Applying Complexity Science with Machine Learning, Agent-Based Models, and Game Engines: Towards Embodied Complex Systems Engineering

Michael D. Norman(✉), Matthew T. K. Koehler, Jason F. Kutarnia, Paul E. Silvey, Andreas Tolk, and Brittany A. Tracy

The MITRE Corporation, Bedford, MA 01730, USA
mnorman@mitre.org

Abstract. The application of Complexity Science, an undertaking referred to here as Complex Systems Engineering, often presents challenges in the form of agent-based policy development for bottom-up complex adaptive system design and simulation. Determining the policies that agents must follow in order to participate in an emergent property or function that is not pathological in nature is often an intensive, manual process. Here we will examine a novel path to agent policy development in which we do not manually craft the policies, but allow them to emerge through the application of machine learning within a game engine environment. The utilization of a game engine as an agent-based modeling platform provides a novel mechanism to develop and study intelligent agent-based systems that can be experienced and interacted with from multiple perspectives by a learning agent. In this paper we present results from an example use-case and discuss next steps for research in this area.

Keywords: Artificial intelligence · Complexity · Emergence
Reinforcement learning

1 Introduction

The application of the science of complexity, specifically attempts at engineering complex adaptive systems (CAS) [9], present the scientist or systems engineer with the non-trivial task of crafting agent policies that achieve the desired emergent properties of the system, assuming they are known. Complex Systems Engineering [8] addresses this new set of challenges, but requires a significantly extended set of methods, many of them from the domain of simulation, and in particular agent-based approaches which can be enriched by artificial intelligence methods [20]. We can use these machine learning (ML) techniques to effectively derive agent-based policies that maximize the probability of the desired

A. J. Morales et al. (Eds.): ICCS 2018, SPCOM, pp. 173–183, 2018.
https://doi.org/10.1007/978-3-319-96661-8_18

emergent effects (via feedback of information on agent decisions during training) while minimizing the probability of emergent outcomes we deem negative, which we refer to as pathological emergence.

The focus of this work is on the application of reinforcement learning (RL) to evolve and engineer collections of agents well-suited to producing and maintaining an emergent property of interest. Ongoing work cross-pollinating the fields of agent-based modeling and ML that these authors are aware of are focused on utilizing ML to derive agent policies from existing datasets [19], whereas in this work we utilize a game engine to enable an embodied RL paradigm [14] based solely on experience gained in the model of the complex system itself.

The recent integration of Unity [22], a popular game engine, with TensorFlow, an open source machine learning framework [18] has created an opportunity to build agent-based models that can learn from their own embodied experience. Our preliminary work here has enabled us to plot a course for using this technology to employ imitation learning from observing human-driven embodied experiences within their world, as well as new depths of human-machine teaming applied to agent-based modeling through true embodiment of the human inside of the model using virtual reality (VR). The implications for the integration of the CAS modeling field and the serious gaming field appear on the surface to be potentially far-reaching, including training autonomous vehicles [9], and perhaps virtual avatars capable of passing the Turing test.

In what follows, we explore the applicability of such a technology stack to designing an agent-based search and rescue model. Development of polices and results will be discussed. Future work will explore the impact of further agent heterogeneity, scaling, and higher-fidelity environments.

2 Technological Enablers

Real-time game engines such as Unity and Unreal [3], which are capable of producing life-like experiences, high-definition head-mounted displays (HMDs) and controllers which are designed to bring hand and body presence into virtual environments such as the HTC Vive [5], and consumer-grade graphical processing units (GPUs) designed for high-performance parallel processing such as those made by NVIDIA [10], have put the power to create agents trained by human interaction in virtual worlds in the average computer enthusiast's hands. Real-time generated virtual worlds could become ideal training grounds for the development of strategies for learning and problem solving in both virtual and real-world environments, as evidenced by the investment being made in companies who are using game-like environments to train artificial intelligence (AI) for real world autonomous car deployment [1].

Our work here represents an initial first step in this direction for the Applied Complexity Science community, and demonstrates the feasibility of agent-based modeling in a high-fidelity game environment combined with an RL mechanism. Future work on imitation learning may leverage alternative learning processes as appropriate.

