毕业论文报告

2019.2.12 谢文韬

0.摘要

研究工作的主题是围绕人机对抗型游戏的资源和策略配置，本阶段研究内容主要集中在多智能体的模拟平台的应用，通过对与博弈和强化学习相关的论文阅读，结合 github 上项目源码的运行和分析，熟练掌握平台的应用和配置，并模拟简单实验的结果。

当前阶段主要研究的是 AAAI2018 的论文关于 MAgent 多智能体平台[6]，运行基础的 demo 来熟悉该框架的 API。对于其他一些主流的多智能体仿真平台进行了解，如实用性好的 AFSIM[1]。在对 AFSIM 和 MAgent 两篇论文进行了仔细研读和分析后，我对于多智能体的模拟和仿真有了初步的了解，由于只有 MAgent 有相应的代码，因此对于 MAgent 的理解更直观，而 AFSIM 是一个军事化的集成开发环境，相较于轻量级的 MAgent 有很多额外的模块和组件从而可以模拟更复杂的任务， MAgent 则对于简单算法的实现和源码阅读调试则更加灵活。//1.8

在上一阶段工作基础上，本周对上周的报告内容进行修订，部分内容重新编写，重点在 MAgent 的战场游戏 demo 的详细描述上。同时开始对博弈论中对于纳什均衡计算的学习，对一些资料和具体问题做整理。//1.13

近期工作围绕 MAgent 平台中的智能单体设计及其群管理的工作，对于 1.14 会议中多智能体强化学习中的算法部分进行学习，修订基础理论综述中关于多智能体强化学习算法的内容。本周继续在机器之心、新智元上搜集关于游戏 AI、强化学习和博弈等内容的国际进展，整理成表。//1.21

寒假的工作确定了最终展示的大致流程，包括展示一个 demo 界面，解释 demo 中的 RL 和博弈参数，解释平台的使用，以及算法之间效果的比较分析图。进一步更新了截至 2019 年 1 月的论文集，暂时未放上 github，会在开学的组会上做简要分享。重新仔细阅读多智能体强化学习算法相关的论文，结合对策论在雷达攻防应用中的实验设计，给出初步的实验设计思路和算法构想。修订了部分文稿的表述和引用文献问题。//2.12

以上部分不作为毕业论文综述，主要用来记录报告进度。

1.基础理论综述

1.1 AFSIM 平台

1.1.1 函数式架构

AFSIM（The Advanced Framework for Simulation, Integration and Modeling，仿真，集成和模拟的高级框架）[1]平台是众多多智能体仿真平台中实用性较强的，是一个面向对象、基于 C++的集成开发环境，用于构建军事级别的资源配置分析和仿真。AFSIM 平台的开发对象包括模拟武器的运动学过程、感知系统、电力伺服系统、通信网络、高级追踪系统等战场管理对象。AFSIM 平台基于早期的 AFNES（Analytic Framework for Network-Enabled Systems 平台），AFNES 平台在 2013 年 2 月开源。该平台主要由三大模块组成，分别是提供重要接口和功能的框架部分、集成开发环境和可视化模块，为用户提供模拟的顶层控制，特别是对于时间和事件的管理，还包括对众多用户定义的行动单位、感知器、武器和处理器的行为控制。

AFSIM 的框架架构如下图 1，组件主要包括（1）仿真对象的类定义，包括行动单位、感知器等等；（2）事件和时间的控制，实体数据产生的消息；（3）支持系统的标准数学库，例如存储数据和对象的标准容器；（4）导入标准地理数据的模块，方便导入标准化的地理情况数据，用于真实战场的仿真；（5）支持脚本化的对象管理，形式上简化 API 的复杂程度；（6）通信网络的模拟构建，提供例如网络节点、路由器、多通道协议和消息队列等等；（7）电力伺服系统仿真，包括各类线路噪声和拥塞控制方面的情况；（8）基于信息流和任务的模拟，构建系统和玩家之间的桥梁；（9）支持批处理和实时系统的不同情况；（10）用户的应用程序接口。除了各个组件的核心部分之外，还支持 RIPR（Reactive Intergrated Planning Architecture）的智能体算法，支持各种量化的任务等等内容。

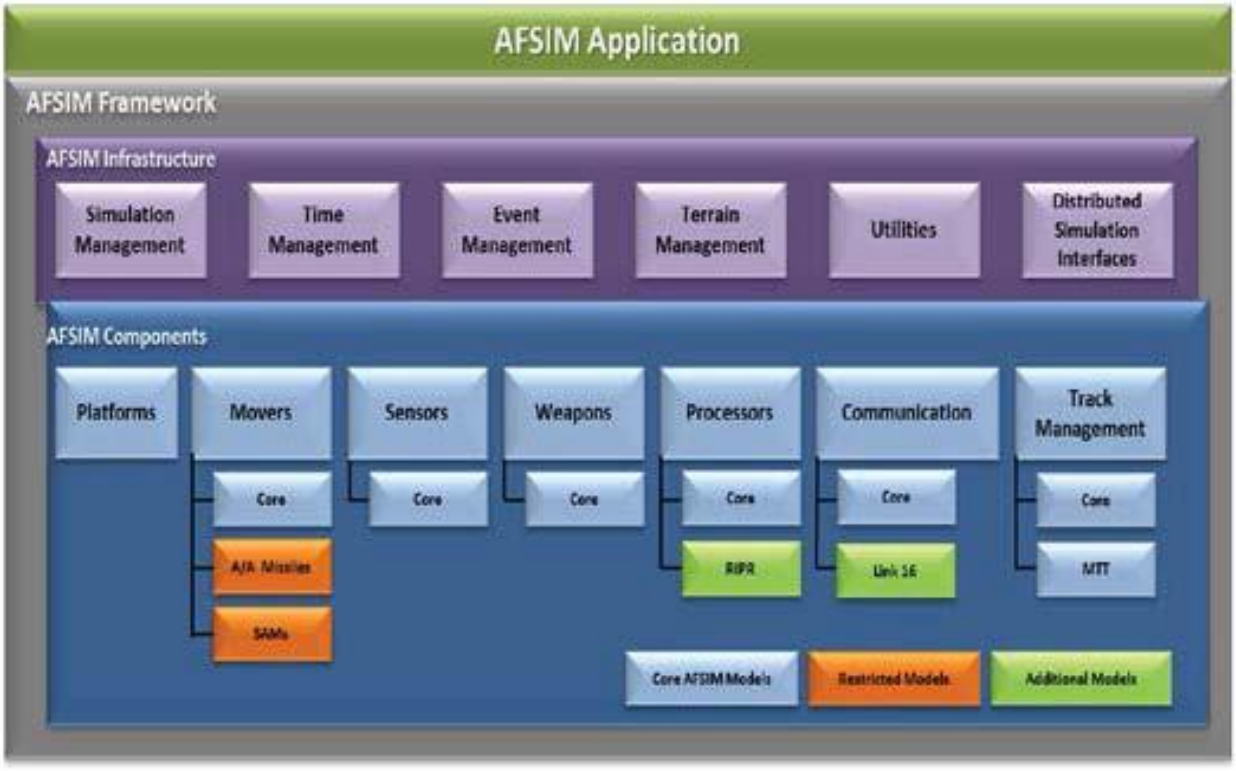


图 1. AFSIM 系统的函数式架构

1.1.2 IDE 和 VESPA

AFSIM 平台的集成开发环境和可视化模块 VESPA（Visualization Environment for Scenario, Preparation and Analysis）主要是提供方便用户直接调用的脚本式管理和图形界面（GUI）如图 2。其中核心的功能包括语法的高亮显示，便于源码和模块的阅读和理解，图形界面则可以方便地在特定的海拔和经纬度放置对象，得到更直观的结果，并可以实时调节参数并观察结果，增加系统的实用性。

