SusNet: Using Reinforcement Learning to Play Among Us

John Henry Rudden, Dmytro Vremenko

https://github.com/jhrudden/Sus-Net

Abstract

This paper explores the application of reinforcement learning to train autonomous agents capable of playing the multiplayer game "Among Us." We formulate the problem as a multi-agent reinforcement learning task, where agents must learn policies for two distinct roles: Crew Members and Imposters. Crew Members aim to complete tasks and identify Imposters through a voting mechanism, while Imposters attempt to sabotage tasks and eliminate Crew Members covertly. We develop a simulated environment for training that captures the key dynamics of Among Us, including partial observability, conflicting objectives, and the need for memory and coordination. Our approach utilizes Deep Q-Networks with convolutional layers to process spatial features and recurrent layers to handle temporal dependencies. However, initial experiments training agents in the full simulated environment proved challenging. We instead adopt an incremental approach, beginning with a simplified 1-vs-1 predator-prey scenario and gradually introducing complexities like obstacles and additional agents. We investigate the impact of different featurizations, hyperparameters, and model architectures on agent performance. While agents successfully learned effective policies in the simplified setups, further work is required to scale to the complete Among Us simulation. Our findings highlight the inherent difficulties in multi-agent reinforcement learning and the importance of systematic complexity management.

Introduction

Multi-agent reinforcement learning (MARL) problems introduce unique complexities not typically encountered in single-agent scenarios. Unlike traditional environments where an agent's actions directly influence outcomes, MARL environments involve multiple agents whose actions create a dynamic and stochastic system. This stochasticity arises from the unpredictable interactions between agents who may have competing or cooperative objectives, adding layers of strategic depth to the learning process. Our project will explore these dynamics within the multiplayer game "Among Us," a context rich with opportunities for both cooperation and competition among agents.

"Among Us" is a game that cleverly combines social deduction with task completion. Players are secretly assigned one of two roles: Crew Members or Imposters. The Crew Members' objectives are to complete various tasks around the map and to deduce who the Imposters are, ultimately voting them out of the game. Conversely, Imposters aim to covertly sabotage these tasks and eliminate Crew Members without revealing their identity. The game ends when either all tasks are completed, Imposters are identified and ejected, or the number of Imposters are greater than or equal to the number of remaining Crew Members.

"Among Us" is hard because it combines many non-trivial issues: multiple agents operate with competing objectives, and although there is an overarching need for coordinated team activity, each agent must act based on individual perceptions of the game state. This is further complicated as agents have limited information about the true state of the environment and other player' intentions. Crew Members, in particular, need to develop a memory system to keep track of other agents' behaviors, enhancing their ability to make informed decisions about whom to trust.

The explorations of social deduction games in the context of RL are not entirely new. Previous related work, investigated an image based factorization approach whereby an actor-critic was trained for sequences of game snapshots available to each agent [1]. Other approaches include Centralized Training with Decentralized Execution (CTDE)[2], where agents are trained in a centralized manner with access to global information but execute their policies in a decentralized fashion during deployment. For simplicity, our initial work focuses on a purely decentralized approach, although exploring CTDE is a potential future direction. We favored decentralized training for its simplicity.

Problem Statement

The objective of this project is to employ Deep Q-Learning to train autonomous agents capable of playing "Among Us." This requires the creation of a simulated environment that is consistent in dynamics compared to the real game. Within this simulated framework, we aim to develop and optimize distinct behavioral policies for two types of agents:

1. **Crew Members:** These agents are tasked with efficiently navigating the environment to complete a series randomly spawned tasks. Concurrently, they must participate in voting sessions to identify and eliminate suspected Imposters based on observed behaviors.

2. **Imposters:** In contrast, Imposter agents are designed to simulate deceptive behaviors by sabotaging tasks and eliminating Crew Members without revealing their identity. The primary computational challenge here is to develop a strategy that optimally balances the act of sabotage and the risk of detection.