3 Agent Policy Development

Agent-based modeling policies are challenging to approach for many reasons, not the least of which is the sensitivity of complex systems to their initial conditions [15], which means a successful policy must be robust enough to withstand latent, unpredictable, and path-dependent edge-cases. This leads a CAS designer or modeler to seek absolute minimalism in their policy implementations in order to avoid generating inexplicable pathological emergence; from simple rules complexity emerges. In this paper we present a different method of developing such robust policies via the application of ML techniques within a game engine environment. The results of a search-and-rescue use-case implemented using the Unity game engine and TensorFlow ML framework are discussed.

3.1 Determining Agent Policies When Modeling Complex Adaptive Systems

One of the best examples of a well-known and human-understandable agent policy set is that of the boids algorithm [11]. Through tuning of the simple rules of attraction, repulsion, and alignment, a flock of birds emerges. Previous work leveraging emergent properties of the boids rule-set has shown the emergence of a flock is an embodied phenomenon which depends greatly on context and scale, as well as very minimalist tweaking of the known, presumably 'stable' rule sets [9]. By stable, we mean capable of producing an emergent stability at the scale of the system. For example, a flock does not cease to be a flock just because there is instability among the population. This is sometimes known as metastability [6], but for our purposes, stability is an adequate term. It is also worth noting that a truly well-engineered CAS would demonstrate the properties of antifragility [16], but this is a lofty goal and will remain unobtainable until the science of complexity is able to catch its tools up to its theory. This paper is but a small attempt to move forward on that front.

If we are willing to accept a certain level of opacity in our agent policy sets and we are interested in problems of embodiment, then the creation of a capability to develop agent-based policies via ML in a 3D game engine is an excellent means of exploring the emergent dynamics of a complex system.

3.2 Reinforcement Learning

RL has a long history as a means of developing learning agents, and some work has even focused on the emergence of cooperation in relatively low-dimensional systems such as Pong [17], as well as within our own experimental framework [14], but its use in the development of agent-based modeling policies for studying and simulating CAS on a larger scale is still quite nascent, with only hints of the implications envisioned thus far [4].

The agents are trained via an RL implementation, so they proceed through many iterations of the simulation before training is considered complete. Training terminates after a certain number of time-steps. We proceed to repeat this

process a number of times and compare outputs. We are then able to insert the newly-trained agent into the simulation to confirm it functions as expected. Selecting which trained agent to instantiate as multiple agents in the system (assuming homogeneous agents) in order to achieve a desired aggregate behavior is currently a manual process.

Five agents simultaneously inhabit the environment, interacting with one another and contributing to the learning of a single shared policy during each training run; the policy also controls each of the agents. The agent policy initially starts out doing very random things, and learns optimal rules through reinforcement-based trial-and-error.

3.3 Proximal Policy Optimization

The RL algorithm implemented at the core of the reinforcement learners is known as Proximal Policy Optimization (PPO). PPO was developed and released by OpenAI [12], and is their default algorithm for RL.

3.4 Performance

The measure of performance we will focus on for this use-case is simply Cumulative Reward at the conclusion of training. We will consider the cumulative reward across all agents at the conclusion of a given policy's training run to be the measure of emergent system performance, despite the fact that it is literally the sum of its parts. We can assume that a stable policy has been learned when the cumulative reward is positive. More interesting measures of emergence will be applied in future work.

4 Technology Stack

For this effort, Unity3D version 2017.3.1f1 was used in combination with TensorFlow version 1.6.0 and TensorFlowSharp version 1.6.0-pre1. Anaconda 5.1 and Python 3.6 were used to create a virtual environment and run Unity's external training API, respectively. The open source ml-agents package [23] from Unity Technology was the critical piece that integrated all of these technologies. Early pilot work involved training with discrete NVIDIA GTX 1080 GPUs, but these are optional. The ml-agents package was used to train a TensorFlow neural network graph from the RL experience of embodied agents situated in a game world.

