用的相同的控制方式。人工智能不是输出所需的运动或目标位置，而是需要指定它前进或后退、转弯、更换武器等的速度。这使得在开发过程中非常容易删除人工智能角色，用人

类来代替它（例如通过网络玩）。大多数游戏，包括那些基于授权的最著名的游戏引擎，都有宏命令--例如：

```
睡眠3
gotoctor PathNodeLoc1
gotoactor PathNodeLoc2
agentcall
Event_U_Wave 1
睡眠2
gotoctor PathNodeLoc3
gotoactor PathNodeLoc
```

由于许多射击游戏中的关卡都是有限的、室内性质的，所以人物几乎肯定需要某种路径搜索。这可能像上面虚幻脚本中的gotoactor state ments那样简单，也可能是一个完整的寻路系统。无论它采取什么形式（我们将在后面回到寻路的考虑），路线都需要被遵循。有了相当复杂的路线，角色就可以简单地沿着路径前进。不幸的是，游戏关卡很可能是动态的。角色应该对其他角色的移动做出适当的反应。这最常用的方法是在所有角色之间使用一个简单的排斥力， acters。如果角色靠得太近，那么他们就会分开移动。在*Mace Griffin：赏金猎人》*[197]中，在游戏的深空部分，同样的技术被用来避免地面上人物之间的碰撞和战斗飞船之间的碰撞。在室内，pathfind- ing被用来创建路线。在太空中，则使用编队运动系统。

光环》[91]及其续集中的洪水，以及《*异形大战》*1[167]中的外星人都是沿着墙壁和天花板以及地面移动的。两者都没有使用严格的2½维（2½D）来表示角色的移动。

射击AI在射击游戏中至关重要（这并不令人惊讶）。毁灭战士的前两代化身因令人难以置信的精确射击而饱受批评（开发者放慢了射入的速度，以使玩家能够移开；否则，精确度会令人难以承受）。更加现实的游戏，如*ARMA*[89]游戏和*Far Cry*[100]系列，使用允许角色失误的射击模型，有时会以令人兴奋的方式调整失误（即，他们试图在玩家可以看到子弹的地方失误）。

14.2.2 决策

决策通常使用简单的技术实现，如有限状态机或行为树。这些可以是非常初级的，只有 "看到的玩家 "和 "没有看到的玩家 "行为。

在射击游戏中，一个非常常见的决策方法是开发一个机器人脚本系统。一个用游戏特定的或引擎提供的脚本语言编写的脚本被执行。脚本有一系列的功能，通过这些功能它可以决定角色能感知到什么。这些功能通常是通过直接查询当前游戏状态

来实现的。然后，脚本可以要求执行一些动作，包括播放动画、

1. 不要与《*异形大战*》[79]、街机和SNES的同名游戏混淆，这两款游戏都是侧向滚动的射击游戏。

脚本语言可以用来修改人工智能，并在某些情况下用于寻路。然后，这种脚本语言被提供给游戏的用户来修改人工智能或创建他们自己的自主角色。这是在*虚幻*[103]和连续的游戏中使用的方法，并且在非射击游戏中也被采用（我注意到的第一个例子是角色扮演游戏*《梦幻西游》*[83]）。对于*《狙击精英》*[168]，Rebellion希望在每次游戏中看到不同的突发行为。为了达到这个目的，他们应用了一系列的状态机，在游戏关卡中的途径点上运行。许多行为取决于其他角色的行动或附近航点的战术形势变化。决策过程中的少量随机性使角色每次都有不同的行为，并以不同的方式行事。

明显的合作，不需要任何基于小队的人工智能。

在*《无人生还2》*[143]中创造了一种稍微不同的自主人工智能的方法。Monolith将状态机与面向目标的行为相融合。每个角色都有一套预先确定的目标，可以影响其行为。这些角色会定期评估他们的目标，并选择当时与他们最相关的一个目标。然后，该目标将控制角色的行为。每个目标里面都有一个有限状态机，用来控制角色，直到选择一个不同的目标。游戏使用航点（他们称之为节点）来确保角色在正确的位置进行行为，如翻阅文件柜、使用电脑和开灯。这些航点在角色附近的存在，使角色能够以了解有哪些行动。

Monolith的人工智能引擎继续进行开发，直到该公司被收购。在*《F.E.A.R.》*[144]中，同样使用了面向目标的行为，但预建的状态机被一个规划引擎所取代，该引擎试图以这样的方式组合可用的行动来实现目标。F.E.A.R.有一个最早的完全面向目标的行动规划系统。

在《*光环2*》[92]和后来的游戏系列中，决策树被用来让人工智能角色在行动中执行基本的计划。当行为树中的选择器中的节点失败时，人工智能会回到代表不同计划的其他节点上，给人工智能提供广泛的战术机会，而这是很难用状态机来指定的。

14.2.3 感知

感知有时是通过在每个敌人角色周围放置一个半径，当玩家在这个半径内时，敌人就会 "活过来 "来伪造。这就是最初的《毁灭战士》所采取的方法。然而，在*《黄金眼007》*[165]的成功之后，更复杂的感知模拟成为人们的期待。这不一定意味着感

知管理系统，但最起码角色应该通过某种信息被告知他们周围发生了什么。

在《*汤姆克兰西的幽灵行动*》[170]游戏中，感知模拟要复杂得多。为人工智能角色提供信息的感知管理系统考虑到灌木丛提供的破损掩护的数量，并测试角色背后的背景以确定他们的伪装是否匹配。这是通过保持一个

游戏中每种材料的图案ID集。视线检查穿过任何部分透明的物体，直到它到达被测试的角色。然后，它继续超越该角色，并确定它所碰撞的下一个物体。然后，伪装ID和背景材料ID被检查是否兼容。

细胞分裂[189]游戏使用了不同的方法。因为只有一个玩家角色（在《幽灵行动》中有很多），每个人工智能只是简单地检查一下自己是否可见。每个关卡都可以包含动态阴影、薄雾和其他隐藏效果。玩家角色会根据这些效果进行检查，以确定隐蔽等级。如果它低于某个阈值，那么敌人的人工智能就会发现玩家角色。

隐蔽关卡并不像《幽灵行动》游戏那样考虑到背景问题；如果角色站在明亮的走廊中间的黑影中，那么它就不会被看到，尽管它在警卫看来是一个明亮背景上的黑色大人物。关卡的设计是为了尽量减少这种限制的明显次数。

细胞分裂》中的人工智能角色也使用视锥检查，还有一个简单的声音模型，根据声音的大小，声音会在当前房间内传播到一定的半径。在《*合金装备*》[129]系列游戏中也使用了非常类似的技术。

14.2.4 寻路和战术AI

在《*命运战士2*》中：*双螺旋*[166]，寻路图中的链接被标记为穿越它们所需的行动类型。当一个角色到达路径中的相应链接时，它就可以改变行为，以显示对地形的了解。这个环节可能代表了一个需要翻越的障碍物，一扇需要打开的门，一个需要突破的障碍物，或者一堵需要滑降的墙。负责的人工智能团队，克里斯托弗-里德和本-盖斯勒，将这种方法称为 "嵌入式导航"。

在射击游戏中加入某种航点战术几乎已经成为一种普遍现象。在最初的《*半条命*》[193]中，人工智能使用航点来计算如何包围玩家。一组人工智能角色将被协调，以便他们占据一组良好的进攻位置，包围玩家的当前位置，如果这是有可能的话。在游戏中，人工智能角色经常会不顾一切地跑过玩家，以占据侧翼位置。

除非你的敌人角色总是冲向玩家，就像最初的《毁灭战士》那样，否则你可能需要实现一个寻路层。大多数射击游戏的室内关卡都可以用相对较小的寻路图来表示，可以快速搜索到。Rebellion在《狙击精英》中使用了相同的寻路系统和战术AI，而Mono- lith为《无人生还2》创造了一个完全不同的表示方法。在Monolith的解决方

案中，角色可以移动的区域由重叠的 "AI体积 "来表示，然后形成寻路图。其行动系统的航点并不直接参与寻路（除了作为寻路者计划的目标）。

在本书第一版的时候，开发者使用了一系列的寻路的表示方法。从那时起，使用导航网格来表示内部空间已经变得几乎（但不完全是）无处不在了。将导航网格的方法与强大的战术分析统一起来是比较费力的，基于网格的战术分析与导航网格的寻路方法并列运行的情况并不少见。

然而，仍然有其他可行的方法。Monolith的寻路卷是另一种方法，许多设置在户外的游戏仍然依赖于基于网格的寻路图。

主要设置在室内的游戏自然而然地将关卡分成若干个区域，通常由传送门（一种渲染优化技术）分开。这些区域可以自然地作为一个更高层次的寻路图，用于长距离的路线规划。这使得分层寻路算法自然适合于能够处理大关卡的实现。

14.2.5 射手类游戏

各种游戏都采用第一或第三人称视角，有类似人类的角色。玩家直接控制一个角色，作为游戏的视角，而敌人角色通常具有类似的身体能力。

结合游戏设置的自然保守性，这意味着许多不能被描述为射击游戏的类型都使用非常相似的人工智能技术。因此，它们往往有相同的基本架构。

与其再次重复同样的内容，我将从这些类型的基本射击游戏设置中增加或删除的内容来考虑。

平台和冒险游戏

与第一人称射击游戏相比，平台游戏通常是为年轻观众准备的。一个主要的设计目标是使敌人的角色变得有趣，但相当可预测。在角色的行为中设计明显的模式是很常见的。玩家通过观察敌人的行动并建立起如何利用其弱点的想法，就会得到奖励。冒险游戏也是如此，在游戏中，敌人成为另一个需要解决的难题。在《*超越善恶*》[192]中，Alpha Sections，一个原本不透风的敌人、例如，在攻击后降低其护盾几秒钟。

在这两种情况下，人工智能都会使用与射击游戏中类似但更简单的技术。移动通常使用相同的方法，尽管平台游戏经常增加飞行的敌人，这需要用2½D（见第三章，3.1.2节）或3D移动算法来控制。特别是冒险游戏，在传达角色动作时，对动

画的负担更大。少数游戏允许他们的角色进行寻路。甚至早在《*Jak and Daxter: The Precursor Legacy*》[148]，有时也会使用导航。那款游戏采用了导航网状表示法，使角色能够智能地移动。在许多平台型游戏中，运动可以安全地保持在本地。决策方面的技术现状仍然是最简单的技术。通常情况下，角色

有两种状态："发现玩家 "状态和 "正常行为 "状态。正常行为通常会被限制在站着播放选择的动画或固定的巡逻路线。例如，在《*奇异世界*》[156]系列中，一些动物使用游荡行为的变体随机移动，直到它们发现主角。

当一个角色发现了玩家，它通常会以寻找或追击的行为来锁定玩家。在一些游戏中，这种定位只限于瞄准玩家并向前移动。其他游戏则扩展了移动角色的能力。例如，在《*古墓丽影III*》[96]和该系列的后期游戏中，人类的敌人会抓住并爬上积木来对付劳拉。这种情况在《*黑暗之魂*》[115]系列的每一款游戏中都有所增加。在《*黑暗之魂3*》[117]中，不同的敌人在被攻击时（即在他们 "对准敌人 "的状态下），可以利用环境的不同特征，包括梯子和单向壁架，在整个关卡中航行很远。而且，与该系列以前的游戏不同，他们不太可能被卡住或从边缘掉下来。

很明显，在此基础上存在变化：一些角色可能有更多的状态，他们可能会呼救，可能有不同的近距离和远距离行动，等等。但我想不出在这些类型的游戏中，有哪个游戏的角色从根本上使用了更复杂的技术。这可能不是一个巧合：更多的复杂性会让玩家难以理解，而且可能显得不公平。可预测的行为，即使冒着可信度的风险，往往也是可取的。

14.2.6 MELEE COMBAT

在上一节中，我提到了《黑暗之魂》及其续集，这款游戏不适合放在关于射击游戏的章节中。尽管射击游戏和近战动作游戏之间有许多相似之处，但当涉及到角色的人工智能时，徒手格斗与使用枪支有根本的不同。近战战斗机制从简单到复杂不等。在简单的一端是不间断的攻击行动，当角色在武器范围内时就会成功，可能有一些随机的机会，取决于受害者的盾牌或躲避统计。在复杂的一端是格斗游戏，它可以有一系列令人困惑的机制，包括取消、反转、连击和特殊。

