Multi-Agent Mission Planning with Reinforcement Learning

Sean Soleyman, Deepak Khosla
HRL Laboratories, LLC
ssoleyman@hrl.com, dkhosla@hrl.com

Abstract

State of the art mission planning software packages such as AFSIM use traditional AI approaches including allocation algorithms and scripted state machines to control the simulated behavior of military aircraft, ships, and ground units. We have developed a novel AI system that uses reinforcement learning to produce more effective high-level strategies for military engagements. However, instead of learning a policy from scratch with initially random behavior, it also leverages existing traditional AI approaches for automation of simple low-level behaviors, to simplify the cooperative multi-agent aspect of the problem, and to bootstrap learning with available prior knowledge to achieve order of magnitude faster training.

Introduction

Simulation software for military applications has revolutionized battle management and analytics, and also provides a gateway for integrating recent developments in machine learning with real-world applications. AFSIM (Advanced Framework for Simulation, Integration, and Modeling) allows military analysts to build a detailed model of a mission scenario that includes aircraft, ships, ground units, weapons, sensors, and communication systems (Clive et al. 2015). However, no mission simulation would be complete without models for how the platforms behave – both at a strategic and tactical level. Therefore, users of this software are not only required to model physical systems and their capabilities, but must also serve as AI designers.

The end objective of our work is development of a more generalizable form of artificial intelligence to address multi-domain military scenarios, with initial focus on battle management and air-to-air engagements. Our goal is to produce a decision-making engine that provides enhanced automation of tactical and strategic decision-making.

The current rule-based approach for specifying platform behaviors in AFSIM is based on video game style AI. Each unit is given a processor that executes tasks such as following a pre-set route, firing a weapon at the appropriate time, or pursuing a particular opponent. However, this approach has several detrimental properties. The development of scripted polices is time consuming, and must be performed by analysts with an aptitude for computer programming as well as an understanding of military strategy and tactics. In addition, scripted policies are fragile. Minor changes to the scenario (such as those that would be explored when analyzing possible contingencies) can often cause the scripted platform behavior to become nonsensical, necessitating the expenditure of even more scenario development resources. Most importantly, there is always the possibility that a human analyst could fail to consider an unexpected strategy employed by a particularly clever adversary.

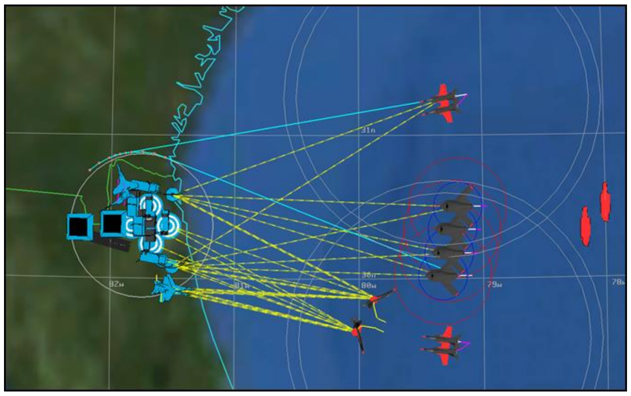


Figure 1 - Example of a complex AFSIM scenario involving air, sea, and ground units. Analysts must model all of these platforms and specify their behaviors with rule-based systems.

Model-free reinforcement learning algorithms provide an alternative solution that eliminates the need for scripting. Instead of specifying behaviors for each platform, the analyst needs only to design an agent-environment interface with a well-defined observation space, action space, and reward function. A reinforcement learning agent takes care of the rest by starting out with completely random behavior and improving by trial and error (Lapan 2018).

First, we will describe our initial effort to apply this naïve baseline approach in a simplified AFSIM-like 2D multi-agent simulated environment (MA2D) that we developed in-house. This simulator is easier to experiment with because it is written entirely in Python. Then, we will provide experimental evidence that reinforcement learning can be much more effective when combined with more traditional non-learning based AI techniques that constitute the current state of the art in practical applications, and will finally demonstrate that this hybrid approach can produce robust results in an actual AFSIM-based scenario that models aircraft and missile dynamics.

