17 Arena：多智能体强化学习平台

在这一章节，我们将介绍一个名为 Arena (Song et al., 2019) 的用于研究多智能体强化学习（Multi-Agent Reinforcement Learning，MARL）的项目。我们提供了一些上手经验来使用 Arena 工具包构建游戏，包括一个单智能体游戏和一个简单的双智能体游戏，并采用不同的奖励机制。Arena 中的奖励机制是一种定义多智能体间社会结构的方式，包括不可学习的（Non-Learnable）、独立的（Isolated）、竞争的（Competitive）、合作的（Collaborative）和混合型的（Mixed）社会关系。不同的奖励机制可以在一个游戏场景中用于同一个层次性结构上，不同的层次性结构上也可以用不同的奖励机制，配合对物理单元的从个体到群体的结构性表示，可以用来全面地描述多智能体系统的复杂关系。此外，我们也展示了在 Arena 中使用基准库的过程，它提供了许多已实现的多智能体强化学习算法来作为基准。通过这个项目，我们希望给读者提供一个有用的工具，来研究在定制化游戏场景中使用多智能体强化学习算法的表现。

Arena 是一个在 Unity 上对多体智能学习进行评估的通用平台。它使用多样化逻辑和表示方式来构建学习环境，并对多智能体复杂的社会关系进行简单的配置。Arena 也包含对最先进深度多智能体强化学习算法基准的实现，可以帮助读者快速验证所建立的环境。总体来说，Arena 是一个帮助读者快速创造和构建包含多智能体社会关系的定制化游戏环境的工具，用以探索多智能体问题。Arena 注重于第一人称或第三人称动作类游戏，借助于 Unity 极好的渲染效果来实现 3D 仿真环境。而其他像最近由 DeepMind 发布的开源项目 OpenSpiel，则专注于多智能体棋牌类游戏。

Arena 中有两个主要的模块：（1）开发工具包（the Building Toolkit），可以用来快速构建有定制特征的多智能体环境；（2）基准库（the Baselines），可以用 MARL 算法来测试所搭建的环境。我们将从构建 Arena 中的环境开始。

17.1 安装

Unity ML-agents 工具包是使用 Arena 的前提，需要在使用 Arena 之前将其安装。Arena 完整的安装过程遵循开发工具包和基准库各自的官方网站。

注意，如果你想在没有图形用户界面（比如 X-Server）的远程服务器上运行或者你无法访问 X-Server，那么你就需要根据 17.3.1 节中或 Arena 官网上的指示来设置虚拟显示。

安装好之后，我们可以发现在 Arena 文件夹下的 `Arena-BuildingToolkit/Assets/ArenaSDK/GameSet/`文件中，有几十个已构建好的或连续或离散动作空间的游戏场景。它们是预先制作好的，作为使用 Arena 的例子，你可以阅读所有这些游戏的脚本来更好地理解 Arena 是如何工作的。所有的游戏和抽象层共用同一个 Unity 项目。每一个游戏都在一个独立的文件夹中，游戏名即文件夹命名。`ArenaSDK` 文件夹存放了所有的抽象层和共享代码、实体和功能。整体代码风格与 Unity ML-agents 工具包尽可能一致。

17.2 用 Arena 开发游戏

我们将使用 Arena 开发工具包内提供的许多现成的实体和多智能体功能，来展示一个多智能体游戏的构建过程，它不需要很多代码。在你开始之前，我们希望你已经有了关于 Unity 使用的基本知识。因此，推荐你先完成官网上的 roll-a-ball 教程来学习关于 Unity 的一些基本概念。

为了使用 Arena，运行 Unity，选择打开项目，选择克隆或下载的“Arena-BuildingToolkit”文件。第一次打开的过程可能会花费一定时间。

我们可以看到 Arena 文件夹下几十个建好的游戏。它们是预先设计作为 Arena 用例的，你可以阅读这些游戏的脚本来进一步理解 Arena 是如何工作的。我们将在下面小节中提供搭建这些游戏的基本指导。

图 17.1 Arena 文件中提供已构建的游戏

17.2.1　简单的单玩家游戏

我们从构建一个基本的单玩家游戏开始：

- 创建一个文件夹来存放游戏。在这部分中，我们为单玩家游戏创建名为“1P”的文件夹。
- 在左边的“Hierarchy”窗口中，我们删除原来的 **Main Camera** 和 **Directional Light**。将 Arena 文件夹 `Assets/ArenaSDK/SharedPrefabs` 下预制的 **GlobalManager**（如图 17.2 所示）拖入左边的“Hierarchy”窗口，如图 17.3 所示。注意到，这些预制物体是 Unity 中公用的模块，可以通过简单的拖曳操作来使用任何提前定制好的物体。Arena 中的 **GlobalManager** 管理整个游戏，因此其他组成部分需要依附在它下面。

图 17.2　Arena 中的预制模块（见彩插）

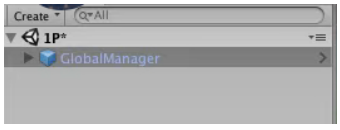

图 17.3　将 Arena 预制模块中的 **GlobalManager** 拖到当前游戏的“Hierarchy”窗口中

- 下面我们需要放置一个智能体的运动场所，我们找到 Arena 中名为 **PlayGroundWithDeadWalls** 的预制模块，然后将它附于 **GlobalManager** 的子节点 **World** 上。**GlobalManager** 也有另一个子节点 **TopDownCamera** 用于提供一个游戏的全局视野。这一步在图 17.4 中有所展示。
- 与上面类似，我们需要从 Arena 预制模块中选择一个 **BasicAgent** 并将其附于 **GlobalManager**，如图 17.5 所示。从而我们现在得到一个在场地上的智能体，我们可以手动拖拽智能体到一个合适的位置，如图 17.6 所示。x 轴、y 轴和 z 轴的位置和旋转角将在智能体的 **Transform** 属性中展示。
- 为了让智能体正常工作，我们需要设定游戏参数，如图 17.7 所示。这里我们只需要改变 **GlobalManager** 中的 **Living Condition Based On Child Nodes**。**Living Condition** 被选择为 **At Least Specific Number Living** 而 **At Least Specific Number Living** 值被设为 1。由于我们在这个游戏中只有一个智能体，上面的设置保证了在智能体数量小于 1 的任何情况下，这

