SMAClite：一个用于多智能体强化学习的轻量级环境

Adam Michalski 爱丁堡大学 英国爱丁堡 contact@adammi.ch

Filippos Christianos 爱丁堡大学 英国爱丁堡 f.christianos@ed.ac.uk

Stefano V. Albrecht 爱丁堡大学 英国爱丁堡 s.albrecht@ed.ac.uk

arXiv:2305.05566v1 [cs.LG] 9 May 2023

摘要

目前缺乏多智能体强化学习（MARL）算法的标准基准测试。《星际争霸》多智能体挑战（SMAC）已广泛应用于多智能体强化学习研究，但它建立在一款庞大且闭源的电脑游戏《星际争霸II》之上。因此，SMAC 的计算开销很大；若要对环境进行任何有意义的修改或贡献，就必须了解并使用该游戏专有的工具。我们推出了 SMAClite——一个基于 SMAC 的挑战环境，它既与《星际争霸II》脱钩，又是开源的；同时还提供了一个框架，使用户无需任何专业知识即可为 SMAClite 创建新内容。我们通过实验表明，SMAClite 与 SMAC 等效：在 SMAClite 上训练多智能体强化学习算法，并复现 SMAC 的结果。随后我们证明，SMAClite 在运行速度和内存占用方面均优于 SMAC。

关键词

多智能体强化学习，《星际争霸》，策略，游戏

1 引言

随着人工智能研究不断取得进展，它不可避免地融入了我们的日常生活。近年来广受欢迎的自主智能体包括机器人吸尘器（例如 [30]）和自动驾驶汽车（例如 [2]），也包括在幕后为生活带来便利的机器人，例如仓库优化机器人（例如 [15]）。随着研究领域对这些技术的关注不断增加，人们自然需要为各种类型的人工智能（AI）模型建立标准化的基准测试。

多智能体强化学习（MARL）是机器学习的一个分支，研究多个自主人工智能实体在同一环境中共存的情形——它们通常被称为智能体 。本文关注协作型智能体，即共同努力实现某个目标的智能体。至关重要的是，目前研究社群尚未就此类智能体应采用何种标准基准测试达成共识。仅看一篇近期的基准测试论文 [20]，我们就能数出五种不同的基准测试环境。

《星际争霸》多智能体挑战赛（SMAC） [24] 已被广泛用于多智能体强化学习研究。它构建于即时战略电脑游戏《星际争霸II》（SC2）之上，并使用了 Vinyals 等人提供的应用程序接口（API） [32]——这是游戏与人工智能智能体之间的接口。该挑战赛提供了一个小游戏：每个智能体在若干可用战斗场景之一中控制一个战斗单位（例如一名士兵），对抗由游戏内置人工智能控制的敌方队伍。SMAC 提出了一个并非易于解决的挑战——在大多数场景中，最直观的策略，即向前推进并发动攻击，并不足以取胜，而且由于敌军拥有更强的单位或更多的兵力，最终会迅速落败。

尽管这个想法很有前景，但如果要让 SMAC 成为普遍认可的基准测试，我们仍能发现其中存在几个问题。我们认为，最大的问题在于 SMAC 将 SC2 作为关键依赖项，因此无论是训练还是推理，都需要下载一个体积庞大的文件（约 3.7 GB），并进行复杂的设置；更不用说始终要同时运行 SC2 和 SMAC，这会额外消耗 CPU 和内存资源。更糟的是，SMAC 只使用了 SC2 功能的一个子集——许多必需的下载内容和计算资源其实都是多余的，因此，在 SMAC 上训练智能体的成本高于完成其所提供任务的实际需求。

除此之外，对于不熟悉其所基于的游戏的人来说，SMAC 仍然很难上手。这体现在许多方面，例如，要为 SMAC 创建自定义场景或单位，就必须使用官方的星际争霸 II 地图编辑器；这要求使用者学习一种陌生且专有的工具，并为单个基准测试投入大量精力。

我们提出了一个与 SMAC 非常相似的挑战，但完全摆脱了对《星际争霸II》的依赖，并表明它保留了 SMAC 具有挑战性的各个方面。我们将这一挑战命名为星际争霸轻量级多智能体挑战（SMAClite）。我们还希望让战斗场景和单位尽可能易于修改，同时也能轻松创建全新的战斗场景和单位。我们的环境保留了 SMAC 的外部接口，仅改变内部实现，从而让人工智能开发者能够以最少的修改复用用于处理 SMAC 的代码。我们将这一环境开源[1]，并免费提供使用。

我们使用在 SMAClite 上训练的模型开展了一系列实验。实验包括定量分析，即比较各种多智能体强化学习算法在 SMAClite 中取得的回报。结果表明，这些算法在 SMAClite 上取得的回报与其在 SMAC 上取得的回报相近，而且它们在 SMAC 上的相对排名也得以保留。我们还进行了定性分析，逐一考察智能体在不同案例中采用的战斗策略，并验证它们确实能够在策略上胜过手工编写的敌方人工智能。此外，我们将使用 SMAClite 训练的模型直接部署到原始 SMAC 环境中，不进行任何额外训练，以考察这两个环境之间的迁移学习潜力。根据这一实验，我们得出结论：在 SMAClite 上训练智能体能够提高它们在 SMAC 上取得的回报，这也表明这两个环境所要求的技能具有相似性。

[1] https://github.com/uoe-agents/smaclite

2 相关工作

对于强化学习 [5, 22] 的单智能体变体，确实存在一些广受研究社群认可、被视为标准的热门基准测试。多智能体强化学习基准论文 [20] 的作者提供了两种多智能体强化学习基准测试，它们都基于简单的二维世界：基于等级的觅食和多机器人仓库。此外，还有其他几种非星际争霸II的多智能体基准测试，例如多智能体粒子环境 [19]，或 Hanabi挑战 [3]。另一个基于现代电脑游戏的多智能体挑战是 OpenAI Five 项目 [7],，它让智能体参与策略游戏 Dota 2 的完整五对五比赛，展示了游戏及训练模型的惊人规模。

随着其应用程序接口的发布 [32],，在《星际争霸》II中训练自主智能体逐渐流行起来，其中一项广受关注的成果是 AlphaStar 模型 [31]。我们的工作主要基于 SMAC 的创作者所做的工作 [24]。自发布以来，SMAC 已在众多研究中被用作自主智能体的基准测试（例如 [20, 23, 33]），其受欢迎程度表明它确实填补了一个空缺。

有几个项目与我们的工作有一定关联，因为它们也旨在改进 SMAC，但采用了不同的方法（例如 SMACv2 [9], SMAC + [14]）。这些项目为 SMAC 范式带来了有趣的补充；然而，据我们所知，它们都没有解决我们希望处理的问题：环境的性能开销，以及其封闭源代码的特性。我们认为，这些项目引入的补充是 SMAClite 未来发展的好思路，但我们的项目旨在将挑战难度维持在与原版相同的水平。

3 背景

3.1 多智能体强化学习

多智能体强化学习（MARL）允许多个自主智能体共存于同一空间。形式上，MARL 的设置由若干智能体 组成，它们能够执行特定动作；此外还包括环境 ——这个术语涵盖了智能体之外的一切，智能体可以与之交互。我们在研究中采用的 MARL 具体形式称为**去中心化部分可观测马尔可夫决策过程**（简称：Dec-POMDP）。

去中心化部分可观测马尔可夫决策过程是一种合作过程，定义为一个由 7 个元素组成的元组（N，S，A，O Ω，P，R），其中 $\mathrm{N} = \{1 \dots N\}$ 是参与该过程的智能体集合。智能体以离散时间步 $t \in \mathrm{N}$ 与环境交互。在每个时间步 t，环境都有一个真实的活动状态 $s_t \in \mathrm{S}$，每个智能体 i都会收到一个观测 $o_t^i \sim \Omega(i,\ s_t)\ \ o_t^i \in \mathrm{O}$。随后，每个智能体 i选择一个动作 $a_t^i \in \mathrm{A}$。每个时间步 t结束后，智能体会收到共享奖励 $\mathrm{R}(a_t^1,\ a_t^2、\ldots\ldots、\ a_t^N,\ s_t) = r_{t+1} \in \mathrm{R}$，环境则进入下一个状态 $s_{t+1} \sim \mathrm{P}(a_t^1,\ a_t^2、\ldots\ldots、\ a_t^N,\ s_t)\ s_{t+1} \in \mathrm{S}$。

在去中心化部分可观测马尔可夫决策过程中，与一般的强化学习一样，智能体在每个时刻的目标 t 是最大化折扣累积奖励（或回报） $\sum_{i=0}^{\infty} \gamma^i r_{t+i}$，其中 γ是折扣因子。当 $\gamma = 1$时，折扣回报等于实际回报 $\sum_t r_t$ ——我们将在报告评估结果时给出这一总和。

为了在 SMAClite 环境中训练和评估智能体，我们将采用近期一篇多智能体强化学习基准论文 [20]中用于测试 SMAC 智能体的同一组算法及其超参数。这包括 9 种可用于解决各种 SMAC 场景的热门算法：IQL [28], IA2C[18], IPPO [25], MADDPG [17], COMA [10], MAA2C [20],MAPPO [33], VDN [27], 和 QMIX [23]。

3.2 SMAC

3.2.1 概述。 本节将介绍与我们的项目相关的 SMAC 环境。我们的项目仅复现 SMAC 的默认配置，不包含任何可选参数，因此这里也不作介绍。

在 SMAC 中，每个智能体控制一个单位（本文将智能体控制的单位称为盟军单位 ），并负责击败由《星际争霸II》内置人工智能对手控制的一组敌方单位。SMAC 定义了多个战斗场景，这些场景在军队编成和地形布局上各不相同，因此难度也有所不同。图 1 展示了两个 SMAC 场景的可视化结果。

