*Proceedings of the 2021 Winter Simulation Conference*
*S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, eds.*

# GRAPH NEURAL NETWORK BASED BEHAVIOR PREDICTION TO SUPPORT MULTI-AGENT REINFORCEMENT LEARNING IN MILITARY TRAINING SIMULATIONS

Lixing Liu
Nikolos Gurney
Kyle McCullough
Volkan Ustun

Institute for Creative Technologies
University of Southern California
12015 E Waterfront Dr
Los Angeles, CA 90094, USA

## ABSTRACT

We introduce a computational behavioral model for non-player characters (NPCs) that endows them with the ability to adapt to their experiences — including interactions with human trainees. Most existing NPC behavioral models for military training simulations are either rule-based or reactive with minimal built-in intelligence. Such models are unable to adapt to the characters' experiences, be they with other NPCs, the environment, or human trainees. Multi-agent Reinforcement Learning (MARL) presents opportunities to train adaptive models for both friendly and opposing forces to improve the quality of NPCs. Still, military environments present significant challenges since they can be stochastic, partially observable, and non-stationary. We discuss our MARL framework to devise NPCs exhibiting dynamic, authentic behavior and introduce a novel Graph Neural Network based behavior prediction model to strengthen their cooperation. We demonstrate the efficacy of our behavior prediction model in a proof-of-concept multi-agent military scenario.

## 1 INTRODUCTION

Adaptive and human-like synthetic characters are essential features of new-generation simulation-based training environments. These characters fill roles traditionally performed by human participants, thus allowing for on-demand and location-agnostic training. Like their human counterparts, synthetic characters must be able to perform in a believable and authentic manner. If they cannot, the desired human-agent collaborations (or competitions) will not be realized. The process of generating such characters is complicated by the inherent complexity of the military training simulations that rely on stochastic and/or partially observable events, rich interactions amongst multiple players, and other non-linear features. Such environmental dynamics render rule-based control of synthetic characters virtually impossible. Building on Shiva, a multi-agent reinforcement architecture (Ustun et al. 2020), we have devised a framework to generate computational behavioral models which perform believable sequences of interactions with other agents.

Multi-agent Reinforcement Learning (MARL) and Graph Neural Network (GNN) modeling paradigms form our computational behavioral modeling framework's backbone. MARL models consider multi-agent systems characterized by a population of autonomous, interacting agents that share a common environment. These agents are generally equipped with sensors to gather information from and actuators that allow them to interact with the environment and other agents. It is common practice to embed state-of-the-art Artificial Neural Networks in MARL algorithms to allow them to interpret and make use of the information they encounter (Buşoniu et al. 2010). Shiva builds on the basic principles of MARL with the inclusion of an

978-1-6654-3311-2/21/$31.00 ©2021 IEEE

imitation learning module. Moreover, it is designed to support various Reinforcement Learning algorithms and interface with diverse simulation environments. In prior work, Shiva interfaced with a Unity-based simulation environment in which machine learning (ML) agents learned to defend in a proof-of-concept room clearing scenario (Ustun et al. 2020). Here, Shiva interfaces with a OpenAI Gym simulation environment that is a simplified abstraction of a Unity-based one. Pseudo-human trainee trajectories are simulated using MARL via Shiva in the simplified environment for this proof-of-concept work. These trajectories are later used to train NPCs to cooperate with their pseudo-human counterparts.

GNNs aggregate information from graph structures into simpler representations (Zhou et al. 2020). What differentiates GNNs from their predecessors, convolutional and recurrent neural networks, is the ability to operate on higher complexity data than what can be represented in regular Euclidean structures, e.g., a picture (2D) or text (1D). GNNs accomplish this by being order-invariant — they propagate on each node in the graph independently and ignore the input order — and by using the graph structure to guide propagation. Standard neural networks simply treat the structural information (edges) as node features. These innovations empower GNN models to "reason" about a graph, that is, draw general inferences, then use those inferences to make predictions and classifications. GNNs' convolutional and recurrent counterparts simply learn the distribution of data and then use what was learned to recreate synthetic versions of what they previously observed.

Military training simulations are challenging environments for MARL models because of the interaction dynamics. There are ways, however, to ease the learning difficulty. First, providing predictive information about the environment and other agents may help to accelerate and stabilize the learning process (Lee et al. 2020). For example, (Racanière et al. 2017) leveraged a recurrent neural network based imagination mechanism to assist Reinforcement Learning. Second, predictive models may help facilitate cooperation in multi-agent settings (Jiang et al. 2020). We propose using GNNs to learn and provide predictions about other agents' behaviors in military training simulations. These predictions support reinforcement learning, specifically by improving the agents' learned cooperation strategies.

We employ this framework to train and develop synthetic characters for roles in the Rapid Integration & Development Environment (RIDE), a Unity-based military training simulation environment (Hartholt et al. 2021). RIDE facilitates rapid development and prototyping of simulated environments in direct service of Army and other Department of Defense simulation communities. A core ability of RIDE is its drag-and-drop user interface that allows users to add elements from geo-specific terrain models, the Army validated Physical Knowledge Acquisitions Documents, non-player characters (NPC) and vehicles, and even AI behaviors without technical expertise. RIDE also supports Machine Learning experiments, allowing to directly train NPCs within the high-fidelity military training simulation environment. In this paper, to facilitate our training framework, we utilize a custom graph-based simulation environment that abstracts the full-scale RIDE simulation environment (Figure 1). This graph-based representation of the simulation environment is then leveraged in a later graph learning stage that exploits the reasoning ability of GNNs to predict agent behaviors. Our synthetic characters are, in a sense, pre-trained for RIDE using simplified graph representations and endowed with the ability to reason about the behaviors of other agents in the environment.

