Robotic Instruction for Multiple Agents via Natural Language

Isaac Peterson
isaac.peterson@usu.edu

Kaiden McMillen
mcmillenka@gmail.com

Braxton Geary
braxton.geary@usu.edu

Abstract

In this project, we explore the application of reinforcement learning (RL) to train agents capable of navigating a 2D grid environment and identifying geometric shapes with distinct colors. The agent receives instruction via natural language and then the agents move one space at a time across the grid, using RL techniques to learn an optimal policy for efficiently locating and identifying shapes such as green triangles and red squares. We define the task as a partially observable Markov decision process (POMDP), where the agent's observations are limited to the grid space it occupies. Our approach involves implementing Reinforcement Learning to teach the agents to maximize reward by minimizing the time and steps required to find and correctly identify a shape. The results demonstrate how reinforcement learning can be applied to shape recognition and navigation tasks, with potential applications in robotics, search-and-rescue missions, and autonomous systems. We hope to explore the challenges that come from the multi-agent natural language task completion problems and present viable solutions.

1 Introduction

As technologies continue to advance, the integration of robots into every day life is continuing to increase. As humans rely mostly on speech for communication and instruction, it is essential that robots are also developed to understand and decipher language, executing commands effectively and efficiently.

There are a myriad of challenges that one confronts when attempting to solve this problem. First and foremost, establishing an interface between the spoken command and correct execution of that command. To properly execute the spoken command, the agent needs to have an accurate understanding of its environment, which in the real world can become extremely complex, and the connection between the environment and the spoken language. Initial attempts were made by utilizing more logic based methods to help the robot understand the task that needs to be solved (Liu, 2016). To mitigate the challenges that come with real world interpretation, many researchers defaulted to simulations to instruct agents in a more structured world (Wang et al., 2024).

Video games form a natural challenge for agents, with clear tasks to complete in a very structured world. (Chaplot et al., 2017) used the environment in the video game DOOM™ to train their agent to follow natural language instructions, with other researchers using similar techniques in Minecraft™ (Tessler et al.), and even developing their own virtual environments for natural language task completion (Anderson et al., 2017) (Wang et al., 2024).

Our goal is to use the simplified environment in multigrid (Oguntola et al., 2023) to explore the process of natural language task completion, with more research on the multi-agent natural language task completion problem. Where instead of instructing a single agent to complete a task, we instruct a fleet of agents and explore the challenges that come with the increased number of agents along solutions to address these challenges.

2 Related Work

Most of the work in this project will be based on the work done by (Chaplot et al., 2017). They utilize a combination of large language models and reinforcement learning to help an agent understand specific instructions to navigate in the game environment of doom. Others utilized the world of minecraft to help agents complete tasks, with less of a focus on instruction and more on generalized learning rather than interpreting language (Oh et al., 2017), (Tessler et al.). Combining the implementation of (Chaplot et al., 2017) with the multigrid en-

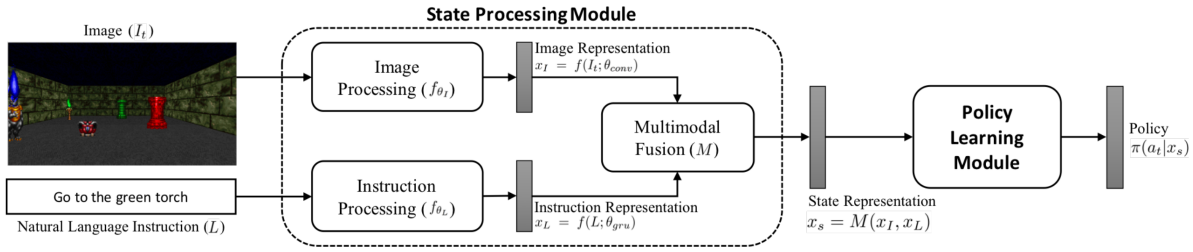

Figure 1: State processing method as developed by (Chaplot et al., 2017).

vironment (Chevalier-Boisvert et al., 2018) we can further explore the challenging problem of robotic instruction with large language models, with the added challenge of addressing multiple agents.