单位分为若干类型，具有不同的属性（生命值、攻击力等）——但请注意，只有当某个场景中单支队伍包含多于一种单位类型时，SMAC 环境才会在状态／观测向量中区分单位类型——我们将此称为场景“区分单位类型”。

某些单位类型一旦受到攻击，生命值就不会以任何方式恢复。其他单位天生具有持续生效的生命值恢复能力。还有一类单位类型除了生命值点数外还拥有特殊护盾；在一段时间内未受到伤害后，护盾会恢复，并且必须先将护盾降至 0，才能对单位的生命值造成伤害。由于《星际争霸II》的限制，同一队伍中的单位要么全部拥有护盾，要么都没有护盾。

3.2.2 动作。 每个智能体都可以执行多种动作，这些动作在任意给定时间步可能可用，也可能不可用——环境提供了一种方法，可获取每个智能体当前可用的动作。如果智能体选择了不可用的动作，SMAC 就会报错并停止执行。可执行的动作包括：no-op（空操作）——不会产生任何效果，且仅对已死亡的单位可用；stop（停止）——命令单位停在原地，不执行任何操作；moveN、moveE、moveS 和 moveW——命令单位沿选定的基本方位（北、东、南或西）移动；target1、target2、.....——命令单位以指定的队伍专属 ID 为目标。对于造成伤害的单位，这表示以敌方单位为目标进行攻击；对于治疗单位，则表示以盟友为目标进行治疗。SMAC 为智能体设定了固定的攻击范围；如果目标超出此范围，该动作就不可用。

3.2.3 状态。 环境的真实状态是一个分为三个部分的向量。第一部分按 ID 顺序包含每个友方单位的信息：其当前生命值、当前冷却时间、其 X 和 Y 坐标（相对于地图中心）、其当前护盾（仅当盟军拥有护盾时），以及一个表示其单位类型的独热向量（仅当场景区分单位类型时）。第二部分按 ID 顺序包含每个敌方单位的信息：其当前生命值、其 X 和 Y 坐标（相对于地图中心）、

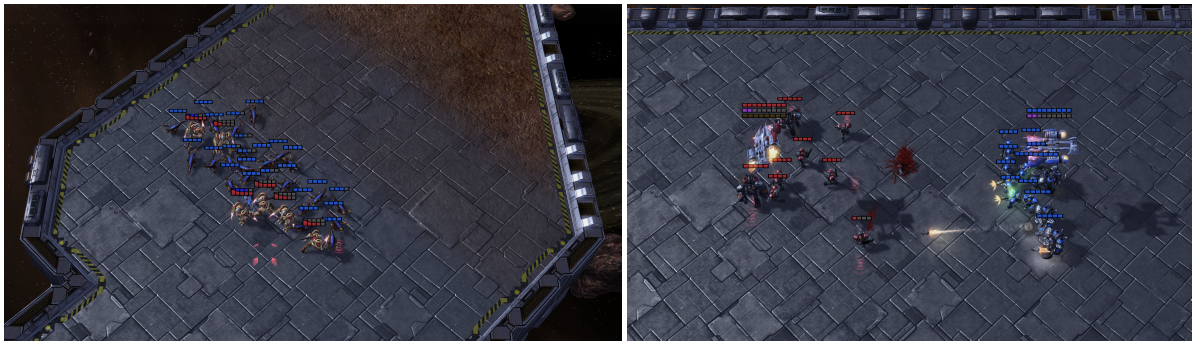


图 1：V　SMAC 环境的可视化。*(*左侧*)* 走廊场景。*(*右侧*)* MMM2 场景。　o.

其当前护盾（仅当敌军拥有护盾时），以及一个表示其单位类型的独热向量（仅当场景区分单位类型时）。最后一部分按 ID 顺序包含每个智能体的信息：一个表示其在前一个时间步所采取动作的独热向量。请注意，状态向量中的所有特征都已归一化到 0 和 1 之间——例如，生命值除以给定单位的最大生命值。

3.2.4 观测。每个智能体在每个时间步都会接收一条观测，表示该智能体在环境中可见的内容。SMAC 为智能体设定了固定的视野范围——如果某单位已死亡，或距离智能体自身单位超过该视野范围，则观测向量中关于该单位的所有信息都会被完全置零。观测向量分为四个部分。第一部分包含每个基本方位上的移动是否可行。第二部分针对每个敌方单位，在其存活时包含以下信息：智能体的单位是否可以攻击它、它与智能体单位之间的距离、相对于智能体单位的X和Y坐标、它的生命值、它的护盾（仅当敌方单位拥有护盾时），以及表示其单位类型的独热向量（仅当场景区分单位类型时）。第三部分针对每个友方单位，包含一个字面值1（用于与已死亡或距离过远的单位区分）、它与智能体单位之间的距离、相对于智能体单位的X和Y坐标、它的生命值、它的护盾（仅当友方单位拥有护盾时），以及表示其单位类型的独热向量（仅当场景区分单位类型时）。最后一部分针对智能体自身的单位，包含其生命值、其护盾（仅当友方单位拥有护盾时），以及表示其单位类型的独热向量（仅当场景区分单位类型时）。请注意，与状态特征类似，所有观测特征都归一化到0到1之间。

3.2.5 奖励。每个时间步结束后，所有智能体都会获得共享奖励，其数值等于该时间步内从敌军身上移除的生命值点数和护盾点数之和。每消灭一个敌方单位，额外增加 10 点小额奖励；赢得场景后，额外增加 200 点更大的奖励。奖励通过除以敌方单位的所有生命值点数和护盾点数之和及所有可能奖励的总和，再乘以 20 进行归一化——因此，可能的回报应介于 0 和 20 之间。不过请注意，由于生命值和护盾会再生，智能体在一个回合结束时实际获得的累积奖励可能超过 20。

3.3 最优互惠碰撞规避

在星际争霸II中，单位会在有其他单位存在的战场上移动；如果它们可能挡住另一个单位的去路，就会向侧面移动以避免碰撞。这使战场感觉更加真实且富有物理感，并支持一些高级策略，例如卡位或包围（本质上是通过战略性地调整自身位置来限制其他单位的移动）。我们认为，在我们的环境中复现这一行为至关重要。然而，由于星际争霸II是一款专有的闭源软件，我们无法使用 SMAC 中采用的确切算法。为填补这一空白，我们选择了 Berg 等人提出的最优互惠碰撞规避（ORCA）算法 [6]。

ORCA 算法满足了我们所需环境的多项标准。与 SC2 类似，每个单位都被视为具有特定半径的圆形区域。单位可以避开其他单位，也可以避开静态多边形障碍物。此外，单位不仅能够避免碰撞，还能同时朝自己的目标位置移动。

该算法由时间范围 τ参数化。在该算法的每次运行中，每个单位 A 都会在二维速度空间中计算一组半平面（我们称之为 *ORCA* 半平面），每个半平面都表示一组安全速度，选择这些速度可以避免与其他单位或障碍物 B 在至少 τ 时间内发生碰撞。

每个单位 A 都会考虑其附近的所有单位和障碍物（即其周围一定半径 r范围内的对象）。对于每个邻居 B，该单位都会计算邻居对其产生的**速度障碍**，即速度空间中会导致该单位在 τ 时间内与该邻居发生碰撞的所有位置集合（对于单位：两者之间的距离小于 $r_A + r_B$，其中 r_A 和 r_B 分别是两个单位自身的半径）。

令$\mathbf{u}$ 为从相对速度$\mathbf{v}_A - \mathbf{v}_B$到速度障碍边界的最短向量——换言之，这是防止在 τ时间内发生碰撞所需对相对速度做出的最小变化。于是，定义邻居对应 ORCA 半平面的直线（称为 *ORCA* 直线）的斜率，由速度障碍边界在点$\mathbf{v}_A - \mathbf{v}_B + \mathbf{u}$处的外法线斜率给出。该直线经过点$\mathbf{v}_A + \frac{1}{2}\mathbf{u}$。这为该单位提供了一个可避免与邻居碰撞的可能速度半平面——注意，调整量为$\frac{1}{2}\mathbf{u}$，这是因为该单位假设邻居采用相同的算法，并会做出$-\frac{1}{2}\mathbf{u}$的调整（因为从邻居的

–

视角来看，一切都是镜像的——因此需要取反）——对于障碍物邻居，调整量不减半；只有对于单位邻居，调整量才减半。

然后，根据邻居所产生的所有半平面，算法会为每个单位求解一个线性规划问题，找出一种速度，使其既能避开所有邻居，又最接近该单位期望的速度。如果无法完全避免碰撞，算法则会求解另一个线性规划问题，以尽量减少单位越过 ORCA 线的距离。

我们将在第 4 节中更详细地介绍如何在 SMAClite 中使用这一算法。请注意，该算法并不等同于寻路算法——遇到墙壁时，单位通常会停在墙前，也不会探寻太远以寻找绕行路线。

4 SMAClite

4.1 环境实现

本节将重点介绍我们的贡献，以及我们实现 SMAClite 的方法。请注意，由于《星际争霸II》是闭源的，我们无法获取其任何实现细节。本节所述算法是基于我们对游戏的了解并通过反复试验设计的，同时也参考了星际争霸II Liquipedia [16] 上的一些通用信息。

