Design and Analysis of Custom Policies and Reward Functions in MMO Survival Games

Zhihao Su
Center for Data Science
New York University
zs1512@nyu.edu

Ziyue Zhou
Center for Data Science
New York University
tz1307@nyu.edu

Qianyi Xiao
Center for Data Science
New York University
qx690@nyu.edu

Chenxing Li
Center for Data Science
New York University
cl7451@nyu.edu

Abstract

Massively multiplayer online (MMO) games present unique challenges and opportunities for the development of sophisticated multi-agent systems. This paper explores the integration and optimization of custom policies and reward functions within the Neural MMO environment, a platform that simulates complex agent interactions in a resource-constrained setting. Utilizing Proximal Policy Optimization (PPO), we implement and refine strategies that emphasize strategic resource management, task completion, and survival. Experimental results demonstrate that our tailored policies enhance agent performance in terms of survival rates and task completion efficiency compared to baseline models. Although the integration with our custom reward bonuses has shown promise, further experiments reveal complexities in optimizing weight distribution. This finding underscores the need for more nuanced adjustments and systematic exploration in future work, aimed at achieving optimal agent behaviors in evolving game dynamics.

1 Background of the Neural MMO Environment

In recent years, the study of multi-agent systems has garnered substantial interest, driven by their potential to simulate complex interactions that mirror real-world phenomena across various domains, such as economics and autonomous driving. Pioneering research by Silver et al. [2017] on AlphaGo introduced groundbreaking methods in strategic gameplay through reinforcement learning, demonstrating profound capabilities of learning agents in controlled, competitive environments. Further advancements were showcased by Vinyals et al. [2019] in the StarCraft II environment, revealing how agents manage complex, dynamic strategic tasks. Building on these foundations, the Neural MMO 2.0 environment, proposed by Suarez et al. [2024], extends these explorations to an MMO(Massively multiplayer online game)-style survival game setting. This platform emphasizes the strategic developments of agents as they interact within a resource-constrained environment, navigating tasks from resource harvesting to combat skill enhancement.

1.1 Current Challenges in Neural MMO

According to Suarez et al. [2024], even top-performing strategies in recent competitions have failed to fully leverage all available game systems, indicating significant potential for optimization. Moreover, agent specialization within teams has remained limited, often due to an overly broad survival objective that promotes dominant, but not necessarily optimal, strategies. These observations suggest that

focusing on refining specific strategies, particularly those related to essential survival mechanics such as resource management and task completion, can substantially improve agent performance. This situation presents a critical challenge: achieving a nuanced understanding of the strategic tasks that are most influential in long-term survival and effectiveness, thereby avoiding the pitfalls of oversimplified or overcomplicated strategies.

1.2 Research Objectives

Our main goal is to address these challenges by achieving a delicate balance between task completion and long-term survival in the dynamic Neural MMO environment. This balance aims to demonstrate the dual capabilities of agents in managing essential resources while also accomplishing designated objectives efficiently. It is important to note that, like many research endeavors in complex simulation environments, our project does not fully utilize all game systems available in Neural MMO. Instead, our approach involves a targeted examination of specific strategies and systems:

Strategic Resource Management: Ensuring agents can sustain themselves by effectively managing food, water, and health, pivotal for long-term survival.

Task Completion: Encouraging agents to complete a variety of tasks that require diverse strategies, focusing particularly on those that align with the core survival mechanics of the game.

Optimization of Policy and Reward Functions: We will specifically focus on optimizing and comparing policy and reward functions related to crucial survival aspects such as resource management, map exploration, and combat performance. This selective approach allows for a more detailed exploration of these areas, which are critical to agent survival and effectiveness.

