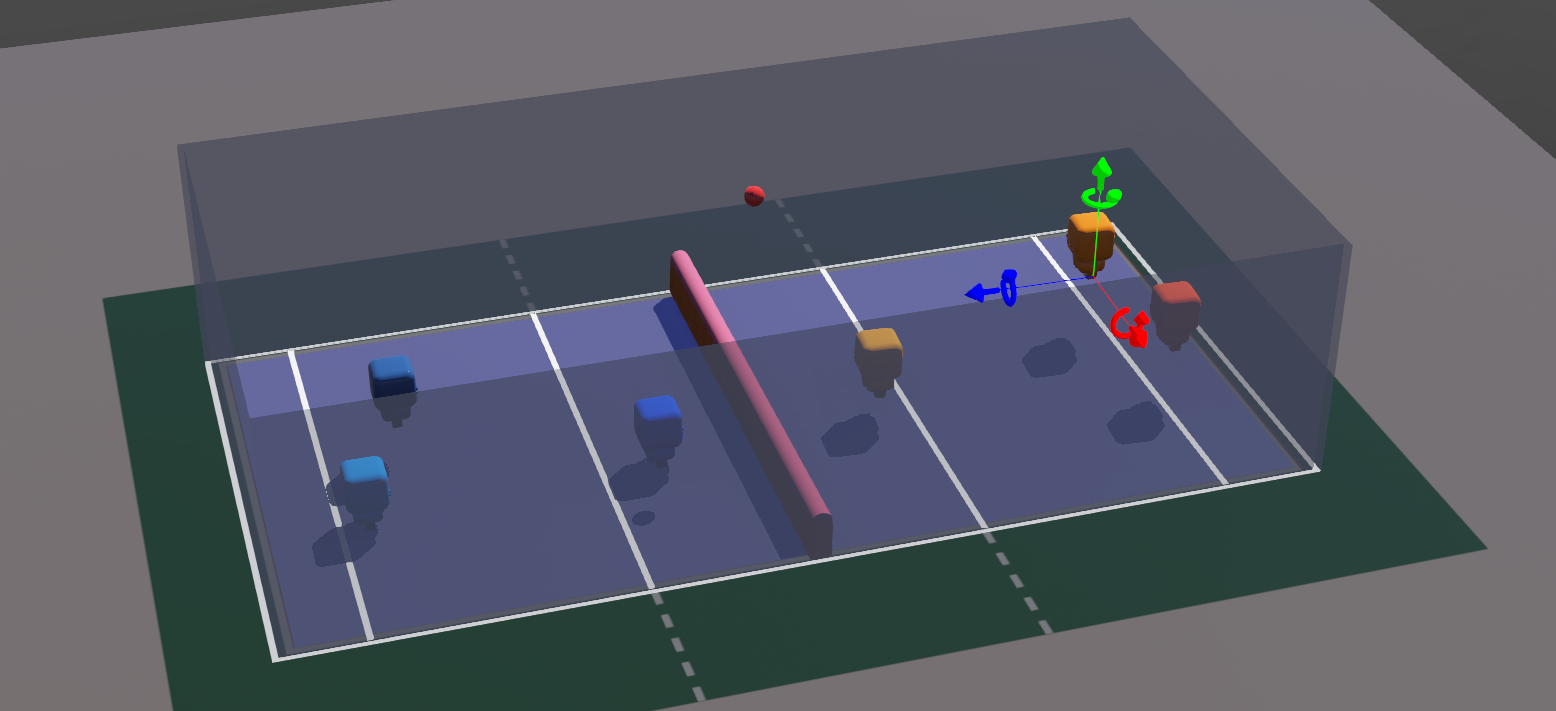
**期末文档-Robot Volleyball：多角色的多智能体排球比赛**



1. **项目信息**

**项目背景**

多智能体系统（multi-agent systems）是由多个独立的智能体组成的系统，这些智能体可以相互交互和合作，共同完成某些任务，也可以相互竞争。多智能体系统已经广泛应用于许多领域，如交通控制、仓储物流、协同机器人和游戏设计等。

