Design, Development, and Simulation of an Automated Material Replenishment System Enhanced by Reinforcement Learning in Industrial Production Lines and Warehouses*

Onur Ulusoy
Department of Control and Automation Engineering
Istanbul Technical University
Istanbul, TURKEY
ulusoyo18@itu.edu.tr

Ahmet Onat
Department of Control and Automation Engineering
Istanbul Technical University
Istanbul, TURKEY
ahmetonat@itu.edu.tr

***Abstract*—This study focuses on optimizing material replenishment in industrial production lines using black robots supported by Reinforcement Learning (RL). The system enhances Just-in-Time (JIT) production principles through AI and swarm intelligence algorithms, ensuring efficient and fast task execution. A 3-dimensional prismatic box grid system dynamically adapts to production demands, optimizing material distribution, reducing stock, and lowering costs. Simulation results demonstrate the effectiveness and accuracy of the AI system, providing a foundation for real-world applications.**

***Index Terms*—Reinforcement Learning, Material Replenishment, Industrial Production, AI, Swarm Intelligence, Just-in-Time, Robotics, Simulation**

I. Introduction

Enhancing the efficiency of material replenishment processes and reducing costs in industrial production lines is a critical need in today's competitive manufacturing environment. This study explores how these processes can be optimized using artificial intelligence and robotics technologies. The project aims to enable black robots equipped with Reinforcement Learning (RL) and swarm intelligence algorithms to perform material replenishment tasks in production lines most effectively. Detailed simulation studies were conducted, and the findings provide valuable data to guide future applications.

A. Objective

The primary objective of this study is to address the challenges faced in material distribution and prioritization processes in modern production lines and develop new solutions by optimizing these processes with artificial intelligence and robotics technologies.

In contemporary production processes, ensuring that materials are delivered to the right place at the right time is crucial, especially due to irregularities in material flow and demand variability. These challenges can impact the efficiency and continuity of production lines, increase costs, and lead to waste. Traditional methods, which mostly rely on manual labor or simple automation systems, struggle to adapt quickly to changes in demand and do not fully align with Just-in-Time (JIT) production principles. This misalignment can negatively affect the continuity and flexibility of production processes.

The proposed solution in this study uses an artificial intelligence system trained with Reinforcement Learning (RL) to optimize material distribution in production lines and efficiently meet material needs within the learning process. The AI-supported system can dynamically adapt to changes in production demands and continuously improve material distribution processes. The designed swarm of robots will manage material flow seamlessly and efficiently, ensuring high precision and efficiency in production areas.

The proposed system fully aligns with JIT production principles. More effective management of material distribution processes helps reduce unnecessary stock, thereby lowering costs. Additionally, it supports the uninterrupted and efficient execution of production processes, contributing to overall production efficiency. This approach offers an innovative and comprehensive solution that overcomes the current challenges in material distribution and prioritization processes, ensuring more effective and continuous operation of production lines.

B. Project Overview

1) Approach: To address the storage challenges encountered in material management and production processes, a 3-dimensional prismatic box grid system capable of containing raw materials or semi-finished products has been designed. Traditional warehouse systems often fail to adapt to rapidly changing production demands due to dynamic arrangement challenges, the need for large spaces, and inflexible structures. The designed new system overcomes these limitations by optimizing space utilization and dynamically adapting to production needs. Material boxes are efficiently managed by an AI-supported swarm of robots moving on a plane above the grid system. Using the methods and designed algorithms, robots capable of specific actions can place required material boxes in strategic positions on the top plane of the grid

or store unnecessary items in desired locations in the lower compartments quickly and effectively.

2) Method: This project progresses cumulatively through different stages. In the first main stage, the design of artificial intelligence algorithms is realized in usable simulation environments using programming languages. This process is critical for testing and improving the effectiveness of the algorithms. The second main stage involves applying the developed artificial intelligence algorithms to real robots. Within this scope, it is planned to design and integrate robots using different simulation platforms. The transition between these two stages will be carefully managed to ensure smooth transfer of algorithms from simulation to real-world applications. The segmented structure of the project allows for the effective optimization of both theoretical and practical applications.

II. Problem Stages

This project consists of various stages meticulously planned with the advisor, leveraging expertise and experience in the field. Each stage provides a systematic approach to achieving the project goals and aims to optimize the integration of theoretical studies with practical applications. With this approach, each step is structured to contribute to the overall success of the project.

A. Discrete Time Artificial Intelligence Algorithms

One of the first and most crucial stages of the project is the design of artificial intelligence algorithms in discrete time and their simultaneous simulation in the Unity game engine environment. These tools and environments were chosen considering the engineer's proficiency and the tools' effectiveness and efficiency in the project. The design and simulation process is divided into three main stages in terms of functionality:

a) Algorithm Design: In this stage, the fundamental principles of artificial intelligence algorithms are established and mathematical models are created. These algorithms, optimized to operate in discrete time intervals, are designed to enable robots to make the most efficient decisions in material replenishment processes. This process requires a rigorous effort, blending theoretical knowledge with practical applications.

b) Simulation Environment Setup: To evaluate and verify the performance of the algorithms, a detailed simulation environment is created using the Unity game engine. This environment allows for analyzing how the algorithms perform in different scenarios by best mimicking the anticipated conditions. Simulation parameters are carefully adjusted and tested in this stage.

c) Concurrent Simulation and Analysis: In the final stage, the artificial intelligence algorithms are run concurrently in the simulation environment and their performances are analyzed in detail. This process is critical for evaluating the effectiveness and accuracy of the algorithms. The simulation results provide valuable insights into how the algorithms will perform and allow for necessary improvements.

1) First Stage: Single Command Execution: The AI model is expected to learn the following behavior over time: A single robot will transport a single box from a specific starting position to the target position in the 3D grid area in the shortest path and according to predetermined rules. If there is another box on top of the selected box, it will be moved to a random location to clear the path for this stage. The algorithm was created in accordance with the desired behavior rules, and numerical calculations were optimized iteratively using the results and graphics from simulations. The data and the algorithm obtained at this stage represent the core of the design and provide important clues on how to develop more complex tasks in subsequent stages. Therefore, the single command execution stage is crucial for reinforcing the robots' basic competencies and laying a solid foundation for more advanced tasks. The rules and the algorithm will be described in more detail in the relevant sections.

2) Second Stage: Sequential Command Execution: In the second stage, the goal is to instruct a single robot to reposition a series of boxes. This stage is designed to test and develop the robot's ability to perform more complex tasks. The robot must learn the optimal movement strategy over time while moving multiple boxes in a specific sequence. For instance, if there is another box on top of a selected box, it must be moved so that other target boxes can be transported without obstructing their paths. In this process, the robot uses various algorithms to ensure each box is repositioned in the most efficient way. These algorithms aim not only to move a specific box but also to ensure the optimal placement of other boxes within the series. This increases the robot's capacity to handle more complex scenarios and optimizes the processes. In this stage, the robot's ability to quickly and accurately respond to environmental factors is also developed. As the robot performs the repositioning operations at each step, it will evaluate the entire process and create the most efficient roadmap. Thus, the sequential command execution stage enhances the robot's ability to manage not only individual tasks but also sequential and interconnected tasks successfully. The successful completion of this stage represents a significant step towards enabling the robot to work effectively in more complex production line scenarios.

3) Third Stage: Collaboration Among Multiple Robots: The third stage ensures significant progress towards more complex and collaborative AI functionalities by exploring the logic and application of swarm intelligence behavior among robots. This stage aims for robots to learn how to accomplish larger and more complex tasks by working together. Given that a single robot may be insufficient to execute commands over a larger map, we decided to implement the design and application of swarm intelligence. This transition requires the use of multiple robots, which will collaboratively execute sequential commands, increasing efficiency and coverage. In this stage, multiple robots will collaboratively perform tasks learned in the first and second stages. Each robot will coordinate with others to execute their tasks by learning the optimal movement strategies over time. This collaboration ensures that robots

find the most efficient paths and complete their tasks without hindering each other. This collaboration among robots not only ensures faster and more effective task completion but also enhances the overall performance and adaptability of the system. This stage is critical for demonstrating how robots can successfully work together in more complex production line scenarios. Swarm intelligence algorithms and learning processes will allow for the creation of a flexible production line by providing tolerance to errors in individual robots. As a result, in the third stage, multiple robots will perform the tasks achieved in the first and second stages collaboratively, making significant progress in the project and developing the learning and adaptation skills necessary to perform more complex tasks.

B. Continuous Time Integration

Various studies will be conducted in three stages to load the algorithms, realized and tested in a computer environment, into robots operating in a continuous environment.

1) Robot Design and Preparation: The most suitable robot unit for the defined environment and working conditions will be created in the Gazebo environment with all its mechanics. Necessary sensor, actuator, and controller connections will be made with various Gazebo plugins, and necessary ROS Nodes will be written to make it ready for operation. During this stage, the integration of mechanical and software components will be focused on while designing and preparing the robot. The robot's mobility, environmental sensing capabilities, and control systems will be carefully established.

2) Navigation and Localization: The ground robot moving on the platform must reach the x-y positions specified in each grid to pick up boxes. For this movement, the vehicle's localization and mapping operations will be carried out, and then navigation and position control approaches will be used to reach the exact position. Mapping algorithms and localization techniques will be developed for the robot to reach accurate positions, and navigation strategies will be applied.

3) Integration of AI Algorithms: AI algorithms will be integrated into the system and robots to be tested in the Gazebo environment as well. During this integration process, ensuring the algorithms work in harmony with the robot's hardware and software will be a priority. AI algorithms will be designed and tested to adapt to different scenarios encountered during the robots' task execution processes. Simulation results will be analyzed to evaluate and optimize the system's performance.

III. Design and Algorithm Overview

Upon the recommendation and guidance of the advisor, the project prioritized the artificial intelligence (AI) component, and extensive work was carried out in this area throughout the year. The focus was on developing and optimizing AI algorithms, with detailed tests conducted in simulation environments. The design and development of the algorithms played a crucial role in achieving the project's objectives and demonstrated the potential to enhance the efficiency and effectiveness of robotic systems to be integrated into production processes.

A. Designed Environment and Conditions

The design and testing processes of the algorithms used in the project were carried out considering specific assumptions and needs. To make these processes more visualizable and to accelerate debugging, the Unity game engine was used, and C# was preferred. Unity's flexible and powerful structure provides an excellent platform for evaluating the accuracy and efficiency of the algorithms. Thus, the performance of the algorithms was tested in a realistic and dynamic environment, allowing for more effective and reliable development of practical applications. Figure 1 shows this environment.

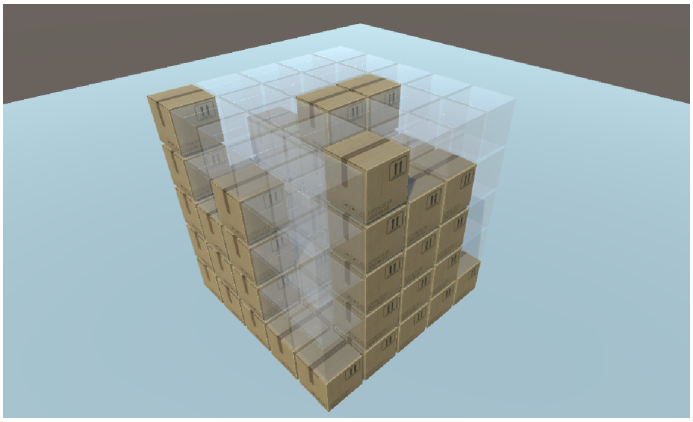


Fig. 1. Visual representation of the Unity environment where the algorithm is applied.