1.3 Preliminaries: Important Features and Rules in NMMO Environment

The Neural MMO environment is designed to challenge agents with a variety of survival and strategic tasks, necessitating a dynamic adaptation approach. Below are some crucial elements of the environment that directly impact our strategy formulation:

1. **Tile Classification**: Tiles are categorized into passable (where agents can walk) and obstacles (which block movement), alongside distinctions between resource-yielding tiles (like foliage, ore, trees, crystals, and herbs) and non-resource tiles (such as grass and harvested areas).
2. **Survival Needs**: Agents maintain health, food, and water levels, losing 5 units of food and water per tick, and up to 20 health when both are depleted (regain 10 if maintaining half food and water level). Resource replenishment is achieved by interacting with specific tiles: walking on foliage restores food, and being adjacent to water tiles refills water.
3. **Combat System**: Agents engage in combat with a rock-paper-scissors mechanic influencing attack effectiveness, impacted by factors such as skill levels and equipment. This strategic element requires agents to choose their battles wisely based on their and their opponent's capabilities.
4. **Task Completion**: Agents are rewarded for completing specific tasks, adding a layer of goal-oriented behavior to the survival elements. Points are awarded for each specific task completed, ranging from defeating an NPC or agent to leveling up to a specific skill level.
5. **Observation Space**: Each agent and NPC does not have a full map view; they can only observe data from nearby 15x15 visible tiles, data from up to 100 entities within its vision, and attack only within 7x7 squares.

2 Proximal Policy Optimization and Framework Integration

Proximal Policy Optimization (PPO) is a widely used reinforcement learning algorithm that stands out due to its stability and effectiveness in environments with high-dimensional action spaces, such as Neural MMO. In our project, we integrate PPO through CleanPuffeRL from PufferLib, a sophisticated framework that combines reinforcement learning algorithms, machine learning models, and complex game environments seamlessly.

2.1 Integration with CleanPuffeRL

PufferLib provides a crucial layer of abstraction that simplifies the integration of various components necessary for RL training. By wrapping the environment, RL framework, and ML models together, PufferLib facilitates efficient data handling and model training without the need for manual configuration of the intricate interactions between these components.

CleanPuffeRL, an implementation provided by PufferLib, acts as a wrapper that simplifies the deployment of PPO by handling the interactions between the Neural MMO game environment and the neural network policies. This encapsulation allows us to focus on enhancing agent strategies without delving into the complexities of the underlying RL mechanics.

2.2 Key Components of the PPO Objective Function

Proximal Policy Optimization (PPO) optimizes a specialized objective function designed to improve learning stability by limiting the size of policy updates, its objective function is expressed as:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{1}$$

The PPO objective function optimizes policy behavior through several interrelated components:

- **Probability Ratio** ($r_t(\theta)$): Defined as $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, this ratio measures how the likelihood of taking action a_t under the current policy compares to the old policy at state s_t. Adjusting this ratio carefully ensures that updates are significant enough to facilitate learning but restrained enough to maintain stability, crucial in the dynamic setting of Neural MMO.
- **Advantage Estimator** ($\hat{A}_t$): This estimator quantifies the relative benefit of taking action a_t over the average action at that state, which is crucial for guiding the direction and magnitude of policy updates. In Neural MMO, we customize the reward function that feeds into this estimator, allowing us to prioritize strategic actions that enhance survival and task success. This customization helps mitigate the risk of converging on suboptimal strategies and maintains the effectiveness of agent behaviors.
- **Clipping Parameter** (ϵ): A hyperparameter that sets the bounds for clipping in the objective function to manage variance and limit the size of policy updates. We have customized ϵ to 0.1, tighter than the typically recommended 0.2, to balance the need for exploration and exploitation in the complex, multi-agent environment of Neural MMO.

2.3 Rationale Behind Selecting PPO

Our choice of Proximal Policy Optimization is not only motivated by its robustness in handling high-dimensional action spaces but also by its documented success in cooperative multi-agent settings as detailed in Yu et al. [2021]. This paper highlights PPO's ability to foster both competitive and cooperative strategies among agents, which is essential for the diverse challenges presented in Neural MMO. The algorithm's flexibility in policy iteration and its capacity to effectively balance multiple objectives make it an ideal choice for our project.