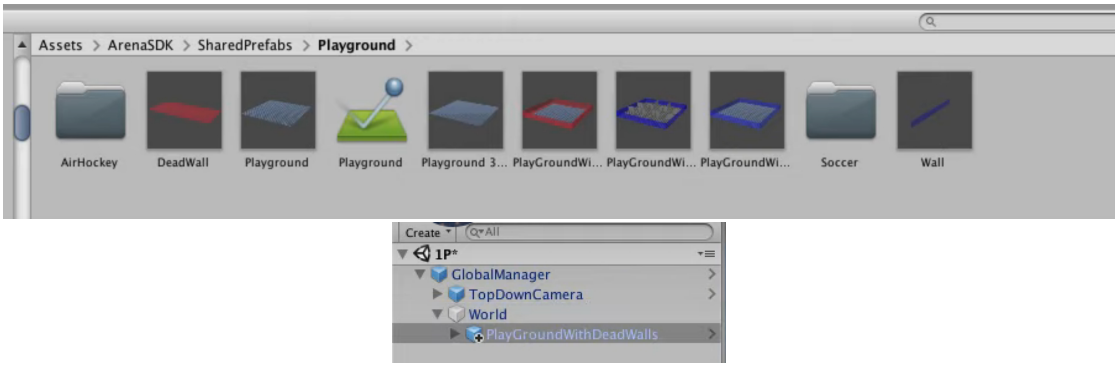

图 17.4 选择 Arena 预制模块中的一个运动场（Playground），并将它附于 **GlobalManager** 的子节点上（见彩插）

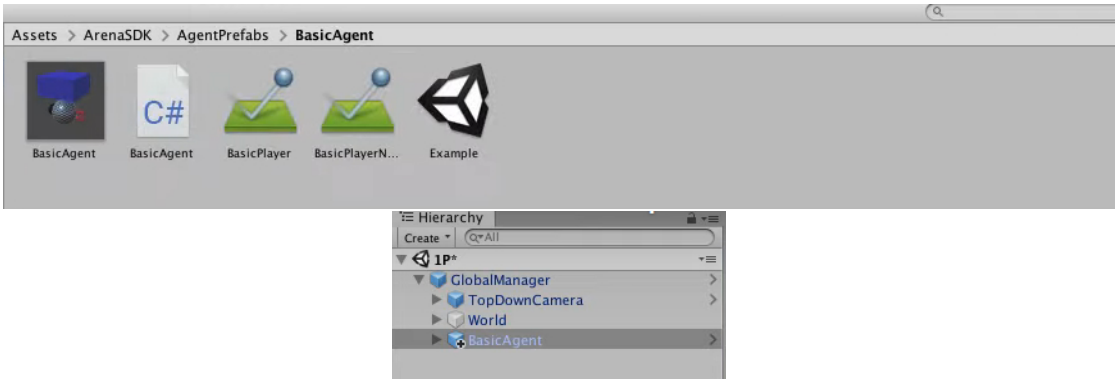

图 17.5 选择并将一个 Arena 预制模块中的 **BasicAgent** 附于 **GlobalManager** 的子节点上（见彩插）

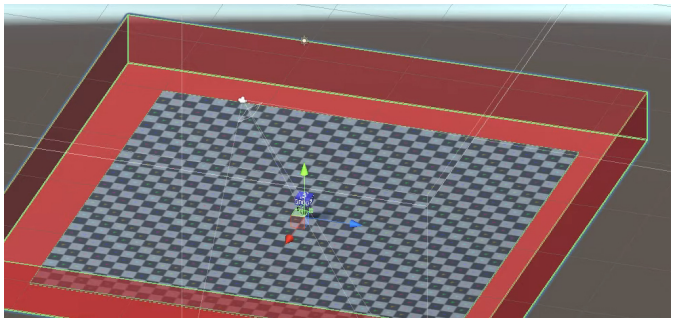

图 17.6 单个智能体在场地上的场景（见彩插）

个游戏片段（Episode）就会结束并重启。现在我们需要按下 **Play** 按钮来开始游戏并用键盘上的“W，A，S，D”操作智能体移动。由于场地的一边是“Dead Walls”，智能体无论何

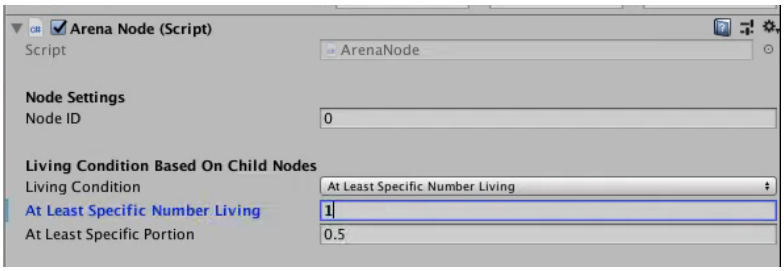

图 17.7　进行单玩家游戏设置

时碰触它都会结束生命并且重新开始游戏。使用 **BasicAgent** 时也会有很多其他性质，包括不同的 **Actions Settings**、**Reward Functions**（用于强化学习）等。你可以调试它们（在当前这个游戏中只有 **Actions Settings** 是有效的）来熟悉 Arena 开发工具包。

17.2.2　简单的使用奖励机制的双玩家游戏

在这一小节，我们将介绍如何在游戏场景中按照社交树（Social Tree）来部署多于一个智能体的游戏。

- 首先，让我们从上面的单玩家游戏开始。如果我们选择 **GlobalManager** 或 **BasicAgent**，那么我们就会发现这些对象都有一个叫作 **Arena Node (Script)** 的脚本，分别如图 17.8 和图 17.9 所示，这个基本概念可以用来帮助理解 Arena 游戏中定义的社会关系。关于 **Arena Node** 的描述将在本小节中提供。

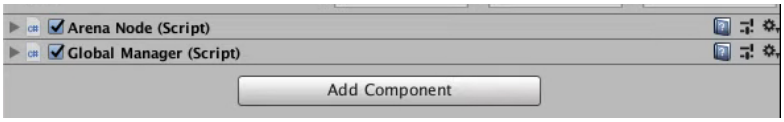

图 17.8　存在于 **GlobalManager** 中的 **Arena Node (Script)**

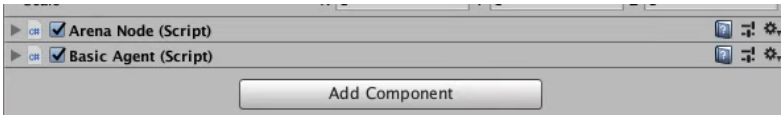

图 17.9　存在于 **BasicAgent** 中的 **Arena Node (Script)**

- 我们选择之前构建的 **BasicAgent** 并将它在左边“Hierarchy”窗口中通过 Ctrl+C 和 Ctrl+V 复制，如图 17.10 所示。现在 **Global Manager** 之下有两个 **Arena Node**s，因此我们需要将两个 **BasicAgent**s 中的任一个设置 **Node ID** 为 1 而非 0 来辨别它们（见图 17.11）。智能体

