Neural MMO: Massively Multiagent Simulation and Learning

by

Joseph Suarez

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2024

Authored by:	Joseph Suarez Department of Electrical Engineering and Computer Science May 18, 2024
Certified by:	Phillip Isola Associate Professor of EECS, Thesis Supervisor
Accepted by:	Leslie A Kolodziejski Professor of Electrical Engineering Graduate Officer, Department of EECS

Neural MMO: Massively Multiagent Simulation and Learning

by

Joseph Suarez

Submitted to the Department of Electrical Engineering and Computer Science on May 18, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Neural MMO is a massively multi-agent environment for reinforcement learning research. It is designed to push the boundaries of environment complexity while maintaining computationally efficiency for academic research. Agents in Neural MMO can forage for a variety of resources, engage in strategic combat with each other, defeat scripted enemies for loot, level-up various interdependent professions, acquire tools, weapons, equipment, etc., and exchange items on a global market. Neural MMO was among the first many-agent simulators for reinforcement learning research, and it is still unique among environments today. To my knowledge, no other project provides large agent populations, high per-agent complexity, and efficient simulation at once. These properties make Neural MMO a suitable environment for a variety of research topics in multi-agent learning that would be difficult to explore without such a simulator. The environment can process 128 agents at up to 25x real-time on a single CPU core, totaling 3,000 agent-steps per second. This speed is owed to simulation techniques borrowed from the games industry. In the course of developing Neural MMO, I made several adaptations for the specifics of reinforcement learning, such as the two-layer structure of Neural MMO's observations and actions and the efficient internal data representation. The contributions of this work include these adapted methods as general-purpose tools for designing RL environments. Through my own experiments and from the results of a series of competitions that I hosted on Neural MMO, we have seen agents capable of long-term coherent strategies, multi-tasking across various objectives, and conditioning for specific goals. The largest discovery of this project has been the extent to which standard reinforcement learning methods with limited compute are able to solve complex tasks. Neural MMO is free and open-source software under the MIT license with comprehensive documentation at neuralmmo.github.io and a 1000+ member community Discord.

Thesis supervisor: Phillip Isola
Title: Associate Professor of EECS

Acknowledgments

For my loving parents who have always supported me

This work was supported in part by:

Andrew (1956) and Erna Viterbi Fellowship

Alfred P. Sloan Scholarship (G-2018-10127)

ONR MURI grant N00014-22-1-2740.

FA8750-19-2-1000

Training compute for Neural MMO 2.0 baselines was provided by Carper AI and Eleuther AI. Development for 2.0 was an open-source project under CarperAI led by Joseph Suarez and managed by Louis Castricato. Web client by Parametrix.AI with artwork by Lucas de Alcântara. Technical documentation by Rose S. Shuman in collaboration with the development team. Engine work for 2.0 by David Bloomin. Special thanks to Kyoung Whan Choe for major contributions to development and ongoing environment support.

Biography

I received my M.S. in EECS from MIT in 2023 and my B.S. in computer science and AI from Stanford in 2019. I took 6 months off from my undergraduate degree in 2018 to intern at OpenAI, where I formalized and released the first public version of Neural MMO. Before that, I worked on dynamic visual question architectures with Justin Johnson in Prof. Fei-Fei Li's group in 2017. And before that, I worked on language modeling architectures in a program run by Andred Ng's lab.

I have run 4 public competitions on Neural MMO with various partners that have received a total of 3700+ submissions from 1200+ participants. Outside of competitions and pure development, I run the Neural MMO community Discord, which currently has 1000+ members. I also run the LIFE monthly group meeting (Learning in Foundation Environments) with invited talks from researchers and developers working on new environments, platforms, and associated tooling. A brief list of amusing AI things I have done:

- My website has always read "I, Joseph Suarez, am the primary author of Neural MMO. I plan to continue development for at least the duration of my EECS PhD at MIT or until someone convinces me that there is a better way to solve AGI." I fully expected some group better resourced than a single Ph.D student to build a better version of Neural MMO half way through my degree. Since nobody did, I kept working on it.

- I got fed up with the "but what is AGI" definitions game in 2017, so I came up with this one: *AGI has been solved when I can cast fireball.* This is just a rephrasing of the old *sufficiently advanced technology is indistinguishable from magic.* You're not supposed to think about it too hard; the point is to stop philosophizing and start building.

- I published a short paper titled "GAN You Do the GAN GAN?" on April 1, 2019. Turns out you can train GANs on the weights of other GANs to produce a GAN GAN.

- The original Neural MMO map renderer was all one shader. Yes, that includes the geometry itself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A Brief History of Neural MMO

I first started a project resembling Neural MMO in 2015 during the spring of my senior year of high school. This attempt did not get very far. I started the project from scratch again in the fall, during my undergraduate freshman year. This was the first version that would be recognizable as what is now Neural MMO. It grew to around 2500 lines of python and included foraging mechanics with various resources, neural networks trained with genetic algorithms, and 3D visualization. However, it was still just a cool project without any guiding vision.

The spark of inspiration struck in July 2017 during the summer of my undergraduate sophomore year. This was the one true eureka moment that has guided my research to this day. It was the simple realization that even the most mundane of human tasks are often guided by a complex and interconnected web of long-term goals. Learning this sort of real-world structured reasoning *tabula rase* is **impossible** inside of an environment that does not require it. In other words, no amount of learning on Atari will produce generally intelligent agents. The likelihood of research on similar simple environments producing general intelligence is vanishingly small. Without a cognitively realistic environment, it is **impossible** to even determine what problems need to be solve. For example, are current learning methods capable of learning theory of mind? Determining this is **impossible** on Atari because Atari does not *require* theory of mind!

Massively Multiplayer Online games, or MMOs, are a genre first introduced in the 90s by games like Ultima Online and popularized in the 2000s by titles like Runescape and World of Warcraft that are still in active development today. MMOs simulate persistent worlds with thousands of players, dynamic economies, and long-term progression. They are designed with such efficiency that 90s hardware could simulate cities of players all interacting in real time. They were the perfect candidate for learning cognitively realistic artificial agents.

The one major logistical issue was that MMOs are not open-source, and even if they were, training agents on a full game would be a massive undertaking requiring a large team and computing resources. Instead, I looked to my previous work on multiagent simulation to construct an efficient MMO-inspired environment that could be used in academic-scale research. Using some of the original code from the previous year, I spent the second half of the summer of 2017 building a prototype that better fit this new vision. It included a more bare-bones 2D renderer but much better integration with neural agents and some early successes in solving basic foraging tasks.

I attended NeurIPS in the winter of 2017 to present unrelated language modeling research. There, I met Ilya Sutskever after his talk on reinforcement learning for Robosumo and DoTA 1v1. He introduced me to my future advisor, Phillip Isola, and I took the 2018 spring semester off from my undergraduate degree to build the first public version of Neural MMO at OpenAI. I ended up staying through the summer to continue building and released Neural MMO 1.0 in the spring of 2019, after taking some time to finish my undergraduate degree and build a reasonable visualizer in Three.js. I graduated a semester early and took the rest of the spring and summer to get a headstart on expanding Neural MMO, in anticipation of starting my Ph.D at MIT with Phillip in the fall. During this time, I released Neural MMO 1.1 and 1.2, which included important infrastructure updates and a new 3D client in Unity3D.

The first year of my PhD was devoted to a number of infrastructure problems, some of which ended up being red herrings and others of which ended up being premature optimization. What I will refer to as *the IO problem* was the major enduring contribution. This is the topic, in whole or in part, of several of my early manuscripts on Neural MMO. The problem deals with how to craft expressive environments while still keeping data in a format that is reasonably easy to ingest with a neural network. A lengthy description will be provided later. The material contributions from this year are the release of Neural MMO 1.3, an initial arXiv manuscript, and an extended abstract at AAMAS 2020.

The remainder of my first year and the summer were devoted to training capable agents in Neural MMO and devising visualizations to interpret their behaviors. I released Neural MMO 1.4 in July and presented this version specifically at an ICML 2020 workshop.

Neural MMO 1.5 was a much larger update that took nearly two years – from July 2020 until February 2021 to develop an initial version and until June 2022 for the final minor release. This update was so significant for a number of reasons. The first release included five major categories of improvements. First, it included engineering work focused on caching that was required to scale Neural MMO to 10x larger maps. This included a large set of non-trivial optimizations using the latest beta builds of Unity 3D to even render it. Second, it included a major expansion to Neural MMO itself consisting of scripted NPCs (Non-Player Characters) that would wander the map and fight with or flee from agents. These NPCs dropped armor when defeated that agents could equip to better protect themselves. Third, the update included a major breakthrough in the core representation of Neural MMO's data that made the whole simulation 100x faster. Fourth, it included new dashboards and statistics for tracking what agents learned. Fifth and finally, it included a clean training and evaluation demo for others to make use of the project.

Neural MMO 1.5.1 was the version used for the first public competition on Neural MMO, hosted in partnership with AICrowd in 2021. This was a smaller competition, and though we had over 200 signups, we received only a handful of capable submissions. The environment was far more complex than anything in use within academic research at the time – which was the point. However, making something of this complexity simple to use was no easy task, and this marked the start of a multi-year journey to turn Neural MMO into more broadly useful resource for researchers.

Versions 1.5.1-1.5.3 included several attempts at making Neural MMO easier to use. I swapped my bespoke dashboard for the recently released WandB, made several improvements to visualizations, and significantly improved the performance of the baselines. This version was published in NeurIPS Datasets and Benchmarks 2021, a new full-paper track that significantly reduced the friction of publishing environments. This version was also used for the IJCAI 2022 Competition, hosted in partnership with AICrowd and Parametrix.ai, who won the previous competition. This competition was a resounding success with over 500 participants and 1600 submissions. Of these, 50 unique participants beat the baseline, making it a contender for the most successful RL competition ever hosted.

As successful as the competition was, there were still a number of rough edges. The first competition used an RLlib baseline, which was flexible but painful to use. The second competition ported to Torchbeast, a less flexible but less painful library. In versions 1.5.4-1.5.5, I attempted to port the baseline to CleanRL, an incredibly simple to use library with no native support for multiagent environments or for many of the other complexities of Neural MMO. This required a number of complicated single-purpose wrappers, but I eventually obtained reasonable performance. Unfortunately, it was not good enough for the next competition, and the reason why would elude me for nearly another two years.

I released Neural MMO 1.6 in the fall of 2022. It was the largest update ever to the environment itself, introducing an item system with 16 unique items available in 10 different qualities, rebalanced combat, five new gathering professions and corresponding resources, weapons and tools, and a global exchange market. This version was used for the third and largest competition, hosted at NeurIPS 2022 in partnership with AICrowd and Parametrix.ai with additional compute sponsorship from AWS. We received over 1600 submissions once again from nearly 600 participants, with nearly 70 surpassing the baseline. For context, we received more high-quality submissions than the 2020 Procgen competition hosted by OpenAI, which had a dramatically more accessible problem, a larger prize pool, and much more media exposure.

The big Neural MMO 2.0 release in the summer of 2023 requires a lot of context. First, while the past two competitions had been partnered with Parametrix.ai and AICrowd, neither organization did significant work directly on the environment. Instead, partners took charge of competition elements such as the submission collection and evaluation process, starter kits, associated devops, and promotion. Additionally, Parametrix.ai developed a new web client capable of rendering replays of the environment. However, I was still personally responsible for all of the dozens of patches required for the competition builds. This changed after 1.6.

The NeurIPS 2023 competition was additionally partnered with Carper AI, an open source organization under the Stability AI umbrella. Their involvement helped recruit additional open-source contributors to the project, which greatly accelerated development after some initial growing pains. Neural MMO 2.0 was a near complete rewrite of the project, including 3x faster core infrastructure, all new visuals, and a task system capable of representing nearly any conceivable high-level goal inside of Neural MMO. We published Neural MMO 2.0 in NeurIPS Datasets and Benchmarks 2023. I would especially like to thank Kyoung Whan Choe for his persistent commitment and contribution to the project, David Bloomin for his work on the core infrastructure, and Louis Castricato for his continued support.

The focus of this fourth and final competition was on learning task-conditional policies that could accomplish a wide range of goals. I finally solved the technical problems preventing CleanRL from working with Neural MMO, so this competition had by far the simplest and easiest to use baseline of all of the competitions. Several non-developers also contributed to the project through Carper AI. Our documentation was redone and professionally edited in consultation with Rose S. Shuman, who has extensive experience writing for the United Nations. Lucas de Alcântara, an artist at Blizzard, created all new graphics for the 2.0 web client. This competition was also the fairest, introducing compute-limited tracks requiring only a single GPU to participate effectively.

Despite all of this preparation, the competition was much smaller, receiving only 200 submissions from less than 300 participants. This was still a large competition relative to most of the others at NeurIPS, but much smaller than we had hoped. Unfortunately, Stability pulled 90% of our compute funding after promising 150k A100 hours, and understandably, Parametrix.ai cut most of the advertising and development budget for the project shortly thereafter. To be clear, I am not blameless in this either, and I made a few key technical and management mistakes that could have bought us another 2 months to recover from this.

The good news is that the submissions we did receive were of high quality, and the winning submission completed 4x more tasks than the baseline. This model and the training code have been made publicly available on the Neural MMO Github. As a result, the latest version of Neural MMO is simultaneously the most cognitively interesting, computationally efficient, user friendly, and has the highest quality baselines.

1.2 Key Problems Solved by Neural MMO

The goal of Neural MMO has been fixed from the project's inception: create an efficient and cognitively interesting environment for AI research. However, many of the problems I expected to encounter along the way ended up being easy, and many of the problems that I thought would be easy were hard. The most substantial problems addressed throughout my PhD were:

1. **The IO Problem:** Finding a rich and efficient data representation for observations and actions

2. **Environment Representation:** Designing a cognitively interesting environment with data stored in a manner that allows observations to be computed efficiently.

3. **Visualization:** Methods for understanding what agents have learned

4. **Learning:** Essential methods and tricks for learning in an environment of this complexity that are not normally needed in the rest of RL.

5. **Goal Conditionality:** Expressive representations for complex, high-level goals.

Other problems: There are several areas where I spent substantial time that ended up being red herrings. A few of these were:

1. Distributing a single environment across multiple cores like a real game server. RL was not and still is not ready to handle environments of this scale. Plus, there were substantial enough engineering challenges that, most of the time, you are better off running one environment per core and letting them be slow. That said, I expect this topic to become relevant again in a few years.

2. Tools for defining flexible observation and action spaces. I wanted to make exposing specific types of data from the environment easy and efficient. There really is not a good way to do this in general, and my various attempts ended up adding more complexity than they were worth.

3. A detailed experiment dashboard. Before WandB became a well-developed service, I built a tool for Neural MMO experiments that enabled users to add several different types of plots to a local dashboard. This revealed several insights about the environment and agent policies that allowed be to build the baselines for Neural MMO 1.3-1.4. However, maintaining this tool was ultimately more trouble than it was worth, and once WandB became a good alternative, I ported logging to it instead.

4. Client-side optimizations. I spent a good deal of time figuring out how to procedurally assemble maps out of non-cubic 3D tiles. I was hoping to develop general methods that would allow certain useful classes of research environments to be rendered more easily. I produced neat visuals that scaled well, but other tools with lower fidelity ended up being more useful.

Chapter 2

Neural MMO 1.0

This chapter covers the early development and findings of Neural MMO. While this technically predates the start of my Ph.D in the fall of 2019, it is essential to include for two reasons. First, it provides important context that grounds my subsequent work. Second, it is also advised by Phillip and contains work that I conducted after being accepted to MIT for the express purpose of starting on my doctoral work.

Many of the key ideas expanded on by my more recent works were motivated by the findings made early in development. For example, I found that multiagent learning dynamics could result in better learning than in a corresponding single-agent environment. This was a significant motivation for continuing to scale the environment to more agents. I also found that complex environments tend to generate structured data that makes training difficult. This motivated my subsequent work on the "IO problem." The visualization tools in this work were also primitive but useful and motivated further work in this direction.

The following paper titled "Neural MMO: A Massively Multiagent Game Environment for Training and Evaluating Intelligent Agents" was published on arXiv in March of 2019, concurrent with an OpenAI blog post on the same topic. It is coauthored with Yilun Du and advised by Igor Mordatch and Phillip Isola.

The emergence of complex life on Earth is often attributed to the arms race that ensued from a huge number of organisms all competing for finite resources. We present an artificial intelligence research environment, inspired by the human game genre of MMORPGs (Massively Multiplayer Online Role-Playing Games, a.k.a. MMOs), that aims to simulate this setting in microcosm. As with MMORPGs and the real world alike, our environment is persistent and supports a large and variable number of agents. Our environment is well suited to the study of large-scale multiagent interaction: it requires that agents learn robust combat and navigation policies in the presence of large populations attempting to do the same. Baseline experiments reveal that population size magnifies and incentivizes the development of skillful behaviors and results in agents that outcompete agents trained in smaller populations. We further show that the policies of agents with unshared weights naturally diverge to fill different niches in order to avoid competition.

2.1 Introduction

Life on Earth can be viewed as a massive multiagent competition. The cheetah evolves an aerodynamic profile in order to catch the gazelle, the gazelle develops springy legs to run even faster: species have evolved ever new capabilities in order to outcompete their adversaries. The success of biological evolution has inspired many attempts at creating “artificial life" in silico.

In recent years, the field of deep reinforcement learning (RL) has embraced a related approach: train agents by having them compete in simulated games [1]–[3]. Such games are immediately interpretable and provide easy metrics derived from the game’s “score" and win conditions. However, popular game benchmarks typically define a narrow, episodic task with a small fixed number of players. In contrast, life on Earth involves a persistent environment, an unbounded number of players, and a seeming “open-endedness", where ever new and more complex species emerge over time, with no end in sight [4].

Our aim is to develop a simulation platform (see Figure 3.1) that captures important properties of life on Earth, while also borrowing from the interpretability and abstractions of human-designed games. To this end, we turn to the game genre of Massively Multiplayer Online Role-Playing Games (MMORPGs, or MMOs for short). These games involve a large, variable number of players competing to survive and prosper in persistent and far-flung environments. Our platform simulates a “Neural MMO" – an MMO in which each agent is a neural net that learns to survive using RL.

We demonstrate the capabilities of this platform through a series of experiments that investigate emergent complexity as a function of the number of agents and species that compete in the simulation. We find that large populations act as competitive pressure that encourages exploration of the environment and the development of skillful behavior. In addition, we find that when agents are organized into species (share policy parameters), each species naturally diverges from the others to occupy its own behavioral niche. Upon publication, we will opensource the platform in full.

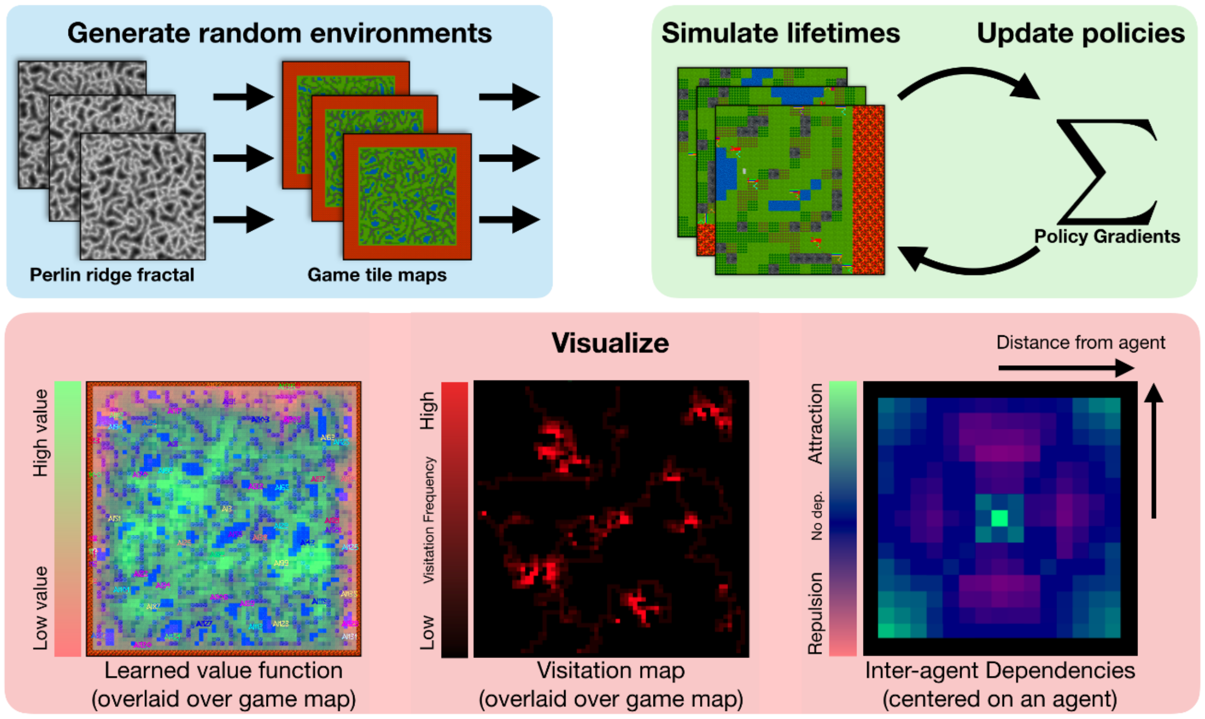


Figure 2.1: Our Neural MMO platform provides a procedural environment generator and visualization tools for value functions, map tile visitation distribution, and agent-agent dependencies of learned policies. Baselines are trained with policy gradients over 100 worlds.

2.2 Background and Related Work

Artificial Life and Multiagent Reinforcement Learning Research in “Artificial life" aims to model evolution and natural selection in biological life; [5], [6]. Such projects often consider open-ended skill learning [7] and general morphology evolution [8] as primary objectives. Similar problems have recently resurfaced within multiagent reinforcement learning where the continual co-adaptation of agents can introduce additional nonstationarity that is not present in single agent environments. While there have been multiple attempts to formalize the surrounding theory [9], [10], we primarily consider environment-driven works. These typically consider either complex tasks with 2-10 agents [2], [3], [11] or much simpler environments with tens to upwards of a million agents [3], [11]–[16]. Most such works further focus on learning a specific dynamic, such as predator-prey [17] or are more concerned with the study than the learning of behavior, and use hard-coded rewards [16]. In contrast, our work focuses on large agent populations in complex environments.

Game Platforms for Intelligent Agents The Arcade Learning Environment (ALE) [18] and Gym Retro [19] provide 1000+ limited scope arcade games most often used to test individual research ideas or generality across many games. Better performance at a large random subset of games is a reasonable metric of quality. However, recent results have brought into question the overall complexity each individual environment [20], and strong

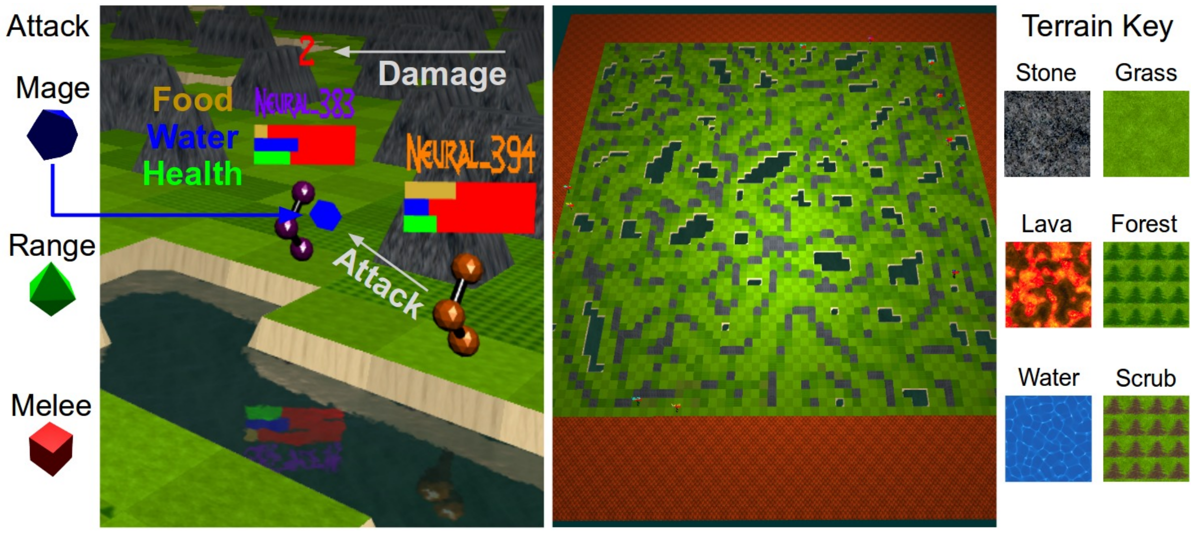


Figure 2.2: Our platform includes an animated 3D client and a toolbox used to produce the visuals in this work. Agents compete for food and water while engaging in strategic combat. See the Neural MMO section for a brief overview and the Supplement for full details.

performance in such tasks is not particularly difficult for humans.

More recent work has demonstrated success on multiplayer games including Go [1], the Multiplayer Online Battle Arena (MOBA) game DOTA2 [2], and Quake 3 Capture the Flag [3]. Each of these projects has advanced our understanding of a class of algorithms. However, these games are limited to 2-12 players, are episodic, with game rounds on the order of an hour, lack persistence, and lack the game mechanics supporting large persistent populations – there is still a large gap in environment complexity compared to the real world.

Role-playing games (RPGs) such as Pokemon and Final Fantasy, are in-depth experiences designed to engage human players for hundreds of hours of persistent gameplay. Like the real world, problems in RPGs have many valid solutions and choices have long term consequences.

MMORPGs are the (massively) multiplayer analogs to RPGs. They are typically run across several persistent servers, each of which contains a copy of the environment and supports hundreds to millions of concurrent players. Good MMOs require increasingly clever, team-driven usage of the game systems: players attain the complex skills and knowledge required for the hardest challenges only through a curriculum of content spanning hundreds of hours of gameplay. Such a curriculum is present in many game genres, but only MMOs contextualize it within persistent social and economic structures approaching the scale of the real world.

2.3 Neural MMO

We present a persistent and massively multiagent environment that defines foraging and combat systems over procedurally generated maps. The Supplement provides full environment details and Figure 2.2 shows a snapshot. The core features are support for a large and variable number of agents, procedural generation of tile-based terrain, a food and water foraging system, a strategic combat system, and inbuilt visualization tools for analyzing learned policies

Agents (players) may join any of several servers (environment instances). Each server contains an automatically generated tile-based environment of configurable size. Some tiles, such as food-bearing forest tiles and grass tiles, are traversable. Others, such as water and solid stone, are not. Upon joining a server, agents spawn at a random location along the edges of the environment. In order remain healthy (maintain their health statistic), agents must obtain food and water – they die upon reaching reaching 0 health. At each **server tick (time step)**, agents may move one tile and make an attack. Stepping on a forest tile or next to a water tile refills a portion of the agent's food or water supply, respectively. However, forest tiles have a limited supply of food; once exhausted, food has a 2.5 percent chance to regenerate each tick. This means that agents must compete for food tiles while periodically refilling their water supply from infinite water tiles. They may attack each other using any of three attack options, each with different damage values and tradeoffs. Precise foraging and combat mechanics are detailed in the Supplement.

Agents observe local game state and decide on an action each game tick. The environment does not make any further assumptions on the source of that decision, be it a neural network or a hardcoded algorithm. We have tested the environment with up to 100 million agent trajectories (lifetimes) on 100 cores in 1 week. Real and virtual worlds alike are open-ended tasks where complexity arises with little direction. Our environment is designed as such. Instead of rewarding agents for achieving particular objectives optimize only for survival time: they receive reward $r_t = 1$ for each time step alive. Competition for finite resources mandates that agents must learn intelligent strategies for gathering food and water in order to survive.

One purpose of the platform is to discover *game mechanics* that support complex behavior and *agent populations* that can learn to make use of them. In human MMOs, *developers aim to create balanced mechanics while players aim to maximize their skill in utilizing them.* The initial configurations of our systems are the results of several iterations of balancing, but are by no means fixed: every numeric parameter presented is editable within a simple configuration file.

2.4 Architecture and Training

Agents are controlled by policies parameterized by neural networks. Agents make observations o_t of the game state s_t and follow a policy $\pi(o_t) \rightarrow a_t$ in order to make actions a_t. We maximize a return function R over trajectory $\tau = (o_t, a_t, r_t, ..., o_T, a_T, r_T)$. This is a discounted sum of survival rewards: $R(\tau) = \sum_t^T \gamma^t r_t$ where $\gamma = 0.99$, T is the time at death and the survival reward r_t equals 1, as motivated previously. The policy π may be different

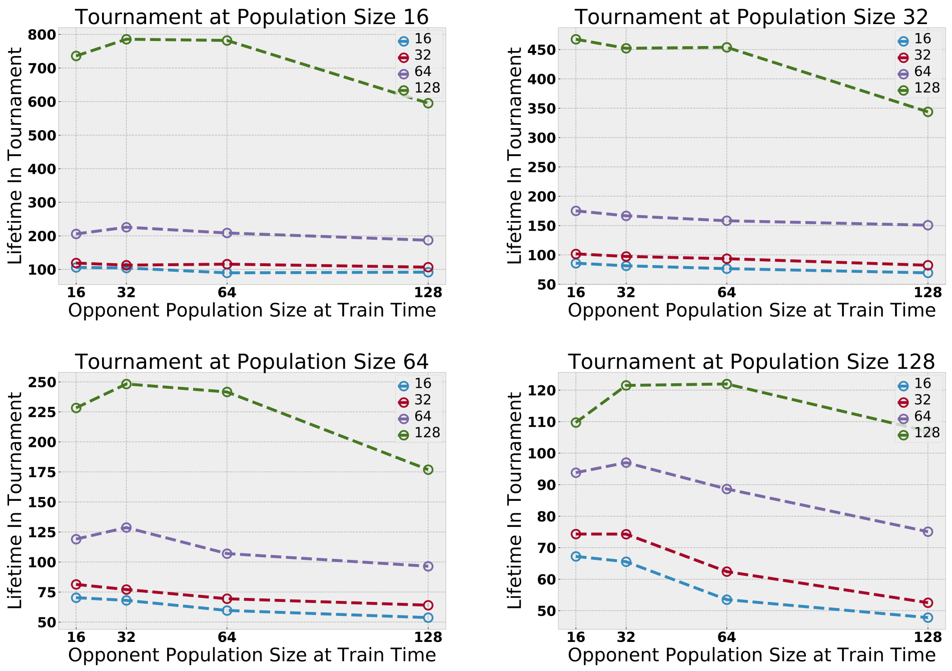


Figure 2.3: Maximum population size at train time varies in (16, 32, 64, 128). At test time, we merge the populations learned in pairs of experiments and evaluate lifetimes at a fixed population size. Agents trained in larger populations always perform better.

for each agent or shared. Algorithm 1 shows high level training logic. The Supplement details the tile-based game state s_t and hyperparameters (Table 1).

Input We set the observation state o_t equal to the crop of tiles within a fixed L_1 distance of the current agent. This includes tile terrain types and the select properties (such as health, food, water, and position) of occupying agents. Our choice of o_t is an equivalent representation of what a human sees on the screen, but our environment supports other choices as well. Note that computing observations does not require rendering.

Output Agents output action choices a_t for the next time step (game tick). Actions consist of one movement and one attack. Movement options are: North, South, East, West, and Pass (no movement). Attack options are labeled: Melee, Range, and Mage, with each attack option applying a specific preset amount of damage at a preset effective distance. The environment will attempt to execute both actions. Invalid actions, (*e.g.* moving into stone), are ignored.

Our policy architecture preprocesses the local environment by embedding it and flattening it into a single fixed length vector. We then apply a linear layer followed by linear output heads for movement and attack decisions. New types of action choices can be included by adding additional heads. We also train a value function to estimate the discounted return. As agents receive only a stream of reward 1, this is equal to a discounted estimate of the agent's time until death. We use a value function baselines policy gradient loss and optimize with Adam. It was possible to obtain good performance without discounting, but training was less stable. We provide full details in the supplements.

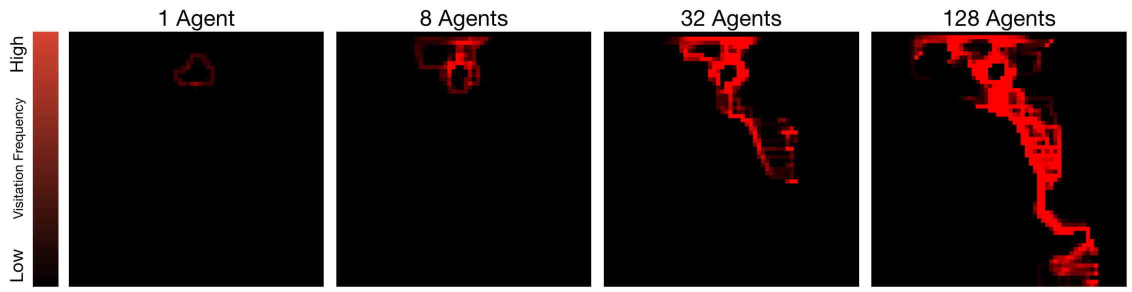


Figure 2.4: Population size magnifies exploration: agents spread out to avoid competition.

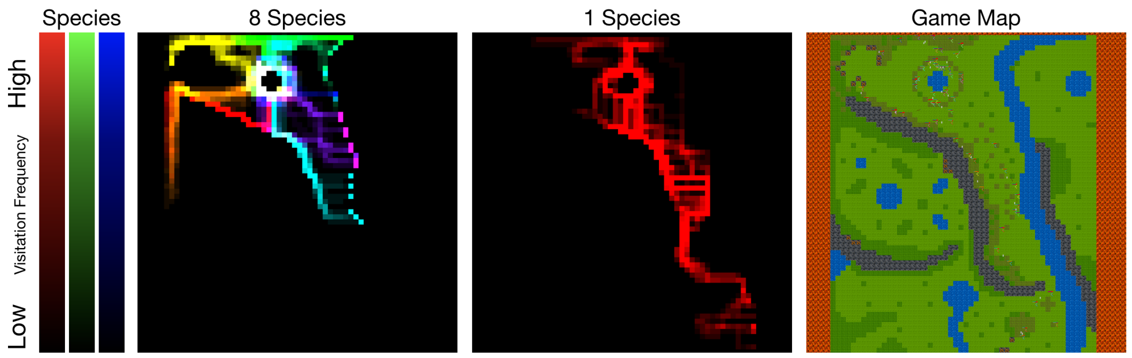


Figure 2.5: Populations count (number of species) magnifies niche formation. Visitation maps are overlaid over the game map; different colors correspond to different species. Training a single population tends to produce a single deep exploration path. Training eight populations results in many shallower paths: populations spread out to avoid competition among species.

Algorithm 1 Neural MMO logic for one game tick. See Experiments (Technical details) for spawning logic. The algorithm below makes two omissions for simplicity. First, we use multiple policies and sample a policy $\pi \sim \pi_1, \ldots, \pi_N$ from the set of all policies when spawning a new agent. Second, instead of performing a policy gradient update every game tick, we maintain experience buffers from each environment and perform an update once all buffers are full.

```
for each environment server do
  if number of agents alive < spawn cap then
    spawn an agent
  end if
  for each agent do
    i ← population index of the agent
    Make observation $o_t$, decide action $\pi_i(o_t) \rightarrow a_t$
    Environment processes $a_t$, computes $r_t$, and updates agent health, food, etc.
    if agent is dead then
      remove agent
    end if
  end for
  Update environment state $s_{t+1} \rightarrow f(s_t, a_t)$
end for
Perform a policy gradient update on policies $\pi \sim \pi_1, \ldots, \pi_N$ using $o_t$, $a_t$, $r_t$ from all agents
across all environment servers =0
```

2.5 Experiments

We present an initial series of experiments using our platform to explore multiagent interactions in large populations. We find that agent competence scales with population size. In particular, increasing the maximum number of concurrent players (N_{ent}) magnifies exploration and increasing the maximum number of populations with unshared weights (N_{pop}) magnifies niche formation. Agents policies are sampled uniformly from a number of "populations" $\pi \sim \pi_1, \ldots, \pi_N$. Agents in different populations have the same architecture but do not share weights.

Technical details We run each experiment using 100 worlds. We define a constant C over the set of worlds W. For each world $w \in W$, we uniformly sample a $c \in (1, 2, ...C)$. We define "spawn cap" such that if world w has a spawn cap c, the number of agents in w cannot exceed c. In each world w, one agent is spawned per game tick provided that doing so would exceed the spawn cap c of w. To match standard MMOs, we would fix $N_{ent} = N_{pop}$ (humans are independent networks with unshared weights). However, this incurs sample complexity proportional to number of populations. We therefore share parameters across groups of up to 16 agents for efficiency.

2.5.1 Server Merge Tournaments

We perform four experiments to evaluate the effects on foraging performance of training with larger populations and with a greater number of populations. For each experiment, we fix $N_{pop} \in (1, 2, 4, 8)$ and a spawn cap (the maximum number of concurrent agents) $c = 16 \times N_{pop}$, such that $c \in (16, 32, 64, 128)$. We train for a fixed number of trajectories per population.

Evaluating the influence of these variables is nontrivial. The task difficulty is highly dependent on the size and competence of populations in the environment: mean agent lifetime is not comparable across experiments. Furthermore, there is no standard procedure among MMOs for evaluating relative player competence across multiple servers. However, MMO servers sometimes undergo merges whereby the player bases from multiple servers are placed within a single server. As such, we propose tournament style evaluation in order to directly compare policies learned in different experiment settings. Tournaments are formed by simply concatenating the player bases of each experiment. Figure 2.3 shows results: we vary the maximum number of agents at test time and find that agents trained in larger settings consistently outperform agents trained in smaller settings.

We observe more interesting policies once we introduce the combat module as an additional learnable mode of variation on top of foraging. With combat, agent actions become strongly coupled with the states of other agents. As a sanity check, we also confirm that all of the populations trained with combat handily outperform all of the populations trained with only foraging, when these populations compete in a tournament with combat enabled.

To better understand theses results, we decouple our analysis into two modes of variability: maximum number of concurrent players (N_{ent}) and maximum number of populations with unshared weights (N_{pop}). This allows us to examine the effects of each factor independently. In order to isolate the effects of environment randomization, which also encourages exploration, we perform these experiments on a fixed map. Isolating the effects of these variables produces more immediately obvious results, discussed in the following two subsections:

2.5.2 N_{ent}: Multiagent Magnifies Exploration

In the natural world, competition between animals can incentivize them to spread out in order to avoid conflict. We observe that overall exploration (map coverage) increases as the number of concurrent agents increases (see Figure 2.4; the map used is shown in Figure 2.5). Agents learn to explore only because the presence of other agents provides a natural incentive for doing so.

2.5.3 N_{pop}: Multiagent Magnifies Niche Formation

We find that, given a sufficiently large and resource-rich environment, different populations of agents tend to separate to avoid competing with other populations. Both MMOs and the real world often reward masters of a single craft more than jacks of all trades. From Figure 2.5, specialization to particular regions of the map increases as number of populations increases. This suggests that the presence of other populations force agents to discover a single advantageous skill or trick. That is, increasing the number of populations results in diversification to separable regions of the map. As entities cannot out-compete other agents of their own population (i.e. agent's with whom they share weights), they tend to seek areas of the map that contain enough resources to sustain their population.

2.5.4 Environment Randomized Exploration

The trend of increasing exploration with increasing entity number is clear when training on a single map as seen in Figure 2.4, 2.5, but it is more subtle with environment randomization. From Figure 2.6, all population sizes explore adequately. It is likely that "exploration" as defined by map coverage is not as difficult a problem, in our environment, as developing robust policies. As demonstrated by the Tournament experiments, smaller populations learn brittle policies that do not generalize to scenarios with more competitive pressure–even against a similar number of agents.

2.5.5 Agent-Agent Dependencies

We visualize agent-agent dependencies in Figure 2.7. We fix an agent at the center of a hypothetical map crop. For each position visible to that agent, we show what the value function would be if there were a second agent at that position. We find that agents learn policies dependent on those of other agents, in both the foraging and combat environments.

2.6 Discussion

2.6.1 Multiagent competition is a curriculum magnifier

Not all games are created equal. Some produce more complex and engaging play than others. It is unreasonable to expect pure multiagent competition to produce diverse and interesting behavior if the environment does not support it. This is because **multiagent competition is a curriculum *magnifier*, not a curriculum in and of itself**. The initial conditions for formation of intelligent life are of paramount importance. Jungle climates produce more biodiversity than deserts. Deserts produce more biodiversity than the tallest mountain peaks. To current knowledge, Earth is the only planet to produce life at all. The same holds true in simulation: human MMOs mirror this phenomenon. Those most successful garner large and dedicated player bases and develop into complex ecosystems. The multiagent setting is interesting because learning is responsive to the competitive and collaborative pressures of other learning agents–but the environment must support and facilitate such pressures in order for multiagent interaction to drive complexity.

There is room for debate as to the theoretical simplest possible seed environment required to produce complexity on par with that of the real world. However, this is not our objective. We have chosen to model our environment after MMOs, even though they may be more complicated than the minimum required environment class, because they are known to support the types of interactions we are interested in while maintaining engineering and implementation feasibility. This is not true of any other class environments we are aware of: exact physical simulations are computationally infeasible, and previously studied genres of human games lack crucial elements of complexity (see Background). While some may see our efforts as cherrypicking environment design, we believe this is precisely the objective: the primary goal of game development is to create complex and engaging play at the level of human intelligence. The player base then uses these design decisions to create strategies far beyond the imagination of the developers.

2.6.2 Additional Insights

We briefly detail several miscellaneous points of interest in Figure 2.8. First, we visualize learned attack patterns of agents. Each time an agent attacks, we splat the attack type to the screen. There are a few valid strategies as per the environment. Melee is intentionally overpowered, as a sanity check. This cautions agents to keep their distance, as the first to strike wins. We find that this behavior is learned from observation of the policies learned in Figure 2.8.

Second, a note on tournaments. We equate number of trajectories trained upon as a fairest possible metric of training progress. We experimented with normalizing batch size but found that larger batch size always leads to more stable performance. Batch size is held constant, but experience is split among species. This means that experiments with more species have smaller effective batch size: larger populations outperform smaller populations even though the latter are easier to train.

Finally, a quick note on niche formation. Obtaining clean visuals is dependent on having an environment where interaction with other agents is unfavorable. While we ensure this is the case for our exploration metrics, niche formation may also occur elsewhere, such as in the space of effective combat policies. For this reason, we expect our environment to be well suited to methods that encourage sample diversity such as population-based training [21].

2.7 Future Work

Our final set of experiments prescribes targeting to the agent with lowest health. Learned targeting was not required to produce compelling policies: agents instead learn effective attack style selection, strafing and engaging opportunistically at the edge of their attack radius. Another possible experiment is to jointly learn attack style selection and targeting. This would require an attentional mechanism to handle the variable number of visible targets. We performed only preliminary experiments with such an architecture, but we still mention them here because even noisy learned targeting policies significantly alter agent-agent dependence maps. As shown in Figure 2.7, the small square shaped regions of high value at the center of the dependency maps correspond to the ranges of different attack styles. These appear responsive to the current combat policies of other learning agents. We believe that the learned targeting setting is likely to useful for investigating the effects of concurrent learning in large populations.

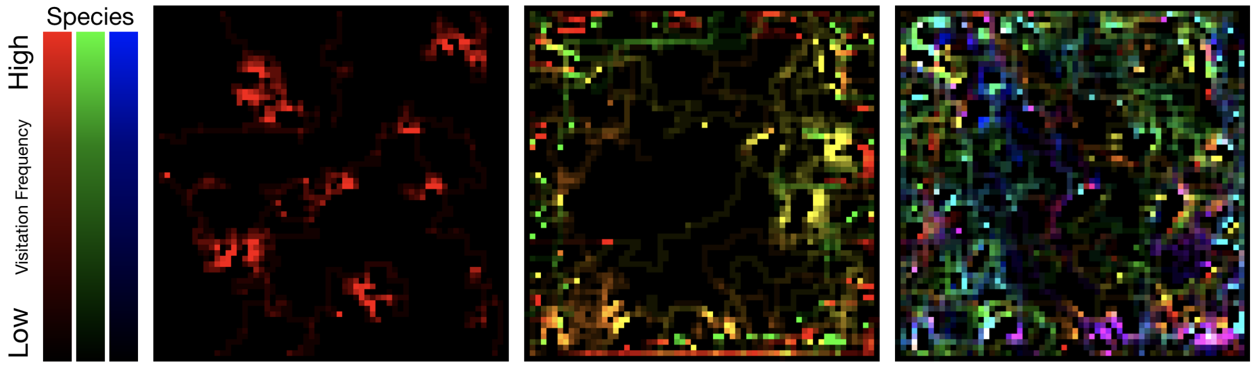


Figure 2.6: Exploration maps in the environment randomized settings. From left to right: population size 8, 32, 128. All populations explore well, but larger populations with more species develop robust and efficient policies that do better in tournaments.