排球是一项受广泛欢迎的团体运动，其不仅考验个人技能，还需要良好的团队协作能力。排球的规则相对简单，场地较小，每队只有六个球员，模拟排球相对来说目的较为明确；同时，与足球等主要在平面上运动的球类不同，排球需要在三维空间中运动，这使agent的行为致使的可能性更多，模拟更具有挑战性。

**项目简介**

本项目旨在利用多智能体技术模拟排球比赛。通过多智能体间的协作和竞争，两个队伍的智能体将会完成对真实排球比赛的仿真。本项目中，每个智能体将扮演排球比赛中的某一角色，如攻手、传球手等，其具备能进行一些动作，来完成拦网、扣球、传球等操作。每一智能体将与同队伍的其他智能体协作来进行比赛。

为了实现本项目，需要进行多智能体系统的设计和实现，包括智能体的动作设计、通信方案的选择、可观测内容和训练策略等。同时，需要对排球运动进行建模，将其与多智能体系统相结合，实现一个完整的比赛仿真。

该项目将利用多智能体强化学习技术对排球运动进行模拟，促进对多智能体技术和其在实际中应用的理解。

**项目设计**

本项目模拟排球运动中两支队伍进行对抗，每队3人。单支队伍中，三名成员通过权重设置进行角色分工，分为一名攻手，一名传手和一名自由人。

*本项目在OpenAI Gym库的强化学习代理环境下，模拟排球比赛的多人配合及对抗过程，并通过Webots进行可视化球场场景、人物智能体和排球。*

*Gym库是OpenAI的强化学习实验环境库，使用python语言实现离散时间智能体/环境接口中的环境部分，基于Gym有SlimeVolleyGym的这样一个简单的gym环境，本项目即在该环境下开发。*

*模拟中定义人物的动作（前进、后退、跳跃、左右），通过检测球与地板的碰撞判断胜负并给相应组奖励，两组代理采用self-play训练，同时创建环境副本加快训练速度。*

**场景**：本项目计划使用Webots构建网球场模拟场景，包括地板、球网、墙壁等，通过检测球与地板、边界的碰撞给与奖惩。Webots支持python，方便我们使用常见的python深度学习框架，同时其可以实现机器人的建模和控制，用以测试和验证机器人算法，其仿真效果也较为真实。同时，Webots包含许多真实移动机器人的接口，使用Webots开发使本算法迁移到真实机器人上成为可能。

**强化学习代理环境**：本项目使用多智能体强化学习对agent进行训练，对此，本项目采用了gym。gym本身实现了“agent-environment”（智能体-环境）循环。

**agent行为**：由于实际环境与游戏场景有差异，本项目中对agent的动作做出调整。设置agent具有前进、后退、向左移动、向右移动、跳跃、原地不动等几种动作，通过agent与排球的碰撞弹射排球，模拟对排球的扣球、传球等动作。人物与球场环境、人物与人物之间也会发生碰撞以模拟协同过程。

**agent角色**：排球队伍中有多种分工，如攻手、传手和自由人。本项目中通过对不同角色某一行为的奖惩设置不同，使某些角色更倾向传球，某些角色更倾向击球等，实现角色的分工。同时可尝试对agent的倾向位置进行划分，进一步实现此效果。

**队伍策略：**同时还涉及对抗过程，不同队伍设置不对等的奖惩策略，模拟实际比赛中的不同队伍的不同策略。比方说，有的队伍对成功传球的奖励较高，有的队伍对接中球的奖励较高，有的队伍对使对方接不中球的奖励奖励较高，等等。通过这种方式，我们可以训练多个队伍，同时也让多个队伍彼此配对训练，展现不同策略下的智能体表现

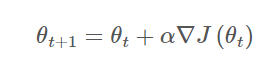
2. **算法调研**

我们主要调研了两种算法，即MADDPG与MAPPO，并分别尝试进行实现和应用。最终使用的是MAPPO，此处对其进行说明。

**MAPPO**

采用强化学习PPO代理训练各组。

*PPO算法是基于参数化的策略，直接根据观测输出动作，常常适用于动作为连续值的情况，通过1. 运行环境获取数据；2. 根据获取的数据计算梯度；3. 根据梯度上升（或下降，如果是针对损失函数）的方向优化策略的参数这三步来不断优化参数，从而使得策略能够获取更高的奖励函数值或更低的损失函数值。即*