Methods

Environment Setup

Given the complexity of fully replicating "Among Us," we opted to simplify the simulation environment to make our reinforcement learning approach more feasible. The environment was constructed as a 2-dimensional, 9x9 grid, reminiscent of a simplified four-room environment. This grid was divided into four symmetrical quadrants, each interconnected by a single passage cell that allows movement between adjacent rooms. Within this grid, a specified number of crew members and imposters are randomly placed at the beginning of the simulation. The order of crew members and imposters within the internal game state is randomized on reset which ensures that crew do not simply learn to vote for agent number 0, which is always the imposter. Tasks are designated as 'Incomplete' at the start, are also randomly distributed across the grid.

To simulate the critical voting mechanism from "Among Us"—the primary method for crew members to eliminate imposters—we implemented a continuous voting system that activates every 50 time steps. At the end of this period, any agent accumulating a majority of votes is eliminated from the simulation. However, each agent may only cast a vote once and there is no mechanism for taking votes back. This process ensures that gameplay does not pause for voting, allowing agents to vote alongside their normal actions within the game environment. This voting system is an obvious deviation from the voting system present in the real "Among Us" game, where game state is paused during a voting stage where the only possible action is to vote or abstain. However, handling multiple game states (voting vs moving) with different action spaces further complicates dynamics, so we chose to combine them.

This simulation is an episodic environment in which episodes are terminated in one of 4 scenarios:

1. Number of time steps reaches 1,000

2. All imposters have been voted out

3. The number of living crew members is less than or equal to the number of living imposters

4. All of the tasks are complete

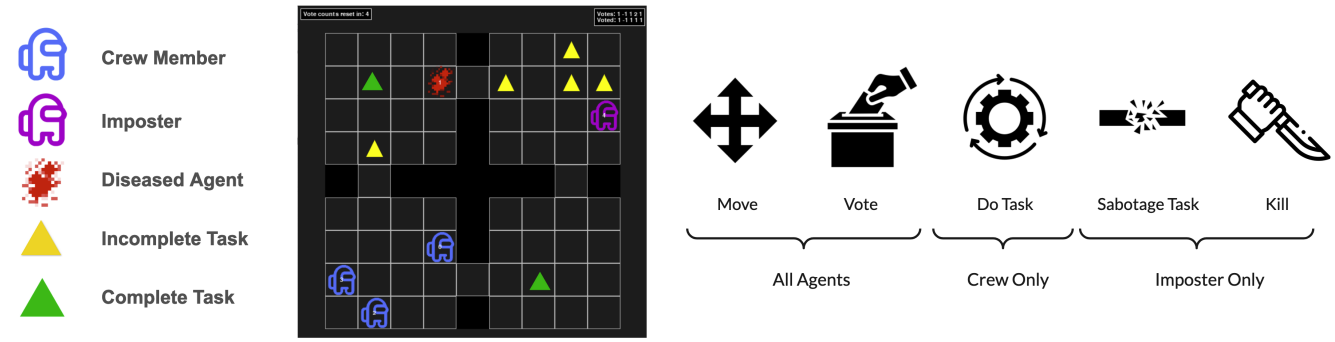

Figure 1: Environment and Action Space

Action Space and Environment Dynamics

The majority of the action space is shared between the imposters and crew members. Each agent can select from a pool of shared actions that can be used at each and every time step: remain stationary, move up, move down, move left, move right, or vote against another agent. Crew members have an additional action available— do task —which they can use when they are at the same location as a task to change its status from incomplete to complete. Conversely, imposters are equipped with two unique actions: sabotage, which allows them to revert a completed task to incomplete if they are at the task's location, and kill, which they can use to eliminate a crew member when in the same position. At each time step, the environment requires an action from each agent. The actions of all agents are executed in a randomly determined order each time step to maintain fairness and prevent any single agent from consistently benefiting from acting first.

Reward Structure

The reward structure in our simulation is designed to motivate both individual and team-oriented behaviors, reflecting the dual objectives present in "Among Us." Rewards are allocated based on actions that benefit either the individual's objectives or the team's overall goals.

Individual Rewards:

- **Killing:** Imposters receive a +3 reward for successfully killing a crew member, to incentivize aggressive play.
- **Task Completion:** Crew members gain a +1 reward for each task they complete, promoting task-focused gameplay.
- **Task Sabotage:** Imposters also earn a +1 reward for each successful sabotage, encouraging disruption of the crew's objectives.

