arXiv:2301.08030v1 [cs.LG] 19 Jan 2023

竞争性生存环境中的多智能体互动

Andrea Fanti 罗马智慧大学

摘要

在强化学习中，解决难探索环境是一项重要挑战。人们已提出并研究了多种方法，例如内在动机、智能体与任务的协同进化，以及多智能体竞争。尤其是，多个智能体之间的互动已被证明能够产生与人类相关的涌现行为，而这类行为在单智能体设置中可能难以甚至无法学会。本研究开发了一个可扩展的多智能体互动竞争环境，该环境具备真实物理机制和与人类相关的语义。此外，研究还针对该环境的不同变体开展了多项实验，得出了一些简单的涌现策略，并为未来改进提出了具体方向。本文内容是作者于2022年在罗马智慧大学攻读人工智能与机器人学硕士学位期间完成的论文《竞争性生存环境中的多智能体互动》的一部分。

1 引言

创建能够解决复杂人类相关任务的人工智能体，是机器学习面临的主要挑战之一。当这些任务需要与环境进行实时交互时，通常会通过强化学习（RL）对其进行建模和求解。在强化学习中，一个或多个智能体通过一系列离散的观测和动作与随机环境交互。为了引导智能体实现目标，还会向它们提供有关决策的反馈，即奖励信号；因此，强化学习智能体的目标是最大化其累积奖励。由于这些概念具有普遍性，强化学习自然适用于大量与人类相关的任务以及涉及物理交互的任务。

近年来，强化学习借助深度学习的泛化能力，发展出了深度强化学习（Deep RL）[22][4][37][29][31][14][18][15][16]。然而，尽管深度强化学习算法显著提升了许多任务的当前最佳水平，但在困难探索 环境中，它们仍面临挑战 [12]。解决这些任务的难点，往往不仅在于任务本身固有的障碍，还在于难以指定合适的奖励信号。具体而言，要设计一种能够引导强化学习智能体逐步完成实现目标所需相关里程碑的奖励信号，往往并不实际，甚至不可能。不幸的是，许多具有挑战性的人类相关问题也是困难探索问题。这是因为，对于这类任务，通常只能或适合指定抽象目标，导致奖励函数无法明确指导智能体如何 实现目标[12]。此外，奖励信号甚至可能产生误导：尤其是在短期内优化表现会使智能体更不可能达成任务主要目标时。

标准深度强化学习算法的另一个问题是，它们专注于为特定任务生成单一解 [38][2]。因此，它们无法直接生成具备多种技能的智能体，也无法生成一组在单项任务上表现相近的多样化智能体。

为了解决这些问题，人们开发并研究了多种方法。内在动机就是一个例子，它通过提供与原始任务无关的额外奖励，激发智能体的好奇心。这种内在奖励经过专门设计，旨在促使智能体探索环境状态空间中的新区域。

事实证明，基于协同进化的算法在生成和解决难探索环境方面也非常有效。Paired Open–Ended Trailblazer [38][39]算法便是一例，它不断演化环境的变体，同时演化能够解决这些环境的智能体

使用进化策略 [28]。POET 通常会生成一些环境，而只在该环境中训练的外部智能体无法解决这些环境 [38], ，这表明有些任务无法通过直接优化解决，智能体必须经历适当的课程。

另一种值得关注的方法，是利用多智能体场景中的竞争性来产生涌现行为。即使这种方法通常依赖标准的深度强化学习算法，多个智能体之间的竞争性交互也能激发复杂的行为和技能，而这些行为和技能若没有这种交互便无法实现。当一个或多个智能体向对手提出足够大的挑战时，就会为原先的对手带来新的挑战。如果环境允许，这种交锋便能无限持续，在此过程中不断产生更高级的技能。这种利用多智能体交互的方法已成功应用于各种复杂游戏 [17][36] 和简单的基于物理的环境 [33][17][3][19][2]。尤其是 Baker 等人，着重关注智能体习得技能与人类的相关性，以及其训练所在的简单模拟世界的物理基础。本研究也关注这些方面，旨在实现双重目标：（1）开发一个计算效率高且易于扩展的多智能体竞争环境；（2）通过对该环境的不同变体进行实验，产生有趣的智能体涌现行为。第一个目标通过开发一个基于 Box2D 物理模拟库、可扩展且高效的框架得以实现，该框架能够构建基于真实物理机制的模块化环境。随后，我们利用该框架开发了具体的多智能体环境，其语义借鉴了常见的生存类电子游戏。这些环境涵盖了不同程度的竞争性和环境复杂度，也将智能体划分为相互对立的团队。第二个目标则通过在若干实验中使用这些具体变体，并采用标准的深度强化学习技术，部分得以实现。这些实验产生了一些简单的涌现策略，也为今后改进这些环境变体提供了具体方向。

2 相关工作

有几种算法显著增强了标准深度强化学习算法的探索能力。内在动机方法会引入一种额外的奖励信号，称为内在奖励，旨在明确鼓励智能体探索环境。在此背景下，原始奖励信号也称为外在奖励。内在奖励由内在奖励函数 R_i给出；与外在奖励函数不同，它通常是非平稳的。大多数内在动机方法会塑造内在奖励，使其在新颖状态下更高，从而产生寻求未见状态的“好奇”智能体。一种实现方式是采用基于计数的方法 [5][23], ，使内在奖励与某个状态被访问的次数成反比。这种方法在状态数量有限的环境中很直接，但在状态空间无限时则需要格外谨慎。另一种截然不同的方法，是将内在奖励定义为与智能体状态转移 [21][34][1][25][8][10][9]相关问题的预测误差。例如，预测环境的正向或逆向动力学，甚至预测恒为零这一类简单函数，在实践中都能取得良好效果。基于预测误差的内在动机方法存在一个问题：预测误差本身可能由多种原因造成，并非所有原因都与智能体的探索表现 [9]有关。随机网络蒸馏（RND） [9][8]是一种先进的内在动机方法，它使用随机任务上的预测误差来计算内在奖励，该任务由一个随机初始化且固定的神经网络表示。尽管与标准强化学习算法相比，内在动机方法在大多数难以探索的任务中取得了显著更好的结果，但仍有一些问题尚未解决，例如脱离和偏轨 [12]。Go-Explore [12]是一种旨在解决这些问题的算法，它通过显式保存有趣状态和轨迹的档案库，使探索能够从中断处继续。

克服深度强化学习算法中的困难探索问题的另一种方法，是为智能体提供一系列中间任务组成的课程 ，这种技术称为课程学习</*style*。在此语境中，课程 是一系列难度逐渐增加的任务，使得智能体解决其中一个任务后，便可迁移到下一个任务，并通过直接最大化其回报来解决该任务。尽管课程可以手动实施，但这种方法难以适应环境复杂度的提升。自动生成合适的课程是强化学习研究的活跃领域。

一种可能的方法是利用多智能体环境。这种方法的思路是，当某个智能体改进其策略并击败其他智能体时，后者便会受到压力，

进而改进自己的策略。只要这些改进不会导致某个智能体强大到其他智能体难以击败，就能为所有智能体持续创造难度不断增加的挑战。这也类似于地球上的生命演化，因为竞争与共同进化在生物体的能力和多样性发展中发挥了根本作用。Sukhbaatar 等人和 Tesauro 将这种方法成功应用于复杂抽象游戏和复杂的多人电子游戏。Sims、Jaderberg 等人、Bansal 等人、Liu 等人和 Baker 等人也利用这种方法，在简单的基于物理的环境中通过多智能体间的相互作用产生了涌现行为。

自动课程学习的另一种重要方法是智能体与任务共同进化，例如 Paired Open–Ended Trailblazer（POET）算法 [38][39]。POET 的灵感来自最小标准进化 [6] 以及其他开放式进化方法 [24]。Dharna 等人提出了 POET 的一种变体，该方法让电子游戏关卡与游玩这些关卡的智能体共同进化。

3 一个竞争性生存环境

本研究的第一部分是开发一个基于生存的多智能体强化学习环境。设计时优先考虑了以下方面：

- 环境动力学的物理基础，尤其是经典力学；
- 能够改变智能体或团队之间语义复杂度和竞争性的水平；
- 计算效率，确保环境不会成为实验设置中的瓶颈；
- 实现的可扩展性，使得无需修改现有代码即可实现更多语义；
- 实现的模块化，使得在需要时可以轻松“关闭”不同的力学机制，或以不同方式组合这些机制。

本节说明所开发环境的动力学和语义，附录 A 则讨论其实现，以及为使其具备可扩展性和模块化而开发的框架。

该环境设定在一个平面世界中，物体遵循经典力学定律。平面上的相关区域是一个由四面墙壁围成的房间 。环境中可以有一个或多个智能体，它们同时行动。智能体可以通过多种方式与环境中的其他实体互动，包括受到各种来源造成的伤害。这些互动的语义详见第 3.1、3.2 和 3.3 节，第 3.4 节则定义了每个智能体的动作空间和观测空间。最后，第 3.5 节讨论奖励机制和回合终止条件。请注意，这种环境有许多可能的变体；图 1、2 和 3 展示了其中一小部分。