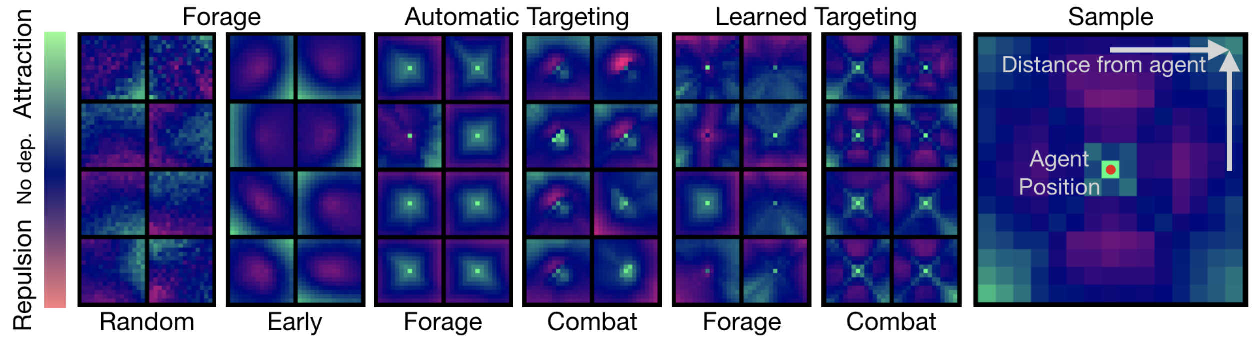


Figure 2.7: Agents learn to depend on other agents. Each square map shows the response of an agent of a particular species, located at the square's center, to the presence of agents at any tile around it. Random: dependence map of random policies. Early: "bulls eye" avoidance maps learned after only a few minutes of training. Additional maps correspond to foraging and combat policies learned with automatic targeting (as in tournament results) and learned targeting (experimental, discussed in Additional Insights). In the learned targeting setting, agents begin to fixate on the presence of other agents within combat range, as denoted by the central square patterns.

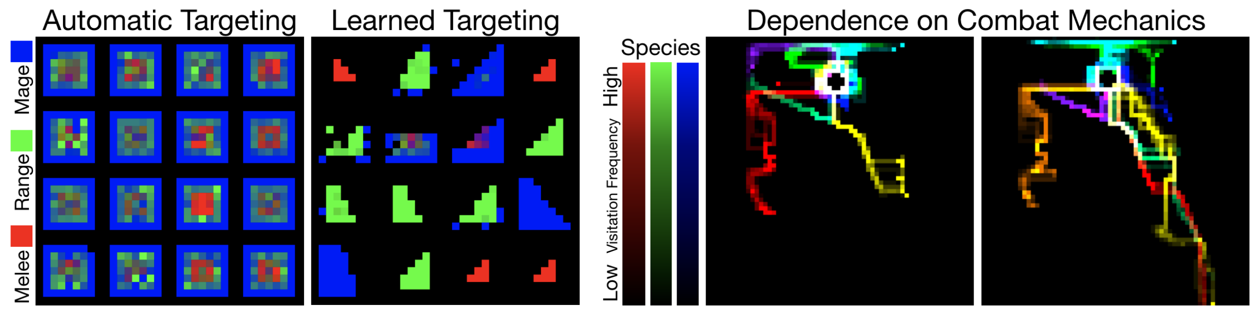


Figure 2.8: Attack maps and niche formation quirks. Left: combat maps from automatic and learned targeting. The left two columns in each figure are random. Agents with automatic targeting learn to make effective use of melee combat (denoted by higher red density). Right: noisy niche formation maps learned in different combat settings with mixed incentives to engage in combat.

2.8 Conclusion

We have presented a neural MMO as a research platform for multiagent learning. Our environment supports a large number of concurrent agents, inbuilt map randomization, and detailed foraging and combat systems. The included baseline experiments demonstrate our platform's capacity for research purposes. We find that population size magnifies exploration in our setting, and the number of distinct species magnifies niche formation. It is our hope that our environment provides an effective venue for multiagent experiments, including studies of niche formation, emergent cooperation, and coevolution. The entire platform will be open sourced, including a performant 3D client and research visualization toolbox. Full technical details of the platform are available in the Supplement.

Chapter 3

Neural MMO 1.3

Chapter 3 covers work from the first year of my Ph.D. Neural MMO was already a crazy project with OpenAI resources. The early experiments relied on access to hundreds of CPU cores on a well-administered cluster. It was a much crazier project to attempt at MIT without those resources. At the time, not a single common RL library or implementation supported Neural MMO or anything remotely close to its complexity. Virtually every portion of the software had to be built from scratch. My early work focused on making learning work *at all* and visualizing whether anything meaningful had been learned.

I published two short-form conference papers on these topics and one long-form arXiv manuscript. An extended abstract titled "Neural MMO v1.3: A Massively Multiagent Game Environment for Training and Evaluating Neural Networks" is published at AAMAS 2020. It is coauthored with Yilun Du and advised by Phillip Isola and Igor Mordatch. "Neural MMO: Ingredients for Massively Multiagent Artificial Intelligence Research" was published at the ICML 2020 workshop on Learning in Artificial Open Worlds. It is advised by Phillip Isola. The longer manuscript titled "Neural MMO v1.3: A Massively Multiagent Game Environment for Training and Evaluating Neural Networks" was published on arXiv in January of 2020. It is coauthored with Yilun Du and advised by Phillip Isola and Igor Mordatch. The longer paper largely duplicates and expands upon the material from the shorter papers, so I omit the shorter papers from this chapter. Also, note that as there is no full-length conference paper on Neural MMO until the introduction of the NeurIPS Datasets and Benchmarks track in 2021. As such, some material from 1.0 is repeated with additional refinement.

The section on "The IO Problem" is particularly crucial to the overall development of Neural MMO and of complex simulators in general. The appendix for this section includes an alternative version of this section that mathematically inclined readers may find more palatable.

Progress in multiagent intelligence research is fundamentally limited by the number and quality of environments available for study. In recent years, simulated games have become a dominant research platform within reinforcement learning, in part due to their accessibility and interpretability. Previous works have targeted and demonstrated success 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 Neural MMO, a massively multiagent game environment inspired by MMOs and discuss our progress on two more general challenges in multiagent systems engineering for AI research: distributed infrastructure and game IO. We further demonstrate that standard policy gradient methods and simple baseline models can learn interesting emergent exploration and specialization behaviors in this setting.

3.1 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]–[3], [22], [23]. MMOs are possibly the most complex class of games in the entertainment industry and are a natural a next step in this progression. They simulate self-contained macrocosms with large, variable numbers of players, user-driven economies, team-oriented strategizing, and realistic long-term planning over hundreds to thousands of hours of persistent gameplay. We argue that, among all game genres in the entertainment industry, MMOs produce a style of in-game learning that comes closest to learning in the real world.

Our eventual goal is capable artificial intelligence within a full MMO through continuous environment development and iterative research upon it. Neural MMO seeks to capture the most important properties of the base game genre in a more simplified setting. In the process of working towards this objective, we encountered two major research engineering problems surrounding infrastructure and IO. This work shares our solutions to these problems in the context of Neural MMO as general methods that enable scalable multiagent learning in complex environments. The key contributions of this publication 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][1]

3. Stand-alone scalable infrastructure and performance logging for massively multiagent environments

[1]We have linked a one minute video of trained policies. Full address in case of hyperlink errors: https://youtu.be/DkHopV1RSxw

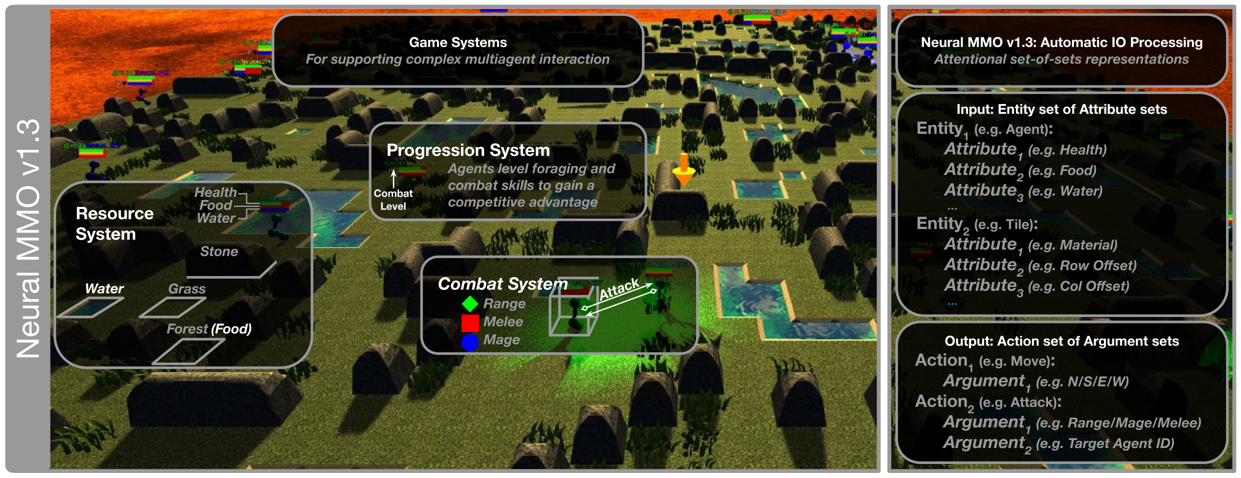

Neural MMO v1.3
Game Systems
For supporting complex multiagent interaction
Progression System
Combat Level
Agents level foraging and combat skills to gain a competitive advantage
Resource System
Health
Food
Water
Stone
Water
Grass
Forest (Food)
Combat System
Range
Melee
Mage
Attack
Neural MMO v1.3: Automatic IO Processing
Attentional set-of-sets representations
Input: Entity set of Attribute sets
Entity$_1$ (e.g. Agent):
Attribute$_1$ (e.g. Health)
Attribute$_2$ (e.g. Food)
Attribute$_3$ (e.g. Water)
Entity$_2$ (e.g. Tile):
Attribute$_1$ (e.g. Material)
Attribute$_2$ (e.g. Row Offset)
Attribute$_3$ (e.g. Col Offset)
Output: Action set of Argument sets
Action$_1$ (e.g. Move):
Argument$_1$ (e.g. N/S/E/W)
Action$_2$ (e.g. Attack):
Argument$_1$ (e.g. Range/Mage/Melee)
Argument$_2$ (e.g. Target Agent ID)

Figure 3.1: 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 high quality 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.

4. Stand-alone methods for interfacing with complex observation and action spaces in multiagent environments

Infrastructure: Modern deep RL frameworks place assumptions on the environment that are untrue or computationally inefficient in large multiagent systems. This situation has forced us to reexamine standard RL infrastructure according to computation placement and communication patterns. We formalize our findings in Ascend, a lightweight wrapper on top of Ray [24]. Ascend provides general abstractions and design patterns for multiagent systems that allow us to implement the full Neural MMO training backend in only a few lines of code.

IO Spaces: 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: these IO spaces cannot interface with standard architectures that expect fixed length tensors.[2] Our solution to this problem parameterizes the observation space as a set of entities (in turn parameterized by a set of attributes) and automatically generates attentional networks to select variable length action arguments by keying against learned entity embeddings.

3.2 Related Work

While deep reinforcement learning (RL) has recently expanded to include a variety of control focused tasks, games have always been an important research platform [25]. The Arcade

[2]Rendering does not solve the issue. See The IO Problem, below

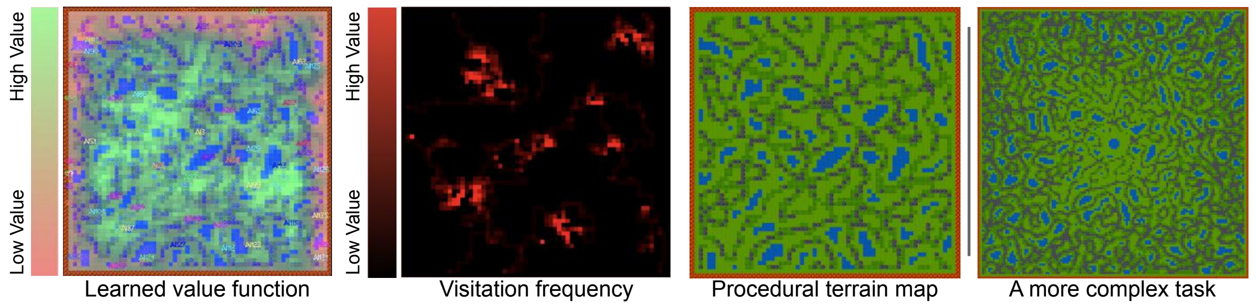


Figure 3.2: Visualizations render the learned value function and agent exploration patterns directly within the game client. In our experiments, we use the map on the left. Procedural generation enables a variety of task complexities: the rightmost map is 2x larger and also manipulates resource distributions to scale difficulty as agents move farther from the center.

Learning Environment (ALE) [18] and Gym Retro [19] provide 1000+ limited scope arcade games most often used to test individual research ideas or generality across many games. More recent work has demonstrated success on multiplayer games including the board game Go [1], the card game Heads-Up No-Limit Poker [26], the Multiplayer Online Battle Arena (MOBA) game DOTA2 [2], the Real Time Strategy game Starcraft 2 [23], first person Hide and Seek [22], and Quake 3 Capture the Flag [3]. These tasks are difficult and important but are limited to 2-10 players, are episodic with game rounds less than an hour, lack persistence, and lack game mechanics supporting large populations. Our work seeks to expand game AI to MMOs, which do possess these properties.

“Artificial life" (ALife) aims to model evolution and natural selection in biological life; [5], [6]. Such projects often consider open-ended skill learning [7] and general morphology evolution [8] as primary objectives. Some environments in this space simulate tens to upwards of a million agents [3], [11]–[16]. Most such works further focus on learning a specific dynamic such as predator-prey [17] or use hard-coded rewards because they are more concerned with studying emergent behavior than learning it from scratch [16]. Our work is based in games rather than biological life. While the real world is substantially more complex, MMOs support similar persistent social dynamics in large agent populations. Unlike the real world, they are efficient to simulate and straightforward to develop; numerous tools and best practices for MMO creators already exist courtesy of the entertainment industry.

3.3 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 (see Figure 3.1).

Environment Representation: Neural MMO is laid out on 2D tile map that is procedurally generated by thresholding a Perlin ridge fractal. For all maps, agents are added (*spawn*) within a designated region at a rate of one per timestep (server *tick*) up to a cap of 128. Agents may move about the grass and forest tiles of the game map, but stone and

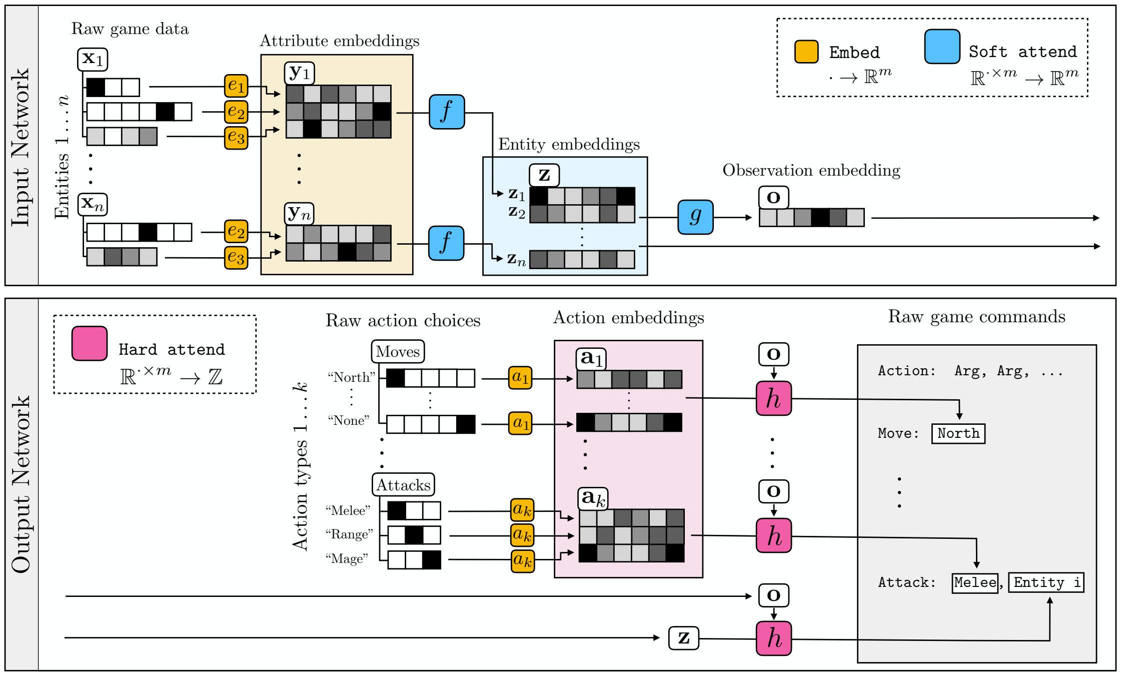


Figure 3.3: The environment provides local game state as a two-layer observation hierarchy over all nearby entities (tiles and other agents) and their associated attributes. The *input* module applies attention over attributes to produce entity embeddings. A second attentional mechanism is applied over entities to produce a flat observation embedding. The *outputs* module keys the intermediate entity embeddings against the network hidden state to select action-argument pairs with hard attention.

water are impassible. The map is also surrounded by a lethal lake of lava. Figure 3.1 shows terrain types, and Figures 3.1 and 3.2 (left) show the latest version of the game map for which we have successfully trained policies.[3] The map generation code is configurable to enable a variety of different studies within our environment. For example, Figure 3.2 (right) shows a larger map in which generation has been tweaked to produce easily navigable but resource poor terrain (see Resource System) near the center and mazelike but resource rich farther out.[4]

Resource System: Agents spawn with 10 food, water, and health. At every tick, agents lose one food and one water. All of these values are configurable. If agents run out of food or water, they begin losing health. If agents are well fed and well hydrated (i.e. above half food and water), 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.[5] Thus, the objective is simultaneously navigate and forage for food/water in the presence of upward of a hundred agents attempting to do the same.

Combat System: Agents can attack each other with three different styles. For flavor,

[3]On this map, agents forage inwards from the borders.

[4]On this map, agents forage outwards from the center.

[5]Agents collect food by moving into a "forest" tile and water by moving adjacent to a "water" tile. Forest tiles turn to grass once consumed, but they regenerate over time. Figure 3.1 shows tile types

we refer to these as Range, Mage, and Melee. Accuracy and damage are determined by the attack style and the combat stats of the attacker and defender (see Progression System). This system, which also allows agents to pilfer resources from their target, was designed to enable a variety of strategies. For example, Range and Melee attacks are more damaging, but a successful magic attack freezes the target in place temporarily. 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, using mage attacks to prevent aggressors from closing distance. In short, agents must balance the reduced risk of attempting to forage passively, protecting themselves only when needed, against the greater reward of attacking their neighbors to pilfer their resources and cull the competition. [6]

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. A higher Hunting level enables agents to gain more from resource tiles and also carry more food. The same is true of Fishing and water tiles, and 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. Higher Constitution directly increases an agent's maximum health. Higher Defense decreases the accuracy of opponents' attacks. An overall level is calculated for each agent based off of their combat stats. 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.

Front End Client: The Neural MMO client is written in C# using Unity3D and follows a design shared by multiple MMOs: while the game state is a 2D tile grid, the client renders in full 3D with smooth animations. As the client is only used for test-time visualization, this approach enables us to achieve far greater computational efficiency than physically simulated environments.[7] At the same time, access to a full game engine allows us to maintain game play complexity without sacrificing interpretability or visual fidelity. A unique programmable name is displayed above each agent along with its health, food, and water values. We also display additional status effects, such as whether an agent is frozen and the level range within which it is attackable. Users can click on specific agents to follow them with the camera or view more detailed stats. Our previous THREE.js client contained a number of useful research overlays for visualizing agent exploration behaviors and value maps (Figure 3.2), which will be ported to the Unity3D client soon.

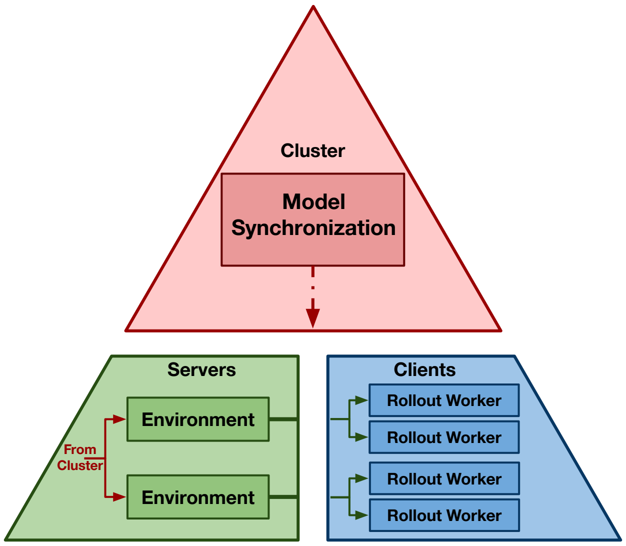


Figure 3.4: Hardware diagram of Neural MMO infrastructure (Algorithm 2) in three Ascend layers. The main difference from Rapid is our placement of the environment on the server rather than the client.

3.4 The IO Problem

Most simulated environments used in reinforcement learning offer an interface in which observations are represented as raw tensors and actions are selected by sampling from flat logits: it is a simple task to input observations into the network and convert network outputs into actions. However, in more complex environments, observation and actions spaces may be complex, of variable size, and hierarchical. As demonstrated by recent scale up work on DoTA, the exact structure of these IO spaces can require complex environment specific architectures simply to enable natural representations of observations and actions. We refer to these issues collectively as *the IO problem*. This work presents pluggable modules that makes interfacing with Neural MMO almost as simple as working with a traditional game environment. Instead of hand engineering a network to match the particular IO spaces of Neural MMO, we have taken the opportunity to design a more generic framework for input and output processing applicable to a broad class of environments. Our solution implements a substantially general automatic network generation process (see Figure 3.3) that can be implemented as a layer on top of existing environments. While our work is in an interpreted language, we will refer to *compile time* and *run time* to denote static and instance specific operations.

Notes: One tempting solution is to simply render the environment and give agents the

[6] Large scale battles at spawn are undesirable in MMOs. For the map used in our baseline (Figure 3.1), agents may not attack agents that have recently spawned. For the new map in Figure 3.2, agents may be attacked only by other agents within a fixed distance of their level (see Progression System). Agents are perfectly safe within a small region of the spawn containing little to no food. As they travel farther, increasingly powerful agents are able to attack them, but more resources are present. This situation adds another risk-reward trade off layer to the environment.

[7] In our experiments, only 0.5-2.0 percent of total CPU is used to simulate the environment

Algorithm 2 The Ascend API provides three abstractions over standard distributed infrastructure.

```
0: class ASCEND
0:   function DISTRIBUTE(arguments, shard=None)
0:     asyncHandles = List()
0:     shardedArguments = Shard(arguments, shard)
0:     for worker in remote workers do
0:       handle = worker.step(shardedArguments)
0:       asyncHandles.append(handle)
0:     return asyncHandles
0:   function SYNCHRONIZE(asyncHandles)
0:     remoteReturns = List()
0:     for handle asyncHandles do
0:       data.append(await(handle))
0:     return data
0:   function STEP(arguments, shard=None)
0:     asyncHandles = distribute(arguments, shard)
0:     return synchronize(asyncHandles)
   =0
```

same interface as humans. However, rendering complex environments is almost always much more computationally expensive than simulating agent policies. Without rendering, agents no longer can use a mouse and keyboard – there is nothing to click on. Our work bypasses this human interface and allows agents to interact with the raw game API.

3.4.1 The Input Problem

Terminology: 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, etc.).

Assumptions: We assume that the attributes for each entity type are declared at compile time. This implies that it is possible to build a static tree of attribute data types without querying specific entities at run time. Our work currently supports continuous and discrete attributes, though it is possible to support additional data types in the future. We assume reasonable estimates of each attribute's scale. For discrete values: a lower bound on the minimum value and an upper bound on the maximum value. For continuous values: a rough estimate of the mean and standard deviation. Note: estimates aid in input normalization but are not strictly necessary. It is also possible to collect them empirically at runtime.

Solution: Figure 3.3 describes our solution to the input problem. At compile time, the user specifies embedding functions for each attribute type (e.g. different functions for embedding continuous and discrete values). Our framework uses these function handles and the static attribute tree in order to generate embedding layers $e_1, e_2, ...$ for each attribute. The user also specifies two attention functions, f and g to be used later. At run time, we use the static graph structure and associated layers e_i to embed $x_1, x_2, ..., x_n$, producing fixed length attribute embedding vectors $y_1, y_2, ..., y_n$. To produce a fixed length vector representation z_i for each entity, we aggregate across attributes using the specified attention function f. The action network will need these entity embeddings later. To obtain a fixed length representation o of the entire observation, we aggregate across observed entities $z_1, z_2, ..., z_n$ using the second specified attention function g. Finally, we return the flat observation embedding and the variable sized lookup tensor of entity embeddings.

3.4.2 The Output Problem

Terminology: We define agent decision space by a set of action-argument pairs. Actions are callable function references that the environment can invoke on the associated argument list in order to execute an agent's decision. For example: Move $\rightarrow$ [North] or Attack $\rightarrow$ [Melee, Agent ID]

Assumptions: We assume that the set of actions and their argument types are declared at compile time. Our work currently supports discrete, and entity valued arguments, though it is possible to support additional data types in the future.

Solution: Figure 3.3 describes our solution to the output problem. At compile time, the user specifies a hard attentional architecture h and output networks (unembedding layers) for each argument type. Using the static action set, our framework then generates decision layers for each argument. At run time, we convert the hidden state of the main network into an action + argument-list pair. To do so, we first embed all arguments to produce fixed length vector representations $a_1, a_2, ..., a_k$. For entity values arguments, we simply copy the $z_1, z_2, ...$ entity representations from the input network. We then compare these embeddings to the hidden state using the attentional function h to produces hard attentional choices over arguments. Finally, we match the selected embedding to the corresponding argument game object.

3.5 Distributed Infrastructure

Modern reinforcement learning infrastructure makes a number of assumptions about the underlying task. Most notably, it has become standard practice to centralize all policy computation and optimization on a centralized GPU server, using a bank of remote CPU clients to produce training data by simulating many environment instances in parallel. This practice is most often implemented as a simple broadcast and aggregate operation in raw MPI. We will refer to this common usage as the MPI approach, though MPI itself is a much more general framework. OpenAI has recently proposed an alternative infrastructure configuration, Rapid, which they used to train agents to play DoTA [2] above human skill. This was one of the largest (and possibly the largest) reinforcement learning project undertaken at the time, and communication latency became a problem. Rapid remedies this issue by collocating agent policies with dedicated GPU rollout workers that simulate full agent trajectories independently. Separate dedicated GPU optimizer servers aggregate experience in large batches from many rollout workers and environments, thus removing the need for constant high-latency communication among remote hardware.

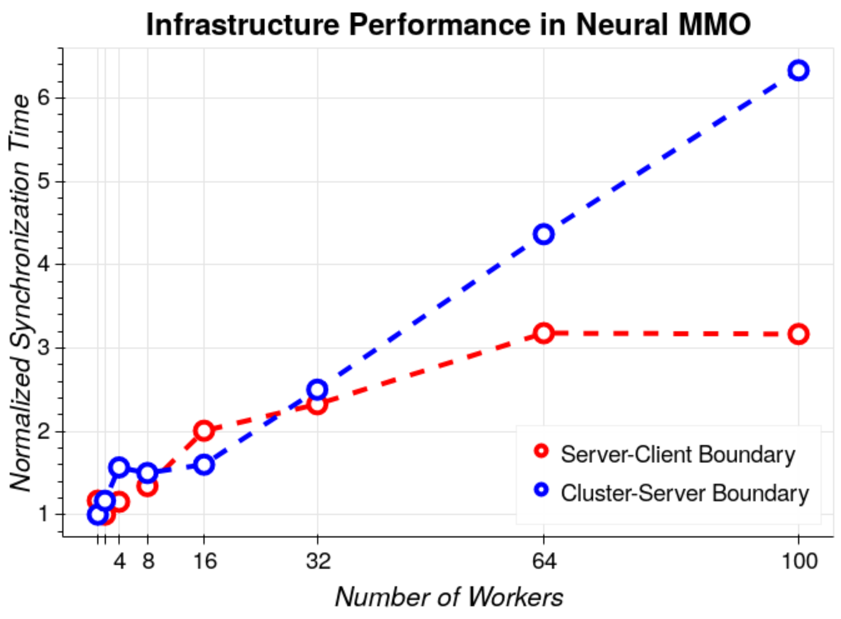


Figure 3.5: Neural MMO sync. times scale linearly with # cluster workers and sublinearly with # server workers.

We argue that neither MPI nor Rapid scale well in massively multiagent settings. In their work on DoTA, OpenAI used approximately 100 environments per GPU optimizer server. Classic MMOs such as Ultima Online and Runescape supported thousands of players per live server on 90s to early 2000s hardware. In a deep learning setting, scaling to this number of agents per environment decreases the number of environments that each server can support. MPI and Rapid both assume a one-to-many ratio of optimizers to environments; real MMOs by necessity operate using a many-to-one ratio. Instead of supporting hundreds of environments per server, suddenly one (or multiple) servers are required to support each environment. The bandwidth optimization of Rapid over MPI is an orthogonal improvement that does not alter this one-to-many ratio assumption. Our work implements MMO style infrastructure in deep learning to perform distributed training in Neural MMO.

3.5.1 Ascend Distributed API

Ascend is a protocol that models infrastructure and associated logging at an arbitrary hardware layer. We implement Ascend as a lightweight wrapper around Ray, a popular general purpose distributed computing library. Our API provides a single eponymous Ascend object which specifies synchronous and asynchronous communication interfaces to the previous and next hardware layers. Stacking three such layers produces the cluster-server-client architecture needed for Neural MMO in only a few lines of specialized code.

Algorithm 2 details the API. Ascend provides three core subroutines: distribute, synchronize, and step. The distribute function invokes all workers in the next layer asynchronously. It returns awaitable handles and supports argument sharding across clients. The synchronize function waits for all remote clients, as invoked by distribute, to terminate and aggregates

Algorithm 3 Neural MMO training logic for one game tick. Note that we abstract rollout collection.

```
0: for each environment server do
0:   if number of agents alive < spawn cap then
0:     spawn an agent
0:   for each agent do
0:     i ← population index of the agent
0:     Make observation $o_t$, decide action $\pi_i(o_t) \to a_t$
0:     Environment processes $a_t$ and computes $r_t$
0:     if agent is dead then
0:       remove agent
0:   Update environment state $s_{t+1} \to f(s_t, a_t)$
0: Perform a policy gradient update on policies $\pi \sim \pi_1, \ldots, \pi_N$ using $o_t$, $a_t$, $r_t$ from all agents
   across all environment servers =0
```

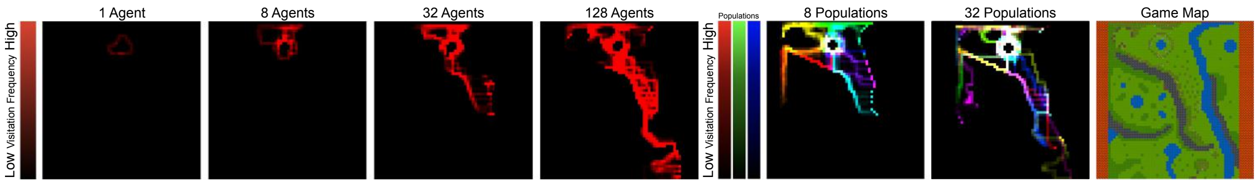


Figure 3.6: Visitation frequency overlaid over the game map. (Left) Population size magnifies exploration: agents spread out to avoid competition. (Right) Populations count (unshared policies, separated by color) magnifies niche formation (8 populations) until all exploration strategies are saturated (32 populations).

their returns. The step function provides a synchronous remote interface by simply calling distribute followed by synchronize. The user specifies behavior at each hardware level by overriding these three methods.

3.5.2 Neural MMO

Ascend is a general framework that can be used to implement the computational models of MPI and Rapid in only a few lines. Figure 3.4 details our specific usage of Ascend in Neural MMO, which has three layers. The cluster layer controls a single node used for top level experiment management, logging, and model update synchronization across a bank of servers. Each server simulates a Neural MMO environment instance, performs various observation preprocessing and action postprocessing, and communicates with a bank of clients. Each client manages a dynamic, variable length subset of the corresponding server's agents. This includes batching observations, running the policy, collecting rollouts, and performing backpropagation. We focus on reinforcement learning in this work, but our framework also supports other approaches, such as genetic algorithms.

3.5.3 Scale

We evaluate the performance of Ascend in Figure 3.5 by considering synchronization times at two boundaries: cluster-server and server-client. We vary the number of cluster workers (environment servers) and server workers (policy clients) independently. For all trials, we evaluate synchronization time for one gradient step over a batch of size 4096 agent decisions per environment. This experiment was conducted on a small local cluster of 10 machines with 10 usable cores each.

The cluster receives a fixed length vector of gradient updates from each server. Thus, linear synchronization time is the ideal result for the cluster-server boundary, with bandwidth as the limiting factor. At the server-client boundary, the same amount of data is shared across all clients regardless of their number. As such, the ideal result is constant synchronization time independent of the number of clients. The empirical results shown in Figure 3.5 are consistent with the expected trends. For both synchronization boundaries considered, results are noisy only at very small scale (one machine). As we scale from 10 to 100 CPU cores, the cluster-server synchronization time converges to linear performance. Server-client synchronization appears to plateau with a roughly constant overhead factor of 3. Some deviation from ideal is to be expected. We expect this factor could be reduced through better load balancing and serialization: clients do not receive perfectly equal splits of the observations, and serializing many small data packets to be sharded across several clients is less efficient than serializing a few larger packets.

3.6 Experiments

The main contributions of this work are our environment and the associated general purpose multiagent systems. However, we also create an end-to-end training pipeline on Neural MMO in order to test our infrastructure and IO. In the Policy and Training sections below, we train populations of agents to perform basic foraging and combat behaviors on the latest version of our environment mechanics and the map generation shown in Figures 3.1 and 3.2 (left). The Emergent Behavior section describes interesting qualitative properties of multiagent interaction discovered using Neural MMO v1.0.

3.6.1 Policy

We define our architecture by the functions f, g, and h in Figure 3.2. For the function f, which attends over attribute embeddings, we use one layer of single-headed scaled dot product attention [27]. The funtion g applies attention over agent embeddings and convolution over tile embeddings to produce a flat observation embedding. In the output network, agents submit one movement and one attack action per game tick. The function h that is used to select arguments first applies a small fully connected network to the hidden state and argument embeddings. We then use dot product similarity followed by a softmax to compute a distribution over action arguments. Sampling from this distribution produces the final argument selections.

3.6.2 Reward Formulation and Training

Agents are controlled by policies parameterized by neural networks. Each agent makes a partial observation o_t of the game state s_t and follows a policy $\pi(o_t) \to a_t$ in order to make action(s) a_t. We maximize a return function R over trajectory $\tau = (o_t, a_t, r_t, ..., o_T, a_T, r_T)$. Neural MMO provides a custom reward shaping API, but for simplicity we use a discounted sum of survival rewards of form $[0, 0, ..., -1]$: $R(\tau) = \sum_t^T \gamma^t r_t$ where $\gamma = 0.95$, T is the time at death, and reward is $r_{t=T} = -1; r_{t \neq T} = 0$. Each rollout corresponds to an agent lifetime. As described in Algorithm 3, we sample agent policies from eight different populations and train with policy gradients [28] using a learned value baseline [29]. The policy weights π are shared only within each population, excepting embedding weights, which are shared across all populations.

The v1.3 baseline uses a batch size of 16k actions [8]. The cluster aggregates gradients across all servers and broadcasts policy updates back to the clients, ensuring weights are never stale. We clip gradients to a maximum absolute value of 5.0 and update the network using Adam with learning rate 3e-4 and weight decay 1e-5. We observe that agents learn basic foraging and combat behaviors and release trained policies.

3.6.3 Emergent behavior

These experiments were performed on an earlier version of the environment, but the core game remains unchanged, and our open source release includes a branch with this version of the environment for reproducibility. Key differences from the current environment include map generation, lack of a progression system, and slight mechanical tweaks.

N_{ent}: #Agents Magnifies Exploration

The left half of Figure 3.6 compares map coverage vs. population size. In the natural world, competition between animals can incentivize them to spread out in order to avoid conflict. We observe that overall exploration (map coverage) increases as the number of concurrent agents increases. Agents learn to explore only because the presence of other competing agents provides a natural incentive for doing so.

N_{pop}: #Populations Magnifies Niche Formation

The right half of Figure 3.6 compares map coverage vs. number of populations (agents with unshared weights). We find that, given a sufficiently large and resource-rich environment, different populations of agents tend to separate to avoid competing with other populations. The real world often rewards masters of a single craft more jacks of all trades. Figure 3.6 suggests that specialization to particular regions of the map increases as number of populations increases. We believe this indicates that the presence of other populations force agents to discover a single advantageous skill or trick: increasing the number of populations results in diversification to separable regions of the map. As entities cannot out-compete

[8] our v1.0 experiments used a noisier policy gradients formulation and required a batch size of 265k actions

other agents of their own population (because they share the same policy), they tend to seek large areas of the map that contain enough resources to sustain their entire population.

3.7 Discussion and Limitations

We have made significant progress towards enabling seamless reinforcement learning research on massively multiagent environments, but much is left to be done. In this section, we describe the portions of our work that we believe could benefit from additional generalization.

3.7.1 IO

The object centric model of observation space enables a robust attentional mechanism over attributes and entities to create a natural and general mechanism for flattening a complex local game state. It further enables direct object queries in the action space by keying against entity embeddings. However, we found it difficult to train attention layers over a large number of tile embeddings. This is the reason we used a convolutional layer over tile embeddings in our experiments. While we expect attention to be trainable with some additional network tuning, memory remains an issue. Attention is quadratic in the number of entities, and agents observe a 15 by 15 crop of tiles, or 225 total. One potential solution is to factorize the attentional mechanism. Standard attention prepossesses the input x with three linear layers, Q, K, and V. Instead, one could, for example, replace each of K, Q, and V with k linear layers. By taking max pooled average over k projections of x along the dimension corresponding to the number of entities, it is possible to reduce the memory required for activations to $O(k^2 + n)$.

Our output module is a more temporary solution: it works for the level of complexity in modern environments, but it will require support for additional data types to be applicable in something as complex as a full MMO. However, we emphasize that reinforcement learning environments, including Neural MMO, are not yet complex enough for this to be a concern. That is, this discussion will not be relevant until after a few iterations of environment development.

3.7.2 Infrastructure

The current environment version is in fact amenable to Rapid style infrastructure. However, neither MPI nor Rapid style infrastructure will scale well once we begin increasing agent population sizes. This factored into our decision to invest in infrastructure early. Another factor in our decision was a quirk of reinforcement learning regarding memory. People play MMOs for hundreds to thousands of hours, and decisions made fifty hours into the game impact play five hundred hours later. Because of reward discounting, we need to keep full trajectories for all living agents in memory. We tried multiple schemes for reducing memory overhead, including recomputing the forward pass, but we found this to be too complex and computationally expensive to merit further consideration. Possible alternative approaches include the trajectory segment processing formulation used in OpenAI Five, off-policy methods that consider single transitions, and evolutionary methods that do not keep trajectories in memory at all. Intrinsic reward methods could also be useful to reduce the effective required time scale.

3.8 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 80 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. The environment has also served as a useful case study in multiagent systems, which has allowed us to address infrastructure and IO problems in this emerging new research space. We hope that our solutions will prove useful to others developing massively multiagent systems.

Chapter 4

Neural MMO 1.5

Chapters 4-5 cover the first truly stable versions of Neural MMO. This included a few additions to the environment itself, such as the introduction of scripted enemies and a basic equipment system. The main advancement, however, was in the core infrastructure and baselines. Neural MMO's game state representation was reformatted using a technique from the games industry to enable faster accesses. This made the main bottleneck operation 100x faster, which enabled the project to continue scaling to 10x more agents and 100x larger maps. At the same time, investment into the baselines produced the first set of models that were clearly better than the 1.0 models trained on OpenAI hardware. This version of Neural MMO was able to train 1 billion agent steps, equivalent to 19 years of real-time data, in 2-3 weeks on a single desktop.

The following paper titled "The Neural MMO Platform for Massively Multiagent Research" was published in the NeurIPS 2021 Datasets and Benchmarks track. It is coauthored with Yilun Du et al. and advised by Phillip Isola and Igor Mordatch.

Neural MMO is a computationally accessible research platform that combines large agent populations, long time horizons, open-ended tasks, and modular game systems. Existing environments feature subsets of these properties, but Neural MMO is the first to combine them all. We present Neural MMO as free and open source software with active support, ongoing development, documentation, and additional training, logging, and visualization tools to help users adapt to this new setting. Initial baselines on the platform demonstrate that agents trained in large populations explore more and learn a progression of skills. We raise other more difficult problems such as many-team cooperation as open research questions which Neural MMO is well-suited to answer. Finally, we discuss current limitations of the platform, potential mitigations, and plans for continued development.

4.1 Introduction

The initial success of deep Q-learning [30] on Atari (ALE) [31] demonstrated the utility of simulated games in reinforcement learning (RL) and agent-based intelligence (ABI) research as a whole. We have since been inundated with a plethora of game environments and platforms including OpenAI's Gym Retro [32], Universe [33], ProcGen [34], Hide&Seek [35], and Multi-Agent Particle Environment [36], [37], DeepMind's OpenSpiel [35], Lab [38], and Control Suite [39], FAIR's NetHack learning environment [39], Unity ML-Agents [40], and others including VizDoom [41], PommerMan [42], Griddly [43], and MineRL [44]. This breadth of tasks has provided a space for academia and industry alike to develop stable agent-based learning methods. Large-scale reinforcement learning projects have even defeated professionals at Go [45], DoTA 2 [46], and StarCraft 2 [47].

Intelligence is more than the sum of its parts: most of these environments are designed to test one or a few specific abilities, such as navigation [38], [39], robustness [34], [42], and collaboration [35] – but, in stark contrast to the real world, no one environment requires the full gamut of human intelligence. Even if we could train a single agent to solve a diverse set of environments requiring different modes of reasoning, there is no guarantee it would learn to combine them. To create broadly capable *foundation policies* analogous to foundation models [48], we need multimodal, cognitively realistic tasks analogous to the large corpora and image databases used in language and vision.

Neural MMO is a platform inspired by Massively Multiplayer Online games, a genre that simulates persistent worlds with large player populations and diverse gameplay objectives. It features game systems configurable to research both on individual aspects of intelligence (e.g. navigation, robustness, collaboration) and on combinations thereof. Support spans 1 to 1024 agents and minute- to hours-long time horizons. We first introduce the platform and address the challenges that remain in adapting reinforcement learning methods to increasingly general environments. We then demonstrate Neural MMO's capacity for research by using it to formulate tasks poorly suited to other existing environment and platforms – such as exploration, skill acquisition, and learning dynamics in variably sized populations. Our contributions include Neural MMO as free and open source software (FOSS) under the MIT license, a suite of associated evaluation and visualization tools, reproducible base-

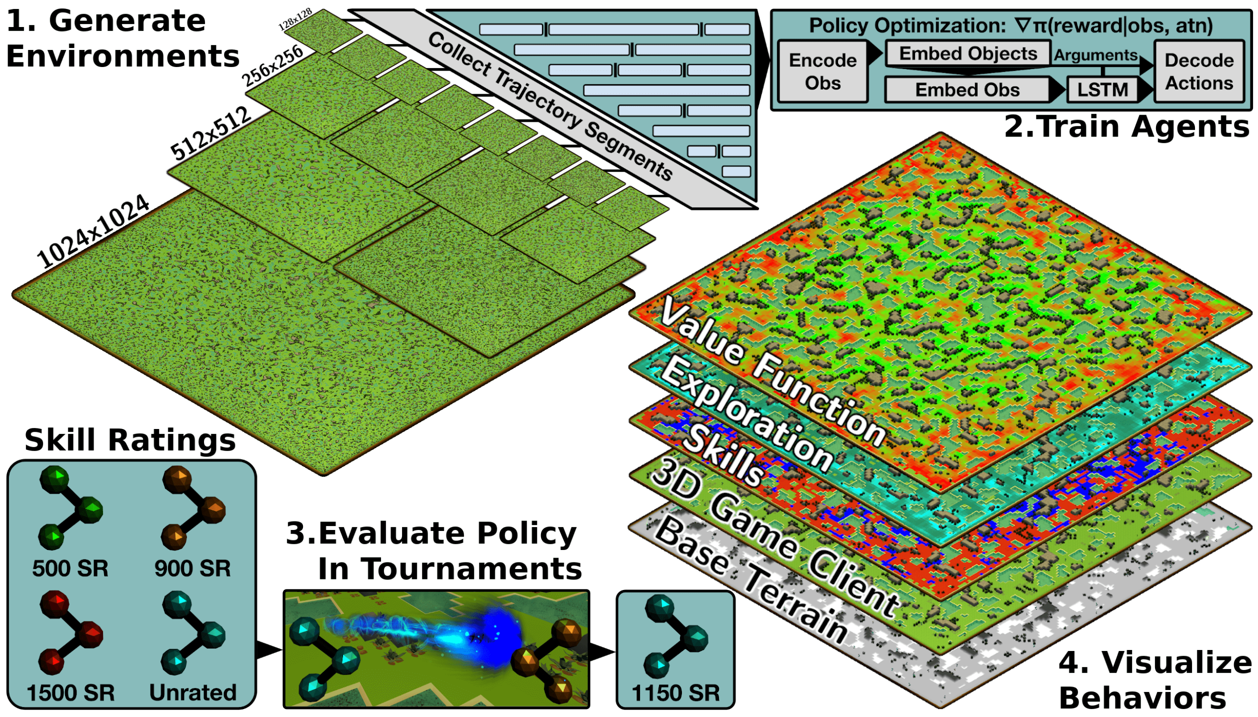


Figure 4.1: A simple Neural MMO research workflow suitable for new users (advanced users can define new tasks, customize map generation, and modify core game systems to suit their needs)
1. Select one of our default task configurations and run the associated procedural generation code
2. Train agents in parallel on a pool of environments. We also provide a scripting API for baselines
3. Run tournaments to score agents against several baseline policies concurrently in a shared world
4. Visualize individual and aggregate behaviors in an interactive 3D client with behavioral overlays.

lines, a demonstration of various learned behaviors, and a pledge of continued support and development.