B. The Environment Where the Algorithm is Applied

With the prepared program, a box grid system of given nxn size with a random fill ratio is created each time it is initialized. This grid system realistically simulates the movements and interactions of the algorithm on the boxes, thus providing insights into the practical applicability of the algorithms. Figure 2 shows detailed visuals of the boxes and transparent movement areas.

Fig. 2. Detailed visuals of the boxes and transparent movement areas.

The robot is planned to be designed as a four-wheeled ground vehicle in real life, located on a platform above the boxes. This arrangement is optimized to allow the vehicle to move effectively in 2D and simultaneously carry boxes. Small holes are placed above each box column on the platform so

that the robot's arm can lift the box upwards. These holes are also designed to provide maximum access without obstructing the vehicle's movement. When the robot lifts the box upwards with the arm mechanism, it will hold the box with the help of an electromagnet when it reaches the top point, and after the arm is detached, the magnetically attached box can be dragged on the platform. The planned structure's top-down and side views are shown in Figure 3.

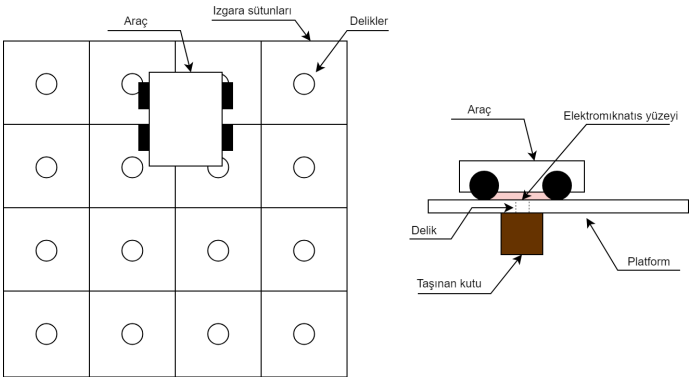


Fig. 3. Top-down and side views of the planned structure.

However, within the scope of this project, no work has been done on how to design the arm or magnetic mechanism in real life. These elements are conceived to add realism to the system and increase the comprehensibility of the concept. The main focus of the project is the development of artificial intelligence algorithms and the effective simulation of robotic movements in a simulated environment.

Nevertheless, these details are important as they are critical elements that need to be carefully planned at the project's outset to effectively manage the movement dynamics of the box during transport. In the specified configuration, the vehicle's movements are primarily designed to occur in a 2-dimensional plane: movements to the right, left, up, and down, along with rules for raising and lowering the box in the third axis. This allows the vehicle to maneuver effectively on both horizontal and vertical axes and plays a role in the box relocation processes.

The importance of these details lies in the fact that these movements performed by the vehicle are integrated in harmony with the actions defined in the reinforcement learning (RL) algorithms. RL algorithms will use these movements to develop strategies for the vehicle to transport boxes to target positions on the grid in the most efficient manner and will continuously make improvements using the experiences gained in this process. This approach provides a realistic simulation environment and plays a critical role in maximizing the vehicle's maneuverability and the efficiency of material handling processes.

C. Q Learning

The Q Learning algorithm is one of the reinforcement learning methods that enables agents to learn optimal actions in a given environment by earning rewards. This algorithm allows agents to make decisions that aim to maximize long-term rewards and develop strategies accordingly. Q Learning enables agents to learn the best course of action over time by assigning a value to each state-action pair. Throughout the project's decision-making phase, algorithms based on Q Learning principles were designed and implemented from the outset. The agents mentioned here are the subjects of the Q Learning algorithms. For instance, the agent in Stage 1 is the robot transporting any box.

1) General Program Flow: The Q Learning algorithms developed in the project operate within a specific program flow logic. This flow covers the process from the start of the algorithm to the completion of training. In addition to the Q Table used as a database, similar structures suitable for requirements were also used in the problem stages, but they will be referred to as the Q Table representatively when explaining the program flow. The states and actions will be explained according to Stage 1, and the specifications for Stages 2 and 3 will be detailed in subsequent sections with references here.

a) Initialization Stage: When the algorithm starts, the Q Table is initialized with starting values (zero). This table assigns a value for each state (position vector of the box carried by the agent in Stage 1) and six actions (movements in the x-z plane: right-left-up-down and movements in the y-axis: pulling or lowering the box) and is continuously updated during the training process.

b) Update Loop: The algorithm updates at specified time intervals (frames). Each update occurs based on a specific timer adjusted to the simulation's animation duration. The following steps are followed for the number of steps specified in the loop:

c) Action Selection and Execution: The agent selects an action based on its current state (the position of the box it is carrying in Stage 1) and attempts to execute this action. This process continues with the agent moving the box towards its target according to the chosen action.

d) Reward Calculation and Learning: After each movement, the agent receives a reward based on the action performed. This reward is calculated according to the Q Learning method and contributes to the agent's learning process by updating the relevant parameters in the Q Table.

If the specified number of steps is reached (the box is not delivered) or the box is delivered to the designated target, the following step is followed.

e) Starting a New Episode: After completing each episode, the algorithm proceeds to a new episode. In the new episode, the positions of the agent and the box are reset, and the grid box structure returns to its initial state. If the training is completed, the algorithm performs the necessary termination procedures.

2) Action Selection: In the Q Learning algorithm, action selection is based on the principles of exploration and exploitation. This process forms the basis of the agent's decision-making mechanism and directly affects learning performance.

a) Exploration and Exploration Rate: During exploration, the agent aims to discover new movements. Therefore, random action selection is active while the algorithm is in exploration mode. This process results in the agent trying unknown or less preferred actions. Exploration or exploitation is managed by a condition dependent on the epsilon value; if a random number is less than the epsilon value, the agent operates in exploration mode. The exploration rate, or epsilon value, decreases (decays) with each episode, causing the agent to explore less and use known actions more over time. This decline allows the agent to make more predictable and optimized decisions as it gains experience, leading to convergence in selections.

b) Exploitation: The agent performs the best-known action for its current state using previously learned information. If the randomly selected number is greater than the epsilon value, the agent enters exploitation mode. In this case, the Q Table is examined, and the action with the highest Q value for the current state is chosen. Exploitation allows the agent to utilize previous experiences and perform the best-known action.

The action selection process enables the agent to explore new strategies and apply what it has learned to achieve its goals most effectively. As epsilon decreases, the agent's behavior shifts from exploration to exploitation over time. This balanced approach allows the agent to discover new and potentially better solutions and effectively use its existing knowledge. This process ensures the continuous development of the agent's decision-making ability and increases the overall success of the algorithm.

3) Decreasing Exploration Rate Over Time: The epsilon value is a crucial parameter that manages the balance between exploration and exploitation in the Q Learning algorithm. The epsilon value starts from an initial value and begins to decrease with each episode. This decrease process has been approached in two different methods:

a) Linear Decay Method: In this method, the epsilon value is reduced by multiplying it with a certain rate in each episode. To ensure that epsilon reaches a minimum value, it is compared with a predetermined minimum epsilon value, and the larger of the two is selected.

b) Decay Method Using a Temperature Function: In this alternative approach, epsilon reduction was attempted using a temperature function based on the current state and action. This method aims to allow the algorithm to try different actions in more complex scenarios. However, no significant performance change was observed during the implementation of this method.

As a result, it was decided to continue with the simpler and more predictable first method. The linear decay method effectively manages the balance between exploration and exploitation, allowing the algorithm to use its learned information more while exploring less over time. This approach enhances the algorithm's overall performance and decision-making efficiency while providing simplicity and understandability advantages.

4) Reward/Penalty Mechanism: The reward/penalty mechanisms designed in the algorithms form the basis of the agent's learning process. These mechanisms evaluate each movement of the agent and provide a reward or penalty. The amount of reward is determined based on the efficiency of the action performed and its outcome.

a) Defining Reward Rules: This process is essentially equivalent to defining the Q Learning problem. Reward rules are carefully determined to guide the agent's movements and ensure it reaches specific goals. In the scope of the project, these rules for each stage's algorithms were created with an experimental and iterative approach according to the defined problems. Each rule shapes the Q table over time.

b) Evaluation and Iteration of Rules: After adding each rule, the agent's performance and changes in the Q Table are examined. This evaluation is used to assess whether the agent is exhibiting a specific behavior, approaching the desired action, and the effectiveness of the learning process. Rules are incrementally added and continuously updated to ensure the agent progresses towards the desired behaviors.

c) Monitoring and Improving Results: In each iteration, the agent's behaviors and changes in the Q table are observed. This observation allows for necessary adjustments in the algorithm to enhance the learning process's effectiveness and achieve better results.

The reward/penalty mechanism forms the basis of the agent's learning process and enables it to act more effectively and efficiently. This process allows the agent to continuously improve and reach the defined goals.

5) Learning Mechanism: The learning process in the algorithm occurs by updating the Q-values in the Q-table based on the agent's experiences. The Q-value represents the expected benefit of taking a specific action in a given state and is shaped during the learning phase, accounting for both immediate and future rewards. The Q Learning update rule is expressed in Equation 1.

$$Q(s,a) \leftarrow (1-\alpha) \times Q(s,a) + \alpha \times (r + \gamma \max Q(s',a')) \quad (1)$$

The block diagram of the Q Learning process is illustrated in Figure 4.

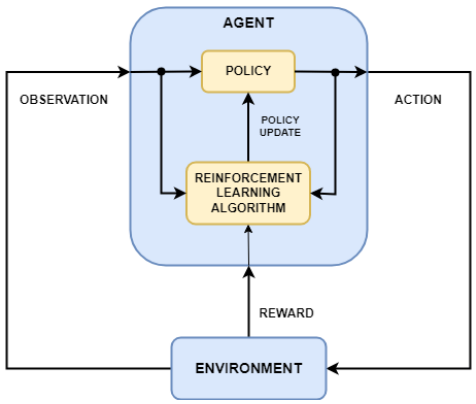


Fig. 4. Q Learning block diagram (Mathworks).

The explanation of the equation is as follows:

- **Old Q-value** ($Q(s, a)$): The Q-value for the action (a) taken in the old state (s) represents the agent's knowledge before this update.
- **Learning rate** (α): α determines the impact of new information on the updated Q-value. A high α gives more priority to new information, while a low α gives more weight to old data.
- **Immediate reward** (r): The gain obtained immediately after taking an action. It reflects the direct result of the action.
- **Discount factor** (γ): γ, between 0 and 1, indicates the impact of future rewards on the current Q value. A high γ makes the agent more strategic by emphasizing future rewards.
- **Maximum future Q-value** ($\max Q(s', a')$): Represents the highest Q-value for all possible actions in the new state. It estimates the optimal future rewards obtainable from the new state.
- **Combining immediate and future rewards**: The term $r + \gamma \max Q(s', a')$ combines the immediate reward with the discounted future rewards. This sum represents the total expected benefit of taking the action in the current state.
- **Updating the Q-value**: The final updated Q-value is a mixture of the old value and this new information, modulated by the learning rate α.

a) Importance of the Gamma (γ) Parameter: The γ parameter increases the agent's "foresight," allowing it to consider long-term results instead of just immediate outcomes. In complex decision-making tasks where future rewards are significant, a higher γ helps the agent make more strategic plans and consider the long-term benefits of its actions.

6) Parameter Adjustment for Specific Problems: This learning mechanism in the Q Learning algorithm has been optimized for each specific problem in the project. For example, in Stage 1, parameters were adjusted using an experimental and iterative approach for a specific movement problem, such as moving boxes according to rules. This optimization allows the agent to consider not only immediate rewards (such as reaching the target) but also the broader context of its actions.

This fine-tuning process enhances the agent's decision-making ability and ensures that it learns to act effectively in the given scenarios.

