Social Learning in Multi-Agent Systems

CSC494 Final Report

Yichen Cai and Gabe Guralnick

December 2023

1 Introduction

Navigating real-world problems often requires finding the delicate balance between gains and sacrifices. Coordinating the actions of a system of separately motivated agents is rarely straightforward. This report aims to understand this balance using multi-agent reinforcement learning (MARL) simulations of an environment where there is no clear optimal solution. We find that the presence of an "influencer" in the environment, capable of directing other agents to fulfill certain roles needed to perform well at the task, is a useful construct for achieving optimal performance.

2 Related Work

2.1 Sequential Social Dilemmas

Sequential social dilemmas (SSDs) are multi-agent environments in which agents face contrasting individual and collective motivations. These environments feature situations in which all agents acting greedily and selfishly would lead to low collective and individual reward, so agents must choose to compromise in order to achieve success. Our analysis focuses specifically on the Cleanup environment, though our modeling could be generalized to other SSDs.

2.1.1 The Cleanup Environment

Our analysis focuses on the *Cleanup* sequential social dilemma. This environment consists of a simulated world where there are multiple agents. In this world, there are two areas called the *orchard* and the *river*. The orchard is where apples will grow, and the river is where waste will accumulate. The system aims to consume as many apples as possible within a certain period (an episode). However, apples will only grow when the waste level in the river is below a certain threshold, and the rate of apple growth is inversely related to the density of waste present in the environment. So, although there must be agents in the orchard to collect apples, some of the population must also be controlling the waste level by cleaning up the river. This cleanup action is not directly rewarded. This creates a dilemma, where some agents must be assigned to do the waste-cleaning task for future benefits [1]. In this report, we demonstrate that finding the balance between the number of agents working as *pickers* and *cleaners* is essential to finding an optimal solution. The formal specification of the environment is given in Section 5.

2.2 The Role of an Influencer in a Community

[5] performed an analysis of the role of social influence in communities; specifically, its role in online social networks. They viewed online social networks as a content market in which community members both produce and consume content. People consume content that is similar to their main interests by allocating the following rate to producers. The amount of attention that consumers can allocate is limited, so they

must prioritize the content they find most interesting. People produce content that achieves the best balance between being similar to their main interests and netting them the most social support (following rate) from the community. In this setting, the influencer acts as an information proxy: because producers are unable to note the specific interests of every consumer in the market, they can gauge consumer interest based on the type of content to which the influencer pays attention. We adopt this role of the influencer as a guide for agents to know what is best for them to do in terms of the overall community welfare and believe that it can provide a useful mechanism to improve coordination among agents in a MARL setting.

2.3 Influence in Multi-Agent Learning

[1] investigated the potential of social influence as an intrinsic reward in a MARL setting to motivate agents to work together more effectively. They defined influence as the amount of impact one agent's choice of action has on another's and provided agents in sequential social dilemma settings like *Cleanup* with an additional reward (on top of the reward for apple collection) based on how strongly their choices of actions influenced other agents' choices. Specifically, for influencing agents they performed a counterfactual simulation of how other agents would have acted if the agent had chosen other available actions; the influence reward was then the divergence between this counterfactual distribution and the true result. They found that this mechanism was an effective mechanism for motivating cooperation among the agents. However, this definition of influence is a heuristic and does not motivate agents to specifically work towards improving community welfare, a deficit we hope to address with our use of an influencer mechanism.

2.4 Roles in Multi-Agent Settings

Previous work has investigated the concept of 'roles' for agents in a multi-agent setting.

RODE: Learning Roles to Decompose Multi-Agent Tasks RODE [2] is a framework under which complex tasks are decomposed into sub-tasks associated with different roles, allowing for more efficient learning and scalability. The paper proposes a two-level learning hierarchy, where a role selector determines role assignments in a smaller role space, and role policies learn in reduced observation-action spaces. Our role decision algorithm (the influencer) is similar to this role selector. RODE defines a role representation as the average of all the available actions for the role. A fully connected network selects the role based on an agent's action-observation history, and a Q-network is used to make predictions on the expected returns given the selection. A global mixing network is introduced to prevent conflicts and duplication of actions due to the concurrent decision-making process. In contrast, in this report, we utilized a TD learning technique for estimating the future return of a certain state and we directly encoded the role allocation information into the state of the environment. Both RODE and our method aim to use the role selection process to decompose the multi-agent task into sub-tasks, each associated with a role, and the results both show that identifying the correct roles for agents drastically simplifies the problem and improves the result.

DRDA: Dynamic role discovery and assignment in multi-agent task decomposition DRDA [3] proposes a new framework for learning dynamic role discovery and assignment. It introduces an action encoder and defines and classifies roles based on action differences and contributions. It proposes a representation-based role selection policy that assigns agents with similar abilities to play the same role. Agents playing the same role share their learning, and different roles correspond to different action spaces.

ROMA: Role-oriented MARL framework ROMA [4] introduced a role-oriented multi-agent system. Each agent has a role embedding in which information about the role is encoded. The embedding serves as an input to Q-learning to inform the actions for this agent. The role embedding generation is regularized to make sure a stable and specific role embedding is produced to determine the agent's actions. The paper suggested an abstract role embedding without a specific meaning assigned to it; the embedding acts as information to guide the decision-making process of the Q-learning. With the gradient flow through the

embedding process, the embeddings themselves will be more informative to guide the prediction of the Q-value. The method does not have an explicit role selection process; instead, it has the role representations as pieces of information. This provides a more fine-grained role but could make learning optimal actions for each role more challenging for the downstream network. In contrast, our approach defines roles with semantic meaning in terms of the two objectives that are present in the environment. A comparison between these approaches could be an interesting avenue for future work.

3 Approaches

Our investigation of the problem focused on two major approaches. First, we evaluated the performance of centralized versus decentralized learning methods for agent action policies, specifically focusing on Q-learning. We then explored the concept of 'social influence' in the MARL setting, focusing on the potential role of an influencer agent in ensuring optimal coordination among the other agents.

3.1 Centralized vs. Decentralized Learning

We first attempted is train agent action policies using Q-learning. Specifically, we compared the performance of centralized and decentralized Q models. The centralized Q-learning employed a single neural network that evaluates for all agents what the future reward would be for every possible move. Then it picks the action with the highest estimated reward. It is trained on the experience of all the agents as well. Similarly, the decentralized model also uses the same decision-making algorithm, but there is one network for each agent. Each network is trained only on one agent's experience. The networks used in both methods have the same architecture.