3.1 智能体

智能体被建模为具有固定半径的圆形。它们可以沿着与自身朝向平行和正交的方向对其物体施加力，还可以沿着垂直于房间平面的轴施加力矩。所有智能体的生命值均为整数；当生命值降至 0 或更低时，它们便会消失，并被视为死亡。智能体可以使用*Melee*攻击其他物体，包括其他智能体。这是一种近距离武器，建模为一条长度固定、与智能体面朝方向平行的线段。只有目标接触到该 Melee 时，攻击才会成功。

智能体还可以在一个槽位数量固定的小型物品栏中携带物品。最后一个槽位中的物品可以使用、丢弃到地面上，或交给距离足够近的另一个智能体。此外，当智能体死亡时，其物品栏中的物品会被丢到地面上，并以该智能体最后所在的位置为中心，随机散落在一小片圆形区域内。物品的详细说明见第3.2节。

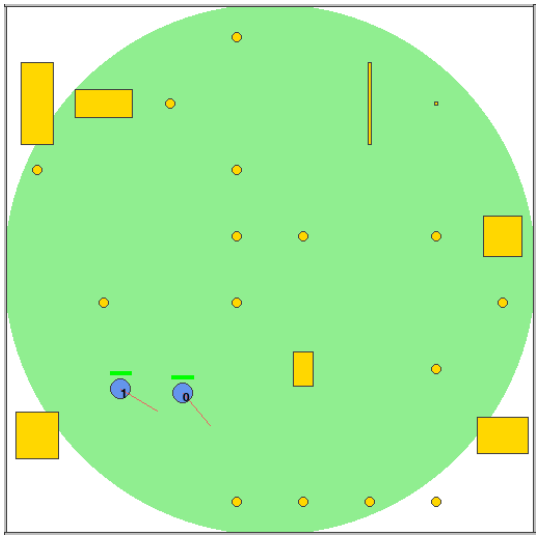


图1：包含治疗物品、箱子和2个智能体的环境变体屏幕截图。智能体发出的线段是Melee，它们上方的绿色条表示生命值，而显示在它们旁边的数字是智能体编号。

根据环境变体，智能体还可以分成不同团队。在这种情况下，智能体只能攻击团队以外的其他智能体，并且只能将物品交给团队内的智能体。此外，同一团队中的智能体完全共享奖励。

3.2 物品与物体

物品被建模为小型、无形的圆形实体。智能体触碰物品实体后即可将其拾取到物品栏中，该物品实体随后便会消失。每种物品在使用时都有特定的行为。此环境包含两类物品：治疗物品 和对象物品。治疗物品只会增加使用者的生命值。对象物品则会在使用者所在位置生成其他物体，例如箱子。这些物体可以被打碎，从而丢弃一个能让它们重新生成的物品。在此环境中，物体是矩形箱子；每个箱子的尺寸都在每个回合开始时随机确定。

3.3 安全区

安全区 是竞技类“大逃杀”电子游戏中一种广受欢迎的机制。它是地图上一个会移动的区域，并会随着游戏推进而缩小，对所有身处区域外的玩家造成少量但持续不断的伤害。因此，最终未能留在安全区内将会致命。此外，安全区通常会在游戏接近尾声时缩小至一片虚无区域，使玩家无法无限期存活。安全区本身并不会在环境中引入竞争性；但是，如果搭配适当的奖励机制，它会间接迫使智能体为争夺安全区内的位置而战，以延长存活时间。

在此环境中，安全区为圆形，并交替经历静止阶段和收缩移动阶段。每个阶段的中心位置都是随机抽取的，因此安全区始终完全位于房间内。在每个收缩移动阶段，安全区的半径线性减小，直至最后一个阶段结束后变为零。

3.4 观测和动作空间

每个智能体可用的观测 x 形式如下

$$x = (x_{\text{self}}, x_{\text{zone}}, x_{\text{entities}})$$

下文将更详细地介绍所有组成部分。观测中的所有标量值均为实数，因此观测空间是连续的。

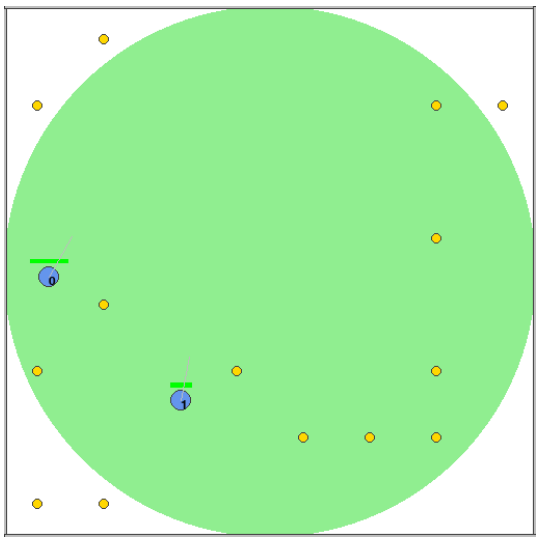


图 2：仅包含治疗物品和 2 个智能体的环境变体屏幕截图。

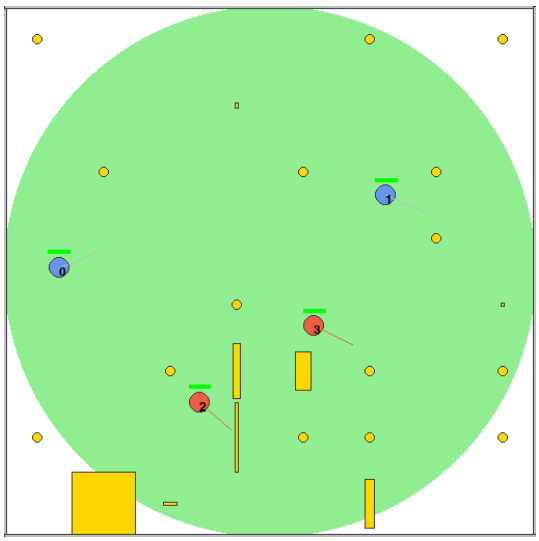


图 3：包含治疗物品、箱子以及分为多个团队的智能体的环境变体屏幕截图。

x_{self} 包含有关观察智能体的信息，其形式为

$$x_{\text{self}} = (i, h, x, y, \theta, v_x, v_y, \omega)$$

其中：

- i 是智能体的标识符；
- h 是其当前生命值；
- $p = (x,\ y)$ 是其绝对位置；
- θ 是其朝向；
- $v = (v_x,\ v_y)$ 是其速度；
- ω 是其角速度。

此外，如果智能体被划分为团队，x_{self} 还会在智能体编号之后包含团队的标识符。

x_{zone} 会观测安全区当前的中心和半径，以及安全区下一阶段的中心和半径。更确切地说，其形式为

$$x_{\text{zone}} = (c_x, c_y, r, c'_x, c'_y, r')$$

其中 $(c_x,\ c_y)$ 是当前安全区的中心，r 是其半径，而 $(c'_x,\ c'_y)$ 和 r' 则分别是安全区在下一个静止阶段的中心和半径。

观测的其余部分 x_{entities}包含环境中各种实体的所有信息，其形式如下：

$$x_{\text{entities}} = (x_{\text{other}}, x_{\text{heal}}, x_{\text{box}}, x_{\text{boxitem}}, x_{\text{healslot}}, x_{\text{boxslot}})$$

x_{entities} 的所有组成部分 x_{entities_i} 结构相似，但大小不同，并且分别对应环境中的不同类型实体。每个 x_{entities_i} 的形式如下：

$$x_{\text{entities}_i} = (x_{\text{allents}_i}, x_{\text{mask}_i})$$

x_{allents_i} 是一个矩阵，其中每一行都存储该类型的一个实体的实际信息。相较之下，x_{mask_i} 部分是一个包含二进制值的单一向量。每个值对应于 x_{allents_i} 中的一行，用于表示观察智能体是否能通过摄像机看到该实体。x_{allents_i} 每一行的维度和内容取决于实体类型，具体如下：