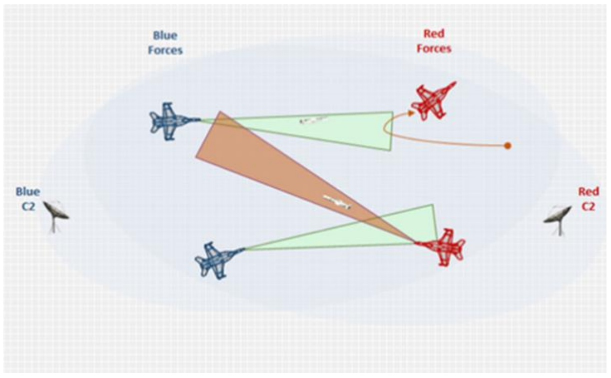


Figure 2 - Conceptual illustration of the AFSIM scenario that we are exploring initially. In each episode, a number of red and blue fighters are placed at random locations on a map. A baseline scripted AI is used to control the red team, and our new hybrid RL agent learns a policy for defeating the red team.

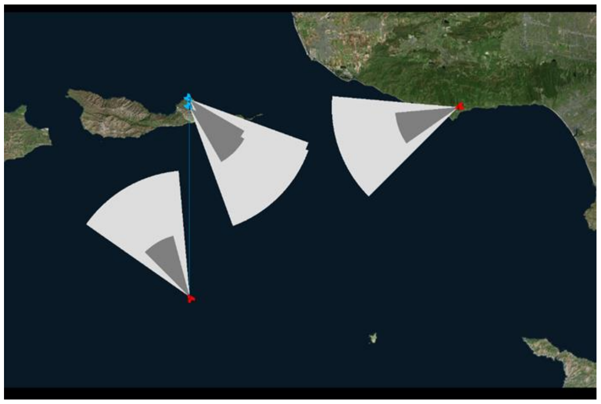


Figure 3 - Simplified MA2D environment, written entirely in Python. This example contains two blue fighters and two red fighters. Dark gray areas represent each unit's weapon zone. The objective is to destroy all opponents by getting each within this zone, while avoiding similar destruction of friendly aircraft. This simplification eliminates the need for modeling missile flight.

Related Work

In recent years, deep reinforcement learning agents have achieved super-human performance in complex multi-player games such as StarCraft II (DeepMind 2019), Defense of the Ancients (DOTA) (OpenAI 2018), and Quake / Capture the Flag (Jaderberg et al. 2019). Although these computer games are not intended to simulate real-world military engagements, they do possess several key similarities that demonstrate the applicability of deep reinforcement learning technology to military decision making.

First, all of these games consist of two adversarial teams, each composed of a number of cooperative platforms. In Starcraft II, each team may contain over 100 individual units with capabilities loosely resembling those of military ground units and aircraft. DeepMind's approach is to use a single centralized reinforcement learning agent to control each team by selecting a set of platforms and issuing a command to the entire set (Vinyals et al. 2017). OpenAI Five's DOTA solution uses a different type of multi-agent environment interface, where each agent receives a separate command at each time-step (Matiisen 2018). DeepMind's Capture the Flag AI uses a distributed approach, where a separate agent controls each unit (Jaderberg et al. 2019). The multi-agent solution we will describe in this paper relates most closely to the last of these three, but also includes a novel hybridization of RL with the non-learning Kuhn-Munkres Hungarian algorithm (Kuhn 1955).

Another major similarity between these computer games and real-world military simulations is that both are designed to model continuous time with short discrete time-steps. As a consequence, each episode may consist of thousands of discrete time-steps and each agent may therefore need to select thousands of actions before it receives a final win/loss reward. This creates a challenging temporal exploration problem that is a key focus of existing work in hierarchical reinforcement learning (Sutton, Precup, and Singh 1999) (Frans et al. 2018). Our hybrid hierarchical approach is more closely related to dynamic scripting, which has been applied to computer games (Spronck et al. 2006) as well as simple air engagements (Toubman et al. 2014).

Finally, success of model-free deep RL in computer game environments demonstrates that this approach will extend naturally to partially-observable environments. In StarCraft II and DOTA, each team can only perceive enemy units that are within visual range of one of their own units. In Capture the Flag, the agent actually perceives visual images of the 3D simulated environment, and it is possible for enemies to hide behind walls. In real-world air engagements, pilots identify enemy units using sensing modalities such as radar, vision, and IR. Implementation of realistic partially-observable air engagement scenarios is the subject of future planned work, and successes in computer game environments demonstrate the capability of deep reinforcement learning agents with LSTM units (Hochreiter and Schmidhuber 1997) to achieve good results even when confronted with imperfect information.