4.2 The Neural MMO Platform

Neural MMO simulates populations of agents in procedurally generated virtual worlds. Users tailor environments to their specific research problems by configuring a set of game systems and procedural map generation. The platform provides a standard training interface, scripted and pretrained baselines, evaluation tools for scoring policies, a logging and visualization suite to aide in interpreting behaviors, comprehensive documentation and tutorials, and an active community Discord server for discussion and user support. Fig. 4.2 summarizes a standard workflow on the platform, which we detail below.

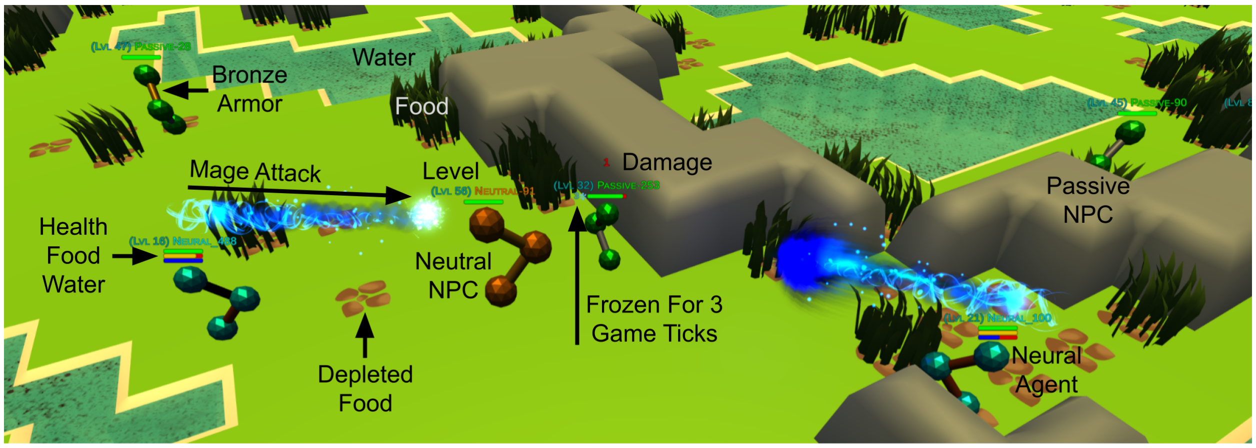


Figure 4.2: Perspective view of a Neural MMO environment in the 3D client

neuralmmo.github.io hosts the project and all associated documentation. Current resources include an installation and quickstart guide, user and developer APIs, comprehensive baselines, an archive of all videos/manuscripts/slides/presentations associated with the project, additional design documents, a list of contributors, and a history of development. Users may compile documentation locally for any previous version or commit. We pledge to maintain the community Discord server as an active support channel where users can obtain timely assistance and suggest changes. Over 400 people have joined thus far, and our typical response time is less than a day. We plan to continue development for at least the next 2-4 years.

4.2.1 Configuration

Modular Game Systems: The Resource, Combat, Progression, and Equipment & NPC systems are bundles of content and mechanics that define gameplay. We designed each system to engage a specific modality of intelligence, but optimal play typically requires simultaneous reasoning over multiple systems. Users write simple config files to enable and customize the game systems relevant to their application and specify map generation parameters. For example, enabling only the resource system creates environments well-suited to classic artificial life problems including foraging and exploration; in contrast, enabling the combat system creates more obvious conflicts well-suited to team-based play and ad-hoc cooperation.

Procedural Generation: Recent works have demonstrated the effectiveness of procedural content generation (PCG) for domain randomization in increasing policy robustness [49]. We have reproduced this result in Neural MMO (see Experiments) and provide a PCG module to create *distributions* of training and evaluation environments rather than a single game map. The algorithm we use is a novel generalization of standard multi-octave noise that varies generation parameters at different points in space to increase visual diversity. All terrain generation parameters are configurable, and we provide full details of the algorithm in the Supplement.

Canonical Configurations: In order to help guide community research on the platform, we release a set of config files defining standard tasks and commit to maintaining a public list of works upon them. Each config is accompanied by a section in Experiments motivating the task, initial experimental results, and a pretrained baseline where applicable.

4.2.2 Training

User API: Neural MMO provides `step` and `reset` methods that conform to RLlib's popular generalization of the standard OpenAI Gym API to multiagent settings. The reward function is -1 for dying and 0 otherwise by default, but users may override the `reward` method to customize this training signal with full access to game and agent state. The `log` function saves user-specified data at the end of each agent lifetime. Users can specify either a single key and associated metric, such as "Lifetime": `agent.lifetime`, or a dictionary of variables to record, such as "Resources":{"Food": 5, "Water: 0}. Finally, an `overlay` API allows users to create and update 2D heatmaps for use with the tools below. See Fig. 4.3 for example usage and `neuralmmo.github.io` for the latest API.

Observations and Actions: Neural MMO agents observe sets of *objects* parameterized by discrete and continuous *attributes* and submit lists of *actions* parameterized by lists of discrete and object-valued *arguments*. This parameterization is flexible enough to avoid major constraints on environment development and amenable to efficient serialization (see documentation) to avoid bottlenecking simulation. Each observation includes 1) a fixed crop of *tile* objects around the given agent parameterized by *position* and *material* and 2) the other *agents* occupying those tiles parameterized by around a dozen properties including current *health*, *food*, *water*, and *position*. Agents submit *move* and *attack* actions on each timestep. The *move* action takes a single *direction* argument with fixed values of *north*, *south*, *east*, and *west*. The *attack* action takes two arguments: *style* and *target*. The *style* argument has fixed values of *melee*, *range*, and *mage*. The agents in the current observation are valid *target* argument values. Encoding/decoding layers are required to project the hierarchical observation space to a fixed length vector and the flat network hidden state to multiple actions. We also provide reusable PyTorch subnetworks for these tasks.

Efficiency and Accessibility: A single RTX 3080 and 32 CPU cores can train on 1 billion observations in just a few days, equivalent to over 19 *years* of real-time play using RLlib's *synchronous* PPO implementation (which leaves the GPU idle during sampling) and a fairly simple baseline model. The environment is written in pure Python for ease of use and modification even beyond the built-in configuration system. Three key design decisions enable Neural MMO to achieve this result. First, training is performed on the partially observed game state used to render the environment, but no actual rendering is performed except for visualization. Second, the environment maintains an up-to-date serialized copy of itself in memory at all times, allowing us to compute observations using a select operator over a flat tensor. Before we implemented this optimization, the overhead of traversing Python object hierarchies to compute observations caused the entire environment to run 50-100x slower. Finally, we have designed game mechanics that enable complex play while being simple to simulate. [1] See `neuralmmo.github.io` for additional design details.

[1]By adopting the standard game development techniques that enabled classic MMOs to simulate small cities of players on 90s hardware. For those interested or familiar, we leave in a few pertinent details in later

4.2.3 Evaluating Agents

Neural MMO tasks are defined by a reward function on a particular environment configuration (as per above). Users may create their own reward functions with full access to game state, including the ability to define per-agent reward functions. We also provide two default options: a simple survival reward (-1 for dying, 0 otherwise) and a more detailed achievement system. Users may select between *self-contained* and *tournament* evaluation modes, depending on their research agenda.

Achievement system: This reward function is based on gameplay milestones. For example, agents may receive a small reward for obtaining their first piece of armor, a medium reward for defeating three other players, and a large reward for traversing the entire map. The tasks and point values themselves are clearly domain-specific, but we believe this achievement system has several advantages compared to traditional reward shaping. First, agents cannot farm reward [32] – in contrast to traditional reward signals, each task may be achieved only once per episode. Second, this property should make the achievement system less sensitive to the exact point tuning. Finally, attaining a high achievement score somewhat guarantees complex behavior since tasks are chosen based upon difficulty of completion. We are currently running a public challenge that requires users to optimize this metric. [2].

Self-contained evaluation pits the user's agents against copies of themselves. This is the method we use in our experiments and the one we recommend for artificial life work and studies of emergent dynamics in large populations. It is less suitable for benchmarking reinforcement learning algorithms because agent performance against clones is not indicative of overall policy quality.

Tournament evaluation solves this problem by instead pitting the user's agents against different policies of known skill levels. We recommend this method for direct comparisons of architectures and algorithms. Tournaments are constructed using a few user submitted agents and equal numbers of several scripted baselines. We run several simulations for a fixed (experiment dependent) number of timesteps and sort policies according to their average collected reward. This ordering is used to estimate a single real number skill based on an open-source ranking library [3] for multiplayer games. We scale this skill rating (SR) estimate such that, on any task, our scripted combat bot scores 1500 with a difference of 100 SR indicating a 95 percent win rate. Users can run tournaments against scripted bots locally. For the next few months, we are also hosting public evaluation servers where anyone can have their agents ranked against other user-submitted agents. The neural combat agent in Table 4.1 scores 1150 and the scripted foraging agent scores 900. We have since improved the neural combat agent to 1600 SR through stability enhancements in the training infrastructure.

4.2.4 Logging and Visualization

Interpreting and debugging policies trained in open-ended many-agent settings can be difficult. We provide integration with WanDB that plots data recorded by the `log` function

footnotes.

[2]Competition page with the latest tasks: https://www.aicrowd.com/challenges/the-neural-mmo-challenge

[3]TrueSkill [50] for convenience, but one could easily substitute more permissively licensed alternatives.

Basic Usage

```
'''Extended OpenAI Gym Interface'''

From neural_mmo.forge.blade.io.action.static
import *
from projekt.config import CompetitionRound1

config = CompetitionRound1()
env  = CustomEnv(config)
obs  = env.reset()

actions = {}
for agentID in obs:
   players = env.realm.players
   agent   = players[agentID]

   actions[agentID] = {
      Move: {Direction: North}}

   obs, rewards, done, _ =(
        env.step(actions))
```

Custom Reward Function

```
from neural_mmo.forge.trinity import Env

class CustomEnv(Env):
   '''Team Spirit shared team reward
   as proposed by OpenAI Five'''

   def reward(self, ent):
      config  = self.config
      players = self.realm.players

      dead = ent.entID not in players
      nDead = len([p for p in
           self.dead.values() if
           p.population == ent.pop])

      individual = -1 if dead else 0
      team = -nDead/config.TEAM_SIZE

      alpha = config.TEAM_SPIRIT
      return (alpha*team +
           (1.0-alpha)*individual)
```

Custom Overlay (with RLlib)

```
from projekt.rllib_wrapper import RLlibOverlay

class Values(RLlibOverlay):
   '''Rendered in the Unity 3D Client'''
   def update(self, obs):
      players = self.realm.realm.players
      for idx, playerID in enumerate(obs):
         if playerID not in players:
            continue

         r, c = players[playerID].base.pos

         self.values[r, c] = float(
         self.model.value_function()[idx])

   def register(self, obs):
      colorized = overlay.twoTone(
          self.values[:, :])
      self.realm.register(colorized)
```

Figure 4.3: Neural MMO's API enables users to program in a familiar OpenAI Gym interface, define per-task reward functions, and visualize learned policies with in-game overlays.

during training and evaluation. The default `log` includes statistics that we have found useful during training, but users are free to add their own metrics as well. See the documentation for sample plots. For more visual explorations of learned behaviors, Neural MMO enables users to render their 2D heatmaps produced using the `overlay` API directly within the 3D interactive client. This allows users to watch agent behaviors and gain insight into their decision making process by overlaying information from the model. For example, the top heatmap in Fig. 4.2 illustrates which parts of the map agents find most and least desirable using the learned value function. Other possibilities include visualizations of aggregate exploration patterns, relevance of nearby agents and tiles to decision making, and specialization to different skills. We consider some of these to interpret policies learned in our experiments.

4.3 Game Systems

The base game representation is a grid map[4] comprising grass, forest, stone, water, and lava tiles. Forest and water tiles contain resources; stone and water are impassible, and lava kills agents upon contact. At the same time, this tile-based representation is important for computational efficiency and ease of programming – see the supplementary material for a more detailed discussion.

Resources: *This system is designed for basic navigation and multiobjective reasoning.* Agents spawn with food, water, and health. At every timestep, agents lose food and 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, in the presence of 100+ potentially hostile agents attempting

[4]It is a common misconception in RL that grid-worlds are fundamentally simplistic: Some of the most popular and complex MMOs on the market partition space using a grid and simply smooth over animations. These include RuneScape 3, OldSchool Runescape, Dofus, and Wakfu, all of which have existed for 8+ years and maintained tens of thousands of daily players and millions of unique accounts

to do the same. The starting and maximum quantities of food, water, and health, as well as associated loss and regeneration rates, are all configurable.

Combat: *This system is designed for direct competition among agents.* Agents can attack each other with three different styles – Range, Mage, and Melee. The attack style and combat stats of both parties determine accuracy and damage. 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. Accuracy, damage, and attack reach are configurable for each combat style.

Skill Progression: *This system is designed for long-term planning.* MMO players progress both by improving mechanically and by slowly working towards higher skill levels and better equipment. Policies must optimize not only for short-term survival, but also for strategic combinations of skills. In Neural MMO, 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. Higher offensive levels increase accuracy and damage while Constitution and Defense increase maximum health and evasion, respectively. Starting levels and experience rates are both configurable.

NPCs & Equipment: *This system is designed to introduce risk/reward tradeoffs independent from other learning agents.* Scripted non-playable characters (NPCs) with various abilities spawn throughout the map. Passive NPCs are weak and flee when attacked. Neutral NPCs are of medium strength and will fight back when attacked. Hostile NPCs have the highest levels and actively hunt nearby players and other NPCs. Agents gain combat experience and equipment by defeating NPCs, which spawn with armor determined by their level. Armor confers a large defensive bonus and is a significant advantage in fights against NPCs and other players. The level ranges, scripted AI distribution, equipment levels, and other various features are all configurable.

4.4 Models

Pretrained Baseline: We use RLlib's [51] PPO [52] implementation to train a single-layer LSTM [53] with 64 hidden units. Agents act independently but share a single set of weights; training aggregates experience from all agents. Input preprocessor and output postprocessor subnetworks are used to fit Neural MMO's observation and action spaces, much like in OpenAI Five and AlphaStar. Full architecture and hyperparameter details are available in the supplementary material. We performed only the minimal hyperparameter tuning required to optimize training for throughput and memory efficiency. Each experiment uses hardware that is reasonably available to academic researchers: a single RTX 3080 and 32 cores for 1-5 days – Up to 100k environments and 1B agent observations.

Scripted Bots: The Scripted Forage baseline shown in Table 4.1 implements a locally optimal min-max search for food and water over a fixed time horizon using Dijkstra's algorithm but does not account for other agents' actions. The Scripted Combat baseline possesses an additional heuristic that estimates the strength of nearby agents, attacks those

Table 4.1: Baselines on canonical SmallMaps and LargeMaps configs with all game systems enabled. Refer to Section 5 for definitions of Metrics and analysis to aid in Interpreting Results. The highest value for each metric is bolded for both configs.

Model	Lifetime	Achievement	Player Kills	Equipment	Explore	Forage
Small maps						
Neural Forage	132.14	2.08	0.00	0.00	9.73	18.66
Neural Combat	51.86	3.35	**1.00**	**0.27**	5.09	13.58
Scripted Forage	**252.38**	**7.56**	0.00	0.00	**37.07**	**26.18**
No Explore	224.30	4.34	0.00	0.00	21.19	21.87
Scripted Combat	76.52	3.45	0.69	0.11	14.81	16.15
No Explore	52.11	2.72	0.73	0.18	9.75	14.35
Scripted Meander	28.62	0.08	0.00	0.00	4.46	11.49
Large maps						
Neural Forage	356.13	2.75	0.00	0.00	10.20	18.02
Neural Combat	57.12	2.34	**0.96**	0.00	3.34	11.96
Scripted Forage	**3224.31**	**28.97**	0.00	0.00	**136.88**	**47.83**
Scripted Combat	426.44	10.75	0.71	0.04	53.15	24.60
Zero-shot						
$S \rightarrow L$	73.48	3.15	0.88	0.03	7.52	13.46
$L \rightarrow S$	35.24	3.15	1.01	0.25	2.93	12.73

it perceives as weaker, and flees from stronger aggressors. Both of these scripted baselines possess an optional Exploration routine that biases navigation towards the center of the map. Scripted Meander is a weak baseline that randomly explores safe terrain.

4.5 Baselines and Additional Experiments

Neural MMO provides small- and large-scale tasks and baseline evaluations using both scripted and pretrained models as canonical configurations to help standardize initial research on the platform.

Learning Multimodal Skills: The canonical SmallMaps config generates 128x128 maps and caps populations at 256 agents. Using a simple reward of -1 for dying and a training horizon of 1024 steps, we find that the recurrent model described in the last section learns a *progression of different skills* throughout the course of training (Fig. 4.4). Basic foraging and exploration are learned first. Average metrics for these skills drop later in training as agents learn to fight: the policies have actually improved, but the task has become more difficult in the presence of adversaries capable of combat. Finally, agents learn to selectively target passive NPCs as they do not fight back, grant combat experience, and drop equipment upgrades. This progression of skills occurs without any direct reward or incentive for anything but survival. We attribute this phenomenon to a *multiagent autocurriculum* [35], [54] – the pressure of competition incentivizes agents not only to explore (as seen in the first

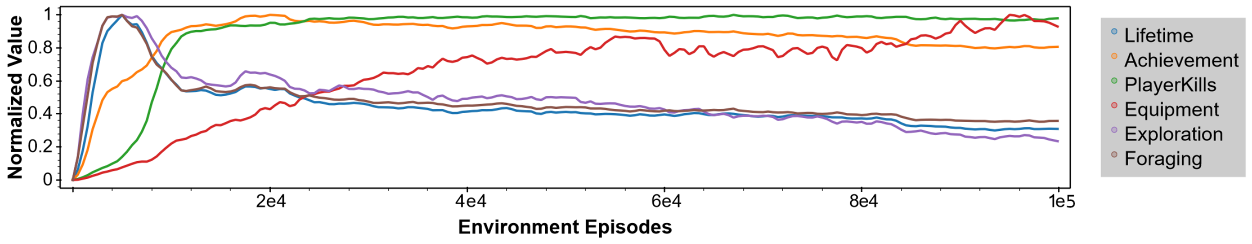


Figure 4.4: Agents trained on small maps only to survive learn a progression of skills: foraging and survival followed by combat with other agents followed by attacking NPCs to acquire equipment.

experiment), but also to leverage the full breadth of available game systems. We believe it is likely that continuing to add more game systems to Neural MMO will result in increased complexity of learned behaviors and enable even more new directions in multiagent research.

Large-Scale Learning: We repeat the experiment above using the canonical LargeMaps setting, which generates 1024x1024 maps and caps populations at 1024 agents. Training using the same reward and a longer horizon of 8192 steps produces a qualitatively similar result. Agents learn to explore, forage, and fight as before, but the policies produced are capable of exploring several hundred tiles into the environment – more than the SmallMaps setting allows. However, the learning dynamics are less stable, and continued training results in policy degradation.

Zero-Shot Transfer: We evaluated the LargeMaps policy zero-shot on the SmallMaps domain. It performs significantly better than random but worse than the agent explicitly trained for this domain. Surprisingly, transferring the SmallMaps model to LargeMaps domain performs better than the LargeMaps model itself but not as well as the scripted baselines.

Metrics: In Table 4.1, Lifetime denotes average survival time in game ticks. Achievement is a holistic measure discussed in the supplementary material. Player kills denotes the average number of other agents defeated. Equipment denotes the level of armor obtained by defeating scripted NPCs. The latter two metrics will always be zero for non-combat models. Explore denotes the average L_1 distance traveled from each agent's spawning location. Finally, Forage is the average skill level associated with resource collection.

Interpreting Results: The scripted combat bot relies upon the same resource gathering logic as the scripted foraging bot and has strictly more capabilities. However, since it is evaluated against copies of itself, which are more capable than the foraging bot, it actually performs worse across several metrics. This dependence upon the quality of other agents is a fundamental difficulty of evaluation in multiagent open-ended settings that we address in the Section 7. Our trained models perform somewhat worse than their scripted counterparts on small maps and significantly worse on large maps. The scripted baselines themselves are not weak, but they are also far from optimal – there is plenty of room for improvement on this task and much more on cooperative tasks upon which, as per our last experiment in Section 5, the same methods perform poorly.

Domain Randomization is an Effective Curriculum: Training on a pool of procedurally generated maps produces policies that generalize better than policies trained on a

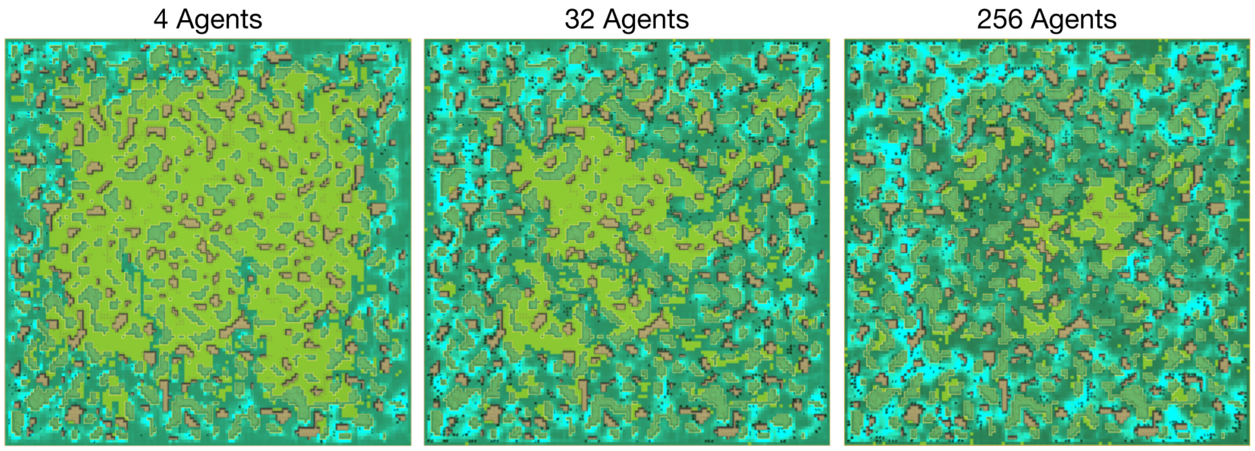


Figure 4.5: Competitive pressure incentivizes agents trained in large populations to learn to explore more of the map in an effort to seek out uncontested resources. Agents spawn at the edges of the map; higher intensity corresponds to more frequent visitation.

single map. Similar results have been observed on the Procgen environment suite and in OpenAI's work on solving Rubik's cube [34], [49]. The authors of the former found that different environments scale up to a different number of maps/levels. We therefore anticipate that it will be useful for Neural MMO users to understand precisely how domain randomization affects performance. We train using 1, 32, 256, and 16384 maps using the canonical 128x128 configuration of Neural MMO with all game systems enabled. Surprisingly, we only observe a significant generalization gap on the model trained with 1 map (Figs. 4.6 and Table 4.2). Note that we neglected to precisely control for training time, but performance is stable for both the 32 and 256 map models by 50k epochs. Interestingly, the 16384 map model exhibits different training dynamics and defeats NPCs to gather equipment three times more than the 32 map model. Lifetime increases early during training as agents learn basic foraging, but it then decreases as agents learn to fight each other.

It may strike the reader as odd that so few maps are required to attain good generalization where the same methods applied to ProcGen require thousands. This could be because every 128x128 map in Neural MMO provides spawning locations all around the edges of the map. Using an agent vision range of 7 tiles, there are around 32 spawning locations with non-overlapping receptive fields (and many more that are only partially overlapping). Still, this is only a total of 1024 unique initial conditions, and we did not evaluate whether similar performance is attainable with 8 or 16 maps. We speculatively attribute the remaining gap between our result and ProcGen's to the sample inefficiency of learning from rendered game frames without vision priors.

Population Size Magnifies Exploration: We first consider a relatively simple configuration of Neural MMO in order to answer a basic question about many-agent learning: how do emergent behaviors differ when agents are trained in populations of various sizes? Enabling only the resource and progression systems with continuous spawning, we train for one day with population caps of 4, 32 and 256 agents. Agents trained in small populations survive only by foraging in the immediate spawn area and exhibit unstable learning dynamics. The competitive pressure induced by limited resources causes agents trained in

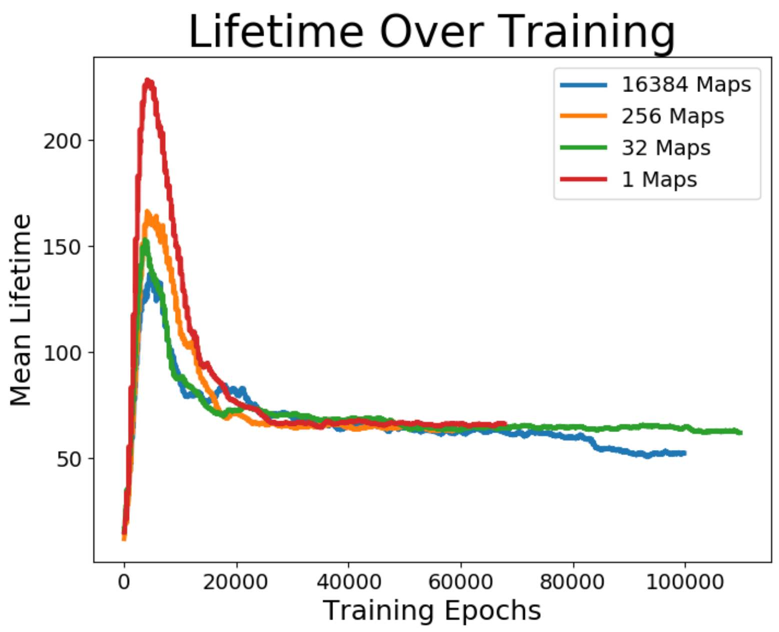


Figure 4.6: Lifetime curves over training on different numbers of procedurally generated maps. Each epoch corresponds to simulating one game map and all associated agents for 1024 timesteps.

Table 4.2: Average lifetime during training on different numbers of maps and and subsequent evaluation on unseen maps.

#Maps	Train	Test	Epochs
1	66.08	52.73	67868
32	61.93	61.56	109868
256	62.67	61.91	58568
16384	52.30	53.25	99868

larger populations to learn more robust foraging and exploration behaviors that cover the whole map (Fig. 4.5). In contrast, as shown by lower average lifetime in Table 4 in the supplementary material, increasing the test-time population cap for agents trained in small populations results in overcrowding and starvation.

Emergent Complexity from Team Play: Multi-population configurations of Neural MMO enable us to study emergent cooperation in many-team play. We enable the resource + combat systems and train populations of 128 concurrently spawned agents with shared rewards across teams of 4. Additionally, we had to disable combat among teammates and add an auxiliary reward for attacking other agents in order to learn anything significant. Under these parameters, agents learn to split into teams of 2 to fight other teams of 2 – not a particularly compelling or sophisticated strategy. Perhaps additional innovations are required in order to learn robust and general cooperation.

Table 4.3: Qualitative summary of related single-agent, multi-agent, and industry-scale environments. **Agents** is listed as the maximum for variable settings. **Horizon** is listed in minutes. **Efficiency** is a qualitative evaluation based on a discussion in the supplementary material.

Environment	Genre	Agents	Horizon	Task(s)	Efficiency	Procedural
NMMO Large	MMO	1024	85	Open-Ended	High	Yes
NMMO Small	MMO	256	10	Open-Ended	High	Yes
ProcGen	Arcade	1	1	Fixed	Medium	Yes
MineRL	Sandbox	1	15	Flexible	Low	Yes
NetHack	Rougelike	1	Long!	Flexible	High	Yes
PommerMan	Arcade	2v2	1	Fixed	High	Yes
MAgent	-	1M	1	Open-Ended	High	Partial
DoTA 2	MOBA	5v5	45	Fixed	Low	No
StarCraft 2	RTS	1v1	20	Fixed	Low	No
Hide & Seek	-	3v3	1	Flexible	Low	Yes
CTF	FPS	3v3	5	Fixed	Low	Yes

4.6 Related Platforms and Environments

Table 4.3 compares environments most relevant to our own work, omiting platforms without canonical tasks. Two nearest neighbors stand out. MAgent [55] has larger agent populations than Neural MMO, NetHack [39] has longer time horizons, and both are more computationally efficient. However, MAgent was designed for simple, short-horizon tasks and NetHack is limited to a single agent. Several other environments feature a few agents and either long time horizons or high computational efficiency, but we are unaware of any featuring large agent populations or open-ended task design

OpenAI Gym: A classic and widely adopted API for environments [56]. It has since been extended for various purposes, including multiagent learning. The original OpenAI Gym release also includes a large suite of simple single-agent environments, including algorithmic and control tasks as well as games from the Atari Learning Environment.

OpenAI Universe: A large collection of Atari games, Flash games, and browser tasks. Unfortunately, many of these tasks were much too complex for algorithms of the time to make any reasonable progress, and the project is no longer supported [33].

Gym Retro: A collection of over 1000 classic console games of varying difficulties. Despite its wider breadth of tasks, this project has been largely superseded by the ProcGen suite, likely because users are required to provide their own game images due to licensing restrictions [32].

ProcGen Benchmark: A collection of 16 single-agent environments with procedurally generated levels that are well suited to studying generalization to new levels [34].

Unity MLAgents: A platform and API for creating single- and multi-agent environments simulated in Unity. It includes a suite of 16 relatively basic environments by default but is sufficiently expressive to define environments as complex as high-budget commercial games. The main problem is the inefficiency associated with simulating game physics during training, which seems to be the intended usage of the platform. This makes MLAgents better suited to control research [40].

Griddly: A platform for building grid-based single- and multi-agent game environments with a fast backend simulator. Like OpenAI Gym, Griddly also includes a suite of environments alongside the core API. It is sufficiently expressive to define a variety of different tasks, including real-time strategy games, but its configuration system does not include a full programming language and is ill-suited as backend for in Neural MMO [43].

MAgent: A platform capable of supporting one million concurrent agents per environment. However, each agent is a simple particle operating in a simple cellular automata-like world. It is well suited to studying collective and aggregate behavior, but has low per-agent complexity [55].

Minecraft: MALMO [57] and the associated MineRL [44] Gym wrapper and human dataset enable reinforcement learning research on Minecraft. Its reliance upon intuitive knowledge of the real world makes it more suitable to learning from demonstrations than from self-play.

DoTA 2: Defense of the Ancients, a top esport with 5v5 round-based games that typically last 20 minutes to an hour. DoTA is arguably the most complex game solved by RL to date. However, the environment is not open source, and solving it required massive compute unavailable to all but the largest industry labs [46].

StarCraft 2 A 1v1 real time strategy esport with round-based games that typically last 15 minutes to an hour. This environment is another of the most complicated to have been solved with reinforcement learning [47] and is actually open source. While the full task is inaccessible outside of large-scale industry research, the StarCraft Multi-Agent Challenge provides a setting for completing various tasks in which, unlike the base game of StarCraft, agents are controlled independently [58].

Hide and Seek: Laser tag mixed with hide and seek played with 2-3 person teams and 1 minute rounds on procedurally generated maps [35].

Capture the Flag: Laser tag mixed with capture the flag played with 2-3 person teams and 5 minute rounds on procedurally generated maps [59].

Pommerman A 4 agent environment playable as free for all or with 2 agent teams. The game mechanics are simple, but this environment is notable for it's centralized tournament evaluation that enables researchers to submit agent policies for placement on a public leaderboard. It is no longer officially supported [42].

NetHack By far the most complex single-agent game environment to date, NetHack is a classic text-based dungeon crawler featuring extended reasoning over extremely long time horizons [39].

Others: Many environments that are not video games have also made strong contributions to the field, including board games like Chess and Go as well as robotics tasks such as the OpenAI Rubik's Cube manipulation task [49] which inspired NeuralMMO's original procedural generation support.

4.7 Limitations and Discussion

Many problems do not require massive scale: Neural MMO supports up to 1024 concurrent agents on large, 1024x1024 maps, but most of our experiments consider only up to 128 agents on 128x128 maps with 1024-step horizons. We do not intend for every research project to use the full-scale version – there are many ideas in multi-agent intelligence research, including those above, that can be tested efficiently at smaller scale (though perhaps not as effectively at the very small scale offered by few-agent environments outside of the Neural MMO platform). The large-scale version of Neural MMO is intended for testing multiple such ideas in conjunction and at scale – as a sort of "realistic" combination of multiple modalities of intelligence. For example, team-based play is interesting in the smaller setting, but what might be possible if agents have time to strategize over multiple hours of play? At the least, they could intentionally train and level their combat skills together, seek out progressively more difficult NPCs to acquire powerful armor, and whittle down other teams by aggressively focusing on agents at the edge of the pack. Developing models and methods capable of doing so is an open problem – we are unaware of any other platform suitable for exploring extended socially-aware reasoning at such scale.

Absence of Absolute Evaluation Metrics: Performance in open-ended multiagent settings is tightly coupled to the actions of other agents. As a result, average reward can decrease even as agents learn better policies. Consider training a population of agents to survive. Learning how to forage results in a large increase to average reward. However, despite making agents strictly more capable, learning additional combat skills can decrease average lifetime by effectively making the task harder – agents now have to contend with hostile potential adversaries. This effect makes interpreting policy quality difficult. We have attempted to mitigate this issue in our experiments by comparing policies according to several metrics instead of a single reward. This can reveal learning progress even when total reward decreases – for example, in Figure 4.4, the Equipment stat continues to increase throughout training. More recently, we have begun to shift our focus towards tournament evaluations that abandons absolute policy quality entirely in favor of a skill rating relative to other agents. This approach has been effective thus far in the competition, but we anticipate that it may not hold for all environment settings as Neural MMO users continue to innovate on the platform. With this in mind, we believe that developing better evaluation tools for open-ended settings will become an important problem as modern reinforcement learning methods continue to solve increasingly complex tasks.

Chapter 5

The IJCAI 2022 Competition

After a small pilot competition on AICrowd in 2021, Neural MMO 1.5 was used for a major competition at IJCAI 2022. The competition defined tasks in four different categories and awarded points based on the number and difficulty of tasks completed. Note that, unlike later works, the tasks were fixed and the submissions were not task-conditional. The level of capability attained by the winning submissions motivated further competitions.

The following paper titled "Benchmarking Robustness and Generalization in Multi-Agent Systems: A Case Study on Neural MMO" was published on arXiv in 2023. It summarizes the results of the IJCAI 2022 competition. This work is coauthored with Yangkun Chen and Junjie Zhang et al. and advised by Phillip Isola. Sections of this paper were invited contributions by the winners of the competition. A shorter version of this paper was published as an extended abstract at AAMAS 2023. I include only the full manuscript because the shorter paper duplicates the same content with many important omissions.

We present the results of the second Neural MMO challenge, hosted at IJCAI 2022, which received 1600+ submissions. This competition targets robustness and generalization in multi-agent systems: participants train teams of agents to complete a multi-task objective against opponents not seen during training. The competition combines relatively complex environment design with large numbers of agents in the environment. The top submissions demonstrate strong success on this task using mostly standard reinforcement learning (RL) methods combined with domain-specific engineering. We summarize the competition design and results and suggest that, as an academic community, competitions may be a powerful approach to solving hard problems and establishing a solid benchmark for algorithms. We open-source our benchmark including the environment wrapper, baselines, a visualization tool, and selected policies for further research.

5.1 Introduction

Real-world applications of reinforcement learning (RL) require robust algorithms [60] that can adapt to dynamic environments. While substantially studied in single-agent RL [61], [62], this subject has been less explored in multi-agent systems. This is of particular importance to multi-agent RL (MARL) algorithms because learned policies must adapt to changes in other agents' behaviors in addition to changes in the environment. Here we suggest three difficulties for establishing benchmarks in multi-agent systems that should resonate with MARL researchers:

1. **Lack of environments:** while there are many single-agent environments of varying complexities that are standard, efficient, and simple to use, few multi-agent environments satisfy all three of these properties.

2. **Lack of infrastructure:** most RL libraries and interfaces are intended for single-agent systems, but multi-agent training requires scalability, flexibility, and other additional features. For example, an accurate skill-rating system is needed for multi-agent evaluation as performance is relative to other agents.

3. **Lack of domain-specific optimization:** minor implementation details and domain-specific tricks like feature engineering often highly influence the final performance of RL algorithms [63]. Although these techniques are not the focus of academic research, without them, it is hard to identify the roots of progress and benchmark algorithms fairly.

This paper summarizes the IJCAI 2022 Neural MMO challenge and offers a solution to these three problems. Neural MMO is a good environment to start with because it supports large-scale populations, is computationally efficient and is actively maintained. In addition to the environment, we built a large-scale parallel evaluation tool and a TrueSkill[64] rating system on the AICrowd platform as the infrastructure. The competition among participants provides an inherent incentive for domain-specific optimization, which is often overlooked in academic research. We hope that our methodology can serve as a stepping stone towards establishing more general benchmarks and promoting future research in Neural MMO and other multi-agent systems. Our main contributions are:

1. **Orchestration:** we detail the structure of our competition, including the environment, resources, the design of tracks, and the evaluation system. While RL competitions are gaining popularity, few resources exist on how to design a good competition. We believe this will be useful to guide future RL competitions.

2. **Insights:** we analyze the emergent behaviors and strategies over the 1600+ submissions received and provide insights about the dynamics of the unique multi-agent system consisting of different participants. For example, we find an interesting arms race between rule-based methods and learning-based methods.

3. **Policy Pool:** we release a pool of 20 submitted policies to promote future research on Neural MMO. The policy pool is diverse, containing both rule-based and RL-based implementations of aggressive and conservative strategies. This will be useful in evaluating policy robustness against a variety of opponents.

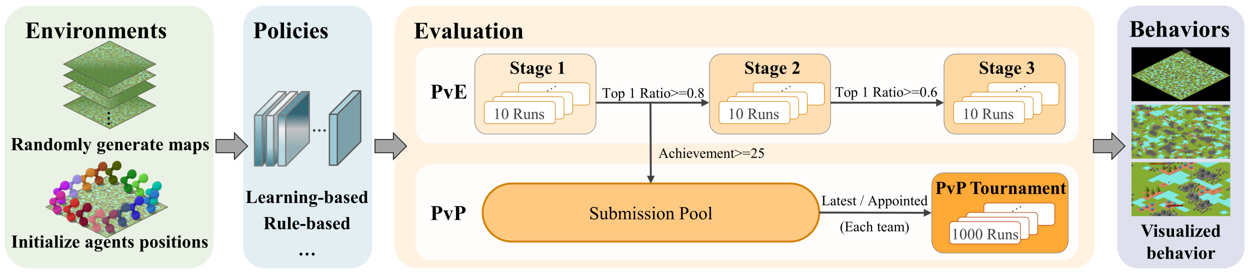


Figure 5.1: Evaluation structure of the competition. Submitted policies are evaluated in two stages. In the PvE track, there are multiple qualifying rounds against increasingly difficult built-in opponents, and participants receive feedback within minutes of submitting. The PvP track features weekly tournaments to determine the relative skill of all qualified submissions.

5.2 Related Works

5.2.1 Environments and Benchmarks

In recent years, wrapping existing games as environments has been a popular approach to increase complexity and promote novel algorithms. MineRL[65] uses Minecraft to highlight hard problems such as hierarchical task structure and sparse rewards. The NetHack Learning Environment [66] provides a rich and challenging environment focused on the problems of exploration and skill acquisition while allowing fast simulation. ProcGen [61] uses 16 procedurally-generated gym environments and is designed to benchmark both sample efficiency and generalization in reinforcement learning.

Among multi-agent environments, the most commonly used is the StarCraft Multi-agent Challenge (SMAC) [67] from the game StarCraft2. SMAC is mainly intended to investigate algorithms for multi-agent cooperation. Google Research Football (gfootball) [68] uses a physics-based 3D football environment in multi-player and multi-agent scenarios, proposed to benchmark algorithms on the sparse reward and multi-agent cooperation. Neural MMO [69] proposes an open-ended Massively Multiplayer Online (MMO) environment with up to 1024 agents to study robustness and teamwork in a massive-agent environment. Agar.io [70] and the similar GoBigger [71] uses the popular online multi-player game Agar.io[1] for multi-agent cooperation and competition. We use Neural MMO as our competition environment because it supports large-scale population simulation with up to 16 teams in one environment.

5.2.2 RL Competitions

Several works [72] attempt to establish solid benchmarks for deep RL algorithms. A key issue in doing so is that performance highly depends on minute implementation details [63], which are often not the focus of academic research. One alternative is an open competition that provides a natural incentive for domain-specific optimization: winning. This format has become popular in recent years, and existing competitions can be roughly categorized into three classes: PvE, 1v1, and FFA.

PvE (Player vs Environment): these competitions evaluate agents against preset (usually randomized) environments as specific algorithmic benchmarks. For example, the NetHack challenge [73] concentrates on sparse reward and exploration while the MineRL competition [74] concentrates on sample efficiency. However, in PvE settings, agents are evaluated against given environments or fixed bots instead of other learning agents. This evaluation paradigm limits the ability to benchmark robustness and generalizability to new opponents.

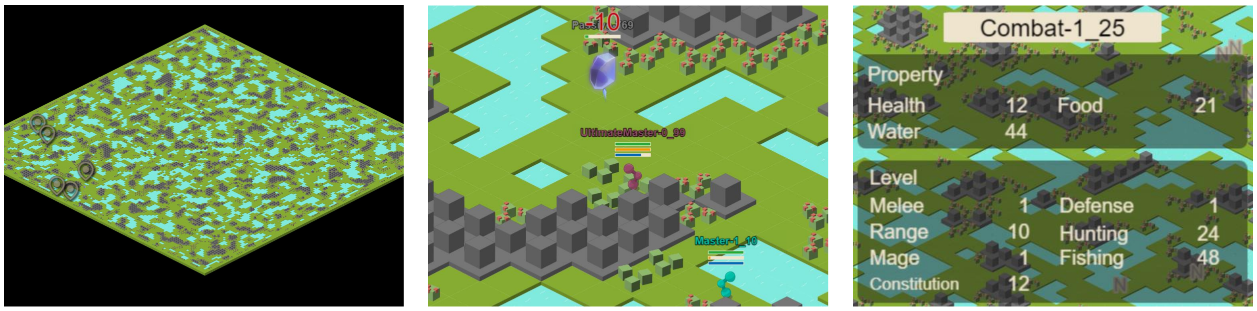
1v1 or one team vs another: these competitions evaluate agents against other participants' policies in a two-agent or two-team mode, such as Google Research Football [68] and Lux-AI [75]. This requires agents to adapt to different kinds of opponents instead of specializing in a fixed environment.

FFA (Free-for-All): this setting places many independent agents or many teams in the same shared environment. Compared to the 2-team mode, FFA competitions can create a vast space of cyclic, non-transitive strategies and counter-strategies because of the combinatorial complexity and the dynamic relationships among agents. To our knowledge, the first Neural MMO challenge in 2021 [2] is the first competition that supports this FFA mode. In the competition, 16 participants' policies are evaluated together on the same map to benchmark their robustness and generalization. Our competition also follows this setting.