- 对于$x_{\text{其他智能体}}$，实体是其他智能体；因此，每一行的形式都与 $x_{\text{自身}}$相同。
- 对于 $x_{\text{治疗物品}}$，实体是治疗物品，每一行都包含该物品的位置（$x,\ y$）。
- 对于 $x_{\text{箱子}}$，实体是箱子对象。每个实体行都包含：其 4 个顶点的二维位置（假设箱子以世界原点为中心）；指向箱子在世界中实际中心的平移向量；以及箱子在世界中的实际朝向。
- 对于 $x_{\text{箱子物品}}$，实体是对应于破损箱子的物品。每一行都包含：其 4 个顶点的二维位置（假设箱子以世界原点为中心）；以及箱子物品在世界中的位置。
- $x_{\text{治疗物品槽}}$ 始终包含单个实体；如果智能体库存的最后一个槽位中有治疗物品，该实体就表示该槽位。否则，其唯一一行会在相应的 $x_{\text{掩码治疗物品槽}}$中被屏蔽。
- 同样，x_{boxslot} 也只包含一个实体，表示智能体物品栏中的最后一个槽位（如果其中包含箱子物品）。否则，其对应的$x_{\text{mask}_{\text{boxslot}}}$中的唯一一行会被屏蔽。

环境的动作空间是离散的，每个动作 a 的形式如下

$$a = (a_x, a_y, a_\theta, a_{\text{atk}}, a_{\text{use}}, a_{\text{give}})$$

其中：

- a_x、a_y 和 a_θ 的值均取自 $\{-1, 0, 1\}$，分别表示智能体可施加于自身的线性力和角力的方向；它们的值会乘以一个控制所产生的力大小的常数参数
- a_{atk} 是一个二元动作，用于控制智能体在当前时间步是否尝试进行近战攻击；
- a_{use} 是一个二元动作，激活时，智能体会尝试使用其物品栏中最后拾取的物品；
- $a_{\text{给予}}$是一个二元动作，用于控制智能体是否尝试将其物品栏中最后拾取的物品给予最近的合适智能体或队友。

3.5 奖励与回合终止

单个智能体或团队的奖励函数 R 是确定性的，仅取决于状态 s 和 s'，定义如下：

$$R_i(s,s') = I_{\text{alive}}(s,s')r_{\text{alive}} + (1 - I_{\text{alive}}(s,s'))r_{\text{dead}} + n_{\text{kills}}(s,s')r_{\text{kill}} + I_{\text{death}}(s,s')r_{\text{death}}$$

其中：

- $I_{\text{存活}}(s, s')$ 在智能体存活时为 1，否则为 0；
- $I_{\text{死亡}}(s, s')$ 若智能体在从 s 到 s'的状态转移中死亡，则为 1，否则为 0；
- $n_{\text{击杀数}}(s, s')$ 表示智能体在从s 到 s'的状态转移中成功击杀的敌方智能体数量；
- $r_{\text{存活}}$ 是一个参数，用于控制存活智能体在每一步获得的奖励；
- $r_{\text{死亡}}$ 是一个参数，用于控制死亡智能体在每个时间步获得的奖励；
- $r_{\text{击杀}}$ 是每次成功击杀所获得的奖励；
- $r_{\text{死亡}}$ 是智能体死亡时所在时间步获得的奖励。

每种环境变体都可以为这四个参数指定不同的数值。

根据环境变体的不同，回合有两种可能的终止条件。第一种是所有智能体或团队都死亡时，回合终止。第二种是仅剩一个或更少的智能体或团队存活时，回合结束。结合上述四个奖励参数，这一选择决定了环境的竞争性程度。例如，考虑这样一种环境：只有 $r_{\text{alive}} = 1$，其他参数均为 0，并且所有智能体都死亡时回合终止。在这种环境中，只要安全区足够大，采取激进策略并攻击其他智能体就没有明显优势。最佳策略只是尽可能长时间地存活。另一方面，如果 r_{kill} 为正，且仅剩一个智能体或团队存活时回合结束，那么击杀对手以获取奖励就有相当大的优势。请注意，具体取决于 r_{kill} 的数值，在与对手交战以获取击杀奖励之前，尽可能长时间存活也可能是有利的。

4 实验与结果

本节详细介绍实验设置以及所有已报告实验的结果。以下段落将讨论所有实验共有的一些方面。第 4.1 节介绍策略优化方法，后续各节则呈现实验结果。

三个　　用于这些实验的环境变体之间的主要区别在于　　：

- 智能体是否被编入团队；
- 奖励机制和回合结束条件的竞争性；
- 环境的复杂度。

这里采用了两种组队模式：自由混战，以及将智能体分成两个对立团队。竞争性分为以下几种：

- 低 竞争性：$r_{\text{存活}} = 1$，其余所有奖励参数均设为 0，且只有在所有智能体都死亡时回合才结束；
- 中等 竞争性：$r_{\text{存活}} = 1$, $r_{\text{死亡}} = -1$，其余两个奖励组成部分设为 0，且只剩一个智能体存活时回合结束；
- 高 竞争性：$r_{\text{存活}} = 1$，$r_{\text{死亡}} = -1$，$r_{\text{击杀}} = 100$，$r_{\text{死亡}} = -100$。

环境的复杂性主要取决于是否存在形状随机的箱子。

除了每个智能体或团队获得的奖励外，训练期间还记录了每个经验回合中的以下变量：

- 消耗的治疗物品数量；
- 通过消耗物品放置的箱子数量；
- 每个智能体或团队造成的击杀数。

对下文所述的所有变体进行了性能测试，结果见表2。

4.1 策略优化

智能体的策略由人工神经网络模型表示。每个智能体都关联着两个网络：第一个网络通过输出动作分布来定义随机策略；第二个网络用于近似价值函数。智能体网络的架构基于全连接层和掩码自注意力机制，以适应每个智能体观察到的实体数量变化。这些模型使用近端策略优化算法 [31] 和广义优势估计 [30]进行优化。我们使用了 StableBaselines 3 中 PPO [27]的一种实现，并对其进行了修改，使其能够同时优化多个智能体。该实现的一个特点是在策略优化的每个步骤中进行提前停止，依据是对新旧策略之间 Kullback–Leibler散度的估计。在该环境中的早期实验表明，这是策略优化算法的关键环节；与提前停止相比，采用固定数量的随机梯度下降步数会导致策略遭到破坏性更新，或使学习速度大幅降低。表 1 列出了策略优化所用超参数的值。

在多智能体强化学习文献中，通常会在所有智能体之间共享智能体网络的参数 [2]。与 Baker 等人、Pinto 等人、Lowe 等人和 Foerster等人采用的方法不同，这些方法使用“全知”观测来训练智能体，而本文在训练期间也会屏蔽不可见实体。之所以做出这一选择，是因为早期实验发现，使用全知观测训练的智能体不太愿意探索周围环境以寻找物品，例如治疗物品。

用于智能体的策略网络和价值网络架构非常相似，唯一显著的区别在于最后几层。请注意，智能体的观测 x 形式如下

$$x = (x_{\text{self}}, x_{\text{zone}}, x_{\text{entities}})$$

而每个x_{entities_i} 的形式如下

$$x_{\text{entities}_i} = (x_{\text{allents}_i}, x_{\text{mask}_i})$$

将输入 x_{self} 和 x_{zone} 拼接后送入全连接层，得到非实体嵌入 z。对于每种实体类型， x_{allents_i} 中对应的行都会经过同一个全连接层，其输出维度为 N_e，并由所有实体类型共享。这样，所有实体嵌入都可以拼接成一个嵌入实体序列 E。接着，将所有实体类型的掩码 x_{mask_i} 拼接起来，得到一个指向E中实体行的掩码。随后，将 E 和该掩码送入自注意力层。自注意力层是一种注意力层，其中查询、键和值都来自同一个序列。该注意力层的输出序列

表 1：策略优化超参数 ers.

