第二章

AI 方法

本章展示了许多在游戏中常用的基础 AI 方法，并且它们将在本书的其余部分进行讨论和涉及。这些都是在入门级的 AI 课程中被提及的方法——如果你学习过这样的一门课程，它至少会让你接触到本章中至少一半的方法。它也应该让你准备好去轻松地理解本章所涵盖的其它方法。

正如前面所述，本书假定读者已经熟悉了大学 AI 入门课程级别中的核心人工智能方法。因此，我们建议你在开始读这本书的其余部分前，确保你至少已经粗略地熟悉本章中介绍的方法。本章中对算法的描述将是**高级别的描述**，这意味着如果你已经在先前的某个时间学习了特定的算法的话，这将会刷新你的记忆；如果你先前不曾见过这个算法，它将解释这个算法的基本概念。每一章都伴随着指向文献，研究论文或者其它教科书的引导，在那里你可以找到每个方法更多的细节。

在本章中，我们将 AI 的相关部分（针对本书的目的而言）划分为六类：特定编辑(ad-hoc authoring)、树搜索、进化计算、监督学习、强化学习和无监督学习。在每一节中，我们使用一般用语来讨论了一些主要的算法，并提出对于进一步阅读的建议。在本章中我们使用《*吃豆人小姐*》（Namco，1982）作为我们涵盖的算法们的首要测试平台。出于一致性的原因，我们涵盖的所有方法都被用来**控制** Ms Pac-Man 的行为，尽管他们可以在这个游戏中找到很多其它的用途（例如生成内容或分析玩家行为）。虽然在本章中有许多其它游戏可以被用作我们的测试平台，但我们选择了 Ms Pac-Man，这是因为它的流行性与游戏设计的简单性，以及它在进行游戏时的高度复杂性。特别需要记住的是，Ms Pac-Man 是一个它的前辈《*吃豆人*》(Namco, 1990)的一个**非确定性的**(non-deterministic)变体，这暗示着鬼魂的移动涉及到了一定程度上的随机。

在章节 2.1, 我们快速地概括了本书中所有方法的两个重要组成部分：表示(representation)与效用(utility)。**行为编辑**(Behavior authoring)，涵盖于章节 2.2，是指采用静态的即时表现而没有任何形式的搜索或者学习，例如有限状态机(finite state machines)，行为树(behavior trees)与基于效用的 AI。**树搜索**，涵盖于章节 2.3，是指搜索未来的动作空间并且建立可能的动作序列的树的方法，通常用于对抗环境下；这包括了极大极小算法，它非常常见的变种 Alpha-Beta，以及蒙特卡罗树搜索。在章节 2.4 所涵盖的内容中，**进化计算**是指基于种群的全局随机优化算法，例如遗传算法(genetic algorithms)，进化策略(evolution strategies)或者粒子群优化(particle swarm optimization)。**监督学习**（见章节 2.5）指学习一个模型，可以将数据集中的案例映射到目标值，例如种类；目标值在监督学习中是不可或缺的。在这里常见的算法是反向传播（人工神经网络），支持向量机，以及决策树学习。**强化学习**被涵盖于章节 2.6，是指解决强化学习问题的方法，其中一些的动作伴随着正面或者负面的奖赏(rewards)，而不是一个“目标值”（正确的动作）。这里的范例算法是 TD-learning 与它流行的实例 Q-learning。

章节 5.6.3 概括了**无监督学习**，这指在没有目标值的数据集种寻找特征（或者聚类）。这这包括了聚类方法，例如 k-means，分层聚类以及自组织映射(self-organizing maps)，以及诸如 Apriori 和广义序贯模式(generalized sequential patterns)的频繁模式挖掘(frequent pattern mining)方法。这一章总结了许多著名的算法，它们组合了上述算法的元素来产生了**混合**(hybrid)方法。特别是我们涵盖了神经进化与带有 ANN 函数逼近的 TD-learning，并作为游戏 AI 领域使用的最流行的算法。

2.1 注意事项

在详细叙述每一种算法类型之前，我们概述了两个与本书中所有的 AI 方法有关联的两个首要元素。第一个是是算法的**表示;** 第二个是它的**效用**。首先，任何 AI 算法都在某种程度上存储和维护了关于需要处理的特定任务的知识。另一方面，大多数 AI 算法都在寻求更好的对于知识的表示。这种寻求过程由某种形式的效用函数所驱动。 我们应该注意到，效用在使用静态知识表示的方法中并无用处，例如有限状态机或行为树之类。

2.1.1 表示

适合地表示知识是大型人工智能的一个关键挑战，它由人类大脑存储以及检索获取关于世界的知识的能力所驱动。推动 AI 的表示的设计的关键问题如下：人们如何表达知识，AI 应当如何模仿这种能力？知识的本质是什么？一个表达体系可以有多通用？然而，对上述问题的常见答案在这一点中还是十分粗糙的。

作为对关于知识以及它的表示的这类公开广泛问题的回应，AI 已经确定了多种并且非常特殊的方法来存储与检索创作，获取或学习到的信息。对于一个任务或者一个问题的知识的表示可以被视为被调查任务的计算映射。在这个基础上，表示以一种机器可以处理的形式来存储关于任务的知识，例如一种数据结构。

为了允许任意形式的需要被以可计算的方式表示的人工智能知识，可能出现的方式有很多种。表示的类型包括了诸如语法演变(grammatical evolution)的**语法**(grammars)，有限状态机和概率模型(probabilistic models)的**图**(graphs)，决策树，行为树以及遗传的**树**(trees)，人造神经网络的**连接机制**(connectionism)，遗传算法和进化策略的遗传(genetic)以及差分学习(temporal difference learning)与 Q-learning 的**表格**(tabular)。我们将在本书的剩余部分看到，所有上述的表示形式在游戏中找到了不同的用处，并且可以与各种游戏 AI 任务相关联。

对于任何在一个特定任务中进行尝试的 AI 算法，有一点是肯定的：所选择的表示方法对于算法的性能有着重大的影响。

对于在特定任务中尝试的任何 AI 算法，有一点是肯定的：所选择的表示方法对算法的性能有重大影响。不幸的是，为一个任务而被选中的表示方法遵循“没有免费的午餐”这个定理[757]，这表明了并没有唯一的表示类型可以作为当前任务的理想选择。然而，作为常见的指导方针，被选中的表示方法应当尽可能的*简单*。简单性通常可以在计算工作量与算法性能之间达成一个微妙的平衡，无论是过于详细或者是过于简化都将会影响到算法的性能。除此之外，考虑到当前任务的复杂度，被选中的表达方法也应当尽可能*小*。无论简单性还是大小对于表示方法而言都不是一个可以忽略的决定。良好的表示方法来源于充足的对于 AI 正在尝试解决的问题的复杂程度及其定性特征的实践智慧以及经验知识。

2.1.2 效用

博弈论中的效用（和经济学）是在进行游戏时对理性选择的一种衡量。通常来说，它可以帮助一个搜索方法来决定采取哪一条路径。出于这个目的，效用函数对于搜索空间的各个方面进行了采样，并且搜集关于空间中“优秀”的区域的信息。在某种意义上，一个效用函数是对我们尝试寻找的解的一个近似值，是一种对于我们搜索到的现有表示的**优秀程度的衡量**。

与效用类似的概念包括了使用被计算机科学与 AI 所使用的**启发式**(heuristic)，作为在其它方法太慢而难以负担的情况下一种快速地*近似*解决问题的方式，特别是与树搜索的范式相结合。**拟合**(fitness)的概念也被近似地作为衡量一个解决方案良好程度的效用函数，其主要是在进化计算领域。在数学优化上，**目标函数**(objective function)，**损失函数**(loss function)，**代价函数**(cost function)，**误差函数**(error function)是需要被最小化的效用函数（或者在作为目标函数的情况下被最大化）。特别是，在监督学习中误差函数代表了一个方法将样本映射到（期望的）目标输出的良好程度。在强化学习以及马尔科夫决策过程中，效用被称为**奖赏**，这是是一个智能体尝试在一个特定状态中学习采取正确的动作来最大化的一个函数。最后，在**无监督学习**的领域中效用通常由内部所提供，并且存在于表示之中，例如竞争学习(competitive learning)或自组织(self-organization)。

与选择一个合适的表示方法类似，对于一个效用的函数也是“没有免费的午餐”。一个效用通常难以设计并且有时候设计任务基本是不可能的。它的设计的简单程度将有所回报，但是它的完整性也是如此。效用函数的质量极大地取决于从对于被研究领域的中获得的经验研究与实践经历。

与选择适当的表示类似，效用函数的选择遵循无免费午餐定理。 实用程序通常很难设计，有时设计任务基本上是不可能的。 其设计的简单性将取决于其完整性。 效用函数的质量在很大程度上取决于在调查范围内获得的彻底的实证研究和实践经验。

2.1.3 学习 = 最大化效用（表示）

效用函数是搜索的驱动，并且对于学习来说是必不可少的。在这个基础上，效用函数是任何机器学习算法的训练信号，因为它对我们拥有的表示的优秀程度提供了一种衡量。因此，它隐含地提供了关于做什么可以进一步提高现有的表示的优秀程度的指示。不需要学习的系统（例如基于即时的设计表示；或者专家知识系统）不需要一个效用。在监督学习中，效用采样于数据——例如，好的输入-输出特征。在强化学习与进化计算中，训练新号由环境提供——也就是在表现良好时的奖赏以及在表现糟糕时的惩罚。最后，在监督学习中，训练信号来源于表示的内部结构。

2.2 特定动作编辑

本章节中，我们将讨论第一个，也可以说是最流行的一种游戏开发中的 AI 方法。**有限状态机**，**行为树**与**基于效用的 AI** 都属于特定的动作编辑方法，在传统上主导了游戏中的非玩家角色的控制。它们的主导地位显而易见，如今游戏开发领域中游戏 AI 这个术语依然是这些方法的代名词。

2.2.1 有限状态机

一个**有限状态机**(FSM)[228]——以及有限状态机的变种，例如分层有限状态机——直到 00 年代中期都是主导了游戏中的非玩家角色的控制与决策过程的游戏 AI 方法。

有限状态机属于专家知识系统领域，并且使用图来表示。一个有限状态机的图是对一个需要被专门设计的相互关联的对象，符号，事件，动作或者现象的属性的集合的抽象表示。特别是，这个图包含了一些嵌入了一些数学抽象的节点（状态）与代表了节点之间的条件关系的边（转换）。有限状态机在一次时间内只能处理一个状态；在对应的转换条件满足的情况下，当前的状态可以改变为另一个。简而言之，有限状态机由三个主要组成部分所定义：

- 一定数量的存储了关于一个任务的信息的**状态**——例如，你当前正处于*探索*状态。
- 一定数量的在状态之间的**转换**，指示了一个状态的改变并且由一个需要被满足的条件所描述——例如，你听到了一声枪响，转换为*警戒*状态。
- 一组需要在每种状态中遵循的**动作**——例如，当处于探索状态时，*随机地移动并且寻找敌人*。

有限状态机的设计，实现，可视化以及调试出奇的简单。此外，它们已经在共同存在的多年时间中证明了它们可以在游戏中表现良好。然而，在大规模的环境下它们的设计可以变得极端复杂，因此，在游戏 AI 中的某些任务中有着计算上的限制。另一个有限状态机的关键限制（对所有特定动作编辑方法来说也是）是它们缺乏灵活性与动态性（除非被特意地设计）。在它们的设计完成之后，测试与调试都有着适应能力以及进化能力上的限制。结果就是，有限状态机最终在游戏中描绘除了非常可预测的行为。我们可以通过将转换表示为一个模糊规则[531]或者概率[108]来在一定程度上克服这个缺点。

2.2.1.1 一个 Ms Pac-Man 的有限状态机

在这个章节里，我们展示了用于控制 Ms Pac-Man 智能体的有限状态机。一个假设的以及简化的 Ms Pac-Man 的有限状态机控制者如图 2.1 所示。在这个例子中，我们的有限状态机有三个状态（寻找豆子，追逐鬼魂以及逃避鬼魂）与四个转换（鬼魂闪烁，没有可见的鬼魂，鬼魂出现在视野内，以及吃下了力量药片）。在*寻找豆子*这个状态，Ms Pac-Man 随机地移动，直到它探测到了一个豆子，然后遵循一个寻路算法来吃掉尽可能多的豆子并且尽可能的快。如果吃到了一个力量药片，Ms Pac-Man 会改变为追逐鬼魂状态，此时它可以使用任何的树搜索算法来追逐蓝色的鬼魂。当鬼魂开始闪烁的时候，Ms Pac-Man 会改变为逃避鬼魂的状态，此时它使用树搜索来逃避鬼魂，使得在一定距离内不存在可见的鬼魂；当达到这一情况后，Ms Pac-Man 切换到寻找豆子的状态。

2.2.2 行为树

一个**行为树**（BT）[109,111,110]是一个专家知识系统，其与有限状态机相类似，在有限的任务（或者行为）的集合之间建立转换。与有限状态机相比，行为树的优势是它们的模块化：如果设计良好，它们可以产生复杂的由简单任务组成的复杂行为。行为树与有限状态机（或者是分层有限状态机）相比，其主要区别在于它们由行为而不是状态所组成。与有限状态机一样，行为树容易设计，测试与调试，这使得它们在诸如《*光环 2*》(Microsoft Game Studios, 2004) [289] 与 *Bioshock* (2K Games, 2007)等游戏上的成功应用之后，主导了游戏开发场景。

行为树采用树形结构，其中包括一个根节点与一定数量的父节点及其对应的表示行为的子节点——参见图 2,2 所示的例子。我们从根节点开始贯穿行为树。然后我们激活图上标记的父节点与子节点的配对。一个子节点可以在预先决定的时长内给父节点返回如下值：如果行为仍然处于激活状态则返回*运行中*，如果行为已完成则返回*成功*，如果行为失败则返回*失败*。行为树由三种节点类型所组成：**序列节点**(sequence)，**选择节点**(selector)，**装饰节点**

(decorator)，它们的基本功能如下所述：

序列节点（见图 2.2 中的蓝色矩形）：如果子行为成功，序列将会继续，在所有的子行为都成功了之后，父节点也将成功；否则序列失败。

选择节点（见图 2.2 中的红色圆角矩形）：选择节点有两种主要类型，概率与优先级选择节点。当使用一个概率选择节点时，将基于行为树设计者设定的父-子概率来选择。另一方面，如果使用优先级选择节点，子行为将在列表中进行排序，并且一个接着一个地进行尝试。不管使用哪一种选择节点，如果子行为成功，则选择节点也成功。如果子行为失败，队列中的下一个子行为将被选中（在优先级选择节点中），或者选择节点失败（在概率选择节点中）。

装饰节点（见图 2.2 中的紫色六角形）：装饰节点为一个单独的子行为增加复杂度或者加强能力。装饰器的例子包括了子行为运行的次数或者给定子行为完成任务的时间。

与有限状态机相比，行为树在设计上更有灵活性，并且易于测试；然而它们仍然有着类似的缺点。特别是，鉴于它们是静态的知识表示，它们的动态性相当地低。概率选择节点可能会增强它们的不可预测性，以及一些适应它们树形结构的方法也展现出了一些希望[384]。行为树与 Mateas 和 Stern 介绍的用于基于故事的可信角色的 ABL(A Behavior Language)[439]也存在着一定程度上的相似性；它们的不同之处也于[750]中所报告。然而需要注意的是，本章节仅仅大概地划过了关于行为树设计的可能性的表层，而在它们的基本结构之上存在着一些拓展，可以帮助行为树改善它们的模块化与它们解决更为复杂的行为设计的能力[169,626]。

2.2.2.1 一个 Ms Pac-Man 的行为树

与上面的有限状态机案例类似，我们使用 Ms Pac-Man 来展现行为树在一个流行游戏中的使用。在图 2.3 中我们展示了一个关于 Ms Pac-Man 的寻找豆子行为的简单行为树。当处于*寻找豆子*的序列行为中时，Ms Pac-Man 首先会*移动*（选择节点），然后它会寻找一个豆子并且最终它会保持吃豆子的状态，直到一个视野中发现了一个鬼魂（装饰节点）。当处于移动行为时——这是一个优先级选择节点——Ms Pac-Man 会将没有鬼魂的走廊优先于有药片的走廊以及没有药片的走廊。