Additionally, the challenges in multi-agent systems, such as non-smooth environment dynamics and effective collaboration, are central to this exploration. (Xu et al., 2023) propose Distributed Targeted Multi-Agent Communication (DTMAC), a fully distributed multi-agent reinforcement learning method that enables agents to generate and exchange messages, thereby enhancing their collaborative strategies. Our approach uses more of a centralized model that can take as input the instruction and the state of all the agents and output the necessary actions for the agents.

3 Experiments

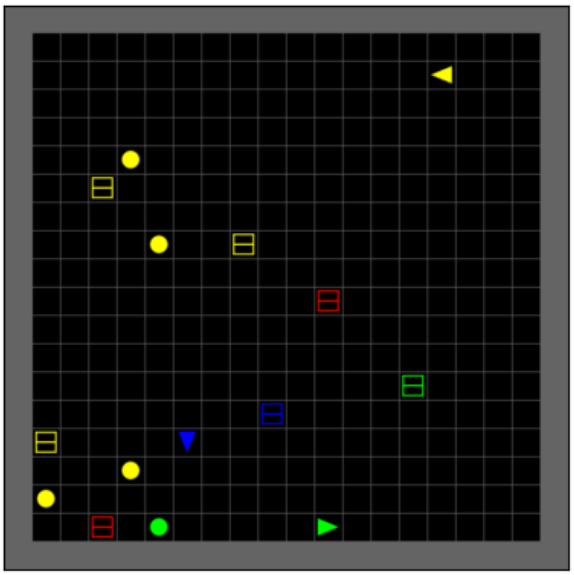

Figure 2: Example of a multigrid level.

As mentioned previously we will be using the multigrid environment to accomplish this task. We will customize the environment in order to be able to accept natural language instructions, tokenize those instructions, and feed the instructions into the RL model as the state space to update the action probabilities.

4 Teamwork

The design of the framework as shown in Figure 3 makes it easy to delegate the tasks into three separate stages for the project.

4.1 Environment Setup

Kaiden McMillen was responsible for setting up the environment for training a reinforcement learning model capable of handling multiple agents simultaneously. The final environment was built using `gym-multigrid`, an extension of `gym-multigrid`, tailored to support multi-agent interactions. The environment is designed to provide a robust testbed for agents to learn coordination and decision-making tasks. Below, we describe the key components of the environment.

Overview The environment spans a 20x20 grid and supports the simultaneous training of multiple agents (default: 3). It includes dynamic object placement and a custom reward structure to encourage strategic behavior. Agents aim to locate, identify, and interact with specific objects (balls and boxes) based on their colors and shapes.

Grid Setup The grid is surrounded by walls to confine the agents and ensure structured navigation. Objects are randomly placed in the grid to introduce variability during training. Specifically, 10 balls and 10 boxes of various colors (red, blue, green, and yellow) are randomly distributed. Agents are also initialized at random positions within the grid.

Agents Each agent has a 7x7 field of view and can perform actions such as moving, picking up objects, and interacting with the environment. The