Reinforcement Learning Baseline Method

Our initial experiments were performed using a simple MA2D environment similar to the one illustrated in Figure 3. A reinforcement learning agent was given control over a single blue fighter, and traditional scripted behavior was used to control the red fighter. In some experiments, the red fighter was set to use a pure pursuit strategy against the blue fighter. In others, it simply traveled straight, providing a moving target for the blue agent to intercept. We introduced variation to the problem by having each of the fighters start each episode in a random location on the map, with random heading. This ensures that the agent learns a generalizable policy, not just a point solution to a single scenario. Each fighter's turn rate is limited to 2.5 degrees per time-step, and each fighter's acceleration is limited to 5 m/s/time-step. An opponent is instantly defeated if it comes within the circle sector shown in dark gray with radius 2km and angle 30 degrees.

Each episode lasts for a maximum of 1000 time-steps. The reward function consists of sparse and dense components. At the end of each episode, the agent receives a large positive reward if it has destroyed its opponent and a large negative reward if it has been destroyed. The exact size of this reward is 10.0 times the number of time-steps remaining when one side has won. This time-based factor provides the agent with an incentive to destroy its opponent as quickly as possible, or to postpone its own demise. In addition, even if there is a draw where neither side wins within 1000 steps, the blue agent still receives a small reward of 1.0 whenever it gets closer to the opponent. This helps to remedy the temporal exploration problem, where it is statistically unlikely that an agent will learn to produce a long sequence of correct actions needed to catch its opponent without the aid of a dense reward. Later, we will see that our novel approach allows us to simplify this reward function while achieving even better results.

In this simple 1v1 environment, the blue agent's observation is a vector consisting of the opponent's relative distance, bearing, heading, closing speed, and cross speed. At each time-step, the agent receives this observation and selects one of the following discrete actions: turn left, turn right, speed up, slow down, hold course. The agent uses an actor-critic reinforcement learning architecture with completely separate value and policy networks. Each network consists of a hidden layer with 36 neurons and ReLU activations, as well as an output layer. The output layer for the policy network contains five neurons (one corresponding to each action listed above) and uses a softmax activation layer with distribution sampling, while the output layer for the value network is a single linear neuron that predicts net reward. Weights are initialized using the method described by He et al. with a truncated normal distribution and based on averaging the number of inputs and outputs (He et al. 2015). Use of the value network for bootstrapping does not improve performance in this particular application, so it is used only as a baseline to reduce variance when computing advantage values (Sutton and Barto 2018).

To compute the gradients needed to train the networks, we use an RMSProp optimizer with learning rate 0.0007, momentum 0.0, and epsilon 1e-10. We use the A3C (Asynchronous Advantage Actor-Critic) parallelization scheme, where 20 workers each run simulations and compute gradients, and these gradients are applied to a centralized learner (Mnih et al. 2016). We have experimented with adding an entropy term to the objective function to help encourage exploration, but this has not been shown to produce a substantial performance improvement. Reward discounting was also determined experimentally to be of limited use in our application, and was therefore omitted. We trained for up to 200,000 episodes, but found 10,000 episodes to be sufficient when training against the straight-flying opponent. In this simplified environment, it has proven difficult to achieve a high win rate against a pure pursuit opponent. However, the reinforcement learning agent does learn to achieve roughly equal numbers of wins and losses (it is able to match, but unable to exceed, the performance of the MA2D scripted opponent). In the next section, we will compare quantitative performance metrics of this machine learning system with those of our hybrid approach.

High-Level Behavior-Based RL

Our novel hybrid approach builds upon this pure reinforcement learning baseline by leveraging traditional AI techniques to produce low-level behaviors and to aid in multi-target allocation. This allows the reinforcement learning agent to focus on the part of the problem for which traditional AI does not offer an out-of-the-box solution. We will continue to discuss the 1v1 case in this section and the next, and will subsequently move on to the multi-agent MvN case, which we will explore in a more advanced AFSIM-based environment.

The 1v1 architecture consists of a high-level controller and a set of low-level scripted behaviors. The high-level controller is a reinforcement learning agent that takes in observations from the environment, and uses a neural net to select behaviors such as "lead pursuit," "lag pursuit," "pure pursuit," or "evade." Once the behavior has been selected, a low-level controller produces output actions with direct control over the fighter's motion. For example, if an autonomous aircraft in a 1v1 engagement selects "pure pursuit," the corresponding low-level behavior script will generate stick-and-throttle actions that cause the plane to head directly toward its opponent. These low-level actions are simply "turn right," "turn left," etc. in the MA2D case, but could also produce continuous control signals needed to pilot high-fidelity aircraft models or even real aircraft.