5 Search and Rescue Use-Case

The open-source ml-agents package [23] was adapted to emulate autonomous search and rescue in a contested environment. In our simulation, the agents were cast as autonomous helicopters attempting to collect soldiers within a contested

environment. A screen capture of the training environment is shown in Fig. 1. We implemented constrained peripheral vision for the agents as we consider it to be a rudimentary emulation of partial observability. With five agents acting and effecting concurrently in the simulation, the presence of multiple autonomous helicopter agents in the same "airspace" seeking the same basic objective was considered analogous to a dynamic multi-agent rescue situation in a theater of battle or other contested environment containing bad actors. Finally, the unpredictable distribution and periodic redistribution of soldiers within the rescue area required the agents to learn responsive behaviors tailored to stochastic conditions that cannot be anticipated in any actionable sense, only responded to prudently (positively rewarding) or imprudently (negatively rewarding): rampant exploration, but also care when maneuvering to a rescue so as not to encounter hostiles. To establish a baseline understanding of helicopter agent behavior, as well as to demonstrate the technological method first and foremost, we modeled five helicopters endeavoring to rescue friendly soldiers and avoid hostile soldiers, but a logical extension of this work could include dividing helicopters into discrete teams, positively rewarding agents when they rescue affiliated soldiers and negatively rewarding them for encounters with hostile soldiers. We populated the model with 20 friendly and 20 hostile soldiers. However, for every friendly soldier rescued, another is promptly respawned (this is not the case for hostile soldiers). The game area is reset at regular intervals to 20 and 20 soldiers of each category, and also randomly reassigns origin positions to the autonomous rescue helicopters.

We executed 10 experimental runs of our Search and Rescue application for 50,000 time steps.

In pilot runs, the initial reward structure (+1 for friendlies and −1 for hostiles) was not sufficient to produce the desired behavior of "colliding" with friendly soldiers and avoiding encounters with hostile soldiers. The helicopters' form entails that they often collide with hostile soldiers unintentionally from the side while they are still on a lateral trajectory (the hostiles are out-of-sight in the agents' periphery). In these initial trials, helicopters learned to remain stationary because they were frequently, though incidentally, negatively rewarded when they collided with clusters of bad soldiers.

In response, the reward structure was modified to negatively reward the helicopters with −1 points for every time step when they were not colliding with a soldier, in order to discourage inactivity. At this point, the helicopters learned to fly in lateral translation across the rescue area, thereby colliding with as many objects for +1 and −1 rewards as possible (either benefiting by +1 points or experiencing a −1 reward, the same result as if they had remained static). To discourage this lateral translation strategy, the reward structure was modified to negatively reward collision with a wall (on the bounds of the rescue area) at −100 points. Finally, the reward for collecting friendlies was increased to +500 points to offset the −1 reward increment applied at every non-collision timestep. Instead of proportional negative reward for collecting hostiles, the value was reduced to only one-fifth of the positive reward, −100 points. Finally, the simulation had

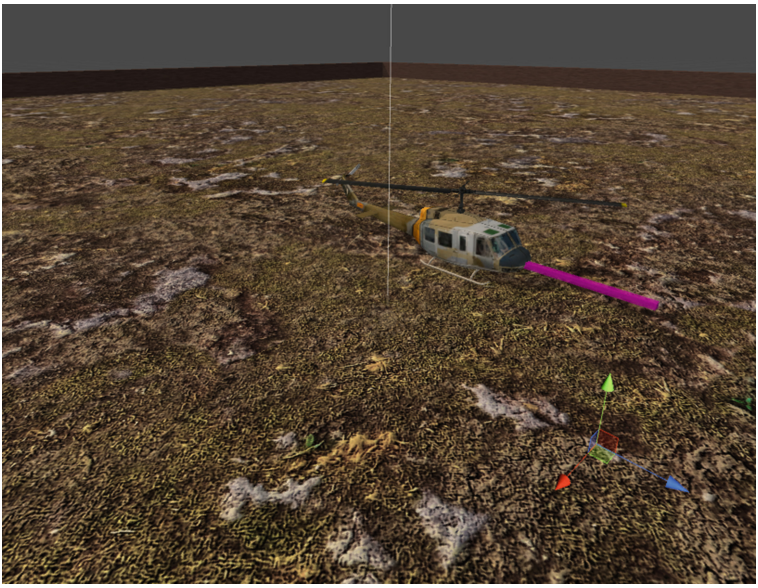

Fig. 1. Training environment screen capture.

achieved a form in which the reward structure was truly conducive to helicopters learning to explore the game area and attempting to rescue friendly soldiers.