2.2.3 基于效用的 AI

正如一些工业界的游戏 AI 开发人员所指出的那样，行为模块度在游戏以及游戏中的任务中的缺乏不利于开发高质量的 AI[604,170]。**基于效用**的 AI 是一种越来越流行的特定动作编辑的方法，消除了有限状态机与行为树的模块化限制，可以被用于设计游戏中的控制与决策系统[424,556]。根据这种方法，游戏中的实例会被赋予一个特定的效用函数，给出一个其对于特定实例的重要程度的值[10,168]。例如，在一个特定距离内出现敌人的重要程度或者在这种特定情况下智能体的健康值很低的重要程度。给予一个智能体所有可用的效用的集合以及所有它拥有的选项，基于效用的 AI 决定哪一个是它在当前时刻需要考虑的最重要的选项[425]。基于效用的方法是基于经济学中的效用理论，并且基于效用函数的设计。这个方法与模糊集中的隶属函数的设计相类似。

一个效用可以衡量从可观察的客观数据（例如敌人的健康值）到诸如情绪，心情与威胁这样的主观概念在内的任意事物。关于可能的行动或者决策的各种效用可以聚合为线性或者非线性的公式并引导智能体基于聚合的效用而做出决策。效用的值可以在游戏中的每 n 个阶段进行检查。所以当有限状态机与行为树在一个时间内只能检测一个决定时，基于效用的 AI 架构会检查所有可行的选项，为它们赋予一个效用并且选择最适合的一个选项（最高的

效用)。

作为基于效用的 AI 的一个例子，我们会基于[425]中出现的武器选择过程来讲解。为了选择武器，智能体需要考虑如下方面：范围，惯性，随机噪声，弹药与室内环境。*范围*的效用函数添加根据距离来增加一个武器的效用值，例如，如果距离很短，手枪将被赋予更高的效用。*惯性*为当前的武器赋予了一个更好的效用值，这样武器的改变将不会非常的频繁。*随机噪声*为选择添加了非确定性，这样智能体就不会在相同的游戏情况中总是选择相同的武器。*弹药*返回了一个关于当前弹药的等级的效用，而*室内环境*将通过一个布尔值的效用函数（例如在室内使用手榴弹则值为 0，否则为 1）惩罚部分武器在室内环境下的使用，例如一个手榴弹。我们的智能体定期检查可用的武器，为所有的武器赋予效用的分数，并且选择有着最佳总效用的武器。

基于效用的 AI 与其它特定的动作编辑技术相比较具有一定的优势。它是*模块化*的，因为游戏智能体的决定取决于许多不同的因素（或考量）；这个因素的列表可以是动态的。基于效用的 AI 也是*可拓展*的，我们可以轻松地创建新类型的考量，只要我们认为它们适合。最后，这个方法也是*可重用*的，因为效用的部件可以从一个决策转移到另一个，从一个游戏转移到另一个游戏。由于这些优点，基于效用的 AI 在游戏产业中逐渐受到了更多的关注[556,170]。基于效用的 AI 已经在许多种类的游戏中被广泛使用，在 *Kohan 2: Kings of War* (Take Two Inter- active and Global Star Software, 2004)中与其它方法共同使用，在 *Iron Man* (Sega, 2008)中用来控制头目，在 *Red Dead Redemption* (Rockstar Games, 2010)中作为武器与对话选择，以及在 *Killzone* 2 (Sony Computer Entertainment, 2009)与 *F.E.A.R.* (Sierra Entertainment, 2005)中作为动态战术决策。

2.2.3.1 Ms Pac-Man 中基于效用的 AI

又一次，我们使用 Ms Pac-Man 来展示基于效用的 AI 的使用。图 2.4 展示了一个案例，拥有三个可以在游玩 Ms Pac-Man 时被考虑到的效用函数。每一个函数对应一个依赖于当前游戏的威胁级别的不同行为；威胁也就是一个关于当前鬼魂位置的函数。在游戏中的任何时刻 Ms Pac-Man 都会选择拥有最高效用值的行为。

2.2.3.2 一个关于特定动作编辑的简短说明

非常重要的是，要记住在本章节中所涵盖的所有三种方法（也可以说，所有在这一章中涵盖的方法）都代表了算法的**基本变体**。因此，我们涵盖的算法表现为对于状态，行为与效用函数的静态表示，然而，也可以通过添加非确定性或者模糊元素来创造它们的动态变体；例如，可以在有限状态机中使用模糊转换或者在行为树中演化行为。除此之外，很重要的是注意这些特定的设计结构可以描述所有本书在本章剩余部分所涵盖的算法的特征，基本的处理元素，例如一个有限状态机的状态，一个行为树的行为或一个效用函数，甚至更为复杂的节点，树或者功能的层次可以被其它任意的 AI 方法所替换来产生混合的算法与智能体结构。注意算法们可能的拓展可以在我们于每个算法对应的章节中引用的工作里找到，也可以在我们接下来提供的阅读列表中被找到。

2.3 树搜索

在很大程度来来说，即使不是全部，大部分的人工智能也仅仅只是搜索而已。几乎每一个 AI 问题都可以作为一个搜索问题，并可以通过寻找最佳的（通过某些措施）的规划，路

径，模型，函数等来解决。搜索算法因此经常被看作是 AI 的核心，许多教材（例如 Russell 与 Norvig 的著名教材[581]）都是从搜索算法的处理开始的。

下面给出的算法都可以被表示为**树搜索算法**，因为它们都可以被视为建立一棵**搜索树**(search tree)，其根节点代表了搜索开始的状态。这颗树中的边代表了智能体从一个状态到另一个所采取的动作， 而节点代表了状态。因为通常在一个给定的状态中可能有数种不同的可以被采取的动作，也就是树的分支。树搜索算法的主要不同在于探索哪些分支以及以什么顺序它们。

下面给出的算法都可以被表征为树搜索算法，因为它们可以被看作构建搜索树，其中根是表示搜索开始的状态的节点。 该树中的边缘表示代理从一个状态到另一个状态所采取的操作，节点表示状态。 因为通常在给定的状态下可以采取几种不同的动作，树分支。 树搜索算法主要不同，哪些分支被探索和以什么顺序。

2.3.1 盲目搜索

盲目搜索算法是在没有任何关于目标的进一步信息时搜索一个状态空间的算法。基本的盲目搜索算法被普遍认为是计算机科学的基础算法，并且在某些时候甚至不被视为 AI。

深度优先搜索(depth-first search)是一种可以在回溯之前尽可能远地探索每个分支，然后再尝试另一个分支的算法。在它的主循环中的每一次迭代里，深度优先搜索选择一个分支，然后在下一次的迭代中将会转移去探索结果节点。当遇到一个末端节点——一个不可能再继续拓展下去的节点——时，深度优先搜索向前移动被访问过的节点的列表，直到它找到了一个带有未探索过的动作的节点。当被用于游玩游戏时，深度优先搜索会探索一个单独的移动的结果，直到游戏胜利或者失败，然后再去探索采取另一个接近结束状态的不同移动的结果。

宽度优先搜索(breadth-first search)与深度优先搜索相反。深度优先搜索探索了所有从一个单独的节点引出的动作，而不是一个单独行动的所有结果，而这些都在探索任何由这些动作导致的节点之前。因此，所有深度为一的节点都会在深度为二的节点之前被探索，然后是所有深度为三的节点，等等。

虽然上述是基础的盲目搜索算法，但这些算法有着许多的变种以及组合，并且新的盲目搜索算法也正在被探索。关于盲目搜索算法的更多信息可以在[581]的第四章中被找到。

在游戏中很少可以看到盲目搜索算法被有效地使用，但是也有例外，例如迭代宽度搜索[57]，通常在游玩视频游戏时令人吃惊地表现良好，以及宽度优先搜索在 *Sentient Sketchbook*[378]中用于评价策略游戏地图的各个方面。此外，将最先进的算法的表现与一个简单的盲目搜索算法进行比较也经常是富有启发的。

2.3.1.1 用于 Ms Pac-Man 的盲目搜索

一个 Ms Pac-Man 中的深度优先方法通常会考虑博弈树中的分支，直到 Ms Pac-Man 达到了等级或者失败。这个搜索对每个可能的动作的结果将决定在一个时刻将采取哪一个动作。宽度优先搜索将会首先探索 Ms Pac-Man 在游戏中的当前时刻所有可行的动作（例如，向左，向上，向下或者向右）然后再探索所有它们会导致的节点（子节点）等。这两个方法在 Ms Pac-Man 案例中的博弈树在可视化上都过于巨大与复杂了。

2.3.2 最佳优先搜索

在**最佳优先搜索**(best-first search)中，搜索树上的节点的拓展由一些关于目标状态的知识

来启发。通常来说，在某些准则上最接近于目标状态的节点会被首先拓展。最著名的最佳优先搜索算法是 A*（读为 A star）。A*算法维护了一个“开放(open)”节点的列表，它们是与已经被探索过的节点相邻但是它们自身尚未被探索过。对于每一个开放节点，会生成一个从其到目标的距离的估计。新的节点将会基于最低的开销基础而被选中以及拓展，在这里开销也就是与初始节点的距离加上与目标节点的距离的估计。

A*很容易被理解为一个在二维或三维空间中的导航。因此这个算法的变体通常被用于游戏中的**寻路**。在许多游戏中，“AI”本质上相当于使用 A*寻径来遍历脚本点。为了应对更大，具有欺骗性的空间，这个基本算法的多种修改已经被提出，包括了 A*的分层版本[60,661]，实时启发式搜索(real-time heuristic search)[81]，对于统一代价的方格的的**跳点式搜索**(jump point search)[244]，3D 寻径算法[67]，使得人群动画处于无碰撞路径[630]的动态游戏世界的规划算法[494]以及用于在导航网格中寻径的方法[67,722]。Steve Rabin 与 Nathan Sturtevant 在基于方格的寻径上的工作[550,662]以及寻径结构[549]也是著名的案例。Sturtevant 以及同事从 2012 年起也一直致力于一个专门的基于网格的路径规划竞赛。对于有兴趣的读者，Sturtevant [663]已经发布了一个基于游戏中的网格寻径的基准列表，包括了《*龙腾世纪：起源*》(Electronic Arts, 2009)，《*星际争霸*》(Blizzard Entertainment, 1998) 与《*魔兽争霸 III：混乱之治*》(Blizzard Entertainment, 2002)。

然而，相对于简单的搜索物理位置，A*也可以被用于在游戏状态的空间内进行搜索。这样，最佳优先搜索可以被用于**规划**而不仅限于导航。区别在于需要考虑世界的状态的改变（而不是仅仅一个单独的智能体的状态改变）。使用 A*的规划可以令人惊奇地有效，例如 2009 马里奥 AI 竞赛（在其中选手需要提交用于游玩《*超级马里奥兄弟*》(Nintendo, 1985)的智能体）的胜利者也是基于一个简单的 A*规划，其只是简单地在所有时刻中试图到达屏幕的右边末端[717,705]（也可以见图 2.5）。

2.3.2.1 Ms Pac-Man 的最佳优先搜索

最佳优先搜索可以在 Pac-Man 中以 A*的形式被应用。遵循 2009 马里奥 AI 竞赛冠军的范例，Ms Pac-Man 可以被一个 A*算法所控制，通过搜索在一个短时间内可能的游戏状态并决定接下来向何处移动（上，下，左或者右）。游戏状态可以以多种方式来表示：从一个非常直接的，但是开销较大的将鬼魂以及豆子的坐标都纳入其中的表示到一个间接地考虑到距离最近的鬼魂或豆子的距离的表示。无论哪一种表示被选中，A*需要设计一个成本函数来驱动搜索。与 Ms Pac-Man 有关的成本函数通常会奖励移动到存在豆子的区域，并惩罚移动到存在鬼魂的区域。

2.3.3 极大极小

对于单人游戏，简单的盲目搜索或者启发式搜索(informed search)算法可以被用于找到一条通向最佳游戏状态的路径。然而，对于双人对抗游戏，存在另一个也试图获得胜利的玩家，并且每一个玩家的动作都非常依赖于另一个玩家的动作。对于这些游戏我们需要对抗性的搜索。基本的对抗性搜索算法被称为**极大极小**(Minimax)，这个算法已经被非常成功地被广泛用于传统的完美信息双人棋盘游戏，例如国际跳棋和国际象棋，并且事实上是以制作一个可以玩国际象棋的程序而被专门（重新）发明的[725]。

Minimax 算法的核心循环在玩家 1 与玩家 2 之间交替——例如国际象棋中的白方与黑方——被称为极小(min)玩家与极大(max)玩家。对于每一个玩家，所有可能的招式都会被探索。对于其导致的每一个状态，另一个玩家所有可能的招式也都会被探索，直到所有可能的

招式组合被探索至游戏结束的节点（例如胜利，失败或者和局）。这一过程的结果对从根节点下降到叶节点的整个博弈树的生成。游戏的结果启发了被应用于叶节点的效用函数。效用函数的估计会显示当前游戏布局下对一个玩家的优势。然后，算法将遍历搜索树，通过从分支节点上的叶节点的回传值，来决定在每一个玩家的每一个假设的状态中什么动作将会被采用。在这期间，它假设每一个玩家都在试图以最优的方式进行游戏。因此，从*极大*玩家的角度来看，它尝试将它的分数最大化，与此同时*极小*玩家也在尝试最小化*极大*玩家的分数；因此，其名为 *Minimax*。换句话说，树中的一个*极大*节点计算它的子节点的值中的最大值，同时一个*极小*节点计算它的子节点的值的最小值。如果在*极小*玩家的回合，*极小*玩家可以使用的*所有招式*都让*极大*玩家有机会获取胜利的话，*极大*玩家将会获得一个最佳的胜利策略。对应的*极小*玩家的最佳策略是在，无论*极大*玩家采取什么招式，总是有一种胜利是可行的时候才获得的。例如，为了获得最大玩家的一个胜利策略，我们从树的根节点开始，迭代地选择会导致最高分数子节点的招式（在最小玩家的回合，拥有最低分数的子节点会被选为替代）。图 2.6 通过一个简单的例子说明了 Minimax 的基本步骤。

当然，探索所有可能的招式以及应对招式对于任何有着令人关注的复杂性的游戏来说都是不可行的，因为搜索树的规模会随着游戏的深度或者假设的招式的数量而指数型地增加。作为例子，井字棋(tic-tac-toe)拥有一个规模为 9! = 362,880 个状态的博弈树，可以被完全的遍历；然而，国际象棋博弈树有接近 10^{154} 个节点，使用现代计算机来进行完全搜索是不可行的。因此，几乎所有 Minimax 的实际应用都会在一个给定的深度上切断搜索，并且使用一个**状态评估**(state evaluation)函数来评价在指定深度的每一个游戏状态的可取性。例如，在国际象棋中一个简单的状态评估函数可以仅仅只是棋盘上白棋的数目的总和，然后减去黑棋的数目；这个数字越高，这个局面对于白方来说就越好（当然，普遍使用更为复杂的棋盘评估函数）。伴随着一些基本的 Minimax 算法的改善，如 **α-β 剪枝**(α-β pruning)与非确定性状态评估函数的使用，许多经典的游戏上浮现了一些非常具有竞争能力的程序（例如，IBM 的深蓝）。更多关于 Minimax 以及其它对抗性算法的信息可以在[581]的章节 6 中被找到。