Team Rewards:

- **Incorrect Voting Consequences:** If a crew member is incorrectly voted out, all remaining crew members receive a -3 penalty, while all surviving imposters receive a +3 reward. This highlights the consequences of misjudgments within team dynamics.
- **Correct Voting Benefits:** Successfully voting out an imposter yields a +3 reward for each remaining crew member and a -3 penalty for the remaining imposters, rewarding accurate judgment and effective communication among crew members.
- **End of Game:** At the conclusion of the game, the winning team receives a +100 reward, and the losing team receives a -100 penalty.

Passive Penalties:

- **Inactive Penalty:** Agents that are eliminated continue to receive a -3 penalty every time step, underscoring the ongoing impact of their absence from active gameplay.

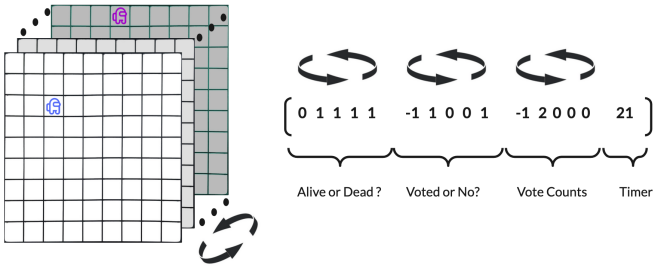

Figure 2: Perspective Feature Representation

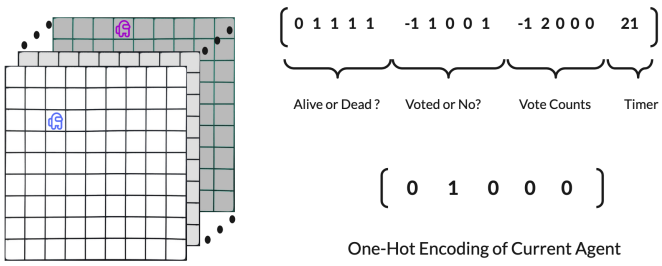

Figure 3: Global Feature Representation

Feature Design

In the design of our simulation, we segregated spatial from non-spatial aspects of the game environment, reflecting their distinct roles in the decision-making process of the agents.

Spatial Features:

- **Agent Positions:** Each agent's location is encoded within a one-hot 9x9 matrix, based on agent (x,y) coordinates. One matrix is created per agent.
- **Task Status:** To track the progress and location of tasks, we utilize two additional 9x9 channels. One channel represents completed tasks, and another represents incomplete tasks.

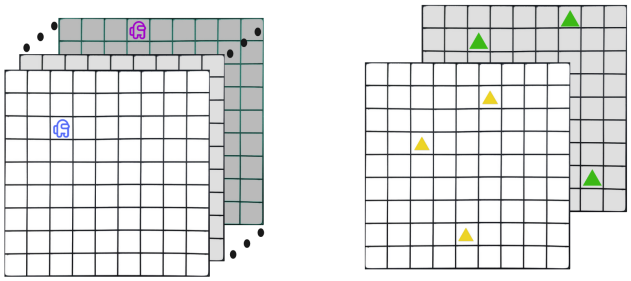

Figure 4: Spatial Feature Representations

Non-Spatial Features:

Non-spatial features are represented as one-dimensional lists, encompassing critical game elements such as voting results, the survival status of agents, current vote counts, and the remaining time until the next voting event.

Temporal Dynamics and Feature Input Regimes

Due to the temporal nature of the game and the need for a memory component, we collected consecutive sequences of states (length of each sequence was a hyperparameter), which were fed into Q-network. Two feature input regimes were tested:

- **Perspective View:** The agent position channels and the agent-specific non-spatial features are rotated to ensure that the agent for which action-values are being estimated always appear first in the order. This was hypothesized to help the network to develop an intrinsic understanding of which agent it is acting for (Figure 2).
- **Global View:** The spatial and non-spatial features do not change in their ordering. Instead, a one-hot-encoded agent index feature is appended to the non-spatial features (Figure 3).

Model Architecture

The Q-network was designed to handle the game's spatial and non-spatial features along with their temporal relationships. We use a four-layer Convolutional Neural Network (CNN) to process spatial features and extract latent information by compressing channel dimensions, but creating complex feature representations. These processed spatial features are flattened and combined with non-spatial features, capturing each time step's comprehensive game state. The combined features are then fed into a three-layer Recurrent Neural Network (RNN) with 64 units each, allowing the model to consider past game dynamics when making decisions. The output from the RNN passes through three fully connected layers of a Multi-Layered-Perceptron (MLP), which produces the final action-value estimates for the agents.

