Multi-Robot Warehouse Project Report

Adam Alvares 21008

December 2024

Introduction

For this MARL project, we simulate a warehouse with robots moving and delivering requested goods/shelves. Humans access these goods, and the robot then delivers them back to an empty spot. This project explores the application of deep reinforcement learning to optimize agent behaviors in a robotic warehouse setting. Using the "rware" simulation environment prepared by the University of Edinburgh, the objective was to train agents capable of efficiently completing tasks, such as moving goods to specified locations, while managing obstacles and constraints specific to the warehouse.

Objective

The goal of this project was for two agents to collaborate in moving requested shelves containing goods to goal positions and then returning the shelves to empty spots. The agents were required to minimize time and collisions in a 2D grid world while avoiding obstacles, which included non-requested shelves and the other agent.

Environment

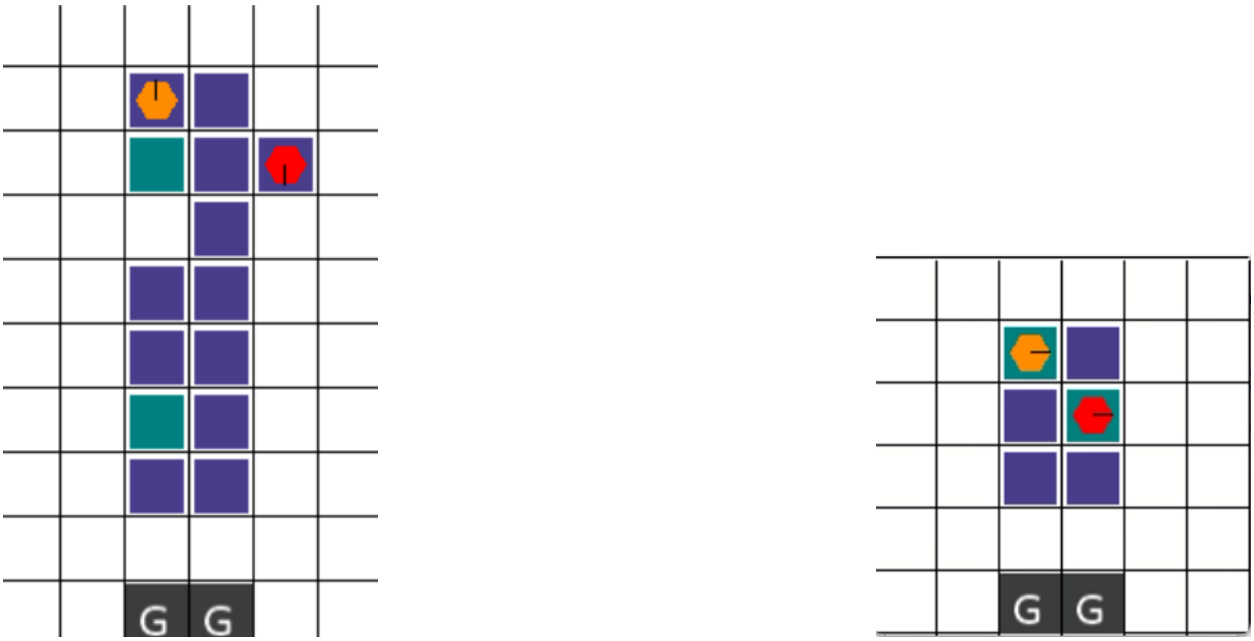
The environment for this project was developed using the `rware` simulation framework, which provides a configurable robotic warehouse setting. The layout consisted of a grid-based map with designated obstacles and goal zones. Agents were tasked with navigating the grid to transport goods while avoiding collisions and inefficiencies. The observation space provided each agent with a vectorized representation of its surroundings, while the action space allowed movements in four cardinal directions and a "load/unload" action for interacting with goods.

We trained and tested the DQN algorithm on two environment layouts: a small layout and a medium-sized layout. Refer to the figure below for visualization of the layouts. The hexagons are the agents. The green boxes are the requested shelves that need to be picked up. The blue boxes are the non requested shelves. If the agent is red then it has picked up the box. If it is yellow then it has not picked up anything. The "G" is the goal position.

Methodology

Environment Setup

- `warehouse.py`: Sets up the environment layout and defines observation and action spaces. It updates the `step()` of the agents and assigns rewards for each agent based on its actions.
- `rendering.py`: Uses the `Pyglet` library to provide a graphical representation of the warehouse for visualizing the environment.
- `__init__.py`: Configures the environment layout and the observation range of agents.



(a) Large layout. (b) Small layout.

Figure 1: Comparison of large and small layouts.

Deep Q-Learning Implementation

Network Architecture:

- Input: Observation vector containing the state of the environment (size: 71).
- Networks: Two networks are present, they are evaluation network and target network.
- Layers: Two fully connected layers with 256 neurons each and ReLU activation functions.
- Output: Q-values for all possible actions (5 actions).