We have devised a proof-of-concept application and pipeline to demonstrate the efficacy of our framework. In this pipeline, we train a GNN model for behavior predictions utilizing observations comprised of environment states and the actions of an agent following a behavior policy independently trained through Reinforcement Learning. We then show that a successive agent introduced in the same environment learns faster and more effective cooperation strategies when provided with the output of the predictive GNN model.

## 2 BACKGROUND

Computer simulation plays a prominent role in modern military personnel training (Hill and Miller 2017). Driven by the simultaneous need for dynamic, human-like NPCs in simulations (our focus) and the

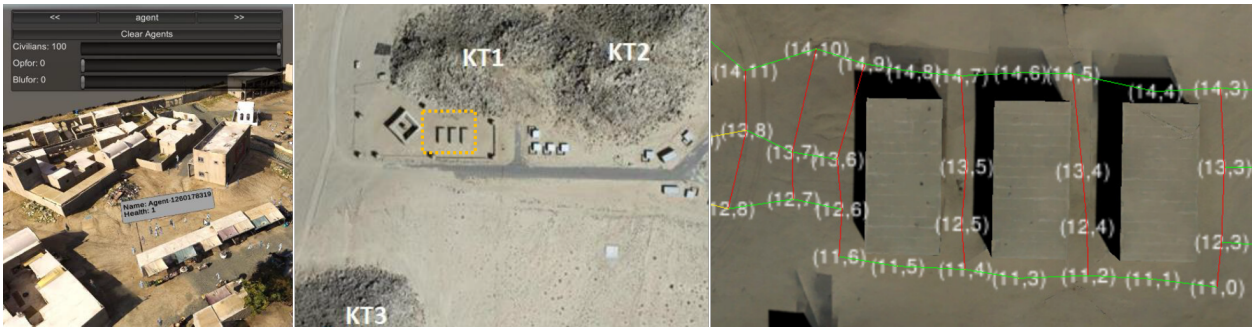


Figure 1: The left image shows a screenshot of a training instance in the RIDE platform. The image in the middle is a real terrain map. In the right image, discretized waypoints are marked on the zoomed-in view of the yellow box area in the map. These waypoints along with their connectivity to other waypoints generate the graph-based simulation environment scenario used in the experiments.

anticipated presence of synthetic teammates in future military operations, integrating intelligent synthetic agents is increasingly considered a necessary capability of the platforms used in military training (Bruzzone and Massei 2017). Prior research has demonstrated the ability to generate NPCs capable of human-like behaviors from a training framework similar to ours (Ustun et al. 2020). The framework employed in that research, however, lacked the training benefits of GNNs and utilized a complete RIDE instance. Despite the efficiency of RIDE, high fidelity simulations increase the turnaround times for the experiments, which is crucial, especially in the hyper-parameter tuning stage. Therefore, we leverage a graph-based simulation environment based on the RIDE model for our experiments.

To better understand the benefits of using a graph-based multi-agent scenario and GNN learning phase, it is helpful to review MARL and its role in generating NPCs. Recall that MARL algorithms are tools for modeling the sequential decision-making process of multiple agents in dynamic environments (Buşoniu et al. 2010). Moreover, the agents work to optimize their individual rewards by interacting with the environment and other agents. Legions of algorithms have been proposed to this end, with recent notable examples dominating games like Go (Silver et al. 2016; Silver et al. 2017), and advancing autonomous car technology (Shalev-Shwartz et al. 2016). In the case of NPCs, MARL seeks to find competent behavioral strategies for optimizing each character's reward based on their given value function — which can take multiple inputs, such as avoiding injury, finding food, and even socializing. MARL models cannot efficiently perform this optimization online during a training session since the data acquisition is a highly demanding process within the existing deep MARL frameworks. Instead, an effective offline model that is pre-trained in a simulated scenario is much preferable in practice. However, the NPCs may continue to update their behavior policies during training sessions to better adjust to human trainees' needs.

Shiva supports myriad MARL algorithms, such as Deep Q-Network (DQN) (Mnih et al. 2015), Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2016), Multi-agent DDPG (MADDPG) (Lowe et al. 2017), etc., along with the capability to add new algorithms to Shiva. MADDPG is one of the most commonly used algorithms in MARL experiments, and we also employ (MA)DDPG in the reinforcement learning stages of our experiments. MADDPG extends actor-critic policy gradient methods for multi-agent settings. It tries to address the challenges in MARL via a centralized training and decentralized execution approach; after the training is complete, only the local information is used during execution. In this approach, during training time, a central critic, which has access to observations of all the actors (agents) and their rewards, is set up for each agent. In contrast, the actors only have access to local information, depending on their central critic's feedback to improve their behavior policies. Centralized critics, as a result, can inform individual agent policies utilizing the information based on other agents' actions and received rewards, potentially yielding a learned consensus in cooperative tasks. However, computationally

intensive hyper-parameter tuning and long training times are needed to learn effective cooperation strategies in experiments run with the MADDPG algorithm.