除了最简单的游戏之外，近战战斗从根本上说是关于时间的，而且是独一无二的。图14.3显示了一个近战战斗系统中的示例动作的示意图。它有几个阶段：

1. 绞杀期，这时攻击动画已经开始，但不能造成伤害，而且角色本身仍然容易受到攻击。
2. 一个可中断的时期，当角色能够造成伤害时，但攻击仍然可以被敌人打断或抵

挡。

3. 一个无敌期，当攻击能够造成伤害时，但它不能被打断。

4. 一个冷却期，此时攻击不再危险，但玩家不能开始另一个行动。

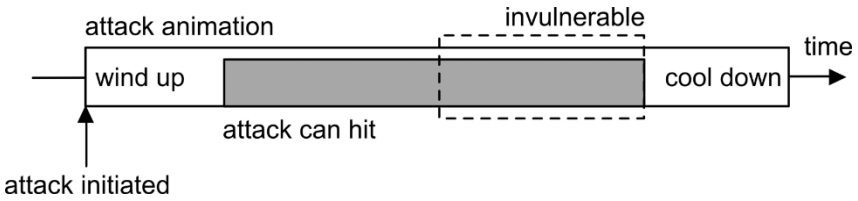


图14.3：近战格斗游戏中的出招时机

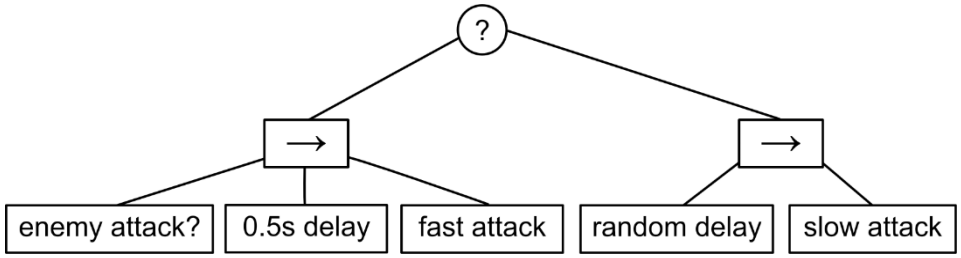


图14.4：一个简化的行为树，包含了近战时机。

这里有几个标准在起作用：动画是否在运行，动作是否在进行中；玩家是否容易被攻击、打断或招架；攻击是否能够伤害对手；以及玩家是否能够结束当前的攻击，开始另一个动作。图中显示了一个如何安排的例子，但每个游戏都会以不同的方式排列它们，通常不同的角色或甚至不同的武器有自己的模式。在几乎所有的情况下，即使阶段是相同的，时间也会有所不同。

这些游戏中的人工智能不仅要对执行哪一招做出决定，而且要对何时执行做出决定。这不需要独特的算法，我们在本书中看到的同样的决策工具都可以使用，但它们的内容（例如状态机中的特定状态模式，或行为树中的节点）必须以包含计时的方式来设计。图14.4显示了一个行为树的部分例子，图14.5显示了使用中的行为树，其中有成功防御的攻击和通过的攻击的时间模式。

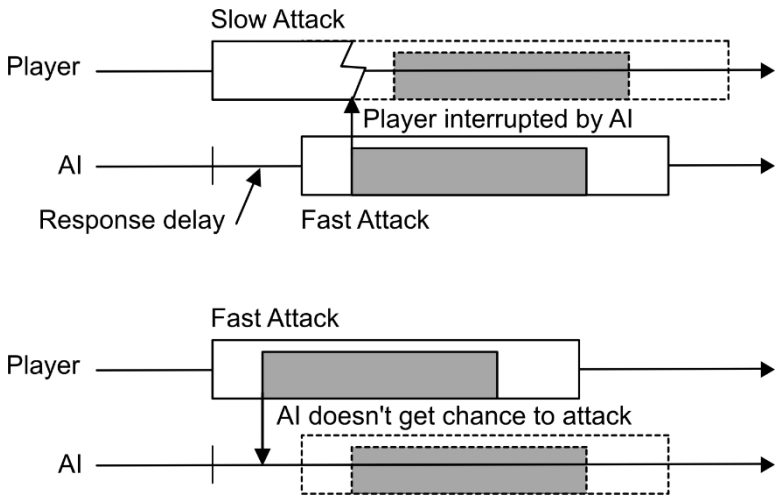


图14.5：两个不同场景中使用的近战行为树。

在有近战的动作冒险游戏中，玩家技能发展的一部分是了解他们的敌人的动作集，以及每个人将如何回应玩家的行动。在这种情况下，一个简单的决策算法是有益的。玩家会了解到某个敌人擅长抵挡慢速剑击，只要它的反应相对一致。当然，它也可能过于机械化。一些随机化可能是必要的，例如，它在五次中只有四次能抵挡。但这可以通过决策树、状态机或行为树来实现--这些简单的决策方法将使玩家能够 "学习 "敌人的攻击。

即使使用简单的工具，也有可能使人工智能变得高度复杂和狡猾。事实上，实现不可能被打败的人工智能是比较简单的，它能够在任何攻击开始的一帧内做出完美的反应。实现这类人工智能的大部分困难在于要有正确的感觉。在挑战和公平之间走一条细线。这就是人工智能成为艺术多于科学的地方，没有任何技术可以拯救你的调整和游戏测试。

网络游戏

大规模多人在线游戏（MMOGs）通常在一个持久的世界中涉及大量的玩家。从技术上讲，它们最重要的特点是将运行游戏的服务器和玩家正在玩的机器分开。

客户端和服务器之间的区别通常在射击游戏（和许多其他类型的游戏）中实现，也是为了使多人模式更容易编程。然而，在MMOG中，服务器永远不会与客户端在同一台机器上运行；它通常会在一套专用硬件上运行。因此，我们可以使用更多的内存和处理器资源。

一些大型多人游戏对人工智能的需求不大。唯一由人工智能控制的角色是动物或奇怪的怪物。游戏中的所有角色都由人类扮演。

虽然这可能是一个理想的情况，但它并不总是实际的。游戏需要一些临界质量的玩家，才值得任何人去玩。大多数网络游戏在游戏中加入了某种基于人工智能的挑战，就像你在任何第一或第三人称冒险中看到的那样。有了这样一个巨大的游戏世界，人工智能开发者面临的所有挑战都是规模方面的。所使用的技术与射击游戏基本相同，但它们的实施需要明显不同，以应对大量的角色和一个更大的世界。一个简单的A*寻路者可以应付射击游戏中的一个关卡，以及5到50个角色使用它来规划路线，而当1000个角色需要使用它时，它很可能会陷入停顿。他们在一个大陆大小的世界中规划自己的道路。

正是这些大规模的技术，特别是寻路和感觉感知，需要更多可扩展的实现。我们已经研究了其中的一些技术。例如，在寻路方面，我们可以汇集规划者，使用分层寻路，或使用实例几何。

14.3 开车

对于开发者来说，驾驶是最专业的、特定类型的AI任务之一。与其他类型不同，关键的人工智能任务都是围绕运动展开的。任务不是创造现实的目标寻求行为、巧妙的战术推理或路线寻找，尽管所有这些都可能出现在一些驾驶游戏中。玩家将通过人工智能的驾驶能力来判断它的能力。

图14.6显示了适合赛道驾驶游戏的人工智能架构，图14.7将这一架构扩展到了城市驾驶游戏中，在这个游戏中，不同的路线是可能的，而且周围的车辆也会共享道

路。

14.3.1 活动

对于赛车游戏来说，开发者有两种选择来实现汽车运动。最简单的方法是允许关卡设计者创建一条或多条赛车线，沿着这些线

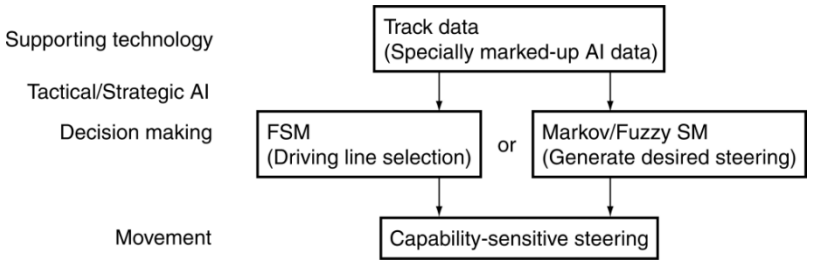


图14.6：用于比赛驾驶的人工智能架构

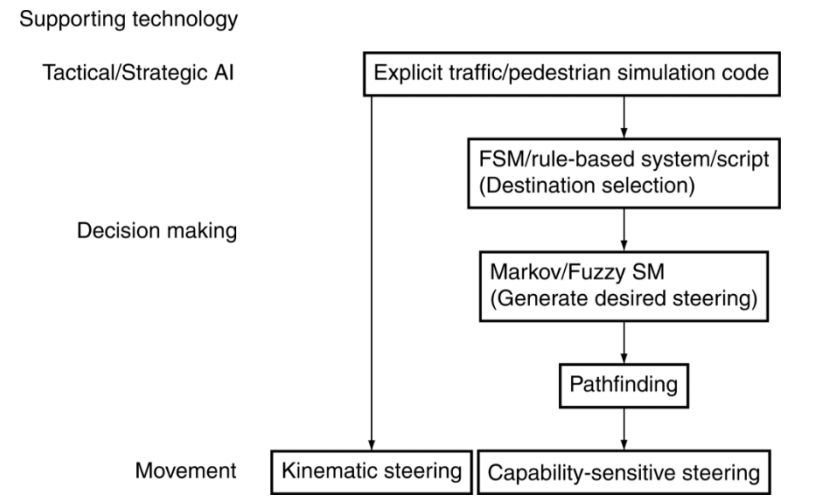


图14.7：用于城市驾驶的AI架构

车辆可以达到其最佳速度。然后可以严格遵循这条赛车线。这甚至可能根本不需要转向。计算机控制的汽车可以简单地沿着预定的路径移动。

通常情况下，这种赛车线是以花键的形式定义的：一种数学曲线。花线是以空间中的曲线来定义的，但它们也可以纳入额外的数据。纳入花键的速度数据允许人工智能在任何时候准确查找汽车的位置和速度，并相应地进行渲染。这提供了一个非常有限的系统：汽车不能轻易地相互超车，它们不会避开前面的碰撞，在与玩家碰撞时也不会被偏离。为了避免这些明显的局限性，增加了额外的代码，以确保如果汽车被撞出位置，可以进行简单的转向行为，使其回到赛车线上。它的特点仍然是汽车倾向于天真地流向被撞的汽车。

大多数早期的驾驶游戏，如*《F1》*[84]，都采用了这种方法。在最近的许多游戏中，它也被用于控制那些旨在成为"背景"一部分的汽车，如*《侠盗猎车手3》*[104]。

第二种方法，在最近的作品中被广泛使用，就是让人工智能驾驶汽车--将控制输入应用到物理模拟中，使汽车表现得很真实。人工智能汽车必须应付的物理学程度与玩家经历的物理学程度相同是一个关键问题。通常情况下，玩家的物理条件比人工智能控制的汽车要难一些，尽管许多游戏现在给人工智能的任务与玩家相同。

在这种游戏中，仍然可以看到赛车线的定义，这是非常常见的。人工智能控制的汽车试图通过驾驶汽车来遵循赛车线，而不是让赛车线作为一个轨道让它移动。这意味着人工智能经常不能达到它所期望的线路，特别是当它被另一辆车挤到的时候。这可能会造成更多问题。在使用这种方法的*Gran Turismo*[160]"驾驶模拟器"系列中，一辆车可能被玩家撞出位置。这时，该车仍会试图驾驶它的赛车路线，这通常会导致它在下一个弯道中超越自己，最终落入砂石陷阱。