2.3.3.1 用于 Ms Pac-Man 的 Minimax

严格来说，Minimax 对于 Ms Pac-Man 来说是不能应用的，因为这个游戏是非确定性的，因此 Minimax 树在形式上是未知的（当然 Minimax 带有启发式评估函数的变体最终是可以应用的）。然而，Minimax 适用于 Ms Pac-Man 具有确定性的前身，《*吃豆人*》(Namco, 1980)上。又一次严格来说，Pac-Man 是一个单人对抗游戏。因此 Minimax 只适用于我们假设 Pac-Man 要与会选择最佳决策的对手（鬼魂）进行对抗时。很重要的是，鬼魂的移动并不被树节点所表示；相反，它们是根绝它们假设的最佳行为来模拟的。Pac-Man 中的博弈树节点可能代表了包括 Pac-Man，鬼魂以及现有的小球的位置和可用的力量药片的游戏状态。Minimax 树的分支是 Pac-Man 在每个游戏状态下可行的动作。最终节点可以被赋值，例如一个二元值的效用（如果 Pac-Man 完成了则为 1，如果 Pac-Man 被鬼魂杀死了则为 0）或者游戏的最终分数。

2.3.4 蒙特卡罗树搜索

在很多游戏中 Minimax 的表现并不良好。特别是具有大量**分支因素**的游戏（在任意时间点都有着许多潜在的动作供执行）导致了 Minimax 只能搜索一个非常浅的树。另一个在游戏中会给 Minimax 的运作造成困扰的方面是当它难以建立一个良好的局面评估函数时。棋盘游戏中的围棋是一个确定性的，完备信息博弈，是这两种现象的一个很好的例子。围棋

有着接近 300 的分支因素，而国际象棋通常由大约 30 个动作可供选择。围棋游戏中的位置的性质，全部都与包围对手有关，导致了很难去正确的估量一个给定棋盘局面的价值。在很长一段时间内，世界上最好的围棋程序中的大部分基于 Minimax，几乎不能超过人类初学者的游戏水平。在 2007 年，**蒙特卡罗树搜索**(MCTS)被发明，最好的围棋程序的游戏水平被极大地增强了。

除了例如围棋，国际象棋与国际跳棋这样复杂的完备信息确定性博弈之外，如海战棋(Battleship)，扑克，桥牌这样的**不完备信息**博弈，以及/或者如西洋双陆棋与大富翁(monopoly)这样的**非确定性的**博弈都由于算法自身的性质而无法被 Minimax 所解决，MCTS 不仅克服了 Minimax 中对于树规模的限制，并且在给予充分计算的情况下，它也能约等于博弈中的 Minimax 树。

那么 MCTS 是如何处理高分支因素，缺少良好的状态评估函数，以及完备信息与确定性的缺少呢？首先，它并不会将搜索树中的所有分支都搜索到一个相同的深度，而是集中于更有前途的分支上。这使得它可以搜索某一个分支到客观的深度，即使分支因素巨大。此外，为了绕过对评估函数，确定性的缺乏以及不完备信息这些问题，标准的 MCTS 方法使用**随机推演**(rollouts)来估计游戏状态的质量，从一个游戏状态随机的进行游戏直到游戏的解决来得到一个预期的胜利（或者失败）结果。通过随机模拟得到的效用值可以被有效地用于调整策略(policy)以直到一个最佳优先的策略（一个 Minimax 树的逼近）。

在 MCTS 的一次运行开始的时候，树由一个单独的，表示游戏当前状态的节点所组成。算法会迭代地通过添加以及评价代表游戏状态的节点来建立一棵搜索树。这个过程可以在任何时间被中断，因此 MCTS 是一个**任意时长**的算法。MCTS 只需要两部分的信息来操作：会交替产生在游戏中可用招式的*游戏规则*，以及*最终状态的评价*——其是否胜利，失败，和局，或者一个游戏分数。MCTS 的初始版本并不需要一个启发式函数，而这也是其对 Minimax 而言的一个关键优势。

MCTS 算法的核心循环可以被分为四个步骤：**选择**(Selection)，**拓展**(Expansion)，**模拟**(Simulation)与**反向传播**(Backpropagation)。前两个选项也被称为*树策略*(tree policy)。这些步骤也被描述于图 2.7。

选择：在这个阶段，将决定哪一个节点应该被拓展。这个过程从树的根节点开始，并且持续到一个有着未被拓展的子节点的节点被选中。每当要从现存的树中选择一个节点（动作时），一个子节点 j 将被选中以最大化 UCB1 公式：

$$\mathrm{UCB1} = \bar{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{2.1}$$

其中，$\bar{X}_j$是这个节点下的所有节点的平均回报，C_p是一个探索常数（通常设为 $1/\sqrt{2}$），n 是父节点已经被访问的次数，以及n_j是子节点 j 已经被访问的次数。非常重要的是，尽管 UCB1 是最流行的用于行动选择的公式，但它并非是唯一可用的公式。在公式(2.1)之外的其它选项包括了小量贪婪算法(epsilon-greedy)，汤普森采样(Thompson sampling)，以及贝叶斯匪徒模型(Bayesian bandits)。例如，汤普森采样根据最优的后验概率[692]来随机的选择动作。

拓展：当一个有着未拓展的节点被选中时——举例来说，这代表着一个状态，其存在着可以被执行但尚未被尝试过的动作——这些子节点中的一个将被选中作为*拓展*，意味着一次模拟将被从这个状态开始直到结束。选择哪一个节点将被拓展经常都是被随机完成的。

模拟（**默认策略**，Default Policy）：在一个节点被拓展之后，一次模拟（或者*随机推演*）将被从刚刚拓展的非末端局面开始直到游戏结束以产生一个估计值。通常，这是通过随机的执行动作直到接触到一个末端局面来执行，例如直到游戏胜利或者失败。在游戏末端的状态（例如，如果失败为-1，如果胜利为 1，但也可以更为细致）被作为本次模拟的奖赏(Δ)，并

且向搜索树中进行传播。

反向传播：奖赏（模拟的结果）将被添加到新节点的总奖赏 X 中。他也会"后退"：被添加至它的父节点的奖赏当中，以及它的父节点的父节点直到树的根节点。

模拟步骤可能看起来有些违反直觉——执行随机动作似乎并不是玩一个游戏的好方法——但是它提供了一个对游戏状态质量的相对无偏估计。从本质上讲，一个游戏状态越好，就有越多的模拟可能会在赢得游戏时结束。至少，像围棋这样的游戏就是这样的，游戏总是会在一定数目的招式内达到末端状态（对于围棋而言是 400）。对于像国际象棋这样的其它游戏来说，它在理论上可以在没有出现游戏上的输赢的情况上进行任意数目的招式。对于许多电子游戏而言，很可能任何随机的动作序列都不能结束游戏，除非一些计时器停止，这意味着大部分模拟都会变得非常的长（几万甚至几十万步）并且不会产生有用的信息。例如，在 *Super Mario Bros* (Nintendo,1985)中，随机动作的应用很可能会导致马里奥在它的出生点附近跳舞，直到它的时间被耗尽[292]。因此在许多情况中，使用一个状态评估函数（正如 Minimax 中被普遍使用的那样）来补充模拟步骤是很有用的，因此模拟可以在一个设定好的步长中被执行，并且如果未达到末端局面，则状态评估将被执行来替代一个非胜即负的评估。在某些情况下，使用状态评估函数来完全替代模拟步骤甚至可能是有益的。

值得注意的是，存在许多的基础 MCTS 算法的变体——事实上将 MCTS 视为一个算法家族或者框架而不是一个单独的算法可能是更有用的。

2.3.4.1 用于 Ms Pac-Man 的 MCTS

MCTS 可以被应用于 Ms Pac-Man 的智能体的实时控制。很明显这里存在许多种方法来表示一个游戏状态（然后成为一个博弈树节点），并且设计一个游戏的奖赏函数，在这里我们将不会进行详细的讨论。作为替代，在这个章节中我们会概述由 Pepels 等人提出的下述方法[523]，这是鉴于其成功地在 Ms Pac-Ma 中获得了一个高分。它们名为 Maastricht 的智能体在 2012 年的 IEEE 计算智能与游戏会议中获得了超过 87000 分并且名列第一（在 36 个智能体中）。

当 MCTS 被用于实时决策，许多挑战都会变得非常关键。首先，算法限制了随机推演的计算预算，这增加了启发式知识的重要性。其次，动作空间可以变得特别细粒度，这表明宏观行为是一个更有力的方式来对博弈树进行建模；不然智能体的规划将会变得非常短暂。第三，视野内可能不存在末端节点，这需要良好的启发式并尽可能地限制模拟深度。Pepels 等人的 MCTS 智能体[523]通过使用一个受限制的博弈树与基于连接的游戏状态表示（见图 2.8），成功地应对了上述所有在实时控制中使用 MCTS 的挑战。

2.3.5 进阶阅读

基本的搜索算法们同样也被涵盖于 Russell 与 Norvig 的经典 AI 教材[581]中。A*算法于 1972 年为了机器人导航而被发明[245]；这个算法的一个很好的描述可以在[581]的第四章中被找到。在如[545]这些专门的游戏 AI 书籍中，有着更多的针对特定游戏问题的裁剪与优化的进阶材料。蒙特卡罗树搜索的不同组成部分[140]是在 2006 年与 2007 年在研究围棋的背景下被发明的[141]；Browne 等人在一份调查中对于 MCTS 以及它的一些变体做出了一个很好的概括以及介绍[76]。

在专门的游戏 AI 书籍（如[545]）中，针对特定游戏问题的裁剪和优化算法有很多更高级的材料。 蒙特卡罗树搜索[140]的不同组成部分是在 2006 年和 2007 年在玩 Go [141]的背景下发明的; Browne 等人在一份调查报告中对 MCTS 及其一些变体进行了很好的概述和介

绍。[76]。

2.4 进化计算

在树搜索算法从表示一个原始状态的根节点开始，并且基于可用的动作来构建一棵搜索树时，优化算法却并不会建立一棵搜索树；它们只考虑完整的解决方案，而不是走到那里去的路径。正如先前在章节 2.1 所提到的，所有的优化算法都假设存在一些方法可以优化解决方案；这里必须有一个**目标函数**，或者称为**效用函数**(utility function)，**评估函数**(evaluation function)或者**适应度函数**(fitness function)，它可以给一个解决方案分配一个数值（**适应度**），并且这个值可以被最大化（或者最小化）。给定一个效用函数，一个优化算法可以被视为一个在解决方案的搜索空间中寻找有着最高（或者最低）的效用值的解决方案的算法。

一个广泛的优化算法家族是基于解决方案的随机变化，其中一个或者多个解决方案将在任意给定的时间内保留，而创造新的解决方案（或者候选人，或者搜索点；不同的术语被不同的著作所采用）是通过随机改变已有的解决方案中的某些部分，也可能是将它们中的某些部分相互结合。保持了多个解决方案的随机优化算法被称为**进化算法**(evolutionary algorithms)，与自然进化相类比。

另一个在探讨优化算法（以及章节 2,1 中涵盖的大规模 AI）时出现的重要概念是它们的**表示**。所有的解决方案都可以以某种方式来被表示，例如固定大小的实数向量，或者可变长度的字符串。一般来说，相同的事物可以被许多不同的方式所表示；举例来说，在搜索可以解决一个迷宫的动作序列时，动作序列可以被几种不同的方式所表示。在最直接的表示中，在步长为 t 处的字符决定了在时间步长 t+1 时需要采取什么动作。一个在某种程度上比较间接的动作序列的表示是使用一个元组序列，其中在时间步长 t 处的字符决定了采取什么动作，以及数字 t+n 决定了在多少个时间步长 n 内采取该动作。表示的选择对于搜索算法的效率和有效性有着很大的影响，在做出这些选择时，可能会有一些权衡。

优化是一个非常普遍的概念，优化算法可以被用于 AI 中的各种类型的任务以及在更为普遍的计算过程中。在 AI 以及游戏中，如进化算法之类的优化算法也被用于充当各种角色。在第 3 章中我们解释了优化算法如何被用于游戏智能体的搜索中，以及在动作序列的搜索中（在游戏环境下同时存在两种非常不同的对优化的使用方法）；在第 4 章，我们解释了我们如何使用优化来创造如关卡之类的游戏内容；在第 5 章中我们讨论了如何使用优化来寻找玩家模型。

2.4.1 局部搜索

最简单的优化算法就是局部优化算法。它们被这么称呼是因为它们仅仅“在局部”进行搜索，也就是在任意时刻，都只在搜索空间的一小部分中进行。一个局部优化算法通常在在任意时刻都只保存了单个的解决方案候选，并且探索这个解决方案的各类变化。

可以说最简单的优化算法可能就是**爬山算法**(hill climber)。在它最常见的，也就是我们可以称为确定性公式的公式之中，它的运作方法如下所述：

1.*初始化*：通过在搜索空间中随机选择一个点来创造一个解决方案 s。评估它的适应度。
2.生成 s 所有可能的邻居。一个邻居是任意的与 s 至多相差一个确定的给定距离的解决方案（例如，在一个单独位置上的改变）。
3.使用适应度函数来评估所有的邻居。
4.如果没有一个邻居比 s 有更好的适应度得分，退出算法并且返回 s。

5.否则，使用有着最高适应度值的邻居替换 s，并返回步骤 2。

确定性的爬山算法只有在每个解决方案都有着少数邻居的表示下才是可行的。在许多表示中，存在着如同天文数字一般的邻居。因此，最好使用可以有效地引导搜索的爬山算法的变种。一种方法是**基于梯度的爬山算法**(gradient-based hill climber)，沿着梯度直到最小化一个代价函数。例如使用这种方法来训练人工神经网络（见章节 2.5）。我们在这里涵盖的另一种方法是**随机爬山算法**(randomized hill climber)。这是来自于**变异**(mutation)的概念：对解决方案的一个小的随机变化。例如，可以字符串可以通过随机地翻转一两个字符到其它字符被变异（参见图 2.9），以及一个实数的向量可以通过向其添加另一个从零附近的随机分布中抽取并且标准差很小的向量来被变异。例如基因**倒置**(inversion)等宏观上的突变也如图 2.9 所示。给定一个表示，适应度函数以及变异操作，随机爬山算法的运作如下：

1.初始化：通过在搜索空间中随机选择一个点来创造一个解决方案 s。评估它的适应度。
2.变异：通过变异 s 产生一个后代 s'。
3.评估：评价 s' 的适应度。
4.替换：如果 s' 有着比 s 更高的适应度，使用 s' 替换 s。
5.返回步骤 2。

尽管非常简单，但随机爬山算法可能会惊人地有效。它的主要限制是容易陷入局部最优。搜索空间中的**局部最优值**(local optimum)是一种“死路”，从中将“无路可走”；也就是一个点的附近区域中不存在更好的（更为适应的）点。处理这个问题有许多种方法。一种是简单地在遇到困难时在一个新的随机选中的点重启爬山算法。另一种是**模拟退火**(simulated annealing)，给予一定的概率来接受移动到有着更低适应度的解决方案；这个概率在搜索过程中减小。一个更为普遍的对局部最优问题的应对是在任何时刻都不要只保留一个单独的解决方案，而应该是一个解决方案的**种群**(population)。

2.4.1.1 用于 Ms Pac-Man 的局部搜索

尽管我们可以想出几种方法来在 Ms Pac-Man 中应用局部搜索，但我们仅一个它在控制路径规划上的应用的案例。举例来说，局部搜索可以产生更短的 Ms Pac-Man 的局部规划（动作序列）。一个解决方案可以被表示为一个需要被应用的行为的集合，其适应性可以通过在应用了这个动作序列之后得到的分数来确定。

2.4.2 进化算法

进化算法是随机的**全局**优化算法；他们被称为全局而不是局部是因为他们在搜索空间中同时地搜索了许多个点，尽管这些点可能相距甚远。他们通过在任意时刻都在内存中保留一个解决方案的种群来完成这一目标。进化计算的总体思路是通过“培育”解决方案来进行优化：产生许多解决方案，丢弃不好的个体并且保留优秀的（或至少是不坏的）个体，并且从优秀的个体中创造新的解决方案。

保留一个种群的想法获得于达尔文进化论中的自然选择，进化算法也是从中得名。种群的规模是一个进化的算法的关键参数之一；一个为 1 的种群规模会产生一个类似随机爬山法的存在，而上千解决方案的种群也并不是前所未闻的。