As previously noted, Shiva is designed to interface with a diverse array of simulation environments. Previous experiments (Ustun et al. 2020) validated its ability to train agents in Unity, MuJoCo, NeuralMMO (Suarez et al. 2019), OpenAI Gym (Brockman et al. 2016), and the RoboCup Soccer 2D Simulation engines. Different use cases — think of military training versus autonomous driving — motivate the selection of different platforms. RIDE is a Unity-based simulation environment developed specifically to support Army training simulations as well as the broader Department of Defense community and an obvious environment for developing NPCs. Although RIDE is continually being developed to serve as a sandbox for prototyping simulated training environment components, deep MARL experiments are very computationally expensive, particularly at the hyper-parameter tuning stage. Training behavioral agents strictly in RIDE, therefore, is often computationally prohibitive. This bottleneck motivated developing a graph-based environment that implements the OpenAI Gym interface from which good candidate hyper-parameters or candidate behavior policies can be derived. These candidates can later be used to seed the main RIDE simulation or further train in the graph-based environment.

We use a Graph Attention Network (GAT) architecture (Veličković et al. 2018) in the graph learning phase of training. Recall that graphs are data structures in which nodes (vertices) are connected via edges. Nodes represent sub-units of data in the larger structure. Edges indicate a relation between two nodes in the structure and can be directed or not. Graph learning aims to extract information from the structure that is useful in making predictions about future states of the graph or its component parts. GATs use a spatial-based approach in which graph convolutions aggregate information from neighbors. This means that rather than considering the graph structure as a whole, data is gathered from small, overlapping neighborhoods of the graph for feature extraction. The GAT architecture is differentiated from similar networks by the attention mechanism. This mechanism allows the model to learn which features of a node are important to its neighbor. Other architectures assume that contributions to a central node from its neighbors are identical (Hamilton et al. 2017) or that they are predetermined (Kipf and Welling 2017). However, a learnable attention mechanism performed better in several problems where the GAT architecture out-performed other architectures (Veličković et al. 2018).

Learning on the graph representation afforded by the OpenAI gym scenario is possible in a variety of ways, from simple to complex. Like many computational tasks, each modeling paradigm presents unique opportunities and trade-offs. Simple models are easy to deploy (i.e., computationally efficient and relatively straightforward to program) and can facilitate learning from limited data. This all, of course, comes at the expense of learning important behavioral nuances. More complex models, on the other hand, can potentially capture more nuance but require richer data and more resources to run. The GAT architecture, for example, can be enriched with edge features. Adding edge types extends the model that GATs are based on to allow graphs to have multiple, independent relationships between nodes (Schlichtkrull et al. 2018). Allowing for different node and edge types yields a heterogeneous graph (Wang et al. 2019). In this type of architecture, the different node types have unique feature sets but can still aggregate information about neighbors of a different type. Edge features can also be added to the network resulting in a rich network capable of capturing subtle nuances about a graph structure. Since our motivation is demonstrating the viability of this training framework and keeping the computational costs low, we decide to use the basic GAT.

## 3 APPROACH

We apply our framework in a proof-of-concept experiment on training Non-Player Characters for roles in simulations with human trainees. A military skirmish scenario serves as the vehicle to demonstrate the viability and effectiveness of our NPC training framework. Our ultimate goal is to endow the NPCs with the ability to perform human-like strategic behaviors in skirmishes with human trainees. This work, in

which computational agents learn, compete against other agents, and collaborate with agents that replicates human behavior, is a first step towards that goal.

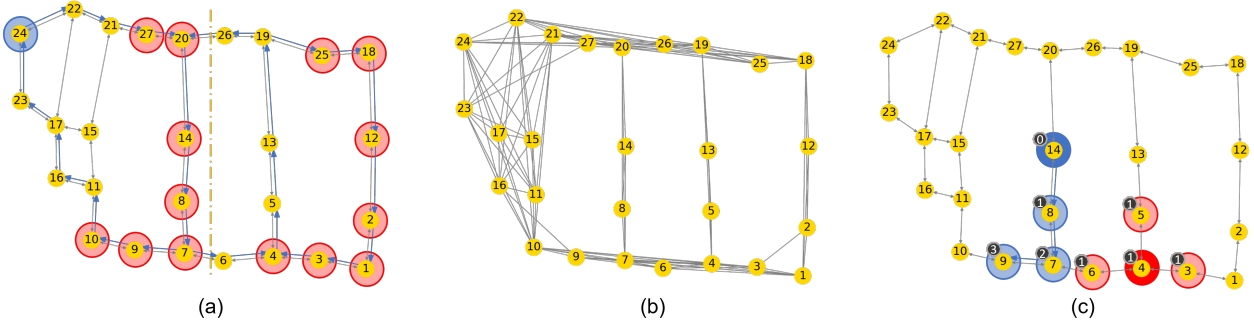


Figure 2: The graph representation of the environment. (a) demonstrates the graph connectivity. The highlighted red nodes depict the potential spawn positions for Red agents. The blue arrows construct the patrol route of the Blue agent. (b) is the visibility graph used for field of view calculations. (c) is an example of the node feature initialization in graph learning phase. The colored nodes represent the agents' current and future positions. The corner mark denotes how many steps ahead would the agent appear on that node.

