Reinforcement Learning Environment for Tactical Networks: Multi-Agent based Scenario Generation

Thies Möhlenhof, Norman Jansen, and Wiam Rachid
Fraunhofer FKIE
GERMANY

{thies.moehlenhof, norman.jansen, wiam.rachid}@fkie.fraunhofer.de

ABSTRACT

Providing situational awareness is a crucial requirement and a challenging task in the tactical domain. Tactical networks can be characterized as Disconnected, Intermittent and Limited (DIL) networks. The use of cross-layer approaches in DIL networks can help to better utilize the tactical communications resources and thus improve the overall situational awareness perceived by the user. The specification of suitable cross-layer strategies (heuristics) which describe the rules for optimizing the applications remains a challenging task.

We previously introduced an architecture for a learning environment aimed for training decentralized, reinforcement learning (RL) agents which are supposed to improve the use of network resources in DIL networks by the use of cross-layer information [1]. Since for the training of these agents a large number of scenarios is needed, an additional tactical model was defined. The purpose of the tactical model is to generate scenarios with dynamically changing network conditions and dynamic information exchanges between the applications and thus build the basis for training the RL agents. The tactical model itself is also based on RL agents, which simulate military units in a gaming environment.

In this paper, we present this tactical model with experimental deep reinforcement agents placed in a tactical environment focused on controlling movement and communication in multi-agent cooperative games. The game is focused on multiple agents on a board to reach a common goal competing against an opposing team by communicating and moving in a two-dimensional space. We investigate how agents interact with each other and the environment to solve an episodic as well as a continuous task. Since the focus of this work is reinforcement learning on communication networks to enhance DIL communication networks later on, we present agents based on Proximal Policy Optimization [2] adapted to a cooperative multi-agent communication network problem. Moreover, resulting trajectories of this game should be used later on to train agents in a DIL setting.

1.0 INTRODUCTION

In network-centric warfare, to increase situational awareness it is essential to build up a *Command and Control Information System* (*C2IS*) from the tactical layer, since relevant data (e.g., positions) originate from this layer. Mobile networks on the tactical layer come with low data rates, short signal ranges and frequent disruptions and are therefore characterized as *Disconnected, Intermittent and Limited (DIL)* networks. They pose a major challenge in the usage of C2IS services, as they have very limited resources which must be used very efficiently.

Prior work [3] has shown that the utilization of the scarce communications resources can be improved if C2IS services are made network-aware in the sense that they can adapt their behavior to the dynamically changing network environment. This can be achieved by a cross-layer information exchange between the C2IS services and the tactical radio networks in combination with a strategy (heuristic) which is responsible to decide how to react to changed network conditions. By the cross-layer information, the heuristics are

enabled to adapt to changing network conditions by reallocating current available network resources to C2IS (or communication services) in order to prevent network congestion. The specification of suitable cross-layer strategies (heuristics) remains a challenging task. To overcome the need to specify rules manually, decentralized, machine learning based reinforcement agents shall be deployed which are supposed to learn more sophisticated strategies on their own based on experiences with the network environment.

Since for the training of these agents a large number of scenarios is needed, we defined an additional *tactical model*. The purpose of the tactical model is to generate scenarios with dynamically changing network conditions and dynamic information exchanges between the applications and thus build the basis for training the RL agents. The tactical model itself is also based on RL agents, which simulate military units in a war gaming environment. The goal of this approach is to create probabilistic scenarios with military realistic behavior instead of defining fixed scenarios and network setups manually. Thus, the course of actions the agents take will vary in each generated scenario in a probabilistic way, but will still follow some military strategies to target the final objective of winning the game. This shall combine the realism of hand-crafted (or recorded) military scenarios and the variance of scenarios generated based on simple probabilistic models (e.g., mobility models).

Other work [4] has shown that probability distributions can also be used to create scenarios. However, the results are not satisfying. They work well in simulation, but agents trained on these models perform poorly in the real world. In [5] a *mobility scenario generation and analysis tool* is developed, which offers different types of scenario generators and implements network simulators. However, network traffic generation is not part of the project. In [20] the authors concentrated on optimizing communication flows with the help of queuing strategies based on cross-layer information exchanges. To generate suitable scenarios, they proposed two statistical models for 1) creating dynamic user data flows and 2) creating dynamic network conditions. Based on a statistical model the network state switches between the states “Disconnected”, “Limited” and “Connected” with some probability, resulting in continuously changing network conditions, in contrast to randomly changing network conditions, which leads to more realistic network behavior.

In contrast to the statistical models described above, our approach deploys machine learning based agents in a simulation environment. The agents are supposed to emulate the behavior and strategies of military units and are thus expected to learn a more realistic behavior than statistical approaches. The Tactical Model presented in this paper is based on a multi-agent gaming environment with two teams competing against each other. Each agent in this environment is based on reinforcement learning and responsible for controlling one military unit. Challenges for the agents are to cooperate in their teams and to learn strategies to successfully compete with the other team. The actions of each agent are based on local observations, which complicates finding a common strategy for the team.