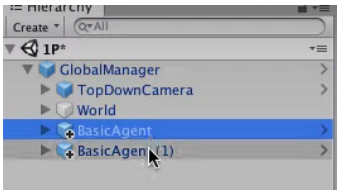

图 17.10 在 **GlobalManager** 下复制 **BasicAgent**

在场景中的位置可以被移动到一个合适的位置，从而将它们区分开，这是因为复制后的两个智能体会有相同的位置。

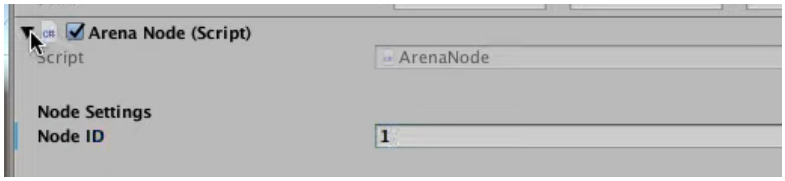

图 17.11 当 **GlobalManager** 下有多个节点时需要改变节点 ID，使得它们各不相同

- 下面我们选择 **GlobalManager** 、**Arena Node (Script)** 来设置游戏的奖励函数，如图 17.12 所示。我们单击 **Is Reward Ranking**，这是一个在 **GlobalManager** 下用于设置智能体竞争性奖励函数的属性。我们也将 **Ranking Win Type** 选择为 **Survive**，这意味着最后结束生命（存活到最后）的智能体会得到一个正的奖励。如果你选择 **Depart**，奖励将会给首先结束生命的智能体。我们也需要取消 **Is Reward Distance**（这是根据智能体到目标的距离给出密集奖励的函数）。上面是 Arena 中内置的不同奖励机制，通过或竞争或合作（有时二者都有）的形式。不同的游戏中会有不同的奖励设置来表示不同的社会关系结构。你可以对不同的游戏使用不同的奖励设置。举例来说，如果你想用智能体到目标的距离来解决一个类似到达目标类的任务，那么可以设置一个基于距离的密集奖励来实现，通过单击勾选 **Is Reward Distance**，以及将一个目标物体拖到 **Target** 空格中来设置。
- 我们需要在 **GlobalManager** 的 **Living Condition Based On Child Nodes** 下设置 **At Least Specific Number Living** 为 2，如图 17.13 所示，从而只有当至少两个智能体存活时，游戏才会继续；否则游戏将终止并重新开始。现在我们单击 **Play** 按钮，游戏应该正常运行，只要一个智能体结束了生命，游戏就会结束，而且奖励或惩罚就会施加给智能体，如 **Console** 中所显示的奖惩记录，见图 17.14。
- 下面我们会使得游戏更加复杂，我们想要两队各自有两个智能体来互相竞争。首先，我们在“Hierarchy”窗口创建一个空的对象并将其命名为“2 Player Team”。随后我们将 **Arena Node** 脚本附加到它上面，如图 17.15 所示。

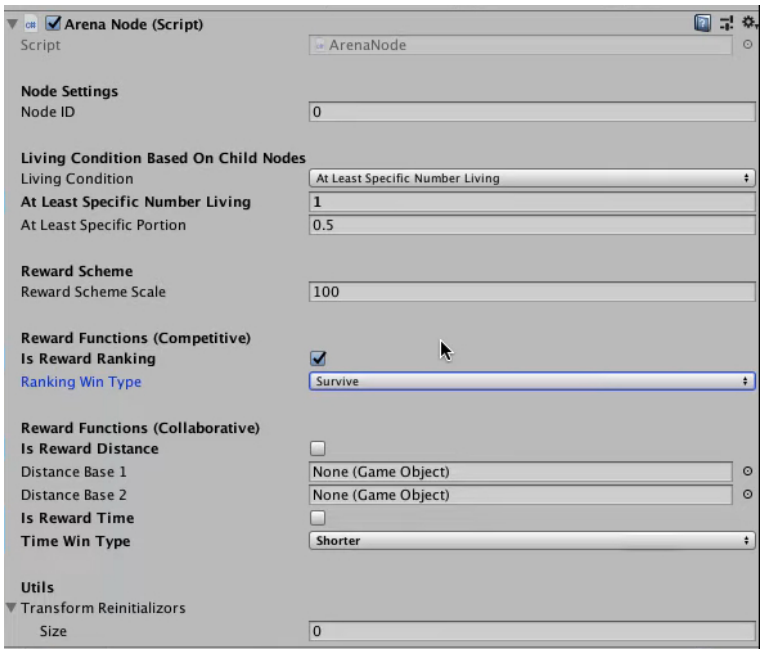

图 17.12　在 **GlobalManager** 下设置奖励函数

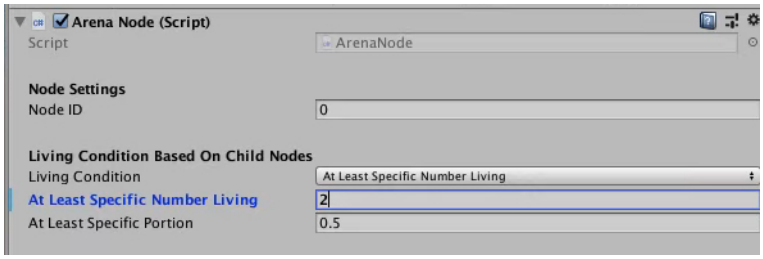

图 17.13　在 **GlobalManager** 中设置最小存活智能体数量

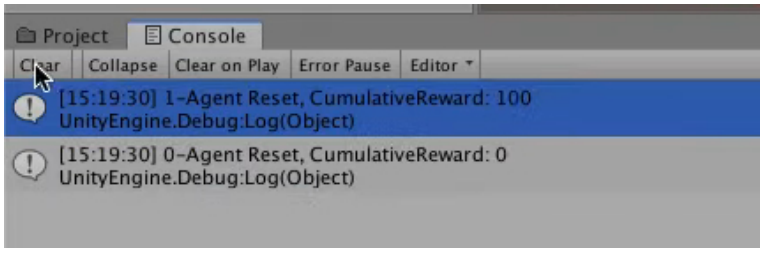

图 17.14　给每个智能体的奖励显示在 **Console** 中

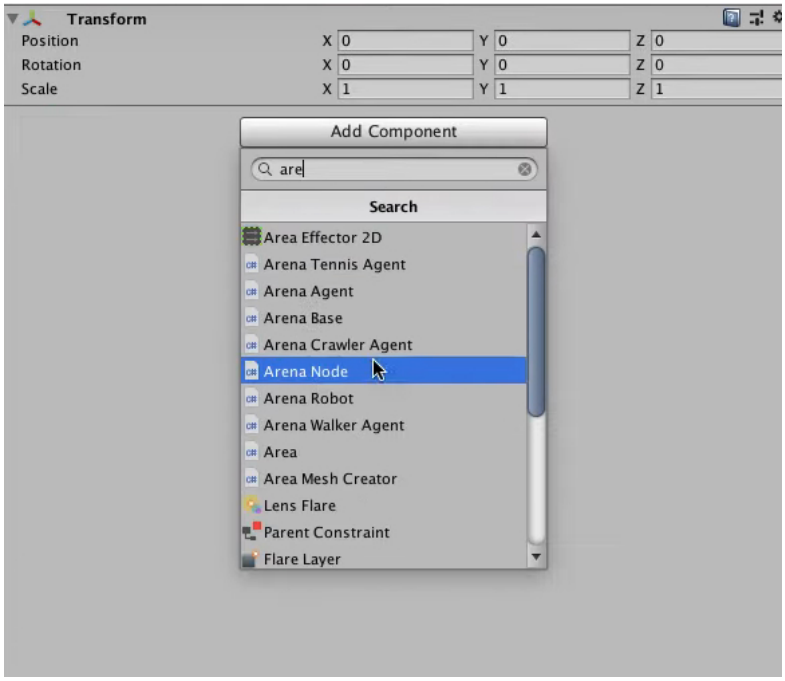

图 17.15 将 **Arena Node** 脚本附加到队伍（Team）对象上

- 现在我们将两个之前的 **BasicAgent** 拖到新创建的队伍对象 **2 Player Team** 中。随后我们复制 **2 Player Team**，将第二组对象的 **Node ID** 从 0 改为 1。现在我们有队伍和智能体的结构如图 17.16 所示。如果我们现在单击 **Play** 按钮，我们将会看到两个各有两个智能体的队伍在场景中，如图 17.17 所示。