### 3.1 Graph-based Simulation Environment

A discretized abstraction of the continuous terrain map from the RIDE platform helps model the relations between the agents and the surrounding area. This discretized map representation is easily translated into a graph representation in which each discrete location is a node connected to its cardinal neighbors via edges. The nodes capture proximity information about the geometric coordinates while the edges model an agent's movement in a single simulation step. More importantly, this allows for predetermined, static graph connectivity during training. This feature significantly reduces both the requisite computational resources and training data.

For reinforcement learning, we developed a graph-based custom OpenAI Gym multi-agent scenario along with its interfacing wrapper to the Shiva architecture to replace the Unity based RIDE simulation environment. This alone resulted in experiments running seven times faster, on average, all else being equal (i.e. hyper-parameter configurations). The NetworkX library (Hagberg, Swart, and Schult 2008) was utilized to model the sparse graph representation of the map (Figure 2a). The learned policies can then be transferred back to RIDE for evaluations and initialization of new training instances. Furthermore, this graph representation for the environment can be leveraged directly (Figure 2c) in instantiating GATs to learn behavior predictions.

### 3.2 Proof-of-Concept Scenario

The building skirmish simulation scenario includes two teams of agents. Red agents are the NPCs of interest, i.e. those being trained via our framework. The Blue agents follow scripted patrol routes informed by actual human behavior in real simulations. Both Blue and Red agents attempt to terminate their opponent in the skirmish. Red agents are awarded points each time they successfully "shoot" a blue agent and lose points for being shot. Points are also awarded at the end of the skirmish based on if a Red agent successfully terminated its opponent and how much health it has remaining.

The 2D graph representation we utilized in this paper captured a local view containing three adjacent building blocks in an approximately 30 meter by 50 meter area, extracted from a 3D geo-specific terrain map instance. We marked 27 way-points based on the agent's step-wised movement limitations in Unity

(Figure 1). In the graph-based simulations, those 27 nodes along with edges generated by the constrained movements were utilized in all training scenarios (Figure 2a). Just like actual human participants, the agents cannot see through the buildings (Figure 2b) and must learn to hide and ambush.

### 3.3 Learning Pipeline

The learning pipeline of our NPC training framework consists of three phases: two Reinforcement Learning (RL) phases and a Graph Learning (GL) phase. The skirmish in the first RL phase pits a single heuristic Blue agent against a single Red — the agents play the roles of human participants. Recall that the Red agents learn via the (MA)DDPG algorithm, an actor-critic method that optimizes a policy (actor) based on a value function (critic). Once learning stabilizes (i.e. performance plateaus) in the initial RL phase, the policy is frozen and a batch of trajectories, which stand in for human participant data at the moment, is processed that can later be used in the GL phase of the training. Each unique set of hyper parameters results in a unique desired policy. Every policy generates inputs for a separate graph model and the hidden states of the output layer of each graph model associated with a policy are inserted into the observations as additional teammate action indicators for a second Red agent in a new RL phase.

The Red agent we introduce in the second phase of RL, which plays the role of an NPC that cooperates with the pseudo-human agent, has an enriched observation tensor that includes the intermediate outputs from the GL phase. The new Red agent, like its teammate, is attempting to learn a behavioral policy. This RL phase also introduces another element to the reward function: if the two Red agents stand on the same node, a point is forfeited. Unlike its teammate, this agent is able to draw on prior knowledge via the GL to predict its teammate's behavior. This means that the new Red agent is not only trying to learn a policy to neutralize the Blue agent, but also coordinate with its teammate. Our hypothesis is that the ability to predict the teammate's behavior will reduce the time required to train the new Red agent and the variance in the total reward obtained across batch learning episodes.

## 4 EXPERIMENT

As stated in Section 3.3, our experiment starts with a RL phase, followed by GL phase to learn from the data generated in the first RL phase, and finally another RL phase that exploits the GL to improve performance. The experiment setup details of these two learning components are further discussed in this section.

### 4.1 Reinforcement Learning

Each agent was initialized with its own actor-critic networks and a hyper-parameter set for reinforcement learning. The potential spawning positions for Red agents were fourteen nodes on the left side and right side of the graph as shown in (Figure 2a). Agents begin with full health points in each new iteration and could lose health points each time their opponent successfully shot them. The end game condition was defined as a team losing all health points or if the simulation reached the maximum step in an episode. To shoot an opponent, an agent had to have them in sight and within combat range. The field of view for all agents was limited to 120 degrees, and the combat range was set to 25 meters.

To simplify the agents' interactions and better control the rewards, all agents were granted 100 health points so that they would never die or freeze. All agents continued engaging with each other until running 40 steps for each episode. A shooting action at a target who was in view and within range was guaranteed to hit and take precisely one health point down. We defined a Red win as the case when at least one Red agent survived, which means the agent lost no more than five health points, and the Blue agent's final health points dropped below a given threshold. The losing threshold was set to 5 for a single Red agent scenarios and 10 for two Red agents simulations. A perfect run happened when all Red agents remained with full health and the Blue agent's health dipped below the threshold.

