Neural MMO v1.3: A Massively Multiagent Game Environment for Training and Evaluating Neural Networks*

Extended Abstract

Joseph Suarez
Massachusetts Institute of Technology
Cambridge, MA
jsuarez@mit.edu

Yilun Du
Massachusetts Institute of Technology
Cambridge, MA
yilundu@mit.edu

Igor Mordach
Google Brain
Mountain View, CA
imordach@google.com

Phillip Isola
Massachusetts Institute of Technology
Cambridge, MA
phillipi@mit.edu

ABSTRACT

Simulated games have become a dominant platform for multiagent intelligence research in recent years. Previous works have succeeded on arcade, first person shooter (FPS), real-time strategy (RTS), and massive online battle arena (MOBA) games. Our work considers massively multiplayer online role-playing games (MMORPGs or MMOs), which capture several complexities of real-world learning that are not well modeled by any other game genre. We present a massively multiagent game environment inspired by MMOs and demonstrate that simple policy gradient methods produce interesting emergent exploration and specialization behaviors.

KEYWORDS

Engineering agent-based simulations; Emergent behavior

ACM Reference Format:
Joseph Suarez, Yilun Du, Igor Mordach, and Phillip Isola. 2020. Neural MMO v1.3: A Massively Multiagent Game Environment for Training and Evaluating Neural Networks. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 3 pages.

1 INTRODUCTION

From arcade to FPS to RTS and MOBA, the use of increasingly complex game environments has accelerated progress in deep reinforcement learning (RL) in recent years [1–5, 7]. MMOs simulate self-contained macrocosms with large, variable numbers of players and realistic social strategy. However, standard assumptions made about observation/action spaces and infrastructure currently prevent RL from scaling to MMOs. Our key contributions are:

(1) Neural MMO as a fully open-source and actively supported environment for multiagent research
(2) Pretrained policies with the associated distributed training code and utility libraries for reproducibility [Video]
(3) Stand-alone scalable infrastructure and performance logging for massively multiagent environments
(4) Stand-alone methods for interfacing with complex observation and action spaces in multiagent environments

*[Full Paper][Project Page][Code] This publication summarizes v1.0-v1.3. * and # denote current author affiliations. Work for v1.0 [6] was performed at OpenAI and work for v1.3 was performed at MIT

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

2 NEURAL MMO

Neural MMO is a massively multiagent environment for artificial intelligence research. Agents forage for resources and engage in strategic combat within a *persistent* game world that is never reset during training. Our environment implements a progression system inspired by traditional MMOs and a full 3D renderer for visualizations. Figure 1 details the three core game systems.

Environment Representation: Neural MMO is laid out on 2D tile map that is procedurally generated by thresholding a Perlin ridge fractal. Agents are added (*spawn*) to the environment constantly at a fixed rate. They may move about the grass and forest tiles of the game map, but stone and water are impassible. The map is also surrounded by a lethal lake of lava.

Resource System: Agents spawn with 10 (configurable) units of food, water, and health. At every time step, agents lose 1 food and 1 water. If agents run out of food or water, they begin losing health. If agents are well fed and well hydrated, they begin regaining health. In order to survive, agents must quickly forage for food, which is in limited supply, and water, which is infinitely renewable but only available at a smaller number of pools. Thus, the objective is to simultaneously navigate and forage for food/water in the presence of upward of one hundred agents attempting to do the same.

Combat System: Agents can attack each other with three different styles – Range, Mage, and Melee. Accuracy and damage are determined by the attack style and the combat stats of the attacker and defender. This system enables a variety of strategies. Agents more skilled in combat can assert map control, locking down resource rich regions for themselves. Agents more skilled in maneuvering can succeed through foraging and evasion. The goal is to balance between foraging safely and engaging in dangerous combat to pilfer other agents' resources and cull the competition.

Progression System: Progress in real MMOs varies on two axes: soft advantage gained through strategic/mechanical talent and hard numerical advantage gained through skill levels/equipment. In Neural MMO, agents progress their abilities through usage. Foraging for food and water grants experience in the respective Hunting and Fishing skills, which enable agents to gather and carry more resources. A similar system is in place for combat. Agents gain levels in Constitution, Range, Mage, Melee, and Defense through fighting other agents. Higher offensive levels increase accuracy and damage while Constitution and Defense increase health and evasion. The global scale of experience awarded for each action is configurable to enable any game progression time scale from a few minutes to thousands of hours.