超参数	值
轨迹缓冲区大小 T	$2^{14} = 16384$
小批量大小 B	$2^9 = 512$
PPO 裁剪参数 ϵ	0.2
折扣因子 γ	0.99
GAE 参数 λ	0.95
目标 KL 散度 δ	0.01

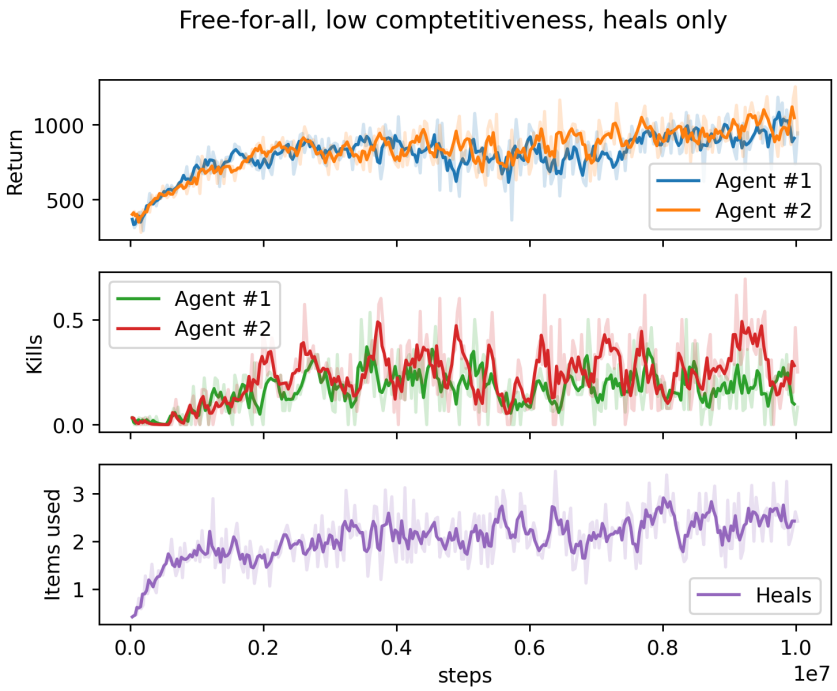


图4：自由混战变体1的结果。击杀数虽然仍低于1，但仍显著高于其他自由混战变体。其趋势是在训练初期上升，随后稳定在一个波动很大但似乎恒定的数值。

随后经过平均池化，得到一个固定长度的向量 z_{ent}，其长度不受实体数量影响。接着，将 z 和 z_{ent} 拼接后送入两个全连接层。策略网络随后设有 N_a 个全连接输出头，每个动作对应一个，且每个输出头的维度等于该动作可能取值的数量。然后使用 Softmax 激活函数，为每个动作分量输出一个动作分布。相比之下，价值函数网络只有一个全连接输出头，仅输出一个值，且不使用激活函数。所有全连接隐藏层均使用 ReLU 激活函数。

4.2 自由混战

第一组 实验涉及对环境变体的测试，其中所有智能体都互相对抗 。所有变体中的智能体数量均为2。变体如下：

1. 低竞争性的自由混战，且仅有治疗物品；
2. 中等竞争性的自由混战，且仅有治疗物品；
3. 中等竞争性的自由混战，包含治疗物品和箱子；
4. 高竞争性的自由混战，包含治疗物品和箱子；

所有变体均分配了 1000 万环境步数用于训练。图 4、5、6 和 7 展示了 4 种自由混战变体的结果。在所有变体中，奖励和治疗物品的使用数量在整个训练过程中都稳步增加。这表明所有智能体都学会了跟随安全区并收集治疗物品以延长生存时间的基本策略。为了确认这一点，我们在环境的多个实例中对所有 4 种变体中智能体学到的策略进行了目视检查，得出了以下观测：

1. 在所有变体中，智能体都会学着跟随安全区，并在治疗物品进入视线范围时收集它们。然而，在变体4中，智能体执行这一策略时更不稳定，常常在安全区外四处移动，或未能收集视线范围内的治疗物品。在其他

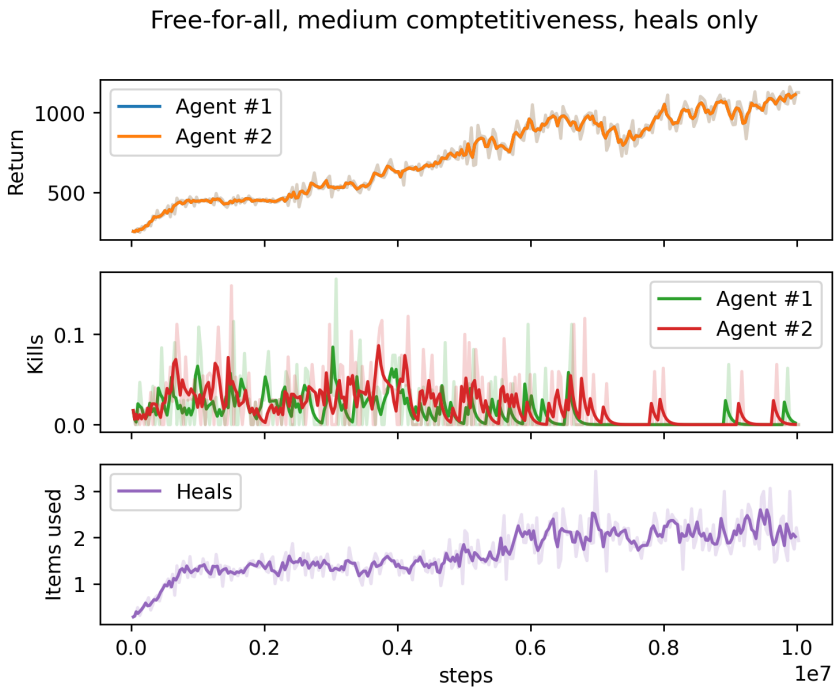


图5：自由混战变体2的结果。在这里，随着智能体学会避免伤害对手，训练后期的击杀数几乎消失。请注意，奖励机制使回报之间仅相差1，因此图中的两条回报曲线看起来重合。

变体中，由于回合往往会进行到安全区消失的阶段，智能体也学会了慌忙移动以寻找治疗物品，从而让自己多存活一会儿。

2. 在变体2和变体3中，智能体学会在对手进入攻击范围时不攻击对手，从而避免伤害对手（请注意，如果一个智能体处于另一个智能体的攻击范围内，它也总是在对方的视野锥内）。此外，它们采取的方式各不相同：在同样包含箱子的变体3中，只有当箱子处于攻击范围内时，智能体才会使用攻击动作。而在变体2中，当没有智能体处于攻击范围内时，智能体似乎会“频繁使用”攻击动作；但当对手处于攻击范围内时，它们几乎会停止使用该动作。

3. 在变体1中，由于没有明确的竞争性激励，智能体不会主动寻找对手，但一旦发现对手处于攻击范围内，几乎总会使用攻击动作。

4. 在变体4中，回合结果非常不一致，主要在三种行为之间交替：(1) 一个智能体在初始阶段击杀另一个智能体，导致奖励较低；(2) 智能体之间没有互动，最终被迫离开安全区而死亡，因此获得较高奖励；(3) 智能体因未能持续跟随安全区而死亡。

5. 在所有变体中，智能体有时会难以控制运动，偶尔会短暂地游荡到安全区外，或者因速度过快而错过治疗物品。

4.3 团队

在第二组实验中，智能体被分为两个团队，共享奖励和回合终止条件。所有变体都包含4个智能体，因此每个团队由2个智能体组成。变体如下：

1. 低竞争性的2对2模式，包含治疗物品和箱子；

2. 中等竞争性的2对2，仅限治疗物品；

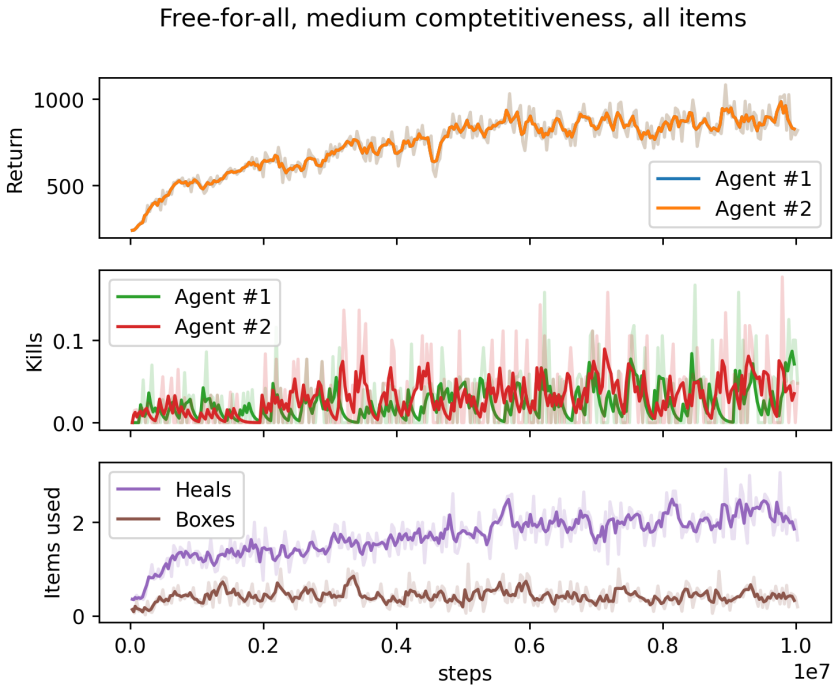


图6：自由混战变体3的结果。击杀数增长非常缓慢，其方差也是如此；不过，其值仍非常接近0。整个训练过程中，放置的箱子数量都接近0。请注意，奖励机制使回报之间仅相差1，因此图中的两条回报曲线看起来重合。