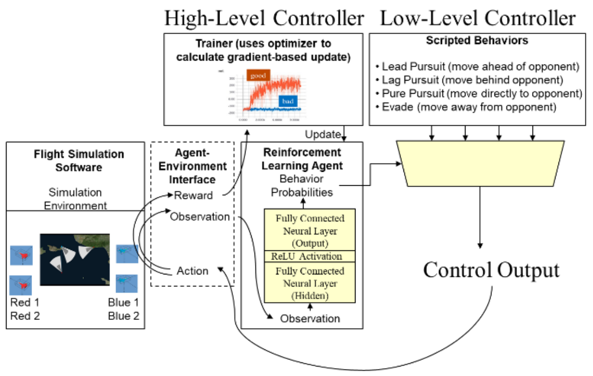

High-Level Controller
Low-Level Controller
Trainer (uses optimizer to calculate gradient-based update)
good
bad
Scripted Behaviors
• Lead Pursuit (move ahead of opponent)
• Lag Pursuit (move behind opponent)
• Pure Pursuit (move directly to opponent)
• Evade (move away from opponent)
Update
Flight Simulation Software
Simulation Environment
Red 1
Red 2
Blue 1
Blue 2
Agent-Environment Interface
Reward
Observation
Action
Reinforcement Learning Agent
Behavior Probabilities
Fully Connected Neural Layer (Output)
ReLU Activation
Fully Connected Neural Layer (Hidden)
Observation
Control Output

Figure 4 - Overview of our hybrid architecture that pairs a high-level reinforcement learner with low-level scripted behavior policies. The reinforcement learning agent selects a scripted behavior, which then produces the actual control output sent to the environment.

The high-level controller's neural net is trained using reinforcement learning. For each training episode, the system keeps track of the high-level behaviors it has selected, the observations that resulted from applying the corresponding low-level actions to the environment, and the rewards that were obtained from the same environment's reward function. After each episode has been completed, we train the agent using a method similar to that described in the previous section.

```
Variables and Constants:
$\pi$ contains the learnable parameters for the policy network
$\theta$ contains the learnable parameters for the value network
N is the number of scripted behaviors

Process:
Randomly initialize the trainable parameters in $\pi$ and $\theta$
For each episode e:
  Initialize trajectory t to an empty list
  For each simulation time-step i:
    Use the high-level agent with policy parameterized by $\pi$ to select a behavior b in [0, N)
    Use the low-level agent corresponding to the selected behavior to select an action
    Append the start state, b, reward, and end state to t
  Apply a gradient update to $\theta$ to reduce the mean square error loss between $V_\theta(s)$ and $\hat{V}(s)$
  Apply a policy improvement gradient update to $\pi$ using the actor-critic method
```

Figure 5 - Pseudocode for the hybrid system consisting of an actor-critic agent and a number of scripted low-level behaviors.

One potential shortcoming of this approach is that the high-level agent must still select a large number of actions within a single episode. This leads to a potentially intractable credit assignment problem (Geron 2017). We now consider three possible remedies, each of which provides a mechanism that restricts the times at which the high-level controller is given a choice to switch to a different behavior.

The first alternative still performs high-level behavior selection at a fixed frequency, but this frequency is lower than the update rate of the low-level controller as illustrated in Figure 6. Similar approaches have been used with pure reinforcement learning (Mnih et al. 2013). In the next section, we will show that this approach provides a slight improvement in performance over the basic hybrid agent, at the expense of increased complexity. We will refer to this add-on as "action repetition."

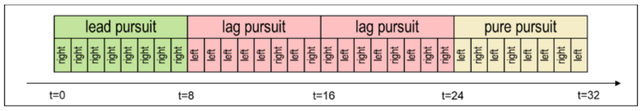

lead pursuit
lag pursuit
lag pursuit
pure pursuit
right right right right right right right right
left left left left left right left right
right left right left right left right right
left right left right left left right left
t=0
t=8
t=16
t=24
t=32

Figure 6 - Fixed-frequency behavior selection with action repetition. In this example, the high-level learner selects four behaviors, but the environment receives 32 low-level actions.