The paper is organized as follows: First, in Section 2 the technological background of our work is described. In Section 3 the architectural concept of the overall learning environment, which was introduced in [1] and builds the overall context of the work described in this paper, is shown. The Tactical Model, which is the focus of this paper, is presented in Section 4. The implementation and experiments to evaluate our concepts are shown in Section 5. Finally, Section 6 concludes and discusses further work.

2.0 BACKGROUND

In this section the background information for the next sections dealing with the Tactical Model is described.

In the most common Reinforcement Learning [7] setup a single agent interacts with an environment. Figure 2-1 shows how a single agent interacts with an environment. An environment can be a video game or a simulation with its own dynamics, rules and properties. The agent receives the current state (s_t) of the environment like the current frame of a video game and a reward signal (r_t) like the current score stating the

current value of the game state. Based on state and reward the agent applies actions (a_t) in the environment like moving the player to the right. This circle is repeated until the game ends and the agent learned how to win the game by collecting trajectories of state, reward and actions over time. To win a game the agent needs to optimise or accumulate as much as reward as possible to derive a game winning policy. In this work an Actor-Critic method is used. The Actor-Critic method consists of an actor which is the policy and a critic with which the policy is updated through time. The critic acts like a trainer for the policy, in other words it criticises the action selection of the actor. The action selection of the actor will be updated accordingly to the critic, to select actions with the highest probability of reward.

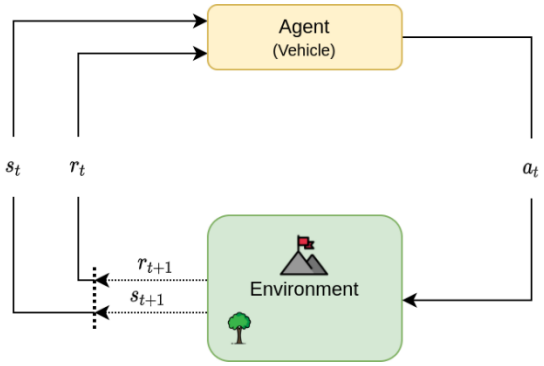

Figure 2-1: Reinforcement Learning

However, the tactical environment is a *Multi-Agent Reinforcement Learning (MARL)* setup. In this kind of setup, not a single agent interacts with one environment. There are multiple agents interacting with the same environment and changing the state of the environment with each individual action shown in Figure 2-2. This breaks the Markov Property that the next state needs to be predictable by the current state which holds in the single agent setup [8]. The dynamics of the environment can be interpreted as a *Decentralised Partially Observable Markov Process* (*Dec-POMP*) [9], which means that the state transition from one state to the next depends on all actions taken by all agents in the environment and not only on the actions taken by a single agent in an environment. In other words, the state transition shifts to a more complex probability distribution.

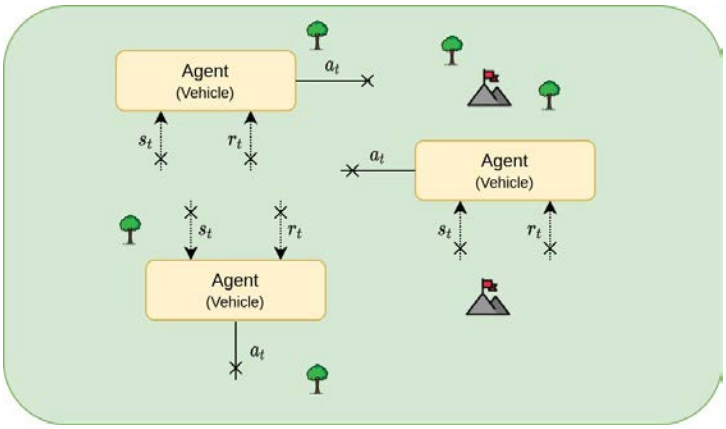

Figure 2-2: MARL

The tactical environment is a Dec-POMP problem. This kind of problem hypothesis can also applied to a DIL network, where a change on one node in the network can influence the whole network state. Moreover,

connections can get lost and messages are sent through the network or a transmission can fail. Moreover, training on a laboratory DIL setup with real applications is very time consuming, thus we come up with the idea to develop the tactical environment with the same or similar process properties to test and develop an RL algorithm to optimise such a process. As a by-product, movements and communication probability distributions can be derived and depend on events which are not completely random.

There are four main challenges to overcome. First, each agent needs to be able to observe its own local state. This are the features based on the current position and surroundings on the map. Features on the map are its own team, the opposing team, obstacles and goals which, when reached, emit a relatively small reward signal in contrast to winning the game. Second, agents need to be able to incorporate the local state of other agents. Whereby, agents can leave or re-join the network and thereby are not directly observable anymore and cannot share their own state. Third, agents need to be trained to cooperate by sharing only messages, local map observations and network information. Additionally, they have to apply the right actions e.g., movement to North, East, South or West, send commands to other agents and disable a member of the opposing team, which would result in winning the game and a high reward signal.