To achieve accurate evaluation and get more participants involved, we set up two tracks: PvE and PvP. In the PvE track, submitted policies are confronted with different levels of preset AIs, which can be seen as a fixed environment. This PvE setting reduces the uncertainty of the evaluation process and helps participants identify potential improvements. In the PvP track, we adopt the FFA setting as it can better benchmark the policy's robustness and also provides a persistent incentive for participants to improve their policies. Our competition is the first RL competition with this dual-track system, which was well received by our participants.

[1] https://agar.io/

[2] https://www.aicrowd.com/challenges/the-neural-mmo-challenge



(a) Overall: shows overall team position and resource distribution.

(b) Close-up: shows important local details such as individual fights.

(c) Details: shows the current numeric properties of the chosen agent.

Figure 5.2: Web Viewer: a light visualization tool to show episode replays. (a) (b) (c) are three levels of view that can be altered during the playback. Users can rewind, pause, and change the playback speed on this web page. This allows participants to better understand the game and thus debug.

5.3 Competition Orchestration

5.3.1 Environment

Introduction to Neural MMO

Neural MMO is an open-source research platform that simulates populations of agents in procedurally generated virtual worlds. It is inspired by classic massively multiagent online role-playing games (MMORPGs or MMOs for short) as settings where lots of players using entirely different strategies interact in interesting ways. Unlike other game genres typically used in research, MMOs simulate persistent worlds that support rich player interactions and a wider variety of progression strategies. We refer the reader to the original publication [76] for full information on Neural MMO and its objectives. Our environment is adapted from version 1.5 of Neural MMO.

Competition Configuration

The competition configuration of Neural MMO places 128 agents in procedurally generated maps. Each map is 128x128 tiles. Scripted non-playable characters (NPCs) are spawned across the map. Agents must collect resources, *Food* and *Water* to survive and can attack each other and NPCs using three combat styles with strategic tradeoffs: *Melee*, *Mage*, and *Range*. The competition focuses on robustness to new maps and new opponents and the team design introduces cooperation and specialization to different roles on top of this.

Environment Wrapper

To make the environment work with different agents, i.e., rule-based and RL agents, we wrap Neural MMO with two major changes. First, agents are spawned uniformly at the edges of the map. We randomize both the map seed and the initial position of each team across episodes to ensure a fair evaluation. Second, the observations of 8 agents in a team are grouped and made available to a single policy when they make decisions. A more strict setting in multi-agent cooperation might require that each agent compute its actions independently from teammates. We loosen that limit as in OpenAI Five [77] and AlphaStar[78] in favor of enabling higher overall policy quality.

5.3.2 Resources

For the participants' convenience, we have created a number of resources:

- **Starter Kit**[3]: a project containing all required segments to make a successful submission. With this guidance, new participants can make their first submission within 15 minutes.
- **Baseline**[4]: an RL baseline implementation in a single file based on *TorchBeast* [79]. This provides RL researchers with a fundamental baseline to start with.
- **Env Docs**[5]: Documents and tutorials to help participants to get familiar with Neural MMO.
- **Web Viewer**[6]: A light web replay viewer for our challenge, which allows participants with visual straightforward feedback for their policy development.

Web Viewer

The web viewer is a light visualization tool to show the replays of the episodes, allowing our participants to review their policy's performance. For RL researchers, an accessible viewer is crucial to analyze learned strategies and improve their policies. The web viewer UI contains three levels of view:

1. An overall view, as shown in Fig.5.2a, demonstrating the whole team's trajectories and the resource distribution of the global map;
2. A close-up view, as shown in Fig. 5.2b, which reveals local details such as individual fights, including the attack target and attack style of each agent;
3. A view of numeric details, as shown in Fig. 5.2c, which shows the current numeric properties of the chosen agent, such as its skill levels, current health, and collected resources.

The right side of the interface shows the achievement scores of each team, allowing participants to interpret the varying abilities of each team to complete the four subtasks.

[3]https://gitlab.aicrowd.com/neural-mmo/ijcai2022-nmmo-starter-kit
[4]https://gitlab.aicrowd.com/neural-mmo/ijcai2022-nmmo-baselines
[5]https://neuralmmo.github.io/build/html/rst/landing.html
[6]https://ijcai2022-viewer.nmmo.org/

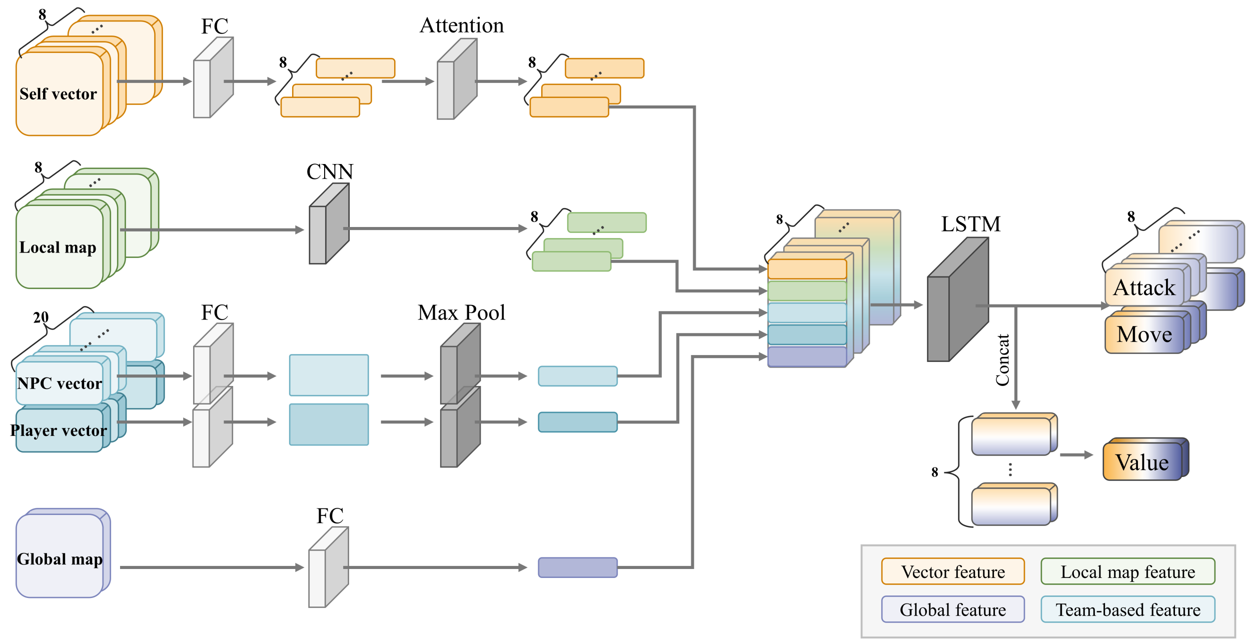


Figure 5.3: Model architecture for the PvE stage 3 baseline. It includes a core LSTM, a domain-specific observation encoder with sub-networks for flat, map, and set data, and a domain-specific decoder for fixed and variable-length actions.

5.3.3 Competition Structure

The competition consists of two tracks: the PvE track and the PvP track. For clarity, PvE refers to one participant's policy vs. 15 built-in policies provided by the organizers. The PvE track serves as a fixed reference to help participants develop their policies. The PvE track contains 3 stages with different built-in policies and increasing difficulties. In the main PvP track, 16 participants' policies are thrown into shared environments. This can better test a policy's robustness and generalization to opponents not seen during training.

5.3.4 PvE (vs. fixed baselines)

The main PvP track enables players to test the robustness and generalization of their agents against a variety of foes. However, there is a high degree of uncertainty: the quality of opponents changes over time, and the only measure of policy performance is relative to that of all other submissions. We have thus set up an additional PvE track of 3 stages with 3 main purposes: (1) To help participants identify their agents' current performance against a fixed set of opponents of increasing quality; (2) To further incentivize participation by providing three reasonably achievable milestones; (3) To help participants understand the environment as guidance from easy to hard.

PvE Stage 1 (vs. Rule-Based Scripted Baselines)

PvE stage 1 is a start-up stage arranged to help our participants familiarize themselves with the environment. We use three rule-based AIs named *Combat*, *Forage*, and *Random*. The

Table 5.1: Tasks defined in this competition to measure the performance of policies. Teams earn 0, 4, 10, or 21 points per task, depending on the hardest difficulty of that task completed by at least one agent in the team. *Achievement* is defined as the sum of this score over all four tasks.

Task	Easy(4 points)	Medium(10 points)	Hard(21 points)
Travel the Lands	32 Meters	64 Meters	127 Meters
Forage for Resources	Skill Lvl 20	Skill Lvl 35	Skill Lvl 50
Secure an Advantage	Lvl 1 Equipment	Lvl 10 Equipment	Lvl 20 Equipment
Eliminate the Competition	Defeat 1 Player	Defeat 3 Players	Defeat 6 Players

Combat policy is hostile and will attack nearby agents. The *Forage* policy focuses only on collecting resources and will attempt to flee from combat. The *Random* policy moves randomly and is intended as a basic sanity check. Considering that all of these policies are open-source, this stage is intended to be relatively easy to beat.

PvE Stage 2 (vs. RL Baselines)

In PvE stage 2, we trained agents with PPO [80] using well-designed features extracted from raw observations. It is worth noting that we applied a *decentralized training* method, which means each agent in our team can only get its own observations and act individually. This provides a medium-level reference for participants, which is much harder than that in Stage 1 but only takes RL agents about one week to conquer.

PvE Stage 3 (vs. Team-Based RL Baselines)

The PvE stage 3 AI has the highest performance over all the 4 sub-tasks and can prevail over the PvE stage 2 baseline. The key distinction here is that we adopt a team-based method to process the whole team's observation and compute actions jointly. To be more specific, we devised a *centralized training* strategy in which one network would concurrently process all 8 agents' observations and output the actions of eight agents. A team-based achievement is used as a reward. The advantage of this approach is that information is shared explicitly among teammates. Details of our modeling solution are as follows.

Feature Design. At the current scale, specialized feature extraction yields a large performance increase over processing raw observations. We divide, featurize, and process observations from all 8 agents on the team as follows:

- **Member features**: Each team member's self-information such as their IDs (to better correlate to the index of the following local map feature), the initial positions (to measure the derivation of the current position), etc.

- **Enemy & NPC features**: The observed entities' key features such as HP, level, and types are embedded here. This information helps agents learn how to behave in the presence of potential adversaries. We aggregate all 8 agents' observed entities together and use an additional vector of each entity to identify which agent is observing this entity to encourage the team-based policy to attack cooperatively;

- **Local map features**: Spatial information, such as the resource distribution on the observed local map, is embedded here to help agents learn to pathfind;
- **Global features:** Key global information such as time elapsed or the number of still-living teammates.

Policy Architecture and Training: The policy has three subnetworks: an observation encoder, the main long short-term memory (LSTM) network, and the action decoder. This architecture is shown in Fig.5.3. The input network contains fully-connected (FC) layers and max-pooling to process all scalar features and global information. Convolutional networks are used to process spatial information and attention modules are used to process position-invariant entity data. The main LSTM network processes the aggregate output of all of these encoders. The action output layers are normal FC layers. The policy is trained in a self-play setup against 15 teams controlled by the same policy. We employ valid action masks to accelerate exploration.

Reward Design: The competition scores teams based on their combat, foraging, and exploration. This mechanism is described in Section 5.4 below; the relevant aspect here is that we use this function directly in order to compute rewards.

5.3.5 PvP (vs. other participants)

Participants must pass the qualifying PvE Stage 1 in order to compete in PvP. Unlike in the PvE stages where opponents are fixed, the PvP opponent pool is dynamically sampled from the latest qualifying submissions from other participants. This includes reinforcement learned, scripted, and hybrid submissions.

We saw a large number of strategies emerge throughout the PvP stages, increasing our confidence in the platform as a proving ground for multi-agent reinforcement learning research, especially in testing the robustness of algorithms to new maps and opponents. We've noticed that some participants can rank high on PvE stage 1 or stage 2 but cannot maintain their advantage on the PvP stage, which indicates overfitting to the training domain.

5.4 Evaluation System

Each participant's policy will control a team of 8 agents and will be evaluated in a free-for-all against 15 other teams on 128x128 maps. After 1024 environment steps, the team with the highest *Achievement* wins.

5.4.1 Metrics

Multi-Task Metrics Definition

To evaluate the generalization of the policy, we design a suite of 4 tasks, as shown in Table 5.1. Each task has 3 difficulty levels: 4 points for easy, 10 points for normal, and 21 points

Table 5.2: Comparison of major RL competitions on the AICrowd platform, the primary venue for these events. Our competition has the most unique submitters and the highest sign-up-to-submission conversion rate.

Competition	Views	Users	Submitted	Entry Rate
IJCAI 2022: Neural MMO Competition[8]	40.3k	540	**111**	**20.50%**
NeurIPS 2021: MineRL BASALT [9]	38.8k	353	17	4.0%
NeurIPS 2021: MineRL Diamond [10]	35.8k	511	65	12.7%
NeurIPS 2019: MineRL Competition[11]	69.3k	1124	41	3.6%
NeurIPS 2021 - NetHack Challenge[12]	49.1k	584	46	7.9%
NeurIPS 2020 Procgen Competition[13]	52.8k	711	85	12.0%
Flatland 3[14]	19.2k	328	24	7.3%
Flatland[15]	72k	1090	65	6.0%
Unity Obstacle Tower Challenge[16]	74k	637	95	14.9%
NeurIPS 2019: Learn to Move [17]	37.9k	364	71	19.5%
NeurIPS 2021 AWS DeepRacer[18]	20.4k	337	41	12.2%
Learn-to-Race: Autonomous Racing[19]	24.1k	476	53	11.1%

for hard. Points were only awarded for the highest tier task completed in each category. The team with the most points at the end of a game (1024 steps) wins. This is a multi-objective task intended to be completed as a team: to achieve the maximum score for a task, only one agent on the team needs to complete it. This means it is reasonable for different agents on the team to employ different strategies. Such design encourages cooperation as a team and specialization of individual players.

TrueSkill in PvP

For the PvP track evaluation procedure, we randomly select 16 submissions from all qualified submissions to start a PvP match. In the final evaluation, each submission will participate in approximately 1000 matches. The mean achievement score is not a good evaluation metric due to the variability of opponents. For example, model A gets a high score against weaker opponents and a low score against stronger opponents, while model B gets an above-average score against all levels of opponents. In this case, the mean achievement scores of the two models may be close, but it is obvious that model B is more robust. To more accurately measure the relative strength of the models, we use TrueSkill [64] to compute scores for each submission.

Top 1 Ratio in PvE

In the PvE track, the participant's model will play 10 matches against our built-in AI. With 16 teams per match, the variance of the mean achievement score for 10 matches is high. To evaluate the robustness of the policy, we use Top1Ratio as the evaluation metric. The Top1Ratio is the ratio of games won (i.e. highest score among all teams) over 10 matches. A Top1Ratio close to 1.0 indicates that the model is significantly stronger than the 15 built-in teams.

5.4.2 Implementation

We developed the distributed evaluation system shown in Fig. 5.1 to quickly process submissions at scale. It can roll out hundreds of matches in parallel using k8s clusters and can return results within 10 minutes of submission.

We use the same distributed evaluation system for both the PvE and PvP tracks. The PvE track contains three levels; the main difference between them is the strength of the built-in AIs. Participants can enter the next level by reaching the specified Top1Ratio in the previous level. Reaching 25 points in PvE stage 1 qualifies a submission for the PvP track against other user submissions. Between PvE and PvP evaluation, we can accurately measure the strength of all models relative both to each other and to fixed baselines. The PvP evaluation is run once per week while PvE evaluation proceeds immediately upon submission. This ensures fast feedback to inform development at all times and more extensive feedback weekly.

5.5 Summary and Analysis

5.5.1 Summary of the competition

The competition received over 40k views, 537 individual signups, 110 team signups, and 1679 submissions. This makes it one of the largest RL competitions to date, outpacing all of the MineRL competitions thus far and Nethack – despite having a significantly higher barrier to entry due to the complexity of the task, lack of a single-agent track, lack of offline data, and complex observation and action representation. Of these participants, 48 teams were able to pass our first-round qualifier. 20 teams were able to win at least some games versus better policies that we trained for round 2, with 16 qualifying for round 3. We trained much stronger baselines for these rounds, but 7 teams were still able to win at least some games, and 6 were convincingly better than our best baseline. The best policies fully accomplished the task of the competition. Table 5.2 compares the metrics of major RL competitions and demonstrates the scope of our contest.

[8] https://www.aicrowd.com/challenges/ijcai-2022-the-neural-mmo-challenge
[9] https://www.aicrowd.com/challenges/neurips-2022-minerl-basalt-competition
[10] https://www.aicrowd.com/challenges/neurips-2021-minerl-diamond-competition
[11] https://www.aicrowd.com/challenges/neurips-2019-minerl-competition
[12] https://www.aicrowd.com/challenges/neurips-2021-the-nethack-challenge
[13] https://www.aicrowd.com/challenges/neurips-2020-procgen-competition
[14] https://www.aicrowd.com/challenges/flatland-3
[15] https://www.aicrowd.com/challenges/flatland
[16] https://www.aicrowd.com/challenges/unity-obstacle-tower-challenge
[17] https://www.aicrowd.com/challenges/neurips-2019-learn-to-move-walk-around
[18] https://www.aicrowd.com/challenges/neurips-2021-aws-deepracer-ai-driving-olympics-challenge/
[19] https://www.aicrowd.com/challenges/learn-to-race-autonomous-racing-virtual-challenge

5.5.2 Analysis of the Competition Design

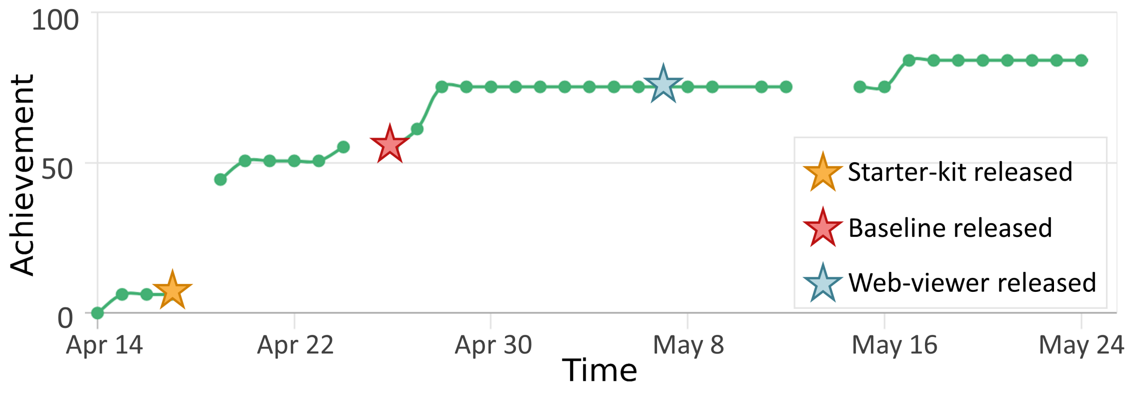


Figure 5.4: Maximum achievement in PvE stage 1 through time, measured over all participants. The release of the baseline corresponds with a large jump in submission quality.

Fig.5.4 shows the increases of max achievement over time. We can find that the three sudden rises are due to the starter kit release, the official baseline release, and the web viewer release: these tools were either useful or at least motivational to participants.

Fig.5.5 shows performance against different stages of the PvE track. The baseline quality increases across rounds, so submission performance declines as expected from stage to stage. Interestingly, the final PvE stage results are similar to those of the final PvP track. This suggests that this track was effective in allowing participants to quickly evaluate their submissions. This is useful because the PvP stage is more computationally expensive, so we can only run it once per week.

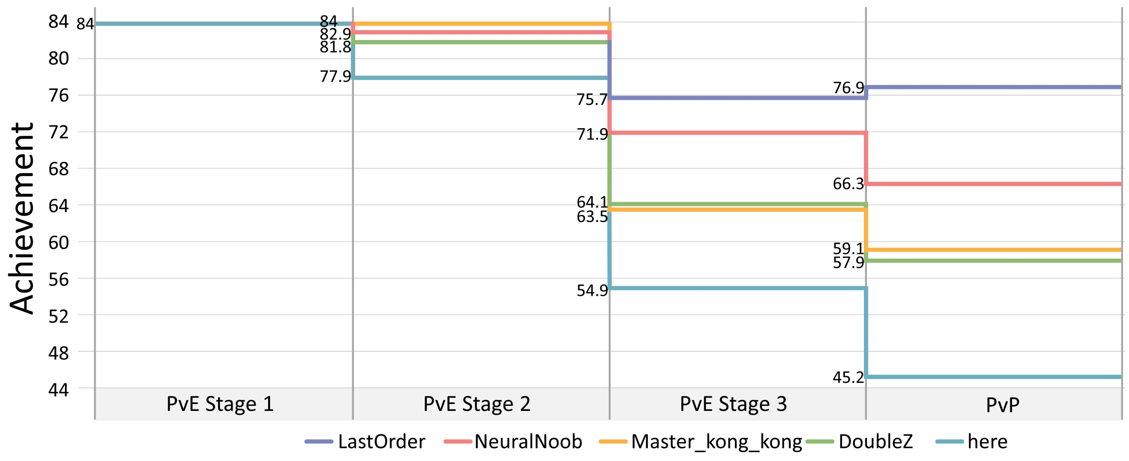


Figure 5.5: Top five participant scores in each round. The increasing difficulty of later PvE rounds corresponds with a decline in achievement score. Performance in stage 3 is comparable to performance in the last PvP stage.

5.5.3 Analysis of 1600+ Policies

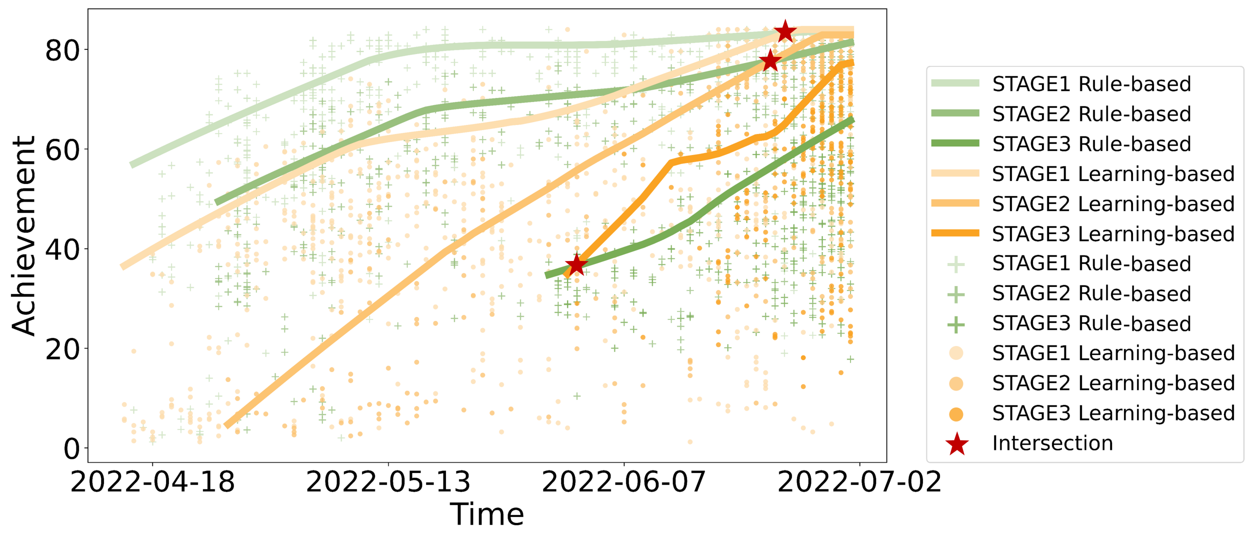


Figure 5.6: The effectiveness of Rule-Based and Learning-Based methods across three PvE stages. The six lines represent the peak performance of the approach at each stage. The red star marks the point at which the highest performance of the two approaches overlaps.

We gathered over 1600 submissions and categorized them as rule-based methods (behavior tree, planning-based methods, heuristic methods, etc.) or learning-based methods (reinforcement learning) based on the algorithms employed by the participants. **We additionally release presentations from the top 5 teams about their approach (camera ready for anonymity).**

Fig.5.6 illustrates the best achievements over time for both classes of methodologies in the three stages of PvE and PvP track. We make several observations. (1) Rule-based or

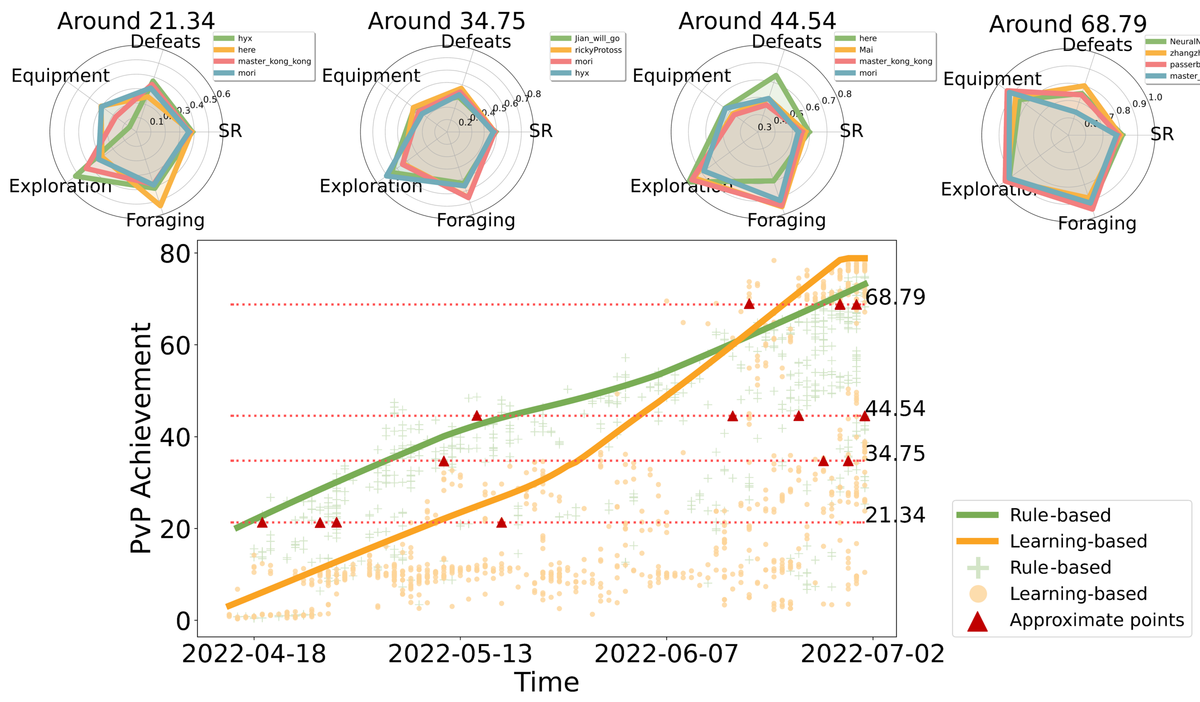


Figure 5.7: We compared the performance of players with the same achievement on each of the four subtasks. Even if the final achievements are identical, participants will employ different methods to accomplish the job, demonstrating that Neural MMO can accommodate a variety of tactics.

learning-based methods both achieve satisfactory performance. (2) The performance curves of rule-based and learning methods cross earlier in later stages. This suggests that rule-based methods are quick to get working but do not scale as well against complex opponents. (3) The green lines of rule-based methods in Stage 1 and Stage 2 almost converge and still climb in Stage 3. The orange lines of learning-based methods are all climbing. Thus, there is still room for further research even on this version of Neural MMO, without even considering some of the more recent additions to the environment.

Robustness and Generalization.

As seen in the Fig.5.5, the performance of the participants' models varies at different stages. As an example, plotted the exploration pattern of the winning policy *passersby*, against a weaker opponent (PvE stage 2) and against a stronger opponent (PvE stage 3). This player's model explores significantly more against inferior opponents than against stronger ones, which is shown in Fig.5.10. This further demonstrates that this environment may facilitate the study of model robustness and generalization by introducing diverse adversaries.

Diversity of Policies.

We find that policies that achieve the same score may employ different strategies, as indicated by differing performance on the four subtasks and the overall trajectories of different agents. The models of different players can have varied strengths and weaknesses on the sub-tasks as shown in Fig.5.7. Using the four players with final achievements close to 68.79 as an example, *zhangzhang*'s model achievement on *Defeat* is high, but their *Equipment* is inadequate, indicating that their agents' primary tactic is to kill other players' agents. The superior performance of *kongkong*'s agents in both *Foraging* and *Equipment* suggests that their strategy is more adept at utilizing map resources.

For the trajectories shown in Fig.5.9, we choose the paths of the top five ranking submissions and observe that various teams have distinct navigation preferences. The team *here*, for instance, will explore in a straight line to maximize their exploration score, whereas the team *DoubleZ* will go deeper into the heart of the map from the beginning because there are higher-level NPCs there, allowing them to quickly upgrade their equipment. *Master_kong_kong*'s team will do the most comprehensive exploration, allowing them to become familiar with the entire area more quickly. Similarly, we count the frequency of visits to each tile by the agents under different strategies, and we can find that there will be differences among models as shown in Fig.5.8.

5.6 Conclusion

To benchmark the robustness and generalization of MARL algorithms, we hosted a multi-agent artificial intelligence challenge and received 1600+ policy submissions. The top five submissions all surpassed the best existing baselines while employing strategies ranging from rule-based to full RL. However, the performance curve till the end of the competition indicates that policies have not yet reached the performance upper bound in the environment and that there is still considerable potential for RL algorithms in further research.

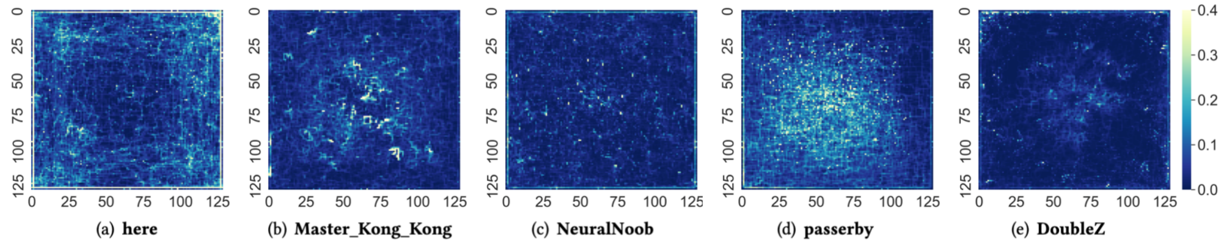


Figure 5.8: Visitation frequency of various policies, computed by summing per-tile exploration counts over over 50 episodes. Different policies demonstrate distinct exploration preferences. For example, the *here* policy primarily explores around the edges of the map while *passerby* spends more time in the center of the map.

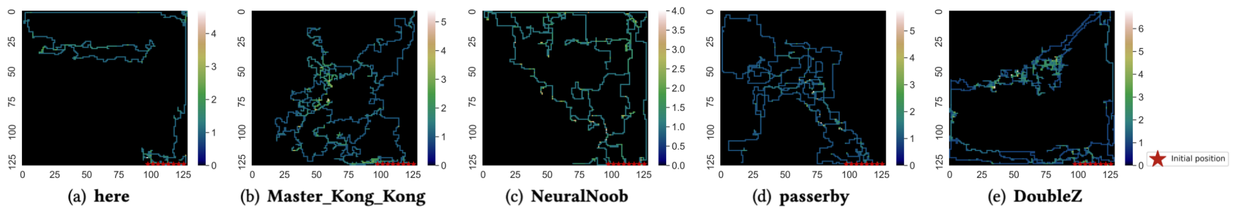


Figure 5.9: Movement path of five teams. Different policies employ different pathing strategies, with some choosing to disperse at the start and converge at the center while others explore as a team.

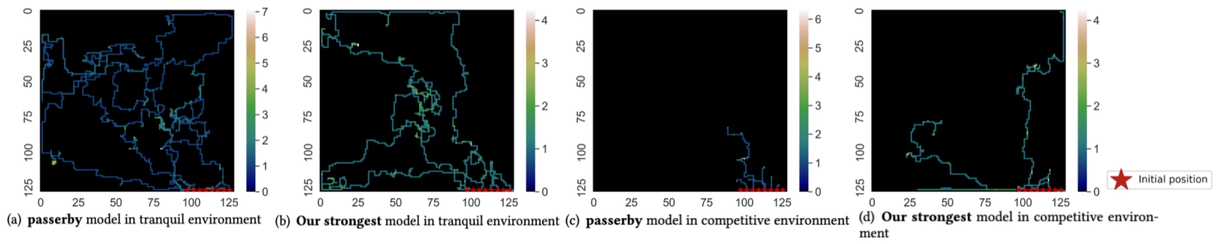


Figure 5.10: The figure depicts the movement trajectory of the participant model against weak (PvE stage 1) and strong (PvE stage 3) opponents. We have compared our strongest model in the same context. **Passerby** employs different pathing strategies against different opponents, demonstrating that our competition may be utilized to evaluate policy robustness.

From an algorithmic perspective, the results of this competition and the analysis of the top-ranking solutions demonstrate that the conceptually simple methods effective in large-scale industry research can also work on complex but academic-scale tasks. We suggest that a gap in tooling and infrastructure, rather than purely algorithms, is the main short-term bottleneck preventing reinforcement learning from working on complex, multi-agent environments. We argue that the simplest way to realize this result in other environments is to run competitions and open-source the tools built by organizers and participants. By aggregating these implementations across multiple domains, we may begin to see the commonalities and build more general-purpose tools. We hope that our work will inspire others to adopt the competition model of research and open-source their tooling as we have.

Chapter 6

Neural MMO 1.6: Specialization and Exchange

Chapters 6-7 cover Neural MMO 1.6, which expands the environment to include several trainable professions, a vast item system, and the ability to trade with other agents, among other core features. I started developing this update before launching any of the competitions, and this is the last update that is fully consistent with Neural MMO's original trajectory. After 1.6, my focus shifted to enabling better competitions on Neural MMO, rather than building out more complexity.

The following short paper titled "Specialization and Exchange in Neural MMO" was published at the ICLR 2022 workshop on Learning in Artificial Open Worlds. This work was advised by Phillip Isola with no other co-authors. It is my favorite paper, even though it was never published as a full-length manuscript and does not include trained models. The results hint at what may be learnable on the later versions of Neural MMO without the more restrictive requirements of a competition.

We present a simulated profession and exchange system for use in multi-agent intelligence research. Each of the eight implemented jobs produces items required by other professions. As a result, each profession must purchase items that they cannot produce themselves from other professions. These items are then used to produce increasingly high-quality goods for resale on a global market. Better and better goods enter the market as trade among professions creates a feedback loop. We integrate our profession and exchange system with Neural MMO, an existing multi-agent reinforcement learning platform capable of efficiently simulating populations of tens to 1000+ agents. We hope that our work will help support new research on *emergent specialization* — the ability to select and commit to a specific long-term strategy that fills a niche left by other learning agents. All of our code, including scripted baseline agents for each profession, will be free, open-source, and actively maintained.

6.1 Introduction

Specialization is a hallmark of human intelligence and achievement. Primitive societies featured distinct hunter and gatherer roles. Later, more specialized tradesman emerged, with different goods and services requiring different expertise. The modern world is so tremendously specialized that no single individual could produce just about any common household object from scratch. Milton Friedman famously described this phenomenon as applied to a pencil: the wood, the saws used to cut the wood, the metal for those saws, the equipment to mine and process that metal, the graphite, the process for compacting it into a pencil, the rubber eraser and glue, etc. Thousands of people minimum are involved in these processes, and yet a pencil is both more effective and cheaper than older implements such as a quill and ink.

Societies of specialized individuals possess a great potential for invention and efficiency. Most or all modern multi-agent learning environments do not incentivize specialization in the way the real world does – we cannot expect specialization to simply emerge if, from an agent's perspective, there is no benefit to doing so. Our work introduces mechanics that enable specialization in simulated environments. We do not claim to accurately mimic human specialization structures – rather, our intent is to at least enable some form of learnable and quantifiable specialization for study.

6.2 Related Work

Multiagent specialization is a common topic in early artificial life research [81]. The problem has been studied more recently in reinforcement learning contexts. For example, [82] explore the problem in the context of multi-robot self-organization while [83] explore per-policy entropy regularization as a mechanism to enable convergence to different specializations. [84] even represent specializations implicitly in latent space. The precise history of algorithmic innovation in this area is outside the scope of the present work, as we do not build upon it. Rather, we aim to create better environments for the study of this problem. Excepting the below, previous works are limited to either physics simulations or simple one-off environments. Physics simulators are important in robotics and control, but this work bets against them for broader emergent behavior research on the basis of inefficiency and difficulty to develop. To our knowledge, there are presently no physics-based environments with support for higher-level mechanics such as progression and exchange (see Environment).

Related environments include the AI Economist [85] and the recent [86]. Neither of these environments include progression systems that enable experienced agents to collect or produce better goods. Different agents in LuxAI can collect different resources, but that is where support for specialization ends. AI Economist implements a basic resource and exchange system for the purpose of studying tax policy. It assumes agents are differentiated by their skills upon creation and has limited potential for emergent specialization.

This work required a simulation platform with support for adding new game systems to existing environments. The only reasonable choice was Neural MMO [87], an open-source and computationally accessible research platform that simulates populations of agents in procedurally generated virtual worlds. Our work builds on this platform by adding new profession and exchange systems. Crucially, we do so in a way that is straightforward to integrate with modern reinforcement learning frameworks and maintains the efficiency of the base platform. We refer to our work as an "environment" for simplicity, but really, Neural MMO is a much larger platform capable of simulating a wide variety of environments. Our profession and exchange systems hook into the platform's configuration system to provide these features in all such environments.

6.3 Environment

Maps are procedurally generated with seven different types of resource. Two of these resources, food and water, are required by all agents to survive. The other five resources are gathered using the associated profession. Figure 1 shows a map with generated terrain and resources. Any agent can harvest any resource, but agents trained in the corresponding profession will obtain better items. Of these five resources, two are used to make consumable items that restore food and water or health. The other three are used as ammunition in the three available combat styles. Combat is used to fight both other agents and scripted non-player characters (NPCs) which drop gold, equipment, weapons, and tools upon defeat. Equipment is useful to all agents as a means of defense. Each weapon type corresponds to a specific combat skill. Each tool type corresponds to a specific gathering skill. Agents can trade consumables, ammunition, weapons, tools, and equipment on a global market using gold. Figures 2 and 3 show the relationships between professions and items. Below, we describe each of these systems. **All of the constants used below are configurable**, but we provide the defaults to give a sense of scale.

Survival: Agents start with 100 food, water, and health. They lose 5 food and water per time step and begin losing health if either hits 0. Agents die at 0 health. Health slowly regenerates if agents have over 50 food and water. Agents restore water by moving adjacent to a water tile and restore food by stepping on a food tile. Food tiles decay once harvested and replenish slowly over time.

Combat: There are 3 combat profession: melee, range, and mage. These use rock-

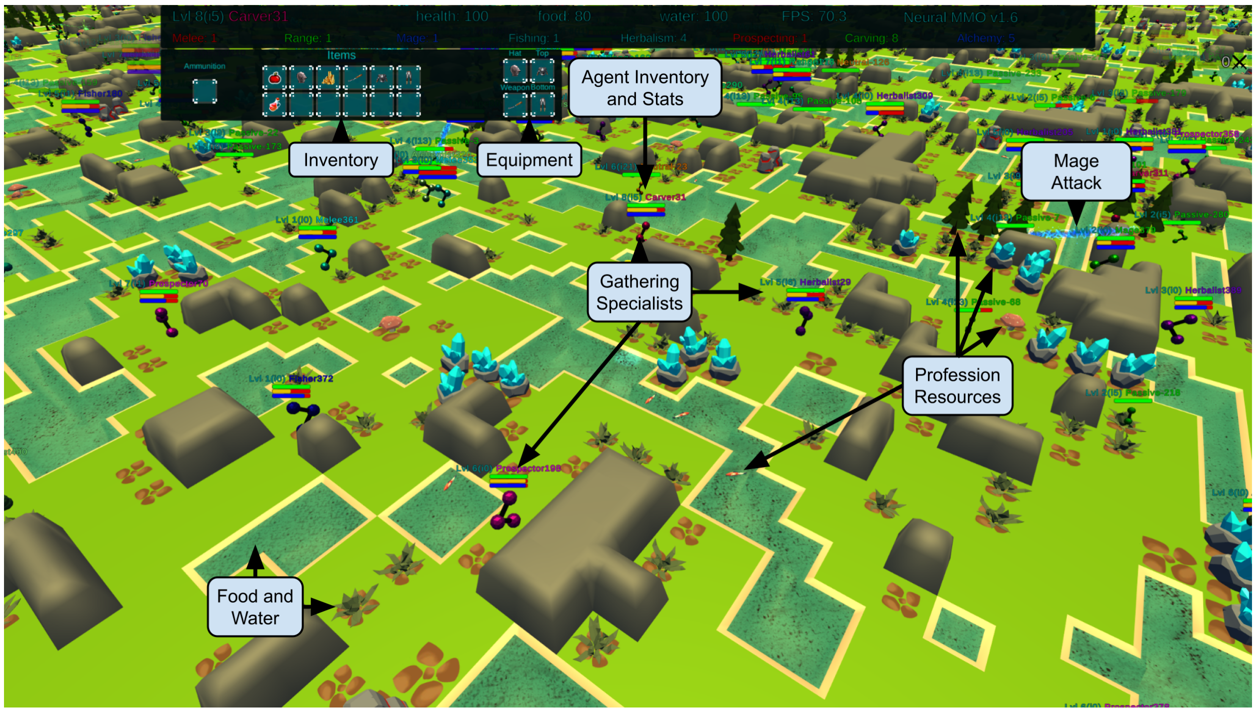


Figure 6.1: Annotated screenshot of a Neural MMO environment with profession and exchange systems. Around a hundred specialized agents are engaging in combat, gathering profession-specific resources, and exchanging goods on a global exchange.

paper-scissors dominance: mage beats melee, melee beats range, and range beats mage. In this context, attacking a specialized agent with their weakness inflicts 1.5x damage.

Gathering: There are 5 gathering professions: fishing, herbalism, prospecting, carving, and alchemy. The former two produce consumable items which restore food, water, and health. These act as supplies when exploring resource-sparse areas or as immediate healing in a pinch. The latter three professions produce ammunition that strengthens the attacks of corresponding combat styles.

Items: There are 17 types of items: 2 consumables, 3 armor pieces, 3 munitions, 3 weapons, 5 tools, and 1 currency (gold). Except for gold, each category contains items from levels 1 through 10, for a total of 161 unique items. Higher level consumables restore more food, water, and health. Higher level armor provides better defensive bonuses in combat. Higher level munitions and weapons increase damage in the respective combat style. Higher level tools enable gathering higher level items using the respective profession. Gold is inherently valuable as the only currency of exchange.

Progression: Each profession begins at level 1 and can be raised up to level 10. Experience points are awarded for using the skill. For example, fighting with melee will raise the melee skill. Higher level agents can use higher level items. This places a strong emphasis on exchange with other agents as a means for acquiring relevant items.

Equipment: Agents can equip a helmet, chestplate, platelegs, held item, and ammunition. The former three armor pieces reduce damage taken in combat. Equipping weapons

Profession	Produces	Requires
Melee	Equipment	Sword, Scrap
Range	Equipment	Bow, Shaving
Mage	Equipment	Wand, Shard
Fishing	Ration	Rod
Herbalism	Poultice	Gloves
Prospecting	Scrap	Pickaxe
Carving	Shaving	Chisel
Alchemy	Shard	Arcane Focus

Figure 6.2: Each profession produces some items and requires others. For example, herbalists require increasingly high quality gloves in order to produce increasingly high quality poultices, which agents can use to restore health.

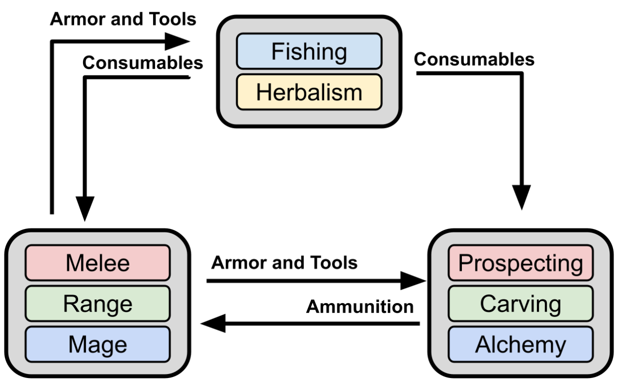


Figure 6.3: Each profession is directly or indirectly dependent upon goods from several other professions. This incentivizes exchange among agents: each profession sells excess products and buys items needed for survival and advancement.

and ammunition increases damage with the associated combat profession. Note that munitions are consumed upon use, so combat agents must constantly replenish their supply. Wielding a tool confers a significant bonus to defense, enabling gathering agents to flee from poorly equipped aggressors. Equipping armor requires that agents have at least one skill of the same level of the armor piece. Equipping a held item or ammunition requires at least the same level in the associated skill. Higher-level tools enable agents to harvest higher-level resources. Higher-level ammunition and weapons enable agents to inflict more damage in combat.