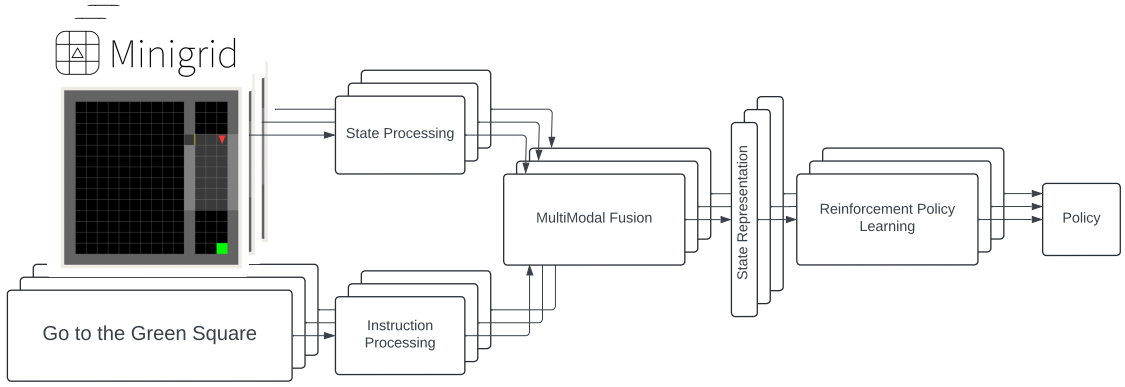

Figure 3: Our approach to solving the multi-agent natural language task-solving problem, showing layers for each agent.

reward structure gives incentive to agents to explore, identify, and collaborate effectively. Custom observations are processed into a flattened format suitable for use with vectorized environments, with observation shapes of $(7, 7, 12)$.

Inspiration from Existing Games The structure of this environment was partly inspired by the `collect_game` environment in `gym-multigrid`. Similar to `collect_game`, our environment involves agents interacting with objects in a shared grid world, focusing on identifying and gathering specific items. Additionally, both environments emphasize multi-agent cooperation and the challenges of competing or coordinating in a confined space. Unlike `soccer_game`, which centers around adversarial dynamics, our design encourages agents to collaborate to achieve shared objectives. This balance of inspiration allows for a unique environment tailored to exploration, coordination, and efficient decision-making.

Reward Function Rewards are structured to encourage correct identification and collection of objects:

- Agents receive 1000 points for identifying and collecting the correct object.
- Partial rewards of 100 points are provided for collecting an object that matches either the correct color or shape.
- Small rewards are given for movement and interaction, while penalties are applied for invalid actions.

Termination Condition The task completes when all target objects are identified and collected. The environment fosters coordination and efficiency among agents by emphasizing group task completion over isolated actions.

Technical Implementation The environment extends the `MultiGridEnv` class and introduces custom methods for grid generation, observation processing, and action handling:

- **Grid Initialization:** Objects and agents are placed randomly on the grid. Walls surround the environment to prevent agents from leaving.
- **Step Logic:** Agent actions, such as movement and object interaction, are processed. Rewards are assigned based on the correctness of these actions.
- **Task Completion:** The environment tracks whether all target objects are collected to determine when the task is complete.
- **Rendering:** The environment supports rendering for both console and graphical outputs.

Gymnasium Integration The environment is registered as a Gymnasium-compatible environment with the identifier `FindShape20x20-v0`. This ensures compatibility with reinforcement learning frameworks that follow the Gym API:

```
import gymnasium as gym

gym.envs.registration.register(
    id="FindShape20x20-v0",
   entry_point="find_shape:FindShape20x20Env",
)
```

Customization and Extensibility The environment can be customized to suit various tasks:

- Adjustments to the number of agents, objects, or grid size can be made.
- Reward functions can be modified for different objectives.
- The environment allows for testing agent behaviors under dynamic and randomized conditions.

This environment supports reinforcement learning research by enabling agents to develop sophisticated strategies in a multi-agent, multi-task context.

4.1.1 Multimodal Fusion

Isaac Peterson was responsible for setting up the state processing module. This includes handling the state information from the multigrid environment and the input instruction L, passing them through their appropriate models and creating the output vector X to be processed by the reinforcement learning model.

4.2 Data

For our project the goal was to get multiple agents to work coherently to accomplish a common task, collecting shapes with different colors. We wanted to be able to use natural language in order to tell the agents what their task was and have them interpret the instruction and execute it effectively. In order to do that, we had to create a data-set where each sample included an instruction, along with the description of the instruction so that we could assign appropriate reward.

```
{
  "instruction": "go to the red boxes",
  "targets": [
      {
          "shape": "box",
          "color": "red"
      }
  ]
}
```