The result of experiments with 10 agents shows that centralized learning can collect around 800 apples and decentralized learning can achieve just below 1000 apples within 1000 time steps. For reference, 10 fully random agents can consume around 500 apples during the same period. It shows the potential of a decentralized learning algorithm, where individual models together could create a result that is better for the total reward than a single coordinator.

The experiments also revealed the major drawback of the model, agents lack the of understanding of their objectives and the future potential of a certain action. Agents, if not rewarded with dirt-cleaning, will not go for the cleaning task which causes the apple to grow very slowly. That gives them almost no chance to obtain a reward for most of the time. Even when they are rewarded with apples, they still fail to understand the importance of cleaning dirt cells. This inspired us to develop a model to specifically tackle the problem of assigning the optimal roles for each agent.

3.2 Agent Roles and the Influencer

Our use of an influencer to improve the coordination of the agents in the market is inspired by an analysis of online social networks. [5] found that an influencer can act as a useful information proxy between producers and consumers of content when attention capacity is too limited for agents to individually keep track of everything going on in the market. We believe that this idea of an information proxy could be useful in the MARL setting as individual agents may have incomplete observations and be unable to think about "the bigger picture" of the task at hand. An influencer, acting as a proxy for the overall environment state, could direct agents to fulfill the most beneficial roles to maximize collective welfare.

As mentioned in Section 2.1.1, the two objectives in the cleanup environment are picking apples and cleaning dirt. We decided that these two objectives could form useful role targets; so, we define 'pickers' as agents assigned to pursue apples and 'cleaners' as agents assigned to pursue dirt. To assign agents to roles during

each environment time step, we use an approach similar to RODE [2] where a role selector (influencer) determines role assignments while agent actions within a role are determined by role-specific policies. Overall, the goal of the influencer is to optimally assign agents between the two roles to maximize the expected future reward. This approach could be adapted to any environment in which distinct objectives can be identified.

We additionally split up the tasks of determining the *quantity* of agents assigned to each role and determining the *specific* agents assigned to each role based on the desired quantities. For the quantity determination, we evaluated three main policies: a fixed proportion policy, a heuristic policy, and a trained reward-optimizing policy using temporal difference (TD) learning. To determine the specific agent assignments, we used a greedy approach in all three cases; a more intelligent model for this decision is an open avenue for future research. Future work could also explore performing role assignments without previously determining desired quantities.

A note about this approach is that we do not provide agents with any amount of agency in their role assignments. In [5], although the influencer's decisions were certainly a motivating factor for the content producers, they were not required to produce content in line with the influencer's interests. In the current analysis, however, the influencer's decisions are binding. Future work could investigate a per-agent decision problem related to whether they wish to follow the influencer's decision or pursue their independent interests.

4 Results

Overall, we found that the use of decentralized agent action policies, as well as the use of an influencer agent, were beneficial in achieving greater rewards on the task. We also found that using a more adaptive, learned policy for the influencer's role assignments was more powerful than heuristic or fixed proportion policies. We will discuss the details of these results more in Section 7.

4.1 Centralized versus Decentralized

Below is the comparison between centralized and decentralized training, it shows that decentralized training can achieve similar (or even better) results given the same setting. However, the agents overall lacks the understanding of the benefits of cleaning waste, as it is not a rewarded action. Without a strong motivation to go for the dirt, they will see few apples and thus having few chances to incorporate the experience of collecting apples into the training.

Random	Centralized	Decentralized
500	800	950

Table 1: Results for centralized and decentralized training

4.2 Roles and the Influencer

The experimental result shows that the performance of the U-Model is consistently better than the other two methods under different environment settings and model parameters. It outperforms the Fixed and the Heuristic models under different numbers of agents; these models provide baseline estimates for the potential of the influencer agent, and it is clear that the more adaptive U-Model is beneficial in comparison.

4.2.1 Simplified Environment

In the simplified environment, we see the advantage of using the reward-maximizing U-Model, regardless of the number of agents present in the environment. The reward collected by each method for varying numbers of agents is shown in Table 2:

	1 Agent	2 Agents	10 Agents	20 Agents	100 Agents
Fixed	0	16.47	2701.25	4217.95	6862.40
Heuristic	0	104.81	2701.23	4179.33	6815.89
U-Model	2.50	288.00	2810.10	4330.00	6909.00

Table 2: Results from the Simplified Environment

4.2.2 1-Dimensional Environment

The U-function also proved to be powerful in the 1-D environment. Tested with 10 agents, the result on average is above the other two methods. Since the 1-D environment has randomness in terms of apple/dirt spawn location, the results are compared using averages.

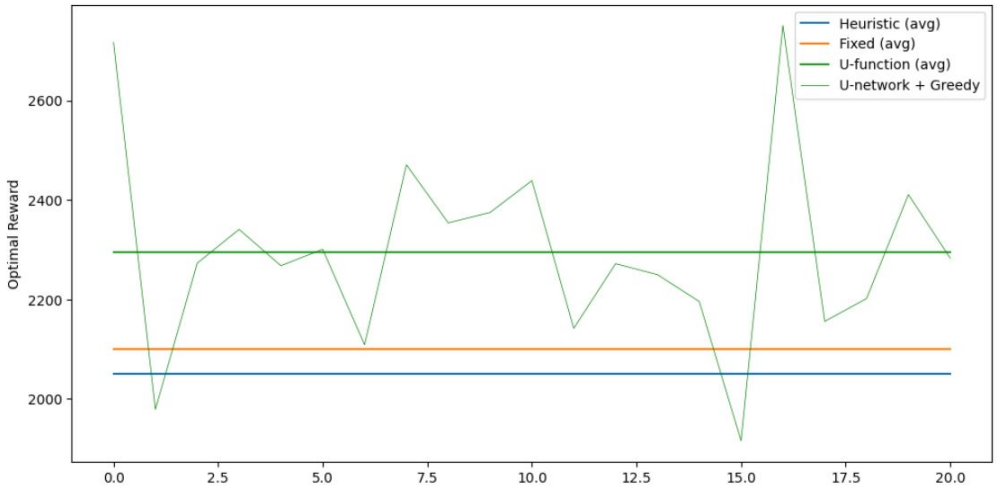

Figure 1: Result in 1-D Environment

5 The Cleanup Social Dilemma

The *Cleanup* environment consists of two areas: the orchard and the river. Agents are classified as being in either one of the areas at a given time. Waste appears at the river at a constant rate during each time step if the current waste density is below a certain threshold. In the orchard, apples grow at a rate inversely related to the current waste density; more waste present means fewer apples will spawn. We refer interchangeably to 'dirt' and 'waste' as this limiting factor. Agents can seek out either apples or dirt, but a reward is only obtained for collecting apples. The collective objective is to collect as many apples as possible within an episode. During each time step, agents act based on the current state and the environment state will update accordingly, generating a reward.