The second alternative uses traditional rule-based AI to specify a termination condition for each behavior. Once a behavior has been selected, execution will continue until this termination condition has been reached, at which time the high-level controller will select a new behavior. This is similar to the "Dynamic Scripting" approach (Toubman et al. 2014). The disadvantage of this approach is that it lacks flexibility. Once the reinforcement learning agent initiates an action, it has no way of terminating this action even if the situation changes entirely at a later time.

The third alternative is illustrated in Figure 7. It includes additional neural nets that restrict the times at which the high-level controller can switch to a different behavior. The agent starts out each episode in the "strategic" state. When the agent is in this state, it selects a low-level behavior using the method described earlier in this section. However, once the agent has selected a behavior, it continues executing this behavior until a low-level "tactical" learner decides to transition control back to the "strategic" learner. Each time the selected low-level controller produces an output action, its corresponding neural net produces probabilities for continuing with the current behavior, or for handing control back to the high-level controller that may then decide to switch to a different behavior. The objective of this approach is to provide improved credit assignment for decisions made by the strategic learner, while still providing the learnable flexibility needed for precision timing of behavior transitions.

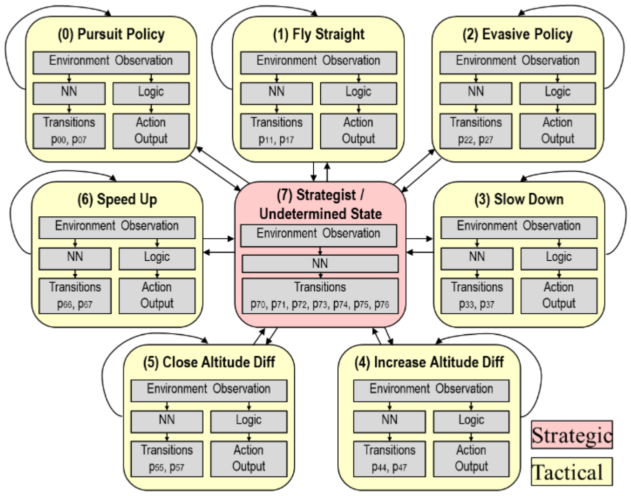

(0) Pursuit Policy
Environment Observation
NN
Logic
Transitions p_{00}, p_{07}
Action Output
(1) Fly Straight
Transitions p_{11}, p_{17}
(2) Evasive Policy
Transitions p_{22}, p_{27}
(6) Speed Up
Transitions p_{66}, p_{67}
(7) Strategist / Undetermined State
Transitions p_{70}, p_{71}, p_{72}, p_{73}, p_{74}, p_{75}, p_{76}
(3) Slow Down
Transitions p_{33}, p_{37}
(5) Close Altitude Diff
Transitions p_{55}, p_{57}
(4) Increase Altitude Diff
Transitions p_{44}, p_{47}
Strategic
Tactical

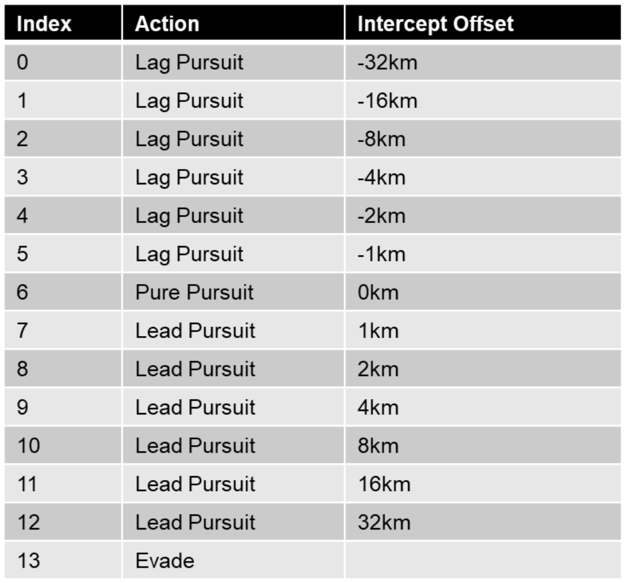
Figure 7 – Depiction of a hierarchical learning agent with seven behaviors as a state machine with eight states. Each state is tied to a separate reinforcement learner. There is one "strategic" learner and there are seven "tactical" learners.