3. 中等竞争性的2对2，包含治疗物品和箱子；

4. 竞争性较高、包含治疗物品和箱子的2对2；

与自由混战变体一样，训练持续到环境步数达到1000万。结果见图8、9、10和11。除变体4外，所有这些变体的奖励趋势都是先增长，直到环境步数达到约400万。此后，两个团队的奖励均大幅下降。四种变体中，治疗物品的使用数量都呈现出与奖励趋势相同的变化。在变体1和变体4中，放置的箱子数量达到略高于1的值，不过在变体1中随后又下降到较低的值。每回合击杀数的趋势很大程度上取决于具体变体，但所有变体的值都没有接近1。对于前三种变体，我们既使用环境步数达到1000万时的参数值，也使用环境步数达到400万时较早的参数值，对策略进行了目视检查。以下是观察结果：

- 训练达到 1000 万步时，在前 3 种变体中，智能体已经学会了跟随安全区、收集并使用治疗物品，但表现不稳定。这种不稳定主要是因为智能体四处移动，经常分散在地图的不同角落。智能体不会主动寻找对手，但对手一旦进入攻击范围，它们几乎总会发起攻击。每个团队中也常有一名智能体因为站在安全区外而在回合较早阶段死亡，而它的队友则能存活更久。请注意，在这种情况下，队友中第一名成员的死亡不会直接影响团队的回合回报。

- 在 400 万步时，前 3 种变体中的智能体比在 1000 万步时更稳定地跟随安全区。此外，虽然尚不清楚这是否出于有意为之，但智能体似乎更倾向于聚集在安全区中心附近。每个团队中有队友在回合开始后的前几步因站在安全区外而死亡的情况也较少。至于 1000 万步时的策略，智能体并不会有意寻找对手，不过几乎总会对攻击范围内的对手（也可能包括箱子）使用攻击动作。

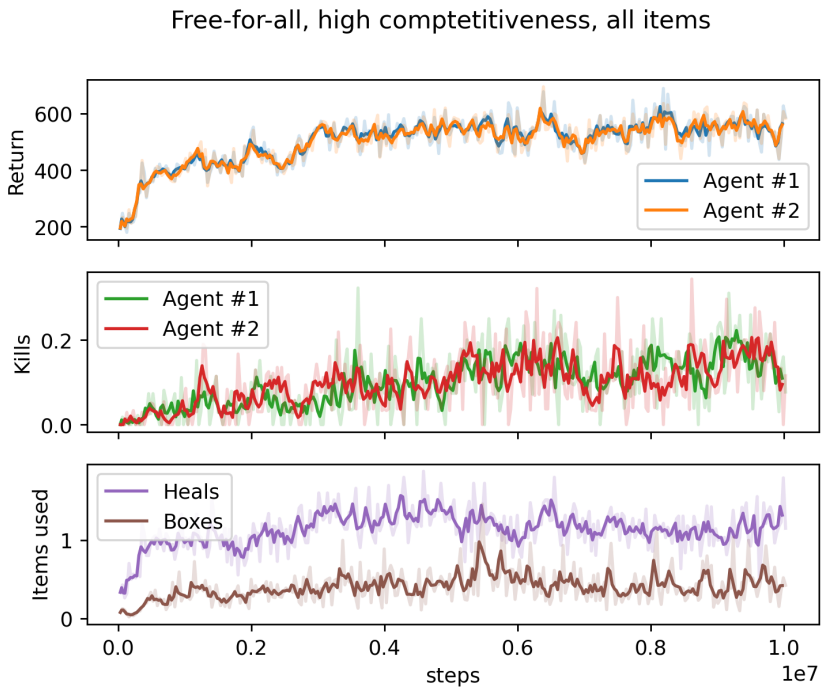


图7：自由混战变体4的结果。击杀数缓慢增加，但仍接近0，不过趋势比变体2更为稳定。放置的箱子数量也仍接近0，治疗物品数量则明显少于其他变体。

- 在变体4中，智能体跟随安全区和收集治疗物品的行为并不稳定。与其他变体一样，它们会攻击范围内的对手和箱子，但不会主动寻找对手并将其击杀。同样，这种不稳定性源于智能体偶尔会在地图上四处游荡，步进到安全区之外。

表2：环境步进函数的计算性能测试。每项结果报告的是一个回合前100步中 env.step调用 所耗时间的均值和标准差。由于此时所有智能体和物体仍在场景中，这些步通常耗时最长，因此报告的估算值偏保守。场景中物体的数量会影响耗时，这一点也得到证实：智能体较少或没有箱子的配置运行速度更快。此外，物体数量相同的配置，其结果几乎完全重合。测试在一台配备 Intel Mobile Core i7 “ Broadwell” (I7-5557U) 中央处理器的 MacBook Pro ’15 笔记本电脑上进行。

变体	每步秒数（$\mu \pm \sigma$）
自由混战变体1	$7.59 \cdot 10^{-4} \pm 1.1 \cdot 10^{-4}$
自由混战变体2	$7.58 \cdot 10^{-4} \pm 1.2 \cdot 10^{-4}$
自由混战变体3	$9.86 \cdot 10^{-4} \pm 6.28 \cdot 10^{-5}$
自由混战变体4	$1.04 \cdot 10^{-3} \pm 2.97 \cdot 10^{-5}$
2 对 2 变体 1	$1.47 \cdot 10^{-3} \pm 1.48 \cdot 10^{-4}$
2 对 2 变体 2	$1.21 \cdot 10^{-3} \pm 1.30 \cdot 10^{-4}$
2对2变体3	$1.40 \cdot 10^{-3} \pm 7.25 \cdot 10^{-5}$
2对2变体4	$1.42 \cdot 10^{-3} \pm 6.08 \cdot 10^{-5}$

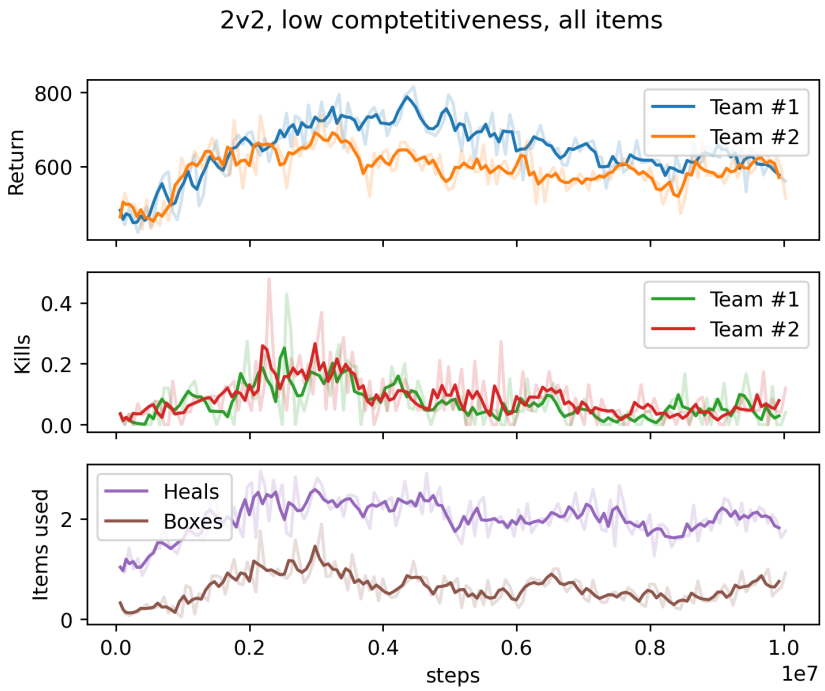


图 8：2 对 2 变体 1 的结果。放置的箱子数量和击杀数在约 300 万步时达到峰值，而峰值回报则在约 400 万步时才出现。尤其是击杀数，仍接近 0，但显著高于变体 2 和 3。使用的治疗物品数量起初有所增加，随后在约 400 万步后缓慢下降。

5 讨论

在所有实验中，智能体都学会了环境的基本规则：待在安全区内，并收集和使用 Heal 物品。不过，智能体遵循这一策略的稳定程度差异很大，具体取决于变体。前三种自由混战变体是智能体能够可靠遵循这一策略的唯一变体。其原因似乎取决于具体变体。

对于训练后期回报下降的 2 对 2 变体，一种可能的原因是某种形式的灾难性遗忘。不过，对习得策略的观测似乎表明并非如此。事实上，策略在训练后期表现不稳定，似乎与智能体更倾向于分散到地图各处有关，即使这有时意味着无视安全区。这可能是因为智能体没有学会避免彼此伤害（如自由混战变体 2 和 3 中那样），反而学会了认为靠近其他智能体很危险。事实上，从理论上说，由于智能体攻击造成的伤害远大于安全区造成的伤害，因此短期内躲避其他智能体比待在安全区内更重要。除了偶然因素之外，组队模式也可能是阻碍智能体避免攻击对手的另一个原因。对于团队型变体，由于奖励由队友共享，一名队友死亡不会直接影响奖励，因此对表现不佳的队友会更宽容。这与 Baker 等人的情况恰恰相反；在他们的研究中，“躲猫猫”奖励机制会直接惩罚表现不佳的躲藏者。

因此，这些实验展现出的仅有的两种涌现行为是：

1. 在自由混战变体 2 和 3 中出现的“和平主义”智能体；以及

2。智能体在地图上四处扩散以避免被杀，这种行为出现在其他大多数变体中 ts.