The hyperparameter settings used for this experiment can be found in Table 1. These were informed by the recommended defaults of the ml-agents package.

6 Results and Discussion

Interesting summary results from the training can be found in Figs. 2 and 3, which show the Cumulative Reward and Entropy, respectively, across all runs.

Learners emerged that were capable of performing the task at hand, but whose entropy levels were still quite high at the termination of the training round that created them. We interpret this result as follows: these agents, although having constructed successful policies for survival, had also created a strategy which involved continual exploration. The implications of a novelty-seeking agent for a search and rescue use case seem rather tangible, as this 'injection of randomness' [16] may lead to nudging an emergent system dynamic out of an unsuccessful attractor's area of influence. For a summary of dynamical systems theory in context of agent-to-agent influence see Liebovitch et al. [7]. The varying levels of entropy across runs that was exhibited at the conclusion of training is indicative of the pervasiveness of exploration as a fruitful strategy. In fact the agent policy with highest total cumulative reward (as shown in the yellow line of Fig. 2) did not have the lowest entropy at the conclusion of its training, as can be seen in the dark blue line of Fig. 3.

Table 1. Hyperparameters.

Hyperparameter	Value
Batch size	1024
Beta	5.00E-03
Buffer size	10240
Epsilon	0.2
Gamma	0.99
Hidden units	128
Lambd	0.95
Learning rate	3.00E-04
Max steps	5.00E+04
Memory size	256
Normalize	FALSE
Num epoch	3
Num layers	2
Time horizon	64
Summary freq	1000
Use recurrent	FALSE

7 Future Work

The general need for research agendas to support better support of system of systems challenges [21] and complex systems engineering [2] has been identified and builds the general frame for this section, which provides particular research topics derived from the machine learning and agent-based metaphor applied in the work presented in this paper.

7.1 Hyperparameter Tuning

Hyperparameter tuning is a fundamental part of machine learning and must be carefully considered if one hopes to construct a high-performing autonomous agent. There are a number of intelligent algorithms that could be incorporated into our training workflow but investigation into the most effective was outside the scope of this paper. Recent research in simulation calibration may be applicable to support these ideas in future efforts [24].

The naïve approach is a simple grid search, but this breaks down when there are numerous parameters to tune or the model's sensitivity is nonlinear with respect to the hyperparameter, which is the case with neural networks. This can be overcome by using a logarithmically-scaled grid where sensitive areas of the hyperparameter space are heavily sampled. Other more advanced methods which attempt to minimize the number of trials necessary to converge on the optimum parameter values include Bayesian optimization and Sequential Model-based Algorithm Configuration. Future work will include more robust hyperparameter exploration.

7.2 Apply Machine Learning to Manage Design of Experiments

Although a very simple measure of performance was used with a human-in-the-loop to judge agent employability, future work may shift the focus on deployment of ML in the service of suggesting what sort of metrics might be useful, as well as determining which training runs have developed the most appropriate learner for a given deployment context. Implementing a ML agent to select the best trained agent for model deployment and determining heterogeneity ratios.

7.3 Evolve Measures of Performance

A relatively simple measure was used to grade the performance of the emergent collective that was quite simply the sum of the system's parts. Future work will focus on putting such aggregate measures in greater context and exploring scenarios that tend to exhibit, and benefit from, emergent properties.

7.4 Imitation Learning and Virtual Reality

Imitation learning allows one to harness the human brain's natural ability to process and integrate multiple (and often multi-modal) informational streams presented in complex environments and then make decisions under uncertainty (the global state of universe is not known to any given agent). Learning via imitation methods avoids the difficult task of explicitly stating the rules and their relative priority in all probable situations that an expert human is following. We believe that fully immersive, embodied experiences will enable a virtuous cycle of human-machine teaming where humans are training artificial agents and vice-versa.