Most RL architectures are based on observations presented as images of games states. Images can be directly represented as matrixes, thus straight forward be used as input of a neural network. Since we are concerned with network structures, we depend on a way to encode theses dynamic networks into a vector representation. A vector representation of dynamic networks can be derived through the use of *Graph Neural Networks (GNN) [10]*. GNNs offer the practicability to work well with graph structured data at scale, since they do not depend on encoding the graph structure manually into flat structures as e.g., adjacency matrix. The data itself creates a computational graph based on the local neighbourhood of each node, which forms the neural network structure. This approach enables learning on large graphs or networks e.g., social networks [11], recommender systems [12] or proteins [13].

Combining reinforcement learning and Graph Neural Networks is a relatively novel approach, since research in this area has only just begun. [14] proposes a MARL algorithm based on policy gradient and centralised critic and emphasises the problems of Q-Learning in a continuous setting, with a smaller sampling rate than policy methods, but tends to fail at growing the number of agents. Moreover, a similar approach is proposed in [15] and named *Counterfactual Multi-Agent* (*COMA*). But both approaches fail if the number of agents grows significantly, since a centralised agent needs to compute a value function on all observations and actions of each agent. [16] divided applications of RL in combination with GNN in Learning-for-Consensus, Learning-for-Communication and MARL with GNNs. These categories need to be considered in combination, because there is a high correlation between consensus and communication to form a capable MARL agent. Moreover, they state a framework of how agents can communicate and form groups to enable structured communication. In [17] the graph structure is encoded through a graph network and forwarded to a Q-Learning approach. Moreover, communication is restricted to the three-hop neighbourhood, whereas learning and consensus is still stable. Authors in [18] combined a soft and hard attention mechanism to learn an adaptive and a dynamic attention value and outperformed on two benchmarks. All these approaches show very promising results and advantages by combining RL and GNN.

3.0 ARCHITECTURAL CONCEPT OF THE LEARNING ENVIRONMENT

We previously introduced an architecture for a learning environment (cf. [1]) aimed for training decentralized, reinforcement learning (RL) agents which are supposed to improve the use of network resources in DIL networks by the use of cross-layer information. According to the architecture (cf. Figure 3-1 and [1]), C2IS Services deployed in a Cross-Layer Environment, which emulates the C2IS systems and DIL network (including tactical router and radios), are supported by *Cross-Layer Agents* (*CL-Agent*s). For this purpose, each instance of the C2IS service is augmented with a CL-Agent. A CL-Agent obtains network status information from the DIL environment (e.g., from the tactical router) and provides its C2IS service

with policies regarding the use of the scarce network resources. E.g., a CL-Agent may assign its BFT service a maximum data rate based on the current or future network status (e.g., connection quality). With help of these policies the BFT service is able to dynamically adapt the sending rate of position messages in a way that it does not produce more data (data rate) than it was assigned to in the policy.

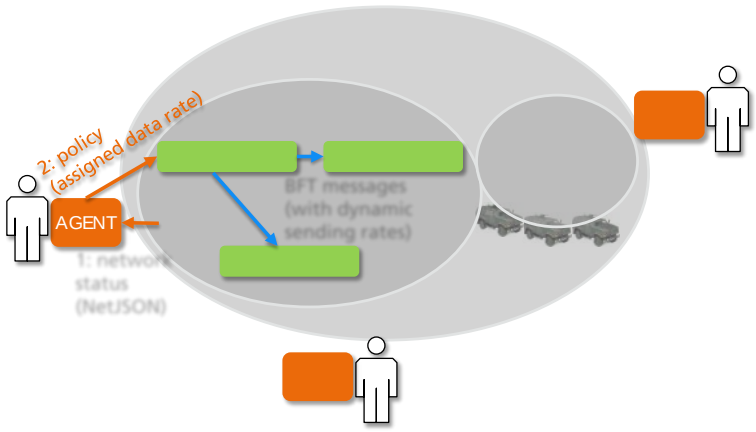

Figure 3-1: Learning environment (high-level architecture).

Since for the training of these agents a large number of scenarios is needed, an additional *Tactical Model* was defined. The purpose of the Tactical Model is to simulate the behavior of the military units (e.g., movements and actions for sending of messages) and thus generate scenarios with dynamically changing network conditions and dynamic information exchanges between the applications. These scenarios will build the basis for training the Cross-Layer RL agents and shall ensure that the agents do not learn optimizations for specific scenarios, but are enabled to learn a more generalized behavior. The Tactical Model itself is also based on RL agents, which simulate military units in a gaming environment.

While the focus in [6] was on the overall architecture of a reinforcement learning environment for tactical networks, this paper will go deeper into the Tactical Model, which will be described in more detail in the following sections.