Exchange: Agents can buy and sell items on a global market using gold. Gold can be obtained from defeating scripted non-players and is inherently valuable because it is the sole currency of exchange. To sell an item, agents specify an item in their inventory and a price. To buy an item, agents purchase one of the current market offers.

6.4 Baselines

We provide 8 scripted policies. Agents spawn in the environment continuously and are uniformly sampled from these 8 (typically 10-30 of each are alive at any one time). Each policy corresponds to a single profession. Crucially, these define strategies for speciali*zing* rather than for speciali*sts* — agents begin with no inherent talents and may train any professions they wish. However, as per the equipment system description in the previous section, agents are only able to use better items according to their *highest* profession level. This strongly incentivizes specialization, as training a single skill will increase max level the fastest. At the same time, the dependency of each profession upon others for items makes it impractical for too many agents to share the same profession, as demand will greatly exceed

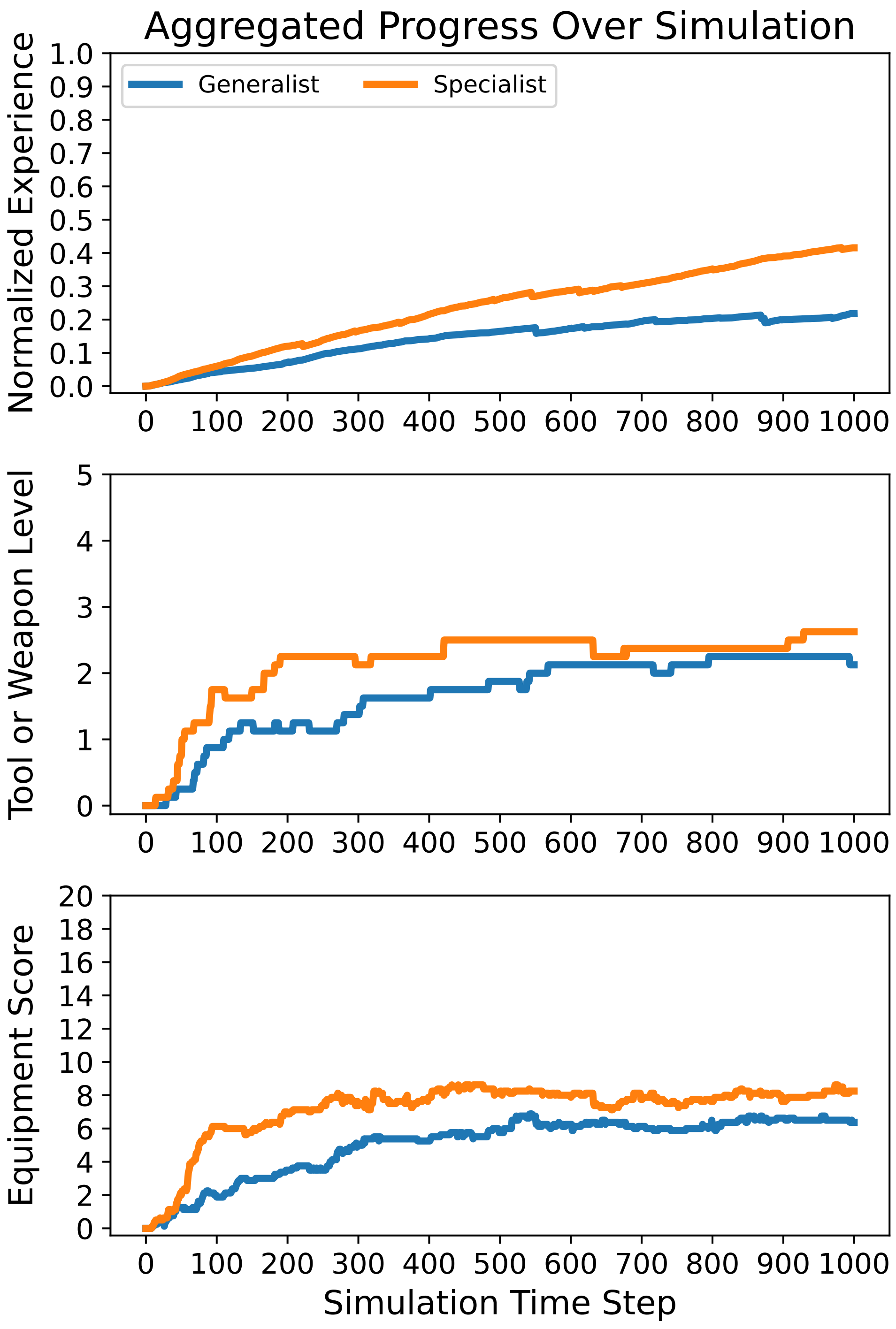


Figure 6.4: These metrics are aggregated over sets of scripted and generalist policies, respectively. Specialists gain experience points faster and aquire better tools, weapons, and equipment for their respective professions. See the appendix for replication details and per-profession statistics.

supply. Thus our choice of baselines is reasonable, though likely not optimal.

The scripted policies share most of the same core logic. All agents process observations of themselves, nearby terrain, the market, and the inventory. In this stage, we extract higher-level attributes useful for scripting — lists of items owned that are not needed and can be sold, lists of equipment owned, lists of equipment that needs to be upgraded, lists of affordable items on the market, and so forth. Next, agents scan through these attributes and select actions. Neural MMO enables agents to take multiple actions simultaneously as long as they are in different categories. In particular, agents may move, attack, use an item, buy an item, and sell an item at the same time. We prioritize some actions over others in case of conflicts. For example, agents will use supplies (to regain food/health/water) before using armor (to equip it). All agents also share foraging, pathfinding, and exploration logic. There are two main distinctions among the 8 policies. First, the 3 combat policies fight using their respective attack type and the 5 gathering policies collect the resource specific to their profession. Second, each policy buys and sells different items according to its profession.

The purpose of the profession and exchange systems is to create a feedback loop in which agents specialize in a profession, acquire items using that profession, trade with other agents depending on their needs, and use their purchases to further enhance their profession, producing better items, and so on. These dynamics emerge even with our fairly simplistic scripted policies. Figure 4 shows that specialist agents gain experience in their respective professions and acquire tools and equipment over the course of simulation. These specialist policies also outperform "generalist" policies in all three metrics. We implemented these by modifying the specialists to spend time training other skills: combat agents use all attack styles and gathering agents collect all profession-based resources. See the appendix for precise replication details and per-profession data.

We also perform a variety of "simulation-first" logging that tracks the first time an event happens during the course of a simulation — for example, when new items enter the market, when more resource from the map have been concurrently depleted than ever before, when an agent inflicts more damage than ever before, when an agent reaches a new total item score (sum of equipment levels), and so on. We include the associated log file in the appendix.

6.5 Discussion and Limitations

It is important that we distinguish between the scope of the base Neural MMO platform and our profession and exchange system. The current public release of Neural MMO already includes resource, combat, and progression systems, as well as a (greatly simplified) equipment system. We added the (full) equipment system, professions, and exchange to this. In addition, we integrated the new equipment and professions with the existing progression system. As opposed to general game development, we had to design these systems from the ground up to be compatible with and efficient using Neural MMO's observation and action processing system, which is the main bottleneck in the platform's simulation speed. Examples of these low-level design requirements include allowing buy prices to fluctuate slightly when agents attempt to buy the same item (race conditions), structuring the market to only serialize observations for the best price of each item, and using a coarse-grained leveling system to avoid overly inflating the observation space.

These details are not relevant to an understanding of the main objective and contribution of this work, but is important to establish their existence. There are a total of 17 unique item types and, including item level, 161 unique items in the environment. To our knowledge, there exist no other research environments with an equipment and exchange system of this complexity. From our experience with this project, we believe that the most likely reason for this is the technical complexity of encapsulating these systems in standard observation and action systems.

There are other possible formulations of items and exchange, including far larger level spreads, fully procedurally generated item stats, peer-to-peer trading, local markets, and bartering. We chose a coarse item level system and global exchange because they are substantially simpler to integrate into decision making processes involving neural networks. Some parameterizations are prohibitively inefficient for use in modern deep learning. Most others are prohibitively complicated, requiring significant additions to the observation and action spaces. For example, bartering would require a trade request action, an incoming trade observation, a trade reply action, and a trade accept action, with each trade requiring a sequence of actions over 3 time steps from both parties to complete. Local markets would require dynamically adding a block of observations whenever a market is within range. These systems are certainly possible to implement, but they are not clearly better and, in the presence of uncertainty, we have chosen the implementation that makes our work the easiest for others to use and build upon.

The main limitation of this work is a lack of pretrained baselines. Thus far, we have modified the baseline networks provided by the official Neural MMO repository to run on our version of the environment. We have only run a couple of small training experiments so far, and we expect that some modifications to the network architecture will be required in order to fully take advantage of the new actions. We plan on doing this in the next few months and open-sourcing this code as well. We anticipate that better models will surpass the scripted baselines and likely learn qualitatively different behaviors. There are many possible intuitive strategies in the environment that were simply too difficult to script.

Our scripted baselines demonstrate one possible set of strategies in our Neural MMO derivative environments, but the strategy space as a whole is relatively unexplored. We expect that it is possible to substantially outperform the baseline by leveraging strategies we cannot currently anticipate. It is also possible that some of these strategies will do so in an "uninteresting" manner by exploiting design flaws in the profession and exchange systems. This is a risk in all new environments, and we will make adjustments if needed. The maintainers of Neural MMO have agreed to include our systems in the next version of the platform, and the profession/exchange system is currently in public beta (though we cannot link it here during double-blind review). We will work with them to ensure it is up to standard with the rest of the platform.

6.6 Conclusion

We presented a profession and exchange system in Neural MMO that enables specialization to different roles, incentivizes trade among those roles, and creates a feedback loop by which specialized agents can produce better goods than comparable generalist agents. Our systems will be included in Neural MMO's next update and are currently in public beta. We commit to supporting this release and hope that it will be useful to researchers studying emergent-behavior in open-ended simulations.

Chapter 7

The NeurIPS 2022 Competition

The following paper titled "The NeurIPS 2022 Neural MMO Challenge: A Massively Multiagent Competition with Specialization and Trade" was published in PMLR 2023 as a summary of the associated competition. It represents a significant leap in the capabilities of agents trained on Neural MMO 1.6 as well as a refinement of the competition scenario. This work was co-authored with Enhong Liu et al. and advised by Phillip Isola. Sections of this paper were invited contributions by the winners of the competition.

In this paper, we present the results of the NeurIPS-2022 Neural MMO Challenge, which attracted 500 participants and received over 1,600 submissions. Like the previous IJCAI-2022 Neural MMO Challenge, it involved agents from 16 populations surviving in procedurally generated worlds by collecting resources and defeating opponents. This year's competition runs on the latest v1.6 Neural MMO, which introduces new equipment, combat, trading, and a better scoring system. These elements combine to pose additional robustness and generalization challenges not present in previous competitions. This paper summarizes the design and results of the challenge, explores the potential of this environment as a benchmark for learning methods, and presents some practical reinforcement learning training approaches for complex tasks with sparse rewards. Additionally, we have open-sourced[1] our baselines, including environment wrappers, benchmarks, and visualization tools for future research.

7.1 Introduction

The ecosystems and populations of Earth feature mixed cooperation and competition. Inspired by this, the Neural MMO environment was designed to accommodate a large number of agent populations with limited resources, necessitating socially-aware planning for survival. We propose the Neural MMO Challenge as a means to explore the emergence of many-agent intelligence and promote research on cognitively sophisticated reinforcement learning environments.

We have held two Neural MMO challenges to date. The previous challenge was designed to study multi-task RL and provide a benchmark for robustness and generalization in multi-agent systems [88]. Although it attracted many players and yielded valuable policies, the exact task definition and unbalanced attack mechanisms limited the emergence of interesting behaviors. Therefore, we require more complex environments and a sparse scoring system to make the environment more engaging.

In the NeurIPS-2022 Neural MMO Challenge, we updated the game mechanics and scoring method of the Neural MMO competition. The specific adjustments are as follows:

1. In order to encourage the emergence of more interesting strategies, we do not set an explicit task for agents. Instead, we demand that agents complete a final, sparse goal: survival for as long as possible.

2. We use the updated v1.6 release of Neural MMO, which introduces combat, professions, equipment, a battle-royale inspired fog mechanic, and a trading mechanism.

3. We have made dozes of fixes to the environment based on the previous competition to ensure that it is fair and strategically interesting.

Other multiagent reinforcement learning (MARL) environments include StarCraft 2 [89] (open-source) and Dota2 [90] (not publically available), as well as a number of smaller and more niche projects. The main issue is that no other environment couples large population

[1]https://github.com/NeuralMMO/NeurIPS2022NMMO-Submission-Pool

capacity with high per-agent complexity. The competition build of Neural MMO features 128 agents spread across 16 teams where each individual agent must make complex decisions about foraging, combat, specialization, cooperation, and trade. The mechanics introduced by the latest version of the environment support more diversified and complicated policies compared to previous versions. Our contributions based on the NeurIPS-2022 Neural MMO competition are as follows:

1. We provide a specific, controlled setting of Neural MMO suitable for benchmarking a wide range of learning methods.
2. We establish an imitation learning (IL) track and associated dataset of agent trajectories. Most competitions in this area are limited to reinforcement learning and rule-based (scripted) approaches. This competition shows that IL can also achieve satisfactory results.
3. We summarize our experience in designing the competition tracks and resources for participants. We have previously held a similar competition at IJCAI, but much of our previous experience needs updating. We believe the structures we present in this paper will be useful for other competitions.

7.2 Related Work

The real world is massively multiagent, and research on MARL is essential for solving large-scale and cognitively realistic problems. To aid in the study of multi-agent behavior, Jiang et al. proposed a basic MARL environment library with several simple grid-based environments [91]. The particle world project [92] allows users to customize MARL environments using their APIs. However, the rules and action space in these environments are limited.

To address this, researchers have created more realistic environments such as Fever Basketball [93], Google Research Football [94], Pommerman [95], and StarCraft II. The latter allows observation of the control ability of a MARL algorithm for multi-agent units and has led to the development of well-known algorithms like Qmix [96] and COMA [97]. While these environments enable RL algorithms to control multiple agents with different action spaces, the number of agents is still far less than the number in natural populations.

Neural MMO, an open-source complex MARL environment, is unique in that it can accommodate a vast number of populations while providing limited resources for survival, forcing populations to learn to cooperate and compete. With the introduction of a battle-royale inspired fog mechanic, equipment mechanism, and trading mechanism, researchers can simulate social behaviors, evaluate the robustness and generalization of RL algorithms, and explore diverse multi-agent research topics.

7.3 Competition Overview

Neural MMO is an environment where agents must fight, forage, barter, and more in order to survive against multiple teams of intelligent adversaries. As shown in Fig. 7.1, agents must make complex decisions at each game tick. The large decision space and long-term dependency of behaviors bring diversity to agents' strategies.

The environment has the following characteristics:

1. **Massively multiagent:** The Neural MMO environment can accommodate 1000+ agents, a capability that few environments possess.

2. **Interesting mechanics:** The battle-royale inspired fog mechanic, combat system, equipment system, and trading mechanism are introduced in Neural MMO. These fresh game mechanics make the environment appealing to researchers and worth exploring.

3. **Flexible settings:** The generation ratio of various terrains in the environment can be adjusted at any time, which enables better testing of the stability and robustness of RL algorithms. Additionally, game mechanics can be controlled flexibly to accommodate different research topics.

4. **Sparse scoring method:** The environment uses a sparse scoring rule for ranking. In other words, agents must complete long-term tasks with unclear rewards, which encourages the emergence of interesting behaviors and diverse strategies.

5. **Support for various algorithms:** The environment supports arbitrary agent controllers, not just RL algorithms.

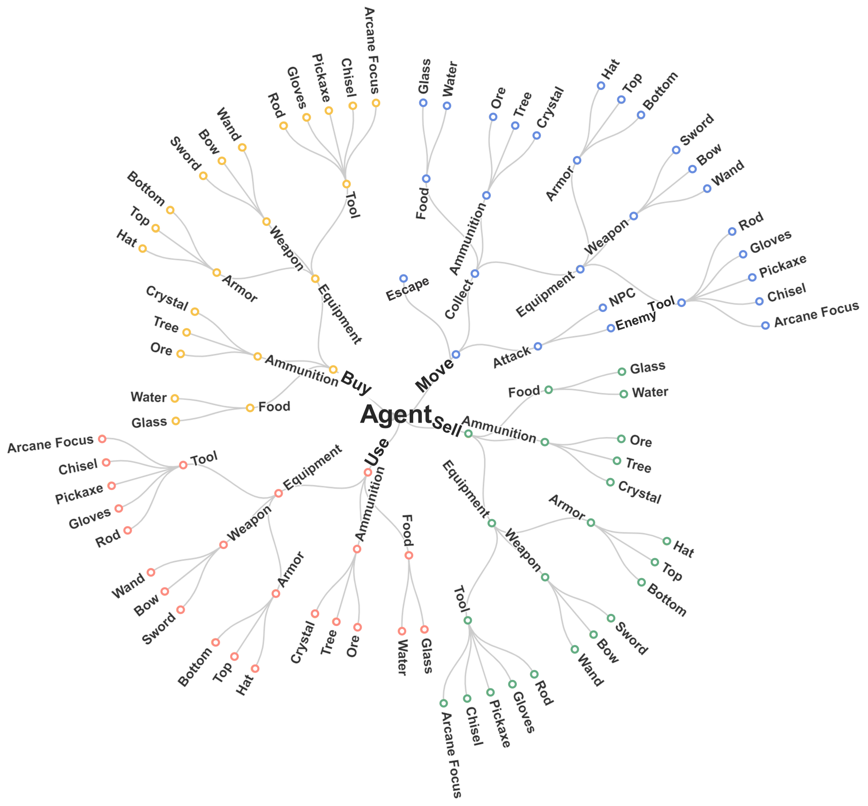


Figure 7.1: The decisions that agents face at every game tick of the environment

At each step of the game, a map with dimensions of 128×128 is randomly generated, with 128 agents from 16 teams spawning around the edges of the map. Each agent begins with initialized health, food, and water levels set to 100. Teams must navigate the environment, gathering resources, equipping weapons, hunt NPCs, and engage in combat. The game includes the following features:

1. **Terrain**: There are 16 different types of terrain on the map, each offering different benefits to agents who pass through them. For example, passing through foliage replenishes food, while passing through ore generates melee ammo.
2. **NPCs**: Three types of NPCs are evenly distributed from the edge to the center of the map. NPCs play a critical role in the game, as agents primarily obtain gold and items by defeating them. Generally, the higher-level NPCs an agent defeats, the better the items they can obtain.
3. **Blue Circle**: From the 240th game tick, a battle-royale inspired fog or "blue circle" begins to shrink. For every 16 ticks, the circle shrinks one tile. Agents within the circle take damage proportional to their distance from the safe zone.
4. **Skills**: Agents can train eight skills. A higher skill level allows agents to use better equipment, which in turn allows them to inflict more damage on enemies, take less damage, or restore more HP/Food&Water.
5. **Combat**: Agents have access to melee, range, and mage attacks, which follow a rock-paper-scissors dominance relationship: melee beats range, range beats mage, and mage beats melee. Dominance is calculated using the attacker's chosen attack skill and the defender's main combat skill. The dominant style causes 1.5 times damage.

7.3.1 Observation Space

To make decisions, each agent can observe various aspects of the game, including its own equipment information, the global information store, some enemy information, all teammate information, and all terrain information within its field of view.

7.3.2 Action Space

Table 1 shows the available actions that the agent can take. These include deciding where to move, who to attack, how to attack, what to use, what to buy, and what interaction information to expose. Note that the agent cannot move to a tile with other agents or certain special terrains, which are outlined in Table 2. Using an item will have different effects based on its properties, which are detailed in Table 3. When selling equipment, the agent can select any item in its backpack, specify the selling price, and place the item on the global market. If the sale is successful, the agent will receive the proceeds from the sale.

Table 7.1: Introduction of the action space of an agent in the Neural MMO.

Action	**Option**	**Dimension**
Move	Up,Down,Left,Right	3
Attack Target	Enemy in sight	225
Attack Style	Melee,Ranged,Mage	3
Use	Item in package	12
Sell	Item in package	12
Buy	Item in global market	170
Communicate	Discrete numerical values	128

7.3.3 Technical Details

In PvE stage 2, participants face RL-based agents that we trained. Figure 7.2 shows the structure of the model we designed. We adopt a team-based modeling approach, which integrates the information of the entire team to obtain a joint policy. For table-type features such as self-features, enemy features, env features, and market features, we use an MLP for encoding. We apply a CNN to process the image-type features, i.e., map features. To improve data efficiency, we introduce an attention component into the model. For instance, in the buy action, buying items is related to the agent and the items existing in the market. Therefore, we perform attention to self-embedding and market embedding and put the attention embedding into the MLP to obtain the final action.

We take a curriculum learning approach to train the model and obtain three different styles of built-in AI, which we refer to as "reckless", "ruthless", and "coward". In the first training stage, we set general rewards for agents to help them learn basic skills such as attacking, gathering, and escaping the blue circle. This produces the "reckless" baseline. These agents do not cooperate well, so we continue training with an additional reward to produce the "ruthless" baseline. In addition, we also set some penalties based on the checkpoint of the first training stage and obtain the "coward" baseline. Compared to the previous two built-in AIs, the "coward" watches and reacts to situations and never strikes more capable rivals first.

7.4 Competition Summary

Improvement

The NeurIPS Neural MMO competition attracted over 20,000 views, 475 registrations, 134 team registrations, and 1651 submissions. Compared to the previous competition, this year's competition was more challenging due to the introduction of more complex game mechanics. As depicted in Fig. 7.3, only a few participants achieved ideal results. In the PvE stage 1 of the IJCAI and NeurIPS challenges, many players were able to successfully overcome the built-in AIs. However, in the PvE stage 2, only 9 participants outperformed the built-in AIs in the IJCAI challenge, and in this challenge, only the top 3 participants were able to outmatch the built-in AIs.

Table 7.2: Different terrains and their corresponding property.

Element	Usage	Passable
Lava	Lethal upon contact	False
Water	Harvest Water	False
Grass	Can walk on	True
Scrub	Forest degrades to Scrub after being harvested	True
Forest	Harvest Forest for producing food	True
Stone	Impassible barriers	False
Slag	Ore degrades to Slag after being harvested	True
Ore	Harvest ore to produce melee ammunition	True
Stump	Tree degrades to Stump after being harvested	True
Tree	Harvest Tree to produce range ammunition	True
Fragment	Crystal degrades to Fragment after being harvested	True
Crystal	Harvest crystal to produce mage ammunition	True
Weeds	Herb degrades to weeds after being harvested	True
Herb	Harvest Herb to produce Ration	True
Ocean	Fish degrades to ocean after being harvested	False
Fish	Harvest Fish to produce Poultice	False

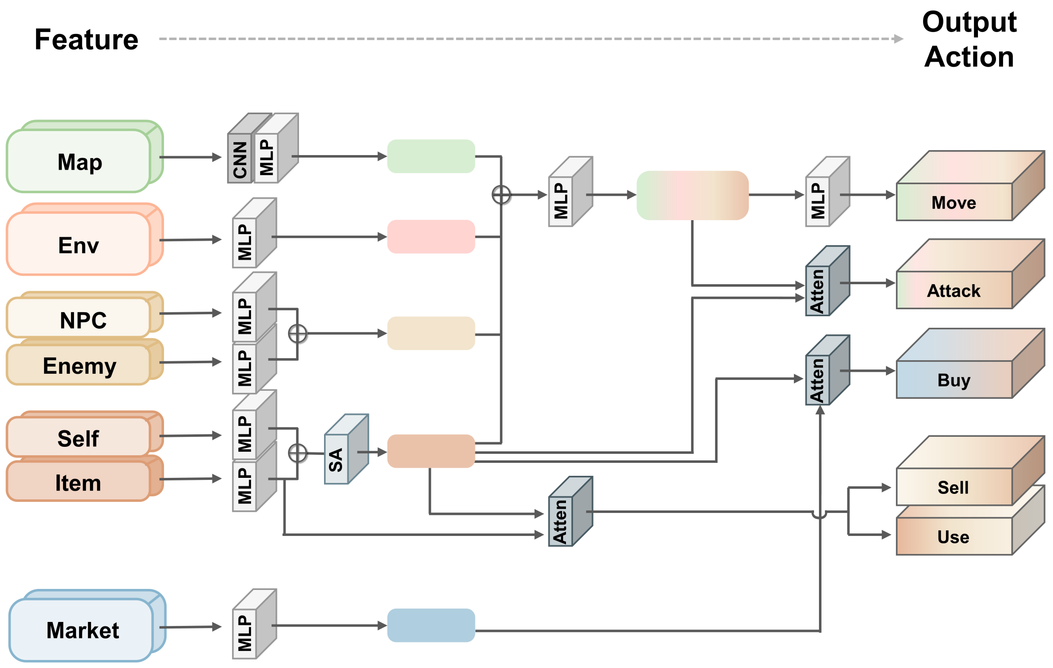


Figure 7.2: Model architecture for the PvE stage 2 built-in AI. Subnetworks include MLP, CNN, self-attention (SA), and attention. Baseline included with code release.

It is worth noting that the change in the scoring mechanism in this competition resulted in a better representation of the strengths and weaknesses of participants' policies. As shown in Fig. 7.4, in the previous competition, the top 5 participants' scores were very close, and it

Table 7.3: Different items and their corresponding property.

Category	Usage	Items	Level
Ammunition	Increase damage	Shaving/Scrap/Shard	1-10
Weapon	Increase damage	Sword/Bow/Wand	1-10
Armor	Increase defense	Hat/Top/Bottom	1-10
Consumable	Restore Food/Water/HP	Ration/Poultice	1-10
Tool	Increase the level of products after gathering resources	Rod/Gloves/Pickaxe Chisel/Arcane Focus	1-10
Gold	Buy items from the global market		N/A

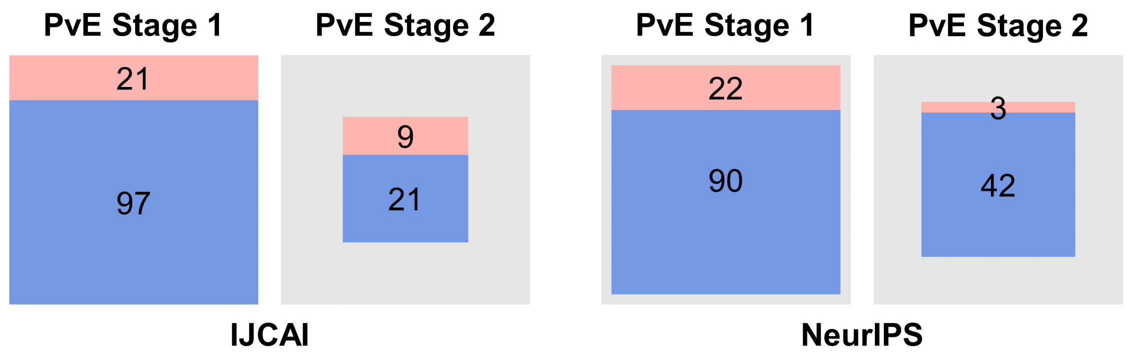


Figure 7.3: Performance of participants in the IJCAI and NeurIPS Neural MMO challenges. In the IJCAI competitions, 21 participants achieved 1.0 top 1 ratio in PvE stage 1 and 9 participants achieved 1.0 top 1 ratio in PvE stage 2. In the NeurIPS Neural MMO challenge, 22 participants achieved 1.0 top 1 ratio in PvE stage 1 and only 3 participants achieved 1.0 top 1 ration in PvE stage 2

was difficult to distinguish their strategies based on the achievement indicators. In contrast, in this competition, although the top 5 participants' results were similar, we were able to distinguish obvious differences among their policies based on evaluation indicators.

To summarize, the game mechanics of this competition were more interesting than those of the previous competition, and the flexible scoring system showcased the potential of the Neural MMO environment for MARL research.

Tools and Service

Figure 7.5 shows the submission performance curve of the NeurIPS-2022 Neural MMO PvE stage 1. Unlike the previous competition, where participants achieved ideal results without a baseline, this time, the number of submissions was infrequent, and none of them achieved good results before we provided the baseline. This indicates that as competition difficulty increases, participants require additional guidance. Therefore, we launched our RL baseline, which resulted in an increase in the frequency of submissions and breakthroughs in submissions by participants.

Apart from providing a baseline, we also offered a starter kit, Q&A, and live broad-

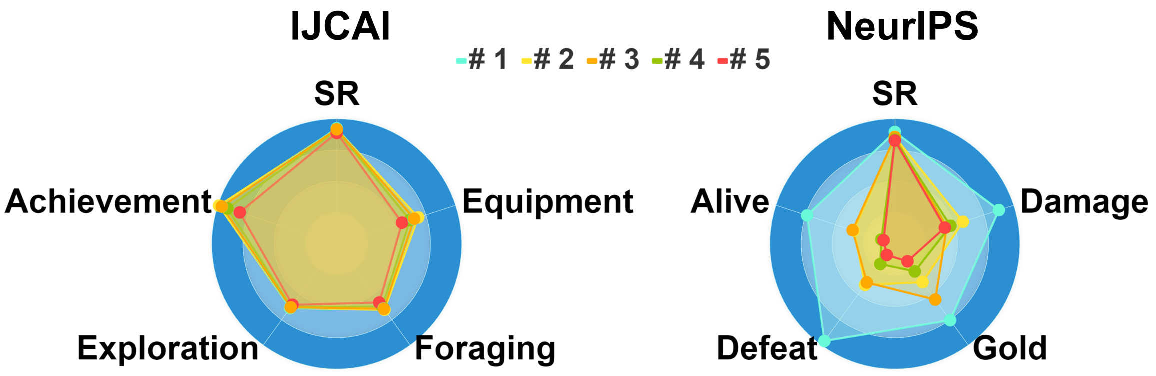


Figure 7.4: An overview of top 5 submissions' capabilities in the past two competitions.

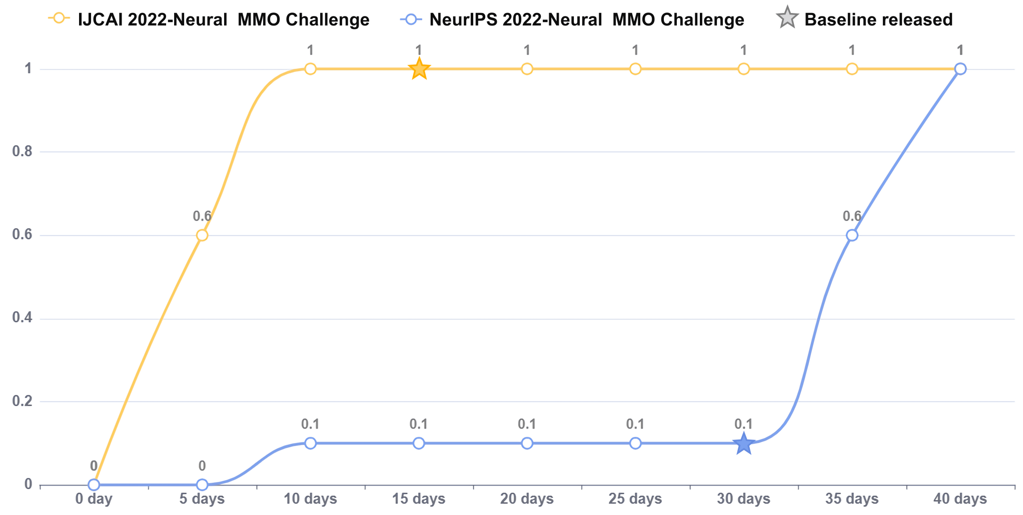


Figure 7.5: Submission performance over time in PvE Stage 1 of the past two competitions.

casts to help participants better understand the competition's progress and details, thereby improving their policies.

In summary, our tools and services enable participants to concentrate on implementing their ideas and refining their policies rather than spending time building from scratch.

7.5 Conclusion

We organized the NeurIPS 2022 Neural MMO challenge, which attracted over 500 participants from 35 countries and regions. The competition featured two tracks, PvE and PvP, and we observed that the same policy may perform differently on both tracks. In PvE, the competition is designed with a difficulty ladder. Most participants achieve a top1 ratio of 1 in the first phase, but only three players can achieve a top1 ratio of 1 in the second phase, and the score difference between participants is significant. This highlights the potential of this environment as a benchmark for algorithm robustness and generalization.

The environment introduces various mechanics, such as trading, skill differentiation, and many-agent cooperation, making it more interesting and suitable for MARL research. Analyzing the item distribution in the environment, we found some human-like trading behaviors, which surprised us.

Before the baseline was provided to participants, the number of submissions was minimal, and the majority of participants received unsatisfactory scores. This phenomenon reflects the increased difficulty of the environment. However, after the baseline was released, we observed an uptick in the number and quality of submissions. This highlights the importance of tools and services surrounding the environment for RL research, rather than relying solely on algorithmic innovation. We believe that with the advent of generic and flexible tools, there is room for substantial improvement to policy robustness and generalization on Neural MMO.

Chapter 8

Neural MMO 2.0

Chapters 8-9 cover Neural MMO 2.0, the latest version at time of writing. This version was a complete rewrite of the project with a 3x faster engine. The key feature of Neural MMO 2.0 is a flexible task system that enables users to define virtually any goal condition on the environment. This system is intended to support work on steerable agents that are able to generalize to tasks not seen during training. This is the first version for which the environment received substantial code contributions from other open source developers, as well as from talented writers and artists.

The following paper titled "Neural MMO 2.0: A Massively Multi-task Addition to Massively Multi-agent Learning" was published at NeurIPS 2023 in the Datasets and Benchmarks track. This work is co-authored with Kyoung Whan Choe et al. and advised by Phillip Isola.

Neural MMO 2.0 is a massively multi-agent environment for reinforcement learning research. The key feature of this new version is a flexible task system that allows users to define a broad range of objectives and reward signals. We challenge researchers to train agents capable of generalizing to tasks, maps, and opponents never seen during training. Neural MMO features procedurally generated maps with 128 agents in the standard setting and support for up to 1024. Version 2.0 is a complete rewrite of its predecessor with three-fold improved performance and compatibility with CleanRL. We release the platform as free and open-source software with comprehensive documentation available at neuralmmo.github.io and an active community Discord. To spark initial research on this new platform, we are concurrently running a competition at NeurIPS 2023.

8.1 Novelty and Impact

Neural MMO is a reinforcement learning platform first released in 2019 [98], with updates featured in short-form at AAMAS 2020 [99] and ICML 2020, and a new version published in the 2021 NeurIPS Datasets & Benchmarks track [100]. Since then, the platform has gained traction through competitions at IJCAI 2022 and NeurIPS 2022, totaling 3500+ submission from 1200+ users, which significantly improved state-of-the-art on the platform. Alongside these developments, our community on Discord has grown to nearly 1000 members.

While previous versions of the environment defined fixed objectives through only the reward signal, Neural MMO 2.0 introduces a flexible task system that allows users to define per-agent or per-team objectives and rewards, expanding the platform's applicability to a broader range of problems. In particular, Neural MMO 2.0 enables research on generalization, open-endedness, and curriculum learning—areas that were difficult to explore with prior versions and which require sophisticated, flexible simulators. There are few if any other environments of comparable scope to Neural MMO available for these problems.

Practical engineering improvements are at the core of Neural MMO 2.0. These include:

1. A 3x faster engine. This was developed as part of a complete rewrite of our 5+ year old code base and is particularly important for reinforcement learning research, where simulation is often the bottleneck. For example, the upcoming competition would not be practical on the old engine.

2. Simple baselines with CleanRL, a popular and user-friendly reinforcement learning library. CleanRL and most other reinforcement learning frameworks are not natively compatible with environments of this complexity, and previous versions required convoluted, environment-specific compatibility wrappers. Neural MMO 2.0 integrates PufferLib to solve this problem.

3. A web client available at neuralmmo.github.io/client, generously open-sourced by Parametrix.AI. This client offers improved visualization capabilities and eliminates setup requirements.

Additionally, the platform's documentation has been professionally rewritten in consultation with the development team. This, along with a more intuitive and accessible website

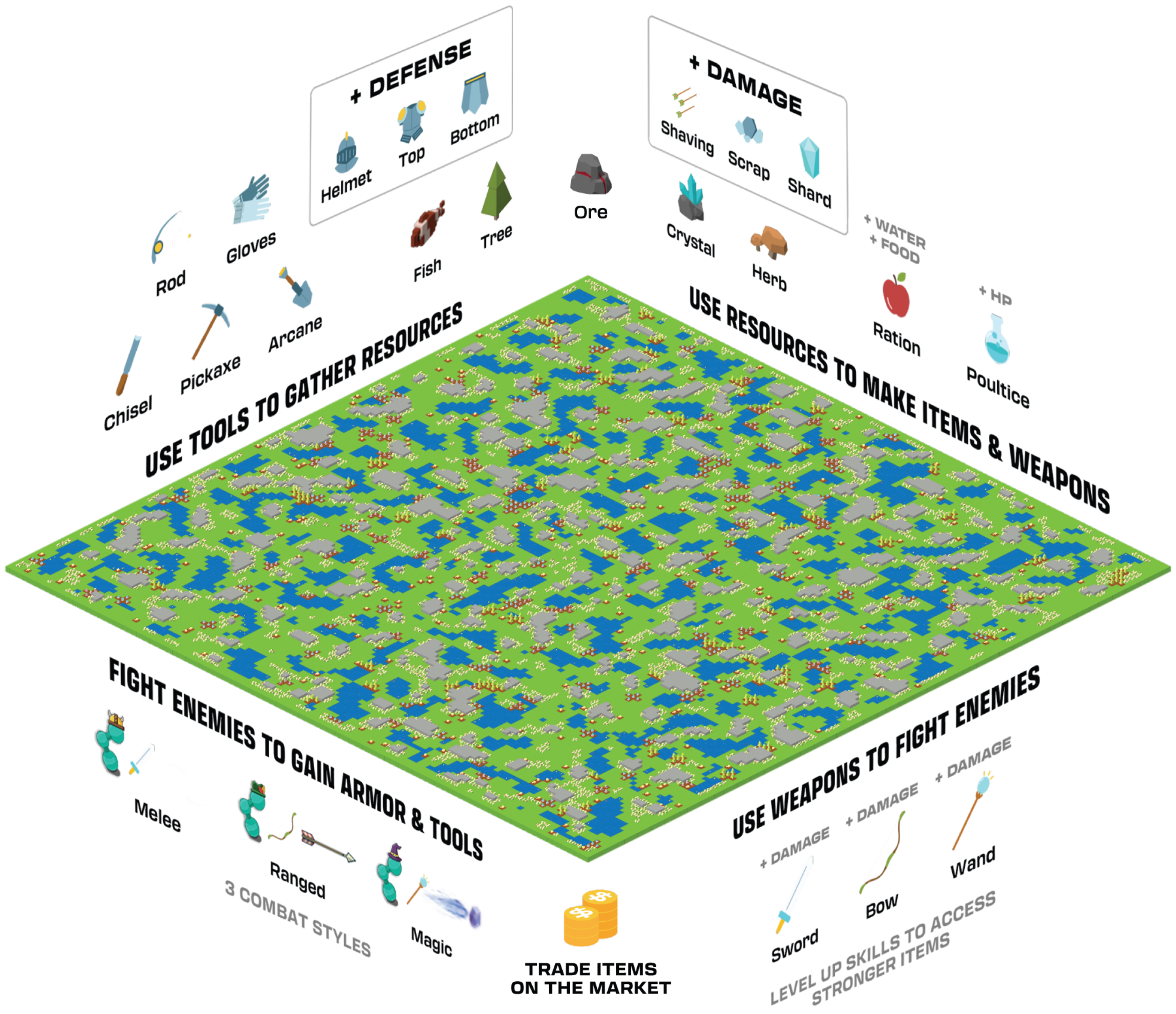
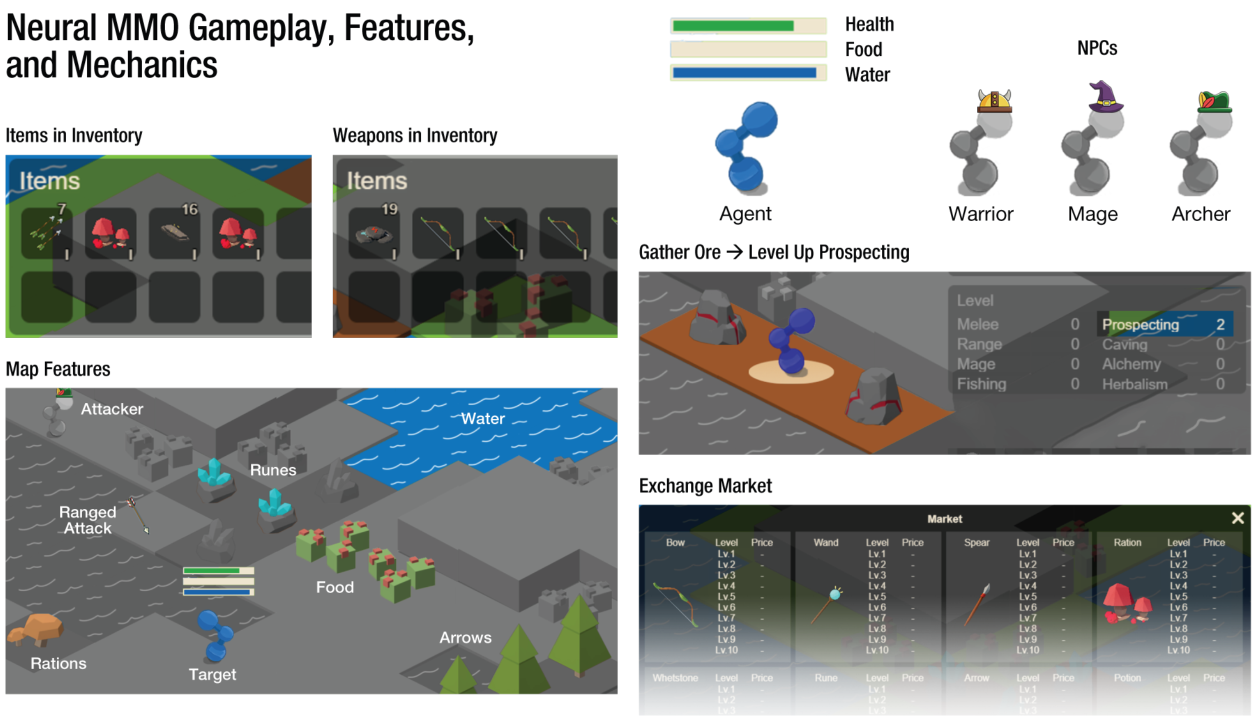


Figure 8.1: Overview of Neural MMO 2.0. Users can define tasks to specify a broad range of agent objective. In general, these involve using tools to gather resources, using resources to make items and weapons, using weapons to fight enemies, and fighting enemies to gain armor and tools. Full documentation is available at neuralmmo.github.io.



Level			
Melee	0	Prospecting	2
Range	0	Caving	0
Mage	0	Alchemy	0
Fishing	0	Herbalism	0

Bow	Level	Price	Wand	Level	Price	Spear	Level	Price	Ration	Level	Price
	Lv.1	-		Lv.1	-		Lv.1	-		Lv.1	-
	Lv.2	-		Lv.2	-		Lv.2	-		Lv.2	-
	Lv.3	-		Lv.3	-		Lv.3	-		Lv.3	-
	Lv.4	-		Lv.4	-		Lv.4	-		Lv.4	-
	Lv.5	-		Lv.5	-		Lv.5	-		Lv.5	-
	Lv.6	-		Lv.6	-		Lv.6	-		Lv.6	-
	Lv.7	-		Lv.7	-		Lv.7	-		Lv.7	-
	Lv.8	-		Lv.8	-		Lv.8	-		Lv.8	-
	Lv.9	-		Lv.9	-		Lv.9	-		Lv.9	-
	Lv.10	-		Lv.10	-		Lv.10	-		Lv.10	-
Whetstone	**Level**	**Price**	**Rune**	**Level**	**Price**	**Arrow**	**Level**	**Price**	**Potion**	**Level**	**Price**
	Lv.1	-		Lv.1	-		Lv.1	-		Lv.1	-
	Lv.2	-		Lv.2	-		Lv.2	-		Lv.2	-
	Lv.3	-		Lv.3	-		Lv.3	-		Lv.3	-

Figure 8.2: Neural MMO 2.0 features procedurally generated terrain, 7 resources to collect, 3 combat styles, 5 gathering and 3 combat professions to train and level up, scripted NPCs that roam the map, and 16 types of items in 10 quality levels including weapons, armor, consumables, tools, and ammunition. An environment-wide market allows agents to trade items with each other.

layout, marks a significant step towards improving user engagement. A collection of papers detailing previous versions and competitions is available on neuralmmo.github.io.