- **Action space:** The action space consisted of moving and turning for each simulation step. All agents first selected a movement from the valid actions in the set $\{NOOP, S(outh), N(orth), W(est), E(ast)\}$. An agent could always choose to stay; however, movement in the cardinal directions was limited by the edges connecting the current node to other nodes. If the present node did not have a neighbor to the North, this option was not available. After the movement, the agent can choose to look in front of the current moving direction or turn 90 degrees to the left or right. Consequently, agents were restricted to a (5, 3) two-branched discrete action space grounded by a valid action mask.
- **Observation space:** The observations for agents were the concatenation of position embedding vectors and engaging indicators. Position embedding vectors were node coordinates embedded by 6-bit binary tokens. The binary engaging indicator matrices were determined by the field of view checks and combat range checks for each pair of the opposing agents in every simulation step. In single Red agent simulations, the observation space for the Red agent had 20 elements. In contrast, in two Red agents baseline cases, the 26-dimension observation tensors were padded with a 6-bit teammate position embedding. In the complete experiment, the observation had been further extended by teammate behavior indicators, yielding a total size of 31 elements.
- **Reward design:** During step $i$, the Red agents got reward $R^i_{r->b} = +3$ for shooting Blue agents and lost $R^i_{b->r} = -4$ for being shot. In the teammate collaboration cases, a negative step reward $R^i_{r->r} = -1$ was applied if a Red agent stood on the same node with its teammates. After a simulation was done, if all Blue agents had been terminated successfully and the Red agent had ever shot, a positive episode reward $R^{ep}_{rHP} = [0, 15, 20, 25, 50, 100]$ $for\ damage\ taken \in [0, \infty)$ would be added to this agent's final reward based on its remaining health points. Note that Red agents were incentivized to be cautious, as the cost of being shot was higher than the reward for successfully shooting the Blue agent. The desired policies were selected based on this additional constraint when both actor and critic losses were stable.

### 4.2 Graph Learning

Our GAT architecture consists of two stacked attention layers with the same connectivity as the graph representation of the environment in RL. These layers first apply a linear transformation to each node's input features to create higher-level features, i.e., embeddings. Next, attention scores are computed for neighbors. This computation is accomplished by concatenating neighboring nodes' respective embeddings, taking the dot product of this concatenation and a learnable weight vector, applying an activation function, and finally applying self-attention on the nodes. Each node "attends" to its neighbors through this mechanism, and the model learns the importance of node $j$'s features to node $i$. Deep Graph Library (DGL) (Wang et al. 2019) provided the GAT modules.

The task is to predict the first Red agent's next position on the graph via node classification. The network was trained using the batch of skirmishes collected after the RL was frozen. For each step in every batch skirmish, node feature vectors are generated for all nodes in the graph. Each vector had ten features. The first was a binary flag, $Pos^{t+1}_{red}$, turned on if a Red agent could appear on this node in the next step. The following 9 flags were three tuples, $(Pos^{t+k}_{blue},\ Dmg^{t+k}_{b->r},\ Dmg^{t+k}_{r->b})$, where $k \in [1, 3]$. Each slot in the tuples represented a successive look forward (Figure 2c). $Pos^{t+k}_{blue}$ contains the next three positions of the Blue agent, meaning if the Blue agent appears on the given node at that time step, then the associated binary flag is turned on. $Dmg^{t+k}_{b->r}$ encodes whether Blue can inflict damage on Red from the given node in each of the following time steps while $Dmg^{t+k}_{r->b}$ captures the same for Red.

After the first round of the single Red agent RL training, all possible trajectories were collected by executing its deterministic policy for 40 simulation steps in evaluation mode from all seven initial positions. The 280 training samples were fed into the graph neural networks for a supervised node classification task. Finally, the second round of RL training introduced another Red agent creating a Red team that battles a single Blue agent. In the two Red agents RL phase, agent $A_{Red_0}$ reused the stored policy from the initial RL

training. In contrast, the new agent $A_{Red_1}$ started a novel RL training period with the extended observations, including the teammate action indicators provided by the trained GAT at run-time.

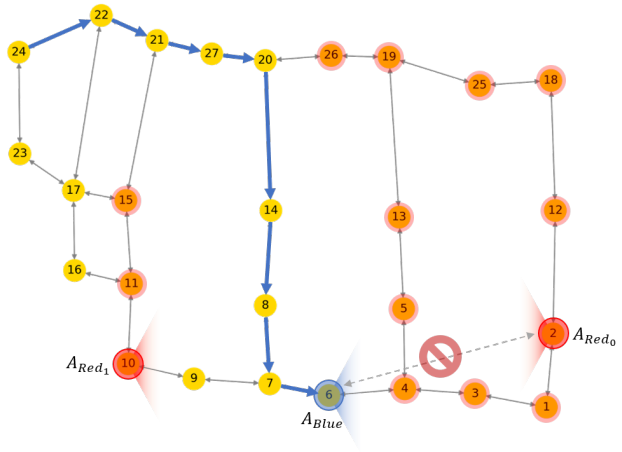


Figure 3: An instance of a perfect run condition that both Red agents shot at least five times and had full health points after 40 simulation steps. The first agent $A_{Red_0}$ visited 11 nodes across the right side of the graph while the second agent $A_{Red_1}$ moved back and forth within a region of 3 nodes on the left side. The figure showed the environment states in step 8 when $A_{Red_1}$ made a shot. There was no interaction between $A_{Red_0}$ and the Blue agent in this step, since they were out of sight of each other.