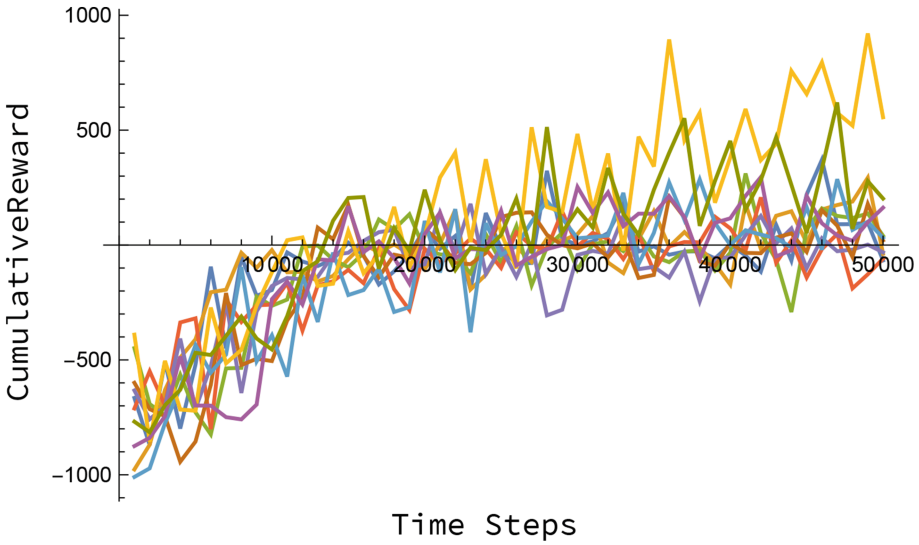

Fig. 2. Cumulative reward across 10 training runs of a stylized Search and Rescue scenario

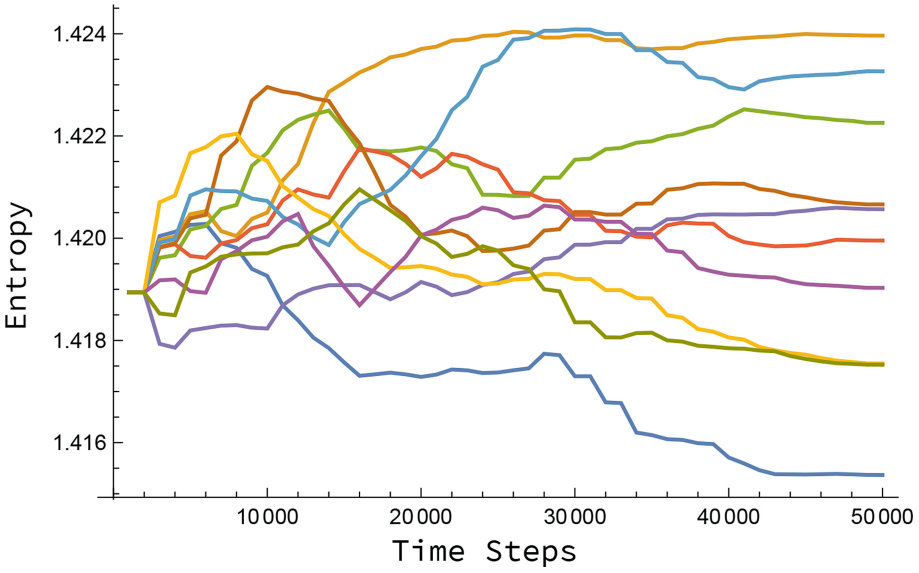

Fig. 3. Entropy across 10 training runs of a stylized Search and Rescue scenario

8 Conclusion

We have shown that agent-based modeling can be conducted using reinforcement learning agents within a fully-featured 3D game environment. Our trained agents, on the whole, were successful in accomplishing their mission. We have learned that these initial attempts at training, while showing promise, still produce a range of behavioral features and resulting cumulative rewards across runs. This indicates there is much work yet to be done to understand how the complexities of the environment, the interactions between agents, and the learning hyperparameters all come together to produce the results we have seen.