3 Optimized Policy Function Architecture

The policy function $\pi : S \rightarrow P(A)$ plays a significant role in reinforcement learning, where S represents the state space and $P(A)$ represents the set of all probability distributions over the action space. The output of the policy π for a given state s is a probability distribution over actions, allowing the model to choose actions based on the learned probabilities effectively. In our project, the baseline policy function adopts neural networks as its base model, leveraging their capability to approximate complex functions. The overall architecture of our baseline policy comprises the following two key components: the State Encoder f_s and the Action Decoder f_a

The state encoder consists of multiple shallow neural networks designed to extract and integrate information from diverse sources such as Tile and Item inputs. These networks process the inputs

to capture and transform the essential features of the state into a more expressive latent space representation $z \in \mathbb{R}^n$. The mathematical formulation of the state encoder is given by:

$$f_s(s) = z$$

This representation z encodes the state information in a reduced dimensionality that preserves critical features necessary for effective policy decision-making.

The action decoder utilizes the latent representation z provided by f_s to output a probability distribution over the action space. This is achieved through a transformation that maps the latent space to the action probabilities, effectively guiding the agent's decisions in the environment. The mathematical representation of the action decoder is:

$$f_a(z) = P(a)$$

where $P(a)$ denotes the probability distribution over actions conditioned on the latent state representation. The action decoder translates the encoded state into actionable decisions, allowing the agents to perform actions that are most likely to lead to optimal outcomes based on the current state.

3.1 Enhancement Methodology

Our contribution to enhancing the policy function encompasses the integration of a more complex neural network model architecture, designed to improve the learning efficiency and generalization potential of the State Encoder and the Action Decoder within the baseline policy. To enhance the robustness and effectiveness of our policy model, we have integrated several key features designed to mitigate common issues in deep learning models. In this section, we will illustrate the motivation and intuition of the architectural and functional enhancements and why it ensure that our policy function is not only more capable of capturing complex state information but also more generalizable to different states and actions.

3.1.1 Additional Hidden Layers with Residual Blocks

The architecture of our policy model integrates additional fully connected layers accompanied by residual blocks to enhance the depth of the network. The enhanced network depth is crucial for a more detailed learning process and a complex model. This approach is inspired by the success of deep residual learning for visual recognition as discussed in He et al. [2016]. The mathematical formulation of these enhancements can be expressed as:

$$y = F(x, \{W_i\}) + x$$

where x represents the input to a fully connected layer, W_i are the weights of the layers within the residual block, and F denotes the transformation composed of several fully connected layers. The use of residual blocks addresses the vanishing gradient problem by introducing direct pathways for gradient flow during back-propagation, thereby stabilizing and accelerating the training process.

3.1.2 Activation Functions

We also add more advanced and complex activation functions to the baseline model. More specifically, Leaky ReLU and Parametric ReLU (PReLU) are utilized. Both Leaky ReLU and PReLU can be expressed mathematically as:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

where α is a small coefficient in the case of Leaky ReLU but is a learnable parameter in the case of PReLU, providing a small gradient when the unit is not active and x is less than zero. This modification helps to keep the gradient flow alive during the training process and is particularly effective in deeper network architectures where the risk of vanishing gradients is more pronounced.

3.1.3 Batch Normalization Layers

In addition, we also utilize batch normalization layers in our enhanced model. Batch normalization is a technique designed to improve the stability of neural networks by normalizing the inputs of each

Table 1: Policy Performance Analysis

Evaluation Metrics	Baseline Policy	Our Policy
Number of Tasks Completed	5	**8**
Number of Ticks Survived	113	**117**

layer. Mathematically, it adjusts and scales the inputs using the formula:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

where $x^{(k)}$ is the input to a layer for a mini-batch $\mathcal{B}$, $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are the mean and variance of the mini-batch, respectively, and ϵ is a small constant added for numerical stability. The output $\hat{x}^{(k)}$ is then scaled and shifted by learnable parameters γ and β, which allows the network to undo the normalization if it is beneficial. The normalization process helps in reducing internal covariate shift, which is the change in the distribution of network activations due to the update in layers during training, leading to faster training and more stable convergence.