为了解决超车问题，当移动速度较慢的车辆坐在赛车线上时，许多开发者会添加特殊的转向行为：汽车会等到一个长直道，然后拉出来超车。这是从Gran Turismo到*Burnout*[98]等许多驾驶游戏中看到的特征性超车行为，也是现实世界中中、低功率汽车的常见超车伎俩。然而，世界上最快的赛车系列（如一级方程式）中的大多数超车行为都是在弯道的制动下进行的。这可以通过关卡设计者定义的替代赛车线来完成。如果一辆车想超车，它就在这条线路上占据一个位置，这将确保它可以稍后刹车并控制出弯。在现实生活中的赛车中，车手能够采取一些防御性行动来阻止潜在的超车（尽管这可能受到系列规则的限制）。这使得这种替代性的赛车线

路难以界定，特别是在玩家被超越的时候。幸运的是，在这一点上，不太成功的人工智能让玩家感觉更熟练，因此不太可能受到负面评价。

在许多拉力赛游戏中使用了跟随赛线的方法的一个变种，有时被称为 "追赶兔子 "。一个看不见的目标（同名的兔子）使用直接位置更新的方法沿着赛车线移动。然后人工智能控制的车辆简单地瞄准兔子；例如，可以使用 "到达 "行为来控制它。由于兔子始终保持在车的前面，它首先开始转弯，确保汽车在正确的位置转向。这特别适合拉力赛游戏，因为它使实施动力滑行相当自然。汽车将在转弯前自动开始转向，如果转弯很严重，它将严重转向，导致物理模拟允许汽车的后端滑出一点儿。

其他开发者已经使用决策工具作为驾驶人工智能的一部分。卡丁车模拟器*Minic Karts*[134]使用模糊决策来代替赛车线。它确定了车辆前方不远处的赛道的左右范围，以及附近的卡丁车，然后用一个手写的马尔科夫状态机来决定下一步该怎么做。

Forza Motorsport[188]使用神经网络来学习如何通过观察人类玩家来驾驶。与游戏一起出货的最终人工智能是开发团队经过数百小时训练的结果。

直到最近，这种技术还很罕见。它们的应用并不广泛，不足以说明它们已经提供了根本性的更好的性能。然而，随着最近在将深度学习应用于游戏方面取得的巨大成功（例如，[43]），神经网络正在整个行业中享受广泛的复兴，并被用于开发几个即将到来的赛车游戏。在接下来的五年里，我们将看到这种情况是否会从热情发展到普遍存在的程度。

14.3.2 寻路和战术AI

随着*Driver*[171]的出现，出现了一种新的驾驶游戏类型。这里没有固定的轨道。游戏设置在城市街道上，目标是赶上或避开其他汽车。汽车可以走它喜欢的任何路线，在逃避警察的追捕时，玩家通常会穿插和折返。单一的、固定的赛车道并不适用于这种游戏。

在这种类型的游戏中，许多敌人的人工智能在逃离玩家时都会遵循一条固定的路径，或者在试图抓住他们时执行一种简单的归位算法。在《侠盗猎车手3》和它的续集中，汽车只在玩家位置周围的几个街区被创造。当警察盯上玩家时，他们会从这个区域聚集，并在适当的位置注入额外的汽车。

然而，由于这种游戏模拟的区域较广，车辆开始需要寻路来寻找其周围的路线，特别是为了抓捕玩家。

利用战术分析找出可能的逃跑路线并加以封锁也是如此。司机使用一个简单的算法来试图包围玩家。基于玩家当前移动方向的战术分析，然后可以要求警车AI进行拦截。然后，警车可以使用战术寻路，在不穿越玩家的路径的情况下到达他们的位置（以避免泄露游戏内容）。

14.3.3 类似驾驶的游戏

用于驾驶游戏的基本方法可以适用于其他一些类型的游戏。

一些极限运动游戏，如类似街机的*SSX*[106]系列和最近的模拟风味的*Steep*[190]，有一个赛车游戏机制（在后者中还有其他模式）。覆盖在赛车系统上的（通常使用与驾驶游戏相同的基于赛车线的人工智能来实现）通常是一个 "技巧 "子游戏，它涉及在跳跃过程中安排动画技巧动作。这些动作可以在赛车线上的预定义点添加（即，一个标记说，当角色到达这个点时，安排一个特定时间的技巧），或者可以由决策系统执行，预测可能产生的空中时间并安排一个具有适当时间的技巧。

未来主义的赛车手，如*《歼灭战》*[162]及其续集，同样是基于相同的赛车AI技术。这类游戏包括武器是很常见的。为了支持这一点，需要额外的人工智能架构来包括瞄准（通常情况下，这不是一个完整的射击方案，因为武器会归位）和决策（车辆可能会放慢速度，让 敌人超越它，以便瞄准他们）。

14.4 实时战略

通过《*沙丘II*》[198]，韦斯特伍德创造了一种新的类型3，并成为出版商投资组合中的主流。虽然它只占游戏总销量的一小部分，但这一类型是PC平台上最强大的类型之一。

实时战略游戏的关键人工智能要求是：

- 寻路
- 团体运动
- 战术和战略AI
- 决策的制定

图14.8显示了一个实时战略（RTS）游戏的人工智能架构。与以前的游戏类型相比，不同游戏之间的差异更大，这取决于所使用的特定游戏元素集。下面的模型应该作为你自己开发的一个有用的起点。

14.4.1 寻路

早期的即时战略游戏，如*《魔兽》：兽人与人类》*[85]和《*命令与征服*》[199]是寻路算法的代名词，因为高效的寻路方法

2. 不要与最初的*《沙丘》*游戏[99]混淆，那是一个相当不伦不类的图形冒险。
3. 一些游戏历史学家将这一类型进一步追溯到*Herzog Zwei*[184]等战略混合游戏，但就人工智能而言，这些早期游戏非常不同。

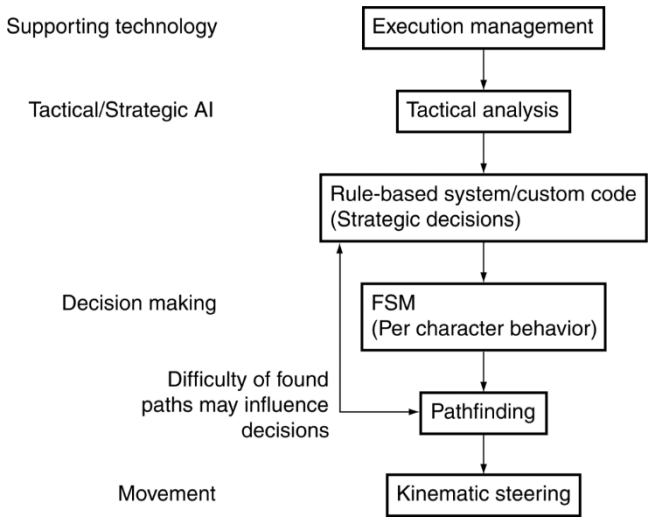


图14.8：RTS游戏的AI架构

是人工智能的主要技术挑战。基于网格的大关卡（通常包括数以万计的独立瓷砖），长距离寻路问题（玩家可以将一个单位直接送过地图），以及许多数十个单位，寻路速度是至关重要的。

尽管大多数游戏不再使用基于瓦片的图形，但底层的表示方法仍然是基于网格的。大多数游戏使用一个有规律的高度数组（称为*高度场*）来渲染景观。然后，这个数组也被用于寻路，给出一个基于网格的规则结构。一些开发者为每个关卡中的常见路径预先计算路线数据。

在《*星际争霸II*》[87]的一些关卡中，可破坏的地形可以在游戏过程中改变导航图，尽管通常变化相对较小：增加或删除少量的连接。像《*英雄连*》[173]这样的游戏将玩家引入了完全可变形的地形，在这种情况下，详尽的预先计算是很困难的。

14.4.2 团体运动

游戏，*如科汉：Ahriman's Gift* [187] 和 *Warhammer：Dark* Omen[4][141]将个人组合成团队，让他们作为一个整体移动。这是用带有预定义模式的编队运动系统来完成的。

4. 战锤》将自己描述为一个角色扮演游戏，因为它的角色发展方面，但在关卡中，它扮演的是一个RTS。

在《*家园*》[172]及其续集中，编队被扩展到三维空间，给人以太空飞行的印象，尽管保持着强烈的上下方向。

Kohan的编队有一定的规模，而在 "家园 "中，任何数量的单位都可以参加。这就需要可扩展的编队，为不同数量的单位提供不同的插槽位置。

现在大多数RTS游戏都使用某种形式的编队。几乎所有的游戏都以一个固定的模式（给定一个固定的阵型中的人物）来定义阵型，作为一个整体移动。在《*全光谱战士*》[157]（另一款以其他方式描述自己的类似RTS的游戏）中，阵型取决于其周围关卡的特征。在一堵墙旁边，小队组成一条线，在提供掩护的障碍物后面，他们会加倍努力，而在空地上他们会形成一个楔子。玩家对阵型的形状只有间接的控制。玩家控制小队移动的位置，而人工智能决定使用的阵型。游戏的不寻常之处还在于，它的阵型只控制人物移动后的最终位置。在移动过程中，各单位独立行动，并可根据要求为对方提供掩护。

14.4.3 战术和战略AI

如果说早期的RTS游戏通过使用寻路技术开创了游戏AI的先河，那么90年代末的游戏在战术AI方面也是如此。影响图是为RTS游戏设计的，最近才开始对其他类型的游戏产生兴趣（通常是以航点战术的形式）。

到目前为止，战术和战略人工智能的输出大多被用来指导寻路。一个早期的例子是《*全面消灭*》[94]，其中的单位在制定路径时考虑到了地形的复杂性；他们正确地在山丘或其他岩石形成的地方移动。同样的分析也被用来指导游戏中的战略决策。

第二个常见的应用是在选择施工地点时。有了显示控制区域的影响图，安全地确定一个重要的建筑设施的位置就变得简单多了。单个建筑只占一个位置，而墙在许多RTS游戏中是一个常见的特征，而且处理起来更加棘手。例如，《战争-工艺》中的墙壁是由关卡设计者预先建造的。在《*帝国地球*》[179]中，人工智能负责建造城墙，使用影响图和空间推理的组合（人工智能试图将城墙置于经济上敏感的建筑物和可能的敌人位置之间）。

在游戏的人工智能圈子里，有很多关于使用战术分析来计划大规模部队演习的讨论--例如，探测敌人阵型中的薄弱点--并部署整个一方的部队来利用这一点。在

某种程度上，每个RTS游戏都会这样做：人工智能会把部队引向它认为的敌人所在的地方，而不是把它们扫到地图上的一个随机位置。这在《*帝国*》等游戏中得到了进一步的发展：*全面战争》（Total War*）[186]等游戏中，人工智能会试图在导弹武器和大炮的射程之外进行机动，然后再将其送回。

对多个侧翼发动攻击。这在代表海战的关卡中变得更加困难，因为盛行风是一个重要的考虑因素。

有可能走得更远，让人工智能根据战术分析和每个单位为利用任何弱点所需的路线来推理可能的攻击策略。我很少看到明显走到这一步的游戏例子。

由于战术分析与RTS游戏有很大关系，第6章的讨论是针对这一类型的。剩下的就是分析你期望你的计算机控制的一方所显示的行为，并选择一套适当的分析方法来执行。

14.4.4 决策

在RTS游戏中，有几个层次的决策需要发生，所以它们几乎总是需要一个多层次的AI方法。

一些简单的决策往往是由个别角色进行的。例如，《魔兽》中的弓箭手会自己决定是守住自己的位置还是前进与敌人作战。

在中级阶段，一个队列或一组角色可能需要做出一些决定。在《全光谱战士》中，当整个小队暴露在敌人的炮火下时，可以决定进行掩护。然后这个决定转给每个角色，让他们决定如何最好地进行掩护（例如，躺在地上）。

大多数棘手的决策发生在游戏中整个一方的层面上。通常会有许多不同的事情同时发生：正确的资源需要收集，研究需要指导，建设应该安排，单位需要训练，部队需要调集防御或进攻。

对于这些要求中的每一项，都要创建一个人工智能组件。这方面的复杂性因游戏的不同而大不相同。例如，为了确定研究顺序，我们可以为每一个进展使用一个数字分数，并选择具有最高值的下一个进展。或者，我们可以使用一种搜索算法，如Dijkstra，来计算出从当前已知技术集到目标技术的最佳路径。