*其中J（θ）即以 θ 为参数的奖励函数，策略的目标为最大化该奖励函数*

*与TRPO算法类似，PPO通过给出一个代理函数来取代对原始的目标函数，通过代理函数的优化来实现对策略本身的优化，采用软约束直接对原函数修改从而限制策略更新的幅度。*

MAPPO（Multi-Agent Proximal Policy Optimization）算法是一种用于多智能体强化学习的算法，它结合了Proximal Policy Optimization（PPO）算法和Actor-Critic框架，用于解决多智能体协同决策问题

**原理**

MAPPO算法的核心思想是通过优化单个智能体的策略和共享价值函数来实现多智能体的协同决策。每个智能体都有自己的策略网络和共享的价值网络。策略网络用于生成智能体的动作，价值网络用于估计智能体的状态价值或行动价值。

MAPPO算法使用PPO算法作为优化器。PPO是一种重要的单智能体强化学习算法，它使用剪切重要性采样比率来执行策略优化，以确保每次优化的步骤都在可接受的范围内。MAPPO通过扩展PPO算法的思想和方法来适应多智能体设置。

**实现步骤**

1. 初始化：初始化每个智能体的策略网络和共享的价值网络，并设置其他算法超参数，如学习率、批次大小等。
2. 采样数据：使用当前策略网络，通过与环境交互来采集训练数据。每个智能体都根据自己的策略选择动作，然后与环境交互并观察奖励和下一个状态。
3. 计算优势估计：使用共享的价值网络估计每个智能体的状态价值或行动价值。计算优势估计是为了评估每个动作相对于当前策略的好坏程度，以便于优化。
4. 执行策略优化：对于每个智能体，使用PPO算法执行策略优化。PPO的优化过程涉及两个重要的步骤：计算策略比率和执行重要性采样。策略比率用于衡量新策略相对于旧策略的变化，重要性采样用于在策略优化时进行采样。
5. 更新共享价值网络：使用优化后的策略网络采集新的训练数据，并使用这些数据来更新共享的价值网络。共享价值网络的更新可以采用类似于单智能体的价值函数优化方法。
6. 重复步骤2-5：重复执行步骤2到步骤5，直到达到预设的训练轮数或达到停止条件。

通过以上步骤，MAPPO算法可以实现多智能体的协同决策。通过策略优化和价值网络更新，智能体可以逐渐改善其策略和决策能力，以实现更好的协同效果。

**策略优化步骤**

在MAPPO算法中，策略优化阶段采用PPO算法的变体，以便在多智能体环境中进行有效的优化。具体来说，MAPPO使用剪切重要性采样比率（clipped importance sampling ratio）来限制策略更新的幅度，以确保优化步骤不会导致策略过大的改变。

下面是MAPPO算法中策略优化步骤的详细过程：

1. 重采样数据：使用当前的策略网络对数据进行重采样，以减少样本相关性。这意味着在每次优化步骤中，我们会使用新的轨迹（trajectories）数据。
2. 计算重要性比率：对于每个智能体，在策略优化之前，我们计算重采样数据和原始数据之间的重要性比率。这是为了调整优化步骤的幅度，以避免过大的策略更新。重要性比率的计算公式如下：



其中，



是更新后的策略参数，



是旧的策略参数。

1. 计算策略损失：使用重要性比率和优势估计来计算PPO算法的策略损失函数。策略损失函数的目标是最大化优势估计并在一定的范围内进行剪切操作，以限制策略更新的幅度。具体而言，策略损失函数的计算公式如下：



其中，Ratio是重要性比率，Advantage是优势估计，Clipped Ratio是一个剪切范围（通常为1+ε和1-ε之间，ε是一个超参数），用于限制策略损失的变化范围。

1. 执行梯度更新：根据策略损失函数计算的梯度，对策略网络的参数进行更新。通常使用随机梯度下降（SGD）或其他优化算法来执行梯度更新。
2. 执行剪切操作：为了确保策略更新在可接受的范围内，我们对计算得到的梯度进行剪切操作，将其限制在一个预定义的范围内。这样可以防止过大的策略更新，以避免破坏当前策略。