IV. Design of Artificial Intelligence and Learning Model

For each problem addressed in the first, second, and third stages, how the mentioned learning logic is used, why it is necessary, and how it is applied will be examined in subsections according to the stage. The unique requirements of each stage and their impact on the design of the artificial intelligence and learning model will be detailed. Additionally, the effect and performance of the methods used on the algorithms will be evaluated, emphasizing the key elements that contribute to the overall success of the project.

A. Stage One: Single Command Learning

In the first stage, the robot's goal is to learn how to move a single box to a specified point in a 3D grid as quickly as possible, following specific rules. The Q-table used in this stage involves mapping states to actions; where each state represents a position in the grid, and each action represents a possible movement of the box.

The primary goal of the robot carrying a single box in this environment is to take the necessary steps to lift the box to the highest plane, move it to the target column, and lower it. The reward system has been gradually balanced and fine-tuned to achieve this goal and maximize performance. These adjustments are meticulously implemented to encourage desired behaviors and discourage deviations. Detailed explanations of the rationale behind each change and how they contribute to achieving the overall goal are provided in the subsequent sections.

1) Reward Rules: The development of the reward system and rules is explained step-by-step.

a) First Chapter Reward Rules and Results: In the first chapter, reward and penalty rules are set to encourage the box to follow the shortest path. These rules aim to influence the agent's movements and decisions to learn to deliver the box to the target most efficiently over time.

Penalty for each step: Each unit movement of the box incurs a small penalty, reflecting the cost of each movement and encouraging the box to reach the target in the fewest steps. This approach helps the agent avoid unnecessary movements and seek more direct paths. Mathematically, this penalty is expressed as:

$$r_s = -0.05 \tag{2}$$

Reward for reaching the target: If the box successfully reaches its target, a significant reward is given to encourage this success, motivating the agent to deliver the boxes to the target. Mathematically, this reward is expressed as:

$$r_h = 1.0 \tag{3}$$

Penalty for immobility: When the box cannot move due to any obstacle or constraint, a larger penalty is applied. This penalty aims to prevent situations where the box cannot move and encourages the agent to seek alternative paths to overcome obstacles. Mathematically, this penalty is expressed as:

$$r_e = -0.5 \tag{4}$$

These initial rules are designed to ensure the box reaches the target in the most effective and efficient way. By emphasizing that every movement has a cost and delivering to the target brings significant reward, they effectively guide the agent's decision-making process.

The result of the first stage reward rules allows the selected box to optimize its route to the target position after several episodes, but it does not comply with the rules of first moving up, then moving on the top plane, and finally lowering down at the target column.

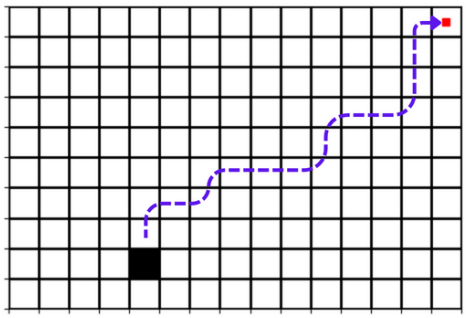


Fig. 5. Goal of first chapter rules in 2D simulation

b) Second Chapter Reward Rules and Results: In the second chapter, specific reward rules are set to direct the box's movements precisely as desired.

Reward for moving to the top plane: A special reward rule is added to encourage the agent to move the box to the top plane in its column. If the agent keeps the x and z coordinates the same as the starting position while moving upward in the y coordinate, it receives a reward for this successful upward movement. Mathematically, this reward is expressed as:

$$r_u = 1.25 \tag{5}$$

This reward rule successfully encourages the agent to move the box upward but can cause the box to get stuck between the highest two positions, as shown in Figure 6.

Fig. 6. The box getting stuck between the highest two positions.

Penalty for downward movement: To suppress the effect mentioned above and ensure the box moves in 2D on the top plane after reaching the top, a penalty is applied if the box moves downward in a column other than the target column. Mathematically, this penalty is expressed as:

$$r_d = -2.25 \tag{6}$$

Although this prevents the box from getting stuck between the two highest positions, it was observed that the agent encountered another problem. The agent would move the box up, move left, lower it, and then move right to return to the initial position, resulting in the box being stuck in cycles of 4 or 6 steps, as shown in Figure 7.

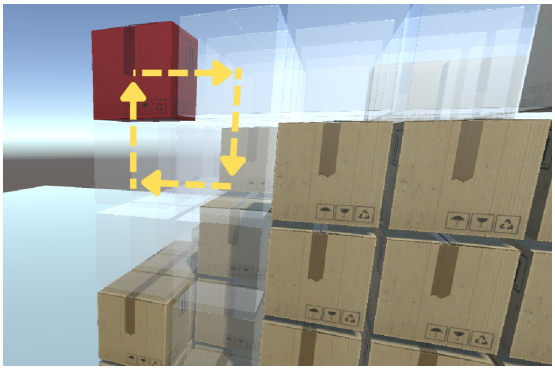


Fig. 7. The box stuck in cycles of 4 or 6 steps.

c) Third Chapter Reward Rules and Results: In the third chapter, new reward rules are added to prevent the box from performing cyclical and undesired movements.

Penalty for cyclical movements: A history buffer keeping a record of recent moves is created to prevent the box from revisiting previously visited states. If the box returns to the same state within a short period, a significant penalty is applied to discourage this repetition. Mathematically, this penalty is expressed as:

$$r_c = -2.0 \tag{7}$$

Penalty for wrong direction: When the box reaches the target column, it is encouraged to move downward. If the box moves in another direction when it is at the x and z coordinates of the target column, an additional penalty is applied. Mathematically, this penalty is expressed as:

$$r_w = -0.75 \tag{8}$$

Adjustment of target reach reward: Initially, a high reward is given for reaching the target. However, this led to the box preferring shorter paths to targets near the starting position and ignoring the upward movement strategy. To correct this, the target reach reward is lowered. Mathematically, this reward is expressed as:

$$r_h = 0.2 \tag{9}$$

These adjustments significantly improved the agent's learning process and decision-making mechanism. The selected box now performs fewer cyclical movements and prefers more direct paths to the target. The iterative improvements in the reward rules enhanced overall performance and the algorithm's effectiveness.

d) Fourth Chapter Reward Rules and Results: After observing all errors sufficiently, several changes were made to the rewards to optimize the process's performance.

Change in the penalty for cyclical movements: The size of the history buffer preventing returns to previously visited states was increased from 6 to 8. This adjustment was made in response to observations of getting stuck in cycles of 8 steps and suggests the possibility of dynamically adjusting the buffer size according to grid dimensions in the future.

Change in the upward movement reward: In rare cases, it was observed that the box reached the target by using a lower plane without moving to the highest plane. This problem is illustrated in Figure 8.

Fig. 8. The issue of the box continuing without reaching the highest plane.

To prevent this, the upward movement reward was increased from 1.25 to 1.75. This adjustment also prevents the box from taking a shortcut to the target nearby. Mathematically, this reward is expressed as:

$$r_u = 1.75 \tag{10}$$

Change in the penalty for each step: The penalty for each step was increased from 0.05 to 0.22 to balance the other newly introduced rewards. Mathematically, this penalty is expressed as:

$$r_s = -0.22 \tag{11}$$

The result of these changes is that no significant errors were observed in Stage 1 after these adjustments and balancing efforts. The fine-tuning of the rewards and penalties effectively mitigated previous issues, providing improved performance and stability in the environment.

e) Summary of All Rewards:

- Penalty for cyclical movements r_c: If the box returns to a state visited within the last 8 moves, a penalty of -2.0 is applied to prevent repetition.
- Penalty for wrong direction r_w: If the box moves in a direction other than downward while in the target column, a penalty of -0.75 is applied.
- Reward for reaching the target r_h: If the box reaches its target, a reward of 0.2 is given for this successful movement.
- Penalty for immobility r_e: If the box cannot move due to obstacles or constraints, a penalty of -0.5 is applied.
- Reward for moving to the top plane r_u: If the box moves upward while aligned with the x and z coordinates of the starting position, a reward of 1.75 is given for this successful ascent.

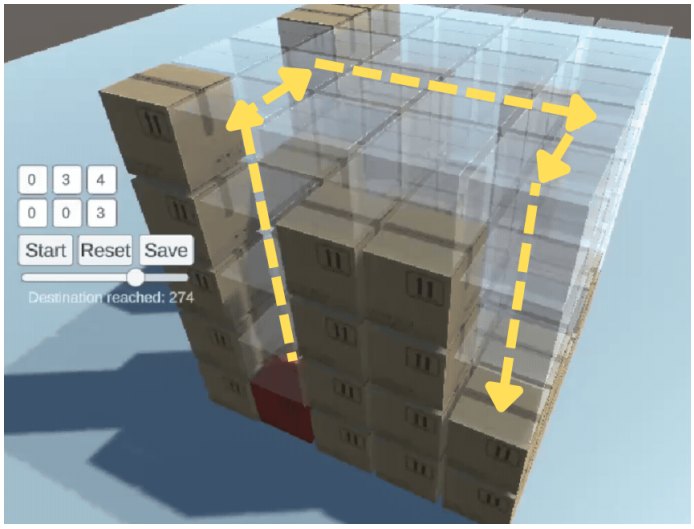


Fig. 9. Visualization of the problem where the box prefers a longer route.

- Penalty for downward movement r_d: If the box moves downward in a column other than the target column, a penalty of -2.25 is applied.
- Penalty for each step r_s: A penalty of -0.22 is applied for each movement.

The total reward for a step r is calculated as:

$$r = r_s + r_h + r_e + r_u + r_d + r_c + r_w \tag{12}$$

2) Results: The first stage of the project has been completed as planned. The optimized algorithm has been successfully tested on the designated route. The obtained results have been visualized and evaluated through various graphs. Figure 10 shows the route and movement strategy of the box moved by the agent during a test phase.

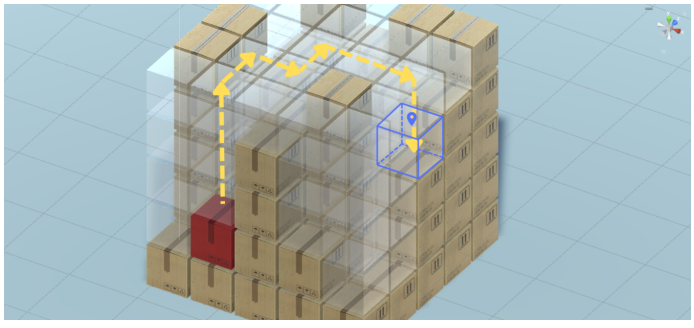


Fig. 10. Visualization of the test route.

The success of reaching the target for each episode is shown in Figure 11, plotted against the episode number. This graph reflects the agent's improvement in reaching the target and the success rate over time. It is observed that the box reaches the target in every episode after the first 15 episodes.

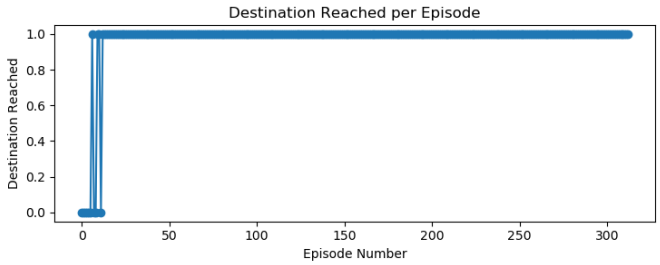


Fig. 11. Graph of success per episode.

Figure 12 shows the distance traveled for each episode compared to the scalar distance between the starting and target positions. This illustrates the efficiency of the agent's route and how much distance was covered to reach the target. It is observed that the box follows the optimal route after a certain episode.