在魔兽等游戏中，这些人工智能模块中的每一个在很大程度上都是独立的。安排资源收集的人工智能并不提前计划为以后的建设工作储备某种资源。它只是分配平衡的努力来收集可用的资源。同样，军事指挥的人工智能也会等待，直到集结了足够的力量才与敌人交战。

像*《魔兽争霸3：混乱之治》*[86]这样的游戏使用了一个中央控制人工智能，它

可以影响一些或所有的模块。在这种情况下，整个人工智能可以决定它想玩一个进攻性的游戏，并且它将为此目的歪曲建设努力、单位训练和军事指挥人工智能。

在RTS游戏中，不同级别的AI通常以军衔命名。一个将军或上校将负责，再往下我们可能有指挥官或中尉，然后是单个士兵。虽然这种命名很常见，但对于每个级别应该叫什么几乎没有一致意见，这可能会让人非常困惑。在一个游戏中，将军

AI可能会控制整场演出。在另一个游戏中，它只是在国王或总统AI的指导下，负责军事行动的AI。

决策技术的选择反映了其他游戏的情况。通常情况下，大部分的决策是通过简单的技术完成的，如状态机和决策树。马尔可夫或其他概率方法在RTS游戏中比其他类型的游戏更常见。军事部署的决策通常是一套简单的规则（有时是基于规则的系统，但通常是硬编码的IF-THEN语句），依赖于战术分析引擎的输出。

14.4.5 蒙特卡洛

多人在线战斗竞技场在2010年代早期崛起，成为最重要的游戏类型之一。起初是"古人的防御"--玩家为《魔兽争霸III》生成的MOD/地图[86]--随着其主要竞争对手《英雄联盟》[175]和其工作室开发的续集《Dota 2》[195]而变得突出起来。尽管人们热衷于在这一流行的新类型中竞争，但很少有其他游戏取得重大的商业成功。尽管淘金热继续发展（尤其是基于英雄的射击游戏和后来的大逃杀热潮），但在撰写本文时，MOBA仍然是电子竞技方面最重要的游戏类型。在20个奖金最高的电子竞技比赛中，18个是MOBA游戏。

这种类型的实时战略的起源延续到了人工智能。游戏中有两种类型的人物：英雄和小兵（根据游戏，这些可能也被称为小兵或暴民）。英雄被设计为由玩家控制。已经开发了人工智能机器人来扮演他们，但他们还不能与人类玩家竞争。小兵可与RTS游戏中的单个单位相媲美。他们或者沿着固定的路线移动，或者潜伏在地图的某个地方（在Dota和英雄联盟中被称为丛林）。巷道小兵通常属于其中一队玩家，并且只会攻击另一队的成员（英雄或小兵）。丛林小兵会攻击任何人，无论是看到还是自己被攻击时。当他们处于攻击状态时，丛林小兵会对最近的玩家进行攻击。

小兵的设计很简单，容易预测。玩这个游戏的技巧之一是操纵小兵，预测他们的攻击，重新引导他们的攻击性。出于这个原因，通常使用最简单的人工智能技术。爬虫不应该以复杂的方式行事。他们的行为应该像由一个状态机实现的那样。

由于小兵的移动受到限制（沿着路线移动，或瞄准敌人），所以很少需要寻路。在受Dota启发的游戏中，团队可能有一个信使：一个自主单位，当被召唤到一个英雄时提供供应商。这个角色可能需要在关卡中导航，但最常见的是以飞行单位的

形式实现，寻路的难度很小。

总的来说，MOBA的人工智能要求通常比它们所衍生的RTS游戏更简单。像RTS游戏一样，可能会有许多小兵在关卡中活动。

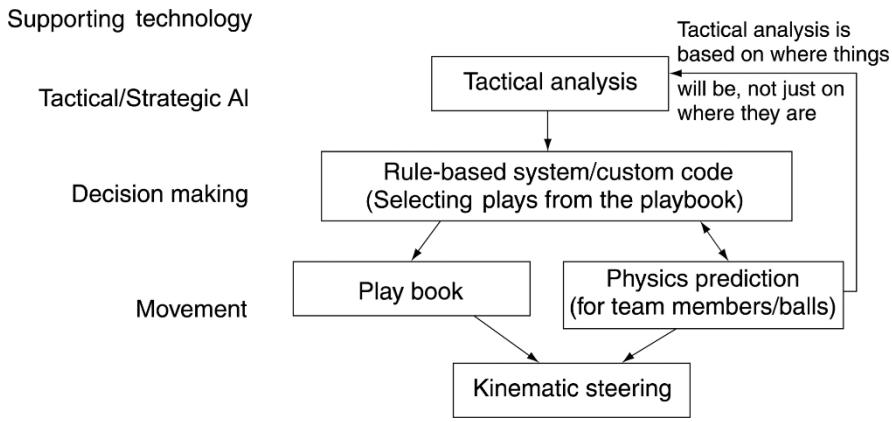


图14.9：体育游戏的AI架构

在同一时间。但他们的人工智能很少需要比基本决策和转向更复杂的东西。

14.5 体育运动

体育游戏的范围很广，从大联盟的体育特许经营权，如《*麦登橄榄球18*》[108]到台球模拟器，如《*世界斯诺克锦标赛*》（最后一年发行的是《102》）。它们的优势在于拥有大量现成的关于良好策略的知识：玩游戏的专业人士。然而，这些知识并不总是很容易编码到游戏中，而且他们还面临着额外的挑战，那就是玩家们希望看到人类水平的能力。

对于团队运动来说，关键的挑战是让不同的角色在考虑到团队其他成员的情况下做出反应。有些运动，如棒球和足球，有非常强的团队模式。第三章中的棒球双打例子（图3.62）就是一个典型的例子。外野手的实际位置将取决于球被击中的位置，但整体的运动模式总是相同的。

因此，体育游戏通常使用某种多层次的人工智能。有高水平的人工智能做出战略决策（通常使用某种参数或行动学习，以确保它挑战玩家）。在较低的层次上，可能有一个协调的运动系统，对游戏事件作出反应的模式。在最低水平上，每个单独的球员将有自己的人工智能来决定如何在整体战略中改变行为。非团队运动，如单打网球，省略了中间层；没有团队需要协调。

图14.9显示了一个典型的体育游戏AI的架构。

14.5.1 物理学预测

许多体育游戏涉及到球在物理学的影响下高速运动。这可能是一个网球，一个足球，或一个台球。在每一种情况下，为了让人工智能做出决定（拦截球或计算出击球的副作用），我们需要能够预测它将如何表现。

在球的动态变化很复杂并且是游戏的一个组成部分的游戏中（例如台球等母球游戏和高尔夫游戏流派），可能需要运行物理学来预测结果。

对于较简单的动力学，如棒球或足球，球的轨迹可以被预先预测。

在每一种情况下，其过程都与我们在第三章中看到的弹丸预测相同。用于枪支的射击方案同样可以用于体育游戏。

14.5.2 游戏手册和内容创建

在团队运动的人工智能中，实现强大的游戏规则是一个常见的问题来源。一个战术手册由一组运动模式组成，一个团队将在某些情况下使用这些模式。有时，战术书指的是整个球队（例如，足球中争球线上的进攻战术），但通常它指的是较小的球员群体（例如，棒球中的挑篮战术）。如果你的游戏不包括像这样久经考验的战术，那么对于购买你的产品的现实世界的游戏迷来说，这将是很明显的。

第三章的协调运动部分包括确保角色在正确时间内移动的算法。这通常需要与同章的编队运动系统相结合，以确保团队成员以视觉上真实的模式移动。

除了驱动游戏手册的技术外，还需要注意允许以某种方式编写游戏。需要有一个很好的内容创建路径，以便将剧本纳入游戏。通常，作为一个程序员，你不会知道所有需要进入最终游戏的战术，你也不希望有测试每个组合的负担。暴露阵型和同步运动是让体育专家为最终游戏创造模式的关键。

14.6 转型战略游戏

基于回合制的战略游戏通常依赖于RTS游戏中使用的相同的人工智能技术。早期的回合制游戏要么是现有棋盘游戏的变种（例如*3D井字棋*[80]），要么是简化的桌面战争游戏（*Computer Bismark*[180]是我早期的最爱之一）。两者都依赖于用于玩棋盘游戏的那种最小化技术（见第九章）。

随着战略游戏变得越来越复杂，每个回合可能的动作数量也大大增加。在最近的游戏中，*如西德梅尔的《文明6》*[113]，几乎有一个

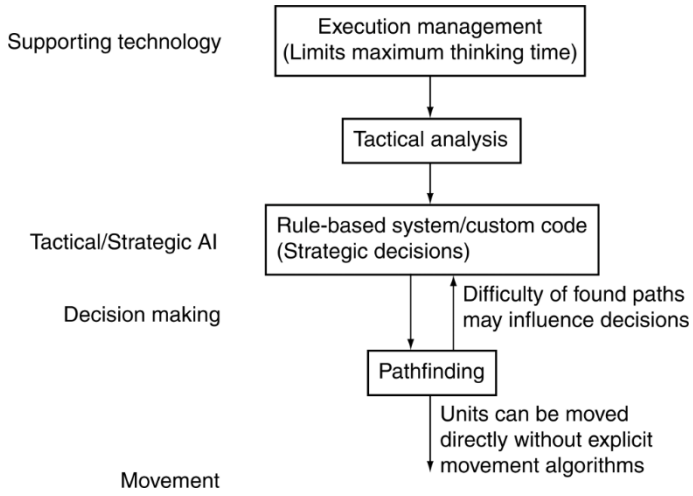


图14.10：回合制战略游戏的AI架构

每一回合向玩家开放的可能移动的数量是无限的，即使每一次移动都是相对离散的（即，一个角色从一个网格位置移动到另一个）。在诸如*《蠕虫》*系列[182]的游戏中，情况甚至更为广泛。在玩家的回合中，他们可以控制每个角色，并以第三人称的方式移动他们（在代表一个回合中可用时间的有限距离内）。在这种情况下，角色可以在任何地方结束。没有任何一种最小化技术可以搜索到如此大小的游戏树。

相反，所使用的技术往往与即时战略游戏中的技术非常相似。基于回合制的游戏往往需要同样种类的角色移动AI。基于回合制的游戏很少需要使用任何一种复杂的移动算法。运动学的移动算法，甚至是直接的位置更新（只是把角色放在它需要的地方）就可以了。在更高的层次上，路线规划、决策、战术和战略人工智能使用相同的技术，并有相同的广泛挑战。

图14.10显示了回合制战略游戏的AI架构。注意这与图14.8中的RTS架构的相似性。

14.6.1 时间

基于回合制的游戏和实时战略游戏之间最明显的区别是计算机和玩家都有一定的时间来进行回合。

鉴于我们并不是要同时做大量的时间密集型的事情（渲染、物理、网络等），对执行管理系统的需求就比较少。使用操作系统线程在几秒钟内运行人工智能进程是很常见的。

然而，这并不是说时机问题不会发生作用。玩家通常可以用无限的时间来考虑他们的行动。如果有大量可能同时进行的行动（如部队移动、经济管理、研究、建设等等），那么玩家可以花时间优化组合，以获得回合的最大收益。为了与这种应用思维水平竞争，人工智能的工作很艰难。其中一些可以通过游戏设计来实现：对游戏的结构做出决定，使其更容易创建人工智能工具，选择容易进行战术分析的关卡的物理属性，创建一个容易搜索的研究树，并使用足够小的回合长度，使每个角色的移动选择数量能够得到管理。然而，这只能让你走到这里。最终将需要一些更实质性的执行管理。

就像RTS游戏一样，通常有一系列不同的决策工具在游戏的特定方面运作：经济系统、研究系统等等。在一个回合制游戏中，值得让这些算法能够快速返回结果。如果有额外的时间，可以要求它们进一步处理。这可能对一个战术分析系统特别有用，因为它可能需要更长的时间来进行计算。