4.0 APPROACH

In this section we introduce a concept (*Tactical Model*) for generating military tactical scenarios based on a multi-agent gaming environment and reinforcement learning (RL) agents. The purpose of the RL agents is to simulate the movements and other actions of military units in a multi-agent gaming environment. For this purpose, the agents are trained according to military inspired games. By execution of these trained agents in the environment, probabilistic tactical scenarios can be generated, which can be used later on as a basis for the training of Cross-Layer (CL) agents which are supposed to optimize the communication in DIL networks. Moreover, we can experiment in an environment with the same properties, but easier to interpret and faster to train.

4.1 High-level architecture of Tactical Model

The overall high-level RL learning environment is described in [1]. According to the architecture two different kinds of reinforcement agents are deployed (*cross-layer agents* and *tactical agents*). Each cross-layer agent is responsible for the cross-layer optimization of the C2IS system of a unit. Each tactical agent is responsible to simulate the movements and other actions of one military unit. For training these agents two environments are needed, a *cross-layer environment* and a *tactical environment*, respectively. Since the focus

in this paper is on the Tactical Model, the high-level architecture of Tactical Model (which is part of the overall high-level architecture) is recapped here and shown in Figure 4-1.

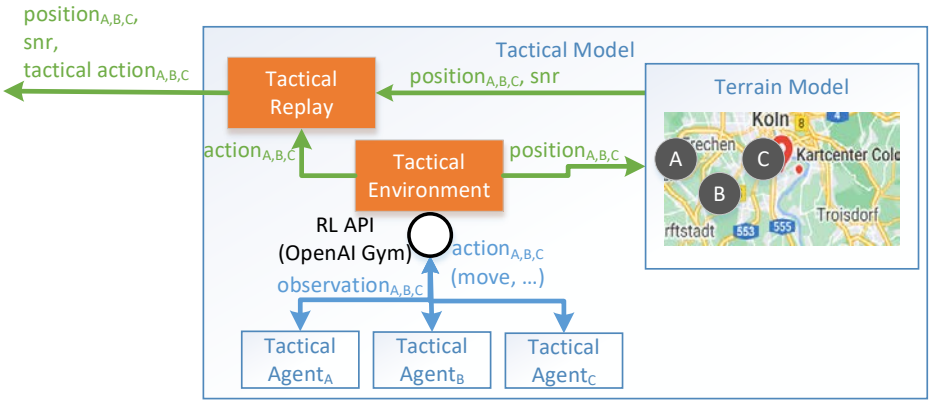

Figure 4-1: High-level architecture of Tactical Model.

Tactical Model consists of a *Tactical Environment*, *Tactical Agents*, a *Terrain Model* and the *Tactical Replay* component (cf. Figure 4-1). (Virtual) military units are controlled by tactical agents and move on a map according to selected (war) games, which are simulated in the tactical environment. To train the CL agents, we depend on a large number of exemplary scenarios. We generate such scenarios by games inspired by war-gaming. Each unit is controlled and represented individually by a separate tactical agent. Consider that in Figure 4-1 exemplarily three units / tactical agents are shown. Each tactical agent is responsible to simulate the movements and other actions of one military unit. The tactical agents are connected to the tactical environment by a *Reinforcement Learning API* (*RL API*) based on OpenAI Gym [19] (cf. *RL API* in Figure 4-1). The actions taken by the tactical agents are recorded and stored in a database for Tactical Replay. A Terrain Model is responsible for calculating the radio signal strengths based on the current positions of units by simulating the physical radio propagation between the units. For this purpose the Terrain Model should take the terrain and obstacles into account. The calculated signal-to-noise ratios (SNR) between each pair of units and the corresponding positions are also stored in the database of Tactical Replay.

Tactical Replay is the foundation for the CL Environment since it is used to generate suitable scenarios for training the CL agents. For this purpose, Tactical Replay provides different recorded runs of the execution of tactical agents and thus in overall provides probability distributions of movements on the map, when a command is sent, the positions of units and the corresponding SNR values describing the connection qualities between units as calculated by Terrain Model. These values are used by the CL environment to emulate the C2IS applications (with help of the actions/commands) and the tactical network (by providing positions and SNR values to the network emulator) (see [1] for details).

In the following subsections, we will describe the tactical environment and the tactical agents in more detail. Tactical Replay and Terrain Model are not needed before the Tactical Model is integrated with the CL Environment and will be described in a subsequent paper.

4.2 Tactical Environment

The Tactical Environment shown in Figure 4-2 is a discrete, multi-agent gaming environment, where more than one agent or actor is interacting with the same environment. As a result, the transition from one state to the next state not only depends on the current observation and action of a single agent, it depends on all actions chosen by all agents at the same time. In this environment an agent acts like the commander and driver of a tank or unit at the same time. Units are separated into a blue and a red team of variable size. The

blue and red team then proceed two different objectives and will compete against each other. One objective is to reach certain *Points of Interest (PoI)* on the map where a small reward signal is emitted, the other is to disable the majority of the opposing team members and win the whole game, which will be remunerated with a very high respectably low reward signal in contrast of reaching a PoI. Two types of reward enable to roll out different situations and experiments. Moreover, having the CL Environment in mind, an agent has to differentiate between high and low rewards in the long run.