保持种群的想法是从自然选择的达尔文进化中获得的，进化算法也从中得名。 人口规模是进化算法的关键参数之一， 人口规模为 1 时产生的东西就像一个随机的登山者，而几千种解决方案的人口也并非前所未闻。

另一种来源于自然界中的进化的想法的**交叉**(crossover)，也被称为**重组**(recombination)。这相当于自然世界中的有性生殖：两个或者更多的解决方案（称为**父代**）通过结合它们自己的元素而产生一个后代。这个想法是，如果我们使用两个良好的解决方案，一个二者相结合的解决方案——或者它们的中间产物——也应当会是良好的，甚至比父代更为优秀。后代操作高度依赖于对解决方案的表示。当解决方案被表示为一个字符串或者一个向量时，如均匀交叉(uniform crossover)（其公平地投掷一次硬币，并且为后代的每一个位置都随机地从每一个父代的值中进行挑选）或者单点交叉(one-point crossover)（后代中一个位置 p 被随机选中，并且在 p 之前的所有位置的值从父代 1 中获取，在 p 之后的所有位置的值从父代 2 中获取）之类的操作方法可以被使用。交叉可以被应用于从比特串到一个实数向量在内的任意染色体表示。图 2.10 展示了这两种交叉操作。然而，在任何情况下都不能保证交叉操作会产生一个在任意方面都比他的父代更为适应的后代。在许多情况下，交叉可能是非常具有破坏性的。如果使用交叉，那么对后代的操作的选择的每一个问题都是很重要的。图 2.11 通过一个简单的二维样例展示了这种可能性：

一个进化算法的基础模板如下：

1.初始化：种群由 N 个随机创建的解决方案所填充，例如搜索空间中的随机点。已知的高适度性的解决方案也可以被添加到这个初始种群中。

2.评估：适应度函数被用于评估种群内的所有解决方案并为他们赋予适应度值。

3.父代选择：基于适应度以及其它准则，例如解决方案之间的距离，选择将被用于复制的种群成员。选择策略包括了直接或间接地依赖于解决方案的适应度的方法，包括了轮盘赌(roulette-wheel)（与适应度成比例），排名（与种群内的排名成比例）以及锦标赛(tournament)。

4.复制：后代通过父代间的交叉，或简单地复制父代的解决方案，或者通过它们的一些相互组合而生成。

5.变体：变异被应用于部分或者全部的父代和/或后代。

6.替换：在这个步骤，我们选择哪一些父代和/或后代将会成为下一代。普遍的对于当前种群的替换策略包括了多世代方法(generational)（父代死亡，后代替换它们）、稳态方法(steady state)（后代在并且只在后代更为优秀时替换最糟糕的父代）与精英方法(elitism)（多世代，但是父代最好的 x%生存）。

7.终止：我们已经完成了吗？这根据经过了多少个世代或者已经过去了几次评估（耗尽为止），最高的适应度被任何解决方案获得，以及/或者一些其它的终止条件而决定。

8.返回步骤 2。

主循环的每一次迭代（也就是每次我们接触到步骤 2 时）被称为一个**世代**(generation)，与自然启发的术语保持了一致。进行适应性评估的总次数通常与种群规模乘以世代的数目成比例。

这个高级模板可以被以多种不同的方式实现与拓展；在这之外还有着数以千计的进化或者类似进化的算法，并且其中的许多种重新安排了整个流程，添加了一些新的步骤并移除了现有的步骤。为了是这个模板更为具体，我们将会在下面给出一个运作的进化算法的简单案例。这是进化策略的一种形式，也是进化算法的主要家族中的一员。尽管 μ +λ 进化策略是一个简单的，可以被 10 到 20 行代码所实现的算法，但它是一个完整运作的全局优化器并且非常有用。两个主要参数中，μ 象征着“精英”，或者说每个世代中被保存下来的部分种群的大小，λ 是每一个世代中被重新生成的部分种群的大小。

1.将种群填入 μ +λ 个随机生成的解决方案。

2.评估所有解决方案的适应度。

3.将种群按照适应度降序排列，所以有着最小编号的解决方案有着最高的适应度。
4.移除最小适应度的 λ 个个体。
5.将被移除的个体使用最好的 μ 个个体替换。
6.变异后代
7.如果成功或者耗尽则停止。否则返回步骤 2。

上面的μ+λ进化策略算法的类型只是进化策略的一个简单例子，其特点是依赖于变异而不是交叉来创造变体，并且通过自适应的使用来调整变异参数（尽管这并不是上述的简单算法的一部分）。它们通常也非常适合于优化各类表示为实数向量的人造物，即所谓的连续优化(continuous optimization)。一些最佳的连续优化算法，例如协方差矩阵自适应进化策略（covariance matrix adaptation evolution strategy ,CMA-ES）[243]以及神经进化策略(natural evolution strategy ,NES)[754]是这个算法家族概念上的后代。

另一个突出的进化算法家族是**遗传算法**(GAs)。它们的特征是依赖于交叉而不是变体的变异（某些遗传算法根本没有变异），适应度比例选择(fitness-proportional selection)以及解决方案经常被表示为比特串或者其它离散的串。然而，要注意的是，不同类型的进化算法之间的区别主要基于它们的历史渊源。在如今，有着许多的变化以及如此广泛的杂交，这使得将一个特定的算法从属于一个或者其它算法家族通常没有太多意义。

进化算法的一个变体来源于对满足特定约束的需求，其中解决方案不仅要适合并且也要**可行**。当进化算法被用于约束优化时，我们面临着许多挑战，例如变异以及交叉不能保持或者保证一个解决方案的可用性。一次变异或者两个父代之间的结合非常有可能会产生一个不可行的后代。一种解决约束处理的方法是修复(**repair**)，这可以是任何将不可行的后代变为可行的后代的过程。第二种方法是修改遗传算子(genetic operators)，使出现一个不可行的个体的概率变得更小。一个流行的方法是只通过赋予一个低的适应度的值，或者与它们违反约束的次数成比例，来惩罚那些不可行的解决方案的出现。然而这个策略可能会过度惩罚一个解决方案的实际适应度，在另一方面会导致它们讯速地被从种群中消除。这样的一种属性可能是不符合需要的，并且被认为需要对进化算法在处理约束上的不佳表现负责[455]。作为对这个限制的一个回应，**feasible-infeasible 2-population**(FI-2pop)算法[339]进化了两种种群，一种是可行的解决方案，另一种是不可行的解决方案。不可行的种群优化它的成员直到最小化其与可行性之间的差距。当不可行的种群收敛至可行性的边界时，发现新的可行个体的可能性就会增加。不可行的父代产生的可行后代会被转移至可行种群，提高它的多样性（对于不可行的后代，反之亦然）。FI-2pop 已经被用于游戏中那些我们需要合适并且可行的情况中，例如良好设计并且可玩的游戏关卡[649][378]。

最后，另一种进化算法的混合在试图找到某个问题的一个解决方案时考虑了多个目标。在许多问题中，将所有需求以及规范混合到一个目标尺度下是很困难的。这些目标经常都是相互冲突的。例如，如果我们的目标是购买最快并且最便宜的笔记本电脑，我们很快就会意识到这两个目标是部分冲突的。直观的解决方法是仅仅添加不同的目标值——作为一个权重总和——并且在优化中使用其作为你的适应度。然而，这样做有着一些缺点，例如目标间的权重的复杂特化设计，欠缺对于目标之间的相互作用的理解（例如对更快的电脑来说什么样的价格门槛才不算贵？），以及一个权重总和的单目标方法是不能接触到那些在其的各个权重目标中达到最佳妥协的解决方案这一事实。对于这些限制的应对是被称为**多目标进化算法**(multiobjective evolutionary algorithms)的算法家族。一个多目标进化算法至少考虑两个目标函数——它们是部分矛盾的——并且搜索这些目标的一个**帕累托前沿**(Pareto front)。帕累托前沿包括了那些不能在一个目标上被改善却又不损害另一个目标的解决方案。关于进化算法在多目标优化中的使用的更多细节可以在[125]中被找到。这个方法被应用于游戏 AI 里那些有着多个目标与我们试图解决的问题所关联的情况中：例如，我们可能希望同时优化一个战

略游戏地图的平衡性以及对称性[712][713]，或者设计在它们的行为空间中有着非凡的多样性的非玩家角色。

最后，另一种演化算法的混合在试图找到问题的解决方案时考虑了多个目标。对于许多问题，很难将所有要求和规范结合到一个客观的测量中。这些目标往往是相互冲突的。例如，如果我们的目标是购买最快和最便宜的笔记本电脑，我们很快就会意识到这两个目标是部分冲突的。直观的解决方案是仅仅添加不同的客观值 - 作为加权总和，并将其用作优化后的适应度。但是，这样做有一些缺点，例如目标之间权重的非平凡特设计，缺乏对目标之间相互作用的洞察（例如，更快的笔记本电脑的价格阈值不是更昂贵？），加权和单目标方法不能达到在其加权目标中达到最佳折衷的解决方案。对这些限制的反应是被称为多目标进化算法的算法家族。多目标进化算法至少考虑两个目标函数 - 部分相互矛盾 - 并搜索这些目标的 Pareto 前沿。帕累托方面包含的解决方案不能在一个目标中得到改善，而在另一个目标中不会恶化。关于通过演化算法进行多目标优化的更多细节可以在[125]中找到。该方法适用于游戏人工智能，其中有多个目标与我们试图解决的问题相关：例如，我们可能希望优化战略游戏地图的平衡和不对称[712,713]，或者设计非玩家角色，他们的行为空间是有趣的多样化[5]。

2.4.2.1 用于 Ms Pac-Man 的进化算法

在 Ms Pac-Man 中使用进化算法的一个简单方法如下：你可以基于许多 Ms Pac-Man 为了在下一步的移动中取得正确决策而必须考虑的重要参数来设计一个效用函数。举例来说，这些参数可以是鬼魂当前的位置，力量药片的存在与否，在这个方向上存在的豆子的数量等等。而下一步就是设计一个效用函数作为这些参数的权重总和。在每一个交汇处，Ms Pac-Man 将需要为它所有可能的移动问询它的效用函数并且选出有着最高效用的移动。效用函数的权重当然是未知的，这就是一个进化算法可以通过演变效用的权重来优化 Ms Pac-Man 的分数而作为帮助的地方。换句话说，每个染色体（效用的权重向量）的适应度是由 Ms Pac-Man 在多次模拟步骤，或者在对游戏关卡的游玩中所获得分数而确定的。

2.4.3 进阶阅读

我们推荐三本书作为进化算法的进阶阅读：Eiben 和 Smith 的《Introduction to Evolutionary Computing》[182]，Ashlock 的《Evolutionary computation for modeling and optimization》[21]，以及最后，Poli 等人的遗传编程指南[535]。

2.5 监督学习

监督学习是一个接近标记数据(labeled data)与他们对应的属性或特征的潜在函数的算法过程[48]。监督学习的一个流行例子是，一个机器被要求通过如水果的颜色以及大小之类的**特征**(features)或者**数据属性**(data attributes)的集合，在苹果与梨之间进行区分（**标记数据**）。一开始，机器通过查看许多可用的水果案例来学习对苹果与梨进行分类，这些案例一方面包括了每一个水果的颜色与大小，在另一方面也包括它们对应的标签（苹果或梨）。在学习完成之后，在理想情况下机器应该可以判断一个新的并且未见过的水果是否是梨或者苹果，而只需要基于它的颜色以及大小。除了苹果与梨之间的区分，监督学习如今已经被应用于包括金融服务，医疗诊断，欺诈检测，网页分类，图像和语音识别以及用户建模（或许多用户之间）在内的大量应用之中。

很显然，监督学习需要一组标记过的训练案例；因此是有监督的。更具体地说，训练信号是数据上的一组监督标记（例如，这是一个苹果而那一个是梨），而这些数据是这些标记的一组表征（例如，这个苹果的颜色是红色，大小是中等的）。因此，每一个数据样例都是一组标记（也可以说输出）以及与这些标记对应的特征（也可以说输入）的组合。监督学习的最终目标不仅仅是从输入-输出组合中学习，而是得到一个接近（更好，模仿）它们之间的关系的函数。所得到的函数应当可以非常好地映射新的并且未见过的输入与输出组合的实例（例如，我们案例中未见过的苹果和梨），这个属性被称为**泛化性**(generalization)。在这里有一些可以在游戏中见到并且与监督学习相关的输入-输出样例：{ 玩家的健康值，自身的健康值，与玩家的距离 } → { 动作（开枪，逃离，闲置） }；{ 玩家之前的位置，玩家当前的位置 } → { 玩家的下一个位置 }；{ 杀人与爆头的数目，子弹的消耗 } → { 技能等级 }；{ 分数，地图探索，平均心率 } → { 玩家的挫败程度 }；{ Ms Pac-Man 以及幽灵的位置，存在的豆子 } → {Ms Pac-Man 的方向}。

在形式上，监督学习尝试通过给定一组的 N 个训练样本$\{(x_1,y_1),....,(x_N,y_N)\}$而得到一个函数 $f:X\rightarrow Y$。其中 X 和 Y 分别是输入与输出空间；一个监督学习任务有着两个核心步骤，在第一个**训练**步骤，训练样本——属性以及对应的标记——被呈现，并且属性与标记之间的函数 f 被得到。正如我们将在下面的算法列表中看到的，f 可以表示为多个分类规则，决策树，或者数学公式。在第二个**测试**步骤，f 可以被用于预测已经给定属性的未知数据的标记。为了验证 f 的泛化性并且避免对数据的过拟合[48]，常见的做法是在一个新的独立（测试）数据集中使用一个性能评估，例如准确度，其是通过我们训练的函数而被正确地预测的样本的百分比。如果准确度是可接受的，我们就可以使用 f 来预测新的数据样本。

但是我们如何得到这个 f 函数呢？一般来说，一个算法过程会修改这个函数的参数，以使我们可以在数据样本中的给定标记和我们尝试接近的函数之间取得很好的匹配。有许多种方法可以找到并且表示这个函数，每一个都对应一种不同的监督学习算法。这包括了人工神经网络(artificial neural networks)，基于案例的推理(case-based reasoning)，决策树学习(decision tree learning)，随机森林(random forests)，高斯回归(Gaussian regression)，朴素贝叶斯分类器(naive Bayes classifiers)，k 最近邻(k-nearest neighbors)和支持向量机(support vector machines)[48]。可用的监督学习算法的多样性，在某种程度上，也是由于不存在某个单独的学习算法可以在所有的监督学习问题上都有着最佳的表现。这就是众所周知的"天下没有免费的午餐"定理[757]。

在开始讨论特定算法的细节之前，我们要强调的是，标记的数据类型决定了输出的类型，而反过来，也决定了可以应用的监督学习方法的类型。根据标记（输出）的数据类型我们可以定义监督学习算法的三种主要类型。第一种，我们会见到**分类**(classification)[48]算法，其试图预测分类的类型标记（离散或定类的），比如先前案例中的苹果和梨，以及玩家在达到最高分数时的水平。第二种，如果输出数据是一个区间——例如某个游戏关卡的完成时间，或者保留时长——则监督学习的任务是度量的**回归**(regression)[48]。最后，**偏好学习**(preference learning)预测了如等级与偏好之类的有序输出，并尝试推导出表征了这些有序标记的潜在全局排序。这些有序输出的案例包括了对于不同的相机视角的偏好等级，或者对一个特定的声效相对于其它声效的偏好。在偏好学习范式中的训练信号提供了关于我们试图接近的，各个现象的案例之间的相对关系的信息，而回归以及分类则分别提供了关于现象的强度以及类别的信息。

在本书中，我们将重点讨论最被看好以及最流行的监督学习算法在游戏 AI 任务上的一个子集，例如游戏体验（见第 3 章），玩家行为模范或者玩家偏好预测（见第 5 章）。本章节其余部分所概述的三种算法是，人工神经网络，支持向量机以及决策树学习。所涵盖的所有三种监督学习算法都可以被用于分类，预测，或者偏好学习任务。