在这两种情况下，这都意味着智能体学会了主动避免与其他智能体发生战斗交互。第二种情况下，这也表明，对这些配置进行更多环境

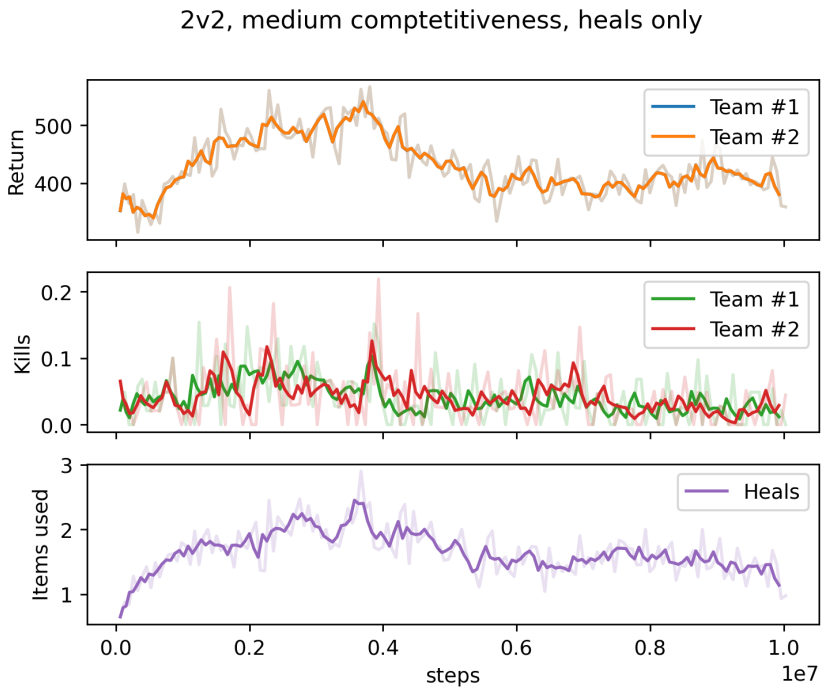


图 9：2 对 2 变体 2 的结果。同样，回合回报在约 400 万回合时达到峰值。击杀数存在波动，但从 200 万步到 400 万步期间数值较高，随后缓慢下降，直至训练结束。尽管如此，击杀数始终更接近 0 而非 1。治疗物品的使用情况与奖励趋势一致。请注意，由于奖励机制使回报之间仅相差 1，因此图中的两条回报曲线似乎重合。

步数的训练可能不会产生其他有趣的涌现交互。相比之下，在第一种情况下，由于智能体停止攻击，但不会主动彼此逃离，如果在回合的最后几个阶段进行更多训练，战斗交互仍可能出现；在这些阶段中，争夺安全区内的位置至关重要。

报告的结果和上述考量还凸显了所考察环境变体中的以下缺陷：

1. 智能体从不留存治疗物品，也不会将其交给其他智能体；相反，它们一旦拾取物品就会立即使用。这是因为不这样做并无明显优势：生命值满时使用治疗物品，与生命值为 1 时使用效果相同。

2. 团队奖励机制导致表现不佳的队友不会影响团队奖励，至少在每个回合的早期阶段是如此。这是因为所有队友存活下来的唯一优势，是有可能在战斗中组队对付对手，或交换治疗物品。由于缺陷 1 的存在，后者不太可能发生，而战斗交互又十分稀少，因此这种奖励机制产生的团队，其智能体表现的一致性不如自由混战配置。

3. 击杀数和死亡所对应的奖励幅度并未带来高水平的战斗交互，反而使高竞争性变体更接近中等竞争性配置。事实上，由于这两种配置下的回合都会在仅剩一个智能体或团队存活时结束，因此智能体最好不要在回合早期参与战斗，因为在战斗中存活下来，相较于击杀奖励，可能会带来高得多的奖励。

4. 智能体运动的动态控制可能限制过多，无法实现精确移动，导致整体表现不够稳定。

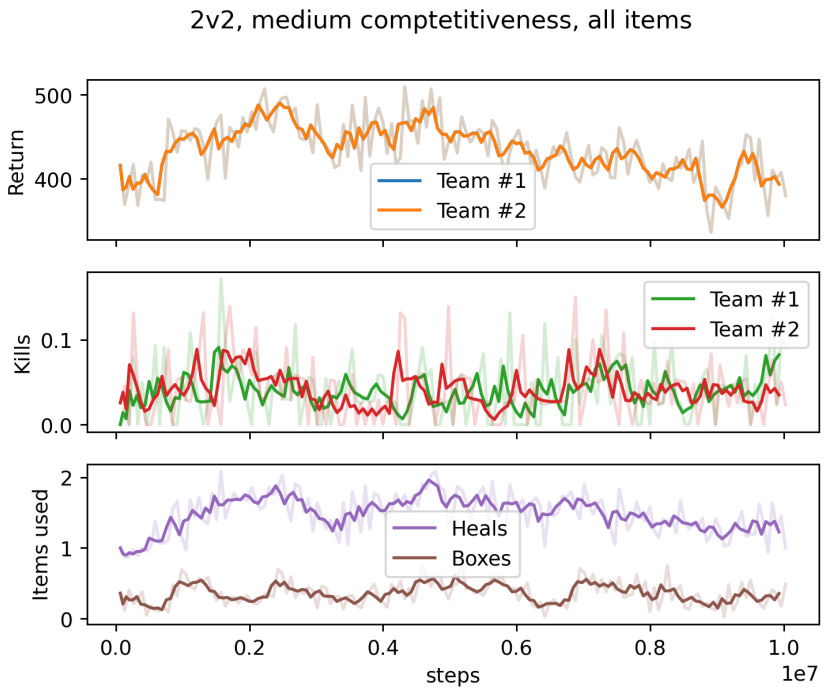


图10：2对2变体3的结果。其趋势不如前两个变体明显，但仍显示回报在训练初期增加，随后在训练后期下降。击杀数存在噪声，但仍低于前两个变体且接近0，没有呈现出明显趋势。请注意，由于奖励机制导致回报之间仅相差1，因此图中的两条回报曲线看起来似乎重合。

6 结论与未来工作

这项工作的两项主要贡献是：

- 开发一个模块化且高效的二维多智能体环境框架，并在其基础上实现具体的基于生存的环境；
- 对所开发的变体开展实验，获得一些基本的涌现行为，并确定改进所测试环境变体的具体方向。

未来工作应在开展进一步实验之前，先解决上述缺陷。缺陷1可以通过引入生命值最大值轻松解决。缺陷2应通过为团队型变体设计不同的奖励机制来解决，从而降低对散漫队友的容忍度；或者，也可以向环境中加入促进团队协作的机制。在这方面，解决缺陷1或许有所帮助，因为这将提高基于物品栏的机制的价值，例如分享物品，或击杀对手以收集其持有的物品。缺陷3可以通过取消击杀奖励、更多地关注间接竞争型变体，或直接提高击杀所获奖励来解决。

考虑到上述实验所采用的训练规模显著小于相关工作（例如 Baker 等人），未来的研究也可以考虑专注于较少的环境变体。这样就能为每种变体分配更多训练步数，同时避免需要更多计算资源。

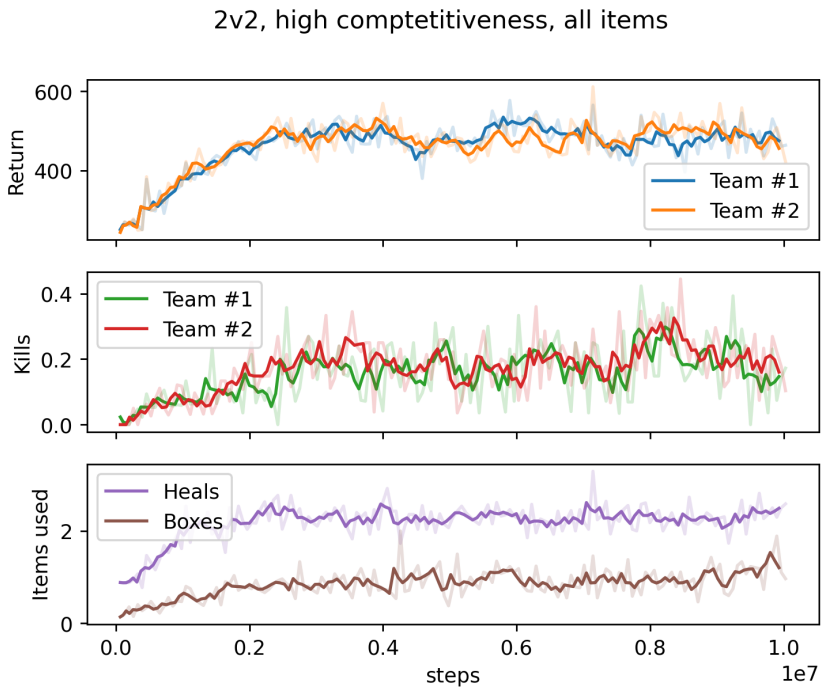


图11：2对2变体4的结果。训练过程中，击杀数呈缓慢上升趋势，其值与变体1的量级相近。放置箱子的数量呈现非常缓慢但明显的上升趋势，而治疗物品的使用数量在约200万步时趋于稳定。