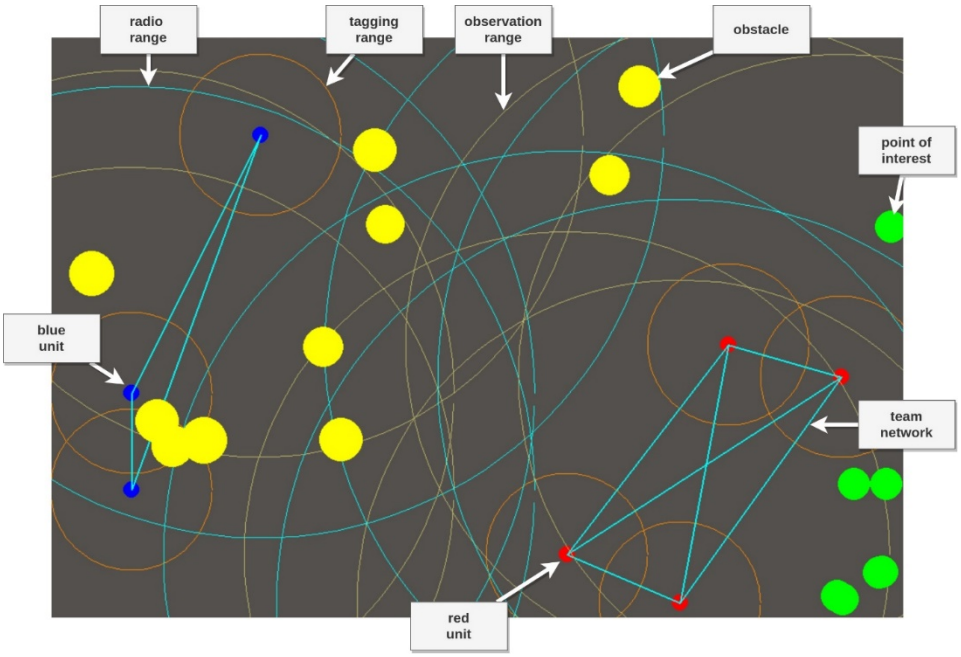

Figure 4-2: Visualization of Tactical Environment

4.2.1 Sensor Model

A unit perceives its surroundings on the map by (virtual) visual sensors, which we call local observation, limited by a circle around the centre of the vehicle (*observation range*). This approach mimics the real world where a person on top of a vehicle observes its surroundings. There are different types of observations on the map. Units observe obstacles in *observation range* (which block the way and the line of side of the units), the own teammates in their range, the opposing teammates in their range and *points of interest*, where a small reward signal is emitted if reached. Moreover, the unit will perceive the local observations of other teammates if they are in the same network *(team network)*. A connection between two units is established if the distance between two units is under a certain threshold (*radio range*). Moreover, a unit will share the intention to tag an opposing unit and command messages to other units. A summary of observations is shown in Table 4-1.

Table 4-1: Observations of a tactical agent.

Observation
Obstacles (if in observation range and in line of side)
Locations of own teammates in observation range (if not hidden by an obstacle)
Locations of opposing teammates in observation range (if not hidden by obstacle)
Points of interest in observation range (if not hidden by obstacle)
Local observations of other teammates (if in the same communication network)

Command Messages of other units (if in the same communication network, ordered by relative position)
Intention of tagging a unit (if in the same communication network, ordered by relative position)

4.2.2 Action Model

Each unit is capable of performing twenty actions. The decision which actions are executed is performed by an agent deployed on a unit. A unit is able to move in four celestial directions, tag another unit to take it out of the game and send a command to another unit. Tagging and command actions are executed in sections. These sections are created by splitting the surroundings into eight regions according to the celestial directions (North, North-West, West, South-West, South, South-East, East, North-East); thus, e.g., if an agent wants to disable an opposing team member which is located right from his point-of-view, he must learn to execute a tagging command in the correct sector. In this way we can see if an agent is capable to understand its local observations and act accordingly. Table 4-2 shows the whole actions space of a single unit in the environment.

Table 4-2: Action space of tactical agents.

Action
Move own unit in four directions: [North, West, South, East]
Tag reachable opponent in one of eight regions: [North, North-West, West, South-West, South, South-East, East, North-East]
Send command to a friendly unit in one of eight regions: [North, North-West, West, South-West, South, South-East, East, North-East]

4.3 Tactical Agent

In this section we will describe how an agent decides which action to choose in the next step. The basic setup is the congruent to the single agent setup. An agent receives the current local observations and a reward and decides what action to execute next.

At the local level, each agent perceives its local information. Local observations are: obstacles, teammates, opponents and points of interest limited to the observation radius around the unit. The local information defines the local state of each agent at each time step. This is a parallelism to the DIL optimization problem, where an agent's view is limited to its local (network) information and global information needs to be shared by or with other agents.