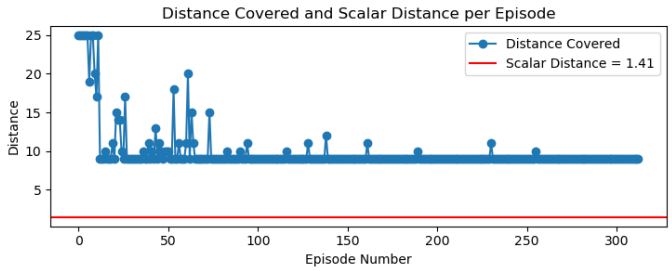


Fig. 12. Graph of distance traveled per episode and scalar distance.

Figure 13 shows how the epsilon value changes per episode according to the decay function we set. This graph examines how the epsilon value decreases over time and the changes in the balance between exploration and exploitation.

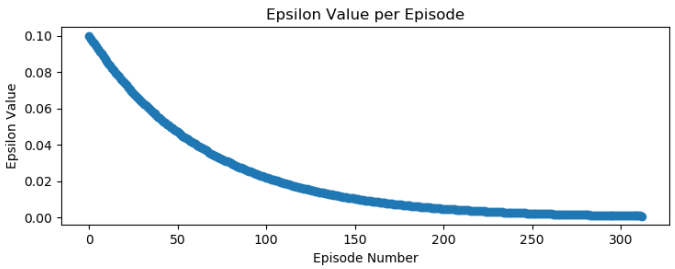


Fig. 13. Graph of epsilon value per episode.

Figure 14 compares the amounts of exploration and exploitation in each episode. This graph reflects the changes in the agent's behavior based on the epsilon value and the evolution of the learning process. Initially, and at some points, there have been fluctuations in these parameters, which later settled at certain values.

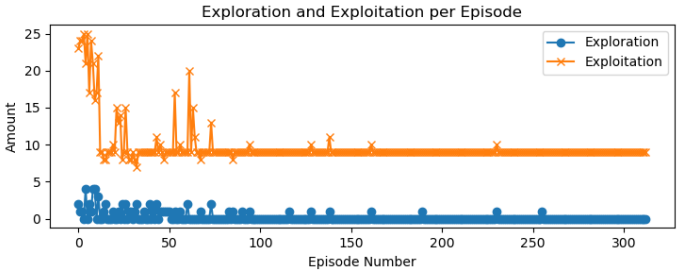


Fig. 14. Graph of exploration and exploitation per episode.

Figure 15 shows the total reward obtained in each episode as a ratio to the distance traveled. This indicates the agent's efficiency and performance changes between episodes. The maximization and stabilization of this parameter over the increasing number of episodes demonstrate the algorithm's success.

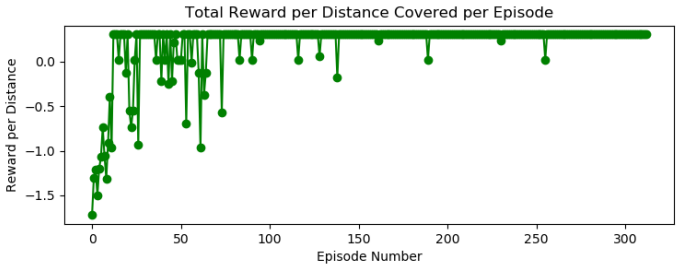


Fig. 15. Graph of total reward per distance traveled per episode.

Figure 16 shows the number of impossible and undesirable movements in each episode. This graph visualizes the errors made during the box transportation process. Impossible movements represent maneuvers that the box cannot make due to lack of space, while undesirable movements represent deviations from the desired behavior.

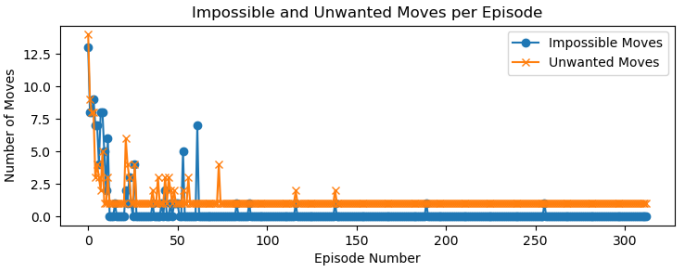


Fig. 16. Graph of impossible and undesirable movements per episode.

B. Stage Two: Sequential Command Learning

In this stage, the goal is for the algorithm to learn how to execute the single commands applied in Stage 1 in a sequential manner, directing a series of boxes to their desired positions in order. It is important to note that the learning process for each command can impact the execution of other commands in the sequence.

The core of Stage 2 involves the implementation of box cleaning mechanics, which necessitates a different learning

model from Stage 1. This process focuses on repositioning boxes according to two main criteria:

- **Boxes on top of the selected box**: The algorithm aims to clear the boxes on top of the currently selected box to create a clear path for its movement.
- **Boxes obstructing the path**: Additionally, the algorithm focuses on clearing the boxes obstructing the path of the selected box, ensuring unobstructed movement towards the target, thereby achieving the best results as seen in Stage 1.

1) Design:

a) Box Cleaning Mechanics: The box cleaning mechanics were first established, defining the process of repositioning boxes to ensure the unobstructed movement of the target box. Two main steps were undertaken to make this process more effective and ensure that the boxes were repositioned in the most efficient manner.

Initially, a method was applied to randomly reposition the boxes on top of the target box according to the established mechanics. This step aimed to create the initial conditions for the box cleaning process. The random repositioning method allows the system to create a variety of initial states, enabling the algorithm to encounter different scenarios and learn from a broader spectrum. Randomness aims to ensure that the algorithm is effective not only in specific arrangements but also in general diverse situations.

After the random repositioning step, a newly designed Q-learning logic was integrated to teach the algorithm to reposition the boxes in the most optimal and logical way. In this stage, the algorithm was directed to place the boxes according to specific rules and reward structures. For example, it aimed to position the boxes moved from above the target box in the least obstructive locations. This approach is critical for increasing the effectiveness of the box cleaning process.

The new Q-learning logic involved designing reward structures and updating Q-values. Reward structures play a crucial role in reinforcing desired behaviors and discouraging undesired behaviors. For instance, appropriately moving a box from above the target box is rewarded, while inappropriate movements are penalized. This reward and penalty system enables the algorithm to develop the most efficient strategies.

The process of updating Q-values allows the algorithm to adjust its strategies based on the feedback received at each step. This continuous feedback and adjustment loop enable the algorithm to become smarter and more effective over time. Moreover, the Q-learning algorithm allows the system to quickly adapt to dynamic changes and achieve the most optimal results. .

b) Initial Conditions for Box Cleaning: The initial condition for the box cleaning mechanics involves repositioning a box on top of the selected box to an adjacent position, if available, according to the rules applied in Stage 1. This approach aims to minimize the distance boxes need to move and reduce unnecessary movements.

If the targeted repositioning point is occupied, the box occupying that space must also be repositioned. This can trigger a chain of repositioning, where each box must find the most suitable spot. Additionally, boxes above the targeted repositioning point must also be moved appropriately. This is necessary to prevent obstruction of movements and maintain system integrity.

The following figures present a demo showing the process of clearing the boxes above the selected box, where the red box is the selected box. The arrows indicate the positions determined by the algorithm for the boxes to be cleared.

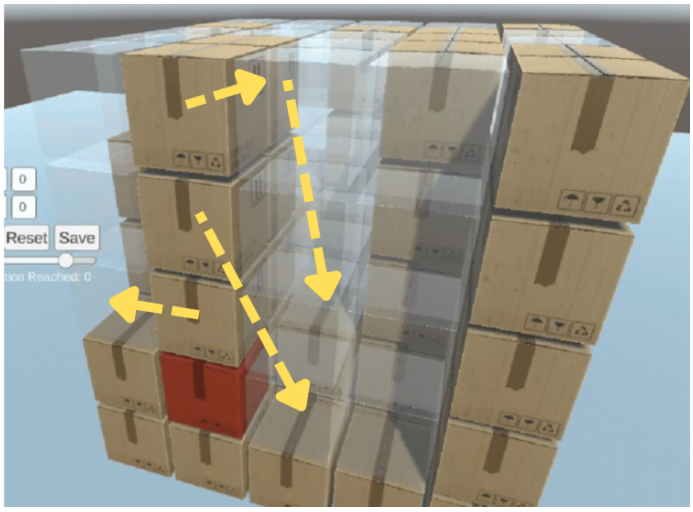


Fig. 17. Demo showing the process of clearing the boxes above the selected box.

In practice, the logic and rules of Stage 1 will be applied for each box repositioning. This ensures consistency in the system and correctly meets the requirements of each stage.

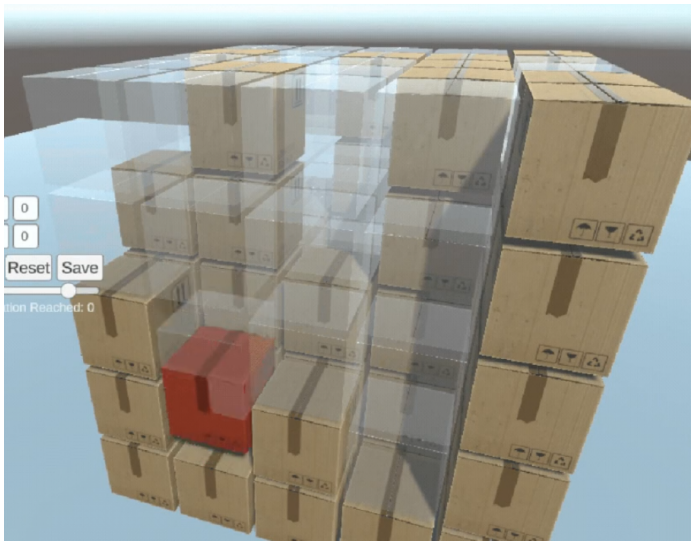


Fig. 18. System after box cleaning.

c) Learning Tree Structure: Instead of relying on random movements in the approach to box cleaning mechanics, the aim is for the agent to learn these mechanics. To achieve this, a

QTree has been introduced instead of a QTable. As mentioned in the previous section, this QTree represents the movement chains from the root to the leaf nodes.

At the root of the QTree is the box that needs to be repositioned first; this could be the box on top of the target box. During operation, repositioned boxes and the final leaf node are added as child nodes in a chain, resembling a tree structure in data structures. This structure allows the agent to make step-by-step decisions and track the results of each decision, developing mastery over the branches of the tree. Each node of the tree contains the following information:

- **Position information**: The discrete position of the box in 3D space.
- **Q-Value**: The Q-value associated with that position.
- **Children**: Child nodes representing possible actions from that position.
- **Parent**: The parent node.

This hierarchical structure allows for a more organized representation and tracking of possible actions and associated Q-values, facilitating more efficient learning and decision-making processes for the agent. The QTree supports logical and effective decision-making by the agent, even in complex problem-solving processes.

Each QTree is added to a list of QTrees, ensuring they are organized and accessible for learning and decision-making processes. This list allows the agent to evaluate all possible scenarios and corresponding actions for any box to be repositioned. Additionally, the QTree structure makes the agent's learning process more flexible and dynamic, allowing quick adaptation to new situations. This structural approach enhances the algorithm's overall performance and efficiency, enabling the system to operate more effectively.

An example figure visualizing the tree structure is provided in Figure 19. Nodes are created during operation.