MAPPO算法通过重复执行上述步骤，不断优化智能体的策略和协同决策能力，以实现更好的性能。

除了上述的基本MAPPO算法，还存在一些改进和变体，如使用多步回报、GAE估计等，以增强算法的稳定性和性能。这些改进可以根据具体的应用场景和问题进行调整和应用。

**更新共享的价值网络**

在MAPPO算法中，除了策略优化，还有一个重要的步骤是更新共享的价值网络。这个步骤用于估计智能体的状态价值或行动价值，并为策略优化提供更准确的估计。

共享的价值网络可以使用各种价值函数估计方法，例如基于蒙特卡洛采样的方法或基于TD误差的方法。一种常用的方法是使用基于TD误差的方法，如Advantage Actor-Critic（A2C）算法。

在MAPPO中，共享的价值网络可以通过均方误差损失函数进行训练和更新。具体而言，我们使用优化后的策略网络对新的训练数据进行采样，并使用这些数据来计算状态价值或行动价值的目标值。然后，我们使用均方误差损失函数来最小化预测值与目标值之间的差异，以更新共享的价值网络的参数。

这样，在MAPPO算法中，策略优化和共享价值网络的更新是交替进行的，以相互促进并提升智能体的性能和协同决策能力。

另外，对于MAPPO算法的实现，您需要考虑以下一些关键因素和技术：

1. 神经网络架构：策略网络和价值网络通常使用深度神经网络来进行函数逼近。您可以选择适合问题的网络结构，如多层感知机（MLP）或卷积神经网络（CNN）等。
2. 学习率和优化算法：在MAPPO算法中，您需要设置适当的学习率和选择合适的优化算法，如随机梯度下降（SGD）或Adam等。
3. 超参数设置：MAPPO算法中有一些关键的超参数需要设置，例如剪切范围（clipping range）、优势估计方法、重采样次数等。这些超参数的选择对算法的性能和收敛性都会产生影响。
4. 并行化和分布式训练：由于MAPPO算法需要多个智能体进行交互和优化，可以考虑使用并行化和分布式训练技术，以提高算法的效率和训练速度。
5. 调试和调优：实现MAPPO算法时，可能需要进行一些调试和调优的工作。您可以使用日志记录、可视化工具和曲线绘制等方法来监控算法的训练过程和性能。

综上所述，MAPPO算法是一种用于多智能体强化学习的算法，它通过优化单个智能体的策略和共享的价值函数来实现多智能体的协同决策。通过交替执行策略优化和共享价值网络的更新，MAPPO算法可以不断提升智能体的性能和协同能力。

**效果**

MAPPO在多智能体强化学习方向表现出很好的效果，并且已经在多个领域和任务中取得了显著的成果。MAPPO算法在多智能体强化学习中具有以下优点：

1. 协同决策：MAPPO算法通过优化单个智能体的策略和共享的价值函数，可以实现多智能体之间的协同决策。智能体可以通过共享的价值网络来学习其他智能体的行为和经验，并据此优化自己的策略。这种协同决策能力使得多智能体系统可以在复杂环境中更好地合作和协作。
2. 支持多智能体互动：MAPPO算法能够处理多个智能体之间的互动和竞争，例如博弈对抗场景或合作任务。通过对不同智能体的策略优化，MAPPO能够在多智能体系统中实现平衡和合作，以达到更好的整体性能。
3. 策略稳定性：MAPPO算法使用PPO作为优化器，其中包含剪切重要性采样比率，以控制策略更新的幅度。这种剪切操作可以提供一定的稳定性，避免过大的策略更新幅度，从而保持策略的稳定性和收敛性。
4. 可扩展性和并行化：MAPPO算法可以与并行化和分布式训练技术结合，以加速训练过程。通过在多个环境实例中并行地运行多个智能体，可以提高数据收集的效率和训练速度。这种可扩展性使得MAPPO算法在处理大规模多智能体系统时具有优势。
5. 适应性和泛化性能：MAPPO算法能够适应不同环境和任务，并具有一定的泛化能力。通过在多个智能体之间共享价值网络和进行策略优化，MAPPO算法可以学习到一般化的决策规则和策略，从而在新的环境和任务中表现出良好的性能。

3. **游戏环境**

**环境设计**

**Observation space**

出于协作和竞争的目的，每个智能体都应当能根据Observation对球的状态和友方、敌方的状态进行考量，也就是说Observation应当包含球、友方和敌方的状态信息。

