Reinforcement Learning Project
Neural MMO by DQN, PPO, and Curriculum

Anonymous Authors (Github repository: https://github.com/jimmily98/INF581-project)

***Abstract*—So far, we have gained experience of basic reinforcement learning methods including Q-learning, SARSA, Deep Q-learning (DQN), and Policy Gradient (PG) methods. All of them were only applied on single agent environments with small observation and action space such as Cartpole or Lunarlander provided by gym. In this project, we deploy DQN to train multiple agents in a Massively Multiplayer Online-Game (MMO) environment. The result is compared with the one by Proximal Policy Optimization (PPO). And we will also examine how curriculum can be designed to influence the learning process.**

I. Introduction

In this project, we are interested in the performance of DQN model in an MMO game environment with multiple agents.

The chosen MMO environment comes from the competition hosted by NeuralIPS. It is close for submission now. But because this environment is well constructed following the frame of PettingZoo, it is easy for us to adapt our existing code. And because it provides tracks for both reinforcement learning model construction and curriculum building, it fits our interests quite well.

Compared with tabular methods, DQN is good at expressing policy with large observation and action space by using neural networks. It can directly process visual information, such as the image input from Atari games, by making use of convolutional neural networks (CNNs). It can understand temporal information by stacking several frames as one input, such as learning the movement of the ball in the game of Pong.

However, so far, we only implemented DQN with single agent environment. If we have several agents interacting with each other in one environment, especially for the case that they need cooperate, how should we train them with DQN?

One way is to train a central brain which gives orders to each agent. But in this method, the space dimension for the central brain increases a lot and we did not find a better alternative way to solve that.

Another intuition from our group is that we can train each of them separately for every time-step. And when we train one of them, we regard the rest agents' information as part of the environment. Every agent uses the same policy. They cooperate with each other by recognising friend agents from their surrounding environment.

During experimenting DQNs, we suffered a lot from their instability. This instability partly comes from the fact we were bootstrapping and optimizing toward a moving target, whi ch was relieved by using double networks. But it also comes from the fact that DQN is an online method. Although batch updating can decrease certain noise during optimization, higher batch size will increase computation time for each time-step, which results in a new trade-off between accuracy and cost.

After researching, we find that PPO is more stable than DQN and it is currently the state-of-the-art (at least the foundation of state-of-the-art) for reinforcement learning. As a policy-based method, PPO directly outputs the probability distribution of actions given the current state. In this way, it handles continuous action space better than DQN which is a value-based method accompanied with a manual policy such as ϵ-greedy policy.

More importantly, PPO is more stable because it cares about how much change was made between the old and new policy. PPO defined a ratio between the probabilities of one action by the new policy and the old policy. It clips the gradient based on the difference between this ratio and one. If the ratio is too large or too small, the clipped gradient prevents the optimization process from stepping away from the optimal path. Thus it avoids stepping into higher gradient region, which means higher possibility to overshoot and higher variance.

There has already been one PPO implementation provided by our MMO game environment. We use this implementation and compare it with our own DQN implementation (if time is permitted, we may implement one PPO by ourselves as well).

The last part that we are interested to dive in is the design of curriculum.

Our MMO environment is quite complicate that it involves harvesting food, building armors, trading in market, combating with NPCs, upgrading professions, and etc. To speed up the convergence process, one possible solution is to build a learning path (or curriculum) for agents to follow. We can request them to learn how to harvest food and combat against enemies first, and postpone the optimization of trading actions until a later stage. We would like to create several curriculum and see how this will affect the learning process. If time is permitted, we would like to study the possibility of building a model to learn how to construct best curriculum for agents.

(final results TBD)

II. Background

A. Reinforcement Learning Model

Brief and essential details of the MMO environment is as the following (we use similar API as the Gymnasium):

- Map: 128x128 tiles. Tiles types include resource and non-resource, passable and obstacle.
- Agent visibility: 15x15 tiles centered at the agent with up to 100 entities on it.

- Observation space: Action Targets (mask indicating target of some action); Own ID; Current Time Step; Entity List in Vision; Inventory; Market; Task; Whole Map
- Action space: Move; Attack; Manage Inventory; Give; Sell; Buy; Communications

We implemented the DQN model from [3] with modifications:

ALGORITHM 1: Deep Q-Learning

```
Initialize replay memory D to capacity N;
Initialize network Q with random weights θ;
Initialize target network Q̂ with weights θ⁻ = θ;
for episode=1, M do
    for t=1, T do
        for each agent do
            With probability ε select a random action a_t;
            otherwise select a_t = argmax_a Q(s_t, a; θ);
            Execute a_t;
        end
        for each agent do
            Observe reward r_t and next state s_{t+1};
            Store transition (s_t, a_t, r_t, s_{t+1}) in D;
        end
        Sample random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from D;
        Set y_j = { r_j                        for terminal s_{j+1}
                  { r_j + γ max_a Q̂(s_{j+1})   for non-terminal s_{j+1} ;
        Gradient descent on (y_i − Q(s_j, a_j; θ))²;
        Every C steps reset Q̂ = Q;
    end
end
```