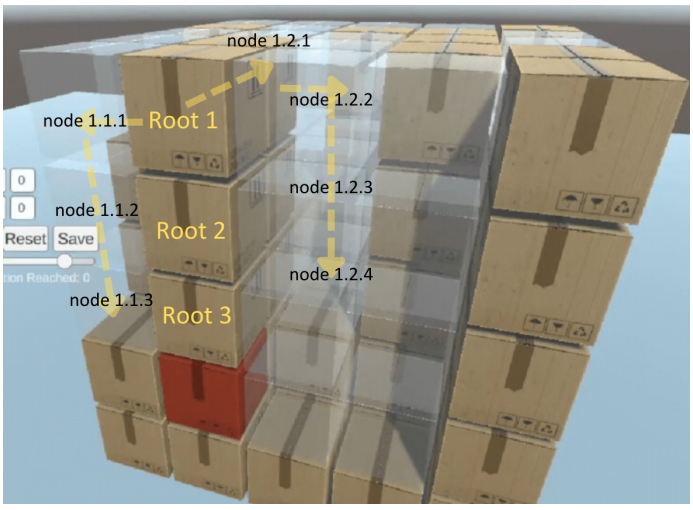


Fig. 19. Visualization of the tree structure.

In our system, there is a method for rewarding the Q-value of a node and the entire parent chain in a tree-like manner. This ensures that when a node is rewarded or penalized, the effects spread along the branch of the tree. This method highlights the impact of each movement on the overall strategy, enabling the agent to make more informed decisions.

Specifically, when a node receives a reward or penalty, all nodes in the branch are similarly encouraged or discouraged. This prevents the agent from focusing solely on individual movements, allowing it to evaluate the broader consequences of each action. For example, when a box is successfully repositioned, not only that movement but all movements leading to it are rewarded. Similarly, if a negative outcome occurs, all movements leading to that result are penalized.

This approach ensures that actions leading to positive outcomes are reinforced along the entire action chain, while actions leading to negative outcomes are discouraged. This allows the agent to make more informed decisions by evaluating the consequences of each action in a broader context. By learning from past experiences, the agent develops more effective and efficient strategies. This method enhances the system's overall performance while accelerating and optimizing the learning process.

d) Rewarding: Penalty for each step within branches: Each branch's leaf node and the entire branch (decreasing towards the root at a certain rate) are penalized similarly to the "penalty for each step" in Stage 1 of the project. The penalty for the leaf node is set at $r_{t,0} = -0.12$, and the penalty value decreases by 10% at each step towards the root. Mathematically, this penalty is expressed as:

$$r_{t,n} = r_{t,0} \cdot (0.9^n) \tag{13}$$

This strategy discourages unnecessary actions, encouraging the exploration of the fastest and easiest branches. Thus, the agent is guided to avoid unnecessary movements and find more efficient paths. This method helps the agent develop optimal solutions by increasing the system's overall efficiency.

Penalty for obstructing nodes: Any box obstructing the path of other boxes in the sequence of commands or those needing to be moved is penalized when obstructing the movement of another box. In this case, the responsible node and the entire branch are similarly penalized, preventing future occurrences of blocking another box's path during transportation. The penalty is expressed as r_e. For a node obstructing primary boxes, the penalty is r_{e1}. Additionally, for movements obstructing less critical boxes being moved to clear the primary boxes, the penalty is r_{e2}. Mathematically, these penalties are expressed as:

$$r_{e1} = -1.25 \cdot (0.9^n) \tag{14}$$

$$r_{e2} = -0.75 \cdot (0.9^n) \tag{15}$$

Timeout penalty: A timeout penalty is applied for the number of boxes that need to be moved during the box cleaning process. This penalty is applied when the number of boxes to be moved in a branch reaches 6, preventing potential time loss or congestion, especially during the initial random selection

phases of Q-learning. Mathematically, this penalty is expressed as:

$$r_z = -0.3 \cdot (0.9^n) \tag{16}$$

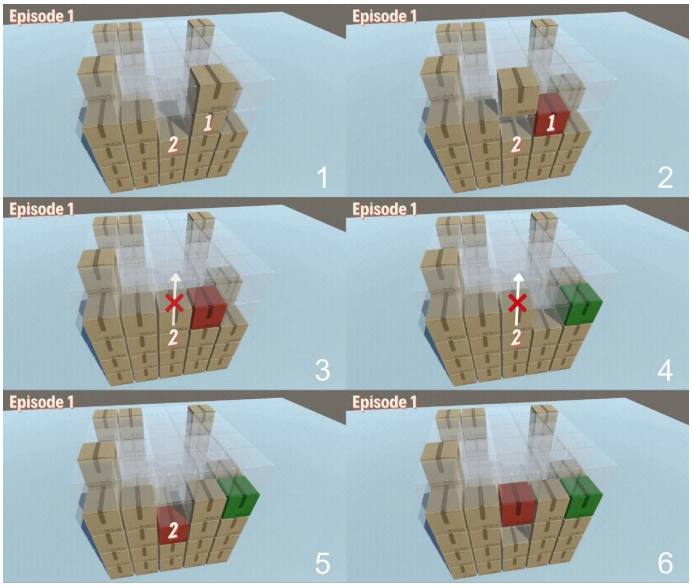
Although the critical number of boxes is currently set statically at 6, there is ongoing work to develop a dynamic structure that adjusts this number based on the occupancy rate around the root box. This mechanism ensures faster and more efficient learning by the algorithm, preventing unnecessary time losses.

According to this new reward and penalty system, the total reward for a step r is calculated as:

$$r = r_{t,n} + r_e + r_z \tag{17}$$

e) Box Cleaning Applications and Results: This section presents examples that examine how the algorithm performs in different scenarios. These examples highlight the learning capability and efficiency of the Q-learning algorithm in developing more effective strategies during the box cleaning and repositioning processes, which are fundamental to the second stage of the project. The figures for the given examples include collages of images for each step in the application. The sequence is noted at the bottom right. Each image contains informative markers and texts. These collages provide a clear visual understanding of each step, offering a clear idea of how the process works.

Example 1) In the following example, the agent is instructed to sequentially move two boxes to specific positions. In the first section, the box on top of the first box is moved to a position that obstructs the path of the second box. This means that when it is time to move the second box, its path is blocked, requiring the previously moved box to be repositioned again. Although both boxes are eventually moved to their required locations, the total movement time and effort increase, complicating the movement process.



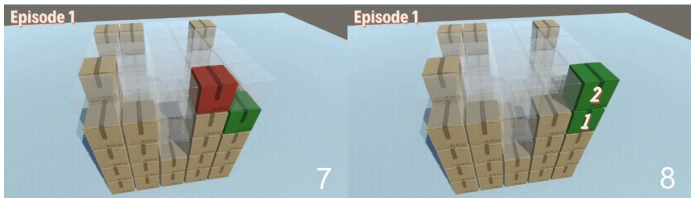


Fig. 20. Repositioning in the first section.

The process in the first section provides a significant learning experience for the agent. By observing the consequences of its actions, the agent learns to make more strategic decisions in the next step.

In the second section, the agent recognizes and learns from this mistake in the first section, avoiding obstructing the path of the second box. This learning represents a critical development for the agent, providing a cleaner and faster path for the second box to reach its target. Implementing this strategy significantly reduces total movement time and effort.

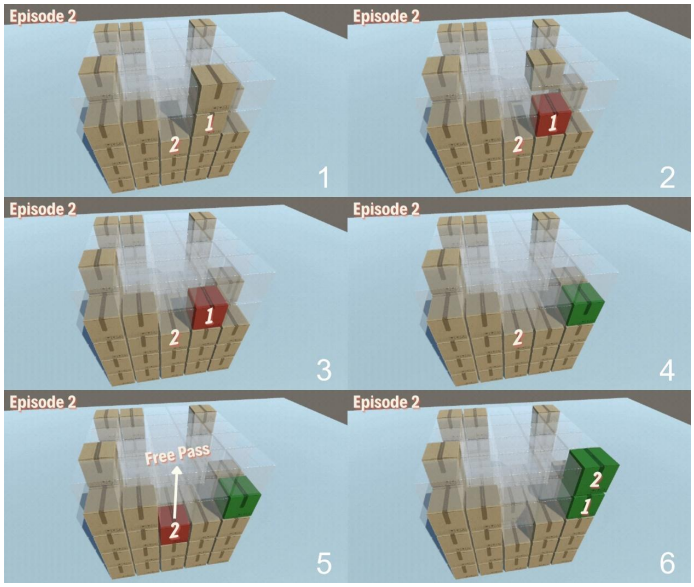


Fig. 21. Second section: Improved strategy after learning.

These examples demonstrate the effectiveness of the Q-learning algorithms designed at this stage and how the learning process evolves. As the agent gains experience in each section, it makes more efficient and strategic decisions, becoming more successful in reaching the target. This process enhances the algorithm's overall performance and provides more optimal solutions over time.

Example 2) In the second example, the agent is similarly instructed to move the selected box to a designated location. Unfortunately, the box above it is placed precisely in the spot where the target box needs to go during the box cleaning process. This action is severely penalized. The issue is presented initially in section 1 and then resolved in section 2 by selecting another branch for box cleaning due to the penalty's effect.

This process demonstrates the agent's learning and adjustment capabilities.

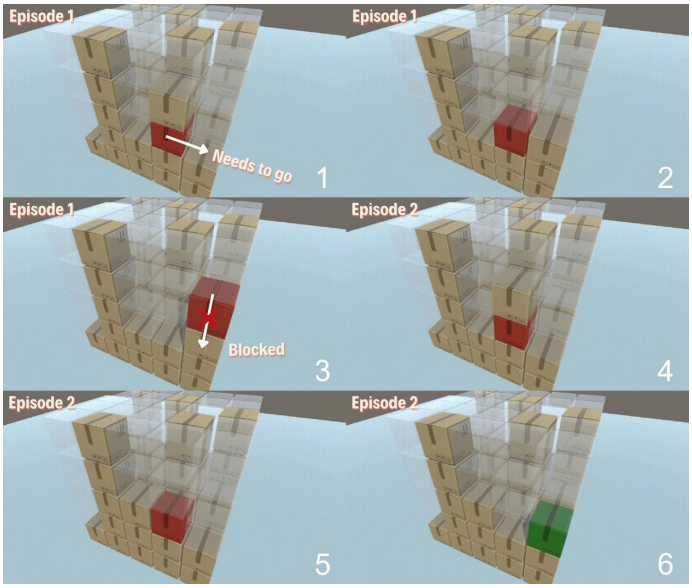
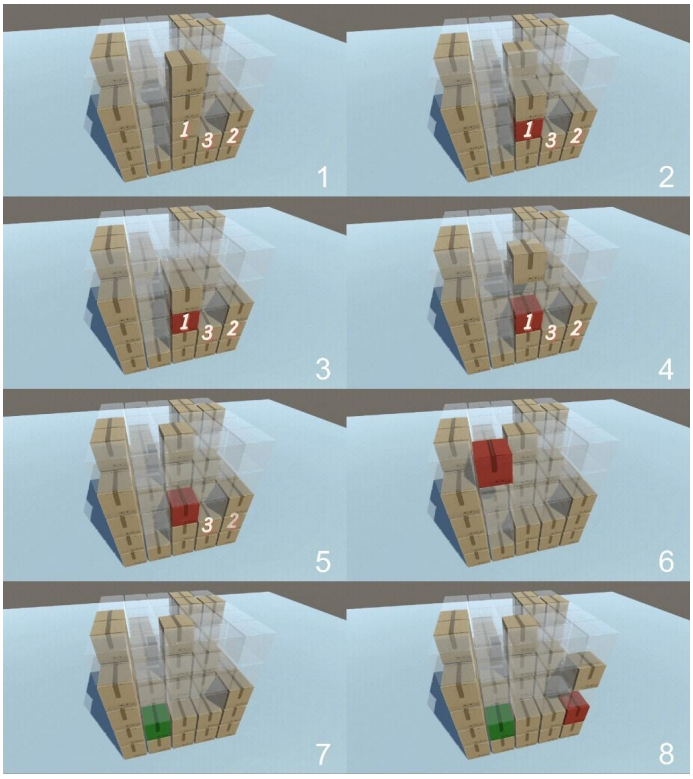


Fig. 22. Section 1 and 2: Problematic repositioning and solution.