因此，对于每个时刻生成了全局的global\_state，包含场上每个智能体的位置、球的位置和速度，编码为[agent\_info,ball\_info]

global\_state对于每个agent个体，又生成了个性化的agent\_state，即以该agent面前的球网中心为坐标原点，向右为x轴正方向，向前为y轴正方向，向上为z轴正方向，按照此坐标系进行转换，以此得到相对于面前球网的自己的位置、友方的位置、敌方的位置、球的位置和速度，编码为[self\_pos,friend\_pos,enemy\_pos,ball\_info]

这个转换的目的是得到相对位置，以此训练建立关系，使得训练结果能够迁移，避免因不同比赛中位置不同而输入输出不匹配的情况。

**Action Space**

动作空间设置为离散，设置了6个基本动作，使用one-hot编码：

* 跳跃：[1,0,0,0,0]
* 向前运动: [0,1,0,0,0]
* 向后运动: [0,0,1,0,0]
* 向左运动: [0,0,0,1,0]
* 向右运动: [0,0,0,0,1]
* 原地停止: [0,0,0,0,0]

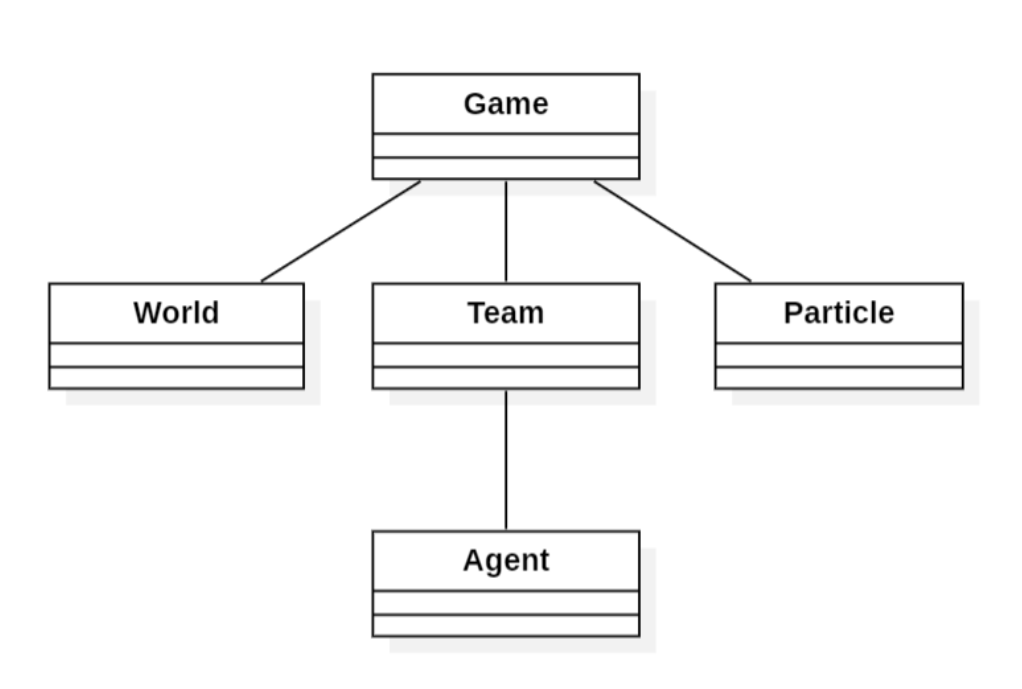
每个agent将设置为一个球型的碰撞体，通过碰撞的角度的不同，可以使排球向不同的方向运动

**环境构建**

环境在强化学习中扮演着至关重要的角色。环境定义了智能体所处的状态，以及智能体与环境之间的交互方式。环境的目的是为智能体提供学习和决策的反馈，从而使其能够逐渐改进其行为以实现预期的目标。

本项目采用Python原生代码而非Unity等平台进行搭建，这使得我们能更快速、更灵活地进行迭代和尝试，而不必受到实际建模的限制，也可以更加灵活地调整和优化环境，提高算法的训练效果。