引用

[1] Achiam, J. 和 Sastry, S.（2017年）。用于深度强化学习的基于惊喜的内在动机。[2] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B. 和 Mordatch, I.（2019年）。多智能体自动课程中涌现的工具使用。[3] Bansal, T., Pachocki, J., Sidor, S., Sutskever, I. 和 Mordatch, I.（2018年）。通过多智能体竞争实现涌现复杂性。载于国际学习表征会议。[4] Bellemare, M. G., Dabney, W. 和 Munos, R.（2017年）。强化学习的分布式视角。[5] Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D. 和 Munos, R.（2016年）。统一基于计数的探索与内在动机。[6] Brant, J. C. 和 Stanley, K. O.（2017年）。最小标准协同进化：一种开放式搜索的新方法。载于遗传与进化计算会议论文集，GECCO '17，第67–74页，纽约，纽约州，美国。计算机协会。[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. 和 Zaremba, W.（2016年）。OpenAI Gym。*arXiv*预印本 *arXiv:1606.01540*。[8] Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T. 和 Efros, A. A.（2018a）。好奇心驱动学习的大规模研究。[9] Burda, Y., Edwards, H., Storkey, A. 和 Klimov, O.（2018b）。通过随机网络蒸馏进行探索。[10] Denil, M., Agrawal, P., Kulkarni, T. D., Erez, T., Battaglia, P. 和 de Freitas, N.（2016年）。通过深度强化学习学习进行物理实验。

[11] Dharna, A., Togelius, J. 和 Soros, L. B.（2020）。游戏关卡与游戏智能体的协同生成。[12] Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O. 和 Clune, J.（2019）。Go-Explore：解决困难探索问题的新方法。[13] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N. 和 Whiteson, S.（2017）。反事实多智能体策略梯度。[14] Fujimoto, S., van Hoof, H. 和 Meger, D.（2018）。解决演员-评论家方法中的函数逼近误差。[15] Haarnoja, T., Zhou, A., Abbeel, P. 和 Levine, S.（2018a）。软演员-评论家：使用随机演员的离策略最大熵深度强化学习。[16] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P. 和 Levine, S.（2018b）。软演员-评论家算法及其应用。[17] Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaˇneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K. 和 Graepel, T.（2019）。通过基于种群的强化学习，在3D多人游戏中达到人类水平的表现。《科学》，364(6443)：859–865。[18] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. 和 Wierstra, D.（2015）。使用深度强化学习进行连续控制。[19] Liu, S., Lever, G., Heess, N., Merel, J., Tunyasuvunakool, S. 和 Graepel, T.（2019）。通过竞争涌现出协调能力。载于国际学习表征会议。[20] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P. 和 Mordatch, I.（2017）。用于混合合作竞争环境的多智能体演员-评论家。[21] Meyer, J.-A. 和 Wilson, S. W.（1991）。在模型构建神经控制器中实现好奇心与无聊感的一种可能方法，第222–227页。[22] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. 和 Riedmiller, M.（2013）。使用深度强化学习玩Atari游戏。[23] Ostrovski, G., Bellemare, M. G., Oord, A. v. d. 和 Munos, R.（2017）。基于神经密度模型的计数式探索。[24] Packard, N., Bedau, M. A., Channon, A., Ikegami, T., Rasmussen, S., Stanley, K. O. 和 Taylor, T.（2019）。开放式进化概述：开放式进化II特刊编者导言。《人工生命》，25(2)：93–103。[25] Pathak, D., Agrawal, P., Efros, A. A. 和 Darrell, T.（2017）。通过自监督预测进行好奇心驱动的探索。[26] Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W. 和 Abbeel, P.（2017）。用于基于图像的机器人学习的非对称演员-评论家。[27] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M. 和 Dormann, N.（2021）。Stable-Baselines3：可靠的强化学习实现。《机器学习研究杂志》，22(268)：1–8。[28] Salimans, T., Ho, J., Chen, X., Sidor, S. 和 Sutskever, I.（2017）。进化策略：强化学习的一种可扩展替代方案。[29] Schulman, J., Levine, S., Moritz, P., Jordan, M. I. 和 Abbeel, P.（2015a）。信赖域策略优化。[30] Schulman, J., Moritz, P., Levine, S., Jordan, M. 和 Abbeel, P.（2015b）。使用广义优势估计进行高维连续控制。

[31] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. 和 Klimov, O.（2017）。近端策略优化算法。[32] Shinners, P.（2011）。PyGame。http://pygame.org/。[33] Sims, K.（1994）。进化虚拟生物。载于第*21*届计算机图形学与交互技术年度会议论文集，第15–22页。ACM。[34] Stadie, B. C., Levine, S. 和 Abbeel, P.（2015）。利用深度预测模型激励强化学习中的探索。[35] Sukhbaatar, S., Szlam, A. 和 Fergus, R.（2016）。通过反向传播学习多智能体通信。[36] Tesauro, G.（1995）。时序差分学习与TD-Gammon。*ACM*通讯，38(3)：58–68。[37] van Hasselt, H., Guez, A. 和 Silver, D.（2015）。使用双重Q学习进行深度强化学习。[38] Wang, R., Lehman, J., Clune, J. 和 Stanley, K. O.（2019）。成对开放式开拓者（POET）：不断生成日益复杂、多样的学习环境及其解决方案。[39] Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J. 和 Stanley, K. O.（2020）。增强型POET：通过无限制地发明学习挑战及其解决方案实现开放式强化学习。

A 环境实现

该环境使用 Python 编程语言实现。它提供了 Gym [7] 接口，并针对多个智能体进行了略微修改。此外，它还支持 VecEnv 接口，以便与 Stable Baslines 3 库 [27], 配合使用，使多智能体环境表现为向量化的单智能体环境。为了使实现具备模块化和可扩展性，我们开发了一个抽象框架，详见 A.1 节。由于 Box2D 提供了高效且紧凑的接口，因此使用 Box2D 库的 Python绑定作为底层物理模拟器。环境状态的可选渲染使用 PyGame [32]作为后端。

A.1 框架

环境框架以一个核心对象仿真为基础，可通过添加由模块组成的<body>群组</body>来扩展，每个群组包含一个或多个模块。仿真对象只是对底层物理引擎的封装，以支持添加群组。每个群组代表一组共享相同模块集的物体，因此它们的行为相似。具体语义由模块自身实现；模块可以且通常会维护特定于各个物体的数据。这种架构既能将行为相似的物体归为一组以保持效率，同时

特征	**每步秒数（$\mu \pm \sigma$）**
5 个智能体（无摄像头）	$4.77 \cdot 10^{-4} \pm 6.70 \cdot 10^{-5}$
5 个智能体（无摄像头），10 个治疗物品	$4.94 \cdot 10^{-4} \pm 8.50 \cdot 10^{-5}$
5 个智能体（无摄像头），10 个治疗物品，10 个箱子	$5.07 \cdot 10^{-4} \pm 7.09 \cdot 10^{-5}$
5 个智能体	$5.72 \cdot 10^{-4} \pm 9.36 \cdot 10^{-5}$
5 个智能体，10 个治疗物品，10 个箱子	$7.24 \cdot 10^{-4} \pm 9.21 \cdot 10^{-5}$
10 个智能体，10 个治疗物品，20 个箱子	$1.64 \cdot 10^{-3} \pm 1.47 \cdot 10^{-4}$
10 个智能体（无摄像头），10 个治疗物品，20 个箱子	$8.03 \cdot 10^{-4} \pm 1.47 \cdot 10^{-4}$

表 3：环境步进函数的计算性能测试。每项结果报告了具有给定特征的仿真对象每一步所用时间的均值和标准差。测试在配备 Intel Mobile Core i7 “Broadwell” (I7-5557U) 中央处理器的 MacBook Pro ’15 笔记本电脑上进行。