3.1.4 Dropout Layers

Finally, we add dropout layers to our model, which is a regularization technique used to prevent neural networks from overfitting. It works by randomly deactivating a subset of neurons in a layer during each training epoch, according to a predefined probability p. Random dropouts force the model to learn more robust features that are less reliant on any individual neuron. This enhances the network's ability to generalize from the training data to unseen data, effectively improving model robustness across diverse scenarios.

3.2 Experiment Results

To compare the performance of our optimized policy and the baseline policy, we use the same evaluate environment to test the capability of the two policies. Table 1 and Figure 1 illustrates the performance comparison, showing that over 118 ticks, our policy reveals a slightly higher survival rates and a significantly higher task completion efficiency.

3.2.1 Agents Alive Over Time

In the initial phase (up to approximately Tick 20), both policies maintain a full count of alive agents, suggesting similar resilience against early challenges. However, as the simulation progresses, the agents based on our policy demonstrates a more pronounced decline in agent survival, particularly noticeable past Tick 20, where a steep drop occurs, culminating in a dramatic reduction to a single agent by Tick 117. In contrast, although the agents based on the baseline policy exhibits a more gradual decline in the number of alive agents, the longest agents alive survived only 113 ticks.

3.2.2 Tasks Completed Over Time

Regarding task completion, our policy starts to diverge noticeably from the baseline policy around Tick 11, where it records its first task completion. Our policy shows a superior task completion rate, achieving a higher total number of completed tasks earlier in the simulation and maintaining this lead throughout. Notably, agents based on our policy completes 9 tasks by Tick 33 and maintains this level, despite the dwindling number of agents, suggesting that the surviving agents are highly efficient. On the other hand, the baseline policy achieves a maximum of 5 completed tasks by Tick 21 and remains at this plateau for the duration of the simulation, indicating either a lower task completion capability.

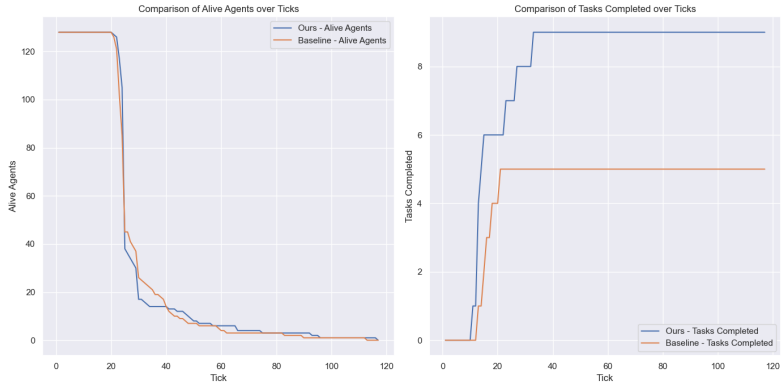

Figure 1: This figure compares the performance of two models using two metrics.

4 Custom Reward Shaping

4.1 Mathematical Formulation of the Reward Function

The reward function in reinforcement learning acts as a critical feedback mechanism that shapes the learning process by providing direct feedback to the agent based on its actions. In our project within the Neural MMO environment, we have devised a reward function that combines the intrinsic rewards of the game with custom bonuses to promote strategic behaviors. The reward function for an agent at time step $t+1$ (tick $t+1$) is expressed as:

$$R_{t+1} = R_{\text{env}}(s_t, a_t, s_{t+1}) + \sum_{i \in \{h,m,e\}} w_i \cdot R_i(s_t, a_t, s_{t+1})$$

where:

- R_{env} represents the environmental rewards provided by Neural MMO, which are influenced by the game dynamics and agent interactions.
- R_h, R_m, and R_e represent the bonuses for healing, meandering, and exploration, respectively, designed to enhance survival skills and encourage proactive exploration.
- w_h, w_m, w_e are the weights assigned to each bonus

4.2 Calculation of Specific Bonuses

4.2.1 Health Bonus

R_h is calculated based on the presence of healing actions. If there is any healing (increase in health due to consuming resources like food or water), a fixed bonus is applied:

$$R_h(s_{t+1}) = \begin{cases} 1 & \text{if health}_{t+1} > \text{health}_t \\ 0 & \text{otherwise} \end{cases}$$

4.2.2 Meandering Bonus

R_m encourages diverse movement patterns and is computed by measuring the entropy of the agent's recent movements (our version considers the movements over the past 8 ticks) to discourage repetitive or predictable paths:

$$R_m(\Delta s) = \text{Entropy}(\text{movements}_{t-8:t})$$

4.2.3 Exploration Bonus

R_e rewards the agent for discovering new areas or engaging in new interactions, quantified by the change in the count of unique events or tiles visited:

$$R_e(\Delta e) = \text{unique\_events}_{t+1} - \text{unique\_events}_t$$

For the exploration bonus,

$$R_e(\Delta e) = \sqrt{\text{unique\_events}_{t+1}} - \sqrt{\text{unique\_events}_t}$$

has also been considered, but we decided to stick with the non-squared root version after few testing runs show a slightly weaker performance

4.3 Considered and Excluded Bonuses

During the development of our reward function, we experimented with various incentives to enhance the performance of agents within the Neural MMO environment.

One particular bonus that was considered but ultimately not implemented in our final model is the *alive bonus*. This bonus was designed to incentivize agents to maintain high health levels by rewarding them for minimal health loss during gameplay. Mathematically, it could be expressed as follows:

$$\text{Alive Bonus} = \max(0, -(\text{health\_lost} - 50)/50 \times 0.001)$$

where health_lost $= 100 -$ current health, activating only when health dropped below 50%.

Reasons for Exclusion:

- **Sparse Rewards:** The bonus was triggered infrequently, leading to slow learning and inadequate agent training due to insufficient feedback on behavior.
- **Conservative Behavior:** There was concern that such a bonus might encourage overly cautious gameplay, potentially leading agents to avoid essential learning experiences such as combat or exploration.
- **Reward Imbalance:** Balancing this bonus with other rewards proved difficult; it risked overshadowing other critical behaviors necessary for holistic agent development.
- **Risk of Overfitting:** Focusing heavily on maintaining high health might lead to agents exploiting specific game mechanics, thereby hindering their ability to adapt and perform well across a broader range of scenarios within Neural MMO.

These considerations led to the decision to exclude the alive bonus, aiming for a reward structure that fosters a more balanced and comprehensive engagement with the environment's challenges.

4.4 An Ablation Study on the weight

For our project, these bonuses are tailored to synergize with the core objectives of survival and task completion, enhancing the agent's ability to navigate and thrive in the Neural MMO environment effectively. Compared to the baseline model, our main contribution to the reward shaping comes from the specific tuning of both the weights and the bonus implementation for our specific needs.

By adjusting the weights of the proposed bonuses in an ablation study, we assess their individual and combined impact on the overall strategy and effectiveness of the agents.

4.4.1 Comparsions of Results using Custom Policy

In our ablation study, we used our customized policy to assess the impact of varying reward bonuses on agent performance. Starting with baseline weights of heal bonus weight = 0.03, meander bonus weight = 0.02, and explore bonus weight = 0.01, we tested seven combinations where each bonus was either set to zero or increased tenfold. We then evaluated our models based on the two metrics for our goal: **survival** and **task completions**.

4.4.2 Insights and Limitations

From the results, we noticed that the baseline weight under our new policy achieved the best task completion rates but also had the worst survival time.