我们决定使用 Python 编程语言实现环境 [29]，因为它在机器学习社区中广为人知，也是迄今为止最受欢迎的语言。环境中的大多数计算都使用 NumPy 库完成 [12]，环境的渲染则由我们编写的脚本借助 Pygame [1] 库实现。在适用的情况下，我们使用 SMAC 环境的默认参数，并省略所有可选参数。SMAClite 环境采用成熟的 OpenAI Gym 框架[8] 来创建强化学习环境。我们在图 22 中展示了环境中的两个场景。

在实现的每个步骤中，我们都确保动作、状态、观测和奖励应用程序接口与 SMAC 完全一致。对于各个单位属性，我们也遵循了这一原则，并参考了 Liquipedia [16]。这使我们能够开展迁移学习实验，例如我们在第 5.3 节中描述的实验。

尽管《星际争霸》II 的地图网格允许单位正方形中包含三角形，但为简化起见，我们选择仅允许 SMAClite 的地形由正方形网格构成。我们发现，这一简化假设并不会降低环境的难度。在内部，为了提升性能，我们会将相邻且包含障碍物的正方形合并为矩形，以减少障碍物总数。定义单位的速度或大小时，基本距离测量单位是地形网格中单个正方形的边长。这与 SC2 保持一致，而且 SMAC 和 SMAClite 中提供的所有场景所用的地图大小均为 32×32 个单位。

为进一步提升环境性能，单位使用 Scikit-learn 库提供的 K-D 树 [21] 来查找邻居（例如，生成观测向量时查找视野范围内的单位），而不是遍历整个单位列表。由于 K-D 树仅支持在圆形区域内查询，而障碍物始终是矩形，因此在查找障碍物邻居时，单位会查询 Python 包 rtree 中的 Rtree [11]。

4.2 基础框架

与 SMAC 类似，SMAClite 主要由各种可用的战斗场景以及参与这些场景的单位定义。作为 SMAClite 环境的一部分，我们提供了一个框架，可以从 JSON 文件中读取自定义场景和单位——这意味着，使用该环境创建新的或修改现有挑战时，不再需要具备《星际争霸 II 地图编辑器》的专业知识。这也意味着，在 SMAClite 中区分不同的单位类型十分容易——要比较“跳虫”和“陆战队员”，只需查看各自的 JSON 定义，了解它们之间的差异即可。

环境随附的所有标准场景和单位均使用此框架编写。本文附录详细说明了场景和单位定义格式，并提供了与该框架兼容的 JSON 文件完整示例。

4.3 单位命令类型

在任意时刻，环境中的每个单位都在执行若干种命令类型中的一种。SMAClite 支持五种不同的命令类型，其中前四种分别与智能体可用的四种动作类型完全等价，具体内容见第3.2节。

我们引入了另一种智能体无法使用、但对人工智能对手的行为至关重要的命令类型，称为攻击_移动——该命令会指示单位向指定位置行进，攻击沿途遇到的任何单位，并在抵达后守卫该位置。我们参考了《星际争霸II Liquipedia》中的“自动索敌”文章来实现攻击移动 [16],，但在相关信息不明确或缺失的地方，根据能够产生最理想行为的结果做出了一些判断。

4.4 环境循环

每个环境步开始时，智能体的单位会被分配与智能体所选动作相对应的指令。完成后，我们会模拟八个游戏步，而环境步返回的奖励是这些游戏步中获得的奖励总和。为了计算速度、冷却时间等，每个游戏步被视为持续 116 秒；这并不意味着它的实际持续时间就是如此，因为实际上，我们的环境运行速度可以远快于实时速度（详情见第 5.4 节）。

简而言之，每个游戏步都包括每个单位按照与其指令相关的逻辑行事，例如移动、攻击或等待。附录中对每个游戏步进行了详细说明。

4.5 自定义 ORCA 实现

为简化安装，我们将 Berg 等人提出的 ORCA 官方实现——以 C++ 编程语言编写的 RVO2 库 [6] ——改写为 NumPy，并将该模块与 SMAClite 一同提供。由于《星际争霸》II 使用编译后的（而且我们认为经过高度优化的）C/C

[2]我们还提供了展示 SMAClite 实际运行情况的视频：https://drive.google.com/drive/folders/1-2YJicUqRzovTa7lRgJilwtecxCPrtVb

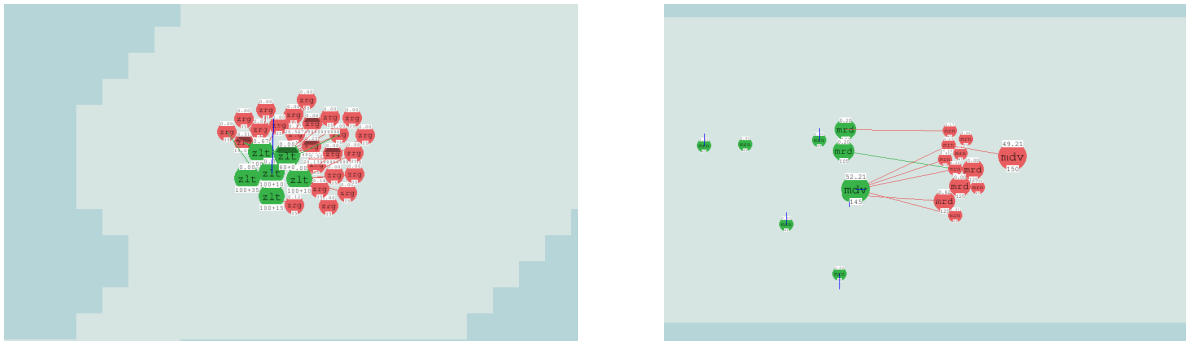


图 2：可视化　化的 SMAClite 环境。*(左)* 走廊场景。*(右)* MMM2 场景　　.

++ 代码，而 Python 代码的运行速度可能远慢于它，因此 SMAClite 开发初期便遇到了性能问题。为解决这一问题，我们还提供了 SMAClite3 的附加组件，它直接使用原版 C++ RVO2 库，并通过 Cython [4] 扩展编写的 Python 绑定来调用。Python 绑定最初由 Stüvel [26], 提供，不过我们针对自身使用场景做了若干修改；这些修改也应用于 RVO2 的 NumPy 版本。

原始 RVO2 库无法逐个移除单位，也无法一次性移除所有单位。这些是 SMAClite 的关键功能，因为每当单位死亡或我们重置环境时，都需要调整用于碰撞规避的单位列表。我们将这两项功能都添加到了 RVO2 中——我们假设单位一旦被消灭，就会立即从战场上消失，因此不希望其他单位为规避碰撞而避开它们。

第二组调整考虑了静态单位——原始 ORCA 算法假设单位完全合作，会不遗余力地让其他单位更容易通过。另一方面，我们的版本假设，如果单位 A 是静态的（即 $||\mathbf{v}_A|| = 0$），它将永远不会调整其速度。这样便可以包围其他单位和/或阻挡其路径，这在星际争霸II中是一种有效的策略。在 ORCA 的原始实现中，阻挡单位只会被简单地“推开”。请注意，为了弥补这一点，在避让静态单位时，任何移动单位都会将其速度调整 $\mathbf{u}$，而不是 $\frac{1}{2}\mathbf{u}$（$\mathbf{u}$的定义见第 3.3 节），从而确保移动单位不会走进静态单位。

由于此附加组件需要通过 CMake 构建并安装 C++ 文件，而这在某些系统上可能会造成问题，因此我们选择将此 RVO2 分支作为可选依赖项提供，供愿意经历更复杂的安装流程以提升环境性能的用户使用。我们将此版本称为 SMAClite_plus。请注意，由于查找邻居的方式不同，这两个版本中的单位行为不会完全相同，但在功能上应当难以区分。如果使用此附加组件，则会使用为 RVO2 实现的自定义 K-D 树（C++），来查找用于碰撞规避的单位邻居和障碍物邻居，而不是使用 Scikit-learn 的 K-D 树或 rtree 的 R 树。安装后，在初始化

[3]https://github.com/uoe-agents/SMAClite-Python-RVO2

环境时，可将参数 use{v4}cpp{v5}rvo2 设为 True，以启用此附加组件。若未安装该附加组件却尝试将此参数设为 True，则会导致错误。

4.6 对手人工智能行为

SMAC 的作者 [24] 声称，对手队伍由星际争霸II 内置的人工智能在极难难度下控制。由于敌方单位的行为看起来非常简单，我们在观察 SMAC 环境中的战斗时，对它们的战略能力产生了怀疑。为此，我们在 MMM2、2c_vs_64zg 和走廊这几个 SMAC 场景中进行了以下测试。

首先，我们在 SMAC 中切换不同的可用难度等级，同时保持随机数生成器的种子不变。然后，让对手人工智能与从可用动作中随机选择的智能体对战。对手单位的行为始终完全相同，而且我们发现，随机智能体在不同难度等级下获得的奖励毫无差异——至少精确到小数点后第十位时，数值都完全相等。接着，我们更进一步，**移除了**游戏中的人工智能对手，使敌方单位不受任何玩家控制。**这也没有改变单位的行为或最终获得的奖励**。

根据这些结果，我们可以相当肯定，《星际争霸》的内置人工智能从未向 SMAC 中的敌方单位下达任何指令。给予这些单位的唯一指令，是在每个 SMAC 地图文件的脚本中手动编写的——指示它们朝某个特定点攻击移动，通常是友方单位初始出现的位置。基于这些结果，我们没有实现任何自定义对手人工智能。我们只在地图初始化时，将敌方单位的指令设为朝地图场景文件中指定的攻击_移动至攻击_点。整个遭遇过程中，敌方单位的指令始终不会改变。