游戏中设计了以下几个类，用以构建完整的游戏环境：



**World**

World类代表了整个游戏进行所处的世界，即排球场。

球场大小的尺寸是整个游戏尺寸设置的基准，其他的类的尺寸设置需要与其相匹配，避免出现不对等的情况。

World设置了以下属性：球场的长度height（y轴）、宽度width（x轴）、高度depth（z轴），agent在各个方向上的速度vx、vy、vz，球的最大速度max\_ball\_v，以及世界中的重力常量g。同时还设置了围墙的长宽高，方便后续建模与可视化。

**Game**

Game类是对游戏本身的抽象，对游戏的开始、更新和重置等操作进行控制和检测。

Game类设置了以下属性，排球的实体ball（由*Particle类生成）*，左右两边的队伍实体team\_left和team\_right（都由Team类生成），标识是否在训练中的training，记录比赛场次的match，记录球是否发生碰撞的BallCollision等。

**Team**

Team 类用于表示进行对抗的队伍（游戏中是两队进行对抗）。该类用于控制各个队伍中各个agent的各种行为（包括移动，位置信息等等）。它具有的属性包括name，n\_mates，team，dir，life等。

**Agent**

Agent 类用于表示队员个人的情况。它具有的属性包括dir，x，y，z，vx、vy、vz，bonus，name，life等信息。

**Particle**

Particle 类用于表示球和栅栏上方的圆桩。 该类在游戏的物理模拟中起到了重要的作用，实现了球和围栏的运动和碰撞检测，为游戏的实现提供了必要的基础。

**TeamVolleyBot**

自定义的gym环境类，模拟双人排球游戏，当agent失去5条命或经过了3000个时间步时结束。

4. **训练与改进**

**多角色-位置分工与角色分工**

本项目期望达到多角色的效果，其中采用了两种多角色方法，第一种是接球的位置分工，第二种则为agent的角色分工。

**位置分工**

最终的reward由两部分组成：

1. agent\_results：代表一个球的输赢（正值为赢，即球在对面落地，负值为输，即球在自己这边落地）

*如果该步中没有球落地，则result为0*

*为了对agent负责的位置尽量做出区分，当输球时，如果球掉在左边，则agent0罚值-2，agent1罚值-1，agent2罚值0；掉在右边则调换。我们希望这能够对agent的倾向防守位置做出分工，agnet0尽量在左边，agent2尽量在右边，agent1则在中间进行灵活防守；赢球则奖励一致，都为1*

1. Bonus：代表对位置倾向的奖惩，其由奖惩两部分组成：
2. bonus：如果在自己应处的位置上，则奖励0.001，如果不在，则惩罚-0.001
3. malus：如果和队友相撞了，则不在自己应处的区域的惩罚-0.01

整体来说，agent\_results的绝对值要比位置的奖惩要大得多，这是因为输赢球才是此情景下真正重要的东西。

**角色分工**

这一版本的reward整体与位置分工相同，但是新增了过程的奖励，即击中球的奖励。同时其中设置了三种角色：

* Hitter：进攻者
* Libero：接应，防守和接发球
* Utility：自由人

与位置分工对比，此时取消了位置的限制，即赢球时所有agent都获得奖励1，输球时所有agent都获得惩罚-1，同时碰撞时所有agent获同样的惩罚-0.01。接中球时，接中球的agent获奖励0.1。

不同角色的agent在计算最终reward的时候各分量权重不同。

角色分工中使用权重表如下：

*备注：这是奖惩迭代的第二版时的权重表，最终权重表请见“奖励的设置与调整”节*

"Hitter":{  
 "ball\_bonus":2,  
 "bonus":1, # side\_bonus  
 "malus":0.5, # 碰撞惩罚  
 "result\_fail":0.5,  
 "result\_win":2  
},  
"Libero": {  
 "ball\_bonus": 2,  
 "bonus": 1, # side\_bonus  
 "malus":2, #碰撞惩罚  
 "result\_fail":2,  
 "result\_win":0.5  
},  
"Utility": {  
 "ball\_bonus": 1,  
 "bonus": 1, # side\_bonus  
 "malus":1, # 碰撞惩罚  
 "result\_fail":1,  
 "result\_win":1  
}