又不会牺牲可扩展性或模块化。此外，模块被设计为回调函数集合，由群组对象和仿真对象自动调用，因此无需修改任何现有代码即可添加新模块。本节其余部分将介绍仿真、群组和模块接口的具体内容；框架所包含的核心模块见 A.2 节。本文实验中具体使用的语义由外部模块实现，相关细节将在后续章节中说明。

仿真仅有两种操作：重置和步进。前者会重置物理引擎和所有群组的状态，后者则执行一次仿真时间步。根据仿真实例和物理引擎的不同，仿真中的一个时间步可能对应物理模拟器中的多个时间步。仿真对象的状态是底层物理模拟器的状态，加上对其群组的引用列表。

群组 对象提供以下操作：

- 生成，即在群组中生成新的物体；
- 消失，即使群组中现有的物体消失；
- 获取模块，即返回群组模块中某个指定模块类的实例。

因此，群组对象的状态由模块列表、活动物体列表，以及可能存在的一个对其所属仿真对象的引用组成。

一个模块对象可以实现以下零个或多个回调函数：

- 重置后回调，在仿真对象重置时调用；
- 步进前回调，在执行每个仿真步进前调用；
- 步进后回调，在执行每个仿真步进后调用；
- 生成后回调，在模块所属群组中生成一个或多个物体后调用；
- 移除前回调，在模块所属群组中的一个或多个物体即将消失时调用。

模块对象的状态完全由其实现决定。

A.2 内置模块

本节概述环境框架本身提供的通用模块。这些模块主要用于对物体施加力、以各种方式观察世界，以及跟踪有用的信息。

IndexBodies 模块会为其群组中生成的所有物体分配唯一的整数索引。它通过维护一个存储物体对象引用的动态数组来实现这一点，因此每个物体的索引就是它在数组中的索引。物体消失时，为了保持索引唯一，不会从数组中删除其条目，而是将其替换为空引用。这也使该模块能够快速检测某个索引对应的物体是否已消失。

TrackDeaths 模块会在每次仿真步骤中，跟踪群组内所有已消失的物体。LogDeaths 模块与之类似，但当群组中的物体消失时，还会向控制台打印一条消息。

Cameras 模块会为群组中的所有物体安装“摄像头”。群组中的每个物体都关联一个以其位置为起点的视野锥。在给定时间步，该模块会为群组中的每个观察物体存储世界中所有对其可见的物体。要被观察物体看见，某个物体必须位于观察者的视野锥内，且其与观察物体之间的视线不能被其他物体遮挡。

DynamicMotors 模块提供对群组中物体的动态控制。在每个时间步，都可以对群组中的每个物体施加特定的线性力和角力。请注意，这些控制针对各个物体分别设定。

A.3 通用模块

一些外部模块实现了本研究特有的环境语义。其中一些仅用于单个身体组，而另一些用途更广，可用于多个群组。在多个群组中使用的三个主要模块是 ResetSpawns、Health 和 Item。智能体身体组专用的主要模块包括 SafeZone、Inventory 及相关模块、Melee 和 TwoTeams。物品身体组使用抽象 Item 模块的不同实现。治疗物品身体组主要使用 Heal 物品模块。箱子身体组主要使用 RandomizedBoxShapes 和 OwnedObject 模块，而箱子物品身体组则基于 OwnedObjectItem 物品模块。本节其余部分将更详细地介绍各模块为核心环境增添的语义。

ResetSpawns 仅实现了一个重置后回调，用于使用给定的生成器对象在群组中生成物体。生成器对象只负责提供物体的生成位置。由于它可以由不同的 ResetSpawns 模块共享（这些模块可能属于不同的身体组），因此，确定生成位置的逻辑可以同时考虑所有物体，即使它们属于不同的群组。此处使用的生成器只需获取一个矩形区域，将其划分为单元格，再为每个物体随机分配一个未使用的单元格进行生成。

Health 模块用于为群组中的所有物体赋予一个整数生命值，并向其他模块提供对这些物体造成伤害和进行治疗的接口。此接口还支持追踪伤害和治疗的原因、添加使物体免疫某些原因的过滤器，以及注册死亡时回调；该回调还会提供物体的死亡原因。

抽象 Item 模块仅定义了所有实现物品的模块都应遵循的接口。这样一来，例如，实现库存或其他物品相关机制的模块就可以不必关心所涉及物品的具体类型。若模块实现了使用操作（即物品被消耗使用时调用的操作），它便是 Item 模块。抽象 Item 模块还提供了具体的丢弃操作：该操作会在其身体组中生成一个物品，并将给定对象作为该物品的专属数据；物品被消耗时，这些数据随后会传递给使用操作。

A.4 智能体专用模块

SafeZone 模块实现了上述安全区机制。具体而言，它通过维护简单的整数计数器作为状态，记录安全区所处的阶段以及距离下一阶段的剩余步数。假设群组包含 Health 模块，伤害将施加于群组中的物体。

智能体库存由 Inventory 模块实现。对于群组中的每个身体，该模块都维护一个固定长度的物品槽位数组，并提供以下操作：

- 拾取，使身体拾取地面上的物品；
- 使用，尝试使身体使用其物品栏最后一个槽位中的物品；
- 丢弃，尝试使身体将其物品栏最后一个槽位中的物品丢弃到地面上；
- 全部丢弃，即使物体将其物品栏中的所有物品丢弃到地面上；
- 给予，即尝试使一个身体将其最后一个槽位中的物品给予另一个拥有物品栏的身体。

物品栏不会自动执行这些动作，而是将它们提供给其他模块。事实上，为了实现与物品栏相关的智能体机制，在模块 AutoPickup、UseLast、GiveLast 和 DeathDrop 中使用了这些操作。AutoPickup 会在智能体接触物品对象时，使其群组中的所有智能体拾取物品。UseLast 和 GiveLast 提供控制，使智能体使用或给予其物品栏最后一个槽位中的物品。最后，DeathDrop 会在其群组中的物体即将消失时，使它们丢弃物品栏中的所有物品；这些物品会以最后位置为中心，略微散落在周围的圆圈内。

智能体的攻击动作由 Melee 模块实现。该模块为每个智能体提供一个二元控制量。每一步中，如果某个智能体的控制量处于开启状态，该模块就会尝试寻找一个处于攻击范围内且拥有 Health 模块的身体。如果找到合适的目标，就会对其造成伤害，并将发起攻击的身体标记为造成伤害的来源。

智能体的团队划分由 Teams 模块实现。Teams 会跟踪各团队的所有成员，并提供获取某个智能体的队友或对手的操作。此外，该模块还提供操作，用于检测团队中的所有智能体是否都已死亡。除了提供团队记录管理外，它还会将 Melee 攻击造成伤害的来源设为团队，并使同一团队中的智能体免疫队友的攻击。

A.5 治疗物品与箱子模块

唯一专门用于治疗物品的模块是 Heal。它是一个具体的 Item 模块，其使用操作只会根据 Heal 模块的初始化方式，为使用者增加一定量的生命值。

专门用于可破坏箱子的模块，主要负责实现这样的逻辑：将被破坏的物体转换为可用于重新生成这些物体的物品。Object 和 ObjectItem 模块负责在这两种实体所属的身体组中实现这一行为：完整物体属于带有 Object 模块的群组，而被破坏的物体则属于带有 ObjectItem 模块的群组。这两个模块相互维护引用，并保存有关物体形状及其他属性的数据。当其群组中的物体即将消失时，Object 模块会利用这些数据生成合适的物品。相反，当智能体收集并使用该物品时，ObjectItem 会利用记录的数据，在 Object 模块所属的群组中生成一个物体。OwnedObject 和 OwnedObjectItem 实现了一种物体变体：只有最初破坏某个物体的实体，才能在放置该物体后再次将其破坏。所有权可以归属于单个身体，也可以归属于团队等其他实体。

A.6 渲染框架

我们还使用 PyGame 库 [32] 作为后端实现了一个可选的渲染框架。其设计仿照了上文所述的可扩展性和模块化特征，主要区别在于无需显式地对身体组进行建模。单个渲染模块称为 $View$，而用于连接 View 与渲染后端的对象则是 $Canvas$。每个 View 都实现一个绘制回调函数，供其所属的 Canvas 使用。另一方面，Canvas 对象会维护与渲染后端相关的状态以及一个 View 列表。它还实现了渲染和绘制操作。具体而言，渲染操作包括清除和渲染，可由 Canvas 的使用者在每次需要帧图像时调用。清除操作只是使用背景颜色清空屏幕。渲染操作会调用每个已注册 View 的绘制回调函数，然后调用后端将渲染好的帧实际显示在屏幕上。另一方面，绘制操作并非供 Canvas 的使用者调用，而是可在每个 View 的实现中使用。它们提供绘制圆形、线段和多边形等基本形状，以及绘制世界中的物体的功能。