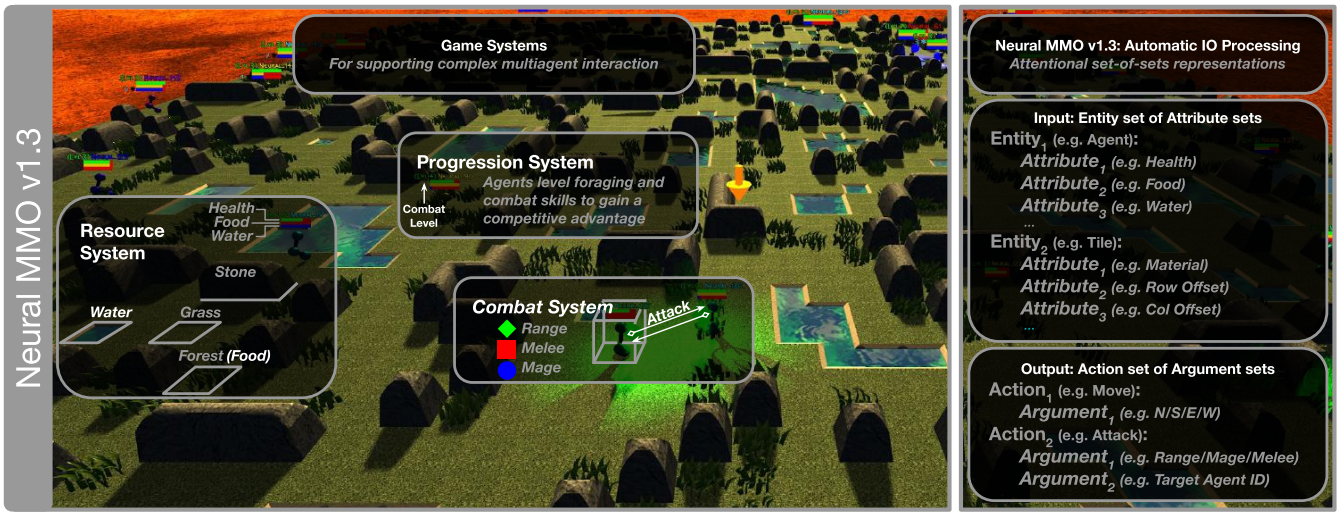

Figure 1: Neural MMO is a massively multiagent environment for AI research. Agents compete for resources through foraging and combat. Observation and action representation in local game state enable efficient training and inference. A 3D Unity client provides visualizations for interpreting learned behaviors. The environment, client, training code, and policies are fully open source, officially documented, and actively supported through a live community Discord server.

3 THE IO PROBLEM

Small scale RL environments typically provide input observations as raw data tensors and output actions as low-dimensional vectors. More complex environments may contain variable length observation and action spaces with mixed data types: standard architectures that expect fixed length tensors cannot process these IO spaces. Our solution to this problem parameterizes the observation space as a set of entities, each of which is parameterized by a set of attributes (Figure 1, Input). We automatically generate attentional networks to select variable length action arguments by keying against learned entity embeddings (Figure 1, Output).

3.1 The Input Problem

We define local game state by the set of observable objects or *entities* (e.g. agents and tiles), each of which is parameterized by a number of local properties or *attributes* (e.g. health, food, water). At compile time, embedding networks $e_{x_1}, e_{x_2}, \ldots$ are defined for each attribute. We also define soft attention functions $\{f_{y_j}\}$ and g to be used later.

Input	*Entity set of attribute sets*
$X := \{x_i\}$	Define attributes x_i
$Y = \{e_{x_i}(x_i)\}$	Embed attributes for each entity
$Z = \{f_{y_j}(\{x_{1,\ldots,n}\} \subseteq Y)\}$	Soft attend f_{y_i} to attribute subsets
$o = g(Z)$	Soft attend g to entity embeddings

At run time, we project $x_1, x_2, \ldots$ to fixed length attribute embedding vectors $y_1, y_2, \ldots$ using embedding layers e_{x_i}. Soft attention layers f_{y_j} aggregrate across the attribute embeddings of each entity to produce a representation z_i for each entity. Finally, an attentional layer g aggregates across all entity embeddings Z to produce a flat observation embedding o. We return both o and Z.

3.2 The Output Problem

We define agent decision space by a list of actions, each of which takes a list of arguments. Actions are callable function references that the environment can invoke on the associated argument list in order to execute an agent's decision, such as Move → [North] or Attack → [Melee, Agent ID]. At compile time, the user specifies a hard attentional architecture h. We reuse the Input module to generate embedding layers for all arguments.

Output	*Action list of argument lists*
$A := [A_i : [a_1, \ldots, a_n], \ldots]$	Define arguments A_{ij}
$B_{ij} = e_{a_{ij}}(A_{ij})$	Embed arguments to B_{ij}
$\arg_{A_i} = h(o, \{B_i, \widetilde{z}_i \subseteq Z\})$	Hard attend f to arguments

At run time, we convert the hidden state o of the main network into an action list of argument lists. To do so, we embed candidate arguments for all actions A_{ij} to fixed length vectors B_{ij} using $e_{a_{ij}}$, similarly to as in the Input module. As entities can be arguments, we will also consider Z from the input network. For each candidate action-argument A_{ij}, we compare embeddings $\{B_i, \widetilde{z}_i\}$ to the hidden state using the attentional function h to produce a softmax distribution. Sampling from this distribution yields a hard attentional choice $\arg_{A_i}$ over arguments. Finally, we return game actions paired with their selected argument lists.

4 CONCLUSION

Neural MMO has been in development for over two years and is now fully open source with dedicated setup, documentation, and tutorial pages, an active Discord community support server with over 100 current members, and major updates every 3-4 months. We plan to support Neural MMO as a robust platform for multiagent research at small and large scale and will continue its development going forward. We hope that our solutions will prove useful to others developing massively multiagent systems.

REFERENCES

[1] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. 2019. Emergent Tool Use From Multi-Agent Autocurricula. (2019). arXiv:cs.LG/1909.07528

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub W. Pachocki, Michael Petrov, Henrique Pond'e de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *ArXiv* abs/1912.06680 (2019).

[3] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. 2018. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281* (2018).

[4] OpenAI. 2018. OpenAI Five. https://blog.openai.com/openai-five/. (2018).

[5] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[6] Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. 2019. Neural MMO: A Massively Multiagent Game Environment for Training and Evaluating Intelligent Agents. *CoRR* abs/1903.00784 (2019). arXiv:1903.00784 http://arxiv.org/abs/1903.00784

[7] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. 2019. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/. (2019).