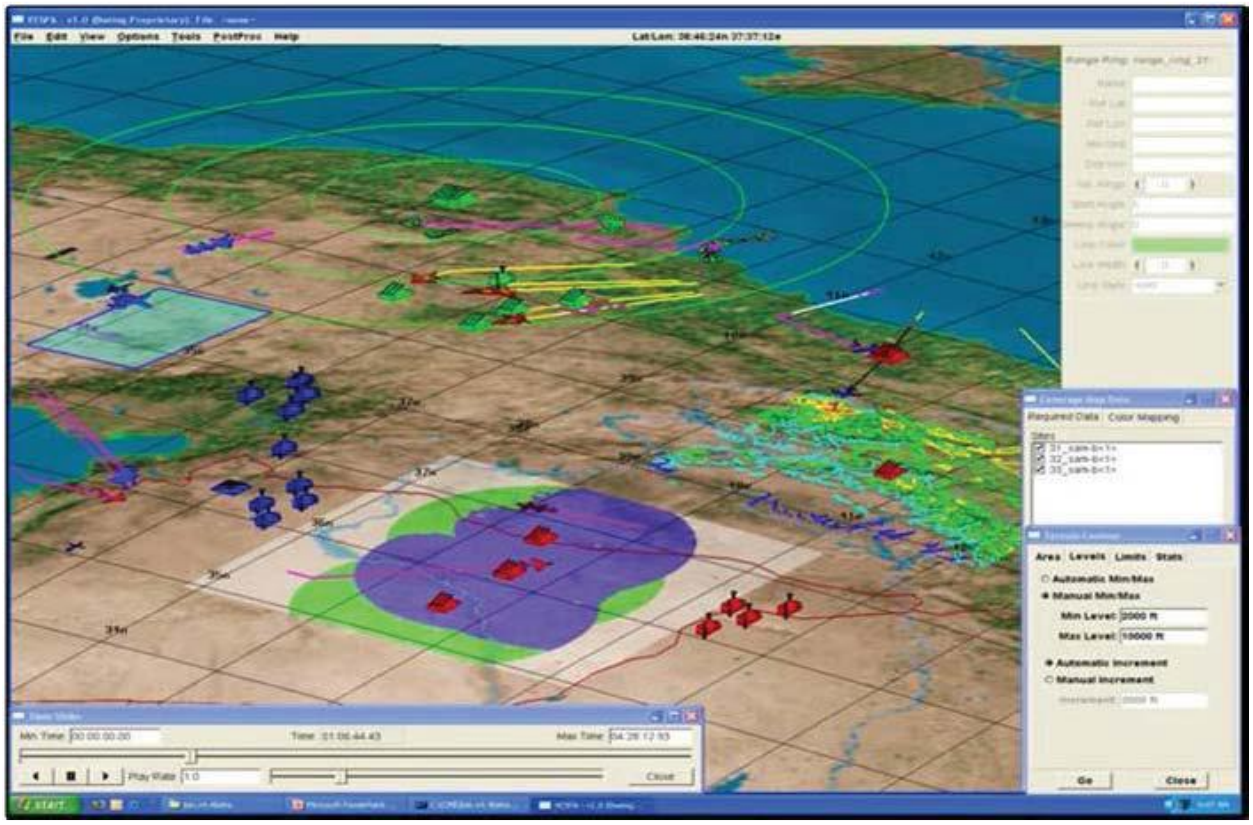


图 2.VESPA GUI

1.1.3 RIPR 集成策略架构

RIPR 是对于行为仿真的模块，基于智能体围绕自身设定的算法进行行为控制，同时又保证了智能体之间的交流和合作，RIPR 是一个构建多智能体的算法和工具集合。RIPR 的智能体，包含一个感知处理器和量化任务处理器，通过对于平台和其子系统的查询感知虚拟环境，再结合自身的人工智能算法处理信息并做出相应决策。

RIPR 的算法实现分为四个模式。第一是认知模型，每个智能体拥有其自身周围威胁、拥有资源和同伴的感知，并为不同的感知智能体构建不同的模型，这通过感知处理器的实现。

第二是量化任务者，如图 3 所示，该过程包括任务的产生、评估和分配以达到最优效果。任务的步骤主要如下，首先从感知部分的矩阵列获取资源和威胁的信息，接着为矩阵行产生任务，通过策略决策是否接受任务、拒绝任务或解决新任务。然后对矩阵对象计算任务的价值，通过分配器进行分配，最终任务被安排接受或拒绝。对于一个决策过程来说，矩阵的行代表自己的策略和应对，而矩阵的列对应对方的策略和环境，由于不同的场景条件下矩阵的行列要素不同，所以以具体条件为转移。例如一个雷达攻防的案例中，我方雷达的策略宽限窄、副瓣对消、副瓣匿隐、自动频率选择等，对于对方的决策来说，这就构成其矩阵列条件，其对应的行就是干扰方采取的瞄准、扫频、阻塞、转发式欺骗和箔条等。[7]

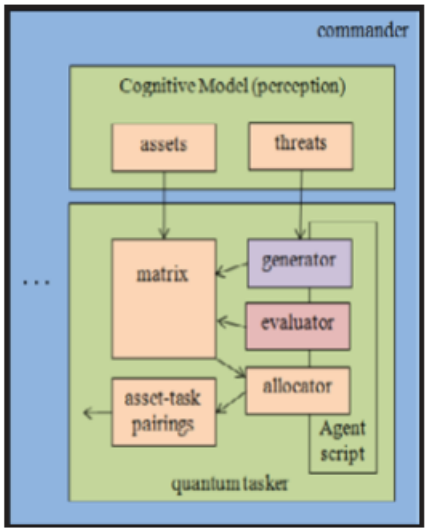


图 3.量化任务的模式

第三是通过行为树，行为树通过一系列的脚本代表最基本的智能体动作行为，通过组织树中结点的结构从而使得行为按照不同的顺序和方式执行，行为树包括行为选择、行为序列、平行行为、随机权重、优先级选择五种结点形式，图 4 是一种行为树的形式。

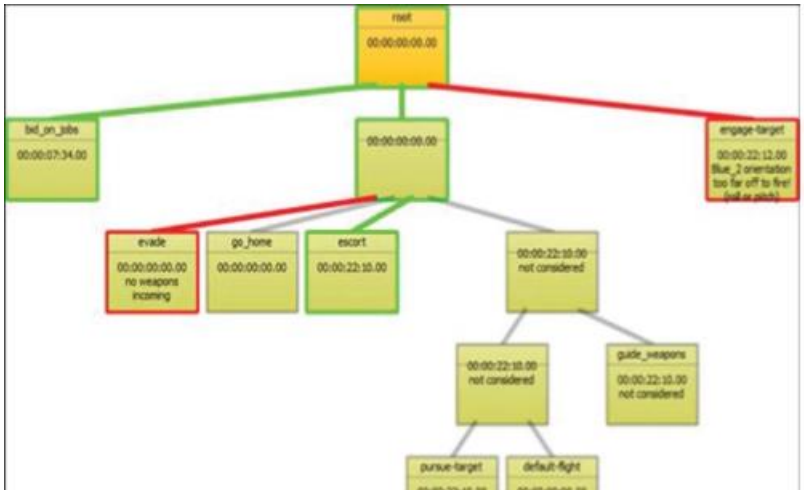


图 4.RIPR 行为树

第四是聚类管理，将类似的智能体通过聚类的方式群体化管理，通过行为的类似性进行分类，主要采用的算法有层次化的最大最小树和常规的 K-Means 聚类法，这些都是在数据挖掘中常见的聚类方式，对于多智能体的简化和分群管理有很有效的作用。