We performed an analysis on the zero-dimensional, one-dimensional environment for the effectiveness of a role allocation influencer, and used two-dimensional environment for analyzing the centralized and decentralized training of agents.

5.1 General Environment Details

The specification in this section is independent of the particular dimensionality of the environment.

There are several set variables for the environment: $A_D, A_A > 0$, which denote the number of cells present in the river and orchard, respectively; T_D, the depletion threshold; T_R, the restoration threshold; and $P_a, P_d > 0$, the default apple and dirt spawn probabilities, respectively. In line with previous work [1], we set $T_D = 0.4, T_R = 0, P_a = 0.05, P_d = 0.5$. We also set $A_D = A_A = 150$ for most testing. We'll also refer to the number of agents N, though this is not an environment setting.

5.1.1 State Space

The state of the environment is a tuple $s = (a, d, p, c, m^a, m^d, m^p, m^c)$ encoding the number of apples a, the number of dirt d, the number of pickers p, and the number of cleaners c. The values m^a, m^d, m^p, and m^c are maps of the locations in the environment where there are apples, dirt, pickers, and cleaners, respectively. The dimensionalities of these maps are specific to the dimensionalities of the environments, so they will be defined more formally in later sections.

The state space $\mathbf{S}$ contains all tuples of the above form that satisfy the following conditions. First, $0 \leq a + p \leq A_A$, $0 \leq d + c \leq A_D$, and $p + c = N$, so agents and objectives must fit in their corresponding regions. We also require that there not be an agent and an objective (apple or dirt) in the same spot in the same region.

Notably, for the state s_t during time step t, the values p_t and c_t refer to the number of pickers and cleaners at the *start* of the time step; i.e. the number of agents who were assigned to each role during the time step $t - 1$.

5.1.2 Action Space

During each time step, each agent decides on an action tuple $a = (r, d)$, where $r \in \{p, c\}$ is the agent's role and $d \in D$ is the agent's movement. The definition of the movement-action space D is specific to each variation of the environment and will be formalized in later sections. The action space is then the set $\mathbf{A} = \{p, c\} \times D$. Agent actions are collected to form a joint N-dimensional action vector $\mathbf{a} \in \mathbf{A}^N$, which motivates a transition in the environment and a reward for the time step.

5.1.3 Decision Problems

We refer above to each agent "deciding" on a role r. In reality, our goal is for these decisions not to be made on a per-agent basis; instead, the influencer is the one to decide which agents should be pickers and which should be cleaners. Overall, we have identified three decision problems that must be addressed during each time step:

1. Determining the *quantity* of agents assigned to be pickers and cleaners.
2. Determining the *specific* agents assigned to each role based on their positions and the positions of other agents.
3. Determining the movements of each agent.

The goal of the Influencer is to perform steps 1 and 2. In Section 6 we discuss the policies evaluated for this task as well as for step 3, determining individual agent movements given their roles.

Each of these decision problems lends itself neatly to a particular dimensionality for the environment. To test the utility of policies for determining the quantity assigned to each role, we used a zero-dimensional environment (Section 5.2) wherein step 1 is the only decision to be made. The one-dimensional environment (Section 5.3) expands the scope to include step 2 and a simplified form of step 3. Finally, the full two-dimensional environment (Section 5.4) includes all three decision problems; however, the result from this environment is limited.

5.1.4 State Transition

The state of the environment transitions from s_t to s_{t+1} in response to the joint action $\mathbf{a}_t$. This transition consists of three phases: movement, consumption, and spawning. In this section we'll provide more intuitive explanations of this process; the sections for specific variations of the environment will go into more formal detail about how all these values are calculated.

Movement In general, agents can choose either to move between regions or to move within their region. These movements are conditional on there not already being another "blocking" agent in the spot to which one tries to move. It should be noted that an agent who switches regions does not additionally move within the new region in this particular time step.

Consumption Once all agents have moved, we evaluate the number of apples and dirt consumed by checking where there is now overlap between agent positions and apple/dirt positions. Wherever there is overlap, we count that objective as 'consumed' and remove it from the map.

Spawning Let a'_t and d'_t be the number of apples and dirt, respectively, present in the environment after the consumption phase. Similarly, let p'_t and c'_t be the number of pickers and cleaners, respectively, present after the movement phase. We define $dirtDensity = \frac{d'_t}{A_D}$ to be the density of dirt within the river region. We then have that

$$d_{t+1} = \begin{cases} \min(A_D - c'_t, d'_t + P_d) & \text{if } dirtDensity < T_D \\ d'_t & \text{otherwise} \end{cases}$$

$$a_{t+1} = \begin{cases} a'_t + (1 - \frac{dirtDensity - T_R}{T_D - T_R}) \cdot P_a \cdot (A_A - a'_t - p'_t) & \text{if } dirtDensity < T_D \\ a'_t & \text{otherwise} \end{cases}$$

Basically, at most one dirt will spawn per time step if there is space and if $dirtDensity < T_D$, while the number of apples spawned is inversely related to the current $dirtDensity$ within the window $[T_R, T_D]$. Notably, in the one-dimensional and two-dimensional environments, we round the values of d_{t+1} and a_{t+1} so they are integers, since in those environments we also actually place the apples and dirt on the maps.

5.1.5 Reward

The immediate reward is calculated based on the number of agents whose position overlaps with the position of an apple after the 'movement' step; it is equal to the number of apples consumed. Each agent receives an individual reward equal to whether or not it consumed an apple, and the collective reward is the sum of all these rewards.

5.1.6 Simulation Details

The environment is simulated using discrete time steps. Initially, half the cells in the river area are filled with dirt and there are no apples in the orchard. Each time step consists of a sequence of operations:

1. Given the current state, determine the allocation of the agents and the corresponding actions for individual agents (including role switches and movements).
2. Once the action of each agent is computed, we execute the joint action. Notably, agent movements are evaluated discretely, so agents with lower ID values will (in the most literal sense) move first; however, our intent is for this to be interpreted as all agents moving at once.
3. The state transition and reward are executed in response to the joint action, following the general process in Section 5.1.4 and Section 5.1.5.

4. Apples and waste are spawned based on the state of the environment (after agent movements and consumption), following the process from Section 5.1.4.

We repeat this process for an episode (typically 1000 time steps). The specifics of each step of the process depend on the specific variation of the environment being simulated, but they all follow this general discrete sequence.