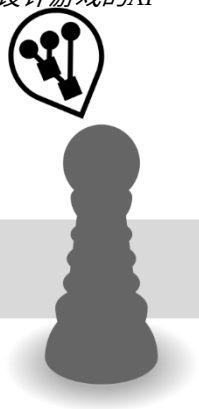
14.6.2 帮助玩家

在回合制游戏中，人工智能的另一个功能（在一些RTS游戏中也有使用，但程度要小得多）是帮助玩家将他们不想担心的决定自动化。

在《*猎户座大师3*》[163]中，玩家可以给人工智能分配一些不同的决策任务。然后，人工智能使用其用于敌军的相同决策基础设施来协助玩家。

以这种方式支持辅助性人工智能涉及建立决策工具，这些工具很少或没有来自更高级别的决策工具的战略输入。例如，如果我们有一个决定在什么星球上建立殖民地的人工智能模块，如果它知道一方打算先向哪个方向扩张，它就能做出更好的决定。如果没有这个决定，它可能会选择一个目前安全的地点，靠近可能爆发战争的地方。

然而，有了这种来自高层决策的输入，当模块被用来协助玩家时，它需要确定玩家的策略是什么。这是很难通过观察来做到的。我不知道有什么游戏曾试图做到这一点。猎户座3》使用无背景决策，所以同一个模块可以用于玩家或敌方。



15

基于AI的游戏类型

大多数游戏在它们使用的人工智能技术和它们想要实现的行为方面都是相当温和的。突出的、显眼的人工智能通常不是好的人工智能。从然而，不时出现的游戏以特定的人工智能技术作为游戏机制。挑战来自于操纵游戏中人物的思想。

作为一个人工智能程序员，如果能看到更多这样的题目就好了，但到目前为止，基于有限的游戏风格的例子还比较少。

本章探讨了以人工智能为中心的游戏玩法的两种选择。所描述的体裁只有少数商业上成功的游戏代表。目前还不清楚是否会有更多的游戏采用完全相同的方法，尽管每一种游戏都可以挖掘出可应用于更多主流类型的相互信任的行为和游戏性。

对于每一种类型的游戏，我将描述一套支持适当的游戏方式的技术。虽然每种类型的具体游戏的一些细节可以在公共领域获得，但许多算法的细节是保密的。即使有了信息，由于游戏数量有限，也很难对哪些是有效的、哪些是无效的进行概括。因此，不可避免的是，这种讨论在某种程度上是推测性的。在本章中，我将尝试指出其他选择。

15.1 教字

教导一个无能的角色按照你的意愿行事，已经在许多游戏中出现过。最初的同类游戏《*生物*》[101]于1996年发布。1现在，这一类型的游戏以2001年的《*黑与白*》[131]最为出名。

少数角色（在《黑与白》中只有一个）有一个学习机制，在玩家的反馈监督下，学习执行它所看到的动作。观察性学习机制观察其他角色和玩家的动作，并试图复制它们。当它复制动作时，玩家可以给予积极或消极的反馈（通常是拍打和搔痒），以鼓励或阻止它再次进行相同的动作。

15.1.1 代表行动

观察性学习的基本要求是能够用离散的数据组合来表示游戏中的行动。然后角色可以自己学习模仿这些动作，可能会有轻微的变化。

通常情况下，动作用三条数据表示：动作本身，动作的可选 对象，以及可选的间接对象。例如，动作可能是 "战斗"、"投掷 "或 "睡眠"；主语可能是 "一个敌人 "或 "一块石头"；间接对象可能是 "一把剑"。并非每个动作都需要一个宾语（例如睡觉），也并非每个有主语的动作都有一个间接宾语（例如扔）。

有些行动可以有多种形式。例如，有可能扔一块石头，也有可能向某个人扔一块石头。因此，投掷动作总是带一个宾语，但也可以选择带一个间接宾语。

在实施过程中，有一个可用的行动数据库。对于每一种类型的动作，游戏都会记录它是否需要一个对象或间接对象。

当一个角色做某事时，可以创建一个行动结构来表示它。行动结构包括行动的类型和游戏中作为对象和间接对象的事物的细节，如果需要的话。

```
动作 （战斗， 敌人， 剑） 动作 （
投掷， 石头） 动作 （投掷， 敌人，
石头） 动作 （睡觉） 。
```

这是表示动作的基本结构。不同的游戏可能会给动作结构增加不同的复杂程度，代表更复杂的动作（例如，需要特定的位置以及间接对象和对象）。

1. Creatures的设计者Steve Grand写了一本关于Crea- tures内部运作的迷人的书[19]。尽管已经有20年的历史，但

它仍然给人以未来主义的感觉，这一点很了不起。

15.1.2 代表世界

除了一个行动之外，人物还需要能够建立起对世界的印象。这使他们能够将行动与背景联系起来。例如，学会吃东西是好事，但当你被敌人攻击时就不一样了。那才是逃跑或战斗的正确时机。

呈现的上下文信息通常是相当狭窄的。大量的情境信息可以提高性能，但会大大降低学习速度。由于玩家负责教导角色，玩家希望在相对较短的时间内看到一些明显的改进。这意味着，学习需要尽可能快，而不导致愚蠢的行为。

通常情况下，角色的内部状态与少数重要的外部数据一起被包含在上下文中。这可能包括与最近的敌人的距离，与安全（家或其他角色）的距离，一天中的时间，观看的人数，或任何其他与游戏相关的数量。

一般来说，如果角色没有被提供一个信息，那么它在做决定时就会有效地忽略它。这意味着，如果一个决定在某些条件下是不合适的，那么这些条件必须向角色说明。

情境信息可以以一系列参数值的形式呈现给角色（一种非常常见的技术），也可以以一组离散事实的形式呈现给角色（很像动作表现）。

15.1.3 学习机制

对于角色来说，各种学习机制都是可能的。已发表的这种类型的游戏已经使用了神经网络和决策树学习；从本书来看，Naive Bayes和re- inforcement学习也可能是有趣的尝试方法。作为一个广泛的工作实例，让我们在本节中详细了解一下使用神经网络。

对于一个神经网络学习算法来说，有两种监督方式的融合：来自观察的强监督和来自玩家反馈的弱监督。

神经网络结构

虽然一系列不同的网络结构可以用于这种类型的游戏，但我将假设使用的是多层感知器网络，如图15.1所示。这是在第7章中实现的，只需做最小的修改就可以应用。

神经网络的输入层从游戏世界中获取背景信息（包括角色的内部参数）。

神经网络的输出层由控制行动类型以及行动的对象和间接对象的节点组成（加上创建行动所需的任何其他信息）。

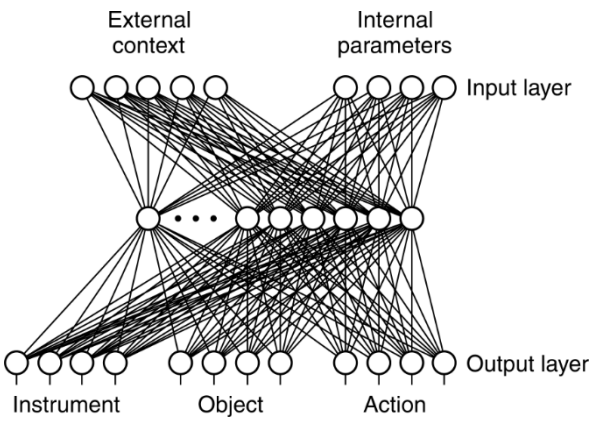


图15.1：生物教学游戏的神经网络结构

独立于学习，网络可以用来为角色做决定，把当前的环境作为输入，从输出中读取行动。

不可避免的是，大多数输出动作都是非法的（当时可能没有这种动作，或者没有这种对象或间接对象可用），但那些合法的动作会被执行。有可能通过在每次提出非法行动时通过一个弱监督的学习步骤来阻止非法行动。在实践中，这可能会在短期内提高性能，但从长远来看，可能会导致病态的问题（见第15.1.4节）。

观察性学习

为了通过观察来学习，角色记录其他角色或玩家的行动。只要这些行动在它的视野之内，它就用这些行动来学习。

首先，角色需要为它所看到的动作找到一个代表，并为当前的环境找到一个代表。然后，它可以用这种输入-输出模式来训练神经网络，可以是一次，也可以是多次，直到网络学会了输入的正确输出。

仅仅通过一次学习算法就有可能使角色的行为产生很小的差异。另一方面，多次迭代可能导致网络忘记它已经学到的有用行为。在学习的速度和遗忘的速度之间找到一个合理的平衡点是很重要的。如果生物的学习速度很慢，玩家会因为不得不重新教导他们的生物而感到沮丧。

读心术

观察学习的一个重要问题是确定与观察到的动作相匹配的背景信息。如果一个不饿的角色观察到一个饥饿的角色在吃东西，那么它就可能学会将吃东西与不饿联系起来。换句话说，你自己的背景信息不能与别人的行动相匹配。

在游戏中，玩家做了大部分的教学工作，这个问题就不会出现。通常情况下，玩家试图告诉角色下一步该怎么做。角色的背景信息可以被使用。

在角色观察其他角色的情况下，其自身的背景信息是不相关的。在现实世界中，当我们看到别人的行动时，不可能了解他们的所有动机和内部过程。我们会试图猜测，或者说是读心术，他们为了执行这个动作一定在想什么。在游戏的情况下，我们能够使用观察到的人物的背景信息，而不需要改变。

虽然可以增加一些不确定性来代表了解他人想法的难度，但实际上这并不能使角色看起来更可信，而且会大大减慢学习速度。

反馈学习

为了通过反馈来学习，角色记录了它为其最近的每一个输入所创造的输出的列表。这个列表至少需要延伸到几秒钟前。

当玩家的反馈事件到来时（例如，一个巴掌或挠痒痒），我们没有办法确切地知道玩家对哪个动作感到高兴或生气。这就是人工智能中典型的 "功劳分配问题"：在一系列的行动中，我们如何判断哪些行动有帮助，哪些没有帮助？

通过保留几秒钟的输入-输出对的列表，我们假设用户的反馈与整个系列的行动有关。当反馈到达时，神经网络被训练（使用弱监督方法）以加强或削弱这段时间内的所有输入-输出对。

随着输入-输出对的时间越来越久远，逐渐减少反馈的数量通常是有用的。如果角色收到反馈，很可能是一秒钟前进行的动作（时间再短，用户仍然会拖着光标去拍打或搔痒）。

15.1.4 可预测的心智模式

这类游戏的人工智能有一个共同的问题：很难理解玩家的行为会对角色产生什么影响。在游戏的某一时刻，角色似乎很容易学习，而在其他时刻，它似乎完全不理会玩家。

运行角色的神经网络太过复杂，任何玩家都无法正确理解，而且它经常出现做错事的情况。

玩家的期望是制造好的人工智能的一个重要部分。正如第二章所讨论的，一个角色可以做一些非常聪明的事情，但如果这不是玩家所期望看到的，它往往会显得很愚蠢。

在上面的算法中，来自玩家的反馈分布在一些输入-输出动作中。这是意外学习的一个常见来源。当玩家给出反馈时，他们无法说出他们正在评判哪个具体的行动，或行动的一部分。

例如，如果一个角色拿起一块石头并试图吃掉它，玩家就会给它一巴掌，让它知道石头是不能吃的。几分钟后，这个角色试图吃一个有毒的蟾蜍。再一次，玩家给了它一巴掌。在玩家看来，他们是在教这个角色什么是好的，什么是坏的，这是合乎逻辑的。然而，这个角色只明白 "吃石头 "是不好的，"吃蟾蜍精 "是不好的。因为神经网络在很大程度上是通过归纳来学习的，玩家只是教角色吃东西是不好的。这个生物慢慢地挨饿，从未尝试过吃任何健康的东西。它从来没有机会因为吃对了东西而被玩家嘲笑。

这些混合的信息往往是角色行为突然急剧恶化的根源。当玩家期望角色的行为越来越好时，往往会迅速达到一个高点，而且偶尔会出现恶化的情况。

没有解决这些问题的一般程序。在某种程度上，这似乎是该方法的一个弱点。然而，通过使用 "本能"（即表现相当好的固定默认行为）与大脑的学习部分，它可以在一定程度上得到缓解。