Example 3 Figure **??** shows the final state of movement in the model trained according to the criteria of stages 1 and 2. Three boxes are sequentially commanded and moved to their targets most efficiently. This example demonstrates that when the learning process is successfully completed, the algorithm can move the boxes to their targets with minimal effort.



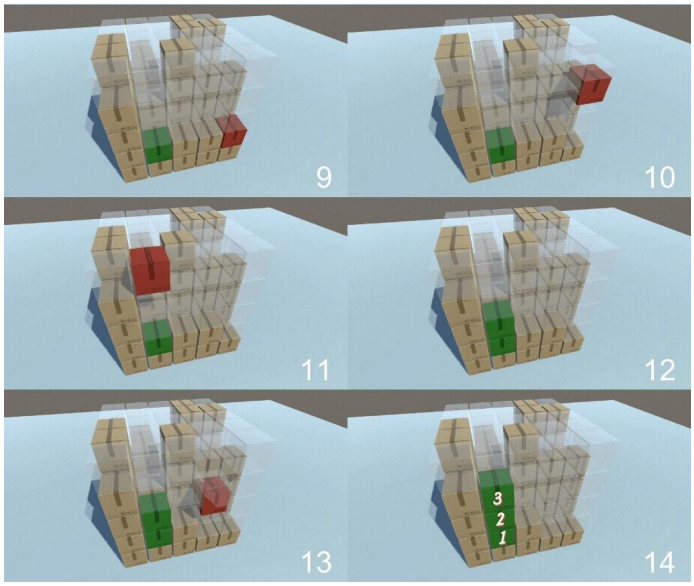


Fig. 23. Sequential movement of boxes after system stabilization.

f) Path Cleaning Optimization for Selected Boxes: When a box initially designated to be moved (red) encounters an obstacle, the repositioning of this obstacle is addressed at the beginning of the subsequent sections. This intervention focuses on efficiency and timing according to the box cleaning rules introduced in Stage 2 and is handled within the same algorithm. It is important to note that this prioritization applies only to the selected boxes, which are initially commanded to move from one place to another, and does not include the boxes causing the obstacles. Including the obstructing boxes in this rule would make the situation overly complex and is avoided, considering that these boxes are more likely to find alternative paths, thus causing less disruption.

Additionally, if the path of the selected box is altered according to Stage 1 strategies to bypass the now-obstructed area, the repositioning of the obstacle becomes unnecessary and can be canceled in future sections. This change reflects a dynamic adjustment to ensure efficient path cleaning without unnecessary interventions.

It should be noted that while the logic is similar, this is slightly different from the local box cleaning mentioned in the previous sections of Stage 2. Local box cleaning aimed to clear the boxes on top of the selected boxes. Here, the strategy focuses on clearing significant obstacles encountered by the selected boxes on their paths.

Providing overall feedback includes evaluating which path was chosen in Stage 1 and how the total efficiency outcome can be maximized in Stages 1 and 2, which was not significant in local box cleaning.

g) Implementation: Below is an example detailing how the agent handles conditions in sections using the described concept.

Section 1: The box encounters several obstacles while progressing to its target, and it becomes clear that it is impossible for the box to enter the area without clearing these obstacles.

The agent takes one step to pick up the box and six steps to reposition the other boxes, making a total of seven side steps.

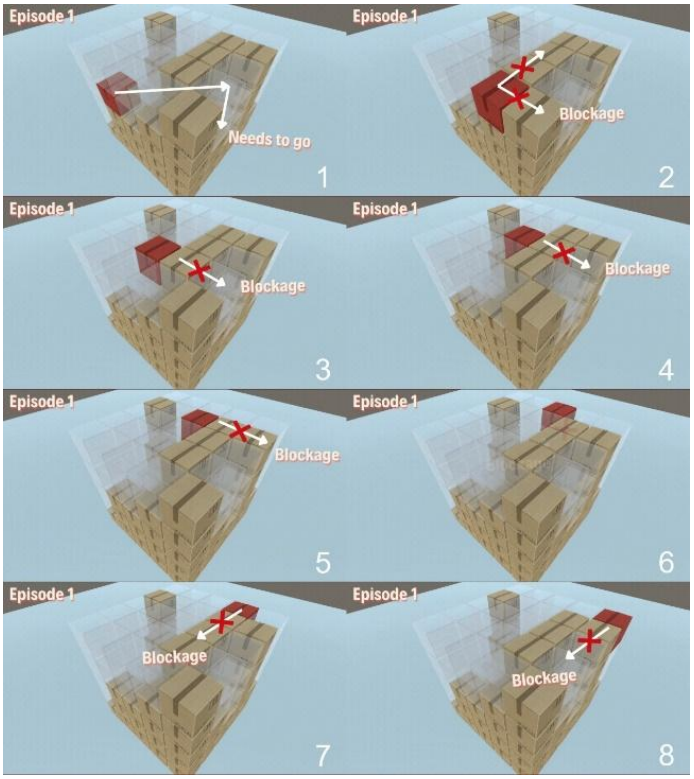


Fig. 24. Section 1: Multiple Obstacles.

Section 2: At the beginning of the section, a box perceived as an obstacle in Section 1 is repositioned based on the box cleaning mechanics. From this point onward, the path for the box to reach its target is cleared. The agent takes one step to pick up the box and six steps to reposition the other boxes, making a total of seven side steps.

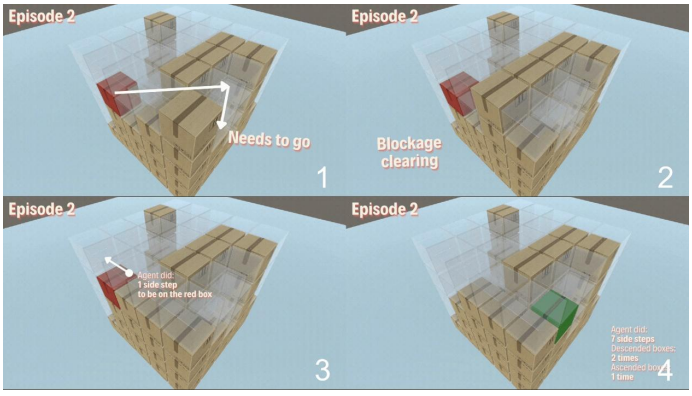


Fig. 25. Section 2: Repositioning of Obstacles.

Section 3: A more efficient method for clearing the path of the selected box is discovered. Instead of two side steps and one downward step in Section 2, another obstructing box is repositioned, requiring only one side step and one downward step. This pattern aligns well with the goals of Stage 1 and makes this method the default cleaning approach. It shows that options can be evaluated when multiple boxes obstruct the path. The total number of steps taken by the agent is reflected inversely as a reward for the repositioned box, and the method with the highest reward is selected. The agent takes two steps to pick up the box and five steps to reposition the other boxes, making a total of seven side steps.

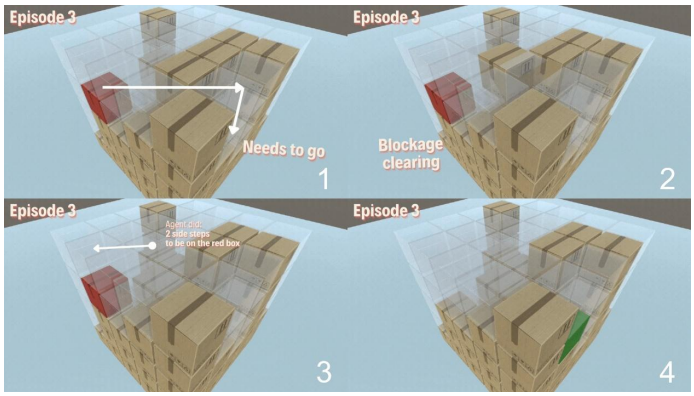


Fig. 26. Section 3: Increased Efficiency in Repositioning.

Note on Agent's Repositioning Time: The focus is not on the agent's repositioning time (collection time) on the boxes, as the command sequences in Stage 3 will be executed simultaneously by multiple agents, rendering this time gap irrelevant. Instead, our algorithm now prioritizes reducing the steps an agent needs to take during delivery using Stage 2's box cleaning methods to achieve more efficient results in scenarios with multiple agents. Therefore, as the steps for repositioning boxes reduce from six to five, the third section will be more advantageous.

C. Swarm Intelligence Design and Implementation

In the third stage of the project, it was decided to design and implement swarm intelligence to make the system more effective. Each agent is considered as an individual. Given that a single agent might be insufficient to execute commands on a large map, the use of multiple agents was proposed to solve this issue. Multiple agents working simultaneously will not only increase efficiency but also manage a larger grid area.

The coordinated work of multiple agents allows each agent to focus on a specific task or area, significantly reducing overall processing time. Swarm intelligence enhances system flexibility and adaptability by enabling agents to interact, share information, and dynamically adjust to tasks.

Moreover, the swarm intelligence approach ensures that if any agent fails, other agents can step in to continue the task, significantly increasing system reliability and resilience for uninterrupted operation. Task sharing and collaboration among agents reduce the workload on each agent, creating a more efficient and effective working environment.

In conclusion, the design and implementation of swarm intelligence will provide a more effective material handling and

placement process in large and complex maps, significantly enhancing overall system performance and efficiency.

1) Design: This section will detail the design of the swarm intelligence system, focusing on key aspects such as agent interactions and learning to work together in a simulation environment. Topics like agent placement, movement strategies, and interaction mechanisms will be discussed. Additionally, the application of the swarm intelligence approach and its effects on system performance will be examined.

a) Agent Characteristics: Various features of the agents have been designed to enhance the overall efficiency and effectiveness of the system. These features maximize the individual capabilities of the agents while ensuring they work in harmony with the global brain:

- **Field of Vision**: One of the most important components of this project is learning to work within a limited area of a certain size. This micro-learning approach allows individuals to gain detailed and focused information. At the same time, the central control unit, defined as the global brain, directs individuals to specific tasks and locations. This structure combines a detailed learning model at the local level with a guiding plan at the global level, ensuring tasks are effectively completed and the learning process is optimized. Without this system, a complex learning model covering the entire warehouse would be required. Such a broad model would make the learning process nearly impossible due to the warehouse's size. However, the current structure enhances efficiency by providing both detailed training at the micro-level and general guidance at the macro-level. Thus, individuals develop the skills to work in small areas, while the global brain directs overall planning, overcoming complexity. This two-tiered approach makes the learning process more manageable and effective.
- **Transfer of Experience Among Individuals**: All agents have access to the Q-table and Q-trees obtained from stages 1 and 2, allowing them to effectively move boxes in any environment previously learned by an agent. This central database enables each agent to act based on previously acquired knowledge, allowing them to learn not only from their own experiences but also from the collective experiences of the entire system.
- **Mobility and Tasks**: Agents can move freely by carrying boxes or being directed to specific areas by the global brain. This mobility allows agents to dynamically change their field of vision. The ability to switch between tasks increases the versatility of agents and supports the overall flexibility of the system.
- **Limited Working Areas**: The global brain can define limited areas to ensure that agents do not leave their designated working regions. This is critical when work intensity is evenly distributed across the plane. Limited working areas reduce the likelihood of agents overlapping and allow them to work more efficiently by concentrating in specific regions.
- **Dynamic Allocation**: The global brain can call the nearest agents to the working area based on the task's difficulty or complexity. This feature ensures optimal use of resources. Agents quickly engage in the required locations, enhancing overall system efficiency and speeding up task completion.

The characteristics of the agents, coordinated with the global brain, optimize the system's overall performance. The global brain, as a central control mechanism, directs and intervenes in the agents' activities as needed. This distinction allows agents to make independent and quick decisions (initiative) at the local level, while the global brain can strategize and plan at the global level. Thus, the system operates effectively at both local and global levels.