### 4.3 Results

In the single Red against single Blue MADDPG experiments, both agents started with five health points. Note that it was technically possible for the Blue agent to win as were ties; however, neither of these event types were used in RL policy derivation. The single Red agent only spawned on the right side of the graph with seven possible initial positions. The scripted Blue agent followed a 34-step patrol route circling the buildings (Figure 2a). Multiple MADDPG training runs were conducted using MSELoss loss function and Adam optimizer. The following learning parameters of the MADDPG agent were fine-tuned: actor and critic learning rates, range of noises, exploration episode, network size, and MADDPG specific configurations. Many good policies resulted in the Red agent finding a strategic waypoint and barely moving from that spot. Because the first phase of RL is designed to supplant dynamic human behavior, we did not include these policies in the GL phase's training set. Instead, we gathered high-reward learned policies that were characterized by more movement. We found that high initial noises with exploration episodes greater than $5,000$ would help to learn frequent moving policies. It also requires that the action's noises gradually reduce to near zero during $30,000$ to $60,000$ episodes after the exploration period. The perfect win shown in (Figure 3) was achieved when actor and critic learning rates were $0.000003$ and $0.000009$, respectively. The best set of actor-critic networks and learning rates, and exploration period were reused in the final phase of RL training.

The learned policies and their stored trajectories were parsed to initialize the node features in the graph learning phase. Five-fold cross-validations were conducted for the supervised graph node classification tasks. For the demonstrated policy, the average accuracy of the cross-validation was $89.3\%$, which indicated that most action prediction providing the next RL phase would be the actual moving action generated by the saved deterministic policy. In this case, we assigned the value of an action that had a probability greater than $0.4$ as *True* to generate multi-hot five-dimensional tensors for the next training phase.

In the two Red agents versus one Blue agent MADDPG training scenario, which was the final phase of the preliminary experiments in this paper, the Blue agent's starting health points were doubled. In this

case, the winning condition for team Red became shooting the Blue agent more than ten times in total. We compared the complete experiment, which included teammate action data from the graph model, to two baseline conditions: 1) $A_{Red_1}$ only had a local view of itself similar to the single-agent simulation scenario and 2) $A_{Red_1}$ had teammate location inserted to its observation. We executed the first baseline experiments to fine-tune a part of hyperparameters, including the range, starting, and ending values of noises to find perfect wins of team Red. Note that until this point, the only changeable hyperparameters were the MADDPG agent-specific variables. The noises, exploration periods, actor-critic networks, and learning rates were all fixed for the second baseline and the final experiments to make it fair for evaluating the training speed.

The second Red agent could also select to spawn on either the left or right side of the map with seven possible initial nodes for each case. We considered these two conditions as different simulation tasks and reported them separately in (Table 1). Two agents starting on the same side of the map turned out to be a more challenging learning case due to the negative $R_{r->r}$ step reward. With this in mind, we decided to set the threshold of the averaged total rewards to 20 for agents starting on different sides of the map and 15 for starting on the same side. We stored trajectories for every 100 episodes in the former and every 500 episodes in the latter case. We measured the earliest simulation episode when the top-performing agent's average reward reached a certain threshold to compare the learning speed. The results showed that the additional teammate observation tensor led to agents who learn to coordinate faster than non-augmented baselines.

Table 1: Experiment results of the two Red agents versus a single Blue agent simulations. A policy reaching to a reward threshold with fewer simulation episodes is learning faster.

| The first episode $Reward_{Red_1}$ passing a given threshold | Started on different sides | | | | Started on the same (right) side | | | |
|---|---|---|---|---|---|---|---|---|
| | $\geq 5$ | $\geq 10$ | $\geq 15$ | $\geq 20$ | $\geq 0$ | $\geq 5$ | $\geq 10$ | $\geq 15$ |
| Baseline[20]: local obs only | 71.7k | 78.6k | 85.3k | 103.7k | 96k | 170.5k | 262.5k | – |
| Baseline[26]: $+A_{Red_0}$ position | 59.1k | 64.2k | **66.0k** | 85.1k | **83.5k** | 98k | 142k | 186.5k |
| Complete[31]: $+A_{Red_0}$ actions | **57.0k** | **63.1k** | 66.5k | **82.5k** | 86.5k | **92k** | **116.5k** | **138k** |

## 5 CONCLUSIONS

Our experiment focused on developing believable behaviors for non-player characters in skirmish simulations. The motivation for such characters is clear: NPCs are critical to running a simulation that has face validity to the players, but filling a simulated training environment with human actors is not always feasible. Having the capability to generate such NPCs helps to 1) create more realistic and challenging training experiences 2) reduces the cost and time to develop them and 3) makes these simulations less dependent on human participants' availability.