本能

本能是一种内在行为，在游戏世界中可能是有用的。例如，一个角色可以被赋予吃饭或睡觉的本能。这些都是有效规定的输入-输出对，永远不会被完全忘记。它们可以通过监督下的学习过程定期得到强化，或者它们可以独立于神经网络，用于产生偶尔的行为。无论哪种情况，如果本能得到玩家的强化，它就会成为角色学习行为的一部分，并会更经常地被执行。

一个人物的脑死亡

有一些学习的组合会让一个神经网络在很大程度上无法做任何有意义的事情。在《生物》和《黑与白》中，有可能使一个被教导的角色变得无能为力。

虽然有可能解救这样的人物，但其中的游戏性是不可预测的（因为玩家不知道他们的反馈的真正效果），而且很乏味。B-

因为这似乎是所使用的人工智能的一个不可避免的结果，值得在游戏设计中考虑这一结果。

15.2 集群和放牧游戏

简单的放牧模拟器自20世纪80年代以来一直存在，但在21世纪初出现了少数更成功的游戏，推动了这一技术的发展。这些游戏涉及在一个（通常是敌对的）游戏世界中移动一群角色。*Herdy Gerdy*[97]是最发达的，尽管它在商业上并不顺利。*皮克敏*》[153]、《*皮克敏2*》[155]，以及《*奇异世界*》的某些关卡：*Munch's Oddysee*[156]使用了类似的技术。

相对而言，大量的角色都有简单的个体行为，这些行为会引起更大范围的出现。一个角色会与其他同类成群结队，尤其是在面临危险时，并以某种方式对玩家做出反应（要么从他们身边跑开，就像他们是捕食者一样，要么跟在他们后面）。角色会做出反应，从敌人手中跑开，并进行基本的转向和避开障碍物的操作。不同类型的角色通常被设置在一个食物链，或生态系统中，玩家试图保持一个或多个物种的猎物安全。

15.2.1 创作的生物

每个单独的角色或生物都由一个简单的决策框架组成，其中包含了一系列的引导行为。决策过程需要以一种非常简单的方式对游戏世界做出反应：它可以被实现为一个有限状态机（FSM），甚至是一个决策树。图15.2中给出了一个简单的羊类生物的有限状态机。

同样，转向行为也可以相对简单。因为这类游戏通常都是在户外进行的，限制条件很少，所以转向行为可以在本地进行，并且无需复杂的仲裁就可以组合起来。图15.2显示了以FSM中每个状态的名称运行的转向行为（吃草可以实现为缓慢的游走，不时地暂停进食）。

除吃草外，每个转向行为要么是基本的目标寻求行为之一（如逃离），要么是目标寻求行为的简单总和（如成群）。更多细节见第三章的运动。

在放牧游戏中，很少需要复杂的生物人工智能，即使是捕食者。一旦生物能够

在游戏世界中自主导航，它通常就会太聪明，不容易被玩家操纵，游戏的意义就会受到影响。

15.2.2 为互动性调整转向

在动画的模拟中，或者在游戏的背景效果中，流畅的转向运动增加了可信度。然而，在一个互动的环境中，玩家往往不能快速反应，以满足他们的需求。

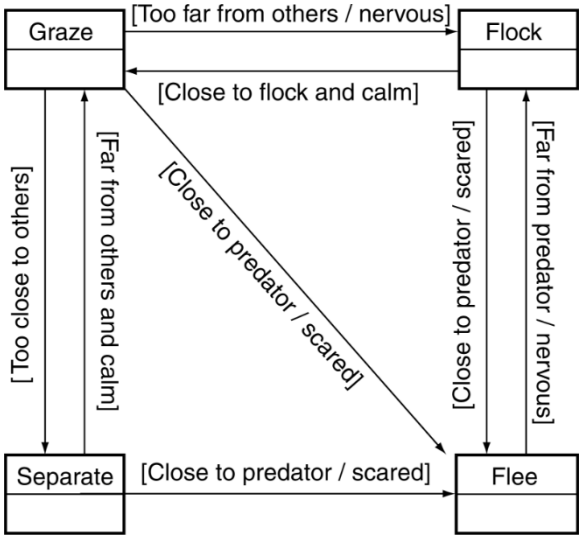


图15.2：一个简单生物的有限状态机

一个群体的运动。例如，当鸟群开始分离时，很难以足够的速度将它们圈起来，使它们重新聚在一起。为角色提供这种运动能力会影响到游戏设计的其他方面。

为了避免这个问题，转向行为通常被参数化，以减少流动性。
人物的行动是小规模的，他们形成凝聚力的愿望也会增加。

在人物的运动中加入停顿，可以减缓他们的整体进度，让玩家围住他们，操纵他们的行动。这可以通过降低他们的运动速度来实现，但这往往看起来很做作，而且当他们直接被追赶时，不允许全速、连续的运动。匆匆忙忙地移动也给生物带来了诡秘和紧张的气氛，这可能是有益的。

就速度和凝聚力而言，减少移动角色的惯性很重要。虽然鸟群模拟中的鸟类通常有很大的惯性（它们需要花很大的力气来改变速度或方向），但被玩家操纵的生物需要被允许突然停止并向新的方向移动。

在高惯性下，一个导致生物改变方向的决定将对许多帧产生影响，并可能影响整个群体的运动。在低惯性的情况下，同样的决定很容易被逆转，而且后果也比较小。这可能会使行为不那么可信，但对玩家来说，控制起来更容易（因此也更不令人沮丧）。

有趣的是，有一些真实世界的国际牧羊比赛，需要多年的训练。牧养几只真正的羊是很难的。一个游戏也许不应该要求同样的技能水平才可以玩。

15.2.3 转向行为的稳定性

当一群生物的决策和指导行为变得更加复杂时，往往会出现这样的情况：这群生物似乎不能理智地独立行动。这通常表现为行为的突然变化和不稳定人群的出现。这些不稳定因素是由决策在群体中的传播引起的，通常在每一步都会被放大。

例如，一群羊可能正在安静地吃草。其中一只羊离它的邻居太近了，邻居就会让开，导致另一只羊移动，如此反复。

正如在所有决策中，需要一定程度的滞后性以避免不稳定。一只羊可能很满意别人离它很近，但只有当别人离它很远时，它才会向别人移动（即形成羊群）。这就提供了一个距离范围，在这个范围内，羊对邻居根本就没有反应。

然而，在一群不同的生物中会出现一种不稳定性，不能简单地用个体行为的滞后性来解决。

一群生物可以表现出振荡，因为每个人都会引起不同群体的行为变化。例如，一个捕食者可能会追赶一群猎物，直到它们离开范围。猎物停止移动，它们是安全的，并且有一个延迟，直到捕食者停止。现在捕食者更近了，猎物又开始移动了。这种震荡很容易失控，看起来很假。只涉及两个物种的周期可以很容易地进行调整，但只有当几个物种在一起时才会出现的周期就很难调试了。

大多数开发商在游戏关卡中把不同的生物放在相隔一定距离的地方，或者一次只使用少数几个物种，以避免许多物种同时出现时的不可预测性。

15.2.4 生态系统设计

通常情况下，放牧游戏中有不止一种生物，正是所有物种的相互作用使玩家的游戏世界变得有趣。作为一种类型，它为有趣的策略提供了很多空间：一个物种可以用来影响另一个物种，这可以导致游戏中的谜题得到意外的解决。在最基本的情况下，物种可以被安排成一个食物链，玩家往往被赋予保护一群脆弱的生物的任务。

在设计游戏的食物链或生态系统时，可以引入不需要的，以及积极但意外的效果。为了避免在游戏关卡中出现崩溃，即所有的生物都被迅速吃掉，需要遵循一些基本准则。

食物链的规模

食物链应该在你的主要生物之上有两个层次，可能还有一个层次。这里，"主要生物"是指玩家通常关注的生物。

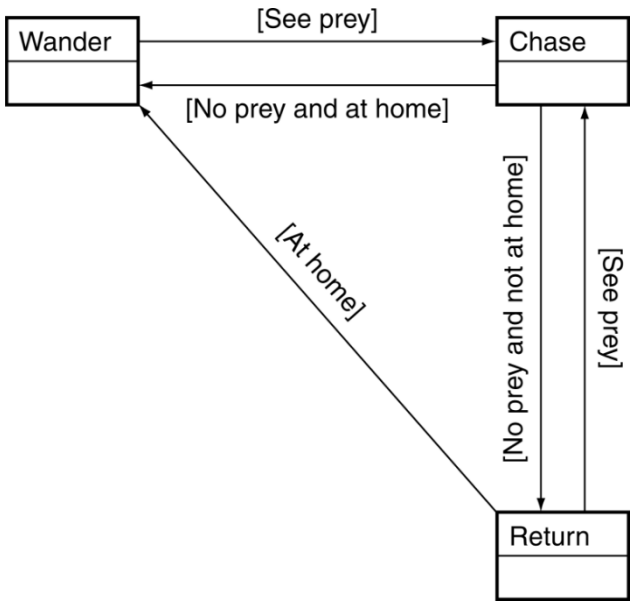


图15.3：单一捕食者的简单行为

驱赶。在生物上面有两个层次，允许捕食者被其他捕食者反击（就像老鼠杰瑞利用斗牛犬斯派克来摆脱猫汤姆的困境一样）。再多的层次，就会有 "有用的捕食者 "不在身边帮忙的风险。

行为的复杂性

食物链中较高的生物应该有更简单的行为。因为玩家在直接影响其他生物的行为，所以随着中间人数量的增加，控制变得更加困难。移动一群生物已经很困难了。用这群生物来控制另一个生物的行为是增加了难度，然后反过来用这个生物来影响另一个生物--这真是一个很高的要求。当你到达食物链的顶端时，这些生物需要有非常简单的行为。图15.3显示了一个单一捕食者的高级行为样本。

食物链中较高的生物不应该成群工作。这源于前面的准则：一起工作的生物群体几乎总是有更复杂的行为（即使它们单独是很简单的）。例如，尽管Pikmin中的许多捕食者以群体形式出现，但他们的行为很少是协调的。它们只是作为个体行动

。

感官极限

所有的生物都应该有明确定义的半径来注意到事物。为生物的注意能力固定一个限度，可以让玩家更好地预测它的行动。将捕食者的视野限制在10米，可以让玩家在11米的距离上带着羊群过去。这种预先支配能力在复杂的生态系统中是很重要的，因为能够预测哪些生物会在什么时间做出反应对战略很重要。由此可见，现实意义上的模拟通常不适合于这种游戏。

运动范围

生物不应该自行移动很远。一个生物的腹地越小，关卡设计者就越能把关卡组合起来。如果一个生物可以随机游荡，那么它就有可能在玩家到达之前发现自己在捕食者旁边。玩家在到达一个地点时发现羊群已经被吃掉了，这是不可取的。限制生物的范围（至少在它们被玩家影响之前）也可以通过设置游戏世界的边界（如栅栏、门或大门）来实现。然而，通常情况下，当玩家不在附近时，这些生物只是睡觉或站在附近。

把它放在一起

正如所有的人工智能一样，获得一个可玩的游戏的最重要的部分是建立和调整角色。牧群游戏的突发性质意味着在你能够建立和测试它之前，不可能预测确切的行为。

提供良好的游戏体验通常需要对游戏中的生物的行为进行严格限制，为了可玩性而牺牲一些可信性。

参考文献

书籍、期刊、论文和网站

[1] Bruce D. Abramson.*双人游戏的预期-收益模型*.博士论文，哥伦比亚大学，美国纽约，1987年。AAI8827528.

[2] A.Adonaac.程序性地牢生成算法。*Gamasutra*，2015年3月。

[3] Nada Amin和Ross Tate。Java和Scala的类型系统是不健全的：空指针的存在危机。In *Proceedings of the 2016 ACM SIGPLAN International Confer- ence on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2016, pages 838-848, New York, NY, USA, 2016.ACM.

[4] Boost。 Boost C++, 2018. URL: **https://www.boost.org/**

[5] Robert Bridson.任意维度下的快速泊松磁盘采样。In *ACM SIG- GRAPH 2007 Sketches*, SIGGRAPH '07, New York, NY, USA, 2007.ACM.

[6] David S Broomhead和David Lowe.径向基函数、多变量函数插值和自适应网络。技术报告，皇家信号和雷达研究所Malvern（英国），1988。

[7] James J. Buckley 和 Esfanfiar Eslami.*模糊逻辑和模糊集简介*》。Springer, 2002.