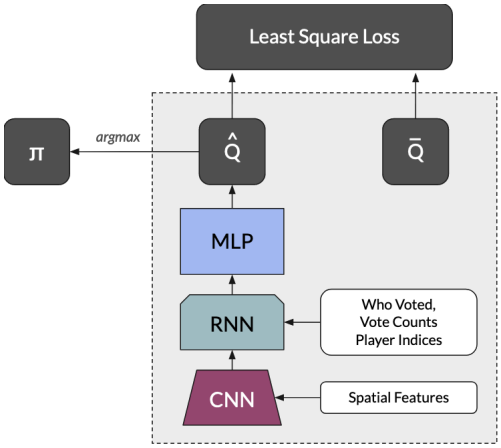

Figure 5: Hybrid Q-Network

Among Us Experiments

We started by fixing the crew members to random, equiprobable behavior, while attempting to train an effective imposter. The imposter DQN was trained over a total of 1.5M time steps, with a replay buffer of size 300K, pre-populated with 50K observations. The epsilon was exponentially annealed from 1 to 0.05 over the course of 1M time steps. The model was trained on batches of 8 trajectories (each with sequence size 1, 3, 6 for different experiments). The target model was updated after every 10K time steps. Adam optimizer was used with a learning rates of 0.1, 0.01, and 0.001. Discount factors of 0.9 and 0.99 were tested.

Among Us Experiment Results

A significant oversight was underestimating the complexity inherent in blending these dynamics within a multi-agent framework. We optimistically assumed that our environment setup and featurization strategy would eventually lead to effective learning. However, without extensive experience in handling each of these subproblems in a multi-agent context, our experimental results fell short of expectations.

These agents consistently exhibited unexpected behavior—migrating towards the right corner of the board—and failing to engage in more sophisticated strategies expected in gameplay. While the source of this behavior is not fully understood, we believe this may be an artifact of our rewards structure and equiprobable random crew policy. Since the crew is acting randomly, there is a high probability of a crew member being voted out at the end of the voting period, which would provide the imposter with a positive reward. Given that our maximum sequence length was 6 time steps, significantly less a single voting period (50 time steps), it is likely that the relationship between voting and associated rewards was not learned. The imposter policy could be stuck in a local minima, where low engagement with the game leads to eventual positive reward when crew is randomly voted out.

Taking a Step Back: Predator-Prey Framework

The failures of initial attempts to model complicated game dynamics including voting, killing, task completion/sabotaging, led to a drastic simplification of the problem. The environment was stripped of walls, tasks, and a plethora of crew members. The environment was reduced to a single imposter agent and a single crew member agent. The sole objective of the imposter is to navigate to the randomly moving crew member and kill them, closely resembling a predator-prey framework [3]. The only reward given to the imposter was +3 after a successful kill action. The Q-network was simplified down to an MLP which operated on the features of a single time step. This MLP consisted of 4 hidden layers (256, 128, 64, 16) and ReLU activation. Successfully training an imposter to complete this very simple task would be followed by more difficult iterations of the problem: addition of walls and other agents.

One Imposter vs One Crew Member

Having some experience of the stationary Four Room Env, we decided to test the simplest features first. We landed on simply encoding the agent positions with two different methods: using raw (x,y) coordinates (2 values per agent) and one-hot encoding of the coordinates (18 values per agent). The one-hot position encodings were fully zeroed out for non-living agents (this doesn't affect the 1v1 setting, but was important for more complex extensions). We also experimented with three discount factors—0.99, 0.9,