5 实验

在本节中，我们描述在我们的环境中进行的各种实验结果。在所有实验中，我们使用了 Papoudakis 等人所采用的全部场景 [20], ，并额外加入了 bane_vs_bane，以涵盖更多单位类型。具体而言，我们使用了 2sc_vs_1sc（2 个追猎者对 1 个脊针爬虫）、3s5z（对称地图，每方有 3 个追猎者和 5 个狂热者）、MMM2（双方各有一架医疗运输机以及一些陆战队员和劫掠者的地图）、corridor（狭窄通道中 6 个狂热者对 24 个跳虫）、3s_vs_5z（3 个追猎者对 5 个狂热者）以及

bane_vs_bane（双方都有跳虫和毒爆虫的地图）场景。作为帮助理解各个场景的可选材料，我们在附录中附上了表3，其中以自然语言描述了每个场景。

通过这些实验，我们希望展示我们的环境确实实现了我们为自己设定的目标。首先也是最重要的是，我们希望证明SMAClite 提供了与 SMAC 等效的挑战。我们希望展示，各种多智能体强化学习算法在其中的表现与在 SMAC 中同样出色，并解释其中的任何差异。我们还希望证明，运行 SMAClite 的成本严格低于 SMAC——我们关注的主要指标是运行它所需的时间，以及它占用的机器 RAM。

5.1 智能体学习曲线

我们在选定的场景上训练了若干种多智能体强化学习算法。每次训练运行 400 万个时间步，并且使用 5 个不同的随机数生成器种子分别重复每次训练。所有训练均在 SMAClite_plus 版本的环境上进行，并且完全使用 CPU，每次训练使用单个 CPU 核心。训练在云集群的不同节点上运行，其中大多数使用 Intel(R) Xeon(R) Gold 6138 CPU @ 2.00GHz，其余节点使用计算能力相当的 CPU。由于技术限制，所有训练均受到严格的 48 小时时间限制。对于所有算法，我们使用了基准论文 [20] 中列出的、在 SMAC 上表现最佳的超参数。对于情节式算法，我们使用了 5000 的缓冲区大小；对于并行算法，我们使用了 10 的缓冲区大小。我们使用 EPyMARL 框架 [20] 运行了所有训练。

关于训练过程，有几点需要说明。首先，bane_vs_bane 场景的速度要慢得多（第 5.4 节对此有所体现），因此我们仅将该场景的时间步数减少到 100 万。此外，我们发现 MADDPG 算法在训练期间极其缓慢，因此在时间限制内，它在两个场景（MMM2 和 bane_vs_bane）中都未能达到所需的时间步数，所以我们在图中省略了这两条曲线。

按训练时间绘制的每回合回报图见图3。在本小节其余部分，我们将学习曲线与基准论文中报告的 SMAC 学习曲线进行比较。

首先，在 SMAC 中，许多算法都能轻松解决 2s_vs_1sc 场景，并迅速达到 20 的最大奖励；而在 SMAClite 中，表现最好的算法（QMIX）平均回报也只有约 15。我们推测，这是因为 SMAClite 不会模拟攻击动画，而在原版《星际争霸II》中，脊针爬虫的攻击动画非常长（0.238 秒，而追猎者为 0.1193 秒），因此在 SC2 中，玩家更容易在最后一刻躲开脊针爬虫的攻击。虽然这种差异在 3s_vs_5z 场景中不那么明显，但也存在类似情况，我们认为原因相同。简而言之，我们认为，由于攻击会即时发生，在 SMAClite 中放风筝 ——即交替逃跑和攻击而不被击中——比在 SC2 中困难得多。

SMAC 与 SMAClite 的学习曲线之间，另一个主要差异出现在 MMM2环境 中：在 SMAClite 中，三种算法（MAPPO、VDN 和 QMIX）似乎都已掌握该环境，最大奖励接近 25（有关这些算法如何做到这一点的详情，请参阅下一小节）；而在 SMAC 中，表现最好的算法勉强达到 17.5 的奖励（但请注意，它们仍是相同的三种算法，这说明这些场景仍有一定的等价性）。我们推测，这可能是 SMAC环境代码 中存在非预期行为所致。在计算奖励时，SMAC 只是用敌方单位新的生命值减去其原有生命值，因此敌方的任何治疗都会使智能体受到负奖励，尽管它们并没有做错任何事。我们认为，这一点再加上造成伤害本应获得奖励，可能会让智能体在训练过程中感到困惑。根据 SMAC论文 [24] 第 4 节中的表述，我们认为这并非有意为之，因此 SMAClite 中不存在相同行为；在 SMAClite 中，我们明确将奖励定义为敌方单位因受到攻击而损失的生命值点数总和，并且不会因敌方治疗而惩罚智能体。

尽管存在这些小问题，我们确实观察到，训练曲线的整体形态与基准论文中 SMAC 的训练曲线相当可比 [20] 在所有场景中都是如此——其中一些场景的差异仅在于达到的最大回报。训练结束时算法的排名在 SMAC 和 SMAClite 之间也基本相同。我们认为，这是有力的证据，表明就学习而言，这些环境是等效的。

5.2 对已学习行为的评估

在本节中，我们为每个场景选择一个模型，并描述智能体为获得较高奖励值而采用的策略，以展示 SMAClite 中所需策略的复杂性。请注意，以下描述仅是我们对智能体在环境中展现的行为所作的解读，这些解读基于我们通过游玩《星际争霸II》和参与 SMAClite 开发所获得的游戏知识。

在进行选择时，我们希望展示每种算法，并为每种算法选择一个表现良好的场景——请注意，这意味着我们可能不会在所有场景中都使用表现最佳的算法，但所选算法的表现至少高于平均水平。然而，由于共有 9 种算法，而场景只有 6 个，且并非所有算法都表现良好，因此我们决定省略 COMA、MADDPG 和 IPPO，因为它们总体表现最弱。在每种情况下，我们都使用了最新可用的智能体参数检查点（即训练时间步数量最多的检查点）。

2s_vs_1sc – QMIX– **测试平均回报：16.22** – 其中一名追猎者是“诱饵者”，负责吸引脊针爬虫的注意力；它们会一起攻击，直到诱饵者的生命值降到很低，此时诱饵者便会撤退，让脊针爬虫转而攻击另一名追猎者。随后，它们一起攻击脊针爬虫，直到死亡；根据随机的攻击顺序，结果有时是胜利，有时是失败。

3s5z – VDN – **测试平均回报：20** – 追猎者和狂热者都会优先攻击敌方追猎者，因为敌方追猎者每秒造成的伤害比敌方狂热者更高。友方狂热者稍微向北移动，友方追猎者稍微向南移动，目的都是引导

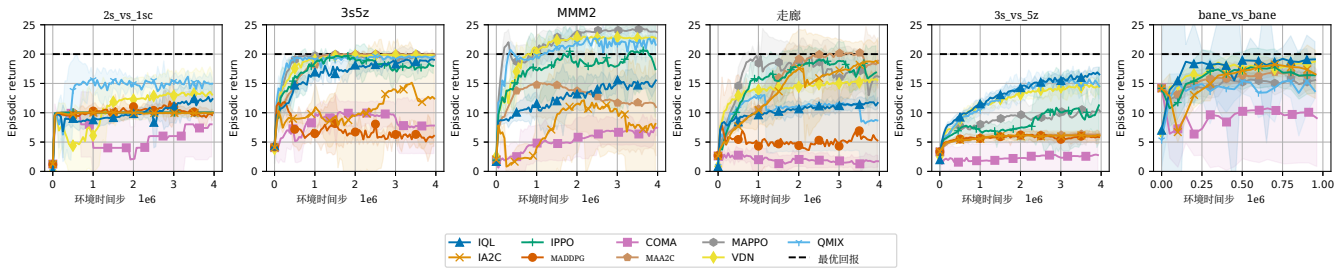


图3：训练过程中，使用不同算法训练的智能体随时间取得的测试回报。图中显示的是5次训练的平均值及其95%置信区间，各次训练使用了不同的随机数生成器种子。

敌方追猎者（其初始位置在敌军北侧）将注意力转向友方狂热者，后者能够承受更多攻击，同时也为友方追猎者在攻击时留出充足的机动空间。还值得注意的是，友方单位在生命值降至危险的低水平时，往往会及时撤离，让附近的其他友军吸引敌人的攻击。

MMM2 – MAPPO –**测试平均回报：24.63** – 所有盟军伤害输出者都优先集火敌方劫掠者，因为它们的攻击力更强；盟军医疗运输机一旦生命值较低，就会躲到其他单位身后，因为它是优先目标，很快就会被击落。有趣的是，盟军单位让敌方医疗运输机存活，并将其最后击杀，利用了治疗敌方单位会带来更高回报这一点，因为造成的总伤害更高——这就是智能体如何突破20的最优回报，达到接近25的数值。它们甚至会停止攻击敌方伤害输出者，任由它们接受治疗，因为医疗运输机无法治疗自己。

走廊 – MAA2C – **测试平均回报：20.25**– 狂热者向两侧散开，面对跳虫排成一条水平线，阻挡它们，使跳虫更难包围狂热者。这会使跳虫聚集在狂热者队列前方，从而减少狂热者随着时间推移所承受的总伤害。