1.2 多智能体的强化学习

1.2.1 背景

多智能体系统由一群有自主性的，可互相交互的实体组成，它们共享一个相同的环境，通过感知器感知环境并通过执行器采取行动。多智能体在现实生活中已有应用，如机器人战队，分布式控制和资源管理。虽然可以预先设定多智能体的行为表现，但因为环境太过复杂，有时甚至会随时间而变化。所以很难提前设计一个良好的行为，或者随时间推移，先前良好的行为也会慢慢变差。通常需要在线学习新的行为，才能提高智能体或整个系统的性能。

在单 Agent 的强化学习中，Agent 在感知完环境的状态后，采取了一个动作，使得环境转移到下一个状态，并得到一个评价这次动作好坏的反馈。Agent 的学习目标就是最大化累计反馈。强化学习的反馈比有监督学习信息量少，但多于无监督学习。通过一些简化或泛化，单 Agent 的强化学习算法也可以运用到多 Agent 中。

多 Agent 强化学习较单 Agent 存在一定优势：不同 Agent 通过共享经验，可以更快更好的完成任务。比如有经验的 Agent 可以当老师，指导无经验的 Agent；如果任务可以拆分不同子任务时，不同 Agent 可以并行执行子任务，以此加速计算；当系统中有 Agent 失效时，其他 Agent 可替代执行任务，从而使整个系统更加鲁棒；而且可方便地加入新的 Agent，扩展性也更好。

但同时也面临着一些挑战：首先维度灾难问题更加严重，之前状态转移概率函数和回报函数都是在联合动作空间下计算的。随着状态和动作的增加，计算复杂度呈指数增长。其次学习目标不好定义，Agent 的回报跟其他 Agent 的行为相关的，没办法单独最大化某个 Agent 的回报。不稳定性也是 MARL 的一个问题，Agent 是同时在学习的，每个 Agent

都是面临着一个不停变化的环境，最好的策略可能会随着其他 Agent 策略的改变而改变。最后，探索和贪婪过程会更复杂，在多 Agent 下，探索不仅是为了获取环境的信息，还包括其他 Agent 的信息，以此来适应其他 Agent 的行为。但是又不能过度探索，不然会打破其他 Agent 的平衡。

基于上述挑战，在 MARL 中，主要关注两方面学习目标，稳定性(stability)和适应性(adaptation)。稳定性指 Agent 的策略会收敛至固定，而适应性确保性能不会因为其他 Agent 改变策略而下降。收敛至均衡态是稳定性的基本要求，这要求所有 Agent 的策略收敛至协调平衡状态，最常用的是纳什均衡。适应性体现在理性或无悔两个准则上。理性指出，当其他 Agent 稳定时，Agent 会收敛于最优反馈。无悔是说最终收敛的策略，其回报要不差于任何其他的策略。

1.2.2 DQN

深度 Q 学习（DQN）是经典 Q 学习算法的变体，有 3 个主要贡献：（1）深度卷积神经网络架构用于 Q 函数近似；（2）使用小批量随机训练数据而不是在上一次经验上进行一步更新；（3）使用旧的网络参数来评估下一个状态的 Q 值。DQN 的伪代码见算法 1。深度卷积架构提供一个通用机制从图像帧的短历史（尤其是最后 4 帧）中评估 Q 函数的值。后面两个贡献主要关于如何使迭代的 Q 函数估计保持稳定。监督式深度学习研究中，在小批量数据上执行梯度下降通常是一种高效训练网络的方式。在 DQN 中，它扮演了另外一个角色。具体来说，DQN 保存大量最近经验的历史，每个经验有五个元组（s, a, s', r, T）：智能体在状态 s 执行动作 a，然后到达状态 s'，收到奖励 r；T 是一个布尔值，指示 s'是否为最终状态。在环境中的每一步之后，智能体添加经验至内存。在少量步之后（DQN 论文使用了 4 步），智能体从内存中进行小批量随机采样，然后在上面执行 Q 函数更新。在 Q 函数更新中重用先前的经验叫作经验回放（experience replay）[Lin, 1992]。但是，尽管强化学习中的经验回放通常用于加快奖励备份（backup of rewards），DQN 从内存中进行小批量完全随机采样有助于去除样本和环境的相关性，否则容易引起函数近似估计中出现偏差。最终的主要贡献是使用旧的网络参数来评估一个经验中下一个状态的 Q 值，且只在离散的多步间隔（many-step interval）上更新旧的网络参数。该方法对 DQN 很有用，因为它为待拟合的网络函数提供了一个稳定的训练目标，并给予充分的训练时间（根据训练样本数量决定）。因此，估计误差得到了更好地控制。

Algorithm 1 Deep Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode 1, M **do** Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in the emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store experience $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of experiences $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the weights θ

算法 1 Deep Q-learning 的实现

1.3 博弈与纳什均衡

1.3.1 博弈问题

博弈问题在生活中非常普遍，大到国家之间的外交博弈，小到两人之间简单的石头剪刀布，都是博弈的过程。博弈有复杂简单之分，常作为理论研究的是基础博弈和它们的变种，常常和其他理论相结合。可以通过研究分析出博弈结果和策略的博弈模型为两人轮流决策的非合作博弈。即两人轮流进行决策，并且两人都使用最优策略来获取胜利。博弈是有限的。即无论两人怎样决策，都会在有限步后决出胜负。主要是公平博弈。即两人进行决策所遵循的规则相同，这些问题可以通过数学推导和计算机模拟推算出各个策略获利的概率。常见的有取物的巴什博弈(Bash Game)、威佐夫博弈（Wythoff Game）等等。

1.3.2 纳什均衡

在博弈论中，以已故数学家约翰福布斯纳什小命名的纳什均衡是一个非合作博弈的提议解决方案，涉及两个或多个参与者，其中假设每个参与者都知道其他参与者的均衡策略，没有玩家可以通过改变他们自己的策略来获得任何收获。在博弈论方面，如果每个参与者都选择了策略，并且没有玩家可以通过改变策略而其他玩家保持不变，那么当前的策略选择集及其相应的支付构成了纳什均衡。简单地说，如果爱丽丝做出最佳决定，爱丽丝和鲍勃处于纳什均衡状态，考虑到鲍勃的决定，而鲍勃的决定保持不变，鲍勃正做出最佳决定，考虑到爱丽丝的决定，而爱丽丝的决定仍然存在不变。同样地，如果每个人都做出最佳决策，一组球员处于纳什均衡状态，只要其他方的决定保持不变，就考虑到其他人在比赛中的决定。纳什表明每个有限游戏都有一个纳什均衡。

游戏理论家使用纳什均衡概念来分析几个决策者的战略互动的结果。换句话说，它提供了一种方法来预测如果几个人或几个机构同时做出决定将会发生什么，以及每个机构的结果是否取决于其他人的决定。约翰纳什的想法背后的简单见解是，如果单独分析这些决策，就不能预测多个决策者的选择结果。相反，必须考虑每个参与者的决策，并考虑其他人的决策。纳什均衡已被用于分析战争和军备竞赛等敌对局势（参见囚徒困境），以及如何通过反复互动减轻冲突（见针锋相对）。它也被用于研究具有不同偏好的人在多大程度