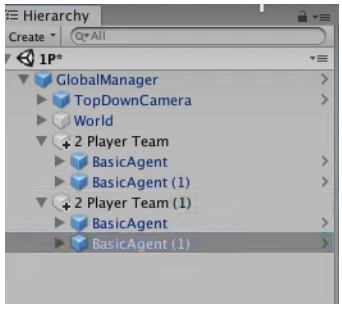

图 17.16 Arena 的 **GlobalManager** 下两个队伍（Teams）各有两个智能体（Agents）的层次性结构

- 由于 **GlobalManager** 的 **At Least Specific Number Living** 被设为 2，任何队伍生命的结束都会造成游戏结束。由于 **2 Player Team** 的 **At Least Specific Number Living** 默认为 1，只有

当同一个队伍中两个智能体都结束生命时才会造成队伍的生命结束。我们也可以设置不同的游戏逻辑，如果我们设置 **2 Player Team** 的 **Living Condition** 为 **All Living**，那么一个队伍中任何智能体结束生命都会导致队伍的生命结束，从而结束整个游戏。从以上来看，通过 **GlobalManager->Team->Agent** 的社交树结构，Arena 基本可以通过定义生存和奖励机制，使用 **Arena Node** 支持任意类型的社会关系。

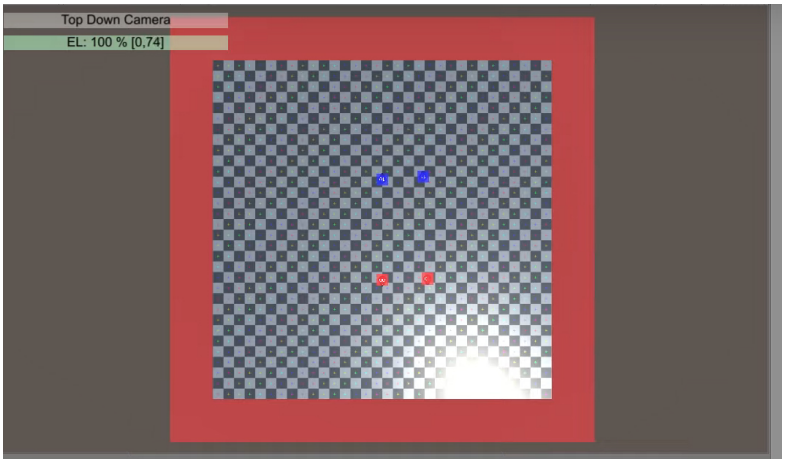

图 17.17　两个各有两个智能体的队伍在游戏中

17.2.3　高级设置

奖励机制

为了构建复杂的社会关系，在 Arena 中有 5 个基本的多智能体奖励机制（Basic Multi-Agent Reward Schemes，BMaRSs）来定义社交树上每个节点的不同社交范式，包括：**不可学习的**（Non-Learnable，NL）、**独立的**（Isolated，IS）、**竞争的**（Competitive，CP）、**合作的**（Collaborative，CL）、**竞争和合作混合型的**（Competitive and Collaborative Mixed，CC）。具体来说，每个 BMaRS 是一个对奖励函数的限制，因此它与能产生某种具体社交范式的一批奖励函数相关。对于每个 BMaRS，Arena 提供了多个可以立即使用的奖励函数（稀疏或密集），简化了有复杂社会关系的游戏构建过程。除提供奖励函数外，Arena 也提供了对定制化奖励函数的验证选项，从而可以将编写的奖励函数置于一种 BMaRS 下而产生相应的具体社交范式。我们将详细讨论这五种不同的奖励机制。

首先我们需要给出一些预备知识。我们考虑基本强化学习中定义的一个马尔可夫（Markov）游戏，它包括多个智能体 $x \in \mathcal{X}$、一个有限的全局状态空间 $s_t \in \mathcal{S}$、一个对每个智能体 x 的有限的动作空间 $a_{x,t} \in \mathcal{A}_x$ 和一个对每个智能体 x 的有限步奖励空间 $r_{x,t} \in \mathbb{R}$。至于环境，它包括一个转移函数 $g : \mathcal{S} \times \{\mathcal{A}_x : x \in \mathcal{X}\} \to \mathcal{S}$，这是一个有随机性的（由于 Unity 模拟器的随机性）函数

$s_{t+1} \sim g\big(s_{t+1}|(s_t, \{a_{x,t} : x \in \mathcal{X}\})\big)$ 和一个对每个智能体的奖励函数 $f_x : \mathcal{S} \times \big\{\mathcal{A}_x : x \in \mathcal{X}\big\} \to \mathbb{R}$。这是一个确定性函数 $r_{x,t+1} = f_x\big(s_t, \{a_{x,t} : x \in \mathcal{X}\}\big)$，以及一个联合奖励函数 $f = \{f_x : x \in \mathcal{X}\}$ 和对每个智能体 x 在联合奖励函数 f 下的片段奖励 $R_x^f = \sum_{t=1}^{\mathrm{T}} r_{x,t}$。对于智能体来说，Arena 考虑以下情况，即它观察 $s_{x,t} \in \mathcal{S}_x$，其中 $\mathcal{S}_x$ 包括全局状态空间 $\mathcal{S}$ 的部分信息。因此，策略 $\pi_x : \mathcal{S}_x \to \mathcal{A}_x$ 是一个随机性函数 $a_{x,t} \sim \pi_x(s_{x,t})$。除此之外，Arena 考虑智能体 x 能够从一个策略集合 Π_x 中采取一个策略 π_x。Arena 假定所有采样操作的随机种子是 k，这是从整个种子空间 $\mathcal{K}$ 中采样得到的。

不同的 BMaRSs 定义使用的基本概念包括，智能体 $\{x : x \in \mathcal{X}\}$、策略 $\{\pi_x : \Pi_x\}$、智能体奖励 $\{R_x^f : x \in \mathcal{X}\}$ 和联合奖励函数 $\mathcal{F} = \{f : \cdot\}$，其中智能体总体为 $\mathcal{X}$。Arena 中五种不同的 BMaRSs 通过以下方式定义：

1. 不可学习的 BMaRSs（$\mathcal{F}^{\mathrm{NL}}$）是一个联合奖励函数集合 f，如下：

$$\mathcal{F}^{\mathrm{NL}} = \big\{f : \forall k \in \mathcal{K}, \forall x \in \mathcal{X}, \forall \pi_x \in \Pi_x, \partial R_x^f / \partial \pi_x = 0\big\}, \tag{17.1}$$