Our algorithm is different from the original one for two reasons. First, we do not use the image of the game as the observation. Therefore, we do not require a pre-processing step for each state. Second, we have 8 agents in one team. Therefore, we have to go through each agent and store their timestep replay in the buffer. There are two points to emphasize here. One is that we have to perform all actions of all agents before observing the reward and the next state. This is because theoretically, all agents perform the action at the same time. Another one is that because all agents share the same neural network, for each timestep, we only perform gradient descent once.

For the PPO algorithm, the implemented one by MMO environment is not different from the one introduced in [4]. The major adaptation here is that trajectories are collected from multiple agents instead of a single agent. The objective function is composed of three parts:

$$L^{CLIP+H+V} = L^{CLIP} + h^H - vL^V$$

where

$$L^{CLIP} = \hat{\mathbb{E}}_t[\min p_t(\theta)A_t, clip(p_t(\theta), 1-\epsilon, 1+\epsilon)A_t]$$

$$p_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

$$L^V = \hat{\mathbb{E}}_t[(V_\omega(s_t) - V_t^{target})^2]$$

and the third part H is the entropy bonus whose definition is quite complicated and can be found in [5].

B. Reward and Task Management

We have not discussed the reward function of this environment so far. This is because it is strongly related to the curriculum part. There is a task list predefined by the environment (or can be us). Before each round (episode) of simulation, one task is randomly sampled from the list. Players receive rewards each time when they accomplish the task.

One example task is to collect 5 level-3 shard (which will be dropped by a level-3 NPC). In this round with fixed maximum timestep, every time one agent collects 5 level-3 shard, the team receive rewards. To achieve that, agent should first learn how to upgrade so that it can defeat a level-3 NPC.

Therefore, a good task list (curriculum) should teach agents different kinds of skills to survive in the environment. Trained agents will be finally assessed by some unknown task list proposed by the competition organization. The idea here is that we never know what task is contained there. So, we have to define a list which can be generalized well such that our agents can accomplish unseen tasks on unseen maps.

One interesting finding by our team is that although tasks are randomly sampled from the list, we can still control the learning path of the agent. But this learning path does not mean the order of skill acquired, it means what should be done at different stages. For example, food harvesting task can be conditioned on agent's level and how long the round last. At early stage with low level, agent may harvest as much food as possible and escape from NPCs as soon as possible. At later stage with high level, agent can maintain just enough food and try spend more time to combat to obtain high-value items.

III. Methodology/Approach (≈ 1-2 pages)

A. Environment

To define our reinforcement learning algorithm, we need to provide a detailed description of our environment, including its observation spaces, action spaces, and reward functions. As mentioned previously, the observation space is complex, encompassing agent locations, agent levels, inventories, and markets. For each agent, we can break down its observation space into eight components: Action Targets, AgentId, CurrentTick, Entity, Inventory, Market, Task, and Tile. The specific meanings of each component are outlined in[7]. We will not delve into the detailed composition of each component here for brevity.

```
observation_space(agent_id) = {
    'ActionTargets', # Action masks for
    ↪ the action space items
```

```
        'AgentId', # Id for each agent
         ↪  (including NPCs)
        'CurrentTick', # Current tick
         ↪  (0~1024)
        'Entity', #
        'Inventory', # Agent's inventory,
         ↪  with item's type and level
        'Market', # Market for weapons and
         ↪  tools
        'Task', # Pre-defined task per tick
        'Tile' # Agent's location
    }
```

Regarding the action space, each agent can take multiple actions per tick—one from each category. Each action accepts a list of arguments, where each argument is a discrete variable (such as a standard index for direction, or a pointer to an entity like an inventory item or agent). The action space for a single agent is divided into ten parts: Move, Attack, Target, Use, Destroy, Give, GiveGold, Sell, Buy, and Comm. The meanings of each part are specified in the comments.

```
action_space(agent_id) = {
        'Move', # Move 1 tile in any
         ↪  available direction
        'Attack', # Attack an target -
         ↪  either NPC or from another team
         ↪  - with one of 3 styles
        'Target', # Target of agent's action
        'Use', # Use an item
        'Destroy', # Destroy an item
        'Give', # Give an item to a target
        'GiveGold', # Give gold to a target
        'Sell', # Sell an item from market
         ↪  at a given price
        'Buy', # Buy an item from market at
         ↪  a given price
        'Comm' # Communication number of
         ↪  tokens
    }
```

B. Curriculum Learning

Our reward function is determined by the tasks predefined before training, forming a curriculum. At the beginning of each round, different tasks are randomly generated for different teams. A team receives one point at the end of the round if it accomplishes its assigned task. We assume that the difficulty of tasks is balanced out as teams compete with each other over 1024 rounds, each with randomly assigned tasks.