Behavior-Based RL Experiments and Results

Experiments were performed using the same MA2D simulated environment described in the section on a baseline reinforcement learning solution. No changes were made to the observation space. However, the action space for the reinforcement learning agent now consists of the set of behaviors listed in Figure 8. When the neural net selects a lag pursuit, it causes the platform that it is controlling to pursue a point behind its opponent. Pure pursuit and lead pursuit are similar, except that the point is at or in front of the target in each respective case. The evade action causes the platform to turn away from its opponent and increase speed as much as possible so that it can escape. Once a behavior is selected, the corresponding low-level script produces an output in the same action space that was described in the previous section so that an apples-to-apples comparison with the baseline approach can be obtained.

One unexpected benefit of the hybrid approach described in the previous section is that it eliminates the need for dense rewards and reward function engineering. In reinforcement learning applications, it is typical for the environment to provide the agent with a more informative "dense reward" function that provides a more continuous spectrum of outcome desirability than just win or loss. These dense reward functions can be difficult to design, especially as scenarios become more complex. Elimination of this requirement makes the method much easier to apply to new scenarios because it removes the need for this trial-and-error design process.

The hybrid agent is able to learn effectively with only a win-loss reward. Each episode ends when one of the platforms enters the other's weapon engagement zone, at which point a reward of +5000 is given to the platform in firing position, and -5000 to the platform that is about to be fired upon. If neither platform enters the other's engagement zone within 1000 time-steps, a draw is declared and each platform receives 0 reward.



Index	Action	Intercept Offset
0	Lag Pursuit	-32km
1	Lag Pursuit	-16km
2	Lag Pursuit	-8km
3	Lag Pursuit	-4km
4	Lag Pursuit	-2km
5	Lag Pursuit	-1km
6	Pure Pursuit	0km
7	Lead Pursuit	1km
8	Lead Pursuit	2km
9	Lead Pursuit	4km
10	Lead Pursuit	8km
11	Lead Pursuit	16km
12	Lead Pursuit	32km
13	Evade	

Figure 8 - Behaviors available to the reinforcement learning agent. The first 13 behaviors consist of lead, lag, and pure pursuits with various offsets. The final behavior causes the agent to fly away from the opponent.

Experimental results are shown in Figure 9. The baseline result uses pure reinforcement learning. It takes approximately 2,500 episodes of experience before the agent learns to win more episodes than it loses. In contrast, the hybrid approach described in this section uses one of its scripted policies to achieve learning that appears almost instantaneous by comparison. Indeed, the prior knowledge encoded in the scripted policy greatly simplifies the reinforcement learning task. We also experimented with an action repetition variant where the high-level behavior is selected 256 times less frequently than the low-level action. This makes it even easier for the reinforcement learning module to find a winning strategy, because it only needs to select a behavior four times per episode instead of 1000 times (assuming that each episode lasts for 1000 steps).

These results demonstrate that our novel method has advantages over both constituent technologies from which it is composed. It can be much faster than reinforcement learning with a flat architecture, and more effective than a simple scripted (traditional) AI opponent.

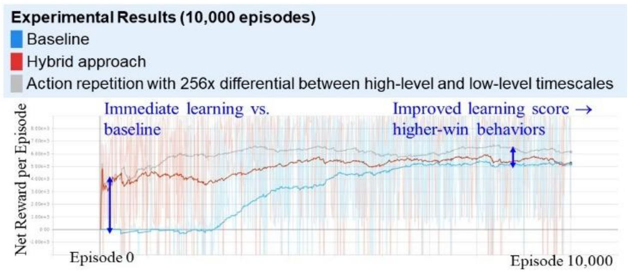


Figure 9 - Results of training the baseline agent, the basic hybrid learner, and an action repetition variant that produces 256 low-level actions per high-level selection.

Multi-Agent Hybrid Learning and Allocation

Having demonstrated that the hybrid RL approach produces vastly improved results in the simple MA2D environment, we apply this AI solution to a more complex decision environment developed with AFSIM. In this scenario, each fighter has five possible actions. It can pursue an opponent, fire a salvo of weapons, provide weapon support, perform evasive maneuvers, or maintain a steady course. When there is more than one opponent, the AI can also select which one to target. In addition to observed enemy positions and velocities, the environment also returns a simple sparse reward at the end of each episode that is +3000 for the winning team, and -3000 for the losing team. For simplicity, a team is declared victorious if it destroys all of the opponents within a time limit. Otherwise, the outcome is declared to be a draw and each team receives zero reward.