5.2 Zero-D Cleanup Environment

We first evaluated a simplified environment to test exclusively the effect of role allocation. The simplified environment does not include agent movement actions; instead, agents simply choose roles and then the amount of apples or dirt consumed is determined probabilistically. Specifically, the apples collected and the waste cleaned are the expected values of a hyper-geometric distribution, i.e., the expected amount of overlap between agents and objectives if the agents were to be randomly distributed in the two regions. As such, the simplified environment treats the quantity of apples and dirt as continuous values.

5.2.1 State Space

Since this variation of the environment is zero-dimensional, we remove the maps m^p, m^c, m^a, and m^d, so the state is simply the tuple $s = (a, d, p, c)$.

5.2.2 Action Space

As mentioned, in this environment there are no movement actions, so an agent's action is simply $a = r \in \{p, c\}$. All actions are still collected into the joint action vector $\mathbf{a}$ that motivates a transition in the environment and a reward.

5.2.3 Decision Problem

The goal of the simplified environment is to focus specifically on the task of determining the optimal number of agents for each role. We can represent this allocation as a tuple (r_p, r_c) for the number of pickers and cleaners, respectively.

5.2.4 State Transition

Given the action vector $\mathbf{a}_t$, we have that $p'_t = \sum_{i=1}^{N} \mathbb{I}(\mathbf{a}_t^{(i)} = p)$ and $c'_t = \sum_{i=1}^{N} \mathbb{I}(\mathbf{a}_t^{(i)} = c)$ are the number of pickers and cleaners, respectively ($\mathbb{I}(x)$ is the identity function that outputs 1 if x is true and 0 otherwise). We can then calculate apple and dirt consumption as follows:

$$a'_t = a_t - \frac{a_t * p'_t}{A_A},\ d'_t = d_t - \frac{d_t * c'_t}{A_D}$$

These values represent the expected amount of apples and dirt consumed given the current amount of each in each region and the number of agents pursuing each. As mentioned earlier, the values $\frac{a_t * p'_t}{A_A}$ and $\frac{d_t * c'_t}{A_D}$ are the expected values of hypergeometric distributions of the probable overlap between apples and pickers (or dirt and cleaners) were agents and objectives all to be uniformly randomly distributed in their respective regions.

Finally, new apples and dirt will be spawned according to the formulas given in Section 5.1.4. This gives us the values a_{t+1} and d_{t+1}.

The new state is then $s_{t+1} = (a_{t+1}, d_{t+1}, p'_t, c'_t)$.

5.2.5 Environment Reward

The reward is the total number of apples consumed at this time step:

$$Reward(s_t, \mathbf{a}_t) = \frac{a_t * p'_t}{A_A}$$

5.3 1-Dimensional Cleanup Environment

5.3.1 Environment State

Recall that the state of the environment is the tuple $s = (a, d, p, c, m^a, m^d, m^p, m^c)$, encoding the number of apples a, the number of dirt d, the number of pickers p, and the number of cleaners c. The values m^a, m^d, m^p, m^c are maps of the locations in the environment where there are apples, dirt, pickers and cleaners, respectively. In the one-dimensional variation of the environment, these maps are 1-dimensional arrays (stripes). For instance, $m^a[i] = 1$ if the i-th position in the apple region contains an apple, $m^a[i] = 0$ otherwise; similarly, $m^p[i] = j$ if the picker j is currently located in the i-th position in the apple region.

To ensure the the counts a, d, p, c and the maps m^a, m^p, m^d, m^c are consistent, we also must have that

$$\sum_{i=0}^{A_A-1} m^a[i] = a, \sum_{i=0}^{A_D-1} m^d[i] = d, \sum_{i=0}^{A_A-1} \mathbb{I}(m^p[i] \neq 0) = p, \sum_{i=0}^{A_D-1} \mathbb{I}(m^c[i] \neq 0) = c,$$

where $\mathbb{I}(x)$ is 1 if x is true and 0 otherwise. Finally, we enforce that $\{i \in \{0, \ldots, A_A - 1\} : m^a[i] \neq 0 \wedge m^p[i] \neq 0\} = \emptyset$ and $\{i \in \{0, \ldots, A_D - 1\} : m^d[i] \neq 0 \wedge m^c[i] \neq 0\} = \emptyset$. The other constraints on the number of pickers and cleaners themselves hold as well (Section 5.1.1).

5.3.2 Decision Problems

This version of the environment has all three decision problems discussed in Section 5.1.3: the quantity of agents to assign to each role, the specific agents to assign to each role, and the movements for each agent within their assigned region.

5.3.3 State Transition

The state transition function consists of three stages as outlined in Section 5.1.4: agent movements, apple/dirt consumption, and apple/dirt spawning.

Agent movements and Apple/Dirt Consumption Given the current state $s_t = (a_t, d_t, p_t, c_t, m^a_t, m^d_t, m^p_t, m^c_t)$, and an action $\mathbf{a_t} = \{a^t_i, 1 \leq i \leq N\}$. We sequentially apply the action for each agent. For agent i's action $a^t_i = (r^t_i, d^t_i)$. It follows the Algorithm 1.

To summarize, for each agent, we apply Algorithm 1 to first check if its action contains a role switch. If there is a role switch we try to move the agent to the new role at the same relative location as it was before in its original role, and we consume the resource there if exists. Otherwise, if there is no role switch, we take the d^t_i as the movement it will make within its original role. It can either go up, down, or stay. And if there is any resource at the position of the agent after its movement, it will be consumed. Any of the steps (role switch or movement) will be blocked if there is another agent in the position that the current agent wants to move to.