**奖励的设置和调整**

对于奖励的调整经过了好几轮的迭代。

初始版本中奖励设置较简单，只有赢球、输球时的奖惩，以及对位置限制的奖惩，但我们发现这样反馈太慢，训练的收敛速度太差。

随后，第二轮中，我们加上了接球的奖励，虽然这引入了局部收敛的隐患（agent可能倾向于不断自己接投球，而不是使别人输球），但这大大加快了训练的速度，很明显的，agent开始有了一定的行动倾向。

最后，我们引入了更多的先验知识，即与球的运动相关的知识。我们让agent在横轴上离球越远惩罚越大，也就是说让agent有意的在水平方向上向球靠近。当然，此距离有一阈值，即水平方向上离球较远时出现此惩罚。

*该惩罚（dis\_malus）的设置如下：z轴长度是24，赢球奖励1，碰撞惩罚0.01，而且dis\_malus是每个时刻都会进行判定的。因此dis\_malus的系数只能很小，比如-0.00005，同时出现惩罚的开始阈值就设为2\*r（r为球的半径）*

*注：此奖惩和碰撞的惩罚结合，或许一定意义上促进了agent在前后场站位的分层*

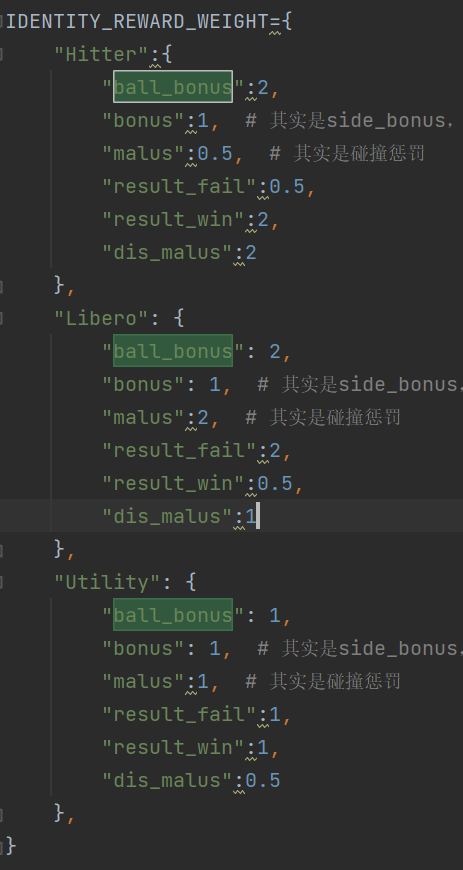
最终，我们得到了以下的三层奖惩结构，在不同的层面上对agent做出引导：

1. 最直接的先验知识的引导：z轴上引导agent向球靠近
2. 稍微高层次一点的：接球的得分，还有相撞的惩罚
3. 最终目的：赢下一局球

奖惩频率逐渐提高，而奖励系数也理应逐渐加大。

同时，三种角色的奖惩系数安排如下：

*注：bonus和malus其实是位置引导的奖励和碰撞惩罚，因为奖惩设置经过迭代，就没有更改最初这两项的命名*



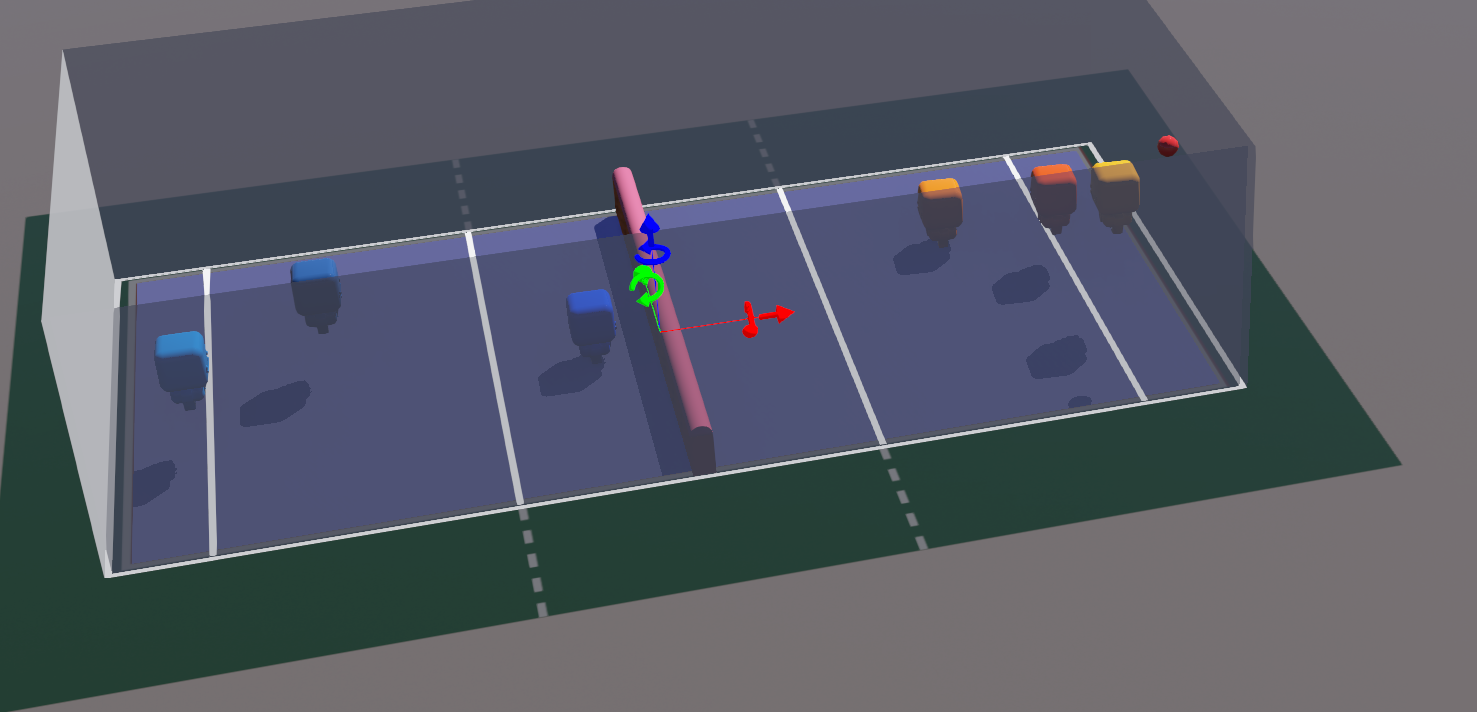
5. **控制模块**

出于游戏性和交互性的目的，我们为项目添加了控制模块，即允许观察者通过键盘选择和控制某一agent移动，此时该agent将不再被其已训练模型的策略控制，而是被观察者的键盘控制。