其中 0 是与定义 π_x 的参数空间同样大小和形状的零矩阵。直观上，$\mathcal{F}^{\mathrm{NL}}$ 意味着对于任何智能体 $x \in \mathcal{X}$ 改进其策略 π_x 都是无法优化 R_x^f 的。

2. 独立的 BMaRSs（$\mathcal{F}^{\mathrm{IS}}$）是一个联合奖励函数的集合如下：

$$\mathcal{F}^{\mathrm{IS}} = \big\{f : f \notin \mathcal{F}^{\mathrm{NL}} \text{ and } \forall k \in \mathcal{K}, \forall x \in \mathcal{X}, \forall x' \in \mathcal{X} \setminus \{x\}, \forall \pi_x \in \Pi_x, \forall \pi_{x'} \in \Pi_{x'}, \frac{\partial R_x^f}{\partial \pi_{x'}} = 0\big\}, \tag{17.2}$$

其中“\”是集合的差。直观上，$\mathcal{F}^{\mathrm{IS}}$ 意味着智能体 $x \in \mathcal{X}$ 接受的片段内奖励 R_x^f 与任何其他智能体 $x' \in \mathcal{X} \setminus \{x\}$ 采取的策略 $\pi_{x'}$ 无关。$\mathcal{F}^{\mathrm{IS}}$ 的 f 中的奖励函数 f_x 在其他多智能体方法 (Bansal et al., 2018; Hendtlass, 2004; Jaderberg et al., 2018) 中通常被称为内部奖励函数（Internal Reward Functions），意味着除了施加到群体层面的奖励函数（比如赢输），还有指引学习过程去取得群体层面奖励的奖励函数。群体层面奖励可能很稀疏而难以学习，但这些内部奖励可以更频繁地获取，即更加密集 (Heess et al., 2017; Singh et al., 2009, 2010) 。$\mathcal{F}^{\mathrm{IS}}$ 在比如当智能体是一个机器人需要连续控制施加在关节上的力的时候变得更加切实可行，这意味着基本的动作技巧（比如运动）需要在生成群体智能前被学习到。因此，Arena 在 $\mathcal{F}^{\mathrm{IS}}$ 中提供了 f 来应对：能量损耗、施加较大力的惩罚、保持稳定速度的激励和朝向目标的移动距离等。

3. 竞争的 BMaRSs（$\mathcal{F}^{\mathrm{CP}}$）是受文献 (Cai et al., 2011) 启发的方式，定义为

$$\mathcal{F}^{\mathrm{CP}} = \Bigg\{f : f \notin \mathcal{F}^{\mathrm{NL}} \cup \mathcal{F}^{\mathrm{IS}} \text{ and } \forall k \in \mathcal{K}, \forall x \in \mathcal{X}, \forall \pi_x \in \Pi_x, \forall \pi_{x'} \in \Pi_{x'}, \frac{\partial \int_{x' \in \mathcal{X}} R_{x'}^f \mathrm{d}x'}{\partial \pi_x} = 0\Bigg\}, \tag{17.3}$$

上式直观上意味着对于任何智能体 $x \in \mathcal{X}$，采用任何可能的策略 $\pi_x \in \Pi_x$，所有智能体在片段内奖励的求和是不变的。如果片段长度为 1，它表示典型的多玩家零和游戏 (Cai et al., 2011)。

关于 $\mathcal{F}^{\text{CP}}$ 中 f 的有用例子为（1）智能体需要为有限的资源斗争，而这些资源在片段结束后通常会耗尽，智能体会为它所得到的资源受到奖励；（2）斗争一直到结束生命，奖励根据生命结束的顺序来给出（奖励也可以基于相反的顺序，从而离开游戏的一方首先接受最高的奖励，比如在一些扑克游戏中，首先打出所有牌的一方获胜）。标准形式（Normal-Form）游戏 (Myerson, 2013) 中的*剪刀石头布*（*Rock, Paper, and Scissors*）和 (Balduzzi et al., 2019) 中*循环游戏*（*Cyclic Game*）都是 $\mathcal{F}^{\text{CP}}$ 的特殊情况。

4. 合作的 BMaRSs（$\mathcal{F}^{CL}$）是由文献 (Cai et al., 2011) 启发的方式，定义为

$$\mathcal{F}^{CL} = \Big\{ f : f \notin \mathcal{F}^{\text{NL}} \cup \mathcal{F}^{\text{IS}} \text{ and } \forall k \in \mathcal{K}, \forall x \in \mathcal{X}, \forall x' \in \mathcal{X} \setminus \{x\}, \forall \pi_x \in \Pi_x, \\ \forall \pi_{x'} \in \Pi_{x'}, \frac{\partial R^f_{x'}}{\partial R^f_x} \geqslant 0 \Big\}, \tag{17.4}$$

上式直观上意味着对任何一对智能体 (x', x) 都没有利益冲突（$\partial R^f_{x'} / \partial R^f_x < 0$）。除此之外，由于 $f \notin \mathcal{F}^{\text{NL}} \cup \mathcal{F}^{\text{IS}}$，至少有一对智能体 (x, x') 使得 $\partial R^f_{x'} / \partial R^f_x > 0$。该式意味着这对智能体有共同利益，从而对智能体 x 其 R^f_x 的提高也会造成智能体 x' 的 $R^f_{x'}$ 提高。最常见的关于 $\mathcal{F}^{CL}$ 中 f 的例子是对于所有 $x \in \mathcal{X}$ 的 f_x 都是相等的，比如，一个物体的移动距离可以由多个智能体的共同努力来推动，或者一个群体的存活时长（只要群体内有一个个体是存活的，群体就是存活的）。因此，Arena 在 $\mathcal{F}^{CL}$ 中提供了 f 来应对：队伍存活时间（正值或负值，因为一些游戏需要队伍尽可能久地存活，而其他一些游戏需要队伍尽可能早地消失，比如扑克中的纸牌）等。

5. 竞争和合作混合型的 BMaRSs（$\mathcal{F}^{CC}$）定义为任何以上四种之外的情况。

$$\mathcal{F}^{CC} = \big\{ f : f \notin \mathcal{F}^{\text{NL}} \cup \mathcal{F}^{\text{IS}} \cup \mathcal{F}^{\text{CP}} \cup \mathcal{F}^{CL} \big\}, \tag{17.5}$$

首先，式 (17.3) 中的 $\partial \int_{x' \in \mathcal{X}} R^f_{x'} \mathrm{d}x' / \partial \pi_x = 0$ 可以写为 $\int_{x' \in \mathcal{X}} \partial R^f_{x'} / \partial R^f_x \mathrm{d}x' = 0$（证明在这里不提供，可以参考原文），这是式 (17.3) 的另一种表示。考虑式 (17.3) 中的 $\mathcal{F}^{\text{CP}}$ 和式 (17.5) 中的 $\mathcal{F}^{CL}$，对 $\mathcal{F}^{CC}$ 的一个直观的解释是，存在 $\partial R^f_{x'} / \partial R^f_x < 0$ 的情形，即智能体在这时是竞争的。但是对整体利益的导数 $\int_{x' \in \mathcal{X}} \partial R^f_{x'} / \partial R^f_x \mathrm{d}x'$ 不总是为 0。因此，整体利益可以用具体的策略来最大化，即智能体在这时是合作的。