Algorithm 1 Algorithm for agent movements and apple/dirt consumption

r_i^{t-1} = current role of agent i, either p or c
r_i^t = target role of agent i given in the action a_i^t
obj_i^{t-1} = current objective of agent i, depending on its current role, either a or d
obj_i^t = target objective of agent i, depending on its target role r_i^t
loc = location of agent i in its original role, $\{loc \,|\, m_t^{r_i^{t-1}}[loc] = i\}$

if $r_i^{t-1} \neq r_i^t$ **then**
  // role switch
  if $m_t^{r_i^t}[loc] = 0$ **then**
    // no agent at new location, move to new location with new role
    $m_t^{r_i^t}[loc] \leftarrow i$
    $m_t^{r_i^{t-1}}[loc] \leftarrow 0$
    // decrement the count of the original role, increment the new role
    $(r_i^{t-1})_t \leftarrow (r_i^{t-1})_t - 1$
    $(r_i^t)_t \leftarrow (r_i^t)_t + 1$
    // decrement the count of apples or dirt if the new location has apple or dirt
    if $m_t^{obj_i^t}[loc] = 1$ **then**
      $m_t^{obj_i^t}[loc] \leftarrow 0$
      $(obj_i^t)_t \leftarrow (obj_i^t)_t - 1$
    end if
  end if
else
  // keeps the original role
  if $m_t^{r_i^t}[loc + d_i^t] = 0$ **then**
    // no agent at new location, move to new location
    $m_t^{r_i^t}[loc + d_i^t] \leftarrow i$
    $m_t^{r_i^t}[loc] \leftarrow 0$
    if $m_t^{obj_i^t}[loc + d_i^t] = 1$ **then**
      // resource exists, consume
      $m_t^{obj_i^t}[loc + d_i^t] \leftarrow 0$
      $(obj_i^t)_t \leftarrow (obj_i^t)_t - 1$
    end if
  end if
end if

We can equivalently express this process mathematically. Let $l_t^{(i)}$ be the location of agent i at the start of time step t. So $0 \leq l_t^i < A_D$ if agent i was previously a cleaner while $0 \leq l_t^i < A_A$ if agent i was previously a picker. For now, we'll use A to refer to either A_D or A_A.

If the agent's role action $r_t^{(i)}$ is not equal to their previous role $r_{t-1}^{(i)}$, we must check whether it is possible for the agent to switch regions. If $m_t^{r_t}[l_t^{(i)}] \neq 0$, then there is another agent in the position where the agent would move were it to switch roles, so the role change is impossible. In this case, the agent does not switch roles and we have $l_{t+1}^{(i)} = l_t^{(i)}$.

If $r_t^{(i)} = r_{t-1}^{(i)}$, then given the agent's direction action d_t^i, we define the agent's next location as

$$l_{t+1}^i = \begin{cases} l_t^i + d_t^i & \text{if } 0 \leq l_t^i + d_t^i < A \text{ and } l_{t+1}^j \neq l_t^i + d_t^i \text{ for all } j < i \\ l_t^i & \text{otherwise} \end{cases}$$

This formula encodes the requirements that agents cannot move to a space already occupied by another agent and must stay within the confines of the environment. Then, we can simply define

$$a'_t = a_t - \sum_{i=1}^{N} \mathbb{I}(r_t^{(i)} = p \wedge m^a[l_{t+1}^i] = 1)$$

$$d'_t = d_t - \sum_{i=1}^{N} \mathbb{I}(r_t^{(i)} = c \wedge m^d[l_{t+1}^i] = 1)$$

We update m^a and m^d to become $m^{a'}$ and $m^{d'}$ by setting those indices where agents are now located to be 0. Similarly, we update m^p and m^c to become $m^{p'}$ and $m^{c'}$ by setting those indices where agents were previously located to be 0, and setting the indices where agents are now located to be equal to the agent's id.

Apple and Dirt Spawning With $s'_t = (a'_t, d'_t, p'_t, c'_t, m_t^{a'}, m_t^{d'}, m_t^{p'}, m_t^{c'})$, we now can determine how many apples and dirt will be spawned. The values a_{t+1} and d_{t+1} are calculated according to the process in Section 5.1.4. We can then define $Spawn_{apple} = a_{t+1} - a_t$ and $Spawn_{dirt} = d_{t+1} - d_t$. Then we randomly sample $Spawn_{apple}$ positions from $\{i : m_t^a[i] = 0 \wedge m_t^p[i] = 0\}$ (i.e., positions where there is not currently an apple or picker), and set them to 1. After the operation, $m_t^{a'}$ becomes m_{t+1}^a Similarly, we sample $Spawn_{dirt}$ positions from $\{i : m_t^d[i] = 0 \wedge m_t^c[i] = 0\}$ (i.e., positions where there is not currently a dirt or cleaner), and set them to 1. After the operation, $m_t^{d'}$ becomes m_{t+1}^d

This entire process generates the new state $s_{t+1} = (a_{t+1}, d_{t+1}, p'_t, c'_t, m_{t+1}^a, m_{t+1}^d, m_t^{p'}, m_t^{c'})$.

5.3.4 Immediate Environment Reward

The immediate reward here is the same as in Section 5.1.5. Specifically, it is given by

$$Reward(s_t, \mathbf{a_t}) = a_t - a'_t$$

the number of apples consumed during the time step as a result of the action $\mathbf{a_t}$.

5.4 2-Dimensional Cleanup Environment

The 2-dimensional Cleanup environment is the full version of the environment following the cleanup game in [1]. In this environment, the maps m^a, m^d, m^p, m^c are matrices rather than arrays that encode two-dimensional representations of the regions. Agents can move in four directions. Our testing within this environment was limited to only a general test of centralized vs decentralized agent action policies; we did not evaluate the use of roles in this environment. During each time step, each agent receives their Manhattan distance to their closest apple and waste cell in the 2-D grid world. Note that in this world, river and orchard are connected by empty cells in the middle. Agents do not switch locations/areas, they only make a move by relocating to the adjacent 4 cells. The objective is still getting the most apples, and each agents only get a reward if they step onto a cell containing an apple. The number of new apples and dirt spawned is random, following Section 5.1.4, the probability of dirt and apple spawning when the $dirtDensity < T_D$ is P_d and $(1 - \frac{dirtDensity - T_R}{T_D - T_R}) \cdot P_a$ respectively. Note that dirt spawning is applied once each time step (max. 1 dirt will spawn within a single time step), while apple spawning probability is applied to each cell in the orchard, every cell in the orchard has this probability of growing an apple within a single time step.

6 Policy Details

Recall the three decision problems present in the cleanup environment from Section 5.1.3: the *quantity* of agents to assign to each role, the *specific* agents to assign to each role, and the movements of each agent given their role. We make use of separate policies for each of these problems.

6.1 Number of agents per role

Given the state s, we must first determine the number of agents that should be assigned to each role. We evaluated three different policies for this decision: a fixed proportion policy, which maintains a constant proportion of agents following each role; a heuristic policy, which matches the ratio of pickers and cleaners to that of apples and dirt; and a reward-maximizing policy, which uses a neural network to perform function approximation of the expected reward for each state and chooses quantities to maximize this reward.

We refer to the output of these policies as an allocation vector $\alpha(s) = (p, c)$, where p is the number of pickers and c is the number of cleaners; we have that $p + c = N, 0 \leq p \leq N, 0 \leq c \leq N$.