Figure 27 shows how the global brain coordinates the nearest agents to the working area. The fields of vision of the called agents are visualized in orange and purple, transparently.

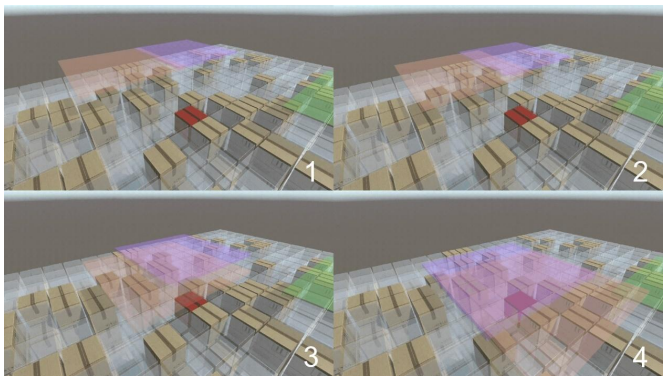


Fig. 27. Global brain coordinating agents.

b) Agent Visualization: Colored planes were used in the simulation environment to visualize the field of vision of the agents over the boxes. During the transport process, the opacity of the agents carrying the boxes was reduced to provide clearer visuals. Additionally, the simulation area was configured as 20x10x20 to better observe activities on the path, design upon them, and simulate. The number and locations of the agents were randomly initialized on the grid system to realistically reflect real-life scenarios. Figure 28 shows the visualizations of the agents and the map.

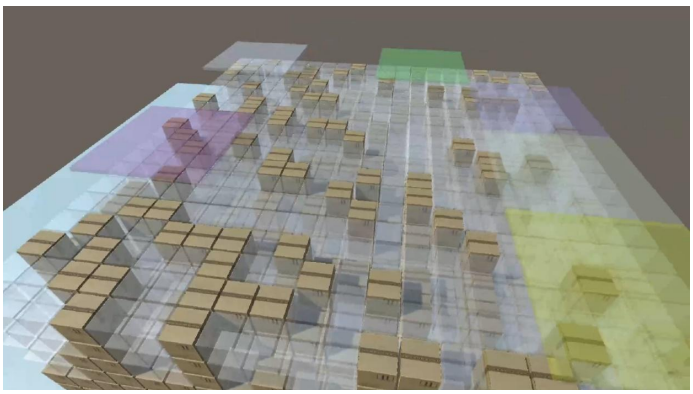


Fig. 28. Agent and map visualization.

c) Area Constraints: The global brain can impose restrictions on the movement areas of agents when necessary. These constraints are crucial for balancing work intensity in specific regions and preventing agent collisions. There are several important reasons why agents need such constraints:

- **Effective Area Utilization**: Limited working areas allow agents to focus on specific regions, enabling each agent to work more efficiently in its area. This reduces collisions and unnecessary movements, thereby increasing overall efficiency.
- **Coordinated Work**: Agents working in specific areas can be better coordinated by the global brain. The global brain can monitor the movements of each agent, intervene when necessary, and utilize resources most efficiently.
- **Task Sharing**: Agents working within limited areas can share tasks and collaborate to complete more complex tasks faster. This allows agents to help each other and balance the workload.

This distinction between the global brain and agents is designed to optimize the overall performance of the system. The global brain, as a central control mechanism, strategizes and plans at a general level, while agents can make independent and quick decisions at the local level. This structure creates a flexible and effective system by leveraging the strengths of both the global brain and the agents.

In the scenario shown in Figure 29, the global brain has imposed movement area constraints on the agents. As a result, one agent had to transfer its box to another agent during the task. This situation demonstrates a dynamic and coordinated approach to managing spatial constraints within the operational environment.

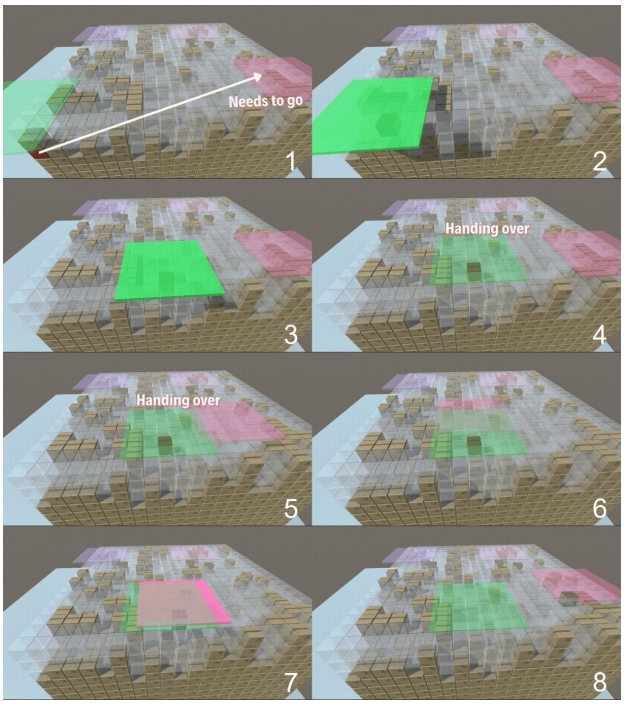


Fig. 29. Agent task transfer due to area constraints.

d) Decision Model for Calling Agents using Q-Learning: A new Q-learning algorithm has been designed to dynamically determine the optimal number of agents to be called by the global brain to perform a task. The decision-making process is driven by various factors reflecting the complexity of operations and the need for efficient resource allocation.

Q-table Structure Used: Similar to the database used in the first stage of the project, a Q-table is used in this model as well. The specific features of the table at this stage are summarized as follows:

- **States**: Each state in the Q-table represents the 3D grid position of the box that needs to be moved, which is at the center of the task. This allows the global brain to evaluate the geometric requirements and constraints specific to the location of each task.
- **Actions**: Each action reflects a real number value indicating the "agent requirement". This number represents the optimal number of agents the global brain deems necessary for optimal task performance. The rounded value of this number determines how many agents will be called. For example, if this value is 1.6, 2 agents will be called.

e) Learning and Reward Mechanisms: During the learning process, the agent requirement values for tasks are adjusted based on the outcomes and efficiency of completed tasks using the following reward structure:

- **Penalty for Ineffective Agent Utilization**: When agents are called to work but are not used effectively, i.e., if resources are overestimated, a dynamic negative reward (penalty) value is set. This adjustment is inversely proportional to the ratio of the agent's working time to the total agent working time at that job site. The penalty value (r_i) is normalized to a range between -2 and 0. Mathematically, it is expressed as:

$$r_i = -2 \times \left(1 - \frac{\text{Agent's Working Cycles}}{\text{Total Working Cycles}}\right) \tag{18}$$

For example, if other agents worked for a total of 10 cycles, and one agent only participated in 2 cycles, the penalty is calculated as:

$$r_i = -2 \times \left(1 - \frac{2}{10}\right) = -2 \times 0.8 = -1.6 \tag{19}$$

Workload Adjustment Based on Intensity: The average workload for each task is calculated by evaluating the contribution of each agent over the time spent. This workload is determined by dividing the total number of boxes moved by the total time spent, averaged across the agents involved in the task. Mathematically, this calculation is expressed as:

$$Y = \frac{1}{n} \sum_{i=1}^{n} \frac{k_i}{t_i} \tag{20}$$

where:

- Y represents the workload,
- n is the number of agents involved in the task,

- k_i is the number of boxes each agent moved,
- t_i is the time each agent spent.

For example, if a total of 12 seconds were spent, with agent 1 moving 3 boxes and agent 2 moving 5 boxes, the average workload is calculated as:

$$Y = \frac{1}{2}\left(\frac{3}{12} + \frac{5}{12}\right) = \frac{1}{2}\left(0.25 + 0.4167\right) = 0.333 \quad (21)$$

This new workload value shows that agents used their time more efficiently. High workload means more tasks are completed in less time, reflecting more efficient task completion.

Adding an Agent: If a third agent is added and this agent moves 2 additional boxes, making a total of 10 boxes moved in 9 seconds by 3 agents, the new average workload is calculated as:

$$Y = \frac{1}{3}\left(\frac{3}{9} + \frac{5}{9} + \frac{2}{9}\right) = 1.1113 \approx 0.37 \quad (22)$$

This workload, compared to previous sections, helps determine efficiency gains or losses. Here, the new workload of 0.37 shows a decrease, indicating that the addition of the third agent decreased efficiency.

Reward Adjustment: Rewards are adjusted based on these workload calculations. The aim is to stay within an optimal workload range of 0.25 to 0.45 to show efficient use of agent time and effort. A positive reward is given if the workload remains within this range. Conversely, if the workload falls outside this range, it indicates inefficient use of resources or overloading, and a negative reward is applied. The reward calculation is given by:

$$r_y = \begin{cases} k \times (x - 0.25) & \text{if } 0.25 \leq x \leq 0.45 \\ -k \times |x - 0.35| & \text{if } x < 0.25 \text{ or } x > 0.45 \end{cases} \quad (23)$$

where k is a coefficient adjusting the reward magnitude, set to 1.

Examples:

- If the average workload is 0.35 (within the optimal range):
$$r_y = k \times (0.35 - 0.25) = 0.10 \quad (24)$$
- If the average workload is 0.20 (outside the optimal range):
$$r_y = -k \times |0.20 - 0.35| = -0.15 \quad (25)$$
- If the average workload is 0.50 (outside the optimal range):
$$r_y = -k \times |0.50 - 0.35| = -0.15 \quad (26)$$

This reward system helps prevent resource waste and overloading under low or high workload conditions. Rewards are kept relatively low and gradually applied, enabling the model to learn more efficient working conditions over time.

Negative Reward for Travel Distance: A negative reward (penalty) proportional to the distance an agent has to travel to reach the work site is assigned. This penalty is directly proportional to the distance d and is calculated using a constant penalty coefficient k. Mathematically, this penalty is expressed as:

$$r_d = -d \times k \quad (27)$$

This penalty discourages agents from moving over unnecessarily long distances, promotes local resource use, and reduces time delays. In our system, k is set to 0.02.

Example: If an agent comes from a distance of 10 units to take on a job, the penalty is:

$$r_d = -10 \times 0.02 = -0.2 \quad (28)$$

This penalty encourages agents to take on jobs from shorter distances, increasing overall system efficiency.

Planned Improvements: Additional considerations, such as whether an agent has pending tasks near its current location, can be included to further refine the decision-making process. Agents with pending tasks nearby can be marked as less suitable for distant tasks, optimizing overall system efficiency.

In conclusion, the total reward (r) in this phase is calculated as shown in Equation 5.5:

$$r = r_i + r_y + r_d \quad (29)$$

f) Agent Self-Assignment Model: After completing the agent calling process, the next critical decision in the workflow is determining which agent will handle which box in the task group at the job site. This decision-making process is carried out by training a local Q-learning model specific to each job site.

Structure of the Local Q-Table: One of the most important components of the project is teaching individuals the ability to work in a limited area of a specific size. This micro-learning approach allows individuals to acquire detailed and focused information. In this context, the local Q-table is a decision-making tool that determines which actions individuals should take in each situation. The Q-table stores reward expectations for each state-action pair and updates these expectations over time, helping individuals learn the most appropriate actions. The use of local Q-tables makes the learning process more manageable and allows individuals to work effectively in small, limited areas. This structure enables larger, more complex systems to be broken down into parts, each of which can be optimized. The states and actions of the local Q-tables are detailed as follows:

- **States**: Each state in the Q-table represents the identities of the boxes to be moved at the job site and their current positions on the 3D grid. Unlike previous stages, where only the positions were kept, the identities remain the same even if their positions on the 3D grid change.
- **Actions**: Actions are defined as the bid values each potential agent offers to move a specific box. These values represent the agents' bids to take responsibility for moving the box, considering their current positions, previous tasks, and other relevant criteria.