[8] B. 杰克-科普兰。*Colossus: The Secrets of Bletchley Park's Code-Breaking Computers*.牛津大学出版社，2010年。

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.*算法入门*》。麻省理工学院出版社，第三版，2009年。

[10] E.W. Dijkstra.A note on two problems in connexion with graphs.*Numerische Mathematik*, 1(1):269-271, 1959.

[11] 理查德-杜斯坦菲尔德算法235：随机互换。*Commun.ACM*, 7(7), July 1964.

[12] 大卫-埃伯里。*Game Physics*.CRC出版社，第二版，2010年。

[13] 大卫-埃伯里在坐标系之间转换。URL: www.geometrictools.com/Documentation, 2014.

[14] Christer Ericson.*Real-Time Collision Detection*.摩根-考夫曼出版社，2005。

[15] Joseph C. Giarratano和Gary D. Riley. *专家系统：原理与编程*》。课程技术公司，第四版，2004年。

[16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio.Deep sparse rectifier neural net- works.在*第十四届人工智能和统计学国际会议上*，第315-323页，2011年6月。

[17] Rafael C. Gonzalez和Richard E. Woods。*数字图像处理*》。Prentice Hall，第二版，2002。

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.*Deep Learning*.MIT Press, 2017.

[19] 史蒂夫-格兰德。*创造：生命和如何创造它*。哈佛大学出版社，2003年。

[20] P.E. Hart, N. J. Nilsson, and B. Raphael.A formal basis for the heuristic determination of minimum cost paths.*IEEE Transactions on Systems Science and Cybernetics*, 4(2):100-107, 1968.

[21] Jeff Heaton.*人类的人工智能，第三卷：深度学习和神经网络*。CreateSpace独立出版平台，2015年。

[22] 唐纳德-赫伯。*The Organization of Behavior*.Wiley & Sons, New York, 1949.

[23] 玛丽-希利尔。*Automata & Mechanical Toys：An Illustrated History*.Bloomsbury Books, 1988.

[24] Feng-hsiung Hsu, Murray S. Campbell, and A. Joseph Hoane, Jr.深蓝系统概述。在*第九届国际超级计算会议论文集*，ICS '95，第240-244页，美国纽约，1995。ACM.

[25] id软件。Quake-III-Arena, 2018. URL: **https://github.com/id-Software/Quake-III-Arena**

[26] Roberto Ierusalimschy.*Programming in Lua*. lua.org, Distributed by Ingram (US) and Bertram Books (UK), 4th edition, 2016.

[27] Inkle工作室。 ink - inkle的叙述性脚本语言。

URL: https://www.inklestudios.com/ink/

[28] Alex D. Kelley, Michael C. Malin, and Gregory M. Nielson.使用溪流侵蚀模型的地形模拟。在*第15届计算机图形和交互技术年会论文集*，SIGGRAPH '88，第263-268页，美国纽约，1988。ACM.

[29] S.Koenig, M. Likhachev, and D. Furcy.终身规划A*.*Artificial Intelligence Journal*, 155:93-146, 2004.

[30] Richard E. Korf.Depth-first iterative-deepening：一个最佳的可接受树形搜索。*人工智能*，27（1）：97-109，1985。

[31] M. Kourkolis少将。*APP-6陆基系统的军事符号*。北约军事标准化机构（MAS），1986年。

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton.用深度卷积神经网络进行图像分类。在*第25届国际神经信息处理系统会议论文集-第一卷*，NIPS'12，第1097-1105页。Curran Associates Inc., 2012.

[33] Joseph B. Kruskal.论图的最短生成子树和旅行推销员问题.*美国数学学会论文集*，7(1)：48-50，1956。

[34] P.J. van Laarhoven 和 E. H. Aarts.*Simulated Annealing： Theory and Applications*.Springer, 1987.

[35] 斯蒂芬-拉维尔。PuzzleScript - 一个开源的HTML5益智游戏引擎。URL: **https://www.puzzlescript.net/**

[36] Eric Lengyel.*游戏引擎开发的基础，第一卷：数学*。Terathon Software LLC, Lincoln, California, 2016.

[37] Aristid Lindenmayer.发展中的细胞相互作用的数学模型：第一和第二部分。*理论生物学杂志*, 18, 1968.

[38] Max Lungarella, Fumiya Iida, Josh Bongard, and Rolf Pfeifer.*人工智能的50年*。Springer Science & Business Media, 2007.

[39] W.S. McCulloch和W. Pitts.神经活动中所包含的思想的逻辑计算。*数学生物物理学公报》*，5，1943。

[40] Ian Millington.*Game Physics Engine Development*.CRC出版社，第二版，2010年。

[41] Melanie Mitchell.*遗传算法简介*》。麻省理工学院出版社，第二版，1998年。

[42] 威廉-米切尔。*The Logic of Architecture*.麻省理工学院出版社，1990年。

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller.*arXiv:1312.5602 [cs]*, 2013年12月，用深度强化学习玩Atari。

[44] 格雷厄姆-纳尔逊。 信息 7, 2018. URL: **http://inform7.com/**

[45] A.Newell and H. A. Simon. 计 算 机 科 学 作 为 实 证 调 查 ： 符 号 和 搜 索 。 *Communications of the Association for Computing Machinery*, 19(3), 1976.

[46] Gerard O'Regan.*A Brief History of Computing*.Springer Science & Business Media, 2012.

[47] 肯-佩林一个图像合成器。*ACM SIGGRAPH Computer Graphics*, 19(3):287- 296, July 1985.

[48] Dan Pilone and Neil Pitman.*UML 2 in a Nutshell*.O'Reilly and Associates, 2005.

[49] 贾米-皮特曼The Pac-Man dossier.*Gamasutra*，2009年2月。

[50] R.C. Prim.最短连接网络和一些概括。*The Bell System Technical Journal*, 36(6):1389-1401, Nov 1957.

[51] Craig Reynolds.自主角色的转向行为。在*《1999年游戏开发者大会论文集》*中，第763-782页。Miller Freeman游戏集团，1999年。

[52] Timothy J. Ross.*Fuzzy Logic with Engineering Applications*.Wiley-Blackwell, 4th edition, 2016.

[53] Stuart Russell.高效的有内存限制的搜索方法。In *In ECAI-92*, pages 1-5.Wiley, 1992.

[54] Stuart Russell和Peter Norvig.*Artificial Intelligence：A Modern Approach*.培生教育，第三版，2015。

[55] A.L. Samuel.使用跳棋游戏的机器学习的一些研究。*IBM研究与发展杂志*，3（3）：210-229，1959年7月。

[56] Philip J. Schneider和David Eberly.*Geometric Tools for Computer Graphics*.摩根-考夫曼出版社，2003年。

[57] Robert Sedgewick 和 Kevin Wayne.*算法*》。Addison-Wesley Professional，第四版，2011年。

[58] Tanya X. Short和Tarn Adams，编辑。*游戏设计中的程序生成*》。A K Peters/CRC Press, 2017.

[59] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray

Kavukcuoglu, Thore Graepel, and Demis Hassabis。用深度神经网络和树状搜索掌握围棋游戏。*自然》*，529:484，2016年1月。

[60] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis.在没有人类知识的情况下掌握围棋游戏。*自然》*，550(7676):354-359, 2017.

[61] 威廉-斯潘诺尔.*Game Theory 101: The Complete Textbook*.CreateSpace独立出版平台，2011年。

[62] Anthony Stentz.未知和动态环境下的最佳和有效路径规划。*International Journal of Robotics and Automation*, 10:89-100, 1993.

[63] Anthony Stentz.用于实时重新规划的重点D*算法。在*国际人工智能联合会议论文集中*，第1652-1659页，1995年。

[64] George Stiny和James Gips.形状语法与绘画和雕塑的生成规范。在*信息处理中：IFIP大会论文集》*，第71卷，第1460-1465页，1971年1月。

[65] 布莱恩-斯托特。聪明的举动：智能寻路。*游戏开发者杂志*，第28-35页，1996年10月。

[66] Peter Su and Robert L. Scot Drysdale.A comparison of sequential Delaunay triangu- lation algorithms.In *The Proceedings of the Eleventh Annual Symposium on Computa- tional Geometry*, pages 61-70.Association for Computing Machinery, 1995.

[67] 史蒂芬-塔德里斯。*博弈论：An Introduction*.普林斯顿大学出版社，2013年。

[68] 杰拉尔德-特索罗时差学习和TD-gammon.*Commun.ACM*, 38(3):58-68, March 1995.

[69] TIOBE.TIOBE指数，2018。URL: **https://www.tiobe.com/tiobe-index/**

[70] 阿兰-M-图灵。计算机械和智能。*Mind*, 59, 1950.

[71] 美国陆军步兵学校。*FM 7-8步兵步枪排和班*。陆军部，华盛顿特区，1992年。

[72] 美国陆军步兵学校。*FM 3-06.11 城市地形的联合武器行动*。陆军部，华盛顿

特区，2002年。

[73] Gino van den Bergen.*交互式三维环境中的碰撞检测*.摩根-考夫曼出版社，2003年。

[74] 阿德尔海德-沃斯库尔。*启蒙运动中的机器人：机械师、工匠和自我的文化*。芝加哥大学出版社，2013年。

[75] Frederick M. Waltz 和 John W. V. Miller.使用有限状态机的高斯模糊的有效算法。在*SPIE机器视觉系统检查和计量第七届会议的论文集中*，1998年。

[76] David H. Wolpert and William G. Macready.没有免费午餐的优化定理. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

[77] Ian Wright和James Marshall.在更少的处理器时间里有更多的人工智能："以自我为中心的 "人工智能。*Gama- sutra*, 2000年6月。

游戏

本节给出了书中提到的游戏的更全面信息。游戏提供了它们的开发商和出版商、游戏出版的平台和发行的年份。它们按开发商名称排序，并沿用了全书的引用风格。

开发商往往会经常改变他们的名字，所以这个列表使用的是游戏开发时的开发商的名字。许多游戏最初是在一个或两个平台上发布的，后来被移植到其他平台上：这个列表指出了游戏发布的原始平台。如果游戏在两个以上的平台发布，则表示为"多平台"。

[78] 2015年公司。荣誉勋章：盟军突击》。由电子艺术公司出版，2002年。个人电脑。

[79] 动视出版公司。异形与掠食者》。由动视出版公司出版，1993年。SNES和街机。

[80] 雅达利。3D井字游戏。由Sears, Roebuck and Co.出版，1980年。雅达利2600。

[81] 湾12游戏。矮人堡垒。由Bay 12 Games出版，2006年。多平台。

[82] 伊恩-贝尔和大卫-布拉本。精英。由Acornsoft有限公司出版，1984年。BBC Micro.

[83] Bioware公司。Neverwinter Nights.由Infogrames, Inc.出版，2002年。个人电脑。

[84] 怪异的创作。一级方程式。由Psygnosis出版，1996年。PlayStation和PC。

[85] Blizzard Entertainment Inc.魔兽争霸：兽人和人类。由暴雪娱乐公司出版，1994年。个人电脑。

[86] Blizzard Entertainment, Inc. 魔兽争霸3：混沌之治。由暴雪娱乐公司出版，2002年。个人电脑。

[87] 暴雪娱乐公司。星际争霸II：自由之翼》。由暴雪娱乐公司出版，2010年。个人电脑。

[88] 暴雪娱乐公司和暴雪北方公司。暗黑破坏神》。由暴雪娱乐公司出版，1996年。多平台。

[89] Bohemia Interactive Studio s.r.o. ArmA: Combat Operations。由Atari公司出版，2006年。个人电脑。

[90] 牛蛙制作有限公司。地下城守护者。由电子艺术公司出版，1997年。个人电脑。

[91] Bungie软件。晕。由微软游戏工作室出版，2001年。XBox.

[92] Bungie软件。光环2》。由微软游戏工作室出版，2004年。XBox。

[93] Bungie软件。光环3》。由微软游戏工作室出版，2007年。XBox 360.