2.5.1 人工神经网络

人工神经网络（ANNs）是一种生物启发的计算智能和机器学习方法。一个人工神经网络是一组相互关联的处理单元（称为神经元），其一开始被设计用于模拟一个生物大脑——包含超过 10^11 个神经元——在某些任务上处理信息，操作，学习，执行的方式。生物神经元拥有一个细胞体，一些将信息带入神经元的树突和在神经元外部传递电化学信息的轴突。人工神经元（见图 2.12）类似于生物神经元，因为它有着许多**输入 x**（对应神经元的树突），每一个输入都有相关的**权重 w**（对应突触强度）。它也有一个将输入与它们对应的权重通过内积（权重的总和）来结合起来的处理单元，并且添加一个**偏置**(bias)（或者阈值）权重 b 而得到如下的权重总和 :$\mathbf{x}\cdot\mathbf{w}+b$。这个值接下来将被送到一个**激活函数**(activation function) g（细胞体），其会产生神经元的输出（对应于一个树突终端）。人工神经网络本质上是简单的数据模型，定义了一个函数$f: x \rightarrow \mathbf{y}$。

各种形式的人工神经网络可以被用于回归分析，分类，以及偏好学习，甚至无监督学习（通过如赫布学习(Hebbian learning)[254]和自组织映射(self-organizing maps)[345]）。核心应用领域包括了模式识别，机器人与智能体控制，游戏，决策，姿势，语言和文本识别，医疗与金融应用，情感建模和图像识别。人工神经网络相比其它监督学习方法的好处是，能够在给予足够大的人工神经网络结构与计算资源下逼近任何连续的实数函数[347][151]。这个能力使得人工神经网络有着泛逼近性(universal approximators)[277]。

2.5.1.1 激活函数

应当在人工神经网络中使用哪一个激活函数？McCulloch 和 Walter Pitts [449]在 1943 年提出的神经元的原始模型使用一个赫维赛德阶跃激活函数(Heaviside step activation function)，可以使得神经元激发或不激发。当这样的神经元被用于并且连接到一个多层 ANN 时，产生的网络智能解决线性可分的问题。训练这种人工神经网络的算法是在 1958 年被发明的，并且被称为 Rosenblatt 的感知机算法。非线性可分的问题，例如异或门只能在 1975 的**反向传播**算法发明之后才得以解决。现在，已经有好几种激活函数被用于人工神经网络的联结与它们的训练。激活函数的使用在另一方面也产生了不同类型的人工神经网络。这些例子包括了在径向基函数（radial basis function，RBF）网络[70]中使用的高斯激活函数(Gaussian activation function)以及可以在组合型模式生成网络（Compositional Pattern Producing Networks，CPPN）[653]中使用的多种类型的激活函数。最普遍的用于人工神经网络训练的函数是 S 型的 Logistic 函数，其函数为 $(g(x) = 1/(1+e^{-x}))$，有着如下性质：1）它有界，单调并且非线性；2）它是连续并且平滑的；3）它的导数是容易计算的 $g'(x) = g(x)(1-g(x))$。有着上述的属性，Logistic 函数可以与基于梯度的优化算法结合地使用，例如下面描述的反向传播算法。其它流行的用于训练神经网络的**深度**结构的激活函数包括了 **rectifier**——被用于神经元时叫做**修正线性单元**(rectified linear unit, ReLU)——以及它的平滑逼近，**softplus** 函数[229]。与 S 型的激活函数想比较，ReLU 可以更快并且（从经验上来说）更为有效的训练深度人工神经网络，其通常在大型数据集中进行训练（更多内容请参阅章节 2.5.1.6）。

2.5.1.2 从一个神经元到一个网络

为了形成一个人工神经网络，许多神经元需要被结构化以及连接。尽管在文献中已经提出了非常多种方法，但其中最常见的那些全部都是将神经元结构化为各个层次。在它最简单

的形式，也被称为**多层感知机**(multi-layer perceptron, MLP)中，一个人工神经网络中的神经元被分为一个或者更多的层次，但是不会连接到同一层中的其它神经元（见图 2.13，一个典型的多层感知机结构）。每一层的每一个神经元的输出都会连接到下一层中的所有神经元。要注意的是，一个神经元的输出值只馈送给下一层的神经元，并成为他们的输入。因此，在最后一层中的神经元的输出将是人工神经网络的输出。人工神经网络的最后一层也被称为**输出层**(output layer)，而输出层与输入层之间的所有中间层都是隐藏层(hidden layers)。值得注意的是，人工神经网络的输入，x，与第一层中的所有神经员连接。我们用一个额外的层来说明这一点，我们称之为**输入层**(input layer)。输入层不包含神经元，因为它只是将输入分配到神经元的第一层。总的来说，多层感知机是：1）分层的，因为它们根据层次来分组；2）前馈的，因为它们的连接是单向的并且总是向前（从前一层到下一层）；3）全连接的，因为每一个神经元都会被连接到下一层中的所有神经元。

2.5.1.3 前向操作

在先前的章节中我们定义了一个人工神经网络的核心组件，而在这个章节中我们将会看见当一个输入特征出现时，我们如何计算人工神经网络的输出。这个过程被称为**前向操作**(forward operation)，将人工神经网络的输入穿过它的隐含层来产生输出。前向操作的基础步骤如下：

1. 为神经元进行标记与排序。我们通常从输入层开始编号，向输出层递增数字（见图 2.13）。注意输入层并不包含神经元，只是出于编号上的目的而做同样的处理。

2. 假设 w_{ij} 是从神经元 i（突触前神经元）到神经元 j（突触后神经元）的连接权重，为这个连接权重进行标记。将连接到神经元 j 的偏置权重(bias weight)标记为 b_j。

3. 提交一个输入特征x。

4.对于每个神经元 j，计算它的输出：$a_j = g\left(\sum_i \{w_{ij} a_i\} + b_j\right)$，其中$a_j$与$a_i$分别是神经元 j 的输出与输入（注意在输入层中$a_j = x_i$）；g 是激活函数（通常是 logistic sigmoid 函数）。

5.输出层的神经元的输出就是人工神经网络的输出。

2.5.1.4 人工神经网络如何学习？

我们应当如何逼近f(x, w, b)来使得人工神经网络的输出来匹配我们的数据集 y 所期望的输出（标记）呢？我们将会需要一个会调整权重（w与b）训练算法来使得f: x → **y**。一个这样的训练算法需要两个不见。首先，它需要一个代价函数来评价任何一组权重的质量。其次，它需要一个在可行解空间中的搜索策略（例如，权重空间）。我们在西面的两个子段列出了这些方面。

代价（误差）函数：

在我们尝试调整权重来接近f之前，我们需要一些对多层感知机的表现的衡量方式。对于在一种监督方式下训练人工神经网络来说，最为普遍的表现衡量方式是在人工神经网络(α)的实际输出的向量与所期望的标记输出 y 之间的平方欧氏距离（误差）（见公式 2.2）。

$$E = \frac{1}{2}\sum_j \left(y_j - a_j\right)^2 \tag{2.2}$$

其中的总和是取所有的输出神经元（在最终层的神经元）的之和。注意是y_j标记是常数值，并且更重要的是，也要注意E是人工神经网络的所有权重的一个函数，因为实际输出是取决于它们。正如我们将在下面看到的，人工神经网络训练算法极大地建立在这种误差与权重之间的关系之上。

反向传播：

反向传播(Backpropagation 或者 Backprop)[578]是基于梯度下降的方法并且它可以说是最普遍的用于训练人工神经网络的算法。反向传播代表着误差的向后传播，因为它计算了权重的更新，其从输出层至输入层来最小化了误差函数——也就是我们之前定义的(2.2)。简而言之，反向传播计算误差函数E相对于 ANN 的每个权重的偏导数（梯度）并跟随可以最小化E的梯度（的相反方向）来调整 ANN。

正如前面提到的，（2.2）的平方欧式误差取决于神经网络输出的权重，其在本质上就是$f(x, \mathrm{w}, \mathrm{b})$函数。因此我们可以计算$E$相对于神经网络中的任意权重$\left(\frac{\theta E}{\theta w_{ij}}\right)$与任意偏置权重$\left(\frac{\theta E}{\theta b_j}\right)$的梯度，其反过来也会决定，如果我们改变权重值，误差的改变程度。我们然后可以通过一个称为**学习率**(learning rate)的参数$\eta \in [0,1]$来决定我们需要多少这样的改变。在缺少任何关于误差与权重之间的函数的通常形状的信息时而只有关于它的梯度的信息时，很明显**梯度下降**(gradient descent)将会是一个好的方法来尝试寻找E函数的全局最小值。由于缺乏关于E函数的信息，搜索可以从权重空间中的一些随机点（例如随机初始化的权重值）开始并且沿着梯度来接近更低的E值。这个过程被迭代地重复，直到我们得到了我们觉得满意的E值，或是我们耗尽了计算资源。

更正式来说，**反向传播**算法的基本步骤如下：

1.将w与b初始化为随机值（通常比较小）。
2.对于每个训练特征（输入-输出的组合）：
(a) 表示输入特征x，最好规范化(normalized)到一个范畴（例如[0,1]）。
(b) 使用前向操作计算人工神经网络的实际输出α_j。
(c) 根据(2.2)计算E。
(d) 从所有输出层到输入层，计算误差对于人工神经网络中的每一个权重与偏置权重的导数。
(e) 分别更新权重与偏置权重。
3. 如果E已经*很小*或是你已经耗尽了*计算预算*，停止！不然回到步骤 2。

注意我们并不希望具体描述步骤 2(d)的导数计算，因为这样做会超出本书的范畴。作为替代，我们为有兴趣的读者转到了最早的反向传播论文[578]来获取额外的公式，或是在这个章节结尾处的阅读列表。

限制与解决方法：

要注意的是，鉴于其的局部搜索（爬山法）属性，反向传播不能保证找到E的全局最小值。除此之外，鉴于它基于梯度的（局部）搜索的性质，该算法无法克服在误差函数范畴内潜在的高原区域。由于这些地区有着接近零的梯度，越过它们会导致接近零的权重更新并进

一步导致算法的过早收敛。该算法用于克服在局部最小值处的收敛的典型的解决方法与提高包括了：

•**随机地重新开始**：人们可以使用新的随机连接权重值来重新运行算法，来期望人工神经网络不要过分地依赖运气。没有一个过于依赖运气的人工神经网络模型是优秀的——例如，它在超过十次的运行中只有一次或者两次表现良好。

•**动态的学习率**：人们可以修改学习率参数并观察人工神经网络的表现上的改变，或是引入一个动态的学习率参数，当收敛慢的时候增加并在收敛到更低的E值的速度很快时减少。

•**动量**(Momentum)：或者，人们也可以如下所示，添加一个总动量到权重更新规则中：

$$\Delta w_{ij}^{(t)} = m\Delta w_{ij}^{(t-1)} - \eta \frac{\theta E}{\theta w_{ij}} \tag{2.2}$$

其中$m \in [0,1]$是动量参数，t是权重更新的迭代次数。前面的权重更新添加的动量值$\left(a\Delta w_{ij}^{(t-1)}\right)$尝试帮助反向传播克服一个潜在的局部最小值。

虽然上述解决方案直接适用于小规模的人工神经网络，然而，现代（深层次）的人工神经网络架构上的实践智慧和经验证据表明，上述缺点已经在很大程度上被消除[365]。

批量 vs 非批量训练：

反向传播可以采用批量或者非批量学习模式。在**非批量**模式(non-batch mode)下，每次向人工神经网络提供样本时，权重都会被更新。在**批量模式**(batch mode)下，权重将会在所有训练样本都被提供给人工神经网络之后更新。在这种情况下，在误差将会在权重更新之前在批量处理的样本上累计。非批量模式更不稳定，因为它迭代地依赖于一个单独的数据点；然而，这可能有利于避免收敛到一个局部最小值。在另一方面，批量模式很自然地是一个更为稳定的梯度下降方法，因为权重更新是由该批中的所有训练样本的平均误差所驱动的。为了充分利用这两种方法的优点，经常在比较小的批量规模上应用随机选择的样本进行来批量学习。

2.5.1.5 人工神经网络的类型

在标准的前馈式多层感知机之外，也有许多种其它类型的人工神经网络被用于分类，回归，偏好学习，数据处理与筛选，以及聚类任务。要注意的是，**递归**神经网络(recurrent neural networks)（例如 Hopfield 网络[276]， 玻尔兹曼机(Boltzmann machines)[4]与长短时记忆模型(Long Short-Term Memory)[264]）允许神经元之间的连接形成有向循环，从而使一个人工神经网络能够捕捉动态与时间上的现象（例如时间序列处理与预测）。此外，还有一些人工神经网络类型主要用于聚类与数据降维，例如 Kohonen 自组织映射[345]和自动编码器(Autoencoder)[40]。

2.5.1.6 由浅到深

人工神经网络训练的一个关键参数是人工神经网络的规模。所以，我的人工神经网络结构应该有多宽与多深才可以在这个特定的任务上表现良好？尽管这个问题没有正式和明确的答案，但被普遍接受的经验法则建议，网络的规模应当与问题的复杂度相匹配。根据

Goodfellow 等人在他们的深度学习书籍[229]提出的，一个多层感知机本质上是一个深度（前馈式）神经网络。它的深度由取决于他包含的隐藏层的数量。Goodfellow 等人指出："正是由于这个术语，才产生了**深度学习**(deep learning)这个名字"。在此基础上，（至少）包含了一个隐藏层的人工神经网络结构的训练可以被视为一个深度学习任务，而单一的输出层结构可以被视为浅层。近年来许多方法都被引入来训练包含有多层的深度结构。这些方法很大程度上依赖于梯度搜索，对于有兴趣的读者来说，这些方法在[229]中得到了详细的介绍。

2.5.1.7 用于 Ms Pac-Man 的人工神经网络

正如本章中的其它方法一样，我们将尝试在 Ms Pac-Man 游戏中应用人工神经网络。在 Ms Pac-Man 中使用人工神经网络的一个直接的途径是尝试模仿游戏的专业玩家。因此，可以请专家玩这个游戏并且记录下他们的游戏过程，许多特征可以从中得到提取并被用于人工神经网络的输入。人工神经网络输入的分辨率可以在简单的游戏统计数据——例如鬼魂与 Ms Pac-Man 之间的平均距离——到具体的游戏关卡图像的逐个像素上的 RGB 值之间变化。在另一方面，输出数据可以包含 Ms Pac-Man 在游戏的每一帧上所选择的动作。给定输入与期望的输出组合，人工神经网络通过反向传播被训练来预测在当前游戏状态（人工神经网络输入）中被专业玩家所选执行的动作（人工神经网络输出）。人工神经网络的规模（宽度与深度）同时取决于可用的来自 Ms Pac-Man 专业玩家的数据总量以及被考虑到的输入向量的大小。

2.5.2 支持向量机

支持向量机(SVM)[138]是一组可以替代并且非常流行的监督学习算法，其可以被用于分类，回归[642]以及偏好学习[300]任务。一个支持向量机是一个二元线性分类器，其被训练用于最大化数据中不同类型（例如苹果与梨）的训练样本之间的间隔(margin)。和其它所有的监督学习算法一样，新的并且未曾见过的属性将被输入到可以预期它们所属的 SVM。SVM 在许多其它领域中已经被广泛用于文本分类，语音识别，图像分类与手写字符识别等。