上可以合作（参见性别之争），以及他们是否会冒险实现合作结果（参见寻找狩猎）。它已被用于研究技术标准的采用，以及银行挤兑和货币危机的发生（见协调游戏）。其他应用包括交通流量（参见 Wardrop 的原则），如何组织拍卖（参见拍卖理论），多方在教育过程中所作出的努力的结果，环境法规等监管立法（参见下议院的悲剧），自然资源管理，分析营销策略，甚至足球中的罚球（参见匹配的便士）。

纳什证明，如果我们允许混合策略，那么每个玩家可以从有限多个纯策略中选择的有限数量的玩家的游戏至少有一个纳什均衡。如果选择集无限且非紧凑，则不需要存在纳什均衡。一个例子是两个玩家同时命名一个自然数字，玩家命名较大的数字获胜。然而，如果选择集紧凑且具有连续收益，则存在纳什均衡。一个例子（均衡是连续多个纯策略的混合）是两个玩家同时选择 0 到 1（包括）之间的实数，玩家一个奖金（由第二个玩家支付）等于平方根两个数字之间的距离。

1.3.3 平均场游戏理论

平均场博弈论是对大量小交互代理人的战略决策的研究。Boyan Jovanovic 和 Robert W. Rosenthal 在经济学文献中考虑了这类问题，在 Peter E. Caines 和他的同事的工程文献中，并且大约在同一时间由数学家 Jean-Michel Lasry 和 Pierre-Louis Lions。术语“平均场”的使用受到物理学中的平均场理论的启发，该理论考虑了大量粒子系统的行为，其中单个粒子对系统的影响可忽略不计。在连续时间中，平均场比赛通常由描述个体的最优控制问题的 Hamilton-Jacobi-Bellman 方程和描述代理的总分布的动态的 Fokker-Planck 方程组成。在相当一般的假设下，可以证明一类平均场比赛是极限的 N-二玩家 纳什均衡。与平均场比赛相关的概念是“平均场型控制”。在这种情况下，社会计划者控制状态的分布并选择控制策略。平均场型控制问题的解决方案通常可以表示为与 Kolmogorov 方程耦合的双 Hamilton-Jacobi-Bellman 方程。平均场型博弈论是平均场型控制的多智能体泛化。

1.3.4 博弈在战场攻防中的应用

博弈的不同子问题侧重点各不相同，以雷达攻防中的策略为例分析其在资源配置中的应用。雷达攻防主要为干扰方的五项策略和反干扰的四项策略进行攻防配置，构建攻防双方的关系矩阵求解。在已知双方资源的关系情况下，求解出战场的纳什均衡即可直观了解场上的敌我势力对比，得出在混合策略情况下双方的优劣势。

当进行单个雷达攻防的时候，由于双方的策略集分为 5 和 4，所以构建一个 4*5 的矩阵进行求解，得到矩阵的内容后，估算出需要迭代的次数，对于 5*4 的规模迭代 10000 次可以获得置信度高的解集。而在进行雷达群攻防的时候，干扰方不变，抗干扰的部分分为多个雷达，因此必须对每个雷达都进行计算，所得的结果不仅仅是策略的混合， 还有雷

达选择上的混合，多了一个决策的维度。具体数学推导过程不在这里赘述，主要学习的是分析此类问题的一般方法和步骤是：先分析双方的决策空间，根据策略集构建双方关系矩阵，之后对矩阵进行迭代计算得到战场的纳什均衡，最后得出该战场的策略结果。

1.4 多智能体强化学习

1.4.1 综述

一个随机博弈可以看成是一个多智能体强化学习过程。其实这两个概念不能完全等价，随机博弈中假定每个状态的奖励矩阵是已知的，不需要学习。而多智能体强化学习则是通过与环境的不断交互来学习每个状态的奖励值函数，再通过这些奖励值函数来学习得到最优纳什策略。通常情况下，模型的转移概率以及奖励函数为止，因此需要利用到 Q-learning 中的方法来不断逼近状态值函数或动作-状态值函数。在多智能体强化学习算法中，两个主要的技术指标为合理性与收敛性。

合理性（rationality）是指在对手使用一个恒定策略的情况下，当前智能体能够学习并收敛到一个相对于对手策略的最优策略。收敛性（convergence）是指在其他智能体也使用学习算法时，当前智能体能够学习并收敛到一个稳定的策略。通常情况下，收敛性针对系统中的所有的智能体使用相同的学习算法。

考虑到博弈对手的特点，算法分为均衡学习（Equilibrium Learner）和最大回报学习（Best Response Learner）。均衡学习考虑的问题在于假设队友同样是理性思维，或者通过博弈或强化学习训练出的模型，则要确保在最坏的情况下获得最稳定的收益。而若对手是一个非完全理性的选手，如普通人类对手，策略则重于获取最高的收益，是一种贪心的行为，也往往可以获得最佳的效果，而不是如均衡学习中的求稳方式。

均衡学习主要算法有三种 Q 算法，分别是 MiniMax Q（1994 Littman）、Nash Q（1998 Hu, 2003 Hu）和 Friend-or-Foe Q （2001 Littman）；最大回报学习分为 Fictitious Play （1951 Brown）、Joint Action Learner （1998 Claus）、Infinitesimal Gradient Ascent (IGA) （2000 Singh）和 WoLF-IGA（2001 Bowling）。总的来说，均衡学习可以保证最坏情况下的 payoff，但是部分算法可能存在多个纳什解，在对手采用非理性策略时容易获得少的收益，使得纳什均衡点并不是一个好的策略。而最大回报学习则可以有效利用对手的非理性行为获得更高的 payoff，同时也可以应对 payoff 矩阵动态变化的情况，但是利用敌人次优解的行为也容易被对手加以利用。

1.4.2 Minimax Q

Minimax-Q 算法[3]应用于两个玩家的零和随机博弈中。Minimax-Q 中 Minimax 指的是使用上一篇文章中的 minimax 方法构建线性规划来求解每个特定状态 s 的阶段博弈的纳什均衡策略。Q 指的是借用 Q-learning 中的 TD 方法来迭代学习状态值函数或动作-状态值函数。在两玩家零和随机博弈中，给定一个状态 s，则定义第 i 个智能体的状态值函数为

$$V_i^*(s) = \max_{\pi_i(s,\cdot)} \min_{a_{-i} \in A_{-i}} \sum_{a_i \in A_i} Q_i^*(s, a_i, a_{-i}) \pi_i(s, a_i), i = 1, 2$$

在执行 Q 学习的过程中，Q 表的表项$Q(s_t, a_t)$ 表示($state_t$, $action_t$) 的 value。更新函数通过学习率α进行更新：$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}))$ [3]。算法的流程图如下：

(1) Initialize $Q(s \in \mathcal{S}, a \in \mathcal{A})$ arbitrarily, and set α to be the learning rate.

(2) Repeat,

(a) Given the current state s, find the equilibrium, σ, of the matrix game $[Q(s,a)_{a \in \mathcal{A}}]$.

(b) Select action a_i according to the distribution σ_i, with some exploration.