Our experiment relied on a computationally generated version of human agents or synthetic characters—the Blue and the first Red agent—to simplify the proof-of-concept. Even though we do not directly address human-synthetic composite teams in this work, their viability has already been demonstrated in a flight simulation training environment (Myers et al. 2019). Synthetic teammates offer training diversity, meaning players can "get to know" various behavioral profiles in a non-combat setting. Moreover, synthetic teammates could potentially be a part of military operations in the foreseeable future (USACAC. 2020). It is crucial that the research and development communities fully understand the dynamics of composite teams before fully deploying synthetic teammates in live missions. Our framework offers rapid prototyping of synthetic teammate behavioral profiles, an indispensable tool for understanding how synthetic teammate characteristics impact composite team performance. Still, learning robust and stable computational behavior policies that can be deployed in training environments is a big challenge. The proposed framework is just a step towards addressing one of the aspects of that challenge.

### 5.1 Future Research

We strive to improve the capabilities of NPCs in military training simulations. Generating NPCs that build better behavior representations of their teammates and opponents is possible by refining our framework and pipeline. To this end, we plan to:

- **Reward design:** In our reinforcement learning experiments, high episode reward plus low step reward made the losses hard to converge. Sharp pulses on the loss replay were observed frequently once a good episode reward was obtained while the average reward over time was stabilized at a relatively low value. Although reducing the episode reward could ease the convergence issues, low episode rewards encouraged the Red agent to learn aggressive behaviors rather than maintain good health, which defied our intention. Our immediate next step following the preliminary experiments would be exploring novel reward design strategies to balance reward signals and loss values.
- **Scripted behaviors:** We utilized a relatively simple behavior script to control the Blue agent. We plan to gradually increase the complexity of the Blue agent's behavior script to further test the robustness of our pipeline.
- **Introducing human data:** We plan to collect and utilize human data in our experiments after the restrictions due to the current pandemic are relaxed. Replacing the RL policy generated trajectories with more realistic ones would allow us to better evaluate the predictions learned by GNNs in the graph learning phase. Furthermore, such predictions will provide the opportunity to observe how the second red agent adjusts to more realistic human behavior.
- **Structured learning:** In the current stage of the framework development, the transition from graph learning to reinforcement learning was a multi-hop binary tensor, which was trivial. Our ultimate goal is to introduce the graph-structured information gathered from the graph model's intermediate layers to the actor-critic training scenarios to better capture the relations between the agents and the local geometric layouts. We plan to adjust agent action selections by leveraging structured data provided by graph models in the next stage.
- **Joint learning:** The graph model in this pipeline was trained offline, and there was only a single agent actively learning in the reinforcement learning experiments. When there are multiple agents learning concurrently, a more sophisticated framework would train the reinforcement and graph components, leveraging behavior prediction to assist learning in RL. Implementing a more expressive network architecture may improve learning with minimal computational cost or facilitate learning a richer repertoire of behaviors — both of which would be valuable in training a synthetic character to be a teammate.

### 5.2 Summary

The two main research questions we explored in this paper are: (1) If we have human behavior data, can a GNN like model learn to predict the behavior?; and (2) if we have such a GNN based behavior predictor, can we use it to help with learning better cooperation policies in multi-agent reinforcement learning? We did not have human data to explore the first question. Instead, like a pseudo-human, we generated trajectories via a behavior policy learned through reinforcement learning to use as training data. Utilizing GNNs as behavior predictors yielded promising results. In pursuit of the second research question, we implemented a MARL-GL architecture that utilizes the outputs of the GNN based behavior prediction model. Including these outputs enabled new agents to better coordinate with existing agents within a reinforcement learning experiment. We have leveraged a simple skirmish scenario in the RIDE simulation environment to facilitate our experiments. Our experiments show that this framework could learn faster-converging behavior policies with better cooperation characteristics for our proof-of-concept simple skirmish scenario. However, more experiments warranted to further investigate our approach.

## ACKNOWLEDGMENTS

Part of the effort depicted is sponsored by the U.S. Army Research Laboratory (ARL) under contract number W911NF-14-D-0005, and that the content of the information does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

## REFERENCES

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. 2016. "Openai gym". In *arXiv preprint arXiv:1606.01540*. June 5th.

Bruzzone, A. G., and M. Massei. 2017. "Simulation-based military training". In *Guide to Simulation-Based Disciplines*, edited by S. Mittal, U. Durak, and T. Ören, 315–361. Cham: Springer International Publishing.

Buşoniu, L., R. Babuška, and B. De Schutter. 2010. "Multi-agent reinforcement learning: An overview". In *Innovations in multi-agent systems and applications-1*, edited by D. Srinivasan and L. C. Jain, 183–221. Berlin, Heidelberg: Springer.

Hagberg, A. A., P. J. Swart, and D. A. Schult. 2008. "Exploring network structure, dynamics, and function using NetworkX". In *Proceedings of the 7th Python in science conferences (SciPy)*. May 13th-17th, Pasadena, California, 11–15.

Hamilton, W. L., R. Ying, and J. Leskovec. 2017. "Inductive representation learning on large graphs". In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. December 4th-9th, Long Beach, California, 1025–1035.

Hartholt, A., K. McCullough, E. Fast, A. Reilly, A. Leeds, S. Mozgai, V. Ustun, and A. S. Gordon. 2021. "Introducing RIDE: Lowering the Barrier of Entry to Simulation and Training through the Rapid Integration & Development Environment". In *Proceedings of the 2021 Virtual Simulation Innovation Workshop*. February 8th-12th.