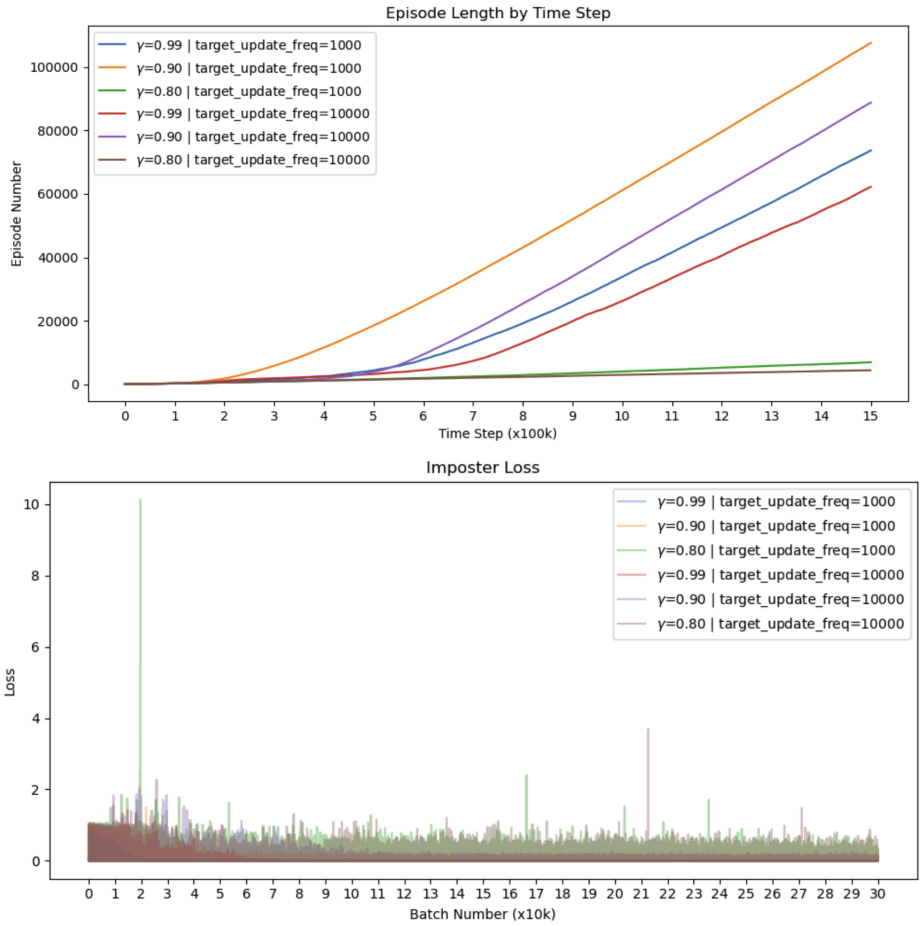

Figure 6: Affects of Hyperparameters on 1v1 (one-hot encodings)

and 0.8. Learning rate was fixed to 0.001 and the rate at which we updated our target network was tested for two values—every 10,000 and 1,000 time steps. Other hyper-parameters remained consistent our "Among Us" experiments. Two settings were tested: A simplified environment without walls to focus on basic agent interactions and a walled environment to see if imposter could overcome obstacles to hunt the single crew member.

1v1 Results

Our initial experiments in this simplified 1-vs-1 setting highlighted the sensitivity of agent behavior to hyperparameter choices, particularly the discount factor (gamma). The discount factor determines the extent to which the agent considers future rewards. With a high discount factor of 0.99, the imposter learned an undesirable strategy of remaining stationary and spamming the kill action, relying on the randomly moving crew member to eventually wander into its position. This behavior was likely due to the crew member's equiprobable random policy, which provided a high probability of eventually receiving a positive reward for a successful kill without actively pursuing the crew member.

Reducing the discount factor to 0.9 encouraged the imposter to actively seek out the crew member, demonstrating the impact of this hyperparameter on shaping agent behavior. However, further diminishing the discount factor to 0.8 led to a resurgence of the undesirable corner-seeking behavior observed in our initial, more complex environment. These results underscored the challenges of learning even the most simple tasks in a multi-agent reinforcement learning environment. This highlights the need for careful hyperparameter tuning and a gradual approach to introducing complexity.

Similar to our discount factor, the rate at which we updated our target network appears to have a significant impact to our learned policy. More frequent updates (every 1k time steps) seemed to create less variance in our loss (as expected), but also increased the rate at which our imposter was able to find and kill the single crew member.