(c) Observing joint-action a, reward r, and next state s',

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma V(s')),$$

where,

$$V(s) = \text{Value}_i\left(\left[Q(s,a)_{a \in \mathcal{A}}\right]\right).$$

理想情况，如果算法能够对每一个状态-动作对访问无限次，那么该算法能够收敛到纳什均衡策略。但是在上述算法中存在几个缺点：

(1)需要不断求解一个线性规划，这将造成学习速度的降低，增加计算时间。

(2)智能体 需要知道所有智能体的动作空间，这个在分布式系统中将无法满足。

(3)只满足收敛性，不满足合理性。Minimax-Q 算法能够找到多智能体强化学习的纳什均衡策略，但是假设对手使用的不是纳什均衡策略，而是一个较差的策略，则当前智能体并不能根据对手的策略学习到一个更优的策略。该算法无法让智能体根据对手的策略来调节优化自己的策略，而只能找到随机博弈的纳什均衡策略。这是由于 Minimax-Q 算法是一个对手独立算法（opponent-independent algorithm），不论对手策略是怎么样的，都收敛到该博弈的纳什均衡策略。就算对手采用一个非常弱的策略，当前智能体也不能学习到一个比纳什均衡策略更好的策略。

1.4.3 Nash Q

Nash Q-Learning 算法是将 Minimax-Q 算法从零和博弈扩展到多人一般和博弈的算法[5]。在 Minimax-Q 算法中需要通过 Minimax 线性规划求解阶段博弈的纳什均衡点，拓展到 Nash Q-Learning 算法就是使用二次规划求解纳什均衡点，具体求解方法后面单独开一章讲解。Nash Q-Learning 算法在合作性均衡或对抗性均衡的环境中能够收敛到纳什均衡点，其收敛性条件是，在每一个状态 s 的阶段博弈中，都能够找到一个全局最优点或者鞍点，只有满足这个条件，Nash Q-Learning 算法才能够收敛[5]。与 Minimax-Q 算法相同，Nash Q-Learning 算法求解二次规划的过程也非常耗时，降低了算法的学习速度。其算法流程如下：

Initialize:
Let $t = 0$,
For all s in S, a^1 in A^1, and a^2 in A^2,
let $Q_t^1(s, a^1, a^2) = 1, Q_t^2(s, a^1, a^2) = 1$
initialize s_0
Loop
Choose action a_t^1 based on $\pi^1(s_t)$, which is a mixed strategy Nash equilibrium solution of the bimatrix game $(Q^1(s_t), Q^2(s_t))$.
Observe r_t^1, r_t^2, a_t^2, and s_{t+1}
Update Q^1, and Q^2 such that

$$Q_{t+1}^1(s, a^1, a^2) = (1 - \alpha_t)Q_t^1(s, a^1, a^2) + \alpha_t[r_t^1 + \beta\pi^1(s_{t+1})Q_t^1(s_{t+1})\pi^2(s_{t+1})]$$
$$Q_{t+1}^2(s, a^1, a^2) = (1 - \alpha_t)Q_t^2(s, a^1, a^2) + \alpha_t[r_t^1 + \beta\pi^1(s_{t+1})Q_t^2(s_{t+1})\pi^2(s_{t+1})]$$

where $(\pi^1(s_{t+1}), \pi^2(s_{t+1}))$ are mixed strategy Nash solutions of the bimatrix game $(Q^1(s_{t+1}), Q^2(s_{t+1}))$
Let $t := t + 1$

Nash Q 算法维护了多个玩家的回报矩阵，多个 Q 表可能存在多个 Nash 解，因此难以收敛。

1.4.4 Friend-or-Foe Q

Friend-or-Foe Q-Learning（FFQ）算法也是从 Minimax-Q 算法拓展而来[2]。为了能够处理一般和博弈，FFQ 算法对一个智能体 i，将其他所有智能体分为两组，一组为 i 的 friend 帮助 i 一起最大化其奖励回报，另一组为 i 的 foe 对抗 i 并降低 i 的奖励回报，因此

$$V_i(s) = \max_{\pi_1(s,\cdot),\cdots,\pi_{n_1}(s,\cdot)} \min_{o_1,\cdots,o_{n_2} \in O_1 \times \cdots \times O_{n_2}} \sum_{a_1,\cdots,a_{n_1} \in A_1 \times \cdots \times A_{n_1}}$$

对每个智能体而言都有两组。这样一个 n 智能体的一般和博弈就转化为了一个两智能体的零和博弈。其纳什均衡策略求解方法如下所示:

假设对方是 friend，则需要保证选取最优策略使得对方收益，而若对方是 foe，则需要保证选取最优策略使得对方吃亏。假如 game 存在多个合作均衡点或对抗均衡点，这些点对应的 value 相同。针对 friend 和 foe 的策略如图:

$$Q_i[s, a_1, \ldots, a_n] := (1 - \alpha_t) Q_i[s, a_1, \ldots, a_n] +$$

$$\alpha_t \left(r_i + \gamma \mathsf{Nash}_i(s, Q_1, \ldots, Q_n) \right)$$

假如对手是friend:

$$\mathsf{Nash}_1(s, Q_1, Q_2) = \max_{a_1 \in A_1, a_2 \in A_2} Q_1[s, a_1, a_2]$$

假如对手是foe:

$$\mathsf{Nash}_1(s, Q_1, Q_2) = \max_{\pi \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} \pi(a_1) Q_1[s, a_1, a_2]$$

1.4.5 Fictitious Play 和 Joint Action Learner

在博弈论中，Fictitious Play 是乔治·W·布朗首先提出的学习规则。在其中，每个玩家都假定对手正在玩静止（可能是混合）策略[4]。在每一轮中，每个玩家因此最好地响应其对手的经验频率。如果对手确实使用固定策略，这种方法当然是足够的，而如果对手的策略是非静止的，那么这种方法是有缺陷的。例如，对手的策略可能取决于虚构玩家的最后一步行动。

在虚拟游戏中，严格的纳什均衡是吸收状态。也就是说，如果在任何时间段所有球员都发挥纳什均衡，那么他们将在随后的所有轮次中这样做。此外，如果虚拟游戏收敛于任何分布，那么这些概率对应于基础游戏的纳什均衡。

这种近似算最佳策略的方法，随着迭代次数的增加，各动作选取的频率会逼近最佳策略，从列视角看 V 的下界，用行视角看 V 的上界，二者不断迭代从而逼近真实的 V。

Joint Action Learner 借鉴了 Fictitious Play 的思想，对对手的行为进行建模，具体分为一般的 JAL 策略和条件性的 JAL 策略。

1.4.6 IGA 和 WoLF-IGA

IGA 和 WoLF-IGA 都是基于策略梯度的算法。在两个玩家中，row player 以α概率选择第一行动作，1-α概率选择第二行动作；col player 以β概率选择第一列动作，1-β概率选择第二列动作。IGA 算法中，双方同时采用 IGA 算法更新策略，则双方均收敛至纳什策略或不收敛但是均值是纳什 Value。

WoLF-IGA 中获胜或优秀策略的含义是指当前策略的累积预期奖励大于当前玩家纳什均衡策略和其他玩家实用当前策略的累积预期奖励。当前策略获胜时则谨慎缓慢学习，给其他智能体适应策略变化的时间；当前策略较差时，快速更新调整，使其能够快速调整适应其他智能体策略变化。WoLF-IGA 只适用于双智能体双动作矩阵博弈。WoLF-IGA 根据累计奖励关于策略的梯度来修正策略，其目的是使更新后的策略能够获得更大的奖励值。核心公式如下：

$$p_1(k+1) = p_1(k) + \eta\alpha_1(k)\frac{\partial V_1(p_1(k), q_1(k))}{\partial p_1}$$

$$q_1(k+1) = q_1(k) + \eta\alpha_w(k)\frac{\partial V_w(p_1(k), q_1(k))}{\partial q_1}$$

$$\alpha_1 = \begin{cases} \alpha_{\min} & \text{if } \mathrm{V_1(p_1(k), q_1(k)) > V_1(p_1^*, q_1(k))} \\ \alpha_{\max} & \text{otherwise} \end{cases}$$

$$\alpha_2 = \begin{cases} \alpha_{\min} & \text{if } \mathrm{V_2(p_1(k), q_1(k)) > V_1(p_1(k), q_1^*} \\ \alpha_{\max} & \text{otherwise} \end{cases}$$