与人工神经网络类似，SVM 构造了一个超平面，其将输入空间分开，并且表示在输入与目标输出之间映射的函数f。与沿着误差的梯度隐式地尝试最小化模型的实际输出与目标（正如反向传播那样）不同，SVM 构建了一个与最接近的任何其它种类的训练数据点都保持着最大距离的超平面。该距离被称为最大间隔(maximum-margin)，并且它对应的超平面在一个总共有 n 个样本的数据集中将带有标记(y_i)1 的点($\mathbf{x_i}$)与那些带有标记-1 的区分开来。换句话说，在推导出的超平面与来自其它类的点 $\mathbf{x_i}$ 之间的距离被最大化。给定一个训练集 **x** 的输入属性，则一个超平面的通常形式可以被定义为：$\mathbf{w}\cdot\mathbf{x}-b=0$，其中，就像在反向传播训练中那样，**w**是超平面的权重向量（法向量），$\frac{b}{\|\mathbf{w}\|}$从原点确定了超平面的偏移量（或者权重阈值/偏差）（见图 2.14）。因此，在形式上，一个 SVM 是一个函数$f(x,\mathrm{w},b)$，其预测了目标输出(**y**)并且试图：

$$\textbf{minimize } \|\mathbf{w}\| \tag{2.4}$$

$$\textbf{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i-b)\geq 1, \text{for } i=1,\ldots,n \tag{2.5}$$

权重**w**与b决定了 SVM 分类器。距离推导出超平面最近的向量$\mathbf{x}_i$被称为**支持向量**(support vectors)。如果训练数据是线性可分的（也被称为一个**硬间隔**(hard-margin)分类器任务；见图

2.14)，则上述问题是可解的。如果数据不是线性可分的（**软间隔**(soft-margin)）,SVM 将转而尝试：

$$\textbf{minimize}\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0,1-y_i(\mathbf{w}\cdot\mathbf{x}_i-b)\right)\right]+\lambda\|\mathbf{w}\|^2 \tag{2.6}$$

如果等式 2.5 的硬性约束被满足，这等于$\lambda\|\mathbf{w}\|^2$——也就是如果在间隔右侧的所有的数据点被正确地分类。等式 2.6 的值与到误差数据的间隔的距离成比例，并且λ被设计成从质量上来决定间隔规模应当增加的程度，确保$\mathbf{x}_i$将会位于间隔的正确边上。显然，如果我们为λ选择一个小的值，我们将近似用于线性可分数据的硬间隔。

用于训练软间隔的标准方法是将学习任务视为一个二次规划问题，并且搜索$\mathbf{w}$和b来寻找最宽的，可以匹配全部数据点的间隔。其它方法包括了子梯度下降(sub-gradient descent)与坐标下降(coordinate descent)。

除了线性分类任务之外，SVM 还可以通过使用许多不用的非线性**核函数**(kernels)来支持非线性分类，其将输入空间映射到高维的特征空间。SVM 任务依然相似，除了每一个点积都被一个非线性的核函数所替代。这使得算法可以在一个变换的特征空间中拟合最大间隔，超平面。常见的与 SVM 结合使用的核函数包括了多项式函数(polynomial functions)，高斯径向基函数(Gaussian radial basis functions)或双曲正切函数(hyperbolic tangent functions)。

虽然 SVM 一开始是被设计为处理二元分类问题的，但仍然有几种可以处理多类别分类[282]，回归[642]与偏好学习[300]的 SVM 变体，有兴趣的读者可以进行参考。

SVM 比起其它监督学习方法有着许多优点。他们在处理大而稀疏的数据集时，可以非常高效的寻找解，因为它们仅依赖支持向量来构造超平面。它们也可以很好的处理巨大的特征空间，因为学习任务的复杂性不依赖于特征空间的维度。SVM 可以被视为一个简单的凸优化问题，其可以保证收敛到一个单一的全局解。最后，过拟合可以很容易地通过软间隔分类器方法来控制。

2.5.2.1 用于 Ms Pac-Man 的 SVM

与人工神经网络类似，SVM 可以被用于模仿 Ms Pac-Man 专业玩家的行为。关于特征（输入）空间与动作（输出）空间的考量仍然一致。除了输入与输出向量的设计，从专业玩家处获取的数据的规模与质量也会决定 SVM 在控制 Ms Pac-Man 最大化它的分数时的表现。

2.5.3 决策树学习

在**决策树学习**(decision tree learning)[66]中，我们尝试导出的函数使用了一种决策树的表示方法，其将数据观测的属性映射到它们的目标值。前者（输入）被表示为节点，而后者（输出）被表示为叶节点。每个节点（输入）的可能值由该节点的多种分支所表示。与其它的监督学习算法一样，决策树可以根据它们尝试学习的输出数据类型进行分类。特别是，决策树可以分为分类，回归与排序树，如果目标输出分别是有限的一组值，一组连续的（有间隔）值，或者是一组观察值(observations)之间的有序关系。

一个决策树的例子如图 2.15 所示。树节点对应输入属性；对于每个输入属性的每个可能值，都存在到子节点的分支。更深一层的叶节点表示输出的值——在这个案例中是车的类型——给定输入的值，根据根节点到叶节点的路径来决定。

决策树学习的目标是构建一个映射（一个树模型），其基于大量输入属性来预测目标输

出的值。最基本并且最普遍的用于从数据中学习决策树的方法是使用一个自顶向下的**递归**树归纳策略，其具备一个贪婪过程的特点。这个算法假设输入属性与目标输出二者都是有限的离散域，并且具备分类性质。如果输入或者输出是连续值，他们可以在构建树之间离散化。一个树是通过将可用的训练数据集基于根据数据集而做出的选择来分割为子集而逐渐构建的。这个过程以递归方式在逐个属性的基础上进行重复。

有几种上述方法的变体导致了不同的决策树算法。但决策树学习最著名的两种变体是**Iterative Dichotomiser 3**(ID3)[543]与它的继承者 **C4.5**[544]。基本的树学习算法有着如下的通用步骤：

1. 在开始时，所有的训练样本都是树的根节点。

2. 在一个启发式的基础上选择一个属性，并且是挑选有着最高启发式值的属性。两个最普遍的启发式如下：

•信息增益(Information gain)：这个启发式被同时用于 ID3 与 C4.5 树生成算法。信息增益$G(A)$基于来源自信息论的熵的概念，并且衡量了数据集D在属性A上进行了分割之前与之后熵H的差异。

$$\mathrm{G}(A) = H(D) - H_A(D) \tag{2.7}$$

其中$H(D)$是D的熵（$H(D) = -\sum_i^m p_i \log_2(p_i)$）；$p_i$是$D$中的一个任意样本属于类别$i$的概率；$m$是类别的总数；$H_A(D)$是对$D$进行分类的所需的信息（在使用属性$A$将$D$分为$v$部分后），并且计算为$H_A(D) = -\sum_j^v \left(\left|D_j\right|/|D|\right) H\left(D_j\right)$，其中$|x|$是$x$的大小。

•信息增益率(Gain ratio)：C4.5 算法使用信息增益率启发式来减少信息增益在有着大量值的属性上的偏差。信息增益率通过在选择某个属性时考虑分支的大小来规范化信息增益。信息增益率也就是信息增益与属性 A 的内在值IV_A之间的比率：

$$\mathrm{GR}(A) = G(A)/IV_A(D) \tag{2.8}$$

其中

$$IV_A(D) = -\sum_j^v \frac{|D_j|}{|D|} \log_2\left(\frac{|D_j|}{|D|}\right) \tag{2.9}$$

3. 基于从步骤 3 选择的属性，构建一个新的树节点，并且根据选择的属性的可能值将数据集分为多个子集。属性的可能值称为节点的分支。

4. 重复步骤 2 与 3 直到如下之一发生：

- 给定节点的所有样本都属于同一类别。
- 没有剩余属性可以用来进一步划分。
- 没有数据样本遗留。

2.5.3.1 用于 Ms Pac-Man 的决策树

与人工神经网络以及 SVM 一样，决策树学习需要数据进行训练。假设这些来自 Ms Pac-Man 的专业玩家的数据的质量与数量都很好，决策树可以被构建，并基于多种游戏状态的特

定设计属性来预测 Ms Pac-Man 的策略。图 2.16 展示了一个假想的简化过的用于控制 Ms Pac-Man 的决策树。根据这个例子，如果附近存在鬼魂，那么 Ms Pac-Man 会检查近距离上是否有可用的力量药丸，并以其为目标；否则它将采取动作来逃避鬼魂。另外如果鬼魂不在视野内，Ms Pac-Man 将会为豆子进行检查。如果其就在附近或者在一个可行的距离上，那么它将会以其为目标；否则它将会以水果为目标，如果其在这一关卡中存在。需要注意的是，在我们的案例中树的叶节点表示了 Ms Pac-Man 的控制策略（宏观动作）而不是实际动作（上，下，左，右）。

2.5.4 进阶阅读

核心的监督学习算法被具体的涵盖在了 Russell 与 Norvig 的经典 AI 教材[581]内，包括了决策树学习（第 18 章）与人工神经网络（第 19 章）。人工神经网络与反向传播的详细描述可以在 Haykin 的书[251]内被找到。人工神经网络的深层结构也在 Goodfellow 等人的深度学习书籍[229]中得到了非常详细的叙述。最后，支持向量机被涵盖在了 Burges 的教程论文中[85]。

浅层与深层结构中的反向传播的偏好学习版本可以在[429, 435]中被找到，而 RankSVM 被涵盖在了 Joachims 的原始论文[85]中。

2.6 强化学习

强化学习(RL)[672]是一种机器学习方法，其受到了行为心理学的启发，特别是人类与动物通过从环境中受到（正面或者负面的）奖赏而采取决策的方式。在强化学习中，有着良好行为的样本通常是不存在的（就像在监督学习中那样）；相反，与进化（强化）学习类似，算法的训练信号由环境根据一个智能体与其互动的如何来提供。在时间t的一个特定点，智能体在一个特定的**状态**s上并且在它当前的状态中所有可用的动作中采取一个**动作**a。作为一个回馈，环境给予了一个瞬时的**奖赏**(reward)，r。通过智能体与它所在环境之间的连续互动，智能体逐渐地学习到了去选择会最大化其奖赏总数的动作。强化学习已经被从多种学科的角度进行了研究，包括了运筹学，博弈论，信息论和遗传算法，并已经被成功应用到涉及了长期奖赏与短期奖赏之间的一个平衡的问题中，例如机器人控制与游戏[463, 628]。一个强化问题的案例可以见参见图 2.17 中的一个迷宫导航任务。

更正式地说，智能体的目标是探索一个**策略**(policy) (π)，这是为了选择可以最大化如期望累积奖赏之类的某种长期奖赏的一种估量的动作。一个策略也就是智能体在考虑到它所处的状态后，在选择动作时会遵守的一种方法。如果表征了每个行动的价值的函数存在或者被学习，最优策略(π^*)可以通过选择有着最高值的动作而得到。与环境的交互发生在离散的时序（$t = \{0,1,2,\ldots\}$）并且被建模为一个**马尔科夫决策过程**(Markov decision process, MDP)。MDP 被定义为：

•S：一组状态$\{s_1, \ldots, s_n\} \in S$。环境状态是一个智能体拥有的有关于环境的信息（如智能体的输入）的函数。

•A：一组动作$\{a_1, \ldots, a_m\} \in A$在每个状态$s$中都是可能的。动作表示了智能体可以在环境中行动的不同方式。

•$P(s, s', a)$：给定a的情况下从s到s'的转移概率。P给出了在从状态s中挑出了动作a之后结束于状态s'的概率，并且它遵循**马尔科夫性质**(Markov property)，这暗示着这个过程过程未来的状态仅依赖于当前状态，而不取决于之前的事件序列。因此，P的马尔科夫事件使得对一步之后的动态的预测成为可能。

•$R(s,s',a)$：给定a的情况下从s到s'的奖赏函数。当一个智能体在状态s挑选了一个动作a并且移动到状态s'，它从环境处收到了一个瞬时的奖赏r。

P与R定义了**世界模型**，并且为每个策略都分别表示了环境的动态性（P）与长期奖赏（R）。如果世界模型是*已知的*，则不再需要学习去评估转移概率与奖赏函数，因此我们可以使用**基于模型**的方法，例如动态规划[43]，来直接地计算出最佳方法（策略）。反过来，如果世界模型是*未知的*，我们通过学习由在状态s中挑选动作a而给出的对未来奖赏的估计来逼近转移函数与奖赏函数。我们然后基于这些估计来计算我们的策略。学习通过**无模型**(model-free)方法发生，例如蒙特卡罗搜索与**差分学习**[672]。在这个章节中我们将重点放在后一组算法上，特别是最常用的差分学习：Q-Learning。在深入研究 Q-Learning 算法的细节之前，我们首先讨论了一些核心的强化学习概念，并根据强化学习问题以及用来解决它们的工具提供了一个高等级的强化学习分类方法。我们将使用这个分类方法来在整体上谈及强化学习中的 Q-Learning。

2.6.1 核心概念与一个高等级的分类方法

在强化学习问题中的一个中心问题是在对当前学习到的知识的**利用**(exploitation)与对在搜索空间中全新并且为见过的区域的**探索**(exploration)。随机地选择动作（在没有利用时）与根据一个表现或者奖赏的估量来贪婪地选择最好的动作（没有探索）通常都是会在随机环境中产生不良结构的方法。尽管有多种方法已经在文献中被提出用于解决探索-利用平衡问题，但对于强化学习动作选择来说，一个普遍并且相当有效的机制被称为ε-greedy，其由参数ε ∈ [0,1]而决定。根据ε-greedy，强化学习智能体会选择它认为会以 1−ε 的概率返回最高的未来奖赏的动作；否则，它会统一地随机选择一个动作。

强化学习问题可以被分类为**情节性**(episodic)或者**增量性**(incremental)。在前一种类别中，算法的训练发生在离线中并且在在一个有限的有着多训练样例的范畴内进行。在这个范畴内收到的有限的状态，动作与奖赏信号的序列被称为一次**片段**(episode)。例如，依赖于重复随机采样的蒙特卡罗方法是情节性强化学习的一个典型案例。相反，在算法的后一种类别中，学习是在线发生的并且它不会被一个范畴所限制。我们将差分学习视为增量性强化学习算法。

另一种区分是在离线(off-policy)与在线(on-policy)强化学习算法之间。一个离线学习器仅通过智能体的动作来逼近最佳策略。正如我们将在下面看到的，Q-Learning 是一种离线学习器，因为它在假设了一个贪婪策略的情况下，估计了对状态-动作组合的奖赏。而一个在线强化学习算法将策略近似为一个包括探索步骤在内的与智能体的动作相关连的过程。

自举(Bootstrapping)是强化学习中的一个中心概念，它基于它们优化状态值的方式对算法进行了分类。自举在估计一个状态有多好时候是基于我们认为下一个状态有多好。换句话说，通过自举，我们更新一个估计是基于另一个估计。差分学习与动态规划都使用了自举来从访问状态的经验中学习，并且更新它们的值。不过蒙特卡罗搜索方法没有使用自举，而是分别学习了每一个状态值。

最后，备份(backup)的概念在强化学习中也是很关键的，并且是强化学习算法中的一个独特特征。通过备份，我们从未来的一个状态$s_{t+\mathrm{h}}$回溯到我们希望评估的（当前）状态s_t，并考虑在我们的估计中处于中间的值。备份操作有两个主要属性：它的**深度**——其差异可以从一步的回退到一个完全的备份——以及它的**宽度**——其差异可以从一个在每次时序中（随机）选择的抽样状态数目到一个完整宽度的备份。

基于上述准则我们可以确认三种主要的强化学习算法类型：

1. **动态规划**。在动态中，需要世界模型的知识（P与R）并且最优策略可以通过自举而计算得到。

2. **蒙特卡罗方法**。对于蒙特卡罗方法来说世界模型的知识是不需要的。这个种类的算法（如 MCTS）对于离线（情节性）训练来说很理想，它们通过与样本一致的宽度以及完全深度的备份进行学习。然而，蒙特卡罗方法不使用自举。

3. **差分学习**。就像蒙特卡罗方法一样，世界模型的知识是不需要的，并且会被估计。这个类型的算法（如 Q-learning）通过自举与备份的变体从经验中进行学习。