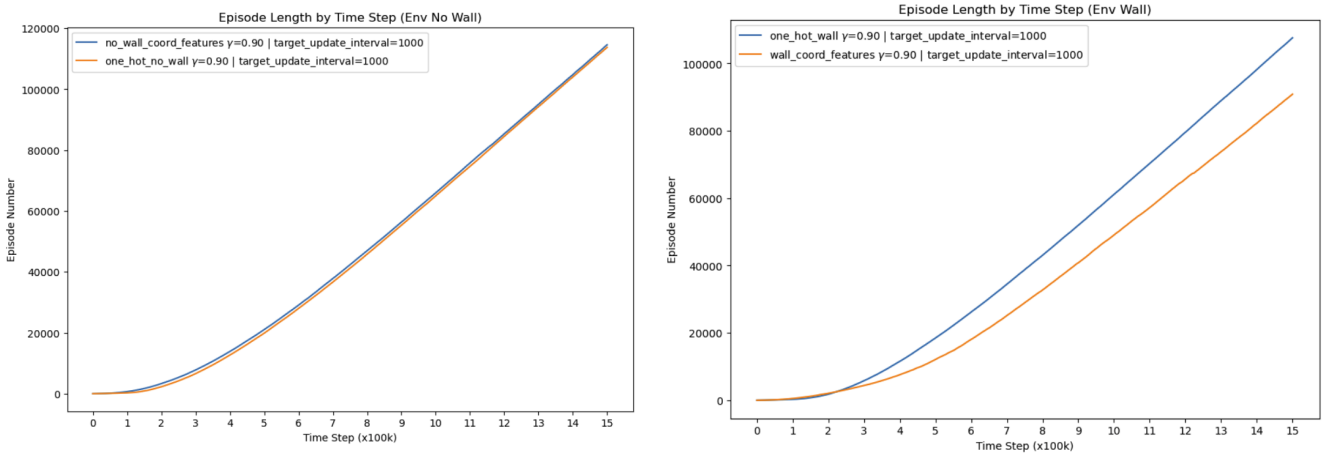

Figure 7: Affect of One-hot vs. Coordinate Encodings on # of Episodes (1v1)

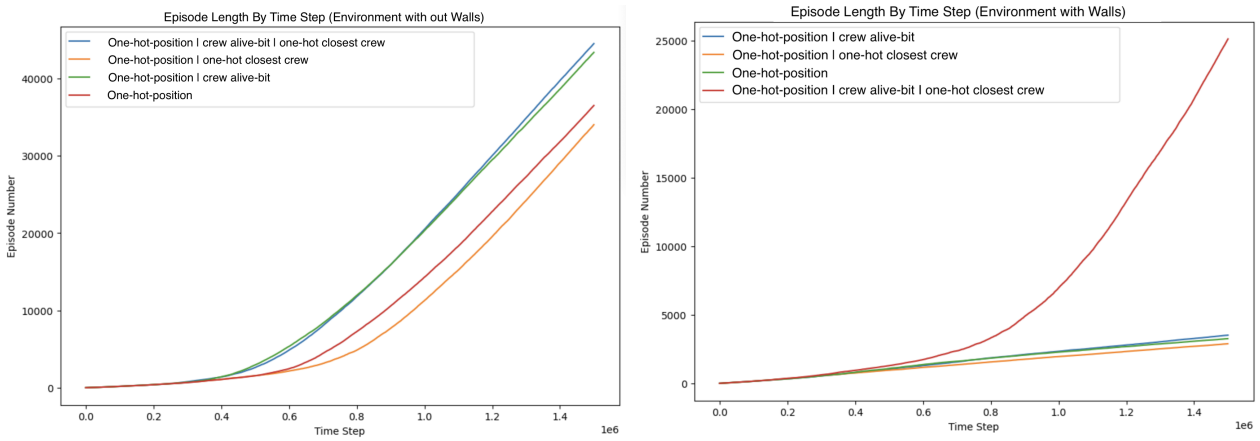

Figure 8: Affect of Various Features on # of Episodes (1v2)

One Imposter vs Two Crew Members

Following the outcomes of our 1v1 experiments, we introduced a more complex scenario with an additional crew member. This change increased the complexity for the imposter, who now has to differentiate between living and non-living crew members and decide which to target first. To aid in this, we implemented new features: one-hot encoding to identify the closest crew member (2 values total) and an alive/dead status bit for each crew member (1 value per crew member). We fixed the discount factor at 0.9 and updated the target model every 1,000 time steps (hyperparameters informed by the 1-vs-1 experiments). The experiments were conducted in environments with and without walls.