Listing 1: An example of the training data for the RL model

We had 20 different instructions with different tasks. The colors could be either red, blue, green or yellow, and the shapes could be box or ball. For both shape and color, the option for "na" was available in the case we didn't care about the shape or color. For example, if we wanted the agents to find all the red objects, the shape would be "na" and the color would be "red".

```
{
  "instruction": "move to the yellow
      boxes and red balls",
  "targets": [
      {
          "shape": "box",
          "color": "yellow"
      },
      {
          "shape": "ball",
          "color": "red"
      }
  ]
}
```

Listing 2: An example of the training data for the RL model with multiple tasks

We also had training data where multiple tasks were involved. The instruction would be based through the multimodal fusion model along with the state of each agent, and reward would be allocated on the status of the tasks solved. This shows the flexibility of the environment for multi-agent coordination training with natural language.

4.2.1 Model Description

We propose a multimodal model that integrates natural language instructions with visual state representations using attention mechanisms and a transformer-based encoder. The model consists of two main components:

- **Simple Attention Module**: A feed-forward neural network with a multi-head attention layer.
- **State Processing Module**: Combines DistilBERT, a transformer-based language model, with state data for multimodal fusion.

4.3 Simple Attention Module

The Simple Attention module is defined as follows:

$$
\begin{aligned}
\mathbf{z} &= \text{MultiHeadAttention}(\mathbf{x}, \mathbf{x}, \mathbf{x}), \\
\mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1), \\
\mathbf{h}_2 &= \mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2, \\
\mathbf{y} &= \sigma\left(\frac{1}{n}\sum_{i=1}^{n} \mathbf{h}_2^{(i)}\right),
\end{aligned}
$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices, $\mathbf{b}_1$ and $\mathbf{b}_2$ are biases, σ is the sigmoid function, and n is the sequence length.

4.4 State Processing Module

The State Processing Module performs the following steps:

1. Tokenize the input natural language instruction using DistilBERT tokenizer.

$$\text{tokens} = \text{Tokenizer}(\text{instruction})$$

2. Encode the tokens using DistilBERT to obtain the last hidden state $\mathbf{H}$:

$$\mathbf{H} = \text{DistilBERT}(\text{tokens}).$$

3. Apply the Simple Attention module to the hidden states:

$$\mathbf{g} = \text{SimpleAttention}(\mathbf{H}),$$

where $\mathbf{g}$ is reshaped to a gating tensor of size $(\text{view\_size}, \text{view\_size}, 6 \times \text{num\_agents})$.

4. Multiply the gating tensor $\mathbf{g}$ element-wise with the state tensor $\mathbf{S}$ to produce the gated state representation:

$$\mathbf{S}_{\text{gated}} = \mathbf{S} \odot \mathbf{g}.$$

We use the attention mechanism along with some linear layers to transform the output from the BERT tokenizer into a shape that matches the number of actions that our agents can take. This way the model can learn which tokens are more correlated with actions that get higher rewards.

4.5 Multi-Agent Reinforcement Learning

Braxton Geary developed the Reinforcement Learning (RL) module for our agent using a Deep Q-Network (DQN) integrated with an Asynchronous Advantage Actor-Critic (A3C) algorithm (Mnih et al., 2016). This RL head plays a critical role in processing the gated state space, estimating action probabilities and value functions, and guiding the optimization of the policy.

4.5.1 Integration with the Gated State Space

The RL head receives a *gated state representation* from the `StateProcessingModule`, once the state and instructions have been preprocessed. This preprocessing step ensures that the RL head operates on an augmented and filtered state space emphasizing pertinent features for the given Natural Language task. The processing steps in the RL head are as follows:

1. **Flattening:** The multi-dimensional state tensor is flattened to prepare it for fully connected layers while maintaining temporal relations.

2. **Feature Extraction:** The flattened state is passed through a fully connected layer with ReLU activation.

3. **Temporal Modeling:** The transformed features are processed through an LSTM network, which uses the temporal dependencies to generate outputs reflecting immediate and historical state contexts. This element is critical to agent performance as this layer acts as the agent's memory, allowing it to remember the relative location of previously viewed objects.