WoLF-IGA 算法的难点在于需要已知大量信息。其信息包括自身的奖励矩阵、其他玩家的策略以及自己的纳什均衡。虽然智能体知道自己的奖励矩阵，也会得到纳什均衡策略。但这样大量的已知信息导致这个算法并不是一个实用的算法，也不是一个分布式的算法。

该算法的收敛性条件为：双智能体双行动一般和矩阵博弈，且纳什均衡为纯策略或混合策略。

2.工作与分析

2.1 工作概述

当前阶段的工作是对 AAAI2018 的论文 MAgent[6]的架构分析和运行测试。MAgent 是一个为多智能体的训练强化进行模拟仿真的平台，和之前的平台相比，不再仅仅局限在单个智能体的强化和少数智能体的强化，而是达到成百上千的规模，为智能体的强化训练过程提供一整套的测试、分析、可视化等功能。

2.2 运行环境

MAgent 的运行环境是 Linux 或 OS X，本阶段采用的测试环境是 ubuntu18.04，作者宣称其对于 python2 和 3 都支持，但是实际测试只能在 python2.7 完美支持，python3.x 的部分包有不少冲突，特别是不能运行在最新的 python3.7 上。经过阅读源码和运行验证，需要配套的安装包有 python2.7 版本下的 matplotlib（需要支持 freetype 等），科学计算库 numpy 和 scipy，运行可视化界面时采用的 pygame，以上的依赖要求是最基本的，另外在进行深度学习训练的时候 python2 下的 tensorflow 和 theano 也是必不可少的，虽然以上的库在 anaconda 的环境下可以同时获取，但是出现了版本支持上的错误，所以我还是采用了 pip 安装所有版本的库。

2.3 平台概述

MAgent 运行的环境设定是一个方块的世界，由智能体和墙组成，多智能体的控制方式为成群控制，成群的智能体拥有同样的总体属性和控制算法。智能体的主要属性有高度、宽度、速度和生命值等等，在类似的游戏规则下可以适当扩充，而这些是必要的属性。多智能体的决策行为包括移动、攻击和转变，通过设定行为的范围可以进一步划分为多个子动作，作为决策的元过程。在进行强化学习的训练过程中，奖励机制可以由测试者进行自行指定，而在进行 DQN 网络构建等过程中的奖励函数也由测试者自行进行调试。在本项目中，主要采用的基线算法是 DQN、DRQN 和 A2C 的 Tensorflow 和 Mxnet 实现，并在 MAgent 的环境下测试出 DQN 具有最佳的运行效果。由于核心的 train 代码需要在 GTX1080Ti 上运行约一天左右的时间，当前尚未具备硬件测试条件，所以当前仅仅是运行了测试可视化的 show_battle_game 模块。

2.4 论文解读

论文 MAgent 中认为平均场论用于解决数百万级别的 Agent 问题可以极大简化交互模式，降低计算量的同时达到较好的实验效果。在处理大规模智能体学习的时候，把多体问题抽象成二体问题是一种有效的方法。首先论文用 MF-Q（Mean-Field Q）求解了物理领域的伊辛模型(ising model)。在一个混合式的合作竞争性战斗游戏中，研究人员证明了平均场 MARL 相对其他多智能体系统的基线获得了更高的胜率。由于大幅降低了计算量，他们的方法可以推广用于很多实际场景，比如终端通讯设备流量分配，互联网广告竞价排名，智能派单等大规模分布式优化场景中。

平均场论（Mean Field Theory，MFT）是一种研究复杂多体问题的方法。在物理学场论和机器学习的变分推断中，平均场论是对大且复杂的随机模型的一种简化。未简化前的模型通常包含巨大数目的含相互作用的小个体。平均场理论则做了这样的近似：对某个独立的小个体，所有其他个体对它产生的作用可以用一个平均的量给出，这样，简化后的模型对于每个个体就成了一个单体问题。

2.5 结果与分析

2.4.1 基础 API

基础 API 的 demo 是一个 MAgent 环境基础用户接口的展示，采用的是一个规则简单的捕食者和被捕食者的游戏规则，地图大小为 100*100，捕食者通过互相的协作来攻击被捕食者，运行的终端输出如下所示，展示了 250 个 step 下的捕食者和被捕食者的 reward 比较：

```
step: 30        predators' reward: 33   preys' reward: -44
step: 40        predators' reward: 32   preys' reward: -42
step: 50        predators' reward: 34   preys' reward: -45
step: 60        predators' reward: 46   preys' reward: -60
step: 70        predators' reward: 41   preys' reward: -55
step: 80        predators' reward: 48   preys' reward: -63
step: 90        predators' reward: 49   preys' reward: -65
step: 100       predators' reward: 52   preys' reward: -69
step: 110       predators' reward: 55   preys' reward: -71
step: 120       predators' reward: 58   preys' reward: -76
step: 130       predators' reward: 59   preys' reward: -78
step: 140       predators' reward: 54   preys' reward: -71
step: 150       predators' reward: 57   preys' reward: -76
step: 160       predators' reward: 53   preys' reward: -70
step: 170       predators' reward: 56   preys' reward: -74
step: 180       predators' reward: 57   preys' reward: -76
step: 190       predators' reward: 58   preys' reward: -77
step: 200       predators' reward: 54   preys' reward: -71
step: 210       predators' reward: 55   preys' reward: -74
step: 220       predators' reward: 56   preys' reward: -75
step: 230       predators' reward: 59   preys' reward: -78
step: 240       predators' reward: 56   preys' reward: -74
step: 250       predators' reward: 57   preys' reward: -76
```

可以看到在迭代次数不断增加的过程中，总体上捕食者的能力得到了强化，伴随一定的 reward 波动。

2.4.2 战场游戏概况

Show_battle_game 展示的战场游戏是一个在 2018 年开源的多智能体模拟平台[6]，两个军队，各拥有 64 个同质的多智能体，通过互相协作来消灭敌人获得更高的奖励，采用的行动分为移动和攻击。默认的奖励机制设定如表 1 的战场信息列表，可以根据用户自己的需求修改 show_battle_game.py 中的各个参数值。而系统设定的 AI 学习策略 MF-Q 和 MF-AC 分别在两只军队上使用，场景设定是经典多智能体问题的战场 mean field，算法分别采用 Q 函数和 Actor-Critic 机制。Q 函数是通过构建深度 Q 网络实现，而 Actor-Critic 维护了一个随机决策的 Actor 函数和对决策进行打分评判的 Critic 函数，两种不同的更新状态机制。在实际测试结果中，MF-Q 和 MF-AC 的表现要明显优于其他的策略，而 MF-Q 甚至是达到了 80%以上的胜率。

战场触发事件	奖励惩罚机制(+为奖励，-为惩罚)
士兵位置移动	-0.005
攻击敌方士兵	+0.2
击杀一个敌方士兵	+5
攻击一个空格	-0.1
被攻击或击杀	-0.1