除了在每个 BMaRS 提供了几个实际的 f，Arena 也对每个 BMaRS 提供了一个验证选项，即可定制 f 并使用这个验证选项来确保编写的 f 属于一个具体的 BMaRS。

上面的内容提供了关于如何使用不同类别奖励函数来定义社会关系的理论。此外，奖励函数应当根据上面定义的类别来实现预期的群体中的社会关系。实践中，奖励函数有一些具体形式，如我们在之前小节中提到的。Arena 框架通常在 **GlobalManager** 的 **Arena Node** 中定义了

Collaborative 和 **Competitive** 的奖励函数，而 **Isolated** 奖励函数定义在像 **BasicAgent** 的智能体的 **Arena Node** 中。

这里是一个便于理解社会树关系的例子，这个树中每个 **Arena Node** 使用了不同的 BMaRs，如图 17.18[1] 所示。奖励机制被指定到各个 **Arena Node** 来定义它的子节点的社会关系。图 17.18(a) 中的图形用户界面（Graphical User Interface，GUI）定义了图 17.18(b) 中的树结构，用来表示一个有四个智能体的群体。这个树结构可以通过在图 17.18(a) 的 GUI 中拖曳、复制或删除来进行简单设置。在这个例子中，每个智能体有个体层面的 BMaRS。智能体是一个机器蚂蚁，而其个体级别的 BMaRSs 是 $\mathcal{F}^{\mathrm{IS}}$，具体来说，ant-motion 的选项使得学习朝向基本的运动技巧，比如向前移等进行，如图 17.18(c) 所示。每两个智能体构成一个队伍（一个智能体或队伍的集合），而这两个智能体有队伍层面的 BMaRSs。在这个例子中，两个机器蚂蚁互相合作来推动盒子前进，如图 17.18(d) 所示。因此，队伍层面的 BMaRSs 是 $\mathcal{F}^{CL}$，具体来说，是推动盒子的距离。在两个队伍之上，Arena 有全局的 BMaRSs。在这个例子中，两个队伍被设定为有一场关于哪个队伍先将盒子推向目标点的竞赛，如图 17.18(e) 所示。因此，全局的 BMaRSs 是 $\mathcal{F}^{\mathrm{CP}}$，具体来说，是将盒子推到目标的先后次序。应用到每个智能体的最终奖励函数是以上三个层次的 BMaRSs 的加权求和。我们也可以想象如何来定义一个超过三个层次的社会树，其中小的队伍组成大的队伍，在每个节点定义的 BMaRSs 给出更加复杂和结构化的社会关系。在定义了社会树并在每个节点使用了 BMaRSs 之后，环境便可以使用了。其抽象层可以解决其他问题，比如，为窗口中的每个智能体分配视图、添加队伍颜色、展示智能体 ID 并生成一个从上到下的视野等。

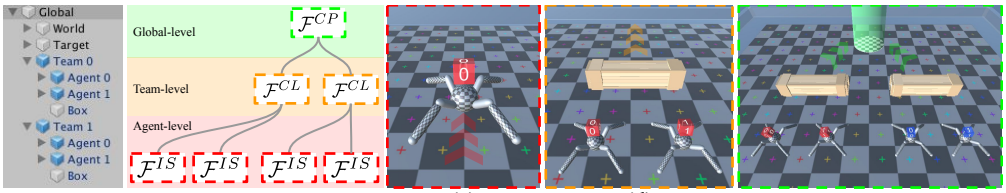

图 17.18　在 Arena 对每个 **Arena Node** 使用不同 BMaRs 定义的社会树（见彩插）

此外，我们可以简单拓展上述框架到其他常见社会关系，如图 17.19[2] 所示。

更多预制智能体

除了之前的 **BasicAgent**，Arena 也有其他更高级的预制智能体可以直接使用，如图 17.20 所示。其他智能体的使用基本与 **BasicAgent** 类似，通过拖曳并将它附于 **GlobalManager** 之下。唯一的不同在于动作空间，你需要改变相应的控制大脑（Brain）来控制不同的智能体。举例来说，

[1]图片来源：Song, Yuhang, et al. "Arena: A General Evaluation Platform and Building Toolkit for Multi-Agent Intelligence." arXiv preprint arXiv:1905.08085 (2019)。

[2]图片来源：Song, Yuhang, et al. "Arena: A General Evaluation Platform and Building Toolkit for Multi-Agent Intelligence." arXiv preprint arXiv:1905.08085 (2019)。

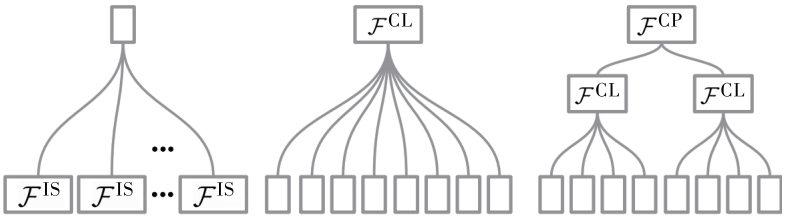

图 17.19　Arena 框架下定义的常见社会关系

图 17.20　Arena 中不同的预制智能体

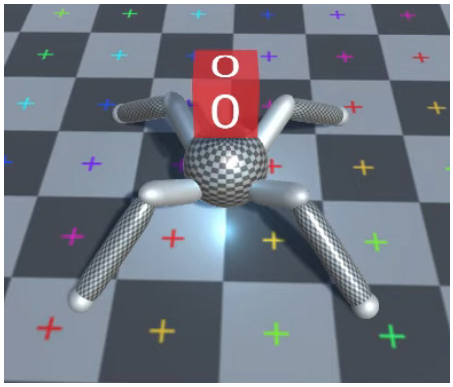

图 17.21　场景中的 **ArenaCrawlerAgent**（见彩插）

预制智能体中的 **ArenaCrawlerAgent** 如图 17.21 所示，它有连续的动作空间来控制关节的动作值。为了恰当地使用这个智能体，我们需要改变 **ArenaCrawlerAgent** 大脑为图 17.22 所示的 **ArenaCrawlerPlayerContinuous (PlayerBrain)**。随后这个游戏可以导出并用作一般的游戏来使用。

17.2.4　导出二进制游戏

当你在 Unity 中的玩家模式下测试了游戏之后，确保游戏设置没有任何问题，就可以将游戏导出为一个独立的二进制文件，并用它与 Python 脚本训练 MARL 算法。这一小节展示了如何导

出游戏。

- 首先，我们需要将大脑的类型从 **PlayerBrain** 改为一个相应的 **LearningBrain**（同样类型），**PlayerBrain** 被用于通过用户键盘操作来控制游戏智能体，而 **LearningBrain** 可以用学习算法来直接控制。如图 17.23 所示，对于这个游戏，我们改变 **GeneralPlayerDiscrete (PlayerBrain)** 为图 17.24 中的 **GeneralLearnerDiscrete (LearningBrain)**。我们也需要取消勾选 **Debugging** 来减少训练中的输出信息。