我们在Agent类中添加新属性controlled\_by\_board，默认为false。在Team的set\_action函数中根据列表为每个agent赋予动作。随后，我们通过键盘选择并释放某一agent（在每个update中检测键盘状态），选择某一agent后即可用上下左右键控制agent移动。

*由于我们没有在Webot的文档中找到界面按钮和其他UI，我们使用键盘进行agent选择和释放：1-6分别控制左队的agent1-3和右队的agent1-3，选择一个时自动释放其他的agent，0释放所有agent。*

*按下1后，此时图中最右边的蓝色agent（即agent1）已被我控制。*



6. **分析与展望**

现在已经训练了相当多轮次，虽然目前agent表现不错，但离真正进行一局游戏还是有距离。经分析，我们觉得可能有以下原因：

1. 可能是因为网络模型比较小
2. 可能是因为MAPPO学习效率较低，收敛较慢
3. 可能是因为奖惩设置的反馈还是比较缓慢（同时可能存在这样的问题：赢球的奖励设置的太小了，况且周期较长，导致agent比起赢球可能更倾向不断地接球，恐怕奖励起码需要设置为10，甚至更大）
4. 可能是因为游戏设置中有不合理的地方，比如agent的半球形的身体设置，比如仅能靠碰撞进行比赛

我们考虑进行如下改进：

从收敛速度上来说：

* 可以引入新的先验知识进行奖惩，比如计算球的落点
* 可以使用学习效率更高的模型

从agent表现上来说：

* 可以使用更复杂的网络结构
* 可以调整动作空间为连续空间

从真实性上来说：

* 现在的场景设计有不科学的地方，比如没考虑好空中的问题，agent在空中也能前后移动，需要调整
* 需要加入真实排球比赛中的其他规则，比如不允许在自己的场次一直传球
* 可以增加新的动作，而不只是简单的跳起碰撞，可以有新动作如传球等