3s_vs_5z – IQL – **测试平均回报：18.1** – 追猎者采用了预期的最优策略，在地图上风筝狂热者。生命值最低的追猎者在攻击时总会确保自己站在另外两名追猎者身后，以避免阵亡。当剩余狂热者的数量变少时，追猎者似乎确实会变得“懒得”风筝，这可能是因为阵亡不会受到惩罚，而且死亡风险较低时，站在原地攻击更容易。

bane_对_bane – IA2C – **测试平均回报：19.14**– 跳虫向西逃跑，以躲避敌方毒爆虫的爆炸；而盟军毒爆虫则向前冲锋，并在被敌方单位包围时爆炸。敌方跳虫和毒爆虫很快就在接连不断的爆炸中丧命。如果盟军毒爆虫爆炸后仍有敌人存活，盟军跳虫便会从藏身处现身并发起攻击。

5.3 跨环境零样本性能

为了验证我们的环境是否确实需要与 SMAC 相同的策略推理能力，我们针对每个场景进行了零样本迁移学习实验。具体来说，我们先在 SMAClite 中训练智能体，然后不进行任何再训练，便将它们放入 SMAC 中的相同场景。我们使用了与前一项实验相同的模型——具体而言，我们首先使用模型首次保存时（即训练过程刚开始时）的参数，然后使用模型最后一次保存时（即训练过程结束时）的参数，并比较测试时的平均回报。结果见表 1。

我们注意到，在所有 场景中，获得的平均回报均有所提升，其中一些场景的提升尤为显著，平均回报甚至翻了一倍或三倍。一个值得关注的例子是 bane_vs_bane 场景：即使是早期版本的智能体也已接近 20 的最优回报，但后续版本仍在此基础上有所提升。由于所有场景在 SMAClite 中训练后都取得了更高的回报，我们认为，这表明这两个环境需要相似的技能组合，并且从一个环境迁移到另一个环境进行迁移学习，是一种可行的训练策略。

5.4 环境性能基准测试

本节将考察环境本身的性能，以确认 SMAClite 的运行成本确实低于 SMAC。为获取此基准测试的数据，我们让智能体在每个场景中运行 20 次，每次都从可用动作中随机选择。以下所有实验均在配备 AMD Ryzen 3700X CPU 的计算机上运行。

在表2中，可以看到各环境每个时间步所花费的时间（不包括不属于环境的逻辑，例如动作选择）。我们注意到，纯 Python 代码确实比原始 SMAC 环境慢，但使用 C++ RVO2 插件后，环境的运行速度会快得多。

请注意，这些计时对应于运行单个环境步，其中依次包含 8 个游戏步，以及仅由环境执行的任何逻辑（例如计算奖励和确定观测）。如果我们要与人类玩家进行实验（假设 SMAClite 获得了某种人类控制扩展），就希望环境能够实时运行。在 SC2 和 SMAClite 中，每个游戏步（一个环境步包含 8 个游戏步）都被视为持续 $_{116}$ 秒；而且大多数人类玩家会以游戏内的“更快”速度玩 SC2，这相当于加速 40% [16]。因此，为了让环境实时运行，每个环境步所用时间最多可达 $\frac{8}{16*1.4} \simeq 0.357$ 秒。由此，所有版本的环境都完全能够运行所有已测试的

表 1：在 SMAClite 上训练的智能体置于原始 SMAC 环境中时取得的测试平均回报，分别采用训练期间首次保存和最后一次保存时的参数。另列出 SMAClite 上的后期回报以供参考。

场景	算法	早期回报	后期回报	SMAClite 上的后期回报
2秒_对比_1秒	QMIX	0	11.4	16.22
3s5z	VDN	3.09	8.96	20
MMM2	MAPPO	1.87	6.70	24.63
走廊	MAA2C	3.35	4.48	20.25
3 秒_对比_5z	IQL	3.13	9.33	18.1
bane_vs_bane	IA2C	18.97	19.84	19.14

表2：在用于训练的 SMAC 场景中，每个时间步的平均秒数。数据通过针对随机智能体运行该场景 20 次获得。

场景	SMAC	SMAClite	变化	SMAClite_plus	变化
2s_vs_1sc	0.004	0.007	+75%	0.003	-25%
3s5z	0.013	0.018	+38%	0.007	-46%
MMM2	0.017	0.028	+64%	0.010	-41%
走廊	0.014	0.092	+557%	0.010	-28%
3s_vs_5z	0.006	0.013	+116%	0.005	-17%
bane_vs_bane	0.049	0.086	+76%	0.024	-51%

实时场景——在运行需要多次执行环境的长时间训练时，SMAClite_plus 的优势最为明显。

在运行上述实验时，我们还测量了每个环境使用的 RAM 容量。不同场景之间差异不大，SMAC 环境的内存用量约为 600 MB，而 SMAClite 和 SMAClite_plus 环境约为 100 MB。因此，我们的轻量级环境运行所需的内存仅为原始 SMAC 环境的六分之一。

6 未来工作与结论

我们介绍了 SMAClite——一个用于多智能体强化学习的轻量级环境，它由一个模拟 SMAC 中《星际争霸II》迷你游戏的游戏引擎，以及一个便于使用熟悉的 JSON 格式为该引擎创建新场景和单位的框架组成。我们通过比较学习曲线和多智能体强化学习算法排名，以及开展零样本学习实验（在 SMAClite 上训练可提升智能体在 SMAC 上的表现），证明 SMAClite 带来了与 SMAC 相当的挑战。我们还表明，这种挑战所需的时间和内存成本都大幅降低。

SMAClite 引擎和框架都具有很强的可扩展性。由于环境不再受限于《星际争霸II》的依赖关系，开发者可以引入与《星际争霸II》无关的游戏机制，也可以突破游戏本身的技术限制。例如，可以实现 SC2 中从未出现过的新单位类型，并为其赋予新的攻击类型或技能；也可以利用场景框架，轻松创建无数复杂的谜题，供智能体解决。

也可以严格将 SMAClite 视为 SMAC 的扩展，并在保持其带来的性能提升的同时，努力使其尽可能接近 SMAC 这可能意味着重点关注第 5.3 节中的迁移学习实验等实验，并消除当前环境之间存在的各种差异。一个例子可能是投射物/动画模拟，这是 SMAC 与 SMAClite 之间的主要差异之一，并且在我们的实验中导致了 2s_与_1sc 以及 3s_与_5z 中的差异。

SMAC 生态系统中一项值得注意的贡献是 SMAC v2 [9], ，它引入了按回合动态生成的场景，并表明这种随机性会带来更具挑战性的场景，迫使智能体的策略更具适应性。这无疑是一个有趣的方向，我们将在不久的将来研究为 SMAClite 实现类似改进。

我们还可以改进经过修改的 ORCA 算法的战斗能力——从走廊场景可以看出，相较于 SMAClite 的跳虫，我们的 MAA2C 智能体操控的《星际争霸II》跳虫更擅长包围狂热者组成的墙。这可能是因为 RVO2 主要是一种通用的碰撞规避算法，而《星际争霸II》的算法则是专门手工编写的，旨在尽可能提升战斗表现。我们还可以借鉴《星际争霸II》的另一项功能：寻路算法，因为当地形比实验场景中的地形稍微复杂一些时，沿直线移动很快就会成为问题。

参考文献

[1] [不详]。Pygame。https://github.com/pygame/pygame [2] Stefano V. Albrecht、Cillian Brewitt、John Wilhelm、Balint Gyevnar、Francisco Eiras、Mihai Dobre 和 Subramanian Ramamoorthy。2021。面向自动驾驶的可解释目标预测与规划。载于*IEEE* 国际机器人与自动化会议，*ICRA 2021，5，2021 30* 1043 1049 10 1109 48506 2021中国西安，*5*月至*6*月。IEEE，–。https://doi.org/。/ICRA。。 9560849

[3] Nolan Bard、Jakob N. Foerster、Sarath Chandar、Neil Burch、Marc Lanctot、H. Francis Song、Emilio Parisotto、Vincent Dumoulin、Subhodeep Moitra、Edward Hughes、Iain Dunning、Shibl Mourad、Hugo Larochelle、Marc G. Bellemare 和 Michael Bowling。2020。Hanabi挑战：人工智能研究的新前沿。人工智能 280 2020 103216 10 1016 2019 103216 ( )，。https://doi.org/。/

[4] j.artint.。Stefan Behnel、Robert Bradshaw、Craig Citro、Lisandro Dalcin、Dag Sverre Seljebotn 和 Kurt Smith。2011。Cython：兼得两全之长。《计算科学与工程》13，2（2011），31–39。出版方：IEEE。[5] Marc G. Bellemare、Yavar Naddaf、Joel Veness 和 Michael Bowling。2013。街机学习环境：通用智能体评估平台。人工智能研究杂志 47 2013 253 279 10 1613 3912 ( )，–。