表 1.battle 的游戏参数设定(事件和其对应奖惩的机制表)

2.4.3 战场游戏测试

战场游戏的 demo 是人类测试者通过图形化的界面和一个自我迭代 2000 次的 AI 进行对战，AI 采用 MF-Q 和 MF-AC 算法。初始状态下双方各有 12 个士兵进入场地，由同样的算法进行随机的 50 次实验，之后玩家可以获得一次放置下一波 100 个士兵的机会，玩家放置后继续由 AI 的算法对移动和攻击做出决策。之后每隔 50 次玩家都有一次放置 100 个士兵位置的机会，直到 500 次后进入完全的 AI 双方博弈状态。人类在 1000 轮的决策中仅仅干预了 10 次的棋子放置位置，其余由自我迭代 2000 次的 AI 进行决策，而测试结果体现出 AI 对于战场资源配置的强大能力，也展示了 MF-Q 在其战场上的实用性。

下图 5，6，7 分别展示出初始状态 AI 的随机决策位置和前 50 次决策，由于是相同算法所以几乎没差别，开始人类可能可以通过局部的观察获取局部的领先，但是很快就会在全局的把控中落于下风，最终输给 AI。

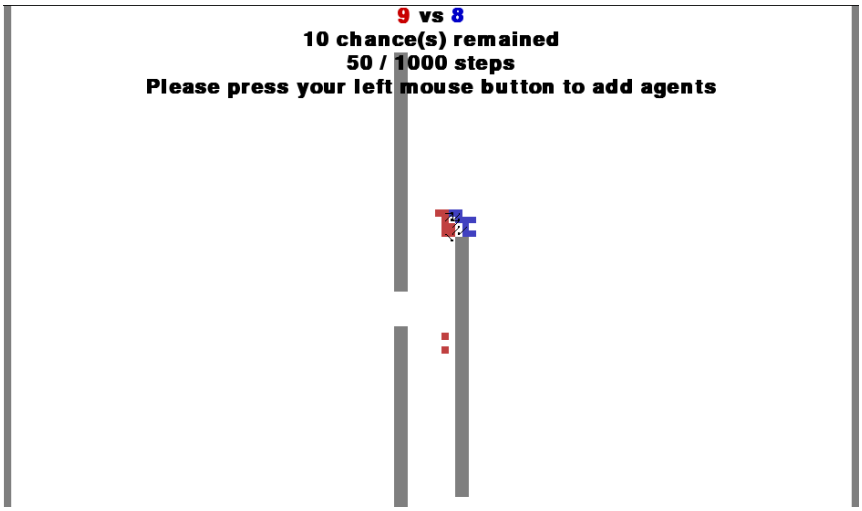


图 5.初始 AI 自身决定初始位置和前 50 次的比比分

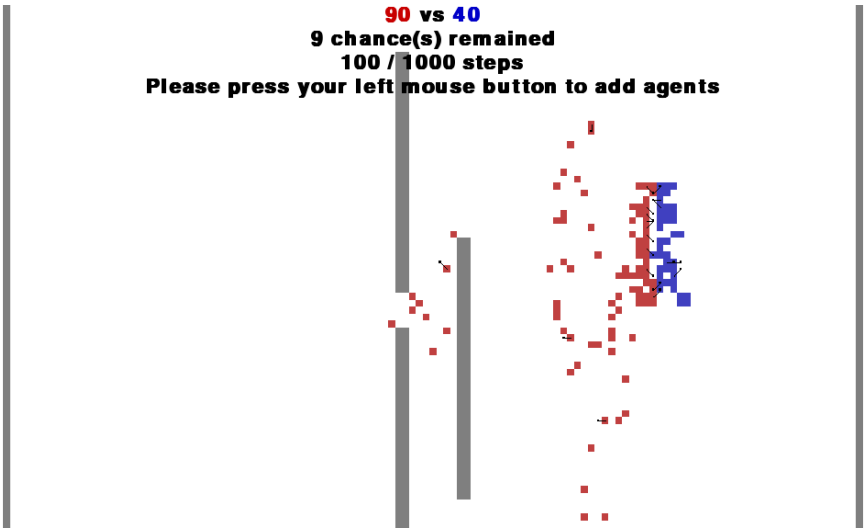


图 6.人类通过初始观察获得一定场面的优势

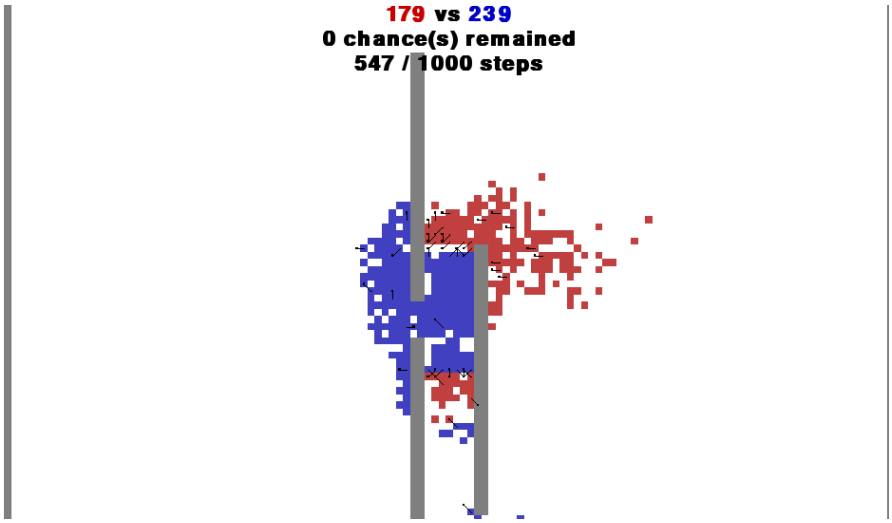


图 7.人类很快在决策上跟不上 AI 并陷入被动，最终输掉比赛

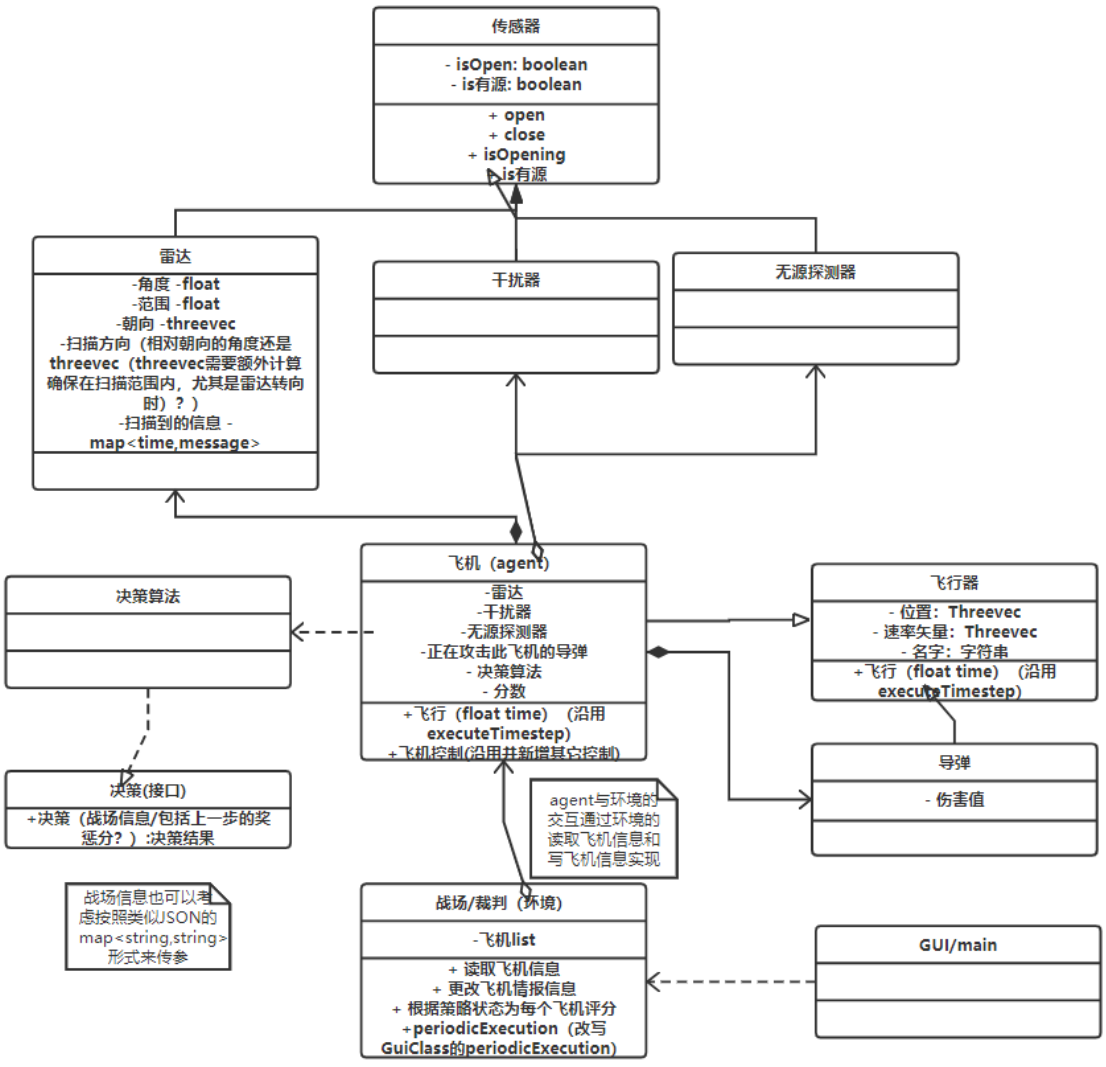
3.设计思路

3.1 概要设计

3.1.1 系统描述

系统主要基于一个飞行器的模拟平台，为其加入雷达干扰机对抗攻防的部分，攻方的飞机配备有干扰器，干扰器有相应的策略属性，即干扰方式；守方的雷达配有抗干扰策略。通过攻防两方对于干扰器和抗干扰策略的配置，验证资源配置算法的有效性。

3.1.2 系统图解



系统围绕战场环境和攻防双方展开，攻防双方的活动载体是飞机，飞机也作为 Agent 单位，飞机通过子类飞行器进行在系统中的运动，攻方配有干扰器，守方配有雷达。

战场环境主要为 Agent 提供信息，维护所有 Agent 的列表，进行全局判定，作为各项奖惩和事件的管理依据。

飞机作为场上的 Agent，由飞行器、导弹两部分构成，而进攻方的飞机会添加干扰器，防守方添加雷达。Agent 根据算法标签控制其调用的算法，并接受战场环境给予的分数，作为决策算法的依据。

算法部分包括战场参数计算和飞机调用的算法。战场上根据对抗关系的矩阵可以算出混合纳什均衡，作为决策算法的依据，或者将其策略空间参数的混合比作为一个检验算法的依据。

3.1.3 模块设计（我的部分）

我的工作围绕决策算法展开，包括进攻方和防守方。我针对的问题有如下四种思路：单雷达的单智能体对抗、雷达阵的单智能体对抗、单雷达的多智能体对抗、雷达阵的多智能体对抗。

单雷达和雷达阵的单智能体对抗是作为后两种思路的基础，主要用于验证强化学习算法在与纳什均衡对抗时的效果，为多智能体的算法提供借鉴和对比分析的依据。单智能体的算法主要通过设计一个 Q 函数或 AC 函数，在迭代后与纳什均衡对抗，或让 AC 算法与 Q-learning 对抗。

如果双方各有多架飞机，多对多的对抗则需要用到多智能体的算法，多个友方飞机间是 friend 关系，和敌方飞机是 foe 的关系。通过多智能体强化学习算法，改良 Minimax-Q 和 Nash-Q 的策略，或采用策略梯度相关的算法，与多智能体的混合纳什均衡对抗，算法之间也可以进行对抗。

3.2 详细设计

3.2.1 算法设计

（1）纳什均衡计算

对于雷达对抗矩阵的纳什解，需要计算出各个混合概率系数，而往往不是线性可解的问题，比较适用于计算机模拟的方法是迭代法。

（2）单智能体算法

Q-learning：

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-}=\theta$
For episode = 1, M **do**
  Initialize sequence $s_1=\{x_1\}$ and preprocessed sequence $\phi_1=\phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t=\text{argmax}_a Q(\phi(s_t),a;\theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1}=s_t,a_t,x_{t+1}$ and preprocess $\phi_{t+1}=\phi(s_{t+1})$
    Store transition $\left(\phi_t,a_t,r_t,\phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j,a_j,r_j,\phi_{j+1}\right)$ from D