在接下来的章节中我们涵盖了在强化学习文献中最流行的差分学习算法，其也被广泛用于游戏 AI 研究。

2.6.2 Q-Learning

Q-learning [749]是一种无模型，离线的差分学习算法，其依赖于一个表格化的对$Q(s,a)$值的表示（并因此得名）。非正式地来说，$Q(s,a)$表示了在状态s上挑选了动作a会有多好。而正式地说来，$Q(s,a)$是状态s上采取动作a的预期折扣强化。Q-learning 智能体通过挑选动作并且经由自举来接受奖赏来从经验中进行学习。

Q-learning 智能体的目标是通过在每一个状态挑选正确的动作来最大化它的期望奖赏。特别是，该奖赏是一个对折扣过的未来奖赏的期望值的权重总和。Q-learning 算法是在一个迭代过程中在Q值上进行的简单更新。最初，Q表格有着由设计者设定的任意值。然后每一次智能体从状态中选择了一个动作a，它访问了状态s，接受到了一个瞬时奖励r，并且更新它的$Q(s,a)$如下：

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left\{r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right\} \tag{2.4}$$

其中$\alpha \in [0,1]$是**学习率**，而$\gamma \in [0,1]$是**折扣因子**(discount factor)。学习率决定了Q的新估计将要覆盖旧估计的程度。折扣因子衡量了早期与后期之间的重要性；γ越接近 1，就会给未来的强化越大的权重。正如公式(2.10)中所见，这个算法使用自举，因为它是根据它对下一个局面有多好（如$Q(s',a')$）的认识来维护对某个状态-动作配对有多好（如$Q(s,a)$）的估计的。它也使用了一个单步深度，完整宽度的备份，通过将新访问的状态s'的所有可能动作a'的所有Q值都纳入考虑来估计Q。已经证明，通过使用方程(2.10)的学习规则，$Q(s,a)$将会收敛到期望的未来折扣奖赏[749]。然后可以基于 Q-values 来计算最佳策略；在状态s中的智能体选择有着最高$Q(s,a)$值的动作a。总而言之，算法的基本步骤如下：

对每个状态中的所有可能动作给定一个瞬时奖励函数r与一个$Q(s,a)$值的表格：

1. 使用任意Q值初始化表格；例如$Q(s,a) = 0$。
2. $s \leftarrow$ 初始状态。
3. 当未结束时*执行：

(a) 基于从Q得到的策略选择一个动作a（例如ε-greedy）
(b) 应用这个动作，转移到状态s'，并且接受一个瞬时奖励r。
(c) 如每一次(2.10)那样更新$Q(s,a)$的值。
(d) $s \leftarrow s'$。

*被最普遍使用的终止条件是算法的速度——例如，在一定次数的迭代之后停止——或者收敛的质量——例如，如果你对获得的策略满意则停止。

2.6.2.1 Q-Learning 的限制

Q-learning 有着许多的限制，主要是与它的表格表示有关。首先，依赖于被选中的状态-动作表示，状态-动作的空间的规模从计算角度来说，处理起来可能会变得非常昂贵。随着Q表格的增长，我们对于内存分配与信息检索的计算需求也会增加。此外，我们可能会经历非常久的收敛过程，因为学习时间与状态-动作空间的规模呈指数关系。为了克服这些障碍并且从强化学习学习器中获得适合的表现，我们需要设计一种减少状态-动作空间的方法。章节 2.8 概述了使用人工神经网络作为 Q-value 函数逼近器的方法，直接地绕过了 Q-table 的限制并且为我们的强化学习学习器生成了压缩过的表示。

2.6.2.2 用于 Ms Pac-Man 的 Q-Learning

只要我们定义了一个适合的状态-动作空间，并且设计一个合适的奖赏函数，Q-learning 就可以用来控制 Ms Pac-Man。Ms Pac-Man 中的一个状态可以被直接地表示为游戏当前的快照——也就是，Ms Pac-Man 在哪里以及鬼魂在哪里，还有哪些豆子与力量药片仍然可用。然而，这个表示对于要构建与处理的 Q-table 来说，会产生一个难以维系的游戏状态数目。因此可能更倾向于选择一个更为间接的表示，例如是否有鬼魂以及附近是否有豆子。对于 Ms Pac-Man 来说，可能的动作可以是它保持现在的方向，转向后方，转向左边，或者是转向右边。最后，奖赏函数可以被设计为在吃豆子，鬼魂或者一个力量药片时正面地奖赏 Ms Pac-Man，并可以在它死亡时候惩罚 Ms Pac-Man。

值得注意的是，Pac-Man 与 Ms Pac-Man 都遵循马尔科夫性质，在这个情况下，任何未来的游戏状态都可以仅依赖于当前的游戏状态。然而，这里有一个核心的区别：尽管 Pac-Man 中的转移概率由于它的确定性是已知的，在 Ms Pac-Man 中由于在这个游戏中鬼魂的随机行为，它在很大程度上是未知的。因此，Pac-Man 在理论上可以通过基于模型的方法（例如，动态规划）来解决，而 Ms Pac-Man 的世界模型只能通过如差分学习之类的无模型方法来逼近。

2.6.3 进阶阅读

高度推荐 Sutton 与 Barto 的强化学习书籍[672]，其全面呈现了强化学习，包括了 Q-learning（第六章）。这本书可以从网上免费获取。这本书的最新版本（2017）的一份草稿也已经可以获取了。Kaelbling 等人的调查[314]是对所涉及的方法的另一个推荐阅读。在最后，对于基于模型的强化学习方法的深入分析，可以参考 Bertsekas 的动态规划书籍[43]。

2.7 无监督学习

就像前面提及的那样，效用类型（或者说训练信号）决定了 AI 算法的类别。在监督学习中训练信号被提供为数据标记（目标输出），在强化学习中它被导出为一个来自环境的奖励。而无监督学习转而尝试通过在所有数据的属性之中搜索模式来探索输入之间的联系，并且不需要拥有一个目标输出——一个通常受 Hebbian 学习[254]与自映射[20]的原则影响的机器学习过程。通过无监督学习，我们集中关注数据的内在结构与关联，而不是试图模范或者

预测目标值。我们涵盖了两种无监督学习任务以及对应的算法：**聚类**与**频繁模式挖掘**。

2.7.1 聚类

聚类是寻找大量数据点的未知类别以使在一个类别（或者是聚类）中的数据可以互相相似并且与其它类别都不为相似的无监督学习任务。聚类已经在检测多个属性之间的数据分组，以及数据压缩，噪声平滑，异常检测与数据集分区之类的数据缩减(data reduction)任务上得到了应用。聚类对于游戏而言，在玩家建模，游玩游戏与内容生成上有着关键性的作用。

就分类来说，聚类将数据放入各个类别；然而，各个类别的标签，是未知的先验，并且聚类算法的目的是通过迭代地评估它们的质量来探索它们。因为正确的聚类是未知的，相似性（以及不相似性）仅仅取决于使用的数据属性。好的聚类具有两个核心属性：1）簇*之内*的相似度高，或者是紧密性(compactness)高，2）簇*之间*的相似性低，或者有良好的分离性(separation)。一种普遍的对紧密性的度量方式是簇内的所有样本与最接近的表示点之间的平均距离——例如质心(centroid)——就被用于 k-means 算法。对分离性的衡量方式的例子包括了**单链**(single link)与**全链**(complete link)：前者是一个簇中的任意样本与其它簇中的任意样本之间的*最小*距离；后者是一个簇中的任意样本与其它簇中的任意样本之间的*最大*距离。尽管紧密性与分离性是对簇有效性的客观度量方式，但值得注意的是它们并不是关于聚类的意义的指标。

除了上述的有效性度量方法，聚类算法还由一个**成员函数**与一个**搜索过程**所定义。成员函数定义了与数据样本相关的簇的结构。搜索过程则是一个我们会遵循的策略，在给定一个成员函数与一个有效性度量方法的情况下对我们的数据进行聚类。这种策略的例子包括了一次将所有的数据点一次性分成各个簇，或者递归地合并（或者分裂）各个簇（就像在层次聚类(hierarchical clustering)中）。

聚类可以通过包括层次聚类，k-means[410]，k-medoids，DBSCAN[194]与自组织映射[345]在内的多种算法来实现。这些算法在定义什么是聚类，以及如何形成聚类上的方式各自不同。选择一个适合的聚类算法以及它对应的参数，例如什么使用什么距离函数或者所期望的簇的数目，取决于研究的目的以及可用的数据。在这个章节的剩余部分，我们概述了我们认为对于 AI 在游戏中的研究最为有用的聚类算法。

2.7.1.1 K-Means 聚类

K-means[410]是一种矢量量化方法，被认为是最普遍的聚类算法，因为它在简单性与有效性之间提供了一个很好的平衡。它遵循一种简单的数据划分方式，根据这个方法，它将对象的数据库分割为了一个 k 个簇的集合，使得数据点与它们对应的簇中间（质心）之间的平方欧氏距离的总和被最小化——这个距离也被称为**量化误差**(quantization error)。

在 k-means 中，每个簇由一个点所定义，也就是簇的质心（centroid），并且每个数据样本都会被分配到最接近的质心。质心是在该簇内的数据样本的平均值。被用于 k-means 的簇内有效性指标是到质心的平均距离。最初，数据样本被随机地分配到一个簇，然后算法通过在将数据重新分配到各个簇与更新所导致的质心之间交替来进行。算法的基本步骤如下：

给定 k

1. 随机地将数据点分成 k 个非空的簇。
2. 计算当前分割的各个簇的质心的位置。质心是簇的中心（平均点）。
3. 将每个数据点分配到有着最接近的质心的簇。

4. 当分配没有产生改变时候停止。否则回到步骤 2。

尽管 k-means 由于它的简单性非常流行，它仍然有着许多客观的缺点。第一，它只适用于在一个连续空间中的数据对象。第二，需要提前指定簇的数字 *k*。第三，它不适合用于探索有着非凸形状的簇，因为它只能找到超球形簇。最后，k-means 对于异常值非常敏感，因为有着极大（或者极小）的值可能会持续地扭曲数据的分布，并且影响算法的表现。正如我们将在下面看到的，层次聚类设法克服了上述缺点的一部分，对数据聚类提出了一种有效的替代方法。

2.7.1.2 层次聚类

尝试构建一个多个簇的分层的聚类方法属于**层次聚类**方法。通常来说，有两种可行的策略：*聚集*(agglomerative)与*分裂*(divisive)。前者通过逐渐地将数据点合并在一起，自底向上地构建层次结构，而后者通过在一种自顶向下的方式中逐渐地分割数据点来构建层次。两个聚类策略都是贪婪的。层次聚类使用一个距离矩阵作为聚类策略（无论是聚集的还是分裂的）。这个方法并不需要簇的数目 *k* 作为一个输入，但是需要一个终止条件。

在陈述上，我们将聚集聚类算法的基本步骤表示如下：

给定 k

1. 为每个数据样本创造一个簇。
2. 寻找两个最接近的数据样本——也就是，寻找两个点（单链）之间最短的欧式距离——并且其不在同一个簇中。
3. 合并包含有这两个样本的簇。
4. 如果已有个簇 *k* 则停止；否则回到步骤 2。

而在分裂层次聚类，所以的数据一开始都在同一个簇中，其将被分割，直到每个数据点都处于它自己的簇中，这是通过一个分割策略的——例如 DIvisive ANAlysis 聚类(DIANA)[328]——或者是通过另一个聚类算法来将数据分割为两个簇——例如，2-means。

一旦数据的簇被迭代地合并（或是分割），就可以通过将数据分解为嵌套的分块的不同层级来对簇进行可视化。换句话说，可以观察一个簇的树形表示，其也被称为树形图(dendrogram)。数据的聚类通过在平方欧氏距离的期望层级上进行切割树形图来获得。对于有兴趣的读者，一个树形图的案例在第五章呈现。

层次聚类将簇表示为包含在它们中的数据样本的集合，因此，一个数据样本就像它最接近的样本一样属于相同的簇。而在 k-means 中，每一个簇都通过一个质心来表示，因此数据样本属于由最接近的质心所表示的簇。除此之外，对于簇的有效性指标来说，聚集的聚类使用在一个簇的任意样本与另一个簇中的某个样本之间的最短距离，与此同时 k-means 使用到质心的平均距离。由于这些不同的算法属性，层次聚类有能力对任何来自一个连续形状的任何形式的数据进行聚类；而在另一方面，k-means 仅限于超球形簇。

2.7.1.3 对 Ms Pac-Man 的聚类

一种可能的用于控制 Ms Pac-Man 的方法是对鬼魂行为进行建模并且将这个信息作为 Ms Pac-Man 的控制器的一个输入。无论是 k-means 还是层次聚类，算法都可以考虑鬼魂的不同属性——例如关卡探索，行为分歧，鬼魂间的距离等——并且将鬼魂聚类到行为特征或

资料之中。然后 Ms Pac-Man 的控制器会将在一个特定关卡中的鬼魂资料考虑为一个额外的输入，用来更好地引导智能体。

可以说，除了智能体控制之外，我们可以考虑聚类对于这个游戏的更好用途，例如分析 Ms Pac-Man 的玩家并且为它们生成适合的关卡或挑战以使游戏平衡。然而，正如我们前面提到的，这个 Ms Pac-Man 例子的焦点是在于对玩游戏的智能体进行的控制，这是为了使用一个相同的案例来贯穿全章。

2.7.2 频繁模式挖掘

频繁模式挖掘是一组尝试得到数据中的频繁模式与结构的技术。模式包括了序列与项目集(itemsets)。频繁模式挖掘首先被提出用于挖掘关联规则[6]，其目的在于识别许多彼此频繁关联的数据属性，从在它们间形成条件规则。有两种类型的频繁模式挖掘对于游戏 AI 来说特别有兴趣：**频繁项目集挖掘**(frequent itemset mining)与**频繁序列挖掘**(frequent pattern mining)。前者的目标是在没有特定内部序列的数据属性之间的寻找结构，而后者旨在基于一个固有的时间顺序来在数据属性之间找到结构。尽管与无监督学习的案例有关，频繁模式挖掘在目标以及它遵循的算法过程上都有所不同。

流行的可拓展的频繁模式挖掘方法包括了用于项目集挖掘的 Apriori 算法[6]，以及用于序列挖掘的 SPADE [794]与 GSP [652, 433, 620]。在本章节的剩余部分我们将 Apriori 与 GSP 分别作为频繁项目集挖掘与频繁序列挖掘的代表性算法进行概述。

2.7.2.1 Apriori 算法

Apriori[7]是一个用于频繁项目集挖掘的算法。这个算法适合用于挖掘包含有案例（也称为事务）的集合的数据集，每个案例都或是事务都具有一组项目，或者一个**项目集**。事务的案例包括了一个亚马逊用户购买的书籍或者一个智能手机用户购买的 APP。这个算法非常简单，其描述如下：给定一个预先决定的阈值，称为**支持度**(support)（T），Apriori 检测在数据集中至少也有着 T 个事务的子集的项目集。换句话说，Apriori 会尝试识别所有至少拥有一个最小支持度的项目集，其是一个项目集存在于数据集中的最小次数。

为了展示在一个游戏案例中的 Apriori，在下面我们陈述性地列出了来自于一个在线角色扮演游戏的四个玩家的事件：

- <完成超过 10 个关卡；解锁大部分成就；购买了法师之盾>
- <完成超过 10 个关卡；购买了法师之盾>
- <解锁大部分成就；购买了法师之盾；找到了巫师的紫帽>
- <解锁大部分成就；找到了巫师的紫帽，完成超过 10 个关卡；购买了法师之盾>

如果在上述的案例数据集中我们假设支持度为 3，如下的 1-项目集（只有一个项目的集合）可以被找到：<完成超过 10 个关卡>，<解锁大部分成就>，<购买了法师之盾>，<找到了巫师的紫帽>。而如果我们带着一个为 3 的支持度阈值寻找 2-项目集，我们可以找到<完成超过 10 个关卡；购买了法师之盾>，因为上述的事务中的三个同时包含了这两个条目。对于支持计数为 3 的情况来说，更长的项目集是不可用的（不频繁）。这个过程可以被重复用于任何我们希望为之检测频繁项目集的支持度阈值。