4. **Output Projection:** The LSTM output is passed to a fully connected layer to predict two key outputs: a scalar value estimate for the state and action logits for policy optimization.

5. **Probability Formulation:** The output is then split as the value estimation is complete. We apply a mask to prevent no-op actions from contributing to the softmax calculation and ensure such actions have no possibility of being chosen.

4.5.2 Loss Calculation

The RL head employs two loss functions to optimize policy and value predictions:

1. **Critic Loss:** This loss minimizes the mean squared error between the predicted state value ($V(s)$) and the target value. The target value is computed as the immediate reward plus the discounted value of all subsequent non-terminal states:

$$L_{\text{critic}} = \text{MSE}\big(V(s_t), r_t + \gamma V(s_{t+1})\big) \quad (1)$$

where r_t is the reward received at time t, γ is the discount factor applied to future rewards, and $V(s_{t+1})$ is the predicted value of the next state. Minimizing this loss causes the Critic to progress toward correctly scoring all states based on immediate and expected rewards.

2. **Actor Loss:** The loss function uses the difference in received and expected reward as a

means to push the model toward actions resulting in large positive rewards.

$$L_{\text{actor}} = -\mathbb{E}\big[\log \pi(a|s) \cdot \big(Q(s,a) - V(s)\big)\big] \tag{2}$$

3. **Total Loss:** The total loss is a simple sum of the actor and critic losses:

$$L_{\text{total}} = L_{\text{actor}} + L_{\text{critic}} \tag{3}$$

4.5.3 Backpropagation and Gradient Propagation

The total loss is then used with standard backpropagation to calculate gradients w.r.t. model parameters. This is done by traversing the computational graph in the reverse beginning at the final layer and through the State Processing Module giving us end-to-end training. The optimizer (Adam) applies these gradient updates to all network parameters incrementally adjusting the policy and value predictions toward their optimal configurations while simultaneously optimizing how the state space is embedded and gated.

4.5.4 Importance of Gating and Temporal Dynamics

The integration of gated state representations ensures our agents focus on task-relevant features while ignoring extraneous data. Additionally, the temporal dynamics modeled by the LSTM enable the network to learn dependencies across sequences, capturing the effects of delayed rewards and complex strategies. Together, these mechanisms allow the RL head to serve as an effective decision-making engine, enabling agents to operate efficiently in dynamic, multi-agent environments.

5 Results

Unfortunately, we were unable to get actual result for our implementation. We ran out of memory on the GPU that we were using. However we still believe the code to be useful as an environment for testing multi-agent coordination with natural language. We also provide the action probability space after a single episode of 200 steps in Figure 4 and Figure 5. We can see that the probability of picking up objects increased, which is the kind of behavior that we want from agents completing a collect objects game. However, we did give a reward mentioned previously for moving, which does not seem to be reflected in the forward action which has a less probability of being selected after the first episode.

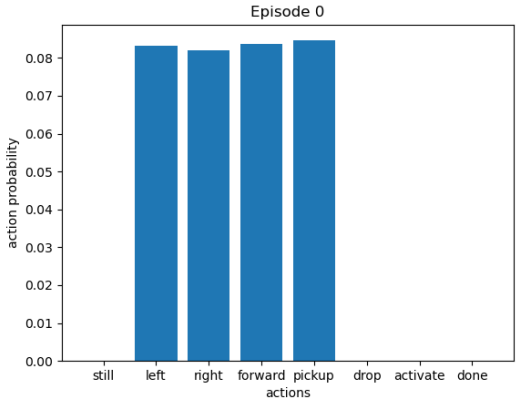

Figure 4: The average probability of the actions among the 3 agents. Initially all the actions are sampled with equivalent probability. We mask out the actions that we don't care for the agents to learn, hence only the four actions an agent can take are turn left, turn right, go forward, and pickup.