In the 1v1 case, our hybrid reinforcement learning agent quickly learns to defeat the scripted AFSIM opponent with 58% win rate, 26% loss rate, and 16% draw rate. Only 50,000 episodes of training are required to reach this level of performance. Due to limitations of the AFSIM-based scenario, we were not able to perform a baseline experiment for comparison as we did for MA2D.

	Nearest Assignment			Hungarian Assignment		
	Win	Loss	Draw	Win	Loss	Draw
1v1	58%	26%	16%	same	same	same
2v2	21%	11%	68%	30%	11%	59%
3v3	6%	4%	90%	13%	4%	83%
4v4	1.2%	1.2%	97.6%	6%	2%	92%
5v5	0.7%	0.9%	98.4%	1.6%	0.6%	97.8%
6v6	0.3%	0.2%	99.5%	0.7%	0.1%	99.2%

Figure 10 - Win/loss/draw results for engagements with up to 12 fighters, with two different target allocation algorithms that we investigated. Each experiment consisted of 1000 trials. These results demonstrate that the hybrid RL agent with Hungarian assignment achieved more wins than losses against a standard AFSIM scripted AI in all experiments, from 1v1 up to 6v6.

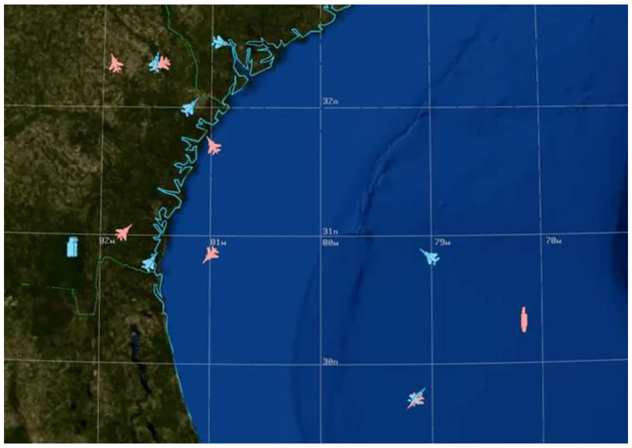


Figure 11 - Muli-agent AFSIM-based environment with 6 blue fighters and 6 red fighters. The blue station on the left and red ship on the right serve only to command their fighters. The fighters fire missiles at one another, and enemy destruction is determined based on missile dynamics and weapon models.

We turn now to the MvN case, where each team contains more than one fighter. Our solution uses traditional target allocation algorithms to handle this part of the problem. First, we compute a matrix with M rows and N columns that contains the distance from each blue agent to each red agent. Then, we either assign each agent to the nearest target, or use the Hungarian algorithm to produce an assignment. If there are more blue fighters than red targets, multiple iterations of the Hungarian algorithm are performed until all blue fighters have been assigned (multiple fighters can be assigned to one target). The following cost matrix is used to formulate this linear sum assignment problem, where D is the distance matrix (with certain rows removed if multiple iterations are needed – those corresponding to already-assigned blue fighters):

$$C_{i,j} = -1.0/(D_{i,j} + 0.001)$$

This effectively reduces the reinforcement learning problem to a 1v1 scenario for each pair. The assignment is re-computed at each time-step so that targets can be re-assigned dynamically. This solution is based on the heuristic assumption that it is better for fighters to engage opponents that are close by. This tends to hold up in practice because rapid destruction of enemy threats involves minimizing the time spent in flight, and therefore the distance travelled. This approach has excellent scalability because an efficient version of the Hungarian algorithm runs in O(n^3) time. It also provides excellent generalizability in the sense that an agent can be trained for a 1v1 engagement, and then used in a much larger scenario. It is challenging to train a reinforcement learning agent to control multiple platforms, and even more challenging to control an arbitrary number of platforms. Although our software framework allows us to train the reinforcement learning agent in up to a 6v6 AFSIM environment, we achieved some interesting results just by training a 1v1 agent and placing it in the 6v6 scenario. Nevertheless, there are still some potential benefits of training within the 6v6 environment. Most importantly, it appears that agents optimized for a 1v1 scenario may be prone to use up all of their missiles very quickly. Training within the 6v6 environment may solve this problem by rewarding agents more frequently when they try to save missiles for later engagements.