Hill, R. R., and J. O. Miller. 2017. "A history of United States military simulation". In *Proceedings of the 2017 Winter Simulation Conference*, edited by V. W. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. H. Page, 346–364. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Jiang, J., C. Dun, T. Huang, and Z. Lu. 2020. "Graph Convolutional Reinforcement Learning". In *Proceedings of the International Conference on Learning Representations*. April 26th - May 1st.

Kipf, T. N., and M. Welling. 2017. "Semi-supervised classification with graph convolutional networks". In *Proceedings of the 5th International Conference on Learning Representations*. April 24th-26th, Toulon, France.

Lee, K.-H., I. Fischer, A. Liu, Y. Guo, H. Lee, J. Canny, and S. Guadarrama. 2020. "Predictive information accelerates learning in RL". In *Proceedings of the 34th International Conference on Neural Information Processing Systems*. December 7nd-12th, 11890–11901.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2016. "Continuous control with deep reinforcement learning". In *Proceedings of the 4th International Conference on Learning Representations*. May 2nd-4th, San Juan, Puerto Rico.

Lowe, R., Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. 2017. "Multi-agent actor-critic for mixed cooperative-competitive environments". In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. December 4th-9th, Long Beach, California, 6382–6393.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al. 2015. "Human-level control through deep reinforcement learning". *Nature* 518(7540):529–533.

Myers, C., J. Ball, N. Cooke, M. Freiman, M. Caisse, S. Rodgers, M. Demir, and N. McNeese. 2019. "Autonomous Intelligent Agents for Team Training". *IEEE Intelligent Systems* 34(2):3–14.

Racanière, S., T. Weber, D. P. Reichert, L. Buesing, A. Guez, D. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li et al. 2017. "Imagination-augmented agents for deep reinforcement learning". In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. December 4th-9th, Long Beach, California, 5694–5705.

Schlichtkrull, M., T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. 2018. "Modeling relational data with graph convolutional networks". In *Proceedings of the 15th European Semantic Web Conference*. June 3rd-7th, Grete, Greece, 593–607.

Shalev-Shwartz, S., S. Shammah, and A. Shashua. 2016. "Safe, multi-agent, reinforcement learning for autonomous driving". In *Proceedings of the NIPS Workshop Learning, Inference and Control of Multi-Agent Systems*. December 9th, Barcelona, Spain.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al. 2016. "Mastering the game of Go with deep neural networks and tree search". *Nature* 529(7587):484–489.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al. 2017. "Mastering the game of go without human knowledge". *Nature* 550(7676):354–359.

Suarez, J., Y. Du, P. Isola, and I. Mordatch. 2019. "Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents". In *arXiv preprint arXiv:1903.00784*. March 2nd.

USACAC. 2020. *STE Industry Day Statement of Need*. US Army Combined Arms Center. https://usacac.army.mil/sites/default/files/documents/cact/STE_Industry_Day_Statement_of_Need.pdf.

Ustun, V., R. Kumar, A. Reilly, S. Sajjadi, and A. Miller. 2020. "Adaptive Synthetic Characters for Military Training". In *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*. November 30th - December 4th, Orlando, Florida.

Veličković, P., G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. 2018. "Graph Attention Networks". In *Proceedings of the Sixth International Conference on Learning Representations*. April 30th - May 3rd, Vancouver, Canada.

Wang, M., D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang. 2019. "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks". In *arXiv preprint arXiv:1909.01315*. September 13th.

Wang, X., H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu. 2019. "Heterogeneous Graph Attention Network". In *Proceedings of the World Wide Web Conference 2019*. May 13th-17th, San Francisco, California, 2022–2032.

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

## AUTHOR BIOGRAPHIES

**LIXING LIU** is a Ph.D. student in the Department of Computer Science at USC ICT. He is a member of the Cognitive Architecture Group. His research interests include graph representation learning, multi-modality human behavior analysis and computational cognitive models. His email address is lxliu@ict.usc.edu.

**NIKOLOS GURNEY** is a postdoctoral research associate at USC ICT. He is a computational social scientist who studies human behavior and decision making, particularly in interactions with intelligent machines. He has a Ph.D. in Behavioral Decision Research. His email address is gurney@ict.usc.edu.

**KYLE MCCULLOUGH** is the Director of Modeling & Simulation at USC-ICT. His research involves geospatial initiatives in support of the Army's One World Terrain project, as well as advanced prototype systems development. His work includes utilizing AI and 3D visualization to increase fidelity and realism in large-scale dynamic simulation environments, and automating typically human-in-the-loop processes for Geo-specific 3D terrain data generation. Kyle received awards from I/ITSEC and the Raindance festival, winning "Best Interactive Narrative VR Experience" in 2018. He has a B.F.A. from New York University. His email address is McCullough@ict.usc.edu.

**VOLKAN USTUN** is the Associate Director of the Cognitive Architecture Group at USC ICT. His general research interests are multi-agent systems, computational cognitive models, cognitive architectures, and simulation. He has a Ph.D. degree in Industrial and Systems Engineering and held postdoc positions at Georgia Tech and Rice University before joining ICT. His email address is ustun@ict.usc.edu.