It is not easy to organize game environment which involves harvesting food, building armors, trading in market, combating with NPCs, upgrading professions, and etc. Fortunately, Pufferlib provides a holistic framework for offline agents' emulation, policy pooling, ranking and vectorization. For managing multiple agents' interactions, one intuitive way is to view their roles as 'teachers' and 'students'. An illustration would be the case of asymmetric self-play, where two agents, Alice and Bob, aims for different goals under the same scenario: Alice opens the door with a key then turns on the light, challenges Bob to achieve the same state and Bob attempts to complete it as fast as he can.

Evidently, bad curriculum is worse than no curriculum. Hence, in our case, one has to be attentive when implementing a teacher-guided model for a partially observable Markov decision process to observe unseen states of a 'student'. Recent studies have found Automatic Curriculum Learning performing well for generalist and multi-goal training, hard tasks' solutions[6] and performance improvements on a restricted task set. For baseline, we apply a fixed curriculum of tasks and OpenELM integration. Given $\mathcal{H}$ with any information about past interactions, we learn a task selection function $D : \mathcal{H} \rightarrow \mathcal{T}$.

$$\text{Obj} : \max_{D} \int_{T \sim \mathcal{T}_{\text{target}}} P_T^N \, dT, \tag{1}$$

In the above objective, ACL algorithm maximizes $\frac{P^N}{T}$ over a distribution of target tasks $\mathcal{T}_{\text{target}}$ (i.e., the agent's behavior on task T after N training steps). Instead of learning solutions separately, it would be better to learn a latent skill space so that every task could be represented in a distribution over skills and thus skills are reused among tasks. Therefore, we consider difficulties of different tasks to initialize ACL, which is followed by reward designs aimed at balancing exploration and exploitation. The typical learning process(LP) is discussed in Algorithm 2. Similar to our baseline model of PPO, it alternates between sampling and and optimizing. Yet the optimization is not done through stochastic gradient ascent of a surrogate objective function as PPO. Instead, it utilizes the model fitting and selection to guide the sampling process of new parameters for the tasks. As for possible developments, it is probable that an adversarial domain generator can strengthen policies trained for Simulation to Reality applications.

ALGORITHM 2: Learning Process

Require: Student $\mathcal{S}$, fitting rate N
for *timestep=1,N* **do**
  Random Sampling $p \in \mathcal{P}$;
  send $E(\tau \sim \mathcal{T}(p))$ to students;
  store its reward and LP;
end
for *inner_loops=1,K* **do**
  Select Model with best Akaike Information Criterion;
  perform epsilon-greedy, send $E(\tau \sim \mathcal{T}(p))$ to students, store its reward and LP;
end

IV. Results and Discussion (≈ 1-2 pages)

For a trained model, we can employ a pre-defined curriculum to evaluate the effectiveness of our model based on the number of living agents and completed tasks. Results obtained from the baseline model and curriculum trained using the PPO algorithm are shown in table I:

At tick 0, the map starts with all 128 agents present and zero tasks completed. As time progresses, the number of completed

Tick	Alive Agents	Task Done
0	128	0
100	50	14
200	37	16
300	24	17
500	16	18
1024(final)	7	19

TABLE I
TICK, ALIVE AGENTS AND TASK DONE

tasks gradually increases, while agents may die due to lack of water and food or being injured in attacks. In theory, we can evaluate the effectiveness of reinforcement learning algorithms or curriculum designs by observing the evolution of the number of completed tasks and surviving agents, as well as their final value at tick 1024.

V. CONCLUSIONS

We devise multiple solutions which aim at improving the performance of multiple agents. It is recognized that different agents don't act independently, but their decision process and consequences are entwined, and they collaborate or compete for various goals. Whilst this would be challenging to model, we could thereby deploy advanced techniques of curriculum learning to explore progress and exploit synergy, and to finally optimize the survival and gain of agents.

REFERENCES

[1] Sutton and Barto. Reinforcement Learning, *MIT Press*, 2020.
[2] Read. Lecture IV - Reinforcement Learning I. In *INF581 Advanced Machine Learning and Autonomous Agents*, 2024.
[3] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236
[4] John Schulman and Filip Wolski and Prafulla Dhariwal and Alec Radford and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017
[5] Bick, Daniel (2021) Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization. Master's Thesis / Essay, Artificial Intelligence.
[6] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. 2021. Automatic curriculum learning for deep RL: a short survey. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI'20). Article 671, 4819–4825.
[7] Joseph Suarez, Neural MMO 2.0 Documentation link.