8.2 Neural MMO 2.0

Neural MMO (NMMO) is an open-source research platform that is computationally accessible. It enables populations of agents to be simulated in procedurally generated virtual worlds. Each world features unique landscapes, non-playable characters (NPCs), and resources that change each round. The platform draws inspiration from Massively Multiplayer Online games (MMOs), which are online video games that facilitate interaction among a large number of players. NMMO is a platform for intelligent agent creation, typically parameterized by a neural network. Agents in teams must forage for resources to stay alive, mine materials to increase their combat and task completion capabilities, level up their fighting styles and equipment, practice different professions, and engage in trade based on market demand.

In the canonical setting of NMMO that will support the upcoming competition, users control 8 out of a total of 128 simulated agents. The ultimate goal is to score more points by completing more tasks than the other 118 agents present in the same environment. Originally, we planned to introduce team-based tasks and objectives, but we decided to postpone the introduction of these given the practical limitations of learning libraries. After the conclusion of the competition, top submissions will be provided as baseline opponents. NMMO includes the following mechanisms to induce complexity into the environment:

- Terrain: Navigate procedurally generated maps
- Survival: Forage for food and water to maintain your health
- NPC: Interact with Non-Playable Characters of varying friendliness
- Combat: Fight other agents and NPCs with Melee, Range, and Magic
- Profession: Use tools to practice Herbalism, Fishing, Prospecting, Carving, and Alchemy
- Item: Acquire consumables and ammunition through professions
- Equipment: Increase offensive and defensive capabilities with weapons and armor
- Progression: Train combat and profession skills to access higher level items and equipment
- Exchange: Trade items and equipment with other agents on a global market

A detailed wiki is available on the project's document site.

8.3 Background and Related Work

In the initial development phase of Neural MMO from 2017 to 2021, the reinforcement learning community witnessed the release of numerous influential environments and platforms. Particularly noteworthy among these are Griddly [43], NetHack [101], and MineRL [44]. A comprehensive comparison of these with the initial Neural MMO can be found in our previous publication [100]. The present work primarily focuses on recent advancements in the reinforcement learning environments sphere. Griddly has sustained ongoing enhancements, while MineRL has inspired several competitive initiatives. Since 2021, only a few new environments have emerged, with the most pertinent ones being Melting Pot [102], and XLand [103]. Melting Pot and its successor, Melting Pot 2.0 [104], comprise many multiagent scenarios intended for evaluating specific facets of learning and intelligence. XLand and its sequel, XLand 2.0 [105], present large-scale projects focusing on training across a varied curriculum of tasks within a procedurally generated environment, with a subsequent emphasis on generalization to novel tasks. Compared to Melting Pot, Neural MMO is a larger environment with flexible task specifications, as opposed to a set of individual scenarios. XLand, while architecturally akin to Neural MMO, predominantly explores two-agent settings, whereas Neural MMO typically accommodates 128. A crucial distinction is that XLand is primarily a research contribution enabling the specific experiments presented in the publication. It does not provide open-source access and is not computationally practical for academic-scale research. Conversely, Neural MMO is an open-source platform designed for computational efficiency and user-friendliness.

8.4 Task System

The task system of Neural MMO 2.0, a central component of the new version, comprises three interconnected modules: GameState, Predicates, and Tasks. This system leverages the new Neural MMO engine to provide full access to the game state in a structured and computationally efficient manner. This architectural enhancement surpasses the capabilities of Neural MMO 1.x, allowing users to precisely specify the tasks for agents, paving the way for task-conditional learning and testing generalization to unseen tasks during training.

8.4.1 GameState

The GameState module acts as a high-performance data manager, hosting the entire game state in a flattened tensor format instead of traditional object hierarchies. This vectorization serves a dual purpose: first, it accelerates simulation speeds—a crucial factor in generating data for reinforcement learning; and second, it offers researchers an efficient tool to cherry-pick the required bits of data for defining objectives. While this format was originally inspired by the data storage patterns used in MMOs, adaptations were needed to support the computation of observations and definition of tasks.

Alongside GameState, we also introduced auxiliary datastores to capture *event* data—unique in-game occurrences that would be not be captured otherwise. These datastores record things that happen, such as when an agent lands a successful hit on an opponent or gathers a resource, rather than just the outcomes, i.e. damage inflicted or a change in tile state. Events enable the task system to encompass a broader range of objectives in a computationally efficient manner.

To illustrate the flexibility provided by GameState access, let's walk through some representative query examples. The snippets in the GameState Appendix employ both the global and agent-specific GameState queries. Global access is useful for game dynamics such as time and environmental constants. We also provide a convenience wrapper for accessing agent-specific data.

This query API gives researchers direct access to the mechanics of the game environment, offering a rich playground for studying complex multi-agent interactions, resource management strategies, and competitive and cooperative dynamics in a reinforcement learning context.

8.4.2 Predicates

The Predicates module offers a robust syntax for defining completion conditions within the Neural MMO environment.

Predicates interface with the game state (the "subject") to provide convenient access to agent data and any additional arguments desired. Predicates return a float ranging from 0 to 1, rather than a boolean. This design choice supports partial completion of predicates—crucial for generating dense reward functions—while still allowing tasks to be considered complete when the return value equals 1. As a starting point, Neural MMO offers 25 built-in predicates that can access every aspect of NMMO. The first example in the Predicates

Appendix illustrates the creation of a more complex objective, building on the game state and subject from the previous section.

The second example in the Predicates Appendix demonstrates how the Predicate system can be used to articulate complex, high-level objectives. The *FullyArmed* predicate demands that a specific number of agents in a team be thoroughly equipped. An agent is considered fully equipped if it has an entire set of equipment (hat, top, bottom, weapon, ammo) of a given level. To acquire a complete equipment set, agents would need to utilize various professions in different locations on the game map, which could take several minutes to accomplish. This task's complexity could be further amplified by setting a condition that each team member be outfitted specifically with melee, ranged, or magical equipment, necessitating the coordinated use of all eight professions.

8.4.3 Tasks

The Task API allows users to formulate tasks by combining predicates and assigning per-agent rewards based on the outcomes of intermediary predicates. This approach not only maintains an account of tasks completed but also provides a denser reward signal during training. We expect that most users will form tasks using the library of pre-built predicates. For advanced users, direct access to GameState enables mapping conditions on the game's internal variables to rewards, circumventing the need for intermediate predicates. The predicate can then be turned into a task. See the Tasks Appendix for an example.

8.5 Performance and Baselines

Neural MMO 2.0's new engine runs at approximately 3,000 agent steps per CPU core per second, up from the approximately 800 to 1,000 in the previous version. Its design focuses on native compatibility with a vectorized datastore that represents game state. This allows us to keep the environment in Python while maintaining efficiency, providing easier access for researchers looking to modify or extend Neural MMO.

Simulation throughput is highly dependent upon agent actions within the game. We compute statistics by having agents take random actions, but to maintain a fair estimate, we eliminate mortality since dead agents do not require any computation time. Given that NMMO equates one action to 0.6 seconds of real time, a single modern CPU core can simulate at 5,000 times real-time per-agent, equivalent to 250M agent steps or roughly 2.5 terabytes of data per day at approximately 10 KB per observation.

We also release a baseline model with training code and pretrained checkpoints. Compared to the previous TorchBeast [106] baseline, our new model builds on top of CleanRL. This is a simpler library that is much easier to work with, but it is not designed to work with complex environments like Neural MMO by default. To achieve interoperability, we integrate with PufferLib, a library designed to streamline the various complexities of working with sophisticated environments.

8.6 Limitations

Despite its enhancements, Neural MMO 2.0 does not incorporate any novel game mechanics absent in version 1.x. However, in the most recent competition, even the top approaches did not learn to comprehend and utilize all of the game systems, and there is substantial room for improvement. Moreover, agent specialization within a team remained limited. These circumstances are likely attributable to the overly broad survival objective that invariably promotes dominant strategies, posing a challenge to balance. However, with the introduction of a more flexible task system in Neural MMO 2.0, we redefine performance as the capability to execute novel tasks, thereby enabling researchers to harness the existing game mechanics in a way not feasible in earlier versions.

8.7 Accessibility and Accountability

Neural MMO has been under active development with continuous support for the past 6 years. Each of the six major releases in this period was accompanied by comprehensive documentation updates, a guarantee of timely user support, and direct access to the development team via through the community Discord. The project will continue to support and maintenance. A fourth competition has been accepted to NeurIPS 2023 and is expected to improve the current baseline. The code for this project is hosted in perpetuity by the Neural MMO GitHub organization under the MIT license. We provide both a pip package and a containerized setup including the baselines. Documentation is consistently available on neuralmmo.github.io with no major outages recorded to date. The entire project is available as free and open-source software under the MIT license.

Neural MMO implements the standard PettingZoo [107] ParallelEnv API, a direct generalization of the OpenAI Gym [56] API for multi-agent environments. Our baselines utilize CleanRL's [108] Proximal Policy Optimization (PPO) [52] implementation, one of the simplest and most widely used reinforcement learning frameworks, with all algorithmic details encapsulated in a single file of approximately 400 lines. While CleanRL was originally designed for simpler environments like single-agent Atari [31] games, Neural MMO extends its capabilities through PufferLib, which provides native compatibility through a multiagent vectorization backend. The details of this library are available at pufferai.github.io.

8.8 Ethics and Responsible Use

Neural MMO is an abstract game simulation featuring systems of combat and commerce. These elements are incorporated for visual interpretability and are not representative of any actual violence or commerce systems. We are confident that these systems are sufficiently removed from their real-world counterparts that Neural MMO would not be a useful training platform for developing such systems. The use of game-like elements in Neural MMO is a deliberate choice to align with human intuition and does not reflect any specific real-world scenario. Neural MMO's primary objective is to facilitate research on understanding and advancing the capabilities of learning agents. The project does not include any real-world human data other than the code and documentation voluntarily submitted by contributors and some 3D asset files commissioned at fair market rate.

8.9 Conclusion

Neural MMO 2.0 is a significant evolution of the platform. We invite researchers to tackle a new challenge in generalization across unseen new tasks, maps, and adversaries. Furthermore, we have achieved significant advancements in computational efficiency, yielding a performance improvement of over 300%, and have ensured compatibility with popular reinforcement learning frameworks like CleanRL. This opens up the potential for broader utilization by researchers and makes the environment significantly more accessible, especially to those working with more modest computational resources. Neural MMO has a five-year history of continuous support and development, and we commit to maintaining this support, making necessary adaptations, and facilitating a lively and active community of users and contributors. With the concurrent NeurIPS 2023 competition, we look forward to sparking new research ideas, encouraging scientific exploration, and contributing to progress in multi-agent reinforcement learning.

Chapter 9

The NeurIPS 2023 Competition

The NeurIPS 2023 competition challenged participants to train agents that could generalize to tasks, maps, and opponents never seen during training. The winning submission improved 4x over our internal baseline. While the winners of previous competitions used at least some scripting for portions of the policy that were tricky to learn, this competition required end-to-end learning. At the same time, less hardware was required than for any previous competition. We reproduced the winning submission from scratch in only a few hours on a single desktop. This success represents the final milestone of my doctoral work.

The following paper titled "The Neural MMO 2.0 Competition on Multi-Task Reinforcement Learning" is expected to appear in the NeurIPS 2024 Datasets and Benchmarks track. This is an invited submission with preference in reviewing as part of the competition's acceptance to NeurIPS 2023. This work is co-authored with Kyoung Whan Choe et al. and advised by Phillip Isola. Sections of this paper were invited contributions by the winners of the competition.

We present the results of the NeurIPS 2023 Neural MMO Competition, which attracted over 200 participants and submissions. Participants trained created goal-conditional policies that generalize to tasks, maps, and opponents never seen during training. The top solution achieved a score 4x higher than our baseline within 8 hours of training on a single 4090 GPU. We open-source everything relating to Neural MMO and the competition under the MIT license, including the policy weights and training code for our baseline and for the top submissions.

9.1 Introduction and Related Work

Neural MMO 2.0 is a reinforcement learning (RL) platform for massively multiagent research. In its competition setting, the environment features 128 agents competing for various resources, items, and equipment in a procedurally generated world with dynamic profession and economy systems. Previous versions of the platform have been used to host major competitions, including at IJCAI and NeurIPS, totaling 1200+ participants and 3500+ submissions.

Related competitions include Procgen 2020 [109], Nethack 2021 [110], and the MineRL 2020-2023 [111] and LuxAI 2021-2023 series. Procgen challenges participants to train sample-efficient agents on arcade games. The competition is important for historical purposes as the first large RL competition. Nethack is important for the unprecedented complexity of the environment. The game is an 80s dungeon crawler with procedural generated levels that typically takes new human players months to years to beat for the first time, even if they are using guides and wikis. The MineRL series has proposed multiple challenges over the years, largely focused around sample-efficient learning. These competitions are important for their focus on learning low-level control from pixels, ease of interpretability, and broad appeal.

The LuxAI **lux-ai-season-2**, [75] series of multiagent challenges is most closely related to Neural MMO. The first competition had a high emphasis on scripted submissions, but the second competition made RL a primary focus and introduced a new two-player real-time strategy game where each player controls several units. The results are discussed in a YouTube video, but to our knowledge, a full analysis has not been published.

9.2 Neural MMO

Neural MMO has been in active development since 2017 and was first published in 2019 [98]. We refer readers to the recent NeurIPS 2023 D&B paper [112] for a full description of the environment used in the competition, as well as for comparisons to related environments outside of competitions. For convenience, we briefly summarize the key details of Neural MMO as relates to the most recent competition.

The environment simulates 128 agents on procedural generated maps with several different types of terrain and resources. Agents may move around the map. They must forage for food and water in order to survive and can obtain several other resources. Agents can engage in melee, ranged, or magic combat, which interact with each other in a rock-paper-scissors (or super effective / not very effective) manner. They can also level up foraging and combat related skills to help them perform better. Items like armor, weapons, and ammunition may also be acquired and used to improve combat prowess, while consumables can help agents stay alive. Agents may list items on a global market, and they may in turn purchase items listed by other agents. Suffice to say that Neural MMO simulates a lot of agents and, at least by the standard of environments used in academic research, that the environment supports rich interactions and strategy. At the start of an episode on a new map, each agent is assigned a task to complete. Tasks may involve traveling to a certain location on the map, acquiring certain items or skills, defeating other agents, and so on. Participants are judged by the number of tasks their agents complete over many episodes on different maps.

9.3 Task-Conditional Learning

To our knowledge, this is the first RL competition on *task-conditional* learning. Instead of a single fixed objective, submitted agents are given the task to solve at inference time. This is made possible by Neural MMO 2.0's new task system. The NeurIPS 2023 D&B paper provides full details, but the key implication for the purpose of this competition is that we can check nearly arbitrary conditions on the agent, environment, and on other agents. The baseline training curriculum has 1297 tasks, including easy goals like eating and drinking. These "scaffolding" tasks are sampled more frequently to enhance agent learning. The evaluation tasks were designed to test if agents successfully engage in all aspects of Neural MMO, spanning survival, combat, exploration, leveling up all skills, using all items, and making money in the market. There are only 63 evaluation tasks, which focus on measuring performance on hard goals. Note that some tasks are too hard to solve without a guiding curriculum. For example, agents perform poorly at the training tasks of equipping medium-level items. Training directly on the evaluation task of equipping high-level items would not improve performance, since agents cannot complete the prerequisites.

Individual tasks are specified in plain code as functions (predicates) called with specific arguments. This means that it is possible to embed tasks into a fixed-length vector by feeding their implementation into a language model. As a result, we have a simple model input that can be used to train task-conditional agents. The training tasks are defined by a set of predicate functions called with specific arguments. When encoded with a LLM, different predicates produce distant embeddings, but the same predicate with different arguments produce similar embeddings. The overlap between training curriculum and evaluation tasks is 22.3 % when considering only predicates and 2.1 % when considering both predicate and arguments. As seen in figure **??**, there is a small but significant gap between most of the training and evaluation task embeddings. Unlike LLM agents, this approach does not require running a large model during inference to control each action. Capable Neural MMO models are only millions to low 10s of millions of parameters.

9.4 Rules & Baseline

Participants were allowed to modify the model architecture, RL algorithm, and reward function. Training was capped to 8 A100 hours equivalent of training compute and 12 CPU cores.

Team Name	Submission id	PvE (%)	PvP (%)
Takeru	246748	17.09	25.21
Yao Feng	246505	16.56	24.88
Saidinesh	246539	10.65	12.56
Mori	244278	8.84	10.61
Jimyhzhu	246707	8.14	10.07
Baseline	n/a	5.98	6.39

Table 9.1: Top submissions and baseline performance. PvE refers to preliminary evaluation in an environment without the other participants' submissions. PvP is the final evaluation where all of the top policies are loaded into the same shared environments. The PvE and PvP numbers indicate the completion rate of the evaluation tasks.

Large hyperparameter sweeps were disallowed, and we informed participants that we would investigate any submission that contained hyperparameters tuned to multiple significant digits. Winners were required to open-source their implementations, and our post-competition investigations (see Discussion) revealed no instances of cheating. We released a 3.9 million parameter baseline model, of which 1 million are contained in the task embedding projection layer. Neural MMO has structured observations and actions that require a structured network architecture to process. A small convolutional subnetwork was used to process observations of the nearby terrain. Linear layers were used to process embeddings of nearby agents as well as items and the market. The action decoder includes a mix of linear layers, for actions such as moving and deciding whether to buy/sell, and pointer networks [113], for selecting targets to attack or an item from the inventory.

The baseline was trained with Clean PuffeRL, which is PufferLib's variant of CleanRL's [114] PPO [52] implementation. The use of PufferLib solves several infrastructure problems, such as batching structured observations and communicating that data across subprocesses. There is also one important modification to the learning algorithm itself that is worth mentioning: because Neural MMO has a variable number of agents, and because most learning libraries (CleanRL included) expect fixed size observations, it is common to zero-pad observations. This is a problem in Neural MMO. It is common for the last quarter of the game to be 90 % padding because many agents die near the start of the game. This destabilizes learning by lowering and randomizing the effective batch size. Clean PuffeRL alters CleanRL's data structures to omit padding. This was the key modification that allowed us to improve the usability of our baselines this year.

9.5 Evaluation

The evaluation consisted of two stages. In the first stage (PvE), all 128 agents were controlled by the submitted policy alone. We ran 32 episodes on the same 4 map seeds, repeating 65 trials per task (32 episode * 128 agents / 63 tasks), which are randomly sampled. As a sanity check, we also evaluated the average agent lifespan. Across 86 submissions, we found a correlation of 0.91 ($p<0.001$, df=84) between rank according to average lifespan and according to tasks completed. Based on the PvE results, we selected the policies (max 1 per participant) that performed better or comparable to our baseline for the PvP evaluation. Note that the PvE evaluation score can be underestimated if the policy is strong because each agent is evaluated on their own. Strong opponents can impede task performance: the top score in PvE was 17.09, but it was 25.21 in PvP.

In the second round of evaluation, all policies substantially better than the baseline (9 total) were put into the same shared environment. In each episode, 128 agents were split into 9 groups of 14 agents. One of the submitted policies independently controlled all agents within a single group. The remaining two agents were controlled by an untrained baseline policy. Each round of evaluation used 256 held-out maps and 200 episodes, repeating 44 trials per task (200 episodes * 14 agents / 63 tasks), and nine rounds with different seed (i.e., 1800 episodes) were performed to determine winners. We ran multiple evaluations over the course of the competition, and the results were consistent for the same models. We invited the creators of the top submissions to coauthor this manuscript and to document their approaches. Their reports are provided below.

9.6 Winning Submissions

9.6.1 Yao Feng

Since the environment is complex and the training time is restricted, our goal is to increase policy expressiveness, decrease exploration difficulty, and help generalization. To achieve these goals, we make changes to the policy, training configurations and the reward function.

Policy Structure

Some encoder layers of the baseline did not end in a ReLU. We added ReLU activations to these. Here are other modifications that we made to encoders:

Tile Encoder: We add a ResNet block and use normalized absolute instead of the relative position.

Player Encoder: We remove useless features such as id, attacker_id and message. Also, we use the same embedding vectors for different attributes to reduce the number of trainable parameters. We also add an MLP over the embedding vectors and layer norm layers before applying activation functions to the outputs.

Item Encoder: We delete the item_offset property because it is never used.

Inventory/Market/Task Encoder: We add layer norm before applying ReLUs to the outputs.

We add an additional ReLU activation function and two LSTM cells before getting the encoded observation to reduce redundant information and consider the historical information. Because the agent can only see the nearby area, taking historical information into account is important. Besides, we add orthogonal initialization to fully-connected layers. We also restrict the action space to reduce exploration difficulty. Giving items or gold is sub-optimal in most of the cases, so we simply disable these actions. We also disable attacking neutral NPCs, because we observe that this frequently cause the death of our agents.

Training Configurations

Here are main changes to the training configurations.

PPO Configurations: We use a learning rate of 1e-4, a clip_coef of 0.1, and a batch size of 128. We collect about 65,536 steps from 8 environments and save them into one buffer each time before we start training, and we use each sample twice (i.e., ppo_update_epochs = 2).

Number of Maps: We increase the number of maps to 1024 to help generalization.

Spawn Immunity: A large value is helpful for initial exploration but not for generalization if the value is not the same in future evaluations. We set it to 20 initially, and the value is selected randomly from $0 \sim 20$ after the death of any agent.

Resilient Population: This was an experimental training-time setting included with the baseline. We disable it because it is disabled in evaluation.

Reward Design

Here are the rewards we used (we skip the task reward because it is given). We do not use the given HP restoration, meander and explore reward because we find them not very useful.

HP: Proportional to changes in HP. Encourages maintaining a healthy status.

Experience: Proportional to the maximum experience across skills. Encourages specialization.

Defense: Proportional to the average defense (melee, range, mage). Encourages obtaining armor.

Attack: Proportional to the change of the summation of inflicted damage to encourage attacks.

Gold: Proportional to the change in gold. This discourages buying useless items.

9.6.2 Takeru

Models and Features

We observed some instability in the training of the baseline, where the value loss could become excessively large, and the policy entropy would become 0 early. One possible reason for this is that the baseline model did not handle the scaling of continuous features for items properly. We addressed this by scaling all continuous features to [0, 1]. In addition, considering the sparse rewards in difficult tasks and to reduce the difficulty of model training, we made some modification.

- Treat player states, such as coordinates, health, level, as continuous features.
- Remove feature coordinates from Tile states, as the positional information is already modeled using CNN.
- Disable the actions of giving item and giving gold, as these actions are not helpful in the current competition where teamwork is not considered.

This modified baseline model significantly outperforms the original baseline model under the same training configuration. And this is the model of our final solution.

Based on the modified baseline model, we made some attempts to optimize the model structure, but we did not achieve significantly better results. For example, encode player or item entities using Transformer encoder, encode tiles using ResNet blocks. We analyzed that the bottleneck lies in the sparse rewards of difficult tasks, so we didn't invest much time in trying out various model structures.

Due to the constraints of the evaluation setting, we did not invest time in studying the use of LSTM or adding historical information features. However, we believe that historical information is necessary because it allows agent to remember the location of resources, for example.

Training Configurations

For the game environment configurations used in training, we conducted some experimental analysis. We have the following different settings compared to the baseline.

Number of Maps: We observed significant differences in task completion rates across different maps. To enhance the model's generalization ability to different maps, we increased the number of maps used for training to 1280.

Early Stop Agent Num: It is used to truncate the game when the number of surviving agents is less than the value of it. The baseline set it to 8 in order to improve sampling throughput. However, we found that it would result in the loss of reward signals for difficult tasks in the later stages of the game, which would hinder learning to complete these difficult tasks. We set it to 0.

Resource Resilient: It is used to reduce the damage suffered by some agents due to starvation or dehydration. It allows agents to survive longer during training, thereby obtaining more reward signals. But we found that it would increase the risk of death of agent under evaluation. We did not use it.

Number of NPCs: The baseline increases it from 128 to 256 during training. But we found that the default 128 has enough signals for agents to learn to fight and utilize NPCs. We set it to 128, the same as the evaluation setting.

We also attempted a two-stage training approach. In the first stage, we improved sampling throughput and obtained more reward signals for difficult tasks using these configurations. In the second stage, we continued training under the configurations for evaluation. However, we did not achieve overall better training efficiency and effectiveness. Other game environment configurations for training are identical to the configurations for evaluation. We have not invested time in doing more exploration.

We use PufferLib's Ray parallel sampling on a single machine. The sampling throughput is significantly improved compared to the baseline's Multiprocessing sampling. All hyper-parameters of PPO remain the same as the baseline, except for reducing the epochs per training batch from 3 to 1.

Rewards

The goal of the game is to complete the task assigned to the agent. By default, the game provides small rewards when there is progress in completing the tasks, in addition to the rewards for task completion. These reward signals are dense enough for easy tasks, such as eating. However, for difficult tasks, such as owning items, the rewards are sparse. We grouped tasks according to their categories and observed that it is challenging to make any progress on difficult tasks.

We studied the 3 custom rewards provided by the baseline. The rewards for restoring health and moving in various directions were unnecessary and the performance was significantly better when these two custom rewards were not used. In contrast, the reward for exploration, which encourages the agent to generate more game events such as causing damage, consuming items, buying items, selling items, etc., had a noticeable positive effect on learning across various tasks. We observed that this custom reward could assist the agent in acquiring general skills, especially in terms of utilizing gold, which would reduce the difficulty of obtaining task progress rewards for difficult tasks. Our final version only utilized the exploration reward and the default rewards provided by the game.

We also attempted to introduce additional custom rewards to enhance the learning of difficult tasks. For example, we attempted to introduce task-specific rewards for attacking or harvesting, as these actions are essential for attaining skill. We also attempted to discourage agent from causing damage to players in order to prolong survival time. However, we did not achieve significantly better results.

9.6.3 Saidinesh

Policy Architecture

In our initial exploration of the NMMO challenge, we identified several shortcomings in the baseline model architecture for centralized training and centralized execution of baseline[1]. Notably, the absence of LSTM layers hindered the model's ability to leverage sequential information from previous observations. Additionally, there was a lack of effective coordination among agents, which we believed could greatly enhance task's performance, even though the task are based individual agent's performance level.

One major limitation we observed was the agents' inability to utilize in-game resources, such as gold, for buying or selling items in the market. Recognizing these deficiencies, we embarked on an extensive investigation, exploring various neural network architectures and techniques to address these issues.

We experimented with different approaches, including cross-attention networks, transformer architectures, and even utilizing ResNet-18, ResNet-34, VIT-tiny. However, we encountered challenges with training stability, prompting us to pivot our focus towards refining the architecture and optimizing model performance with different seeds to avoid zero entropy in earlier steps.For our final submission, we made these modifications to the baseline model architecture:

[1]https://github.com/saidineshpola/nmmo-rl

Integration of PyTorch's Multi-Head Attention and Additional Convolutional Layers: We incorporated multi-head attention mechanisms between agent tiles and introduced an additional convolutional layer to deepen the model's understanding of spatial relationships within the environment.

Inclusion of LSTM Layers and Activations: To capture temporal dependencies and leverage sequential information from past observations, we integrated five LSTM layers into the architecture. This enhancement enabled the model to better understand the evolving dynamics of the game environment and make more informed decisions over time. We adopted the SiLU activation function facilitating smoother gradient flow during training.

Multi Head Attention(MHA) Communication Channel: We utilized cross-attention mechanisms not only for action decoding but also as a communication channel [115] between agents for better co-ordination. This facilitated the exchange of task-relevant information among agents, ensuring that each agent was aware of the tasks assigned to team. This allowed agents to exchange relevant information about the task at hand, facilitating more coherent and collaborative decision-making processes.

Reward Tuning and Training

We encouraged desired behaviors by providing bonuses to the agent's reward: 0.00056 for killing enemies, 0.01 (levels 1-5) or 0.02 (levels above 5) for leveling up combat/fishing skills, 0.03 for giving items or gold to other players, 0.01 for harvesting items, and a variable bonus for meandering (exploring through varied movement patterns) based on the entropy of recent moves. These bonuses incentivize combat, trading, resource gathering, and exploration, shaping the agent's behavior towards engaging objectives to complete different tasks.

For training, we used the default baseline configuration with 768 agents from 6 environments. To introduce diversity in the training environments, we created 128 new maps for each training run, using different random seeds to generate varied map layouts.

9.6.4 Mori

Our solution is grounded in the baseline method, with modifications confined exclusively to the reward function and training parameters.

Model Design Approach

Due to task variability, we chose to generalize rewards towards improving agent survival, dividing them into two stages—initially emphasizing positive rewards, then shifting to punishment, with manual design followed by genetic algorithm optimization.

Significantly, we initially aimed to optimize TotalScore, but belatedly discovered CompletedTaskCount as the true metric. By the time of this realization, the contest neared its end, so the second-stage design didn't affect the current results. However, it showed promise in boosting TotalScore performance.

Rewards Design

Given insignificant progress above the baseline in Curriculum Generation, our research focused on Reinforcement Learning, adopting and empirically validating select reward elements from the previous winner realikun's design. In Stage One, the following reward mechanisms were implemented:

Health Recovery Reward: If health increased due to sufficient food and water intake, a reward of 0.02 was allotted; no penalty was imposed otherwise.

Health Point Reward: A reward proportional to the change in health points (0.005 times the change), providing a 0.05 increment for every 10-point increase and vice versa. This reward was considered effective across all tracks and tasks.

Inverted Death Penalty: Initially intended as a penalty upon agent death, a coding error resulted in a seemingly beneficial effect (+0.02 reward) when the agent died.

Task Frequency Reward: A reward of 0.05 was given for each successfully completed task.

Task Progress Reward: The reward (r) equaled the task completion percentage.

Task Completion Reward: A fixed reward of 3 was awarded for task completion.

Alongside these, we refined three baseline rewards, setting the event reward to 3 and the exploration reward to 0.5. In Stage Two and Genetic Algorithm Optimization: Due to time constraints, further testing was not conducted. However, employing the Stage Two rewards and genetically optimized reward values led to improved TotalScores and a more stable agent performance, with approximately a 10% increase observed. The following were found to be ineffective:

Food Scarcity Penalty: Punishment was triggered when food or water levels reached zero.

Step-based Rewards: Phased rewards based on game progression were tested, such as penalizing for food scarcity within the first 500 steps but not thereafter.

Poultice Usage Reward: A reward was for using poultices when the agent had no remaining health.

Regarding Training Methodology Reflections

The random nature of the competition highlights the dependence of model training on seed choice. Given the time-consuming and resource-intensive nature of verifying reward settings through multiple model runs, a practical shortcut involves treating seeds as tunable variables and using automatic search algorithms to find advantageous ones. Although this does not inherently enhance the model's abilities, it proves practically useful in boosting scores.

9.6.5 Jimyhzhu

During our exploration of the environment, we identified challenges such as sparse rewards. We believe using historical data will help with the agents decision making; however, due to the constraint on the evaluation system, this is not viable, so we focus on the optimisation of the policy structure.

Policy Structure

We identified several weaknesses of the baseline network architecture. This includes the lack of pre-processing of categorical data, scaling issue for continuous data, and lack of global information of other agents in the game. As a consequence, we made the following adjustments:

Observation Space Customization: We expanded the network architecture to incorporate global agents information into Observations(Hidden), which is expected to enhance decision-making by providing a summary of the overall state of other agents in the environment.

We introduced one-hot encoding for categorical attributes: Attributes such as "item type id," and "item equipped status" are transformed into a binary vector representation where the index corresponding to a category's value is set to 1, and all other positions are set to 0. We believe this prevents the model from assuming a natural ordering between categories.

Continuous Data Scaling: The baseline transform continuous data of range [0,100] to a range of [0, 10000] making these values several order of magnitude different from other input. We replaced the baseline's upscaling method, with downscaling, normalizing continuous data to a [0,1] range for balanced feature representation.

Average Pooling reduces sensitivity to order: We applied average pooling to market embedding and item embedding instead of using a multi layer perceptron as in baseline before combining them into the Observations. This is beneficial because the strategic significance of entities like items or players often does not depend on their order of appearance in the input data.

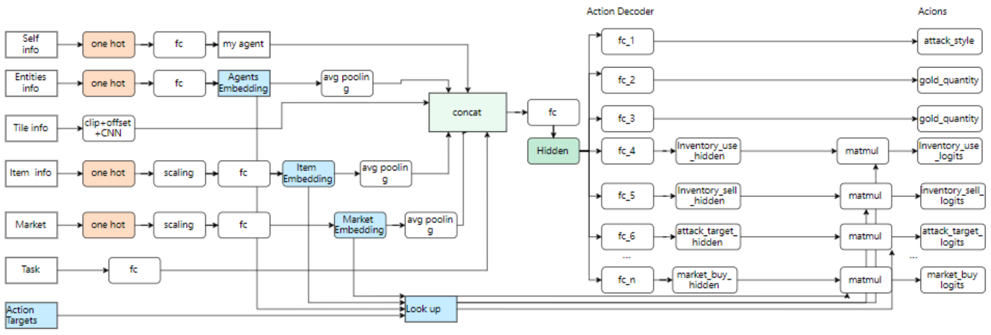


Figure 9.1: Jimyhzhu Final Network Architecture

In Fig. 9.1, we can see that actions are decoded using a specific embedding and the the Hidden state, so we decided to experiment with cross attention in the action decoder. The intuition behind this is that by using the Observation as a query, and specific embedding as key and values, the model can selectively extract information from the specific embedding, that are most relevant to the current state as represented by the Observation. However, no major improvement is shown, possibly because the specific embeddings are not rich and diverse enough to allow the cross-attention mechanism to exhibit its full potential. As a consequence cross attention not used in our final design.

Training Configurations and Reward Design

For training configurations we increased the number of environments from 6 to 8 and doubled the number of maps from 128 to 256 to improve the model's generalization ability to different maps. We used the default reward function due to limited compute resources and long training time to investigate the effect of reward settings.

9.7 Ensuring Fair Evaluations

We collected source code from the top 5 participants who were eligible for prizes. We inspected their implementations manually and retrained the model from scratch for 8 hours on our hardware. The models were trained between 15M to 22M steps, depending on their speed. This was increased from 10M steps used in the original baseline, as we obtained faster hardware in the interim. Four of the top 5 models matched their original ranks and matched or exceeded original training performance. One model (Mori's) did not quite reproduce, but after investigation, we concluded that this was based on seed instability, not manipulation. In any case, the discrepancy in performance was not so substantial to affect awarding of prizes. We further checked training curves and ensured that everything looked reasonable.

Before determining the final ranks, we gave participants to challenge our evaluation results. A few participants requested and received extended evaluation – mostly, they wanted to ensure that we had used their best submission and that it performed consistently. There was some contention between the top 2 policies, and after extended evaluation, we declared a tie as cowinners.

Chapter 10

Conclusions

During the early stages of the project, I focused on figuring out what scientific questions would have to be answered in order to learn in an open-ended many-agent environment. In other words: what fundamental problems prevent Rl from scaling to the real world? The conclusion I came to was that, at least in so far as is required for another large wave of progress, there are none. I did not encounter a single fundamental problem that could not be addressed by building better tools and infrastructure. And despite all of my findings, these are still the major barriers to further progress. The good news is that there is a high degree of certainty: there are no unresolved questions for now, just tools that have yet to be built.

Fewer new tools are required than when Neural MMO began. My work includes general-purpose methods for developing fast, complex simulations that are relatively simple to use for RL research. The structuring of observations and actions, database format for simulation state, and the design principles of the mechanics are all contributions first formalized by Neural MMO. These techniques enable the construction of a variety of different environments of comparable or greater scope.

Given the methods and tools developed over the course of Neural MMO, I would make different considerations if I were to build a new environment today. These largely lean in the direction of taking more inspiration from the games industry and less from artificial life and natural sciences. For example, the core foraging problem of Neural MMO limits what the rest of the environment can implement, since agents are constantly needing to stop what they are doing to collect food and water. Foraging is a simple problem that was essential when learning libraries were slower and more finicky, but it is now largely dated. Having to constantly compete over resources also makes the environment sensitive to initial conditions: in many weaker baselines, agent's just fight over resources near where they spawn. The core survival problem also limits the extent to which other agents will engage with more sophisticated mechanics, such as the exchange system. If there is no immediate benefit to survival, agents will not use it. Tying these two systems together was an unnecessary burden on development.

There are also several core technical decisions I would have made differently. The largest and most obvious is to make the environment have a constant number of agents. There is no theoretical basis for this, but variable sized agent populations are the source of all infrastructure evils. The latest competition ultimately solved this, but the software stack would be substantially simplified if it could simply assume a constant number of agents. I would also make human playability a priority from the start and design the action space around a small set of buttons with no continuous inputs (i.e. no mouse or controller sticks). I would *consider* dropping the tile-based internal representation of game state, since I am reasonably certain that the latest infrastructure would still work. I would also *consider* low-fidelity rendering of observations during training. This is slower for environments up to the complexity of Neural MMO now, but it is probably faster going forwards. One decision I would *not* make is implementing the environment on GPU. This has been a recent trend in RL, but GPU acceleration is only suited to running simple environments fast – not to running intricate simulations with complex branching logic.

Competitions were the other main area of work on Neural MMO. They were designed to test what push the limits of the environment to discover what is possible. Based on the orchestration and analysis of the various different tasks in Neural MMO, the main insight is that competitors solved every problem we threw at them with increasingly general solutions. The first competitions involved heavy scripting; the most recent competition had none. The conclusion is that it is possible to learn both team-based and per-agent policies that capably solve a variety of complex tasks without much compute. The architectures of the winning submissions also reveal the standard playbook of competition submissions, which comprises a bag of tricks that are not generally known within academia.

Of all of the contributions of Neural MMO, the one I am most proud of is this: the project sets a bar on the complexity of tasks and environments that can be studied effectively with minimal compute. Neural MMO both motivates the development of more complex environments and provides methods for building them.

Appendix A

arXiv 2019

A.1 Environment

The environment state is represented by a grid of tiles. We generate the game map by thresholding a Perlin [116] ridge fractal, as shown in Figure A.1. Each tile has a particular assigned material with various properties, but it also maintains a set of references to all occupying entities. When agents observe their local environment, they are handed a crop of all visible game tiles, including all visible properties of the tile material and all visible properties of occupying agents. All parameters in the following subsystems are configurable; we provide only sane defaults obtained via multiple iterations of balancing.

A.1.1 Tiles

We adopt a tile based game state, which is common among MMOs. This design choice is computationally efficient for neural agents and can be made natural for human players via animation smoothing. When there is no need to render the game client, as in during training or test time statistical tests, the environment can be run with no limit on server tick rate. Game tiles are as follows:

- **Grass:** Passable tile with no special properties
- **Forest:** Passable tile containing food. Upon moving into a food tile, the agent gains 5 food and the tile decays into a scrub.
- **Scrub:** Passable tile that has a 2.5 percent probability to regenerate into a forest tile on each subsequent tick
- **Stone:** Impassible tile with no special properties
- **Water:** Passable tile containing water. Upon moving adjacent to a water tile, the agent gains 5 water.
- **Lava:** Passable tile that kills the agent upon contact

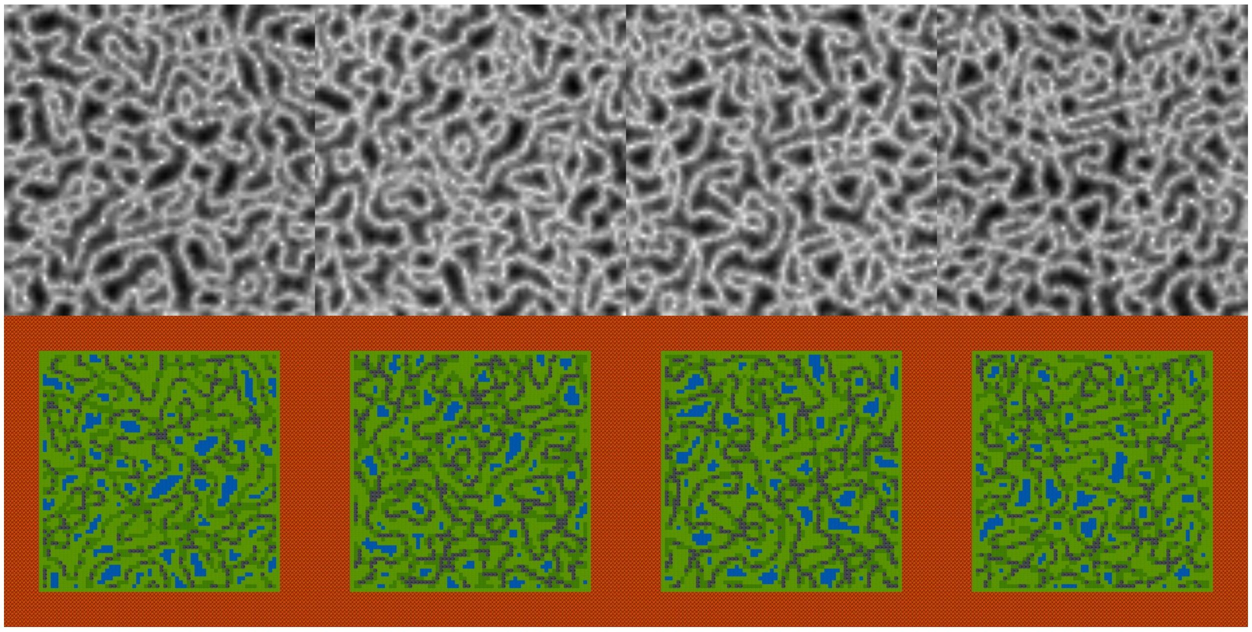


Figure A.1: Procedural 80X80 game maps

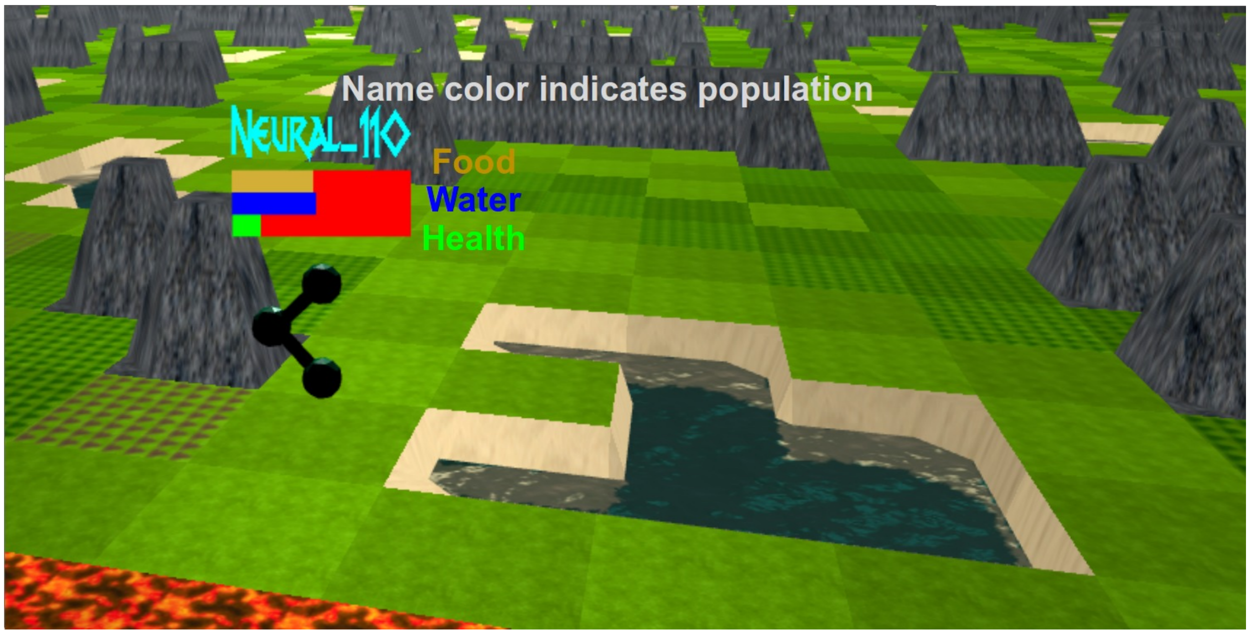


Figure A.2: Example Agent

A.2 Agents

Input: On each game tick, agents (Figure A.2) observe a 15x15 square crop of surrounding game tiles and all occupying agents. We extract the following observable properties:

Per-tile properties:

- **Material**: an index corresponding to the tile type
- **nEnts**: The number of occupying entities. This is technically learnable from the list of agents, but this may not be true for all architectures. We include it for convenience here, but may deprecate it in the future.