The local state is then merged with the other local observations of teammates if they are interconnected by the same communication network. How local and global observations are merged is learned behaviour, by a Graph Neural Network approach. GNNs have the advantage that a computational graph is formed on the network or data structure itself and is able to encode dynamic networks with leaving and joining nodes. This is in contrast to approaches with a flat representation of the graph structure, e.g., in adjacency matrices, where each node is a row and a column in this matrix, and connections are entries in the matrix. By this approach only a fixed size of nodes can be represented. By the use of GNN, the state of an agent is formed by a combination of its two-hop neighbours and is then further processed.

Figure 4-3 shows the process of merging local states by aggregating the local neighbourhood of a unit. The first *1 – Input Network* shows the input network with the goal to aggregate local states of other units in Unit 0. At first, shown in *2 - Local Neighbourhood*, nodes are sorted by their hop count with respect to Unit 0.

Moreover, self-loops are added to encode the local state of each node.

3 – Aggregation shows the computational graph created to encode the state in Unit 0. Each unit computes its own state by aggregating information from its direct local neighbours, which is then forwarded through a Neural Network and aggregated in the following unit. The process of aggregating and forwarding is recursively applied until the final Unit 0 is reached.

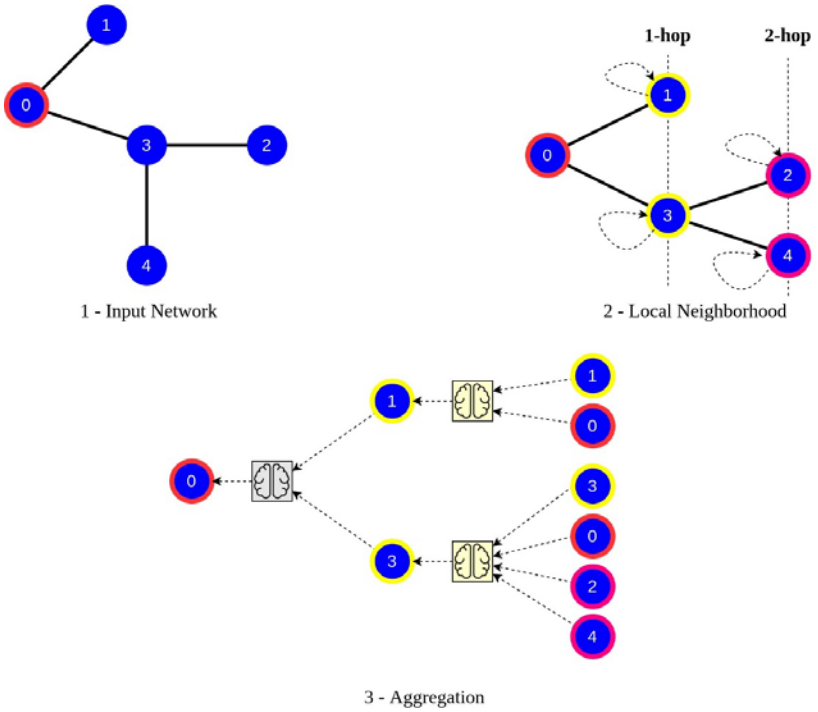

Figure 4-3: Graph Neural Network example.

With the help of GNN agents we are able to aggregate over the dynamic network structure to compute embeddings of the network to form the input for the Deep-RL algorithm. We choose the RL algorithm *Proximal Policy optimization (PPO)* [2] for optimization, with two separate, but similar neural networks architectures for critic and actor (cf. Figure 4-4).

All in all, the combination of GNN [6] and RL enables experiments in a multi-agent environment with a variable number of agents leaving and re-joining an interconnected network. We further investigate if this RL setup is able to learn strategies and is able to enhance performance over time.

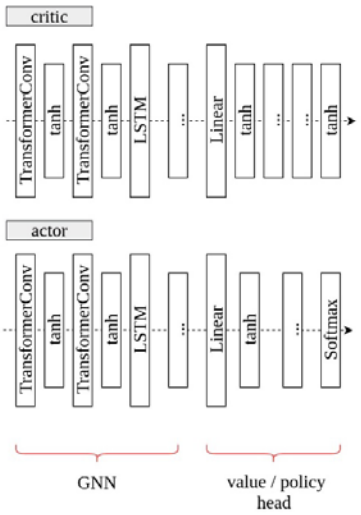

Figure 4-4: Neural Network Architecture, for actor and critic.