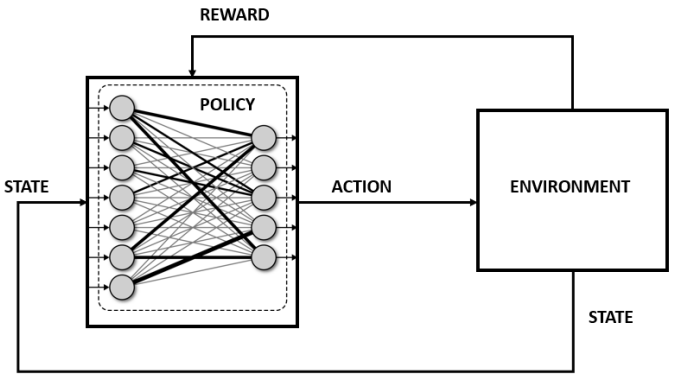


Figure 2: DQN Architecture with evaluation network

Adding the Target Network:

$$R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$$

This is the target q-value. It helps stabilize the learning process. MSE Loss is computed between the evaluation network and the target network.

Hyperparameters:

- Learning rate: 0.001
- Discount factor (γ): 0.95
- Batch size: 64
- Memory size: 100,000 transitions
- Exploration: Epsilon-greedy policy with decay from 1.0 to 0.01
- Target Network Update Frequency: Every 100 steps

Parameters Used:

- Action space: 0, 1, 2, 3, 4
- Observation space: 71 in length, providing information about surrounding grids.
- Number of agents (`n_agents`): 2
- Reward type: Reward shaping was done with the following structure:
  - -0.4 for any action that is not an intermediate state.
  - -0.1 for picking up a requested shelf.
  - 2.0 for delivering a requested shelf to the goal.
  - 3.0 for delivering the shelf back to the empty spot.

Program Setup

- `main.py`: Calls `dql.py` for training the agents.
- `dql.py`: Implements and trains the DQN network.
- `testlarge.py`: Tests the saved DQN models on the environment.

Training and Testing Process

The training setup included two agents, each using its own DQL policy. Training ran for 2,600 episodes, with a maximum of 800 steps per episode. At each step, agents selected actions independently based on their respective policy networks. The agents updated their networks using experiences stored in their replay buffers, iteratively improving their ability to handle the warehouse environment.

Evaluation and testing were conducted separately by loading the trained models into the environment. Over a series of test episodes, the performance of the agents was assessed based on their ability to achieve goals while avoiding penalties.

Results

Training Performance

During training, the performance of the agents improved steadily. Average scores across the last five episodes were tracked and used to measure progress. The learning curve showed that agents learned to navigate the warehouse efficiently, reducing penalties associated with invalid moves and increasing rewards for task completion. We can see that the avg reward for large layout decreases after a certain number of iterations. This is because as epsilon decreases the exploration decreases, and so the agent finds it difficult to find requested shelves that are far away.

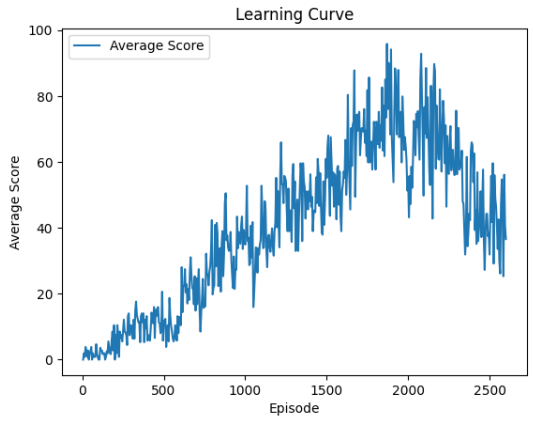


Figure 3: Avg reward for large layout.

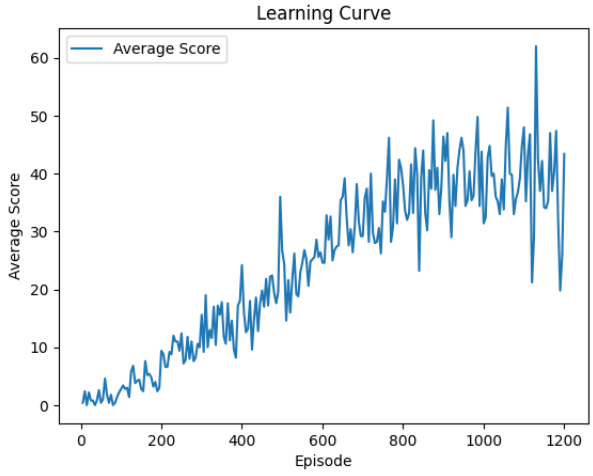


Figure 4: Avg reward for small layout.

Conclusion

This project demonstrated the effectiveness of Deep Q-Learning in multi-agent robotic warehouse environments. The agents successfully learned to collaborate, optimizing their movements and actions to fulfill task requirements while avoiding penalties. Future work could involve testing more complex layouts and integrating more advanced algorithms.