Conclusion

When combined with traditional AI approaches, reinforcement learning can produce high-level strategies that are more effective than the previous state of the art. However, a game theoretic perspective is needed to produce truly robust strategies for a pair of adversaries. In this paper, the blue agent learned an approximate best response to a scripted red opponent. This capability is useful in and of itself, but we are also applying empirical game theoretic methods (Lanctot et al. 2017) that allow the reinforcement learning agent to learn without a pre-existing opponent against which to train. This is the subject of a future planned publication.

Acknowledgements

This work was funded by DARPA as part of the Serial Interactions in Imperfect Information Games Applied to Complex Military Decision Making (SI3-CMD) program (contract # HR0011-19-90018). The authors thank Boeing for providing AFSIM scenarios and scripted behaviors. The AFSIM software is property of the Air Force Research Laboratory. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or the Air Force Research Laboratory.

References

Clive, P. D.; Johnson, J. A.; Moss, M. J.; Zeh, J. M.; Birkmire, B. M.; and Hodson, D. D. 2015. Advanced Framework for Simulation, Integration, and Modeling (AFSIM). In Proceedings of the 2015 International Conference on Scientific Computing. Las Vegas: CSREA Press.

DeepMind 2019. AlphaStar: Mastering the Real-Time Strategy Game of StarCraft II. https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii

Frans, K.; Ho, J.; Chen, X.; Abbeel, P.; and Schulman, J. 2018. Meta Learning Shared Hierarchies. Paper presented at the International Conference on Learning Representations. Vancouver, BC, April 30 – May 3.

Geron, A. 2017. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. Sebastopol: O'Reilly.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. Paper presented at the IEEE International Conference on Computer Vision, Santiago, Chile, December 7-13.

Hochreiter, S., and Schmidhuber, J. 1997. Long Short-term Memory. *Neural Computation* 9(8): 1735-1780.

Jaderberg, M.; Czarnecki W. M.; Dunning, I.; Marris, L.; Lever, G.; Castaneda, A. G.; Beattie C.; Rabinowitz, N. C.; Morcos A. S.; Ruderman A.; Sonnerat N.; Green T.; Deason L.; Leibo J. Z.; Silver D.; Hassabis D.; Kavukcuoglu K.; Graepel, T. 2019. Human-level performance in First-Person Multiplayer Games with Population-Based Deep Reinforcement Learning. *Science* 364(6443): 859-865.

Kuhn, H. W. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly* 2(1-2): 83-97.

Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Perolat, J.; Silver, D.; and Graepel, T. 2017. A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning. Paper presented at the 31st Conference on Neural Information Processing Systems. Long Beach, CA, December 4-9.

Lapan, M. 2018. *Deep Reinforcement Learning Hands-On*. Birmingham, UK: Packt Publishing.

Matiisen, T. 2018. The Use of Embeddings in OpenAI Five. https://neuro.cs.ut.ee/the-use-of-embeddings-in-openai-five/

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning. arXiv preprint. arXiv: 1312.5602v1 [cs.LG]. Ithaca, NY: Cornell University Library.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Harley, T.; Lillicrap, T.; Silver D.; and Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning. In Proceedings of the 33rd International Conference on Machine Learning. New York: Association for Computing Machinery.

OpenAI. 2018. OpenAI Five. https://openai.com/blog/openai-five/

Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2006. Adaptive Game AI with Dynamic Scripting. *Machine Learning*, 63(3), 217-248.

Sutton, R.; Precup, D.; and Singh, S. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1-2), 181-211.

Sutton, R. S., and Barto, A. G. 2018. Reinforcement Learning: An Introduction. Cambridge: The MIT Press.

Toubman, A.; Roessingh, J. J.; Spronck, P.; Plaat, A.; and Herik, J. 2014. Dynamic Scripting with Team Coordination in Air Combat Simulation. In Proceedings of the 27th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems. Kaohsiung: Springer International.

Vinyals O.; Ewalds T.; Bartunov S.; Georgiev P.; Vezhnevets A. S.; Yeo M.; Makhzani A.; Kuttler H.; Agapiou J., Schrittwieser J.; Quan J.; Gaffney S.; Petersen S.; Simonyan K.; Schaul T.; Hasselt H.; Silver D.; Lillicrap T.; Calderone K.; Keet P.; Brunasso A.; Lawrence D.; Ekermo A.; Repp J.; and Tsing R. 2017. StarCraft II: A New Challenge for Reinforcement Learning. arXiv preprint. arXiv: 1708.04782 [cs.LG]. Ithaca, NY: Cornell University Library.