5.0 IMPLEMENTATION AND EXPERIMENTS

We evaluated this approach by creating an environment with blue and red teams of variational size between three and five. Each individual team size is chosen at random for each game. Furthermore, PoIs in the environment appear at start and reappear at random if reached by a unit or will accumulate in a specific region on the map. Reaching a PoI is rewarded with r=0.5 for the specific unit. If an agent tags another unit, the whole team will be rewarded with r=1, the other team with r=-1. If one third of a team is disabled, the game ends and the whole team will be rewarded with a reward of r=10 or r=-10 respectively. With this setup we can enforce movement of units, since a PoI is rewarded, but the main game objective needs to be reached to achieve the highest possible reward. Moreover, units will be enforced to avoid the opposing team. To prevent agents to learn based on a fixed map size and create more aggressive behaviours, we alternate between two board sizes randomly. We pre-trained teams to collect PoIs and disable opposing teams, which were not able to move. Later on, we expect and observed games to increase in game steps at average with a noisy reward signal, shown in Figure 5-1, based on the reward function.

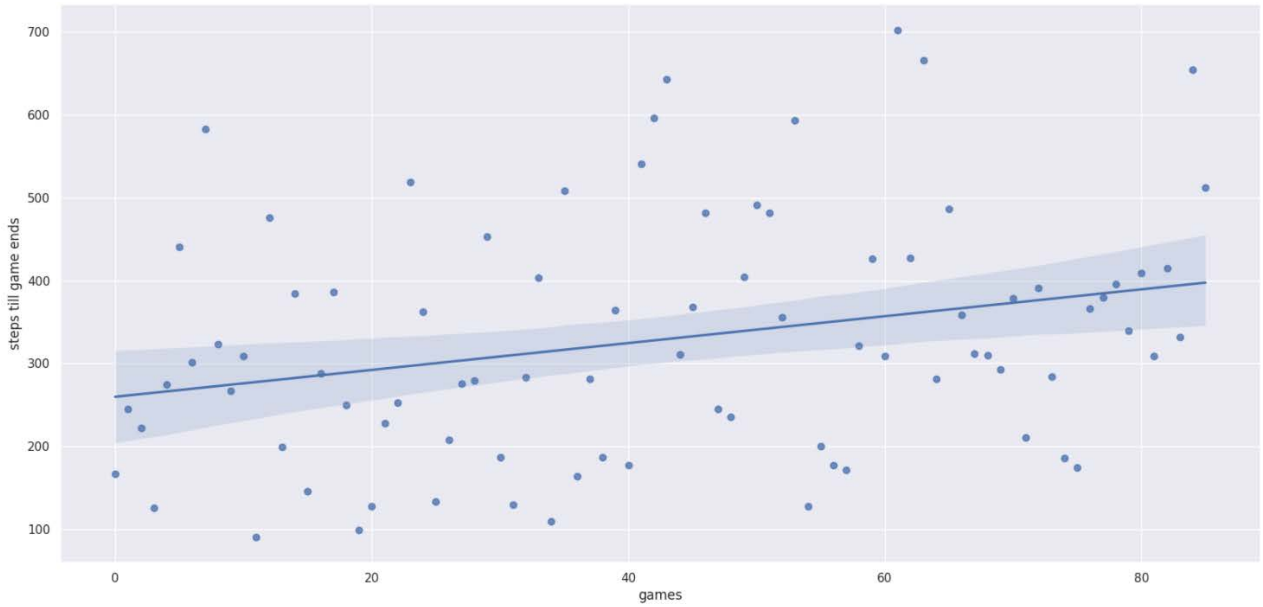

Figure 5-1: Steps to finish a game is increasing over played games.

This behaviour is desired since we want to generate trajectories for the CL environment, where longer episodes are useful. At evaluation we observed multiple times interesting behaviors like single agents' abandonment the team network (Figure 5-2 left) and movement towards the opposing team (Figure 5-2 right) after the opposing team becomes visible. Another observation was that teams are accumulating around PoIs. Analyzing if these behaviors are statistically significant, is part of further investigation.

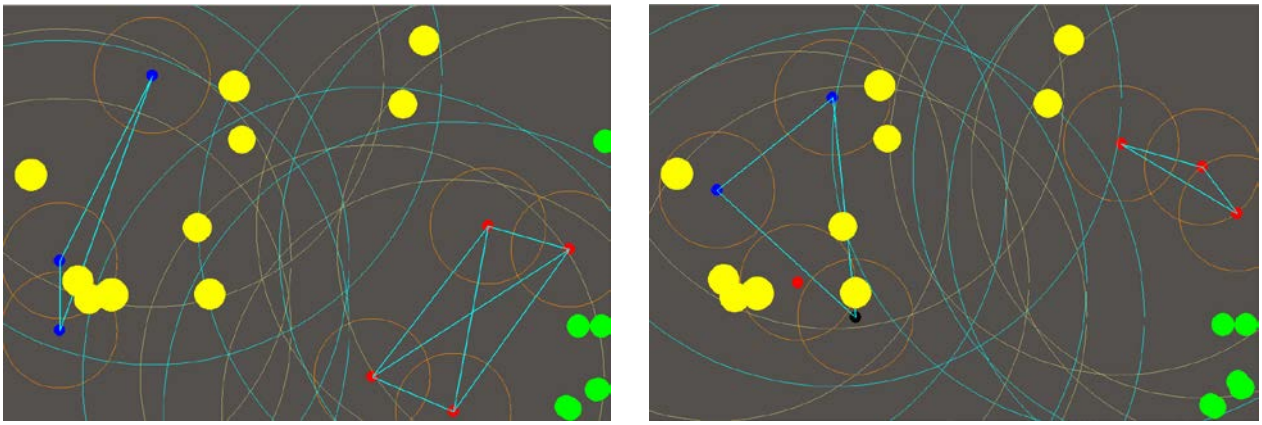

Figure 5-2: Units accumulate before PoIs and single unit breaks from team tagging opposing unit.