6.2 Fixed Proportion Policy

The fixed proportion policy maintains a constant proportion p of agents assigned to be pickers. Then $1 - p$ is the proportion assigned to be cleaners. We then have that

$$\alpha_{fixed}(s) = (p \cdot N, (1 - p) \cdot N)$$

6.3 Heuristic Policy

The Heuristic method allocates the agents by matching the ratio of pickers and cleaners to the ratio of apples and waste in the environment. The intuition is that we should allocate more agents to the area where the resource (apple or waste) is more abundant. Given the state $s = (a, t, p, c)$ and number of agents N, we have that

$$\alpha_{heuristic}(s) = (N - (\frac{d}{a+d} \cdot N), \frac{d}{a+d} \cdot N)$$

So, for $\alpha_{heuristic}(s) = (p, c)$ we ensure that $\frac{p}{c} \approx \frac{a}{d}$.

6.4 Reward-Maximizing (U-Function) Policy

To more adaptively and intelligently tackle this decision problem, tried training a function approximator to estimate the value of a state so that we could simply select the assignment quantities to maximize expected values. We call this function the U-Function.

The U-function is an estimate of the current and future reward from a state. Given policy π, the U-function is calculated by:

$$U^{\pi}(s_t) = Reward^{\pi}(s_t) + \gamma U^{\pi}(s_{t+1})$$

The U-function calculates the expected total return from following the policy π starting at state s with future reward discounted by γ.

The reward-maximizing policy greedily selects the allocation of agent roles to maximize the prediction of the U-function, following the following process:

Let $\boldsymbol{\alpha} = \{(p, c) | p + c = N, 0 \leq p < N, 0 \leq c < N\}$ which is a set of all possible allocations of the number of pickers and cleaners. For each potential allocation $\alpha = (p, c)$, we counterfactual simulate the assignments, movements, corresponding state transition, and reward that would occur were we to use the

allocation. The assignments and movements are simulated using the processes outlined in Section 6.5 and Section 6.6; we define $\mathbf{a}(\alpha, s)$ to be the counterfactual action vector. The state transition to s_{t+1} and reward $Reward(s_t, \mathbf{a})$ are simulated using the processes in Section 5.1.4 and Section 5.1.5. Our reward-maximizing policy can then be defined as follows:

$$\alpha_U(s_t) = \pi(s_t; U) = argmax_{\alpha \in \boldsymbol{\alpha}} Reward(s_t, \mathbf{a}(\alpha, s_t)) + \gamma U(s_{t+1})$$

where γ is the discount factor, s_{t+1} is the next state based on action $\mathbf{a}(\alpha, s)$. This policy chooses the allocation of pickers and cleaners that maximizes the U-function value based on the way we generate actions ($\mathbf{a}$) from the allocation (α).

6.4.1 U-Model Updates

The U-function is updated using TD learning updates to improve its accuracy in estimating current and future reward:

$$U(s) \leftarrow U(s) + lr * (Reward(s, action(\pi(s; U), s) + U(s') - U(s))$$

where s' is the actual next step of the environment after applying $action(\pi(s; U))$ on state s, lr is the learning rate.

6.5 Role Assignments

With the quantities determined, we need to determine the individual assignment of roles for all agents. We greedily assign the pickers and cleaners (following the number determined in section 6.1) based on the objectives that agents are closest to, with closeness to apples being prioritized as number of apples collected is what determines the reward. Let (p, c) be some allocation determined by step 1, we assign the p agents closest to apples to be pickers, and the remaining c agents are assigned to be cleaners. Note that when assigning pickers, we compare the distance to the closest apple for all agents, i.e., waste cleaners are considered as well. We calculate the distance by treating the positions of the cleaners as if they are in the orchard. For example, if a cleaner is at position 10 in the river, and there is an apple at position 9 in the orchard (and no apple at position 10), the distance for this cleaner agent to its closest apple is: $10 - 9 = 1$ (unit).

6.6 Agent Movements

What remains is a direction of movement for each agent. We assume each agent will greedily pursue the closest resource (apple or waste) based on their new role assignment (picker or cleaner). Note that any movement from agents who just switched roles based on the policy in Section 6.5 will be ignored.

7 Discussion and Future Work

For the simplified environment, because of the deterministic nature of the environment, Fixed and Heuristic methods' performances are also deterministic. It is clear that in all cases, the U-Model performs the best. However, as the number of agents increases, the gap between the three methods generally shrinks. This trend is expected as more agents in the environment makes collecting any resource "easier". More precisely, the expected value of any resource collected increases proportional to the increase in the agent number.

We also observed that controlling the number of waste cleaners is crucial to the success of the system. Too few cleaners will have the waste accumulate and apples stop growing, and too many cleaners will compromise the apple collection speed. The U-Model provides the best outcome in terms of "waste control". The experiment with 20 agents (Figure 2) illustrates this effect.

The fixed and heuristic method controls the dirt amount at a very stable level after the initial stages. The growth of dirt and the agents consuming the dirt strikes a balance. However, the two methods (tuned

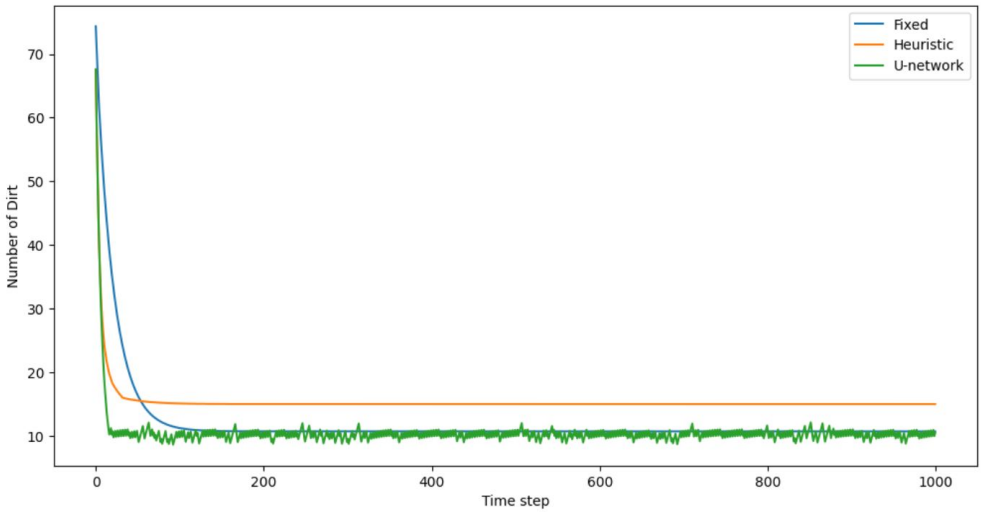

Figure 2: Dirt Quantity with 20 Agents

to optimal for the fixed method) converge to different levels of dirt. The Heuristic allocates fewer agents on dirt - it has a balance that is higher than the (optimal) Fixed method. This is because the Heuristic does not consider future growth of the apple if the dirt is kept low, it simply allocates based on the current ratio which stabilizes at an undesirable place. The U-function learns the best allocation and follows closely to the Fixed method which is manually tuned to the best allocation. It is also worth noting that the U-Model still outperforms the Fixed method as it consumes the initial waste in the river very quickly so the apples start to grow early in the episode, causing a quicker convergence to the optimal allocation of roles. The Fixed method cannot do that because if we assign that many dirt cleaners, there will not be enough apple pickers later. Since the dirt grows at a slower rate (in general), once it is at a low level, it requires much fewer agents to maintain, which makes the U-model an optimal strategy.