[94] Cavedog娱乐公司。全面毁灭》。由GT互动软件欧洲有限公司出版，1997年。个人电脑。

[95] 核心设计有限公司。古墓丽影》。由Eidos Interactive Inc.出版，1996年。多个平台。

[96] 核心设计有限公司。古墓丽影III：劳拉的冒险》。由Eidos Interactive Inc.出版，1998年。多个平台。

[97] 核心设计有限公司。赫迪-格迪。由Eidos互动有限公司出版，2002年。PlayStation 2.

[98] Criterion软件。倦怠症。由Acclaim Entertainment出版，2001年。多个平台。

[99] Cryo互动娱乐公司。沙丘》。由维珍互动娱乐公司出版，1992年。多个平台。

[100] Crytek。Far Cry.由UbiSoft出版，2004年。个人电脑。

[101] 网络生命科技有限公司。生物。由Mindscape Entertainment出版，1997年。个人电脑。

[102] 暗能量数字有限公司。2011年世界斯诺克锦标赛实况：世界斯诺克锦标赛。由暗能量体育有限公司出版，2011年。PlayStation 3, Xbox 360.

[103] Digital Extremes, Inc.和Epic MegaGames, Inc.虚幻。由GT互动软件公司出版，1998年。多平台。

[104] DMA设计。Grand Theft Auto 3.由Rockstar Games出版，2001年。PlayStation 2.

[105] 代尔姆克斯。部落II。由Sierra On-Line出版，2001年。个人电脑。

[106] 加拿大电子艺术公司。SSX。由电子艺术公司出版，2000年。PlayStation 2.

[107] Electronic Arts Tiburon.Madden NFL 2005。由电子艺界出版，2004年。多个平台。

[108] 电子艺界Tiburon公司。Madden NFL 2018。由电子艺界出版，2017年。多个平台。

[109] 伊莱克斯尔工作室有限公司共和国：革命》。由Eidos公司出版，2003年。个人电脑。

[110] Elixir工作室有限公司。邪恶的天才。由塞拉娱乐公司出版，2004年。个人电脑。

[111] Epic Games, Inc., People Can Fly, and Sp. z o.o. Fortnite：大逃杀》。由Epic Games, Inc.发布，2017年。多平台。

[112] Firaxis Games.席德梅尔的《文明III》。由Infogrames出版，2001年。个人电脑。

[113] Firaxis Games.席德梅尔的文明6》。由2K游戏公司出版，2016年。多平台。

[114] Firaxis Games East, Inc.席德梅尔的《文明4》。2K游戏公司出版，2005年。多个平台。

[115] FromSoftware, Inc.黑暗之魂》，2011年。PlayStation 3, Xbox 360.

[116] 来自软件公司。血源》。由索尼电脑娱乐美国有限公司出版，2015年。PlayStation 4.

[117] FromSoftware, Inc.黑暗之魂III》，2016年。多平台。

[118] 边疆发展有限公司。精英：Dangerous。由Frontier Developments plc出版，2015年。多平台。

[119] Hello Games Ltd.无人的天空》。由Hello Games Ltd.出版，2016年。多平台。

[120] 浩然正气。地降。由Holospark出版，2017年。多平台。

[121] id Software, Inc.狼人杀3D》。由动视、Apogee和GT互动出版，1992年。个人电

脑。

[122] id Software, Inc.毁灭。由id软件公司出版，1993年。个人电脑。

[123] id Software, Inc.雷神之锤II》。由动视公司出版，1997年。个人电脑。

[124] id Software, Inc.雷神之锤III》。由动视公司出版，1999年。多平台。

[125] id Software, Inc.Doom 3.由动视出版，2004年。个人电脑。

[126] Incog Inc娱乐公司。下坡统治。由Codemasters出版，2003年。PlayStation 2.

[127] 离子风暴。Deus Ex.由Eidos Interactive出版，2000年。个人电脑。

[128] K-D实验室游戏开发。周边。由1C公司出版，2004年。个人电脑。

[129] 科乐美公司。金属齿轮固体。由Konami公司出版，1998年。PlayStation。

[130] 林登研究公司。血与桂冠。由林登研究公司出版，2014年。iPad。

[131] 狮头工作室有限公司。黑与白。由电子艺术公司出版，2001年。个人电脑。

[132] 展望玻璃工作室，公司。盗贼》：黑暗计划。由Eidos, Inc.出版，1998年。个人电脑。

[133] LucasArts娱乐公司有限责任公司。星球大战：第一集--赛车。由LucasArts Entertainment Company LLC出版，1999年。多平台。

[134] 躁动的媒体制作。Manic Karts.由Virgin Interactive Entertain- ment出版，1995年。个人电脑。

[135] Maxis Software, Inc.模拟城市》。由Infogrames Europe SA出版，1989年。PC / Mac。

[136] Maxis Software, Inc.模拟人生》。由电子艺术公司出版，2000年。个人电脑。

[137] Maxis Software, Inc.孢子。由电子艺术公司出版，2008年。Macintosh, PC。

[138] 埃德蒙-麦克米伦和弗洛里安-希姆尔斯。以撒的束缚》。由Edmund McMillen出版，2011年。个人电脑。

[139] 格伦-维奇曼（Glenn Wichman） 迈克尔-C-托伊（Michael C. Toy）和肯-阿诺德（Ken Arnold）。流氓。自由软件，1982年。VAX-11。

[140] Midway Games West, Inc.吃豆人》。由Midway Games West, Inc.出版，1979年。商场。

[141] 敏锐的洞察力。战锤：黑暗之兆。由电子艺术公司出版，1998年。PC和

PlayStation。

[142] Mojang AB.Minecraft.由Mojang AB出版，2010年。多平台。

[143] 巨石制作公司。没有人可以永远活着2。由Sierra出版，2002年。个人电脑。

[144] 巨石制作公司（Monolith Productions）。F.E.A.R. 由维旺迪环球游戏公司出版，2005年。个人电脑。

[145] 巨石制作公司（Monolith Productions）。F.E.A.R. 2: Project Origin.由华纳兄弟互动娱乐公司出版，2009年。PC / Mac。

[146] Mossmouth, LLC.Spelunky。由Mossmouth, LLC出版，2009年。个人电脑。

[147] MPS实验室。文明》。由MicroProse软件公司出版，1991年。多个平台-形式。

[148] 顽皮狗公司。雅克和达克斯特：前驱者的遗产。由SCEE有限公司出版，2001年。PlayStation 2.

[149] 顽皮狗公司。最后的我们》。由索尼计算机娱乐美国有限公司出版，2003年。PlayStation 3.

[150] Niantic, Inc., Nintendo Co., Ltd., and The Pokémon Company.Pokémon Go。由Niantic, Inc.发布，2016年。多个平台。

[151] Nicalis, Inc.以撒的捆绑》（The Binding of Isaac：重生。由Nicalis, Inc.出版，2014年。多平台。

[152] 任天堂股份有限公司和系统研究与开发有限公司。超级马里奥兄弟》，由任天堂有限公司出版，1985年。任天堂娱乐系统。

[153] 任天堂娱乐，分析与发展。皮克敏。任天堂公司出版。Ltd., 2001.GameCube.

[154] 任天堂娱乐，分析与发展。超级马里奥阳光》。Pub- lished by Nintendo Co.Ltd., 2002.GameCube.

[155] 任天堂娱乐，分析与发展。Pikmin 2.由Nin- tendo有限公司出版。Ltd., 2004.GameCube。

[156] 奇异世界的居民。奇异世界》：Munch's Oddysee.由微软游戏工作室出版，1998年。XBox。

[157] Pandemic工作室。全光谱战士。由THQ出版，2004年。个人电脑和XBox。

[158] 游戏。Tiny Keep。由数字部落娱乐公司出版，2014年。多平台。

[159] Pivotal Games Ltd.冲突：沙漠风暴。由SCi Games Ltd.出版，2002年。多平台。

[160] Polyphonic Digital.Gran Turismo.由SCEI出版，1997年。PlayStation。

[161] 程序艺术有限责任公司。外墙。由Procedural Arts LLC出版，2005年。PC / Mac。

[162] Psygnosis。歼灭战》。由SCEE出版，1995年。PlayStation。

[163] Quicksilver Software, Inc.猎户座的主人3。由Infogrames出版，2003年。个人电脑。

[164] 激进的娱乐。疤面煞星。由维旺迪环球游戏公司出版，2005年。多个平台。

[165] 罕见的有限公司。黄金眼007》。由任天堂欧洲有限公司出版，1997年。任天堂64。

[166] 乌鸦软件。命运战士2》：双螺旋。由动视出版，2002年。个人电脑和XBox。

[167] 叛乱。外星人与捕食者》。由雅达利公司出版，1994年。捷豹。

[168] 叛军。狙击手精英》，2005年。多个平台。

[169] 叛乱。网络空间，未发布。任天堂游戏机。

[170] 红色风暴娱乐公司。汤姆克兰西的幽灵行动。由Ubi Soft娱乐软件公司出版，2001年。个人电脑。

[171] 反思互动。驱动器。由GT互动出版，1999年。PlayStation和PC。

[172] 遗迹娱乐公司。Homeworld.由Sierra On-Line出版，1999年。个人电脑。

[173] 遗迹娱乐公司。英雄连》。由THQ公司出版，2006年。个人电脑。

[174] 革命软件有限公司。在钢铁的天空下。由维珍互动公司出版，1994年。个人电脑。

[175] Riot Games, Inc.英雄联盟》。由Riot Games, Inc.出版，2009年。多平台。

[176] 世嘉娱乐，公司。金斧子。由世嘉娱乐公司出版，1987年。商场。

[177] Sick Puppies Studio和International Hobo Ltd。幽灵大师。由帝国互动娱乐公司出版，2003年。个人电脑。

[178] 索尼电脑娱乐公司。Otostaz.由索尼电脑娱乐公司出版，2002年。PlayStation 2 (仅日本发行).

[179] 不锈钢工作室公司。帝国地球。由Sierra On-Line, Inc.出版，2001年。个人电脑。

[180] 战略模拟公司。计算机俾斯麦。由战略模拟公司出版，1980年。苹果II。

[181] 高级软件有限公司。流放。由Peter J. M. Irvin, Jeremy C. Smith出版，1988年。

BBC Micro。

[182] 17队。蠕虫3D》。由世嘉欧洲有限公司出版，2003年。多个平台。

[183] 樱桃团队。空心骑士。由Team Cherry出版，2017年。多平台。

[184] TechnoSoft.Herzog Zwei.由TechnoSoft出版，1989年。世嘉创世纪。

[185] The Creative Assembly Ltd.罗马：全面战争》。由动视出版公司出版，2004年。个人电脑。

[186] The Creative Assembly Ltd.帝国：全面战争》。由美国世嘉公司出版，2009年。个人电脑。

[187] 时空门工作室。科汉：阿利曼的礼物。由Strategy First出版，2001年。个人电脑。

[188] Turn 10工作室。Forza Motorsport.由微软游戏工作室出版，2005年。Xbox。

[189] Ubi Soft蒙特利尔工作室。细胞分裂。由Ubi Soft Entertainment Soft- ware出版，2002年。多平台。

[190] Ubisoft Annecy工作室。陡峭，2016年。多平台。

[191] Ubisoft Divertissements Inc.Far Cry 2.由UbiSoft出版，2008年。多个平台。

[192] UbiSoft蒙彼利埃工作室。超越善与恶。由UbiSoft Entertain- ment出版，2003年。多个平台。

[193] 阀门公司。半条命》。由Sierra On-Line出版，1998年。个人电脑。

[194] 阀门公司。半条命2》。由Valve公司出版，2004年。个人电脑。

[195] 阀门公司。Dota 2。由Valve公司出版，2013年。多平台。

[196] Valve South和Valve公司。左4死。由Valve公司出版，2008年。多平台。

[197] Warthog Games.梅斯-格里芬赏金猎人。由维旺迪环球游戏公司出版，2003年。多个平台。

[198] 西木工作室。沙丘II》。由维珍互动娱乐公司出版，1992年。个人电脑。

[199] 西木工作室。命令与征服》。由Virgin Interactive Enter- tainment出版，1995年。个人电脑。

[200] Zipper Interactive.SOCOM：美国海豹突击队。由SCEA出版，2002年。PlayStation 2.

索引

M