6.0 CONCLUSION AND FURTHER WORK

In this paper, we presented a concept (*Tactical Model*) for generating tactical scenarios based on deep reinforcement learning (RL) agents placed in a tactical gaming environment. The agents are focused on controlling movement and communication in multi-agent cooperative games, thus emulating strategies of military units. This shall lead to more realistic scenarios than can be achieved with simple statistical models. Since the view of every unit is limited to its local observations, we have designed the gaming environment in a way that the units are able to share information through a communication network to coordinate their strategies. With the use of a GNN architecture for a node level embedding we are able to encode this network information to train RL agents. Each game produces a sequence of movements and commands which are communicated between agents.

We will further investigate if these sequences are similar to other approaches for generating scenarios like [20] or real-world scenarios [21]. Moreover, we will compare learned behaviour over time to see if beneficial strategies will emerge. In addition, we plan to apply the same algorithm later on for network optimizations in DIL networks based on cross-layer information.

REFERENCES

[1] T. Möhlenhof, N. Jansen, and W. Rachid. "Reinforcement Learning Environment For Tactical Networks," *International Conference on Military Communications and Information Systems (ICMCIS)*, 2021.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms," arXiv preprint, arXiv:1707.06347, 2017.

[3] Jansen, N., Krämer, D., Barz, C., Niewiejska, J. and Spielmann, M., 2015, May. Middleware for coordinating a tactical router with SOA services. In 2015 International Conference on Military Communications and Information Systems (ICMCIS) (pp. 1-7). IEEE.

[4] Ciucu, F. and Schmitt, J., 2012, August. Perspectives on network calculus: no free lunch, but still good value. In Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication (pp. 311-322).

[5] Aschenbruck, N., Ernst, R., Gerhards-Padilla, E. and Schwamborn, M., 2010, March. Bonnmotion: a mobility scenario generation and analysis tool. In Proceedings of the 3rd international ICST conference on simulation tools and techniques (pp. 1-10).

[6] Shi, Y., Huang, Z., Wang, W., Zhong, H., Feng, S. and Sun, Y., 2020. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*.

[7] Sutton, R.S. and Barto, A.G., 2018. Reinforcement learning: An introduction. MIT press.

[8] Vlassis, N., 2007. A concise introduction to multiagent systems and distributed artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *1*(1), pp.1-71.

[9] Bernstein, D.S., Givan, R., Immerman, N. and Zilberstein, S., 2002. The complexity of decentralized control of Markov decision processes. Mathematics of operations research, 27(4), pp.819-840.

[10] Xu, K., Hu, W., Leskovec, J. and Jegelka, S., 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

[11] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C. and Sun, M., 2020. Graph neural networks: A review of methods and applications. AI Open, 1, pp.57-81.

[12] Wu, S., Sun, F., Zhang, W. and Cui, B., 2020. Graph neural networks in recommender systems: a survey. arXiv preprint arXiv:2011.02260.,

[13] Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A. and Kim, P.M., 2020. Fast and flexible protein design using deep graph neural networks. Cell Systems, 11(4), pp.402-411.

[14] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P. and Mordatch, I., 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*.

[15] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N. and Whiteson, S., 2018, April. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, No. 1).

[16] Sheng, J., Wang, X., Jin, B., Yan, J., Li, W., Chang, T.H., Wang, J. and Zha, H., 2020. Learning structured communication for multi-agent reinforcement learning. *arXiv preprint arXiv:2002.04235*.

[17] Jiang, J., Dun, C., Huang, T. and Lu, Z., 2018. Graph convolutional reinforcement learning. *arXiv preprint arXiv:1810.09202*.

[18] Liu, Y., Wang, W., Hu, Y., Hao, J., Chen, X. and Gao, Y., 2020, April. Multi-agent game abstraction via graph attention neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 34, No. 05, pp. 7211-7218).

[19] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W., 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

[20] Lopes, R.R.F., Balaraju, P.H., Rettore, P.H.L., and Sevenich, P., 2020. Queuing over Ever-changing Communication Scenarios in Tactical Networks. In *IEEE Transactions on Mobile Computing*, doi: 10.1109/TMC.2020.3005737.

[21] Suri, N., Marcus, K.M., van den Broek, C., Bastiaansen, H., Lubkowski, P. and Hauge, M., 2019, May. Extending the anglova scenario for urban operations. In *2019 International Conference on Military Communications and Information Systems (ICMCIS)* (pp. 1-7). IEEE.