[6] https://doi.org/。/jair。Jur van den Berg、Stephen J. Guy、Ming C. Lin 和 Dinesh Manocha。2009。互惠式 n 体碰撞规避。载于机器人研究——第*14*届国际研讨会，*ISRR 2009，2009*年*8*月*31*日至*9*月*3*日，瑞士卢塞恩（*Springer*高级机器人学丛书，第*70*卷），Cédric Pradalier、Roland Siegwart 和 Gerhard Hirzinger（编）。Springer，3–19。https://doi.org/10.1007/978-3-642-19457-3 1_[7] Christopher Berner、Greg Brockman、Brooke Chan、Vicki Cheung、Przemyslaw Debiak、Christy Dennison、David Farhi、Quirin Fischer、Shariq Hashme、Christopher Hesse、Rafal Józefowicz、Scott Gray、Catherine Olsson、Jakub Pachocki、Michael Petrov、Henrique Pondé de Oliveira Pinto、Jonathan Raiman、Tim Salimans、Jeremy Schlatter、Jonas Schneider、Szymon Sidor、Ilya Sutskever、Jie Tang、Filip Wolski 和 Susan Zhang。2019。通过大规模深度强化学习玩 Dota 2。*CoRR* abs/1912.06680（2019）。http://arxiv.org/abs/1912.06680 arXiv：1912.06680。[8] Greg Brockman、Vicki Cheung、Ludwig Pettersson、Jonas Schneider、John Schulman、Jie Tang 和 Wojciech Zaremba。2016。OpenAI Gym。*arXiv*预印本 *arXiv:1606.01540*（2016）。[9] Benjamin Ellis、Skander Moalla、Mikayel Samvelyan、Mingfei Sun、Anuj Mahajan、Jakob Nicolaus Foerster 和 Shimon Whiteson。2022。SMACv2：合作式多智能体强化学习的新基准测试。（2022年6月）。https://openreview.net/forum?id=pcBnes02t3u [10] Jakob N. Foerster、Gregory Farquhar、Triantafyllos Afouras、Nantas Nardelli 和 Shimon Whiteson。2018。反事实多智能体策略梯度。载于第三十二届*AAAI*人工智能会议（*AAAI-18*）、第*30*届人工智能创新应用会议（*IAAI-18*）和第八届*AAAI*人工智能教育进展研讨会（*EAAI-18*）论文集，美国路易斯安那州新奥尔良，*2018*年*2*月*2*日至*7*日，Sheila A. McIlraith 和 Kilian Q. Weinberger（编）。AAAI出版社，2974–2982。https://www.aaai.org/ocs/index.php/AAAI/AAAI 18 17193 /paper/view/ Gillies、Sean、Butler、Howard、

[11] Pedersen、Brent、Matthias 和 Adam Stewart。[不详]。Rtree：面向 Python 的空间索引。https://github.com/Toblerity/rtree [12] Charles R. Harris、K. Jarrod Millman、Stéfan J. van der Walt、Ralf Gommers、Pauli Virtanen、David Cournapeau、Eric Wieser、Julian Taylor、Sebastian Berg、Nathaniel J. Smith、Robert Kern、Matti Picus、Stephan Hoyer、Marten H. van Kerkwijk、Matthew Brett、Allan Haldane、Jaime Fernández del Río、Mark Wiebe、Pearu Peterson、Pierre Gérard-Marchant、Kevin Sheppard、Tyler Reddy、Warren Weckesser、Hameer Abbasi、Christoph Gohlke 和 Travis E. Oliphant。2020。使用 NumPy 进行数组编程。《自然》585，7825（2020年9月），357–362。https://doi.org/ 10 1038 41586 020 2649 2 。/s - - - Peter Hart、

[13] Nils Nilsson 和 Bertram Raphael。1968。最小成本路径启发式确定的形式化基础。《*IEEE*系统科学与控制论汇刊》4，2（1968），100–107。https://doi.org/10.1109/tssc.1968.300136 出版方：电气电子工程师学会（IEEE）。[14] Mingyu Kim、Jihwan Oh、Yongsik Lee、Joonkee Kim、Seonghwan Kim、Song Chong 和 Se-Young Yun。2022。《星际争霸》多智能体挑战+：在没有精确奖励函数的情况下学习多阶段任务与环境因素。*CoRR* abs/2207.02007（2022）。https://doi.org/10.48550/arXiv.2207.02007 arXiv：2207.02007。[15] Aleksandar Krnjaic、Jonathan D. Thomas、Georgios Papoudakis、Lukas Schäfer、Peter Börsting 和 Stefano V. Albrecht。2022。面向仓储物流的可扩展多智能体强化学习：与机器人和人类协同工作者合作。_电子预印本：2212.11498。[16] Liquipedia。[不详]。《星际争霸II》Liquipedia。https://liquipedia.net/starcraft2/Main_Page [17] Ryan Lowe、Yi Wu、Aviv Tamar、Jean Harb、Pieter Abbeel 和 Igor Mordatch。2017。混合合作竞争环境中的多智能体行动者-评论家。载于神经信息处理系统进展*30*：*2017*年神经信息处理系统年度会议，*2017*年*12*月*4*日至*9*日，美国加利福尼亚州长滩，Isabelle Guyon、Ulrike von Luxburg、Samy Bengio、Hanna M. Wallach、Rob Fergus、S. V. N. Vishwanathan 和 Roman Garnett（编）。6379–6390。https://proceedings.neurips.cc/paper/2017/hash/68a9750337a418a86fe06c1991a1d64c-Abstract.html [18] Volodymyr Mnih、Adrià Puigdomènech Badia、Mehdi Mirza、Alex Graves、Timothy P. Lillicrap、Tim Harley、David Silver 和 Koray Kavukcuoglu。2016年。深度强化学习的异步方法。载于《第*33*届国际机器学习大会论文集，*ICML 2016*，纽约市，纽约州，美国，*2016*年*6*月*19*日至*24*日》（*JMLR*研讨会与会议论文集，第*48*卷），Maria-Florina Balcan 和 Kilian Q. Weinberger（编）。JMLR.org，1928–1937。http://proceedings.mlr.press/v48/mniha16.html [19] Igor Mordatch 和 Pieter Abbeel。2018年。多智能体群体中具身组合语言的涌现。载于《第三十二届*AAAI*人工智能会议论文集（*AAAI-18*）、第*30*届人工智能创新应用会议（*IAAI-18*）及第八届*AAAI*人工智能教育进展研讨会（*EAAI-18*），美国路易斯安那州新奥尔良，*2018*年*2*月*2*日至*7*日》，Sheila A. McIlraith 和 Kilian Q. Weinberger（编）。AAAI出版社，1495–

[20] 1502 18 17007 。https://www.aaai.org/ocs/index.php/AAAI/AAAI /paper/view/ Georgios Papoudakis、Filippos Christianos、Lukas Schäfer 和 Stefano V. Albrecht。2021年。协作任务中的多智能体深度强化学习算法基准测试。载于《神经信息处理系统数据集与基准专题论文集》第*1*卷，*NeurIPS*数据集与基准*2021，2021*年*12*月，线上，Joaquin Vanschoren 和 Sai-Kit Yeung（编）。https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/a8baa56554f96369ab93e4f3bb068c22-Abstract-round1.html[21] F. Pedregosa、G. Varoquaux、A. Gramfort、V. Michel、B. Thirion、O. Grisel、M. Blondel、P. Prettenhofer、R. Weiss、V. Dubourg、J. Vanderplas、A. Passos、D. Cournapeau、M. Brucher、M. Perrot 和 E. Duchesnay。2011年。Scikit-learn：Python中的机器学习。《机器学习研究期刊》12（2011年），2825–2830。[22] Matthias Plappert、Marcin Andrychowicz、Alex Ray、Bob McGrew、Bowen Baker、Glenn Powell、Jonas Schneider、Josh Tobin、Maciek Chociej、Peter Welinder、Vikash Kumar 和 Wojciech Zaremba。2018年。多目标强化学习：具有挑战性的机器人环境与研究倡议。*CoRR* abs/1802.09464（2018年）。http://arxiv.org/abs/1802.09464 arXiv：1802.09464。[23] Tabish Rashid、Mikayel Samvelyan、Christian Schroeder、Gregory Farquhar、Jakob Foerster 和 Shimon Whiteson。2018年。QMIX：用于多智能体深度强化学习的单调价值函数分解。载于国际机器学习大会。PMLR，4295–4304。[24] Mikayel Samvelyan、Tabish Rashid、Christian Schröder de Witt、Gregory Farquhar、Nantas Nardelli、Tim G. J. Rudner、Chia-Man Hung、Philip H. S. Torr、Jakob N. Foerster 和 Shimon Whiteson。2019年。《星际争霸》多智能体挑战赛。载于《第*18*届自主智能体与多智能体系统国际会议论文集，*AAMAS '19*，加拿大魁北克省蒙特利尔，*2019*年*5*月*13*日至*17*日》，Edith Elkind、Manuela Veloso、Noa Agmon 和 Matthew E. Taylor（编）。国际自主智能体与多智能体系统基金会，2186–2188。http: // dl.acm.org/citation.cfm?id 5932052 [25] John Schulman、Filip

Wolski、Prafulla Dhariwal、Alec Radford 和 Oleg Klimov。2017年。近端策略优化算法。*CoRR* abs/1707.06347（2017年）。http://arxiv.org/abs/1707.06347 arXiv：1707.06347。[26] Sybren A. Stüvel。[未注明日期]。用于最优互惠碰撞规避的Python绑定。https://github.com/sybrenstuvel/Python-RVO 2

[27] Peter Sunehag、Guy Lever、Audrunas Gruslys、Wojciech Marian Czarnecki、Vinícius Flores Zambaldi、Max Jaderberg、Marc Lanctot、Nicolas Sonnerat、Joel Z. Leibo、Karl Tuyls 和 Thore Graepel。2018年。基于团队奖励的协作多智能体学习价值分解网络。载于《第*17*届自主智能体与多智能体系统国际会议论文集，*AAMAS 2018*，瑞典斯德哥尔摩，*2018*年*7*月*10*日至*15*日》，Elisabeth André、Sven Koenig、Mehdi Dastani 和 Gita Sukthankar（编）。国际自主智能体与多智能体系统基金会，南卡罗来纳州里奇兰／美国计算机协会，2085–2087。http://dl.acm.org/citation.cfm?id 3238080