Per-agent properties:

- **Lifetime**: Number of game ticks alive thus far
- **Health**: Agents die at 0 health (hp)
- **Food**: Agents begin taking damage at 0 food or water
- **Water**: Agents begin taking damage at 0 food or water
- **Position**: Row and column of the agent
- **Position Deltas**: Offsets from the agent to the observer
- **Damage**: Most recent amount of damage taken
- **Same Color**: Whether the agent is the same color (and thereby is in the same population) as the observer
- **Freeze**: Whether the agent is frozen in place as a result of having been hit by a mage attack

Output: Agents submit one movement and one attack action request per server tick. The server ignores any actions that are not possible or permissible to fulfil, such as attacking an agent that is already dead or attempting to move into stone. *Pass* corresponds to no movement.

Movement:	**North**	**South**	**East**	**West**	**Pass**
Attack:	**Melee**	**Range**	**Mage**		

A.3 Foraging

Foraging (Figure A.3) implements gathering based survival:

- **Food:** Agents begin with 32 food, decremented by 1 per tick. Agents may regain food by occupying forest tiles or by making use of the combat system.
- **Water:** Agents begin with 32 water, decremented by 1 per tick. Agents may regain water by occupying tiles adjacent to water or making use of the combat system.

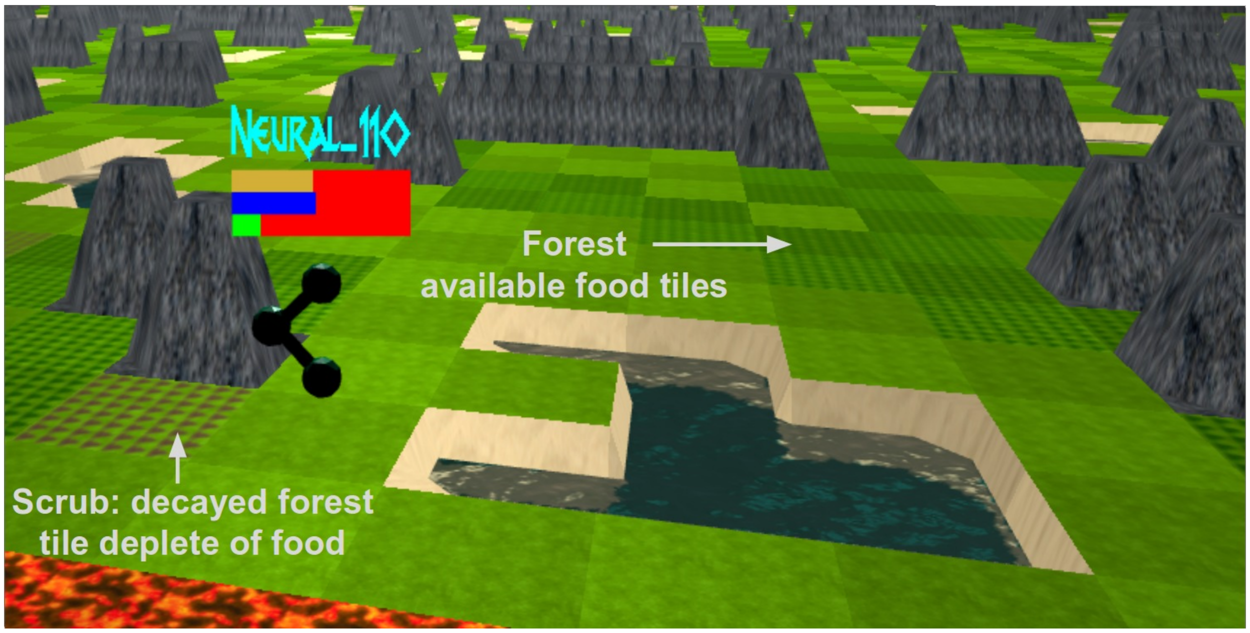


Figure A.3: Example foraging behavior

- **Health:** Agents begin with 10 health. If the agent hits 0 food, they lose 1 health per tick. If the agent hits 0 water, they lose 1 health per tick. These effects stack.

The limited availability of forest (food) tiles produces a carrying capacity. This incurs an arms race of exploration strategies: survival is trivial with a single agent, but it requires intelligent exploration in the presence of competing agents attempting to do the same.

A.4 Combat

Combat (Figure A.4) enables direct agent-agent confrontation by implementing three different attack "styles":

- **Melee:** Inflicts 10 damage at 1 range
- **Ranged:** Inflicts 2 damage at 1-2 range
- **Mage:** Inflicts 1 damage at 1-3 range and freezes the target in place, preventing movement for two ticks

Each point of damage inflicted steals one point of food and water from the target and returns it to the attacker. This serves as an incentive to engage in combat. It is still fully possible for agents to develop primarily foraging based strategies, but they must at least be able to defend themselves. The combat styles defined impose clear but difficult to optimize trade offs. Melee combat fells the target in one attack, but only if they are able to make their attack before the opponent retaliates in kind. Ranged combat produces less risky but more prolonged conflicts. Mage combat does little damage but immobilizes the target, which

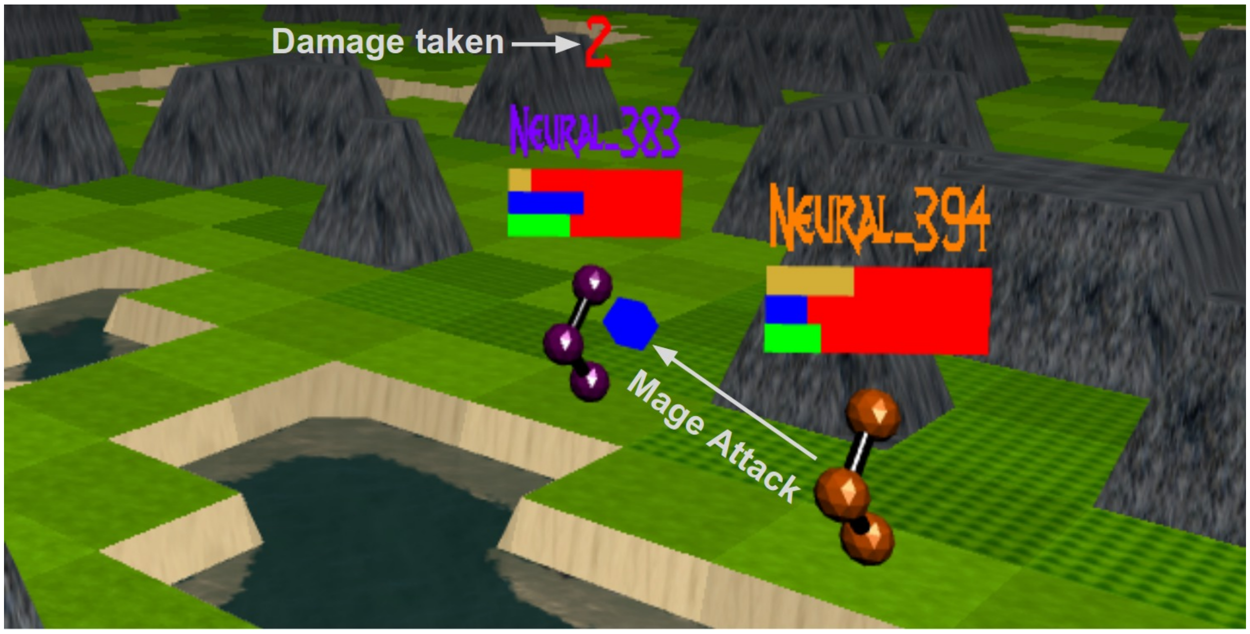


Figure A.4: Example combat behavior

allows the attacker to retreat in favor of a foraging based strategy. More aggressive agents can use mage combat to immobilize their target before closing in for the kill. In all cases, the best strategy is not obvious, again imposing an arms race.

Technical details:

- **Attack range** is defined by l1 distance: "1 range" is a 3X3 grid centered on the attacker.
- **Spawn Killing** Agents are immune during their first 15 game ticks alive. This prevents an exploit known as "spawn killing" whereby players are repeatedly attacked immediately upon entering the game. Human games often contain similar mechanism to prevent this strategy, as it results in uninteresting play.

A.5 API

The initial release is bundled with two APIs for running experiments on our platform. All of our experiments are RL based, but the API implementation is intentionally generic. Evolutionary methods and algorithmic baselines should work without modification.

Gym Wrapper We provide a minimal extension of the Gym VecEnv API [117] that adds support for variable numbers of agents per world and at any given time. This API distributes environment computation of observations and centralizes training and inference. While this standardization is convenient, MMOs differ significantly from arcade games, which are easier to standardize under a single wrapper. The Neural MMO setting requires support for a large, variable number of agents that run concurrently, with aggregation across many randomly generated environments. The Gym API incurs additional communications overhead that the native API bypasses.

Native This is the simplest and most efficient interface. It pins the environment and agents on it to the same CPU core. Full trajectories run locally on the same core as the environment. Interprocess communication is only required infrequently to synchronize gradients across all environments on a master core. We currently do the backwards pass on the same CPU cores because our networks are small, but GPU is fully supported.

A.6 Client

The environment visualizer comes bundled with research tools for analyzing agent policies. In the initial release, we provide both a 2D python client (Figure A.5) and a 3D web client (Figure A.6, A.8). The 3D client has the best support for visualizing agent policies. The 2D client is already deprecated; we include it only because it will likely take a few weeks to fully finish porting all of the research tools. We include visualization tools for producing the following visualization maps; additional documentation is available on the project Github:

- Value ghosting
- Exploration
- Interagent dependence
- Combat

A.7 Policy training and architecture

Parameters relevant to policy training are listed in Table 1. The neural net architecture, shown in Figure A.7, is a simplest possible fully connected network. It consists of a preprocessor, main network, and output heads. The preprocessor is as follows:

- Embed indicies corresponding to each tile into a 7D vector. Also concatenates with the number of occupying entities.
- Flatten the tile embeddings
- Project visible attributes of nearby entities to 32D
- Max pool over entity embeddings to handle variable number of observations
- Concatenate the tile embeddings with the pooled entity embeddings
- Return the resultant embedding

The main network is a single linear layer. The output heads are also each linear layers; they map the output hidden vector from the main network to the movement and combat action spaces, respectively. Separate softmaxes are used to sample movement and combat actions.

Technical details

- For foraging experiments, the attack network is still present for convenience, but the chosen actions are ignored.

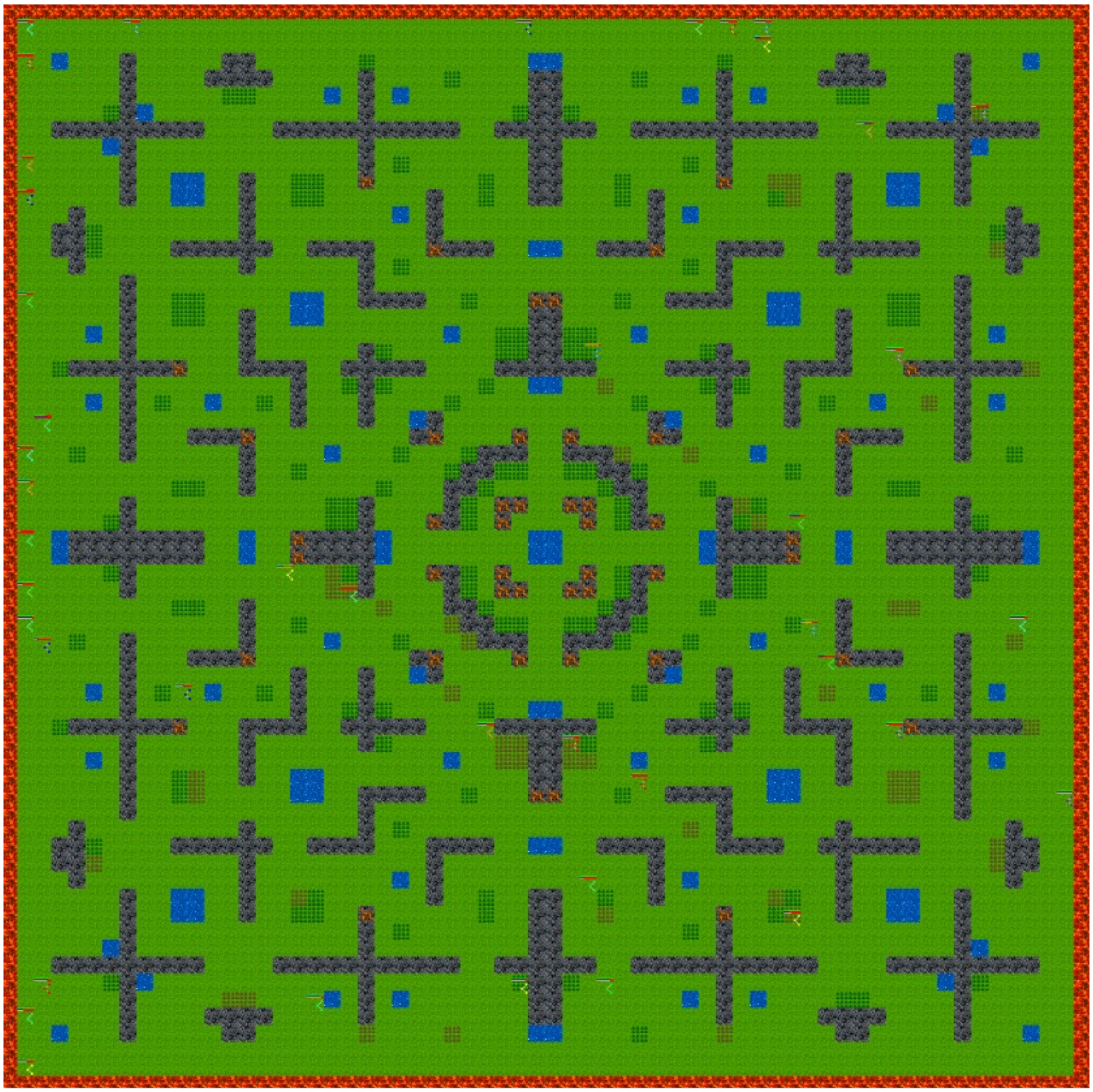


Figure A.5: Example map in the 2D client

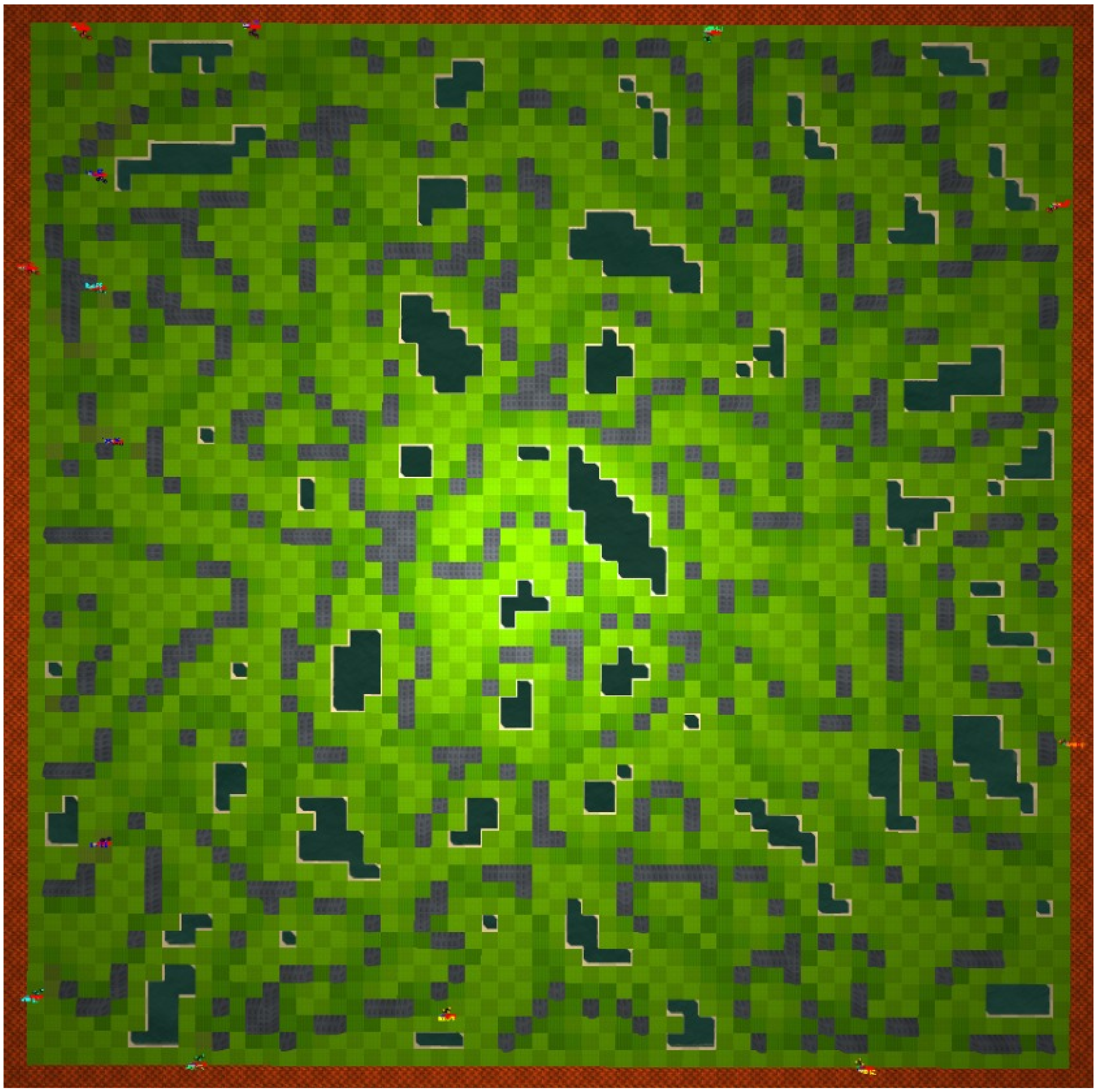


Figure A.6: Example overhead map view in the 3D client

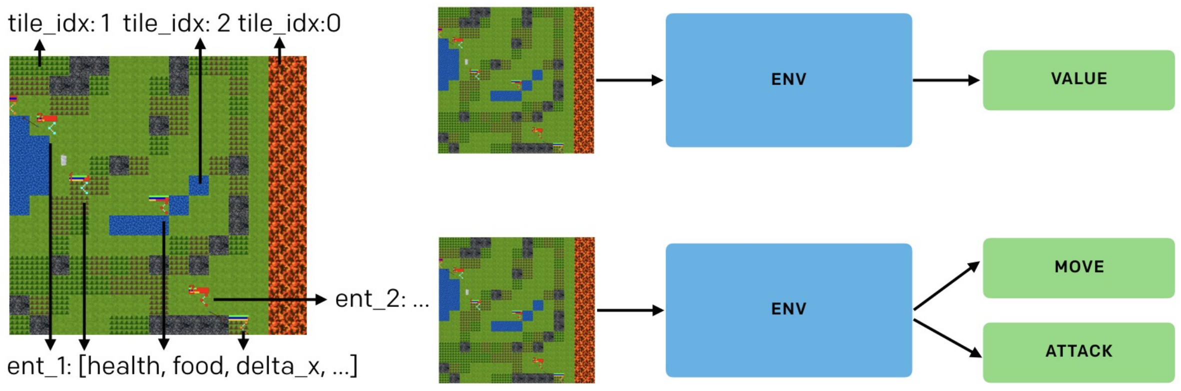


Figure A.7: Agents observe their local environment. The model embeds this observations and computes actions via corresponding value, movement, and attack heads. These are all small fully connected networks with 50-100k parameters.

Table A.1: Training details and parameters for all experiments

Parameter	Value	Notes
Training Algorithm	Policy Gradients[119]	+ Value function baseline
Adam Parameters	lr=1e-3	Pytorch Defaults
Weight Decay	1e-5	Training stability is sensitive to this
Entropy Bonus	1e-2	To stabilize training; possibly redundant
Discount Factor	0.99	No additional trajectory postprocessing

- Note that 1D max pooling is used to handle the variable number of visible entities. Attention [118] may appear the more conventional approach, but recently [2] demonstrated that simpler and more efficient max pooling may suffice. We are unsure if this is true at our scale, but used max pooling nonetheless for simplicity.

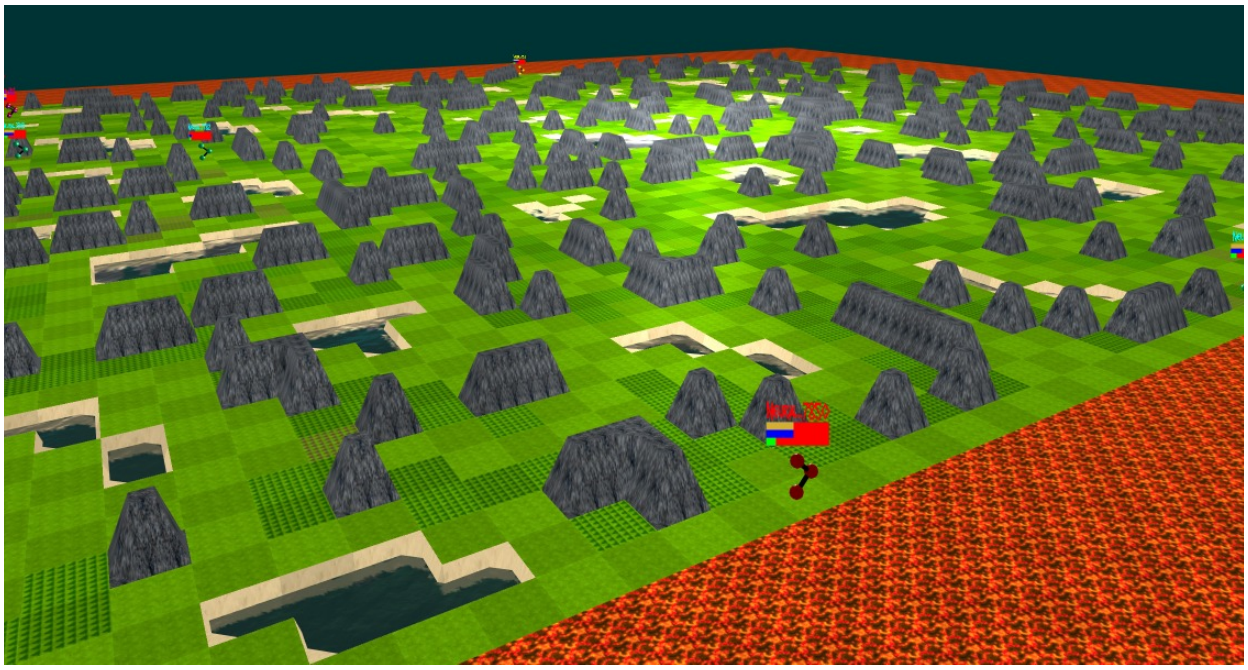


Figure A.8: Perspective screenshot of 3D environment

Appendix B

AAMAS 2020 Extended Abstract

This excerpt from the AAMAS 2020 extended abstract covers the same IO problem introduced in the arXiv 2020 paper, but it provides a more mathematical description that will be clearer for some readers. It is included for the sake of completeness on one of the more important technical problems addressed in my work.

B.1 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).

B.1.1 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 $e_{x_1}, e_{x_2}, ...$ 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
$Z = \{f_{y_j}(\{x_{1,...,n}\} \subseteq Y)\}$	Soft attend f_{y_i} to attribute subsets
$o = g(Z)$	Soft attend g to entity embeddings

At run time, we project $x_1, x_2, ...$ to fixed length attribute embedding vectors $y_1, y_2, ...$ 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.

B.1.2 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 $\rightarrow$ [North] or Attack $\rightarrow$ [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*
$A :- [A_i : [a_1, ..., a_n], ...]$	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.

Appendix C

NeurIPS 2021 Datasets and Benchmarks

C.1 Training

We use RLlib's default PPO implementation. This includes a number of standard optimizations such as generalized advantage estimation and trajectory segmentation with value function bootstrapping. We did not modify any of the hyperparameters associated with learning. We did, however, tune a few of the training scale parameters for memory and batch efficiency. On small maps, batches consist of 8192 environment steps sampled in fragments of 256 steps from 32 parallel rollout workers. On large maps, batches consist of 512 environment steps sampled in fragments of 32 steps from 16 parallel workers. Each rollout worker simulates random environments sampled from a pool of game maps. The optimizer performs gradient updates over minibatches of environment steps (512 for small maps, 256 for large maps) and never reuses stale data. The BPTT horizon for our LSTM is 16 timesteps.

Table C.1: Training hyperparameters

Parameter	Value	Description
batch size	512/8192	Learning batch size
minibatch	256/512	Learning minibatch size
sgd iters	1	Optimization epochs per batch
frag length	32/256	Rollout fragment length
unroll	16	BPTT unroll horizon
λ	1.0	Standard GAE parameter
kl	0.2	Initial KL divergence coefficient
lr	5e-5	Learning rate
vf	1.0	Value function loss coefficient
entropy	0.0	Entropy regularized coefficient
clip	0.3	PPO clip parameter
vf clip	10.0	Value function clip parameter
kl target	0.01	Target value for KL divergence

C.2 Population Size Magnifies Exploration

As described in the main text, agents trained in larger populations learn to survive for longer and explore more of the map.

Population	Lifetime	Achievement	Player Kills	Equipment	Explore	Forage
4	89.26	1.40	0.00	0.00	7.66	17.36
32	144.14	2.28	0.00	0.00	11.41	19.91
256	227.36	3.32	0.00	0.00	15.44	21.59

Table C.2: Accompanying statistics for Figure 4.5.

C.3 The REPS Measure of Computational Efficiency

Formatted equation accompanying our discussion of efficient complexity on the project site

$$\textbf{Real-time experience per second} = \frac{\textbf{Independently controlled agent observations}}{\textbf{Simulation time} \times \textbf{Real time fps} \times \textbf{Cores used}} \tag{C.1}$$

C.4 Architecture

Our architecture is conceptually similar to OpenAI Five's: the core network is a simple one-layer LSTM with complex input preprocessors and output postprocessors. These are necessary to flatten the complex environment observation space and compute hierarchical actions from the flat network hidden state.

The input network is a two-layer hierarchical aggregator. In the first layer, we embed the attributes of each observed game object to 64 dimensional vector. We concatenate and project these into a single 64-dimensional vector, thus obtaining a flat, fixed-length representation for each observed game object. We apply self-attention to player embeddings and a conv-pool-dense module to tile embeddings to produce two 64-dimensional summary vectors. Finally, we concat and project these to produce a 64-dimensional state vector. This is the input to the core LSTM module.

The output network operates over the LSTM output state and the object embeddings produced by the input network. For each action argument, the network computes dot-product similarity between the state vector and candidate object embeddings. Note that we also learn embeddings for static argument types, such as the north/south/east/west movement direction options. This allows us to select all action arguments using the same approach. As an example: to target another agent with an attack, the network computes scores the state against the embedding of each nearby agent. The target is selected by sampling from a softmax distribution over scores.

Table C.3: Architecture details

Parameter	Value	Description
		Encoder
discrete	range$\times$64	Linear encoder for n attributes
continuous	$n\times$64	Linear encoder for n attributes
objects	64$n\times$64	Linear object encoder
agents	64$\times$64	Self-attention over agent objects
tiles	conv3-pool2-fc64	3x3 conv, 2x2 pooling, and fc64 projection over tiles
concat-proj	128$\times$64	Concat and project agent and tile summary vectors
		Hidden
LSTM	64	Input, hidden, and output dimension for core LSTM
		Decoder
fc	64$\times$64	Dimension of fully connected layers
block	fc-ReLU-fc-ReLU	Decoder architecture, unshared for key/values.
decode	block(key) $\cdot$ block(val)	state-argument vector similarity
sample	softmax	Argument sampler over similarity scores

Appendix D

ICLR 2022 ALOE Workshop

D.0.1 Specialist vs. Generalist Simulation Details

We ran 20 total simulations of 1000 steps each – 10 for both specialist and generalist policies. Each simulation is around 10 minutes of real-time play and contains around 100 agents (the precise number is variable and depends on how long agents survive).

We collect experience, tool level, and equipment score data for each profession in each simulation. In each simulation, we then take the max profession level across agents at each time step. Finally, we average across parallel simulations. The quantities plotted below are therefore the average maximum values obtained at each point in simulation. We chose these metrics to indicate how well, on average, the best agent in each profession performs. Figure 4 further averages over all professions, but the per-profession breakdown for both generalists and specialists is shown in Figures 5 and 6. This reveals that, despite higher initial performance, fisher and hunter levels actually drop off later in simulation for specialist policies. The generation patterns for the resources corresponding to fishing and hunting are slightly harder to gather than those in other professions, and our scripted exploration algorithm is not particularly good. In seeking to also gather other resources, generalist policies effectively explore more. A better exploration policy would likely eliminate this anomaly. Even with this oversight, specialists still surpass generalists in aggregate on all three metrics.

D.0.2 Event Logs

Our logging records "interesting" events that occur for the first time during any given simulation. They provide another way of interpreting agent progress throughout the course of a simulation. We will be incorporating them into a user-friendly dashboard in the full release of our profession and exchange systems. Below are the first, middle, and final sections of logs for one environment simulation.

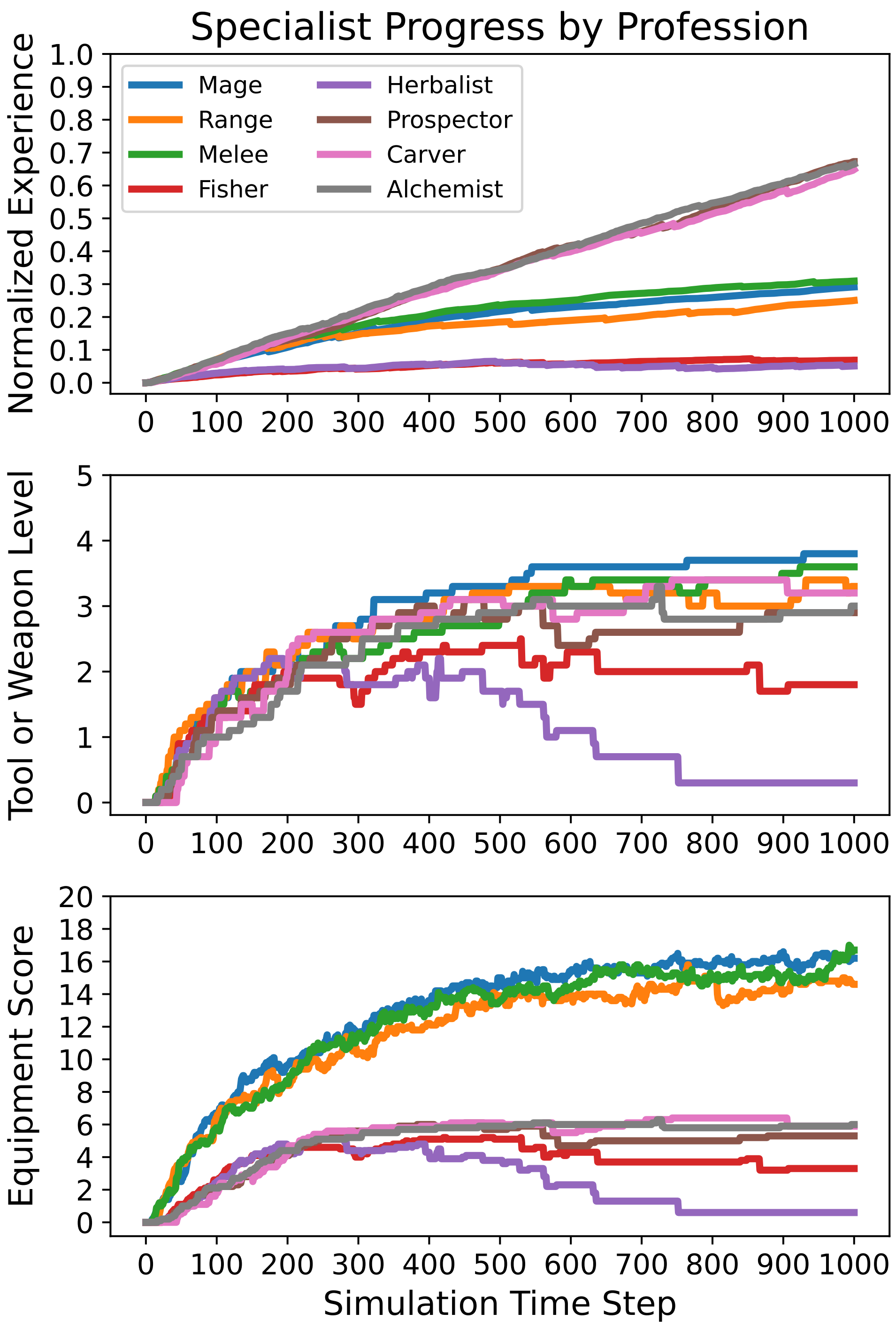


Figure D.1: Per-profession specialist performance metrics. All but two professions improve across all metrics over the course of simulation. See above for possible explanations.

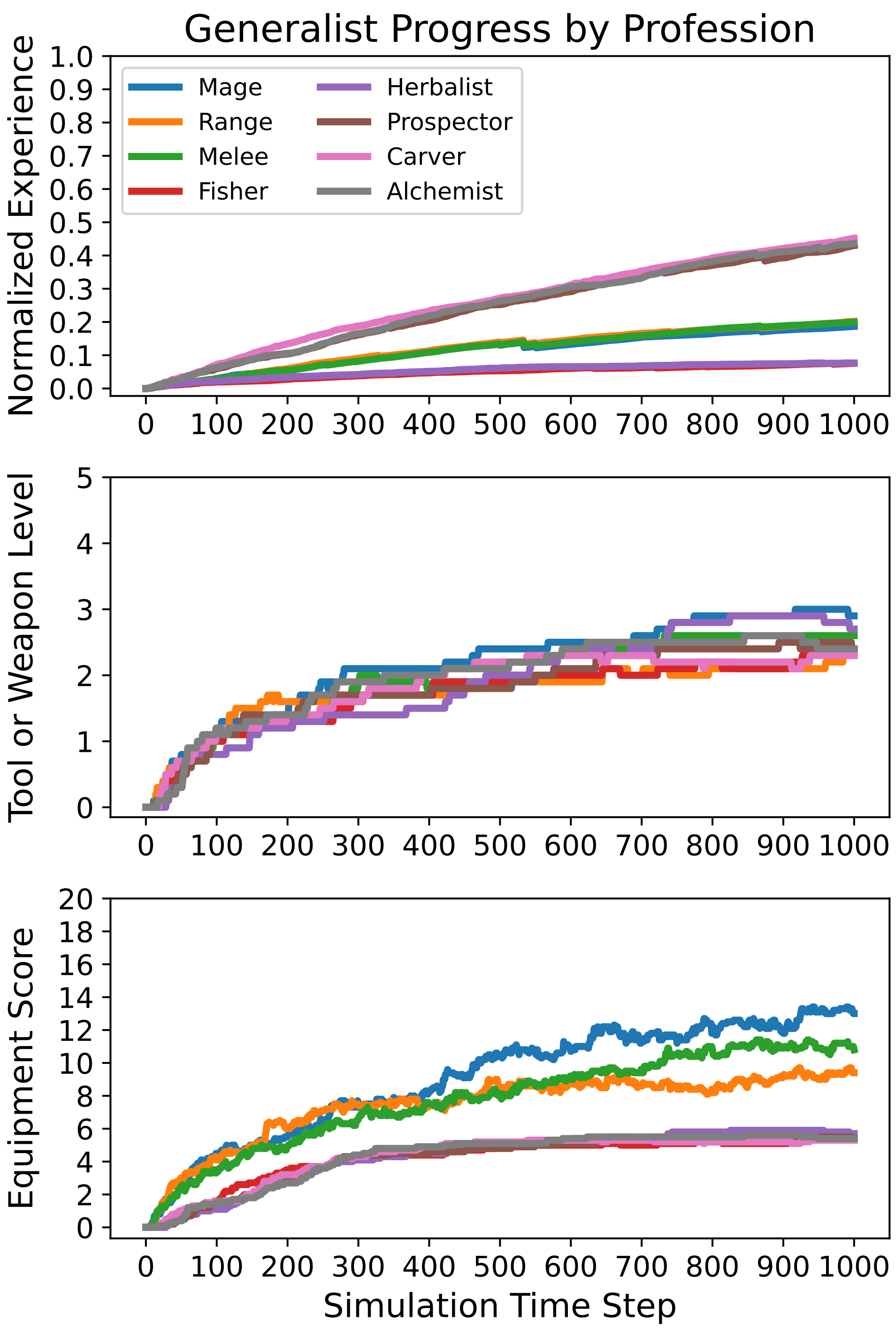


Figure D.2: Per-profession generalist performance metrics. All but two professions underperform the corresponding specialists over the course of simulation. See above for possible explanations.

```
Beginning
INFO:RESOURCE: Depleted 0 resource tiles
INFO:RESOURCE: Depleted 1 resource tiles
INFO:RESOURCE: Depleted 3 resource tiles
INFO:COMBAT: Inflicted 25 Mage damage (lvl 1 i0 vs lvl 1 i1)
INFO:PROGRESSION: Reached level 1 Mage
INFO:RESOURCE: Depleted 5 resource tiles
INFO:PROFESSION: Gathered level 1 Ration (level 1 Fishing)
INFO:PROGRESSION: Reached level 1 Fishing
INFO:COMBAT: Inflicted 30 Range damage (lvl 1 i0 vs lvl 1 i1)
INFO:PROGRESSION: Reached level 1 Range
INFO:RESOURCE: Depleted 8 resource tiles
INFO:COMBAT: Inflicted 25 Melee damage (lvl 1 i0 vs lvl 1 i1)
INFO:PROGRESSION: Reached level 1 Melee
INFO:EXCHANGE: Total wealth 16 gold
INFO:RESOURCE: Depleted 12 resource tiles
INFO:PROFESSION: Gathered level 1 Poultice (level 1 Herbalism)
INFO:PROGRESSION: Reached level 1 Herbalism
INFO:RESOURCE: Depleted 16 resource tiles
INFO:EQUIPMENT: Equipped level 1 Hat
INFO:EQUIPMENT: Item level 1

Middle
INFO:PROGRESSION: Reached level 5 Carving
INFO:EQUIPMENT: Item level 13
INFO:EXCHANGE: Offered level 3 Bottom for 4 gold
INFO:EXCHANGE: Bought level 3 Chisel for 4 gold
INFO:EQUIPMENT: Equipped level 3 Chisel
INFO:PROFESSION: Gathered level 3 Ration (level 3 Fishing)
INFO:PROGRESSION: Reached level 5 Alchemy
INFO:EXCHANGE: Offered level 4 Wand for 5 gold
INFO:EXCHANGE: Offered level 3 Pickaxe for 4 gold
INFO:EXCHANGE: Bought level 4 Wand for 5 gold
INFO:EQUIPMENT: Equipped level 4 Wand
INFO:EQUIPMENT: Item level 15
INFO:EQUIPMENT: Mage attack 70
INFO:EXCHANGE: Bought level 3 Pickaxe for 4 gold
INFO:EQUIPMENT: Equipped level 3 Pickaxe
INFO:COMBAT: Inflicted 142 Mage damage (lvl 4 i15 vs lvl 4 i1)
INFO:RESOURCE: Depleted 1167 resource tiles
```

```
End
INFO:EQUIPMENT: Equipped level 5 Top
INFO:EXCHANGE: Offered level 4 Bottom for 5 gold
INFO:EXCHANGE: Bought level 5 Bow for 6 gold
INFO:EXCHANGE: Transaction of 6 gold (level 5 Bow)
INFO:RESOURCE: Depleted 1519 resource tiles
INFO:EQUIPMENT: Equipped level 5 Bow
INFO:EXCHANGE: Bought level 4 Bottom for 5 gold
INFO:RESOURCE: Depleted 1524 resource tiles
INFO:EQUIPMENT: Item level 22
INFO:EQUIPMENT: Range attack 95
INFO:EXCHANGE: Bought level 4 Sword for 5 gold
INFO:PROGRESSION: Reached level 7 Carving
INFO:PROGRESSION: Reached level 7 Alchemy
INFO:COMBAT: Inflicted 142 Melee damage (lvl 6 i15 vs lvl 2 i0)
INFO:EXCHANGE: Offered level 5 Hat for 6 gold
INFO:COMBAT: Inflicted 165 Melee damage (lvl 5 i15 vs lvl 1 i0)
INFO:COMBAT: Inflicted 150 Mage damage (lvl 5 i14 vs lvl 1 i1)
```

Appendix E

NeurIPS 2022 Competition

E.1 Environment Setting for NeurIPS Neural MMO Challenge

E.1.1 Environment Wrapper

To make the environment more extensible and participant-friendly, we have incorporated some new functions in addition to the original environment. These changes include:

1. Random assignment of team numbers and spawns, which allows teams start at different initial positions on the map after multiple evaluations. This is helpful for addressing asymmetry in evaluation of participants' algorithms.

2. Aggregation of observations of single agents at the team level, allowing each agent to have the information of other teammates. Our approach is different from OpenAI Five and AlphaStar as it does not strictly conform to the MARL definition within an individual team.

E.1.2 Tracks

As shown in Fig. E.1, the competition is divided into two tracks: PvE and PvP. In the PvE track, participants' policies are evaluated against 15 built-in policies (i.e. copies of policies that we trained), while in the PvP track, they are evaluated against policies submitted by 15 other participants. From a competitor's perspective, the PvE track is designed to help players quickly familiarize themselves with the environment against a fixed set of opponents, while the PvP track is intended to increase the competitiveness of the game. The structure of the different tracks is as follows:

PvE Stage 1 vs Scripted AI

In this stage, participants' policy will be thrown into a game with 15 scripted built-in AI teams and evaluated for 10 rounds. The result of each submission will be presented as a Top 1 Ratio (see section E.1.3). Each built-in AI team (Mixture Team and Combat Team) in

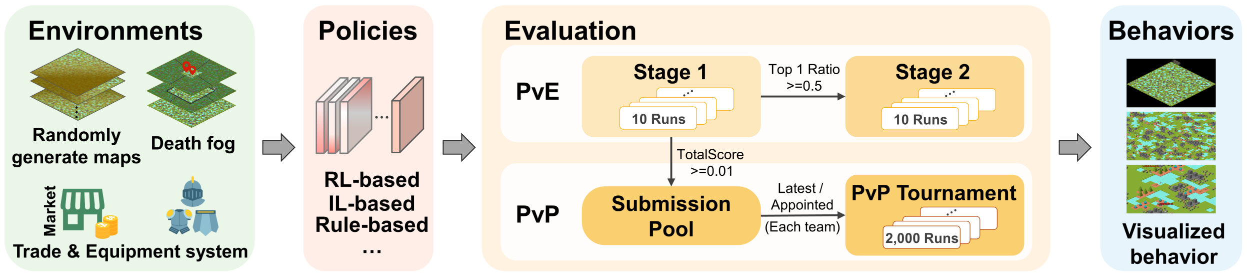


Figure E.1: An overview of the track structure and evaluation process of the competition.

stage 1 consists of 8 agents with 8 respective skills. These policies are open-sourced so that an evaluation environment is accessible during training. Each member of the mixture team specializes in a specific skill. The combat team is hostile and will attack every agent close to them. The aim of this track is to help new participants get familiar with the challenge.

PvE Stage 2 vs RL-based AI

To qualify for Stage 2 in the PvE track, a participant must achieve a 0.4 Top 1 Ratio in Stage 1. In Stage 2, the rules are the same as in Stage 1, except that the built-in AIs are designed by the organizers and trained using a deep reinforcement learning framework developed by Parametrix.AI. This stage is intended to be more challenging than Stage 1, as the built-in AIs are much stronger, and defeating them is a significant achievement. There are three types of built-in AI teams in Stage 2:

1. Reckless: These bots are skilled in combat and also in using, selling, and buying items.
2. Ruthless: An evolution of the Reckless policy, Ruthless bots are both hostile and cooperative. They will fight enemies independently but also recognize the importance of team cooperation.
3. Coward: These bots are risk-adverse and avoid combat with high-level NPCs. They move quickly to the edges of the map to ensure a survival advantage and snipe incoming opponents for points.

PvP

In this stage, each participant's policy is pitted against the policies sampled from the pool of qualifying submissions. Given the number of submissions from different teams employing different approaches, this procedure is an effective assessment of robustness and generalization. To ensure an accurate assessment, each participant will play at least 1000 matches.

E.1.3 Metrics

Score

The participants' policy score is determined by the number of other agents (excluding NPCs) they defeat (i.e., inflict the final blow) and how long at least one team member remains alive.

Each agent defeated earns 0.5 points for the participants' policy, and the final defeat score is the sum of the per-agent defeat scores across the team. The team's survival time is determined by when the last agent dies. The survival scores in each tournament range from 10 to 0. Specifically, the survival scores in each tournament from high to low are 10, 6, 5, 4, 3, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, and will be assigned to the team surviving the longest, the second longest, the third longest, and so on respectively. In the case of tied survival times, the team with more survivors ranks higher. If the number of survivors is equal, the average level of the agents is taken into account. If all factors are still equal, the tied teams will split the ranking scores. For example, if a policy lands the last hit on 5 agents and survives the longest, they receive 5*0.5+10 = 12.5 points.