图 17.22　为 **ArenaCrawlerAgent** 更改大脑

图 17.23　玩家模式下原先的控制大脑类型

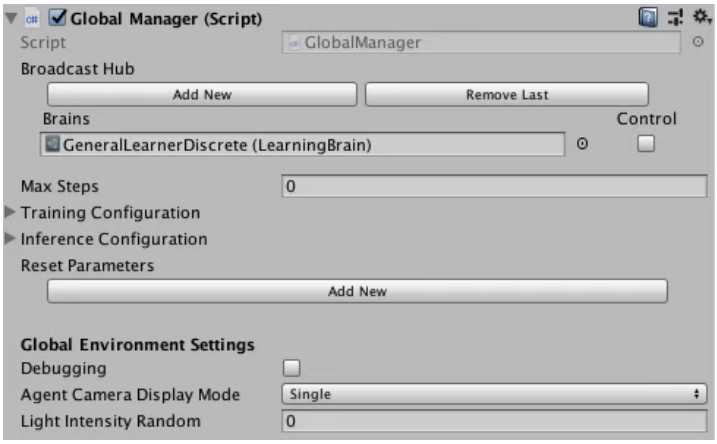

图 17.24　更改控制大脑类型为 **LearningBrain** 来导出训练游戏

- 为了导出游戏，我们选择 File->Build Settings，相应得到一个如图 17.25 所示的窗口。通过这个窗口，我们可以设置 **Target Platform** 和 **Architecture**。
- 我们也需要单击 **Player Settings** 来检查其他设置，如图 17.26 所示。一个需要注意的点是：**Display Resolution Dialog** 需要设为 **Disabled** 来正常工作。随后我们回到之前的窗口并单击 **Build**，这样就可以在创建游戏之后得到其二进制文件。

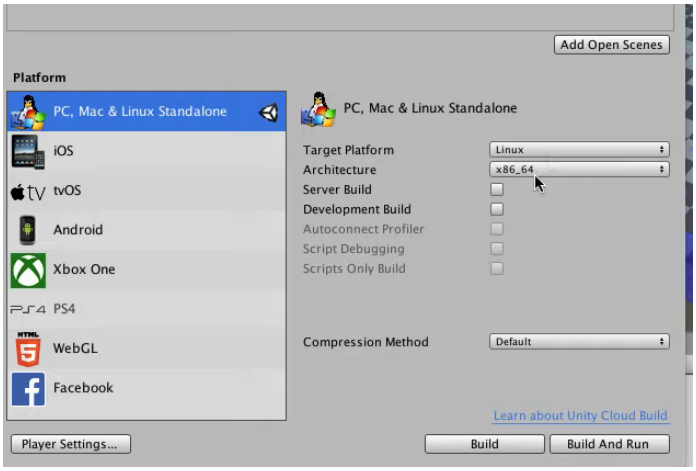

图 17.25　检查创建游戏的设置

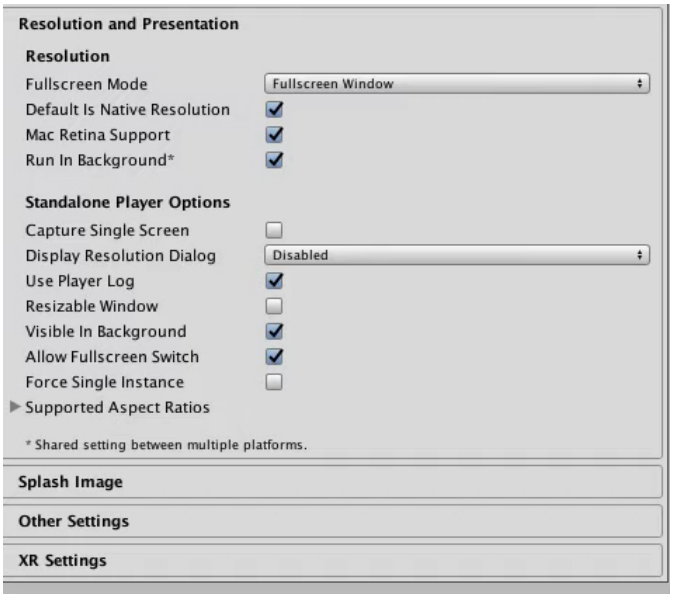

图 17.26　设置游戏导出的窗口

17.3 MARL 训练

有了用 Arena 构建并导出的独立（Standalone）游戏，我们可以设置训练过程来研究多智能体强化学习（Multi-Agent Reinforcement Learning，MARL）中的各种问题。

在开始训练之前，我们需要先配置系统。由于 MARL 一般需要大量的计算，我们通常需要用一个服务器来应对训练过程。Arena 环境的基本设置遵循 17.1 节中的内容。如果你在服务器上不能正常使用 X-Server，那么可以遵循以下部分内容来设置虚拟显示，否则可以直接跳过该部分到训练小节。

17.3.1 设置 X-Server

使用虚拟显示的基本设置如下：

```
# 安装 Xorg
sudo apt-get update
sudo apt-get install -y xserver-xorg mesa-utils
sudo nvidia-xconfig -a --use-display-device=None --virtual=1280x1024

# 获得 BusID 信息
nvidia-xconfig --query-gpu-info

# 添加 BusID 信息到你的/etc/X11/xorg.conf 文件
sudo sed -i 's/ BoardName    "GeForce GTX TITAN X"/ BoardName    "GeForce GTX TITAN X"\n
    BusID       "0:30:0"/g' /etc/X11/xorg.conf

# 从/etc/X11/xorg.conf 文件中移除小节"Files"
# 并且移除包含小节"Files" 和 EndSection 的两行
sudo vim /etc/X11/xorg.conf

# 为 Ubuntu 下载和安装最新的 Nvidia 驱动器
wget
    http://download.nvidia.com/XFree86/Linux-x86_64/390.87/NVIDIA-Linux-x86_64-390.87.run
sudo /bin/bash ./NVIDIA-Linux-x86_64-390.87.run --accept-license --no-questions
    --ui=none

# 禁用 Nouveau，因为它会使 Nvidia 驱动器崩溃
sudo echo 'blacklist nouveau' | sudo tee -a /etc/modprobe.d/blacklist.conf
sudo echo 'options nouveau modeset=0' | sudo tee -a /etc/modprobe.d/blacklist.conf
```

```
sudo echo options nouveau modeset=0 | sudo tee -a /etc/modprobe.d/nouveau-kms.conf
sudo update-initramfs -u

sudo reboot now
```

用以下三种方式（不同的方式可能在不同的 Linux 版本上运行）之一关闭 Xorg：

```
# 方式 1：运行以下命令并运行这个命令的输出
ps aux | grep -ie Xorg | awk '{print "sudo kill -9 " $2}'
# 方式 2：运行以下命令
sudo killall Xorg
# 方式 3：运行以下命令
sudo init 3
```