2.7.2.2 广义序贯模式

如果事件的序贯是我们希望从一个数据集中挖掘到的关键信息，那么频繁项目集挖掘算

法是不足的。数据集可能包含了一个有序的序列集合，例如时序数据或者时间学列。然而，我们需要选择一种频繁序列挖掘方法。序列挖掘问题可以被简单地描述为，给定一个序列或者一组序列，寻找频繁发生的子序列的过程。

更正式地说，给定一个数据集，其中每个样本都是一个事件的的序列，也叫一个**数据序列**(data sequence)，序贯模式定义为一个事件的子序列，如果它会周期性地在数据集的样本中发生，就是一个**频繁序列**(frequent sequence)。一个频繁序列可以被定义为一个至少被一个最小的数据序列总量所支持的序贯模式。这个总量由一个称为最小支持度值的阈值所决定。一个数据序列，当且仅当它在一个相同的顺序中包含了所有出现在了模式中的事件时，才支持一个序贯模式。例如，数据序列<$x_0, x_1, x_2, x_3, x_4, x_5$>支持了模式<$x_0, x_5$>。就像频繁项目集挖掘那样，支持了一个序列模式的数据序列的总量被称为**支持计数**(support count)。

广义序贯模式(Generalized Sequential Patterns, GSP)[652]是一个流行的用于挖掘数据中的频繁序列的方法。GSP 首先使用一个单独的事件，也称为 1-序列来提取频繁序列。这组序列时自加入来生成所有的 2-序列备选，对于其我们计算了它们的支持计数。这些频繁的序列（也就是，它们的支持计数大于一个阈值）然后被自加入来生成 3-序列备选的集合。这个算法在每个算法步骤逐渐地增加序列的长度，直到下一组备选为空。算法的基本原则是，如果一个序列模式是频繁的，则其*连续*的子序列也是频繁的。

2.7.2.3 用于 Ms Pac-Man 的频繁模式挖掘

各个序列的事件的模式可以被抽取来协助 Ms Pac-Man 的控制。在给定一个特定支持计数时，项目集可以通过 Ms Pac-Man 的专业玩家的成功事件来确定。例如，一个 Apriori 算法运行在几个不同的专业玩家的事件之间，可能会发现一个如下的频繁的 2-项目集：<玩家首先进入左上角，玩家首先吃了右下角的力量药片>。这样的信息对于设计用于控制 Ms Pac-Man 的规则是明显有用的。

除了项目集，对 Ms Pac-Man 游戏来说，鬼魂事件的频繁度也可以被考虑。例如，通过运行 GSP 来抽取鬼魂的属性，可能会发现当 Ms Pac-Man 吃下了一个力量药片，Blinky 鬼魂很可能向走移动（<力量药片，Blinky 左转>）。这样的频繁序列可以形成任何 Ms Pac-Man 的额外输入——例如，一个人工神经网络。第五章详细描述了在一个 3D 捕食游戏中这种频繁序列挖掘的一个案例。

2.7.3 进阶阅读

频繁模式挖掘的一个通用导论提供于[6]。Apriori 算法在 Agrawal 与 Srikant 的原始文献[7]中被详细叙述，而 GSP 被彻底涵盖于[652]。

2.8 知名的混合算法

AI 方法可以通过许多方式交织在一起，产生新的复杂算法，增强它们组合部分的能力，经常伴随着一种完型效果的发生。例如，你可以让遗传算法进化你的行为树或者有限状态机；你也可以通过用于树剪枝的人工神经网络评估器来增强 MCTS ；或者你还可以在之前提到的每个搜索算法中添加一个局部搜索的组件。我们将 AI 方法的结合的结果命名为**混合**算法，并且在这个章节中我们涵盖了两种在我们的观点中最具有影响力的混合游戏 AI 算法：神经进化(neuroevolution)与带有人工神经网络函数逼近器的差分学习。

2.8.1 神经进化

人工神经网络的进化，也可以说神经进化，指人工神经网络的设计——它们的连接权重，它们的拓扑，或二者都是——使用进化算法[787]。神经进化已经被成功用于人工生命，机器人控制，生成系统与计算机游戏领域。这个算法的广泛应用型主要是出于两个关键原因：首先，许多 AI 问题可以被视为函数优化问题，其基本的通用函数可以通过一个人工神经网络来逼近。其次，神经进化是一种基于生物学上的隐喻与进化论的方法，受到了大脑进化方式的启发[566]。

这个进化（强化）学习方法适用于当可用的误差函数不可微分时，或是目标输出不可用时。例如，前者可能发生于人工神经网络中使用的计划函数不连续并因此不可导时（这是一个突出的现象，例如在组合模式生成网络中(compositional pattern producing networks)[653]）。后者可能发生在一个我们对其没有好（或者坏）的行为的样本，或是不能定义什么会是一个好行为的领域。神经进化通过元启发式(metaheuristic)（进化）搜索来设计人工神经网络，而不是基于梯度搜索来反向传播误差并且调整人工神经网络。与监督学习相反，神经进化不需要一个数据-输出组合的数据集来训练人工神经网络。相反，它只需要一种对人工神经网络在被研究问题上的表现的度量，例如，被人工神经网络控制的一个游戏智能体的分数。

神经进化的核心算法步骤如下：

1. 一个代表了人工神经网络的染色体种群被进化来优化一个适应度函数，其表征了人工神经网络的效用（质量）。染色体（人工神经网络）种群通常是随机初始化的。
2. 每个染色体被编码为一个人工神经网络，然后其将在优化下在任务上被测试。
3. 测试过程为种群的每个人工神经网络赋予了一个适应度值。
4. 一旦当前种群中的所有基因类型的适应度值被确定，一个选择策略（例如轮盘赌，锦标赛）将被用于为下一代选择父代。
5. 通过在选择的人工神经网络编码的染色体上应用基因操作，一个新的后代的种群将被产生。变异与/或突变就像任何进化算法中的相同方式那样被应用。
6. 一个替换策略（例如稳态方法，精英方法，多世代方法）将被用来决定新种群的最终成员。
7. 与一个典型的进化算法类似，生成循环（步骤 2 到 6）将被重复直到我们耗尽了我们的计算预算或者我们对当前种群所获得的适应度感到满意。

通常由两种类型的神经进化方法：那些只考虑了一个网络的连接权重的进化的，以及那些同时考虑了网络的连接权重与拓扑的（包括连接类型与激活函数）。在前一种神经进化的类型，权重向量被编码并且基因化地表示为一个染色体；在后一种类型，基因表示包括了一个人工神经网络拓扑的编码。除了简单的多层感知机，被考虑用于进化的人工神经网络类型包括了增强拓扑的神经进化算法(NeuroEvolution of Augmenting Topologies, NEAT)[655]以及组合模式生成网络[653]。

神经进化已经在游戏领域得到了广泛应用，例如评估一个游戏的状态-动作空间，选择一个适合的动作，在可能的策略中抉择，为对手的策略建模，生成内容，以及为玩家体验建模[566]。这个算法的有效性，可拓展性，广泛的应用性以及开放式的学习是几个让神经进化对许多游戏 AI 任务成为一个良好的通用的方法的原因[566]。

2.8.1.1 用于 Ms Pac-Man 的神经进化

一种简单的在 Ms Pac-Man 中实现神经进化的方式是先为 Ms Pac-Man 设计一个将游戏状态作为输入然后输出动作的人工神经网络。这个人工神经网络的权重可以使用一个典型的进化算法来进化，并且遵循上面描述的神经进化的步骤。种群中的每个人工神经网络的适应度通过为种群中的每个人工神经网络配备 Ms Pac-Man 并让它玩一会游戏来获得的。在模拟时间中智能体的表现（例如分数）可以决定人工神经网络的适应度值。图 2.18 展示了人工神经网络编码的步骤以及这个 Ms Pac-Man 中的神经进化的假设实现里的适应度赋值。

2.8.2 带有人工神经网络函数逼近器的差分学习

强化学习通常使用表格化的表示来存储知识。正如之前在强化学习章节提到的，以这种方式表示知识可能会耗尽我们可用的计算资源，因为查找表的规模相对于动作-状态空间以指数方式增长。应对这一挑战最普遍的方式就是使用一个人工神经网络作为一个值（或者是 Q 值）逼近器，从而替换表格。这一做使得将算法应用到动作-状态表示的巨大空间上成为可能。在此之外举例来说，一个人工神经网络作为一个 Q 的函数逼近器，还可以处理有着无限大的连续状态空间的问题。

在这个章节，我们概述了在差分学习上利用人工神经网络的通用逼近能力的两个里程碑算法案例。TD-Gammon 算法与深度 Q 网络(deep Q network)已经被分别应用于精通西洋双陆棋游戏与以超越人类的水平玩雅利达 2600 游戏。两个算法都适用于这些特定游戏之外的任何强化学习任务，但是让它们知名的游戏将在下面被用来描述这些算法。

2.8.2.1 TD-Gammon 算法

可以说 AI 在游戏上最成功的故事之一就是 Tesauro 那可以以游戏大师的水平玩西洋双陆棋的 TD-Gammon 软件了[689]。这个学习算法是一个多层感知机与一个称为 TD(λ)的差分变体的一种混合组合；TD(λ)算法更为详细的阐述，见[672]的第七章。

TD-Gammon 使用一个标准的多层神经网络来逼近值函数。多层感知机的输入是一个棋盘的当前状态的表示（Tesauro 使用 192 个输入），而多层感知机的输出是在给定当前状态后的预测胜利概率。奖赏对于所有棋盘状态都被定义为零，除了那些游戏获得胜利的。然后多层感知机通过与自己对弈来迭代地训练，并且基于估计的对胜利的概率来选择动作。每场游戏被视为一个包含了一个位置的序列的训练片段，其被用于通过反向传播它的输出的差分误差来训练多层感知机的权重。

TD-Gammon 0.0 自我对弈了大约 300,000 局游戏，然后设法与当时最佳的西洋双陆棋计算机玩的一样好。尽管 TD-Gammon 0.0 并没有赢得表演赛，但是它给我们带来一种迹象，即使没有任何专家知识被集成在 AI 算法中，强化学习也可以实现什么。这个算法的下一个迭代（TD-Gammon 1.0）很自然地通过特定的西洋双陆棋特征结合了专家知识，改变了多层感知机的输入并且实质是达到了更高的性能。从那时起，隐藏层的数量以及自我对弈的游戏次数极大地决定了算法的版本以及它产生的能力。从 TD-Gammon 2.0（40 个隐藏的神经元）到 TD-Gammon 2.1（80 个隐藏的神经元），TD-Gammon 的表现逐渐地增加，而在 TD-Gammon 3.0（160 个隐藏神经元），它接触到了最强的人类玩家在西洋双陆棋上的游戏水平[689]。

2.8.2.2 深度 Q 网络(Deep Q Networks)

尽管强化学习与人工神经网络造就了非常强大的混合算法，但算法的表现传统地依赖于人工神经网络的输入空间的设计。正如我们前面看到的，即使最强大的强化学习应用，例如

TD-Gammon 智能体，也要通过在输入空间中集成游戏的特定特征，从而添加关于游戏的专家知识来达到人类级别的游戏表现。直到最近，强化学习与人工神经网络的组合才能不需要考虑特定设计的特征，而几乎只需要通过学习发现它们来在一个游戏中达到人类的水平。来自谷歌 DeepMind 的一个团队[463]开发了一个称为*深度 Q 网络*（DQN）的强化学习智能体，其通过 Q-learning 训练了一个深度卷积人工神经网络。DQN 在它被训练的 Arcade Learning Environment[463]中的 46 个街机（雅利达 2600）游戏上的 29 个中达到或者超越了人类级别的游戏水平。

DQN 收到 TD-Gammon 的启发，因为它通过梯度下降将一个人工神经网络作为差分学习的函数逼近器。就像在 TD-Gammon 中那样，梯度通过反向传播差分误差来计算。然而，DQN 使用 Q-learning ，而不是使用 TD(λ)来作为基本的强化学习算法。此外，人工神经网络不是一个简单的多层感知机，而是一个深度卷积神经网络。DQN 在 ALE 的每一个游戏上都进行了大量的帧数（5 千万帧）。对每个游戏来说，玩的时间的总和大约是 38 天[463]。

DQN 同时分析一个带有四个游戏屏幕的序列，并且根据当前状态来逼近每一个可能的动作的未来游戏得分。特别是，DQN 使用了来自四个最接近的游戏屏幕的像素作为它的输入，结果是 84×84（像素的屏幕大小）×4 的人工神经网络输入大小。除了屏幕像素信息，没有其它游戏特定的知识被提供给 DQN。被用于卷积人工神经网络的结构有着三个隐藏层，其分别产生 32 20×20, 64 9×9 以及 64 7×7 的特征映射。DQN 的第一层（最低层）处理了游戏屏幕的像素并且抽取特定的视觉特征。卷积层之后跟着一个完全连接的隐藏层与一个输出层。每个隐藏层都跟着一个非线性修正单元。给定一个由网络的输入表示的游戏状态，DQN 的输出是对应的状态-动作组合的最优估计动作值（最优 Q-values）。通过从游戏环境中接受瞬时奖励，DQN 被训练来接近 Q-values（游戏的实际分数）。特别是，如果分数在两个连续的时间步骤（帧）之间增加，奖赏将是+1，如果分数减少，它将是-1，否则为 0。DQN 将一个 ε-greedy 策略用于动作-选择策略。值得一提的是，在撰写本文时，出现了更新并且更为有效的深度强化学习概念的实现，例如 Asynchronous Advantage Actor-Critic (A3C)算法[462]。

2.8.2.3 用于 Ms Pac-Man 的带有人工神经网络函数逼近器的差分学习

我们可以设想一种用于控制 Ms Pac-Man 的 DQN 方法，其与 ALE 智能体被训练的方式[463]相似。一个深度卷积神经网络以一个像素级别的基础来扫描关卡的图片（见图 2.19）。图片穿过多个卷积与完全连接层，其最终作为一个多层感知机的输入，并输出 Ms Pac-Man 四种可能的行动（保持方向，向后移动，向左转。向右转）。一旦某个动作被应用，游戏的分数将被用作瞬时奖赏来更新深度网络的权重（卷积人工神经网络与多层感知机）。通过玩足够的时间，控制器会收集经验（截图，动作，以及对应的奖赏），其可以用于训练深度人工神经网络来逼近一种能最大化 Ms Pac-Man 的分数的策略。

2.8.3 进阶阅读

对于一份近期的关于游戏中神经进化的应用的深入综述，读者可以参考[566]。有关神经进化的完整评价，请参阅 Floreano 等人的[203]。CPPN 与 NEAT 被分别被详细地涵盖于[653]与[655]中。TD-Gammon 与 DQN 分别被详细地涵盖于[689]与[463]中。它们在即将到来的[672]的第二版中都被置于更大的强化领域中。关于 A3C 算法的细节可以在[462]中找到，并且算法的实现作为 Tensorflow 的一份可以被直接地找到。

2.9 总结

本章涵盖了我们认为本书的读者需要熟悉的 AI 方法。然而，我们期望读者在阅读本书之前就具有一个 AI 的基本背景或者已经完成了一门 AI 的基础课程。因此，这些算法并没有被详细地介绍，因为本书的重点是 AI 在游戏领域上的英语，而不是 AI 本身。在此基础上，我们使用游戏 Ms Pac-Man 作为本章所有算法的整体应用测试平台。

我们探讨的算法家族包括了传统的特定行为编辑方法（例如有限状态机与行为树），树搜索（例如最佳优先搜索，极大极小与蒙特卡罗树搜索），进化计算（例如局部搜索与进化算法），监督学习（例如神经网络，支持向量机以及决策树），强化学习（例如 Q-learning），无监督学习（例如聚类与频繁模式挖掘），以及如进化人工神经网络与将人工神经网络作为期望奖赏的逼近器之类的混合算法。

在这一章我们达到了本书的第一个，也是导论部分。下一个部分将从一个关于游戏中的 AI 最传统以及最广泛探索的任务的章节开始：玩游戏！