1v2 Results

Figure 7 shows the training results of various feature combinations on environments with and without walls. It is apparent that the one-hot position encoding serve as a sufficient baseline for learning to target and kill 2 crew members in an environment that lacks obstacles. Some additional benefit to training speed was provided by the inclusion of alive-bit features for each of the crew members, while adding a one-hot encoding of the closest crew member was not as fruitful. Transitioning to the significantly more difficult environment with walls, the inclusion of both additional features showed to be incredibly beneficial to the learning process. Adding either crew alive-bit or one-hot closest encoding did not perform better compared to only having one-hot position features. These results show the need for careful and informed feature selection as the complexity of the problem increases.

Discussion

We initially set out to train two distinct policies for playing a simulated version of "Among Us," one for imposters and one for crew members. Our environment included multiple agents, obstacles, tasks, and a voting mechanism. We hypothesized that a deep Q-network (DQN), using CNNs for spatial features and RNNs for temporal features, would efficiently process perceived game state and allow agents to learn to act intelligently. However, after running initial experiments, it became clear that the problem was more complex and nuanced than we anticipated. Despite conducting a hyperparameter search over learning rates, sequence sizes, and discount factors, the imposter's actions were consistently limited to moving right. With a stochastic transition function, a mix of team-based and individual rewards, heavily engineered features, an overly-complicated DQN model, and numerous hyperparameters, debugging our model's shortcomings proved to be nearly impossible. The sensible forward direction was to simplify our environment, rewards, model, and objective.

Subsequent experiments proved to be more manageable. We began with a simplified 1-vs-1 predator-prey setup, removing tasks, team rewards, and obstacles. We incrementally introduced complexity by adding obstacles and more crew members. Initially, in the 1-vs-1 scenario with and without walls, basic feature sets like coordinate or one-hot positions were sufficient for training. However, the addition of another crew member complicated this process. The basic features were inadequate for teaching the imposter to prioritize one crew member over another. Typically, using one-hot-encoded positions for all agents (zeroed out for deceased ones) led the imposter to navigate to the grid's center and passively wait for crew members, rather than actively pursuing them. Introducing alive-bit encoding for crew members and one-hot encoding of the nearest crew member significantly enhanced performance, particularly with obstacles present. Well-informed feature selection proved to be a critical attribute of both 1v1 and 1v2 experimental successes.

This phased approach underscores the need for incremental complexity in applying reinforcement learning to multi-agent problems. Starting with highly convoluted environments can lead to suboptimal policies trapped in local minima. An incremental strategy helps develop an intuition for the problem at hand and make them more approachable. Performance in simpler scenarios provides valuable insights for adjustments as complexity increases. Future work should focus on conducting multiple trials of proposed experiments to derive statistically significant conclusions, expanding the search for optimal hyperparameters to include model architecture, training duration, learning rate, optimizer, etc., and gradually escalating the environment's complexity towards the initially proposed "Among Us" simulation. Immediate next steps should involve training crew members to avoid being killed by the imposter. Lastly various extensions of DQN [4] should be explored for improved performance in addition to alternative algorithms such as CTDE [5], which were specifically developed for multi-agent applications.

References

[1] Kavya Kopparapu et al. *Hidden Agenda: a Social Deduction Game with Diverse Learned Equilibria*. 2022. arXiv: 2201.01816 [cs.AI].

[2] Vladimir Egorov and Aleksei Shpilman. *Scalable Multi-Agent Model-Based Reinforcement Learning*. 2022. arXiv: 2205.15023 [cs.LG].

[3] Michael Kölle et al. *Aquarium: A Comprehensive Framework for Exploring Predator-Prey Dynamics through Multi-Agent Reinforcement Learning Algorithms*. 2024. arXiv: 2401.07056 [cs.AI].

[4] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. arXiv: 1710.02298 [cs.AI].

[5] M. Saifullah et al. *Multi-agent deep reinforcement learning with centralized training and decentralized execution for transportation infrastructure management*. 2024. arXiv: 2401.12455 [cs.MA].