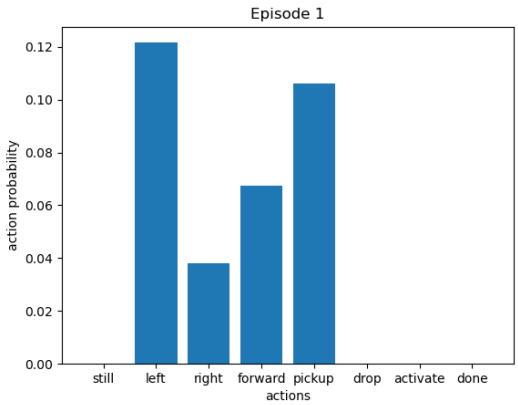

Figure 5: Probability of the action space after 1 episode of 200 steps. We get an increase probability for picking up objects. We speculate this is a result of the high reward given for picking up.

6 Conclusion

In this work, we presented an exploration of applying reinforcement learning (RL) to train agents for navigating a 2D grid environment while identifying geometric shapes through natural language instructions. The task was framed as a partially observable Markov decision process (POMDP), where agents make decisions based on limited observations of their current grid space. The goal was for agents to efficiently locate and identify shapes, such as green triangles and red squares, while minimizing the time and steps required to complete the task.

While we encountered a technical issue due to running out of GPU memory, limiting the execu-

Table 1: Hyper-parameters used during training. Each parameter and its corresponding value are provided, along with a brief description of its purpose.

Parameter	Value	Description
`./data/instructions.json`	–	Path to the instructions file
`-num_episodes`	20	Number of episodes for training
`-max_steps`	200	Maximum steps per episode
`-agent_view_size`	7	Size of the agent's view grid
`-num_agents`	3	Number of agents in the simulation
`-seed`	42	Random seed for reproducibility
`-train_every_n_iters`	1	Training frequency (in iterations)
`-prob_fig_path`	`./log_probs.png`	Path to save the probability figure

tion of our RL model, the results provide valuable insights. The action probability plots (Figure 4 and Figure 5) demonstrated the expected increase in the probability of actions like "pickup," which aligns with the goal of agents completing tasks such as a collect objects game. Interestingly, the forward movement action, despite being rewarded, showed a lower probability of selection after the first episode, suggesting that agents might prioritize object collection over movement, which might indicate a need for further tuning of the reward structure.

Although we were unable to achieve a fully functioning RL implementation due to resource and time constraints, we believe that the proposed environment holds significant potential for testing multi-agent coordination and natural language task completion. This study opens the door for future research in areas like robotic navigation, autonomous systems, and natural language-driven tasks, where reinforcement learning can provide a robust framework for developing intelligent, adaptable agents.

Our code, along with a README on how to set it up and execute is located here.

References

2016. *Natural-Language-Instructed Industrial Task Execution*, volume Volume 1B: 36th Computers and Information in Engineering Conference of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*.

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. 2017. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. 2017. Gated-attention architectures for task-oriented language grounding.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2018. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Ini Oguntola, Joseph Campbell, Simon Stepputtis, and Katia Sycara. 2023. Theory of mind as intrinsic motivation for multi-agent reinforcement learning. *arXiv preprint arXiv:2307.01158*.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. 2017. Zero-shot task generalization with multi-task deep reinforcement learning.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft.

Hanqing Wang, Jiahe Chen, Wensi Huang, Qingwei Ben, Tai Wang, Boyu Mi, Tao Huang, Siheng Zhao, Yilun Chen, Sizhe Yang, Peizhou Cao, Wenye Yu, Zichao Ye, Jialun Li, Junfeng Long, Zirui Wang, Huiling Wang, Ying Zhao, Zhongying Tu, Yu Qiao, Dahua Lin, and Jiangmiao Pang. 2024. Grutopia: Dream general robots in a city at scale.

Chi Xu, Hui Zhang, and Ya Zhang. 2023. Multi-agent reinforcement learning with distributed targeted multi-agent communication. In *2023 35th Chinese Control and Decision Conference (CCDC)*, pages 2915–2920.