[28] 1993 Ming Tan。多智能体强化学习：独立智能体与协作智能体。在第十届国际机器学习会议论文集中。330–337。[29] Guido Van Rossum 和 Fred L Drake Jr. 1995年。*Python*参考手册。荷兰数学与计算机科学研究中心，阿姆斯特丹。[30] Florian Vaussard、Julia Fink、Valerie Bauwens、Philippe Rétornaz、David Hamel、Pierre Dillenbourg 和 Francesco Mondada。2014年。机器人吸尘器进入家庭生态系统的经验教训。《机器人与自主系统》62，3（2014年），376 391 10 1016 2013 09 014 – 。https://doi.org/ 。 /j.robot. 。 。Oriol

[31] Vinyals、Igor Babuschkin、Wojciech M. Czarnecki、Michaël Mathieu、Andrew Dudzik、Junyoung Chung、David H. Choi、Richard Powell、Timo Ewalds、Petko Georgiev、Junhyuk Oh、Dan Horgan、Manuel Kroiss、Ivo Danihelka、Aja Huang、Laurent Sifre、Trevor Cai、John P. Agapiou、Max Jaderberg、Alexander S. Vezhnevets、Rémi Leblond、Tobias Pohlen、Valentin Dalibard、David Budden、Yury Sulsky、James Molloy、Tom L. Paine、Caglar Gulcehre、Ziyu Wang、Tobias Pfaff、Yuhuai Wu、Roman Ring、Dani Yogatama、Dario Wünsch、Katrina McKinney、Oliver Smith、Tom Schaul、Timothy Lillicrap、Koray Kavukcuoglu、Demis Hassabis、Chris Apps 和 David Silver。2019年。通过多智能体强化学习在《星际争霸II》中达到特级大师水平。《自然》575，7782（2019年11月），350–354。https://doi.org/10.1038/s41586-019-1724-z 期号：7782 出版者：《自然》出版集团。

出版集团。[32] Oriol Vinyals、Timo Ewalds、Sergey Bartunov、Petko Georgiev、Alexander Sasha Vezhnevets、Michelle Yeo、Alireza Makhzani、Heinrich Küttler、John P. Agapiou、Julian Schrittwieser、John Quan、Stephen Gaffney、Stig Petersen、Karen Simonyan、Tom Schaul、Hado van Hasselt、David Silver、Timothy P. Lillicrap、Kevin Calderone、Paul Keet、Anthony Brunasso、DavidLawrence、Anders Ekermo、Jacob Repp 和 Rodney Tsing。2017年。《星际争霸II》：强化学习的新挑战。*CoRR* abs/1708.04782（2017年）。http://arxiv.org/abs/1708.04782 arXiv：1708.04782。[33] Chao Yu、Akash Velu、Eugene Vinitsky、Yu Wang、Alexandre Bayen 和 Yi Wu。2021年。PPO在协作多智能体博弈中的惊人效果。*arXiv*预印本 *arXiv:2103.01955*（2021年）。

附录 A SMACLITE框架详细信息

A.1 场景定义

我们的框架接受包含单个 JSON 对象的场景 JSON 文件。每个场景都应具有用于标识它的名称，并应通过 num_allied_units 和 num_enemy_units 参数指定盟军单位和敌军单位的数量。场景中的单位应通过 groups 参数列出，其中每个群组都是一个 JSON 对象，包含用于指定群组中心的 x 和 y 参数、用于指定单位所属队伍的 faction 参数（ALLY 或 ENEMY），以及一个 units 参数，其中包含单位类型及其数量的对象。每个群组最初都会按照以指定位置为中心的正方形形状进行排列。请注意，这些群组除了确定单位位置外没有其他作用——初始单位部署完成后，智能体和单位都不会获得任何关于其所属群组的信息。

框架支持两种指定单位类型的方式。若要使用标准单位类型，应使用其大写名称（例如 ZERGLING）。另一种指定单位类型的方式是提供包含其规格的 JSON 文件路径。框架还支持可选的 custom_unit_path 参数——如果指定了该参数，该路径将添加到所有以路径形式指定的单位类型之前。请注意，如果缺少 .json 扩展名，框架会自动重新添加该扩展名；因此，如果自定义单位路径为 path/to，单位类型规格为 type，框架将查找文件 custom/unit/type.json。框架还要求提供一个 attack_point，该点通常靠近友方单位的初始位置——在整个场景过程中，敌方单位将向该点行进（详情请参见第4.6节）。

该框架还支持两种定义地形的方式：一种是设置 terrain_preset 参数，并指定标准地形的名称（需大写，例如 CORRIDOR）。也可以在场景定义文件中通过 terrain 参数提供地形，该参数应为由字符串组成的列表，构成一个矩形二维数组。该框架支持两种地形类型：_ 表示可通行，X 表示不可通行。框架还要求始终为场景指定宽度和高度，且二者应与地形尺寸相符。请注意，所有基于 SMAC 改编的场景尺寸均为 32 × 32。

最后，该框架还需要提供一些有关参与场景的单位的基本信息。参数 ally_has_shields 和 enemy_has_shields 用于指定哪些队伍（如果有）启用护盾；参数 num_unit_types 和 unit_type_ids 则指定智能体在观察中针对各种单位类型所获得的 ID。前者应为一个数字（请注意，对于双方均为同质单位，即每队都只有一种单位类型的场景，基于 SMAC 改编的场景均使用 0）；后者应为一张映射表，将单位类型说明（同上）映射到从 0 到 num_unit_types 减 1 的数字。映射表的长度必须等于 num_unit_types。

我们将多个标准场景作为独立的 OpenAI Gym 环境随本环境一同提供（例如 smaclite/2s3z-v0）。如需使用自定义场景文件，应使用 smaclite/custom-v0 Gym 环境，并通过 map_file 参数提供场景文件的路径。

A.2 单位定义

每个单位都会被分配一种特定的 **类型**，并具有一组会影响环境机制的不同属性。我们的框架支持在定义各种单位类型的 JSON 文件中设置多种属性，以便轻松进行自定义。

首先，该框架支持通过多个属性定义单位的资源，包括最大生命值和生命值恢复，分别由 hp 和 hp_regen 属性指定。若单位具有护盾，则可通过 shield 属性指定；最大能量和初始能量则分别通过 energy 和 initial_energy 属性指定。要指定单位的尺寸，可在 thesize 属性中提供其直径；要指定单位的速度（每秒移动距离），则可使用 speed 参数。

接下来，该框架支持通过多个属性定义单位的战斗能力。combat_type（始终为 DAMAGE 或 HEALING）定义了单位在战场上的主要职责；damage 定义其攻击力，armor 则定义其防御能力。每个单位都有一个 attack_range，用于定义单位必须靠近目标到什么程度（按边界到边界的距离计算）才能攻击或治疗目标[4]。请注意，虽然此属性通常应为数值，但也接受特殊值 MELEE，表示该单位应采用标准近战距离，即只能近距离攻击。有些单位每次攻击可以造成多次伤害，可通过 attacks 属性实现（例如，设为 2 表示一次攻击两次）；攻击后，每个单位都必须等待其特定的冷却时间才能再次攻击。每种单位类型还应定义一个 minimum_scan_range，用于控制单位搜索目标时的搜索范围。

在 SMAClite 中，每个单位都位于特定的平面，目前支持三种：GROUND、AIR 和 COLOSSUS。移动时，单位只会避开与自己处于同一平面的单位发生碰撞，而且只有地面单位会受到静态障碍物的影响。SMAClite 中的所有单位都只能将其有效_目标列表中所列平面上的单位作为目标。例如，如果某个单位的列表中没有 AIR，那么它就永远无法攻击空中单位。但请注意，无论有效_目标列表如何，所有单位都能以 COLOSSUS 平面上的单位为目标。

为了支持让某些单位类型对特定的其他类型更具优势，该框架还支持单位属性系统（例如 BIOLOGICAL）以及基于属性的加成；加成定义为从属性到加成值的映射。例如，如果某个单位类型对 ARMORED 单位有 20 点加成，那么它每次攻击具有该属性的单位时，都会造成额外 20 点伤害。

最后，该框架支持多种单位攻击类型，由 targeter 和 targeter_kwargs 属性定义。最常见的攻击类型是 STANDARD，即单位每次只攻击一个目标。其他攻击类型各自仅用于一种标准单位，但也可以轻松复用于自定义单位类型。KAMIKAZE 攻击类型会使单位在攻击时爆炸，对自身周围指定半径的圆形区域造成伤害，并在此过程中死亡。LASER_BEAM 类型会沿着与攻击者和目标之间连线垂直的方向发射激光——

[4]请注意，这与第3.2节中提到的智能体目标范围不同——该范围只影响智能体观测，而这里的范围则实际决定了何时能够发动攻击。

表3：实验中使用的战斗场景。

Name	友方单位	敌方单位	描述
2秒_对比_1秒	2	1	两只追猎者——强大但脆弱的远程单位——对战一只脊针爬虫——非常强大的单位，但无法移动。追猎者需要利用脊针爬虫无法移动这一弱点，削减它的生命值。
3s5z	8	8	一张对称地图，每支队伍有三名追猎者和五名狂热者——近战单位它们造成的伤害较低，但在阵亡前可以承受大量攻击。
MMM2	10	12	每支队伍都有一些陆战队员——脆弱但攻击速度快的远程单位，还有一些劫掠者——更耐打、攻击力更强的单位，但无法攻击飞行单位；此外还有一艘医疗运输机 – 一种飞行治疗单位。
走廊	6	24	盟军队伍只有 6 名狂热者来抵挡 24 只跳虫——它们极其脆弱，但速度非常快，且有望凭借数量优势压倒毫无防备的敌人并包围他们。
3s_vs_5z	3	5	三名追猎者寡不敌众，需要利用自身的利用范围优势。
bane_对_bane	24	24	每支队伍都有一些跳虫和一些具有爆炸性的 KAMIKAZE 毒爆虫，每个毒爆虫都可以通过一次爆炸轻松消灭数只跳虫。