用该命令开启虚拟显示：

```
sudo ls
sudo /usr/bin/X :0 &
```

你应当可以看到虚拟显示正常启动，并输出以下内容：

```
X.Org X Server 1.19.5
Release Date: 2017-10-12
X Protocol Version 11, Revision 0
Build Operating System: Linux 4.4.0-97-generic x86_64 Ubuntu
Current Operating System: Linux W5 4.13.0-46-generic 51-Ubuntu SMP Tue Jun 12 12:36:29
    UTC 2018 x86_64
Kernel command line: BOOT_IMAGE=/boot/vmlinuz-4.13.0-46-generic.efi.signed
    root=UUID=5fdb5e18-f8ee-4762-a53b-e58d2b663df1 ro quiet splash nomodeset acpi=noirq
    thermal.off=1 vt.handoff=7
Build Date: 15 October 2017 05:51:19PM
xorg-server 2:1.19.5-0ubuntu2 (For technical support please see
    http://www.ubuntu.com/support)
Current version of pixman: 0.34.0
   Before reporting problems, check http://wiki.x.org
   to make sure that you have the latest version.
Markers: (--) probed, (**) from config file, (==) default setting,
   (++) from command line, (!!) notice, (II) informational,
   (WW) warning, (EE) error, (NI) not implemented, (??) unknown.
```

```
(==) Log file: "/var/log/Xorg.0.log", Time: Fri Jun 14 01:18:40 2019
(==) Using config file: "/etc/X11/xorg.conf"
(==) Using system config directory "/usr/share/X11/xorg.conf.d"
```

如果你看到报错，回到“用以下三种方式之一关闭 Xorg”并尝试用另一种方法。

在新窗口中运行“Arena-Baselines”之前，运行以下命令来将一个虚拟显示端口附于窗口：

```
export DISPLAY=:0
```

17.3.2 进行训练

创建 TMUX 会话（如果你用的机器是一个可以用 SSH 连接的服务器）并进入虚拟环境：

```
tmux new-session -s Arena
source activate Arena
```

连续动作空间

Arena 中连续动作空间的游戏列表：

* ArenaCrawler-Example-v2-Continuous
* ArenaCrawlerMove-2T1P-v1-Continuous
* ArenaCrawlerRush-2T1P-v1-Continuous
* ArenaCrawlerPush-2T1P-v1-Continuous
* ArenaWalkerMove-2T1P-v1-Continuous
* Crossroads-2T1P-v1-Continuous
* Crossroads-2T2P-v1-Continuous
* ArenaCrawlerPush-2T2P-v1-Continuous
* RunToGoal-2T1P-v1-Continuous
* Sumo-2T1P-v1-Continuous
* YouShallNotPass-Dense-2T1P-v1-Continuous

运行训练命令，将 **GAME_NAME** 用上面的游戏名替换并根据你所用计算机选择合适的 **num-processes**（**num-mini-batch** 需等于 **num-processes**）：

```
tmux new-session -s Arena
CUDA_VISIBLE_DEVICES=0 python main.py --mode train --env-name GAME_NAME --obs-type
    visual --num-frame-stack 4 --recurrent-brain --normalize-obs --trainer ppo
    --use-gae --lr 3e-4 --value-loss-coef 0.5 --ppo-epoch 10 --num-processes 16
```

```
    --num-steps 2048 --num-mini-batch 16 --use-linear-lr-decay --entropy-coef 0 --gamma
    0.995 --tau 0.95 --num-env-steps 100000000 --reload-playing-agents-principle
    OpenAIFive --vis --vis-interval 1 --log-interval 1 --num-eval-episodes 10
    --arena-start-index 31969 --aux 0
```

离散动作空间

Arena 中离散动作空间游戏列表：

* Crossroads-2T1P-v1-Discrete
* FighterNoTurn-2T1P-v1-Discrete
* FighterFull-2T1P-v1-Discrete
* Soccer-2T1P-v1-Discrete
* BlowBlow-2T1P-v1-Discrete
* Boomer-2T1P-v1-Discrete
* Gunner-2T1P-v1-Discrete
* Maze2x2Gunner-2T1P-v1-Discrete
* Maze3x3Gunner-2T1P-v1-Discrete
* Maze3x3Gunner-PenalizeTie-2T1P-v1-Discrete
* Barrier4x4Gunner-2T1P-v1-Discrete
* Soccer-2T2P-v1-Discrete
* BlowBlow-2T2P-v1-Discrete
* BlowBlow-Dense-2T2P-v1-Discrete
* Tennis-2T1P-v1-Discrete
* Tank-FP-2T1P-v1-Discrete
* BlowBlow-Dense-2T1P-v1-Discrete

运行训练命令，将 **GAME_NAME** 用上面的游戏名替换并根据你所用计算机选择合适的 **num-processes**（**num-mini-batch** 需等于 **num-processes**）：

```
CUDA_VISIBLE_DEVICES=0 python main.py --mode train --env-name GAME_NAME --obs-type
    visual --num-frame-stack 4 --recurrent-brain --normalize-obs --trainer ppo
    --use-gae --lr 2.5e-4 --value-loss-coef 0.5 --ppo-epoch 4 --num-processes 16
    --num-steps 1024 --num-mini-batch 16 --use-linear-lr-decay --entropy-coef 0.01
    --clip-param 0.1 --num-env-steps 100000000 --reload-playing-agents-principle
    OpenAIFive --vis --vis-interval 1 --log-interval 1 --num-eval-episodes 10
    --arena-start-index 31569 --aux 0
```

你也可以改用其他 MARL 算法来替代上面的 PPO 去测试你所创建的游戏。

17.3.3 可视化

为了用 Tensorboard 可视化分析训练过程的学习曲线，运行：

```
source activate Arena && tensorboard --logdir ../results/ --port 8888
```

并访问相应端口打开 Tensorboard 可视化。

17.3.4 致谢

我们特别感谢 Yuhang Song、教授 Zhenghua Xu、教授 Thomas Lukasiewicz 等人对 Arena 项目的巨大贡献。

参考文献

BALDUZZI D, GARNELO M, BACHRACH Y, et al., 2019. Open-ended learning in symmetric zero-sum games[J]. arXiv:1901.08106.

BANSAL T, PACHOCKI J, SIDOR S, et al., 2018. Emergent complexity via multi-agent competition[C]// International Conference on Learning Representations.

CAI Y, DASKALAKIS C, 2011. On minmax theorems for multiplayer games[C]//Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics.

HEESS N, SRIRAM S, LEMMON J, et al., 2017. Emergence of locomotion behaviours in rich environments[J]. arXiv:1707.02286.

HENDTLASS T, 2004. An introduction to collective intelligence[M]//Applied Intelligent Systems.

JADERBERG M, CZARNECKI W M, DUNNING I, et al., 2018. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning[C]//CoRR.

MYERSON R B, 2013. Game theory[M]. Harvard university press.

SINGH S, LEWIS R L, BARTO A G, 2009. Where do rewards come from[C]//Proceedings of the annual conference of the cognitive science society.

SINGH S, LEWIS R L, BARTO A G, et al., 2010. Intrinsically motivated reinforcement learning: An evolutionary perspective[J]. IEEE Transactions on Autonomous Mental Development.

SONG Y, WANG J, LUKASIEWICZ T, et al., 2019. Arena: A general evaluation platform and building toolkit for multi-agent intelligence[J]. arXiv preprint arXiv:1905.08085.