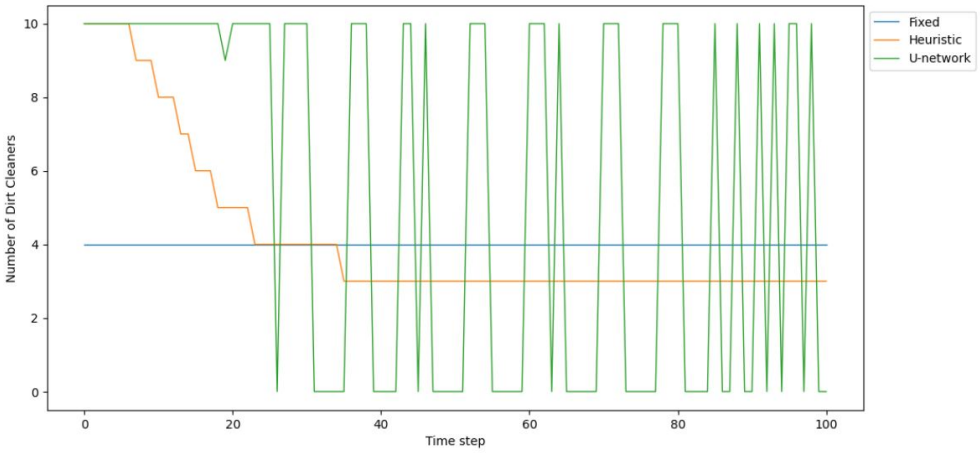

Figure 3: Agent Allocation with 10 Agents

The U-model has a very aggressive allocation policy as shown in Figure 3, i.e., it tends to allocate all agents to one task to quickly consume a certain number of resource and then switch everyone back. This

is evident in all the experiments. This is the better strategy as we focus all the agents immediately on the most urgent task. However, regularization is needed if switching agents is costly.

We have also experimented with different spawning rates for the two types of resources (apples and waste), to see if the U-Model (and the two baselines) can adapt. For this, we evaluated varying numbers of agents in the environment and increased the spawning rates depending on the number of agents. We found that only the U-Model can adapt effectively to increased spawn rates. The reward is shown in Table 3. In Figure 4,

	10 Agents (1x spawn)	20 Agents (2x spawn)	100 Agents (10x spawn)
Fixed	2701.25	5697.78	36976.15
Heuristic	2701.23	5622.22	35233.01
U-model	2810.1	5800.83	40153.74

Table 3: Reward when the spawning rate increased proportionally to the number of agents

we see the U-Model controls the dirt at an optimal level regardless of the spawning rate, demonstrating that it can learn the optimal strategy with different settings. On the other hand, the Fixed and Heuristic method fails to adapt to the new settings. Those two methods are not able to maintain the dirt at a reasonably low level as the spawning rate increases.

Following the findings of the Simplified Environment, we apply the same algorithm in the 1-Dimensional Environment. We saw that the U-Model still performs best compared to the other methods, showing that the learning and allocation of agents is key to the optimal solution of the cleanup problem. With the movements, the allocation strategy becomes relatively more gentle (Figure 5). Agents do not switch roles that often as there is a cost. For example, an agent that has been moving towards an apple (and is close to an apple) might not be suitable to be shifted to clean waste. In a more complex environment like a 2-D environment, where switching roles takes even more steps, it is necessary to consider the cost of switching.

In terms of the extensions to the project, there are two main focuses. The first extension is to apply the U-Model on a more complex world. With a 2-D environment, the connection between action and result is much less apparent (to the model). Possibly new information should be given to the model to correctly estimate the effect of an action. This may require a larger model and more analysis in terms of which piece of information to provide. The second focus will be extending the influencer (powered by the U-Model) so that it provides per-agent suggestions as to what role they should take on and where they should move towards. This suggestion can be non-mandatory as agents themselves can have individual learning algorithms (e.g. Q-learning) to determine where they should go. They can choose to incorporate the suggestion of the influencer into their decision-making process and decide how much they would rely on this information through learning.