更具体地说，激光线是一个具有指定宽度和高度的矩形。最后，治疗单位使用 HEAL 攻击类型。

附录 B SMAClite 框架示例

B.1 有效场景文件示例

以下是一个自定义场景示例，与内置场景 10m_vs_11m 类似，但使用的是自定义单位。

{ "name": "10m_对_11m", "custom_单位_path": " smaclite/env/units/smaclite_units", "num_盟军_单位": 10, " num_敌军_单位": 11, "群组": [{ "x": 9, "y": 16, "阵营": "盟军", " 单位": { "example_custom_单位": 10 } }, { "x": 23, "y": 16, "阵营": " 敌军 ", " 单位 ": { "example_custom_ 单位 ": 11 } }]," 攻击 _ point": [9, 16],"地形_preset": "NARROW", "num_单位_types": 0,
"ally_has_护盾": false, " enemy_has_护盾": false }

B.2 有效单位文件示例

以下是一个自定义单位示例，与内置的 MARINE 单位类似，但扫描范围要大得多（实际上是全局范围）。

{ "hp": 45, "护甲": 0, "伤害": 6, "冷却时间": 3, " speed": 3.15, "attack_range": 3, "size": 3, " attributes": ["LIGHT", "BIOLOGICAL"]," minimum_scan_range": 100, "valid_targets": ["GROUND", "AIR"]}

附录 C SMAClite 游戏循环详解

在本节中，我们将详细描述每个游戏步骤的逻辑；每个环境步骤中包含多个游戏步骤。每个游戏步骤由多个阶段组成。这些阶段由我们设计，我们并不知道《星际争霸II》中的游戏步骤是否遵循任何类似的顺序——这种结构只是产生了最合理的、类似于《星际争霸II》的游戏规则集。在下一个阶段开始之前，所有单位都必须执行上一阶段的逻辑。这是确保单位对其他单位的感知（注意：与智能体的观测无关）在整个游戏步骤执行过程中保持一致所必需的。

C.1 目标清理

首先，根据特定于命令的逻辑，每个单位都可能失去它在上一个游戏步中正在攻击或治疗的目标。所有单位都必须先执行此逻辑，然后才能为当前游戏步声明各自的期望速度，因为其他单位在计算自己的期望速度时可能会访问目标信息。

执行 noop、停止和移动命令的单位会立即失去已有目标，因为这些都是非战斗命令。执行目标命令的单位会获得命令指定的目标；如果它们已经在攻击该目标，则会保留该目标。对于攻击_移动命令，单位会考虑多个因素：它们总会失去死亡目标，而绝不会失去在上一个游戏步中攻击过自己的目标。除此之外，如果目标超出攻击范围，单位就会失去该目标。

C.2 速度准备

在此阶段，所有单位都会声明自己在当前游戏步的期望速度。必须让它们在通过 ORCA 算法进行速度调整之前完成此操作，因为单位能够感知彼此的期望速度，并据此做出碰撞规避决策。

对于 noop 和停止命令，单位的期望速度始终为 $\mathbf{0}$。对于移动命令，如果单位的最大速度为 v_{max}，其当前位置为 $\mathbf{x}$，而它想要移动到的位置为 $\mathbf{y}$，且 $\mathbf{x} \neq \mathbf{y}$，那么该单位始终会将 $(\mathbf{y}-\mathbf{x})\frac{v_{max}}{||\mathbf{y}-\mathbf{x}||}$ 声明为期望速度。如果 $\mathbf{x}=\mathbf{y}$，那么期望速度则为 $\mathbf{0}$。简而言之，单位始终希望沿直线朝目的地移动——这不同于星际争霸II的引擎，该引擎内置了基于 A*算法 的寻路算法 [13] ——但我们决定省略这一功能，因为它只在 SMAC 提供的众多场景之一（2c_vs_64zg）中才有影响。

如果单位的命令是目标命令，单位会考虑自己与目标单位之间的距离。设单位的攻击范围为 d_{max}，半径为 r_A，目标半径为 r_B，并将它与目标之间的距离定义为 $d(A, B)$。那么，当 $d(A, B) > r_A + d_{max} + r_B$ 时，单位会判定自己正沿直线朝目标移动，并按照移动命令处理。否则，它会判定自己正在攻击或治疗，且其期望速度为 $\mathbf{0}$。

最后，如果单位的命令是攻击_移动，该单位会先在其扫描范围内寻找有效目标。需要注意的是，如果该单位是伤害输出者，它会将任何敌方治疗者视为优先目标，并将所有其他敌方单位视为非优先目标——如果在目标清理阶段后该单位仍有目标，此时它切换目标的唯一情况是：当前目标不是优先目标，且其扫描范围内存在优先目标。如果它没有目标或需要切换目标，就会从有效目标中选择优先级最高且距离最近的目标。治疗者的过程略有不同：它们会将己方队伍中生命值未满或正在攻击其他单位的非治疗单位视为有效目标。治疗单位会从有效目标中选择生命值最低的单位作为目标。目标选择过程结束后，如果该单位有目标，就按目标命令的方式行动；如果没有目标，则朝攻击移动目的地行动，方式与移动命令相同。请注意，治疗者会在每个游戏步中调整其最大速度，使之与扫描范围内速度最慢的盟军单位保持一致——我们实现这一行为，是为了防止它们在自身速度最快时跑到军队前方。

这一阶段有几处与 SC2 有意不同：在 SC2 中，治疗者并非只能以非治疗者为目标，也可以将其他治疗者或自己作为目标。这只适用于 SMAC 中有限的单位集合；我们认为，这样简化 SC2 规则集很合适，而且不会改变 SMAC 中的任何内容。另一个不同之处是，在 SC2 中，伤害输出者并不会将所有治疗者视为优先目标；只有在 SMAC 中，由于其作者对地图文件所做的修改，伤害输出者才会如此。我们再次认为，这种简化符合 SMAC 的设定，但不符合 SC2。

C.3 速度调整

每个单位 A 使用 ORCA 算法来确定能够规避碰撞的实际速度，同时考虑其自身的期望速度、邻居单位和障碍物。为进行碰撞规避，该单位会将自身半径（$r_A + r_{max}$）τ 范围内的所有单位视为邻居，其中 r_{max} 是场景中任意单位的最大半径，τ 是时间范围；并将自身半径 $r_A + \tau v_{A,max}$ 范围内的所有障碍物纳入考虑。在 SMAClite 中，我们始终使用 $\tau = 1$，即单位希望确保在 1 秒或 16 个游戏步内避免碰撞。ORCA 算法会根据每个单位的期望速度，返回其实际速度。有关我们对 ORCA 算法的实现细节，请参阅第 4.5 节。

C.4 游戏步骤执行

这是游戏步骤的最后阶段。由于顺序在这里很重要（某个单位可能会在执行指令之前被消灭），因此单位会执行各自的阶段逻辑，顺序随机，并由环境随机种子控制；每个游戏步骤都会重新随机排列顺序。请注意，我们没有关于《星际争霸II》是否也会随机排列游戏步骤执行顺序的数据；这只是我们认为在公平性和简单性之间比较合理的折中方案。

每个单位，无论其命令类型如何，都会通过执行若干标准更新来开始此阶段。它根据上一阶段计算出的实际速度更新其位置（即 $\mathbf{x}_{new} = \mathbf{x} + \mathbf{v}_{16}^{actual}$，因为速度始终以每秒距离为单位定义），在适用时减少其冷却时间（即如果冷却时间大于 0，则减少 $\frac{1}{16}$ 秒），并在适用时恢复生命值、能量和/或护盾。随后，它继续执行特定于命令的逻辑。

此时唯一会执行命令特定逻辑的情况是：单位的命令为 target，或带有非空目标的 attack_move；并且单位在速度准备阶段声明自己正在攻击或治疗，且其当前冷却时间为 0。除此之外，单位的游戏步逻辑到此结束。在唯一可执行动作的情况下，它会攻击或治疗其目标。

攻击造成的伤害会先作用于敌方护盾，再作用于敌方生命值；对目标生命值造成的伤害会受到其护甲的减免。如果单位的攻击属性大于 1，那么在此

步骤中，它会连续造成多次伤害。如果单位进行了攻击，其冷却时间随后会设为最大值，该值由单位类型决定。治疗单位以每秒 9 点生命值的速度为目标治疗，每恢复一点生命值会消耗13 点能量。

请注意，此时单位可能会攻击或治疗技术上已超出其射程的单位，因为攻击/治疗指令是在速度准备阶段发出的，而目标可能自那之后稍微移动了位置。这是有意为之，旨在避免单位速度相近时无休止地追逐。

同样，我们没有数据表明《星际争霸》II 引擎是否采用了类似的简化，但这条规则在 SMAClite 中效果良好。

此外，这里也有意与 SC2 做了一些变化。为求简洁，我们不模拟攻击动画或攻击弹道——单位一旦声明发动攻击，攻击便会在该时间步立即发生。这会导致与 SC2 存在一些细微差异，但我们认为误差范围可以接受。