Regarding the survival time, we realized that the best performer for our policy completely froze the health regeneration bonus which showcases the intricacies of combining different reward function together. In this case, ignoring the healing might lead to more aggressive exploration and exploitation that gain more combat power to survive longer from winning the duel, but also sacrifice some task completion that awards resource collection that is guided by the learning from the healing mindset.

We also acknowledge the result is constrained by our current setup (training time, training environment configuration, and evaluation configuration). Still, the findings provided additional insights on the reward structure under our custom policies.

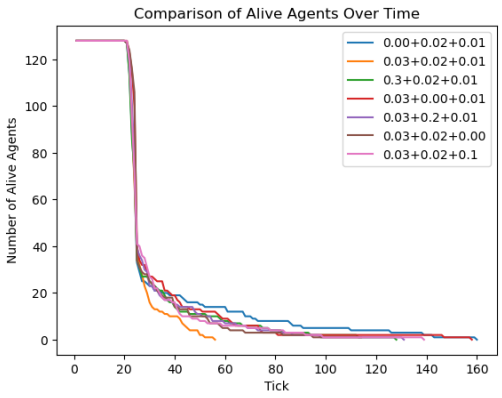

Figure 2: Agent survival over time

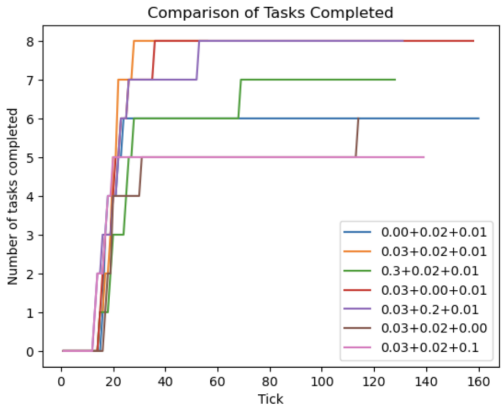

Figure 3: Task completion rates

5 Conclusion and Future Directions

This work has shown that by a custom policy and reward function, significant improvements are possible in the Neural MMO environment, as we have witnessed large increases in survival rates and task completion efficiency compared to baseline models. These results show that PPO is a powerful tool in complex, resource-constrained multi-agent environments. In the future, we plan to refine our methodologies in the following three aspects:

5.1 Optimization of Training Configuration

Determining the number of training epochs that are most appropriate in a given training configuration is essential for improving the efficiency and effectiveness of the learning process. In addition to training epochs, a lot of influences are created by other factors in the configuration such as map size, number of agent, and number of NPC. Therefore, through adjusting these parameters, we can optimize the model's performance to better suit different scenarios and environments.

5.2 Enhancement of Survival Strategies

The current model can be further enhanced by focusing on survival strategies that leverage alive bonuses and targeted rewards. Implementing rewards for proactive resource management encourages behaviors that promote sustainability and effectiveness in survival scenarios. This approach not only improves the model's capability in managing resources under pressure but also aids in developing strategies that are more aligned with real-world decision-making processes.

5.3 Encouragement of Exploration

To foster a more balanced and equitable resource distribution, incentivizing exploration through dynamic rewards could prove beneficial. By driving agents to venture into and utilize the central areas of the map, we can ensure that no single agent or group monopolizes key resources. This strategy not only enhances the game dynamics but also ensures that the AI agents develop capabilities to adapt and respond to changing conditions effectively.

References

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Joseph Suarez, David Bloomin, Kyoung Whan Choe, Hao Xiang Li, Ryan Sullivan, Nishaanth Kanna, Daniel Scott, Rose Shuman, Herbie Bradley, Louis Castricato, et al. Neural mmo 2.0: A massively multi-task addition to massively multi-agent learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

C Yu, A Velu, E Vinitsky, Y Wang, A Bayen, and Y Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. arxiv 2021. *arXiv preprint arXiv:2103.01955*, 2021.

6 Appendix

Figure 4: Overview of Neural MMO 2.0

Figure 5: Game Snapshot