The INCOSE complexity primer [13] recommends the use of a variety of methods and approaches from a variety of disciplines that have to cope with complexity in support of analyzing, diagnosing, modeling, and synthesizing complex systems. We followed these recommendations in our research and applied machine learning, agent based models, and simulation-enabling game engines in support of architectural analysis challenges, demonstrating how the combination of these methods supports the complex systems engineering processes captured in the use case of autonomous search and rescue. This successful application shows the usefulness of this approach for a new category of engineering challenges requiring the application of complexity science.

Acknowledgements and Disclaimer. The work presented in this paper was partly supported by the MITRE Innovation Program. The authors wish to specifically acknowledge Sham Chakravorty and Muhammad Sungkar for their contributions to this effort. The views, opinions, and/or findings contained in this paper are those of The MITRE Corporation and should not be construed as an official government position, policy, or decision, unless designated by other documentation. It is approved for Public Release; Distribution Unlimited. Case Number 17-3081-17.

References

1. Cognata. http://www.cognata.com
2. Diallo, S., Mittal, S., Tolk, A.: Research agenda for next-generation complex systems engineering. In: Emergent Behavior in Complex Systems Engineering: A Modeling and Simulation Approach, pp. 379–397 (2018)
3. Epic. https://www.unrealengine.com
4. Holland, J.H., Miller, J.H.: Artificial adaptive agents in economic theory. Am. Econ. Rev. **81**(2), 365–370 (1991)
5. HTC. https://www.vive.com
6. Kelso, J.S.: Dynamic Patterns: The Self-organization of Brain and Behavior. MIT Press, Cambridge (1997)
7. Liebovitch, L.S., Peluso, P.R., Norman, M.D., Su, J., Gottman, J.M.: Mathematical model of the dynamics of psychotherapy. Cogn. Neurodyn. **5**(3), 265–275 (2011)
8. Norman, M.D.: Complex systems engineering in a federal it environment: lessons learned from traditional enterprise-scale system design and change. In: 2015 9th Annual IEEE International Systems Conference (SysCon), pp. 33–36. IEEE (2015)

9. Norman, M.D., Koehler, M.T., Pitsko, R.: Applied complexity science: enabling emergence through heuristics and simulations. In: Emergent Behavior in Complex Systems Engineering: A Modeling and Simulation Approach, pp. 201–226 (2018)
10. NVIDIA. https://www.nvidia.com
11. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioral model. In: ACM SIGGRAPH Computer Graphics, vol. 21, pp. 25–34. ACM (1987)
12. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. ArXiv e-prints (2017)
13. Sheard, S., Cook, S., Honour, E., Hybertson, D., Krupa, J., McEver, J., McKinney, D., Ondrus, P., Ryan, A., Scheurer, R., et al.: A complexity primer for systems engineers. INCOSE Complex Systems Working Group White Paper (2015)
14. Silvey, P.E., Norman, M.D.: Embodied cognition and multi-agent behavioral emergence. In: Proceedings of the Ninth International Conference on Complex Systems (ICCS 2018) (2018, in press)
15. Strogatz, S.: Sync: The Emerging Science of Spontaneous Order. Penguin, London (2004)
16. Taleb, N.N.: Antifragile: Things That Gain from Disorder, vol. 3. Random House Incorporated (2012)
17. Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., Vicente, R.: Multiagent cooperation and competition with deep reinforcement learning. PLoS ONE **12**(4), e0172395 (2017)
18. TensorFlow. https://www.tensorflow.org
19. Tolk, A.: The next generation of modeling & simulation: integrating big data and deep learning. In: Proceedings of the Conference on Summer Computer Simulation, pp. 1–8. Society for Computer Simulation International (2015)
20. Tolk, A., Diallo, S., Mittal, S.: Complex systems engineering and the challenge of emergence. In: Emergent Behavior in Complex Systems Engineering: A Modeling and Simulation Approach, pp. 79–97 (2018)
21. Tolk, A., Rainey, L.B.: Toward a research agenda for m&s support of system of systems engineering. In: Modeling and Simulation Support for System of Systems Engineering Applications, pp. 581–592 (2015)
22. Unity3D. https://www.unity3d.com
23. Unity3D. https://github.com/Unity-Technologies/ml-agents
24. Xu, J.: Model calibration. In: Advances in Modeling and Simulation: Seminal research from 50 Years of Winter Simulation Conferences, pp. 27–46 (2017)