    Set $y_j=\begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j+\gamma \max_{a'} \hat{Q}\left(\phi_{j+1},a';\theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j-Q\left(\phi_j,a_j;\theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q}=Q$
  End For
End For

Actor-Critic：

Actor–Critic with Eligibility Traces (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; step sizes $\alpha^{\boldsymbol{\theta}}>0$, $\alpha^{\mathbf{w}}>0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Initialize S (first state of episode)
  $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (d'-component eligibility trace vector)
  $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ (d-component eligibility trace vector)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R+\gamma \hat{v}(S',\mathbf{w})-\hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma\lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}}+I\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma\lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}}+I\nabla_{\boldsymbol{\theta}}\ln\pi(A|S,\boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w}+\alpha^{\mathbf{w}}\delta\mathbf{z}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}+\alpha^{\boldsymbol{\theta}}\delta\mathbf{z}^{\boldsymbol{\theta}}$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

（3）多智能体算法

暂定为 Minimax Q、Nash Q 和 DDPG

3.2.2 评价标准

算法之间的两两对抗和与混合纳什均衡的对抗（迭代多次，绘制图）。

前期进行单智能体对抗，后期进行多智能体对抗。

3.2.3 模块交互

算法模块从战场模块获取奖励信息，并给 Agent 提供算法接口，直接反馈决策结果。

[1] P. D. CLIVE，, JEFFREY A JOHNSON and M. J. LOSS, *Advanced Framework for Simulation, Integration and Modeling (AFSIM)* (2015).

[2] M. L. LITTMAN, *Friend-or-Foe Q-learning in General-Sum Games*, (2003).

[3] M. L. LITTMAN, *Markov games as a framework for multi-agent reinforcement learning*, (1994).

[4] M. V. MICHAEL BOWLING, *Multiagent Learning Using a Variable*, (2001).

[5] S. SINGH, *Nash Convergence of Gradient Dynamics in General-Sum Games*, (2000).

[6] Y. YANG, R. LUO, M. LI, M. ZHOU, W. ZHANG and J. WANG, *Mean Field Multi-Agent Reinforcement Learning*, *AAAI*, 2018.

[7] 李鸿, 吴嗣亮 and 杨春山, *对策论在雷达反干扰作战中的应用*, (2008), pp. 10-12.