Top 1 Ratio in PvE

The PvE track's fixed opponents offer a consistent performance metric for both scripted and reinforcement learned submissions. A Top 1 Ratio of 1.0 over 10 games (i.e. 10 wins in a row) indicates that the submission is significantly better than the built-in AI. This makes Top 1 Ratio a reasonable first screen of the performance of competitors.

TrueSkill in PvP

Top 1 Ratio is a poor indicator of performance in PvP evaluations against unknown opponents. We instead use a TrueSkill scoring mechanism that takes into account the ranking of all policies in a match, not just the winner. In other words, even if a strategy loses first place in a competition, its ranking under TrueSkill can still improve.

E.1.4 Resources

Starter Kit

This project includes environment installation instructions, submission demos, and FAQs for common problems to ensure that participants have a positive experience. Everything necessary for participants to submit their work is included.

Baseline

To accommodate the limited device resources of most participants, we have developed a baseline using torchbeast that is available to all participants. This baseline enables participants to achieve a 0.5 Top 1 ratio on a PC with a GPU within one day.

Environment Documentation

The documentation provided by the Neural MMO website is intended for a broader audience of researchers using the base Neural MMO platform for a variety of research agendas. This makes it difficult for competitors to get the information they need in a short time. To help participants get started quickly, we have published a competition-specific document containing important information about the environment.

Web Viewer

We have created a browser-based video replay tool that allows participants to view their policy performance in terms of movement, collection, fighting, and trading, enabling them to make better policy adjustments. The web viewer includes different perspectives, controllable playback speed, and detailed information for display, as shown in Fig. E.2.

Evaluation System

We have designed a distributed evaluation system based on k8s to handle large-scale submissions and fully evaluate the performance of each player's policy. During the PvE phase, each player's strategy will compete against built-in AI on a randomly generated map. In the PvP phase, all players in PvE stage 2 will compete against each other. For the PvP track, we provide both daily and weekly evaluations. Daily evaluations run only 100 matches against other participants. This functions as a more frequent feedback signal to help participants improve their policies. We also run a more extensive 1000 match evaluation once per week. As the performance of a policy is always relative opponents, this system helps to provide an accurate evaluation of player strategies.

E.2 Policy Analysis

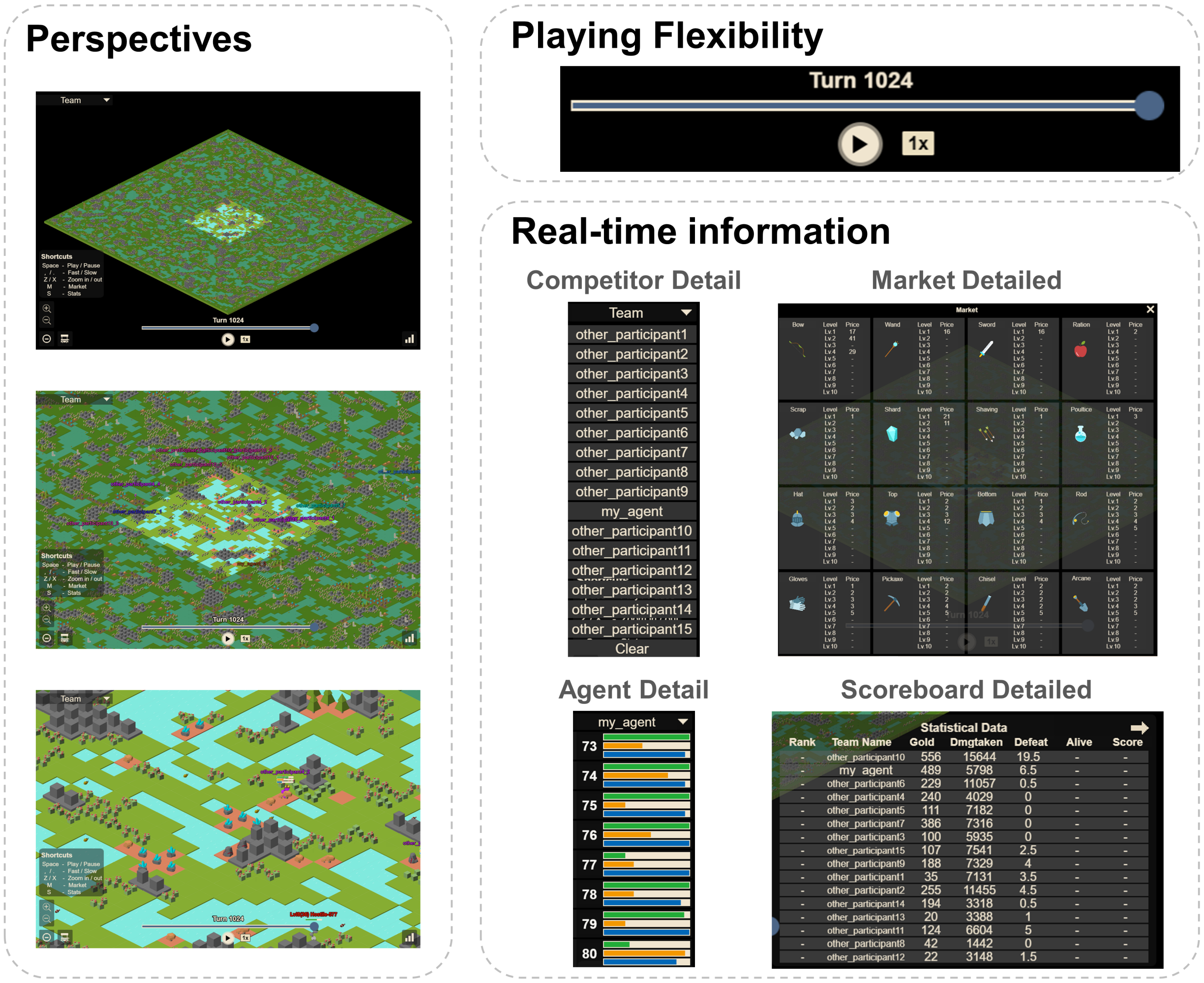
Different Types Algorithms

We collected over 1600 policies and sorted participants into three categories based on the algorithms they used: rule-based, RL-based, and IL-based. Fig. E.3 shows a comparison between these three types of algorithms. As depicted, most participants chose the RL algorithm for the challenge, and some of them achieved a top 16 ranking. However, a few participants submitted scripted policies, and one of the IL submissions achieved 5th place.

Overall, RL took participants longer to get up and running but delivered higher performance by the end of the competition. The theoretical upper bound of IL-based learning is close to that of RL but is limited by the quality of the data (which, in this case, is generated by an RL policy because human data is unavailable). Rule-based methods allow participants to control the actions of agents directly. We noticed that scripted approaches were quick to develop and effective at the start of the competition, but their performance fell off compared to RL by the end, and they were overly adapted to the opponents of the first two PvE stages.

Different Tracks

The main challenges of this competition are robustness and generalization to new opponents in a complex environment. In the NeurIPS-2022 Neural MMO challenge, we set up two different tracks that can fully test the robustness and generalization of participants' algorithms. As shown in Figure E.4, the top 3 participants in PvE stage 1 all demonstrate good robustness and generalization. Realikun consistently wins in all tracks, while the rankings of the other two players remain steady. However, for most other participants, their rankings change continually. For example, the Vanilla algorithm is weak on generalization, performing well



Team
other_participant1
other_participant2
other_participant3
other_participant4
other_participant5
other_participant6
other_participant7
other_participant8
other_participant9
my_agent
other_participant10
other_participant11
other_participant12
other_participant13
other_participant14
other_participant15
Clear

Market

Level	Bow	Wand	Sword	Ration	Scrap	Shard	Shaving	Poultice	Hat	Top	Bottom	Rod	Gloves	Pickaxe	Chisel	Arcane
Lv.1	17	16	16	2	1	21	1	3	3	2	1	2	1	2	2	2
Lv.2	41	-	-	-	-	11	-	-	2	2	2	2	2	2	2	2
Lv.3	-	-	-	-	-	-	-	-	3	3	3	3	3	2	2	2
Lv.4	29	-	-	-	-	-	-	-	4	12	4	4	3	4	2	3
Lv.5	-	-	-	-	-	-	-	-	-	-	-	5	5	5	5	5
Lv.6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Lv.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Lv.8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Lv.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Lv.10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Statistical Data

Rank	Team Name	Gold	Dmgtaken	Defeat	Alive	Score
-	other_participant10	556	15644	19.5	-	-
-	my_agent	489	5798	6.5	-	-
-	other_participant6	229	11057	0.5	-	-
-	other_participant4	240	4029	0	-	-
-	other_participant5	111	7182	0	-	-
-	other_participant7	386	7316	0	-	-
-	other_participant3	100	5935	0	-	-
-	other_participant15	107	7541	2.5	-	-
-	other_participant9	188	7329	4	-	-
-	other_participant1	35	7131	3.5	-	-
-	other_participant2	255	11455	4.5	-	-
-	other_participant14	194	3318	0.5	-	-
-	other_participant13	20	3388	1	-	-
-	other_participant11	124	6604	5	-	-
-	other_participant8	42	1442	0	-	-
-	other_participant12	22	3148	1.5	-	-

Figure E.2: An overview of web viewer. Left: various zoom levels. Top Right: draggable timeline. Bottom Right: Detailed statistics about the agent and market states.

in PvE stage 1 but poorly in PvE stage 2 and PvP stage. These observations demonstrate that this environment can facilitate the evaluation of policy robustness and generalization by introducing different opponents. They also illustrate that when a policy is robust enough, it is able to outperform all other policies in the environment.

Different Combat Preferences

After analyzing Fig. 7.4, we observed that even strategies with very close total scores behaved differently. We focus on the analysis of the strategies used by the top three players. As depicted in the figure, it is evident that realikun aimed to obtain higher points by defeating more participants. Typically, most participants believe that the maximum survival score is 10 points and that achieving this score will create a significant lead over their opponents. As a result, most participants tend to be relatively conservative and avoid fighting aggressively to attain this score. However, if most participants are conservative, they may be at a disadvantage when fighting becomes inevitable later in the game. This allows a more aggressive policy to become substantially stronger than most participants, ensuring that it can defeat all other agents and survive to the end. Evidently, realikun is based on this idea and successfully won first place in all tracks of the competition. In contrast to realikun, doublez avoids unnecessary fighting (rarely fighting powerful NPCs in the environment), while passerby's agents are more aggressive. Passerby has more combat points than doublez, but it takes more damage and has lower survival rate, as shown in the figure.

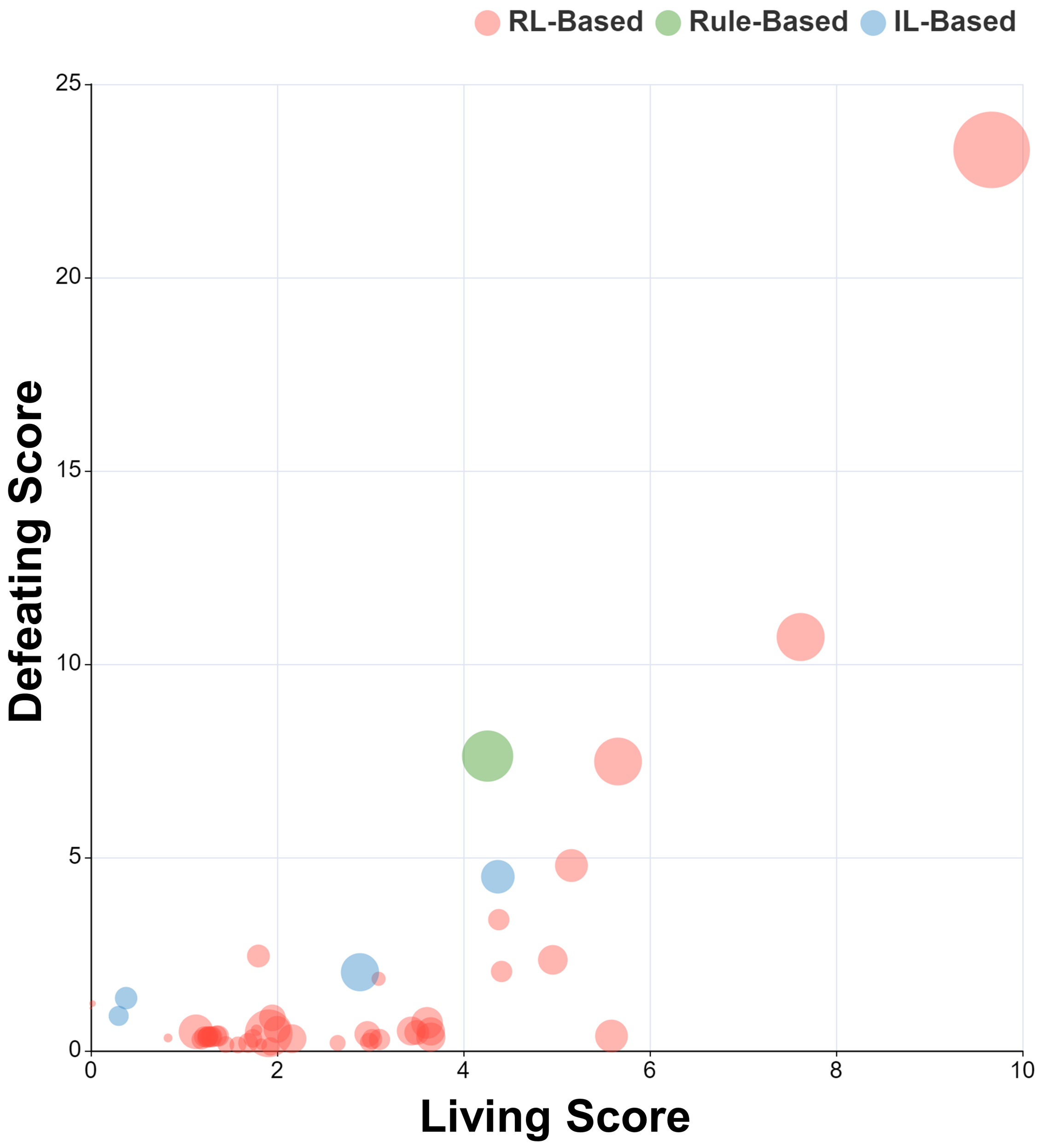


Figure E.3: Performance of different type algorithms in the NeurIPS-2022 Neural MMO challenge PvP stage. Every point in the chart denotes a participant. The size of the point represents the total number of gold obtained by that participant.

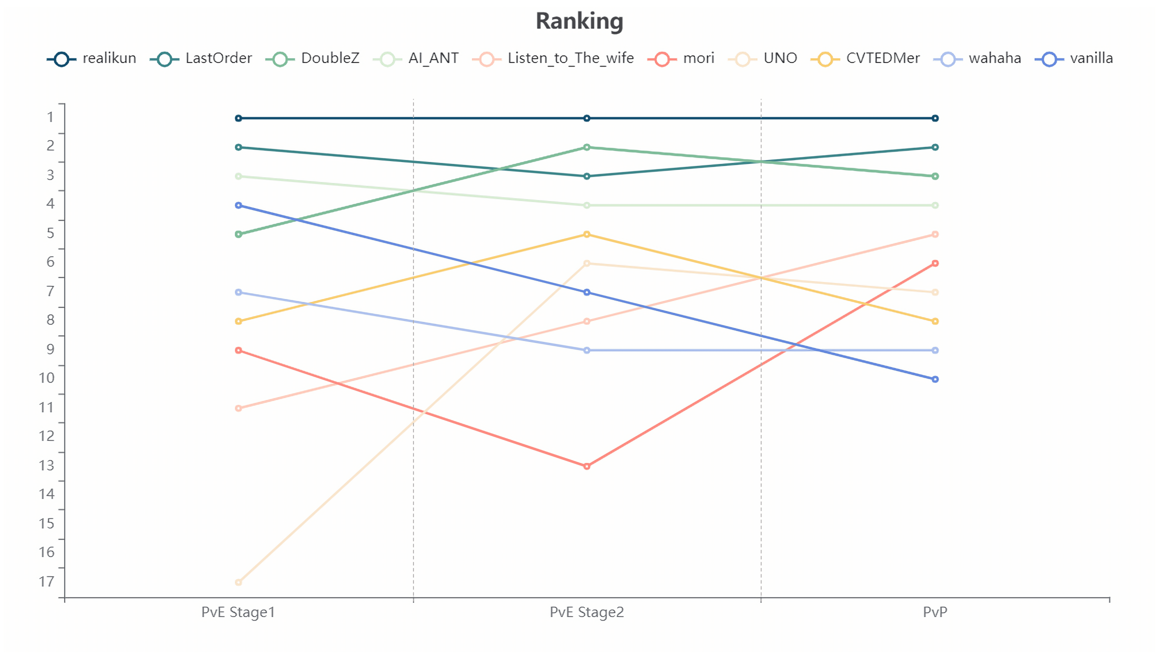
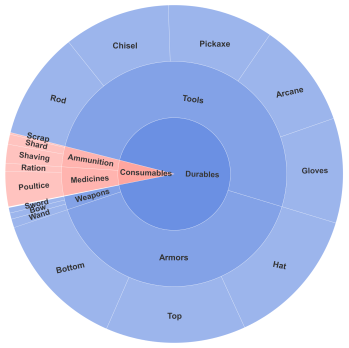
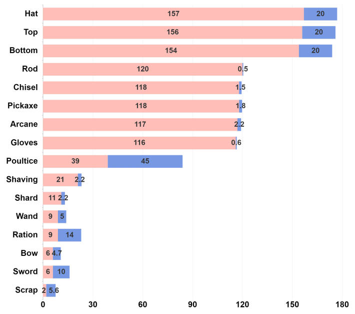


Figure E.4: The ranking of participants in different tracks.

Trade Overview

Figure **??** (a) shows the distribution of items in the environment, which has numerous tools and few consumables. Additionally, Figure **??** (b) depicts the amount of items, revealing that the total number of items of the same type is comparable, indicating the environment's setting is reasonable. Based on this, we can deduce that different items hold different importance for agents, as shown in Fig. **??** (c). Agents prioritize owning weapons and restoration potions rather than selling them to the global market. However, since agents' inventory is constrained, they need to trade their some supplies in the global market to make space. As seen in Fig. E.6, we observe some interesting trading strategies: (1) For universally preferred items (such as weapons), agents' offers usually increase with the level (difficulty to obtain) of the weapon. (2) Similarly, the value of such as tools also increases with their level, but if agents realize the expected price is too high to sell top-level supplies, they may reduce the price for trading.


(a) Distribution of items


(b) Quantity of items

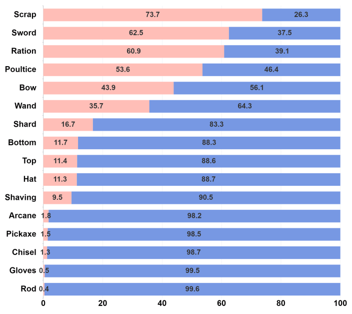

(c) Buy/sell ratio

Figure E.5: Details of the distribution of items in the NMMO environment.

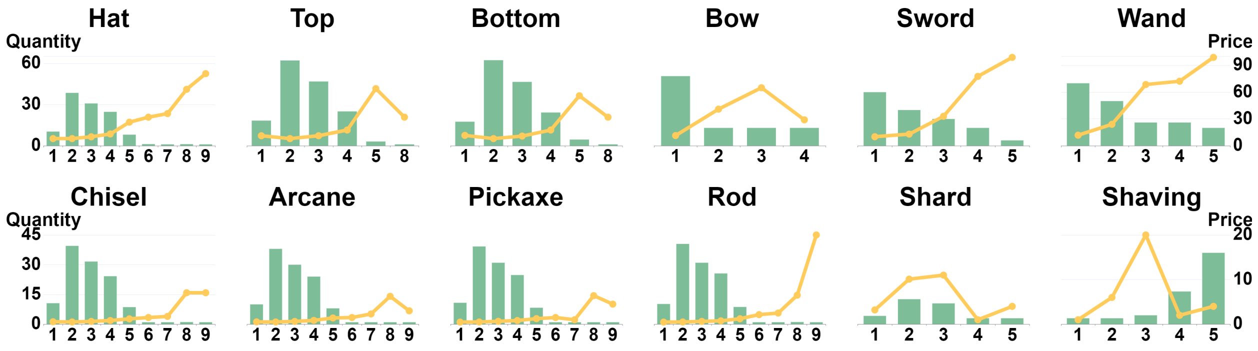


Figure E.6: Quantity available and price of each item at various levels. Acquiring higher-level items is generally more difficult.

Appendix F

NeurIPS 2023 Datasets and Benchmarks

F.1 Game State

```
# True if any agent in subject can see a tile of tile_type
any(tile_type.index in t for t in subject.obs.tile.material_id)

# True if all subjects are alive.
np.count_nonzero(subject.health > 0) == len(subject)

# Computes the maximum l-inf distance between teammates
current_dist = max(subject.row.max()-subject.row.min(),
      subject.col.max()-subject.col.min())

# Tracks hits scored with a specific combat style
hits = subject.event.SCORE_HIT.combat_style == combat_style.SKILL_ID

# Computes the summed gold of all teammates
subject.gold.sum()

# Evalutaes to >= 1 if the current game tick is >= the specified tick
gs.current_tick / num_tick
```

F.2 Predicates

F.2.1 Example 1

```
# Signature for predicates
def predicate(gs: GameState, subject: Group, **kwargs) -> float:

def DistanceTraveled(gs: GameState, subject: Group, dist: int):
  """True if the summed l-inf distance between each agent's current pos and
  spawn pos is greater than or equal to the specified _dist."""
  if not any(subject.health > 0):
    return 0
  r, c = subject.row, subject.col
  dists = utils.linf(list(zip(r,c)),[gs.spawn_pos[id_]
        for id_ in subject.entity.id])
  return max(min(dists.sum() / dist, 1.0), 0.0)
```

F.2.2 Example 2

```
def FullyArmed(gs: GameState, subject: Group,
               combat_style: nmmo_skill.CombatSkill,
               level: int,  num_agent: int):
  """True if the number of fully equipped agents >= _num_agent

  To determine fully equipped, we compare the levels of
  the hat, top, bottom, weapon, ammo to _level."""
  WEAPON_IDS = {
    nmmo_skill.Melee: {'weapon':5, 'ammo':13}, # Spear, Whetstone
    nmmo_skill.Range: {'weapon':6, 'ammo':14}, # Bow, Arrows
    nmmo_skill.Mage: {'weapon':7, 'ammo':15} # Wand, Runes
  }
  item_ids = { 'hat':2, 'top':3, 'bottom':4 }
  item_ids.update(WEAPON_IDS[combat_style])
  lvl_flt = (subject.item.level >= level) & (subject.item.equipped > 0)
  type_flt = np.isin(subject.item.type_id,list(item_ids.values()))
  _, equipment_numbers = np.unique(
        subject.item.owner_id[lvl_flt & type_flt], return_counts=True)
  if num_agent == 0:
    return 1.0
  return max(min((equipment_numbers >= len(
        item_ids.items())).sum() / num_agent, 1.0), 0.0)
```

F.3 Tasks

```
def KillPredicate(gs: GameState, subject: Group):
  """The progress, the max of which is 1, should
       * increase small for each player kill
       * increase big for the 1st and 3rd kills
       * reach 1 with 10 kills"""
  num_kills = len(subject.event.PLAYER_KILL)
  progress = num_kills * 0.06
  if num_kills >= 1:
    progress += .1
  if num_kills >= 3:
    progress += .3
  return min(progress, 1.0)
```

This predicate can be turned into a task like this.

```
from nmmo.task import predicate_api
from nmmo.task.group import Group

# Create a Predicate class with  interfaces to GameState and Group
pred_cls = predicate_api.make_predicate(KillPredicate)

# Create a task for each agent
task_list = []
for agent_id in agent_list:
  task_list.append(pred_cls(subject=Group(agent_id)).create_task())

# Create a task that evaluates and rewards a whole team together
team_task = pred_cls(subject=Group(agent_list)).create_task()

# Make a task that agent 1 gets rewarded for the agent 2's evaluation
pred_cls = predicate_api.make_predicate(AllDead)
task_for_agent_1 = pred_cls(
        subject=Group(agent_2)).create_task(assignee=agent_1)
```

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[2] OpenAI, *Openai five*, https://blog.openai.com/openai-five/, 2018.

[3] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, *et al.*, "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning," *arXiv preprint arXiv:1807.01281*, 2018.

[4] K. O. Stanley, J. Lehman, and L. Soros, *Open-endedness: The last grand challenge you've never heard of*, https://www.oreilly.com/ideas/open-endedness-the-last-grand-challenge-youve-never-heard-of, Accessed: 2017-09-26, 2017.

[5] C. G. Langton, *Artificial life: An overview.* Mit Press, 1997.

[6] S. G. Ficici and J. B. Pollack, "Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states," in *Proceedings of the sixth international conference on Artificial life*, MIT Press, 1998, pp. 238–247.

[7] L. Yaeger, "Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or poly world: Life in a new context," in *SANTA FE INSTITUTE STUDIES IN THE SCIENCES OF COMPLEXITY-PROCEEDINGS VOLUME-*, ADDISON-WESLEY PUBLISHING CO, vol. 17, 1994, pp. 263–263.

[8] K. Sims, "Evolving 3d morphology and behavior by competition," *Artificial life*, vol. 1, no. 4, pp. 353–372, 1994.

[9] J. Hernández-Orallo, D. L. Dowe, S. España-Cubillo, M. V. Hernández-Lloreda, and J. Insa-Cabrera, "On more realistic environment distributions for defining, evaluating and developing intelligence," in *International Conference on Artificial General Intelligence*, Springer, 2011, pp. 82–91.

[10] C. Strannegård, N. Svangård, D. Lindström, J. Bach, and B. Steunebrink, "Learning and decision-making in artificial animals," *Journal of Artificial General Intelligence*, vol. 9, pp. 55–82, Jul. 2018. DOI: 10.2478/jagi-2018-0002.

[11] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," *arXiv preprint arXiv:1710.03748*, 2017.

[12] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Neural Information Processing Systems (NIPS)*, 2017.

[13] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," *arXiv preprint arXiv:1703.04908*, 2017.

[14] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, J. Perolat, D. Silver, T. Graepel, *et al.*, "A unified game-theoretic approach to multiagent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4190–4203.

[15] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," *arXiv preprint arXiv:1802.05438*, 2018.

[16] L. Zheng, J. Yang, H. Cai, W. Zhang, J. Wang, and Y. Yu, "Magent: A many-agent reinforcement learning platform for artificial collective intelligence," *arXiv preprint arXiv:1712.00600*, 2017.

[17] Y. Yang, L. Yu, Y. Bai, Y. Wen, W. Zhang, and J. Wang, "A study of ai population dynamics with million-agent reinforcement learning," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2133–2135.

[18] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[19] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta learn fast: A new benchmark for generalization in rl," *arXiv preprint arXiv:1804.03720*, 2018.

[20] G. Cuccu, J. Togelius, and P. Cudre-Mauroux, "Playing atari with six neurons," *arXiv preprint arXiv:1806.01363*, 2018.

[21] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, *et al.*, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

[22] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, *Emergent tool use from multi-agent autocurricula*, 2019. arXiv: 1909.07528 [cs.LG].

[23] O. Vinyals, I. Babuschkin, J. Chung, *et al.*, *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*, https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[24] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging AI applications," *CoRR*, vol. abs/1712.05889, 2017. arXiv: 1712.05889. [Online]. Available: http://arxiv.org/abs/1712.05889.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602.

[26] M. Moravcık, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. H. Bowling, "Deepstack: Expert-level artificial intelligence in no-limit poker," *CoRR*, vol. abs/1701.01724, 2017. arXiv: 1701.01724. [Online]. Available: http://arxiv.org/abs/1701.01724.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[28] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, vol. 12, MIT Press, 2000, pp. 1057–1063.

[29] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K. Müller, Eds., MIT Press, 2000, pp. 1008–1014. [Online]. Available: http://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 00280836. [Online]. Available: http://dx.doi.org/10.1038/nature14236.

[31] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *CoRR*, vol. abs/1207.4708, 2012. arXiv: 1207.4708. [Online]. Available: http://arxiv.org/abs/1207.4708.

[32] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta learn fast: A new benchmark for generalization in rl," *arXiv preprint arXiv:1804.03720*, 2018.

[33] OpenAI, *Universe*, Dec. 2016. [Online]. Available: https://openai.com/blog/universe/.

[34] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," *arXiv preprint arXiv:1912.01588*, 2019.

[35] B. Baker, I. Kanitscheider, T. M. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *CoRR*, vol. abs/1909.07528, 2019. arXiv: 1909.07528. [Online]. Available: http://arxiv.org/abs/1909.07528.

[36] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *CoRR*, vol. abs/1706.02275, 2017. arXiv: 1706.02275. [Online]. Available: http://arxiv.org/abs/1706.02275.

[37] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," *CoRR*, vol. abs/1703.04908, 2017. arXiv: 1703.04908. [Online]. Available: http://arxiv.org/abs/1703.04908.

[38] C. Beattie, J. Z. Leibo, D. Teplyashin, *et al.*, “Deepmind lab,” *CoRR*, vol. abs/1612.03801, 2016. arXiv: 1612.03801. [Online]. Available: http://arxiv.org/abs/1612.03801.

[39] Y. Tassa, Y. Doron, A. Muldal, *et al.*, “Deepmind control suite,” *CoRR*, vol. abs/1801.00690, 2018. arXiv: 1801.00690. [Online]. Available: http://arxiv.org/abs/1801.00690.

[40] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, “Unity: A general platform for intelligent agents,” *CoRR*, vol. abs/1809.02627, 2018. arXiv: 1809.02627. [Online]. Available: http://arxiv.org/abs/1809.02627.

[41] *Vizdoom*. [Online]. Available: http://vizdoom.cs.put.edu.pl/research.

[42] C. Resnick, W. Eldridge, D. Ha, D. Britz, J. Foerster, J. Togelius, K. Cho, and J. Bruna, “Pommerman: A multi-agent playground,” *CoRR*, vol. abs/1809.07124, 2018. arXiv: 1809.07124. [Online]. Available: http://arxiv.org/abs/1809.07124.

[43] C. Bamford, S. Huang, and S. M. Lucas, “Griddly: A platform for AI research in games,” *CoRR*, vol. abs/2011.06363, 2020. arXiv: 2011.06363. [Online]. Available: https://arxiv.org/abs/2011.06363.

[44] W. H. Guss, C. Codel, K. Hofmann, *et al.*, *The minerl 2019 competition on sample efficient reinforcement learning using human priors*, 2021. arXiv: 1904.10079 `[cs.LG]`.

[45] D. Silver, A. Huang, C. J. Maddison, *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, ISSN: 0028-0836. DOI: 10.1038/nature16961.

[46] C. Berner, G. Brockman, B. Chan, *et al.*, “Dota 2 with large scale deep reinforcement learning,” *CoRR*, vol. abs/1912.06680, 2019. arXiv: 1912.06680. [Online]. Available: http://arxiv.org/abs/1912.06680.

[47] *Alphastar: Mastering the real-time strategy game starcraft ii*. [Online]. Available: https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii.

[48] R. Bommasani, D. A. Hudson, E. Adeli, *et al.*, “On the opportunities and risks of foundation models,” *CoRR*, vol. abs/2108.07258, 2021. arXiv: 2108.07258. [Online]. Available: https://arxiv.org/abs/2108.07258.

[49] OpenAI, I. Akkaya, M. Andrychowicz, *et al.*, “Solving rubik’s cube with a robot hand,” *CoRR*, vol. abs/1910.07113, 2019. arXiv: 1910.07113. [Online]. Available: http://arxiv.org/abs/1910.07113.

[50] B. Schölkopf, J. Platt, and T. Hofmann, “Trueskill™: A bayesian skill rating system,” in *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. 2007, pp. 569–576.

[51] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, “Ray rllib: A composable and scalable reinforcement learning library,” *CoRR*, vol. abs/1712.09381, 2017. arXiv: 1712.09381. [Online]. Available: http://arxiv.org/abs/1712.09381.

[52] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.

[53] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[54] J. Suarez, Y. Du, P. Isola, and I. Mordatch, "Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents," *CoRR*, vol. abs/1903.00784, 2019. arXiv: 1903.00784. [Online]. Available: http://arxiv.org/abs/1903.00784.

[55] L. Zheng, J. Yang, H. Cai, W. Zhang, J. Wang, and Y. Yu, "Magent: A many-agent reinforcement learning platform for artificial collective intelligence," *CoRR*, vol. abs/1712.00600, 2017. arXiv: 1712.00600. [Online]. Available: http://arxiv.org/abs/1712.00600.

[56] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, cite arxiv:1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540.

[57] J. M., H. K., H. T., and B. D, "The malmo platform for artificial intelligence experimentation," *International Joint Conference on Artificial Intelligence*, vol. Proc. 25th, p. 4246, 2016.

[58] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, "The StarCraft Multi-Agent Challenge," *CoRR*, vol. abs/1902.04043, 2019.

[59] M. Jaderberg, W. M. Czarnecki, I. Dunning, *et al.*, "Human-level performance in 3d multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, 2019, ISSN: 0036-8075. DOI: 10.1126/science.aau6249. eprint: https://science.sciencemag.org/content/364/6443/859.full.pdf. [Online]. Available: https://science.sciencemag.org/content/364/6443/859.

[60] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Sep. 2019, pp. 1282–1289.

[61] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 2048–2056. [Online]. Available: http://proceedings.mlr.press/v119/cobbe20a.html.

[62] S. P. Mohanty, J. Poonganam, A. Gaidon, *et al.*, “Measuring sample efficiency and generalization in reinforcement learning benchmarks: Neurips 2020 procgen benchmark,” in *NeurIPS 2020 Competition and Demonstration Track, 6-12 December 2020, Virtual Event / Vancouver, BC, Canada*, H. J. Escalante and K. Hofmann, Eds., ser. Proceedings of Machine Learning Research, vol. 133, PMLR, 2020, pp. 361–395.

[63] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, “Implementation matters in deep RL: A case study on PPO and TRPO,” in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[64] R. Herbrich, T. Minka, and T. Graepel, “Trueskill™: A bayesian skill rating system,” *Advances in neural information processing systems*, vol. 19, 2006.

[65] R. Shah, S. H. Wang, C. Wild, *et al.*, “Retrospective on the 2021 minerl BASALT competition on learning from human feedback,” in *NeurIPS 2021 Competitions and Demonstrations Track, 6-14 December 2021, Online*, D. Kiela, M. Ciccone, and B. Caputo, Eds., ser. Proceedings of Machine Learning Research, vol. 176, PMLR, 2021, pp. 259–272. [Online]. Available: https://proceedings.mlr.press/v176/shah22a.html.

[66] H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel, “The nethack learning environment,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 7671–7684, 2020.

[67] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, “The StarCraft Multi-Agent Challenge,” *CoRR*, vol. abs/1902.04043, 2019.

[68] K. Kurach, A. Raichuk, P. Stanczyk, *et al.*, “Google research football: A novel reinforcement learning environment,” in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press, 2020, pp. 4501–4510.

[69] J. Suarez, Y. Du, P. Isola, and I. Mordatch, “Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents,” *arXiv preprint arXiv:1903.00784*, 2019.

[70] Z. Tang, C. Yu, B. Chen, H. Xu, X. Wang, F. Fang, S. S. Du, Y. Wang, and Y. Wu, “Discovering diverse multi-agent strategic behavior via reward randomization,” in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=lvRTC669EY%5C_.

[71] Anonymous, “Gobigger: A scalable platform for cooperative-competitive multi-agent interactive simulation,” in *Submitted to The Eleventh International Conference on Learning Representations*, under review, 2023. [Online]. Available: https://openreview.net/forum?id=NnOZT_CR26Z.

[72] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1329–1338.

[73] E. Hambro, S. Mohanty, D. Babaev, M. Byeon, D. Chakraborty, E. Grefenstette, M. Jiang, J. Daejin, A. Kanervisto, J. Kim, *et al.*, "Insights from the neurips 2021 nethack challenge," in *NeurIPS 2021 Competitions and Demonstrations Track*, PMLR, 2022, pp. 41–52.

[74] S. Milani, N. Topin, B. Houghton, W. H. Guss, S. P. Mohanty, K. Nakata, O. Vinyals, and N. S. Kuno, "Retrospective analysis of the 2019 minerl competition on sample efficient reinforcement learning," in *NeurIPS 2019 Competition and Demonstration Track*, PMLR, 2020, pp. 203–214.

[75] B. Doerschuk-Tiberi and S. Tao, *Lux AI Challenge Season 1*, version 1.0.0, Jul. 2021. [Online]. Available: https://github.com/Lux-AI-Challenge/Lux-Design-2021.

[76] J. Suarez, Y. Du, C. Zhu, I. Mordatch, and P. Isola, "The neural MMO platform for massively multiagent research," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, J. Vanschoren and S. Yeung, Eds., 2021.

[77] OpenAI, : C. Berner, *et al.*, *Dota 2 with large scale deep reinforcement learning*, 2019. DOI: 10.48550/ARXIV.1912.06680. [Online]. Available: https://arxiv.org/abs/1912.06680.

[78] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in starcraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019. DOI: 10.1038/s41586-019-1724-z. [Online]. Available: https://doi.org/10.1038/s41586-019-1724-z.

[79] H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette, "Torchbeast: A pytorch platform for distributed RL," *CoRR*, vol. abs/1910.03552, 2019. arXiv: 1910.03552. [Online]. Available: http://arxiv.org/abs/1910.03552.

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

[81] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, pp. 15–22.

[82] P. Trueba, A. Prieto, F. Bellas, P. Caamaño, and R. J. Duro, "Self-organization and specialization in multiagent systems through open-ended natural evolution," in *Applications of Evolutionary Computation*, C. Di Chio, A. Agapitos, S. Cagnoni, *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 93–102, ISBN: 978-3-642-29178-4.

[83] M. J. Gasparrini, R. Solé, and M. Sánchez-Fibla, *Individual specialization in multi-task environments with multiagent reinforcement learners*, 2019. arXiv: 1912.12671 [cs.AI].

[84] K. Derek and P. Isola, "Adaptable agent populations via a generative model of policies," *CoRR*, vol. abs/2107.07506, 2021. arXiv: 2107.07506. [Online]. Available: https://arxiv.org/abs/2107.07506.

[85] S. Zheng, A. Trott, S. Srinivasa, N. Naik, M. Gruesbeck, D. C. Parkes, and R. Socher, *The ai economist: Improving equality and productivity with ai-driven tax policies*, 2020. arXiv: 2004.13332 [econ.GN].

[86] Lux-AI-Challenge, *Lux-ai-challenge/lux-design-2021: Home to the design and engine of the @lux-ai-challenge season 1, hosted on @kaggle*. [Online]. Available: https://github.com/Lux-AI-Challenge/Lux-Design-2021.

[87] J. Suarez, Y. Du, C. Zhu, I. Mordatch, and P. Isola, "The neural mmo platform for massively multiagent research," vol. 33, 2021. [Online]. Available: http://arxiv.org/abs/2110.07594.

[88] J. Suarez and P. Isola, "Specialization and exchange in neural mmo," in *ICLR Workshop on Agent Learning in Open-Endedness*, 2022.

[89] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, pp. 1–5, 2019.

[90] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, and C. H. et al., "Dota 2 with large scale deep reinforcement learning," *ArXiv*, vol. abs/1912.06680, 2019.

[91] S. B. Jiang and C. Amato, "Multi-agent reinforcement learning with directed exploration and selective memory reuse," *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021.

[92] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *ArXiv*, vol. abs/1706.02275, 2017.

[93] H. Jia, C. Ren, Y. Hu, Y. Chen, T. Lv, C. Fan, H. Tang, and J. Hao, "Mastering basketball with deep reinforcement learning: An integrated curriculum training approach," in *Adaptive Agents and Multi-Agent Systems*, 2020.

[94] K. Kurach, A. Raichuk, P. Stanczyk, *et al.*, "Google research football: A novel reinforcement learning environment," *ArXiv*, vol. abs/1907.11180, 2019.

[95] C. Resnick, W. Eldridge, D. R. Ha, D. Britz, J. N. Foerster, J. Togelius, K. Cho, and J. Bruna, "Pommerman: A multi-agent playground," *ArXiv*, vol. abs/1809.07124, 2018.

[96] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning," *CoRR*, vol. abs/1803.11485, 2018. arXiv: 1803.11485. [Online]. Available: http://arxiv.org/abs/1803.11485.

[97] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," *CoRR*, vol. abs/1705.08926, 2017. arXiv: 1705.08926. [Online]. Available: http://arxiv.org/abs/1705.08926.

[98] J. Suarez, Y. Du, P. Isola, and I. Mordatch, *Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents*, 2019. arXiv: 1903.00784 `[cs.MA]`.

[99] J. Suarez, Y. Du, I. Mordach, and P. Isola, "Neural mmo v1.3: A massively multiagent game environment for training and evaluating neural networks," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20, Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 2020–2022, ISBN: 9781450375184.

[100] J. Suarez, Y. Du, C. Zhu, I. Mordatch, and P. Isola, "The neural mmo platform for massively multiagent research," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung, Eds., vol. 1, 2021. [Online]. Available: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/44f683a84163b3523afe57c2e008bc8c-Paper-round1.pdf.

[101] H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel, *The nethack learning environment*, 2020. arXiv: 2006.13760 `[cs.LG]`.

[102] J. Z. Leibo, E. A. Duéñez-Guzmán, A. S. Vezhnevets, J. P. Agapiou, P. Sunehag, R. Koster, J. Matyas, C. Beattie, I. Mordatch, and T. Graepel, "Scalable evaluation of multi-agent reinforcement learning with melting pot," *CoRR*, vol. abs/2107.06857, 2021. arXiv: 2107.06857. [Online]. Available: https://arxiv.org/abs/2107.06857.

[103] O. E. L. Team, A. Stooke, A. Mahajan, *et al.*, *Open-ended learning leads to generally capable agents*, 2021. arXiv: 2107.12808 `[cs.LG]`.

[104] J. P. Agapiou, A. S. Vezhnevets, E. A. Duéñez-Guzmán, *et al.*, *Melting pot 2.0*, 2023. arXiv: 2211.13746 `[cs.MA]`.

[105] A. A. Team, J. Bauer, K. Baumli, *et al.*, *Human-timescale adaptation in an open-ended task space*, 2023. arXiv: 2301.07608 `[cs.LG]`.

[106] H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette, "Torchbeast: A pytorch platform for distributed RL," *CoRR*, vol. abs/1910.03552, 2019. arXiv: 1910.03552. [Online]. Available: http://arxiv.org/abs/1910.03552.

[107] J. K. Terry, B. Black, N. Grammel, *et al.*, *Pettingzoo: Gym for multi-agent reinforcement learning*, 2021. arXiv: 2009.14471 `[cs.LG]`.

[108] S. Huang, R. F. J. Dossa, C. Ye, and J. Braga, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *CoRR*, vol. abs/2111.08819, 2021. arXiv: 2111.08819. [Online]. Available: https://arxiv.org/abs/2111.08819.

[109] S. Mohanty, J. Poonganam, A. Gaidon, *et al.*, *Measuring sample efficiency and generalization in reinforcement learning benchmarks: Neurips 2020 procgen benchmark*, 2021. arXiv: 2103.15332 [cs.LG].

[110] E. Hambro, S. Mohanty, D. Babaev, *et al.*, *Insights from the neurips 2021 nethack challenge*, 2022. arXiv: 2203.11889 [cs.LG].

[111] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, *et al.*, "The MineRL competition on sample efficient reinforcement learning using human priors," *NeurIPS Competition Track*, 2019.

[112] J. Suárez, P. Isola, K. W. Choe, *et al.*, *Neural mmo 2.0: A massively multi-task addition to massively multi-agent learning*, 2023. arXiv: 2311.03736 [cs.AI].

[113] O. Vinyals, M. Fortunato, and N. Jaitly, *Pointer networks*, 2017. arXiv: 1506.03134 [stat.ML].

[114] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html.

[115] Y. Zhou, S. Liu, Y. Qing, K. Chen, T. Zheng, Y. Huang, J. Song, and M. Song, *Is centralized training with decentralized execution framework centralized enough for marl?* 2023. arXiv: 2305.17352 [cs.AI].

[116] K. Perlin, "An image synthesizer," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 287–296, Jul. 1985, ISSN: 0097-8930. DOI: 10.1145/325165.325247. [Online]. Available: http://doi.acm.org/10.1145/325165.325247.

[117] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. arXiv: 1606.01540. [Online]. Available: http://arxiv.org/abs/1606.01540.

[118] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014. arXiv: 1409.0473. [Online]. Available: http://arxiv.org/abs/1409.0473.

[119] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.