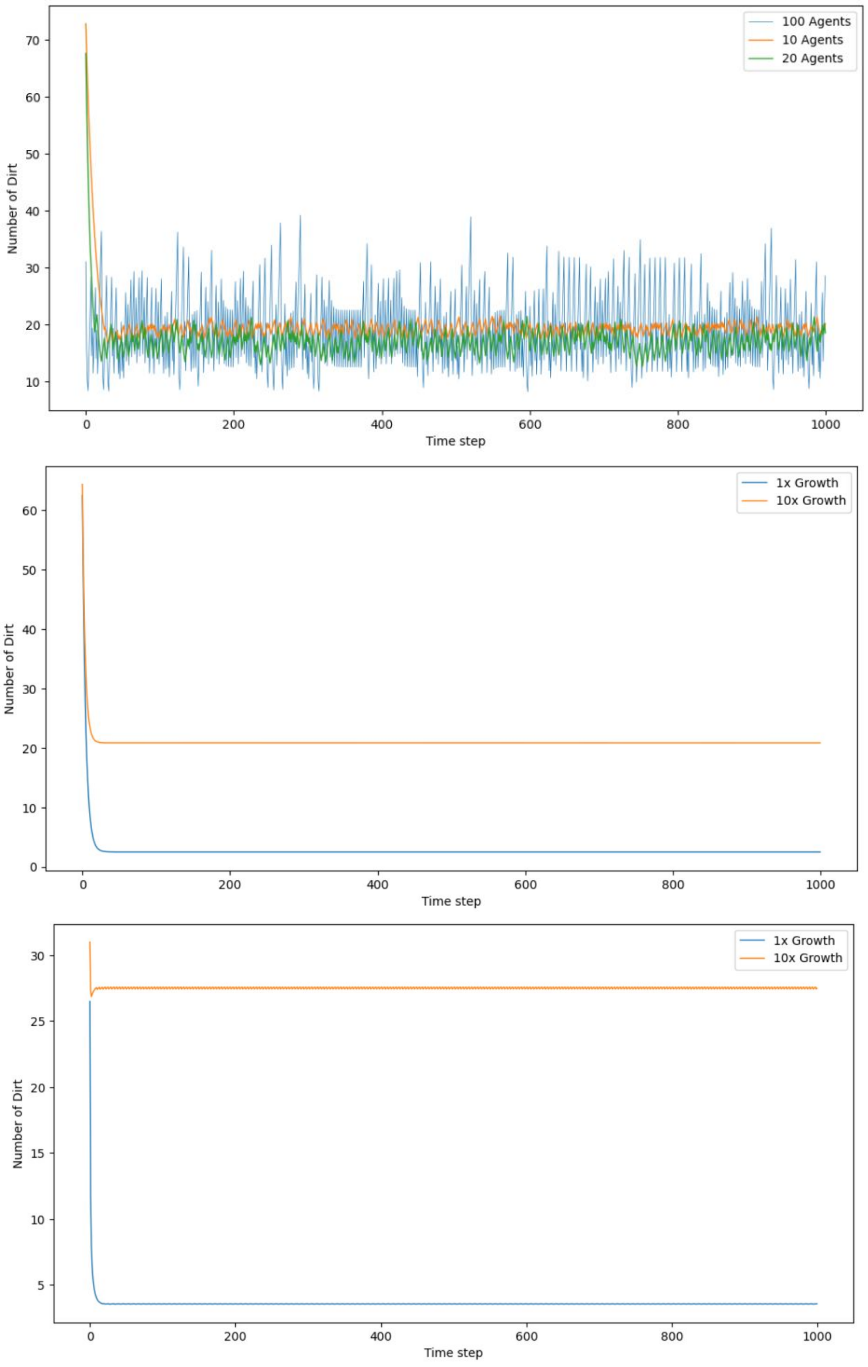

Figure 4: Dirt Number for U-Model, Fixed, and Heuristic policies with spawning rate is increased proportionally to the number of agents

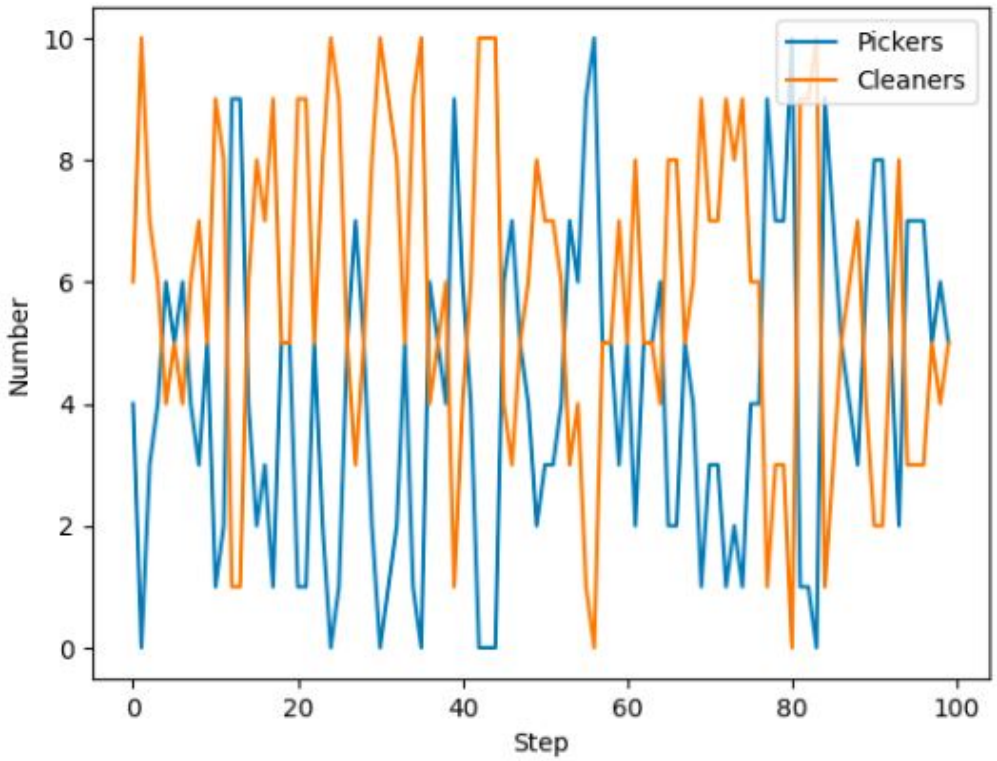

Figure 5: Number of Pickers and Cleaners with 1-D Environment, U-Model

8 Conclusion

To conclude, the project analyzed the importance of role assignment in a complex environment where the optimal answer is not straightforward. An "influcencer" that is specifically responsible for making decision on the role allocation task can direct the agents meaningfully and proved to outperform the heuristic methods. It divides the problem into simpler sub-problems so that learning models can handle them more easily. It also analyzed the potential in a decentralized model for learning - a model that can match/surpass performance of a centralized learning algorithm. With the influencer drastically cuts down the complexity of each agent's decision-making, plus the decentralized learning for each agent within their role/objective, the two components together could be a promising way of tackling such multi-agent sequential social dilemma problem.

9 References

[1] Jaques, N., "Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning", *arXiv e-prints*, 2018. doi:10.48550/arXiv.1810.08647.

[2] Wang, T., Gupta, T., Mahajan, A., Peng, B., Whiteson, S., and Zhang, C., "RODE: Learning Roles to Decompose Multi-Agent Tasks", *arXiv e-prints*, 2020. doi:10.48550/arXiv.2010.01523.

[3] Y. Xia, J. Zhu, and L. Zhu, "Dynamic role discovery and assignment in multi-agent task decomposition," Complex &; Intelligent Systems, vol. 9, no. 6, pp. 6211–6222, 2023. doi:10.1007/s40747-023-01071-x

[4] Wang, T., Dong, H., Lesser, V., and Zhang, C., "ROMA: Multi-Agent Reinforcement Learning with Emergent Roles", *arXiv e-prints*, 2020. doi:10.48550/arXiv.2003.08039.

[5] Su, J., and Marbach, P., "The Role of Social Support and Influencers in Content Markets," Association for Computing Machinery, 2023.