Initialization and Updating of Q-Values

- **Initialization**: Initially, the Q-table is set so that the agent closest to each box receives the highest value, as

proximity is the default primary factor in determining task allocation. This default setting continues until overridden by values learned from actual task execution results.

- **Updates through Learning**: Each agent's bid for a box is dynamically updated based on the results of task execution, balancing the discovery of new strategies with the use of known efficient strategies. Q-values are adjusted at the end of each session based on various reinforcement signals.

Reinforcement Criteria After learning the task rewards, the agent with the highest Q value for each turn takes the task because it has made the highest bid.

- **Proximity Bonus**: A positive reward is applied if a box is closer to an agent, prioritizing the agent with the least effort to start the task. The reward is inversely proportional to the distance (d) of the agent to the box. Mathematically, this reward is expressed as:

$$r_p = k \times \frac{1}{d} \tag{30}$$

For example, if the agent is 2 units away from the box and $k = 1$, it receives a reward of 0.5 units.

- **Recent Task Bonus**: A strong positive reward is given if the agent is working on a box it has recently moved, encouraging agents to continue the tasks they started. The reward is proportional to the time (t) the agent has spent moving the box, with $k = 1.5$. Mathematically, this reward is expressed as:

$$r_l = t \times k \tag{31}$$

For example, if the agent worked on the same box in the last 1 cycle, it receives a reward of 1.5 units.

- **Column Constraint**: A negative reward is given if the agent works on a box within the same column it has recently worked on, promoting spatial diversity in task assignments and preventing unnecessary movements. The reward is proportional to the time (t) the agent has spent moving the box, with $k = 1.1$. Mathematically, this penalty is expressed as:

$$r_k = -t \times k \tag{32}$$

For example, if the agent works in the same column, it receives a penalty of 1.1 units.

- **Constraint Penalty**: A strong negative reward is applied if the agent works on a box outside its movement area. If an agent tries to move a box outside the area restricted by the global brain, it receives a penalty of -12 points. Mathematically, this penalty is expressed as:

$$r_r = -12 \tag{33}$$

- **Idle Reward**: A positive reward is given for each box if an agent has been idle for a long time. The reward is proportional to the time (t) the agent has been idle, with $k = 0.4$. Mathematically, this reward is expressed as:

$$r_i = t \times k \tag{34}$$

For example, if the agent has been idle for 2 cycles, it receives a reward of 0.8 units for each box it moves.

Based on this new reward and penalty system, the total reward (r) for a step is expressed as:

$$r = r_p + r_l + r_k + r_r + r_i \tag{35}$$

2) Applications: This section examines the applications where the global brain decides which agents will come to the job site and how the agents arriving at the job site distribute the tasks among themselves. These applications detail the coordinated work of the global brain and agents and the effective distribution of tasks.

a) Application 1: This example examines how agents and the global brain work over three sections.

Section 1: The global brain decides to call a green agent because no agent is directly above the selected box. This single agent is responsible for cleaning the two boxes above the box to be moved and then moving the selected box to its destination since there are no other agents. Assuming each movement takes 1 second, the agent spends 11 seconds cleaning the boxes and 2 seconds taking the boxes.

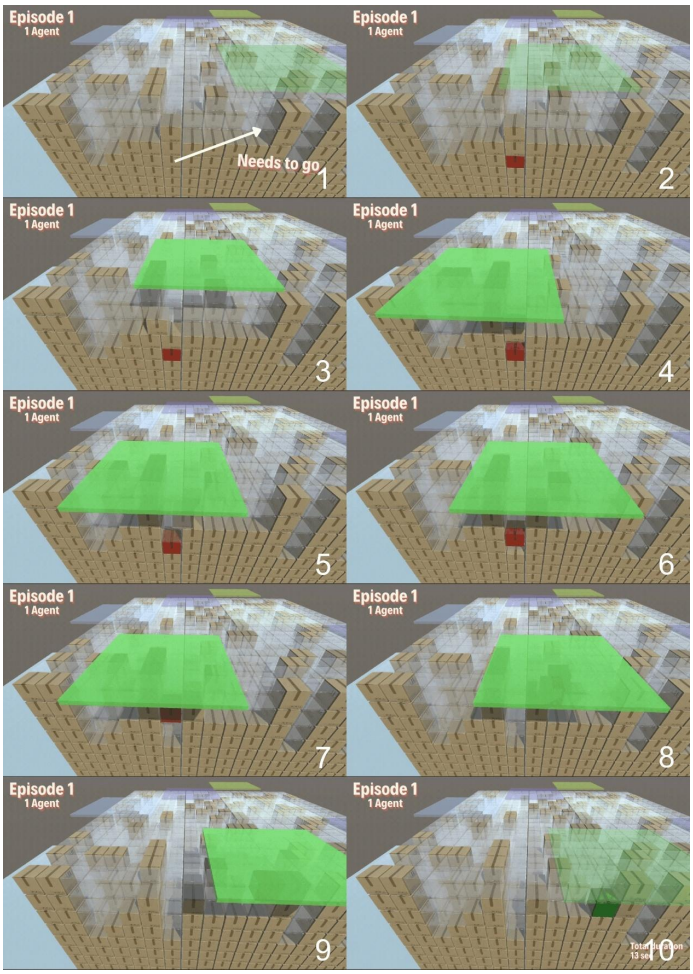


Fig. 30. Section 1: Single agent operation

Section 2: The global brain decides that 2 agents (blue and green) could be more efficient for this task. After the agents are called, in local decisions, the blue agent cleans the boxes while the green agent moves the selected box. This coordination causes the agents to spend 12 seconds moving the box and 1 second approaching the boxes. Due to the difference in box cleaning strategy, there is a loss of 1 second compared to the 11 seconds spent in the first section. Thus, the total task time is 12 seconds.

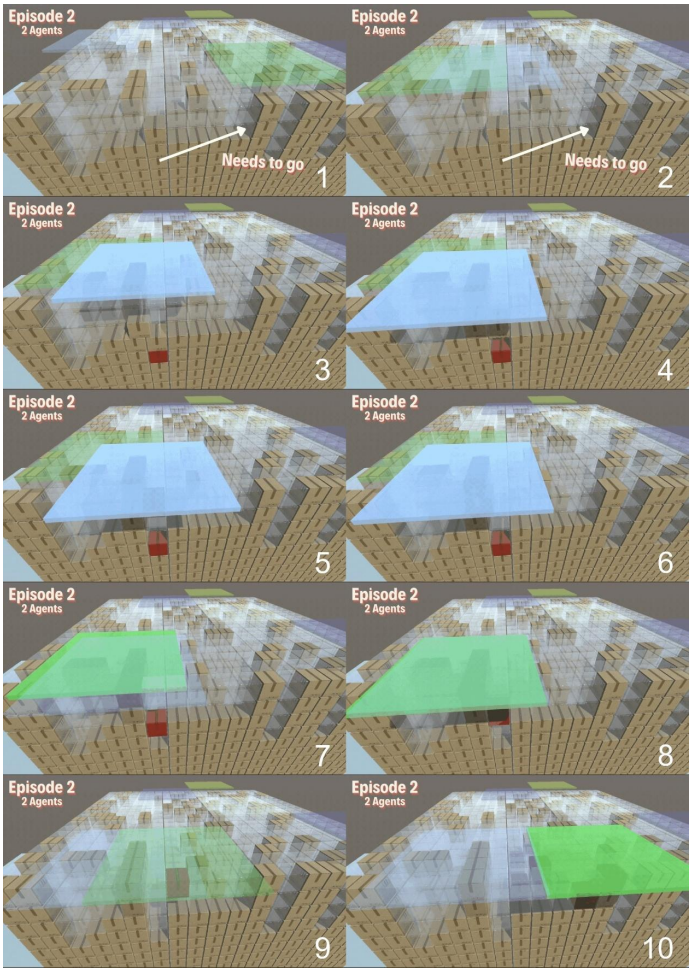


Fig. 31. Section 2: Two agents' cooperation

Section 3: After the agents are called, it is decided to execute operations simultaneously with 2 agents so that one does not wait while the other works because the penalty for carrying in the same column has been activated. Each box above the red box is carried by different agents, and then the blue agent carries the red box. This synchronization eliminates the travel time to pick up the boxes, reducing the total task time to 11 seconds, proving it more efficient than the previous learning step.

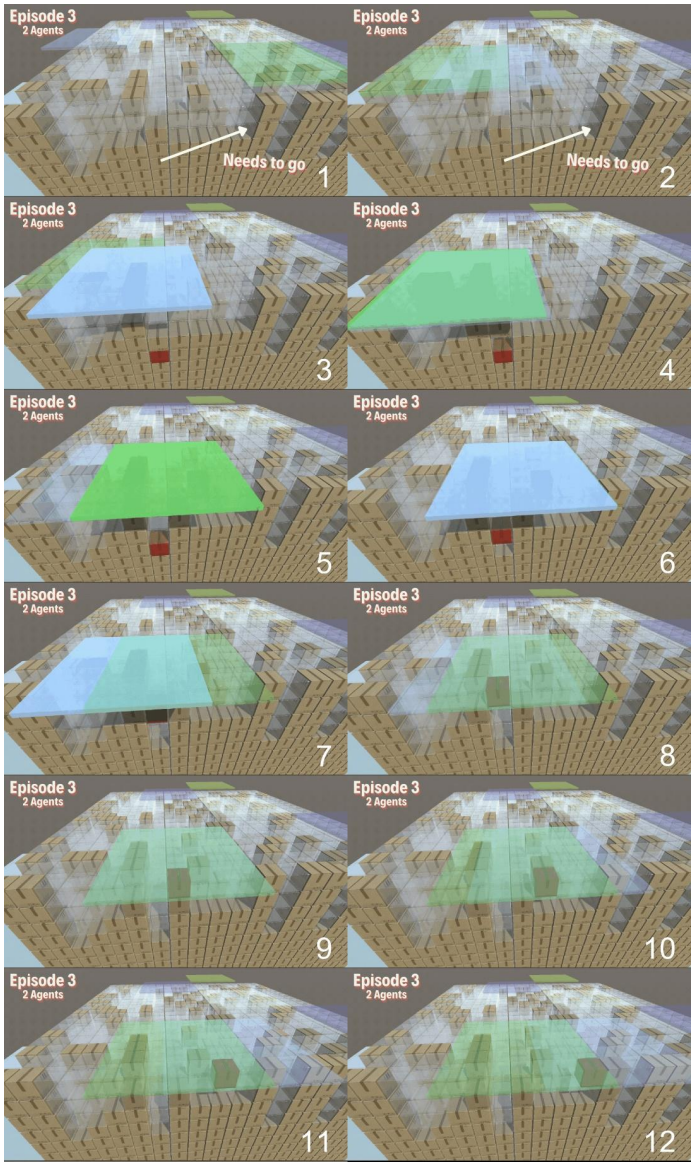


Fig. 32. Section 3: Fully synchronized operation

b) Application 2: Two selected boxes are to be moved sequentially to their targets. In the figure, the coordination after the movements have settled at the end of the training process is seen. The view area of the agents is represented as planes that can be opaque or transparent on the grid structure, with opacity indicating active participation. First, the global brain assigns agents 1 and 2 (dark green and light green) for box 1. Together, they sequentially clean the boxes above box 1. Then, agent 2 moves the red box to its destination. For the task of box 2, agents 1 and 3 (light green and pink) are directed to the job site by the global brain in the same way. After agent 1 cleans the only box above box 2, agent 3 moves the box to its desired target. Thus, we observe the models trained to provide the uninterrupted and effective movement chains we want in operation.

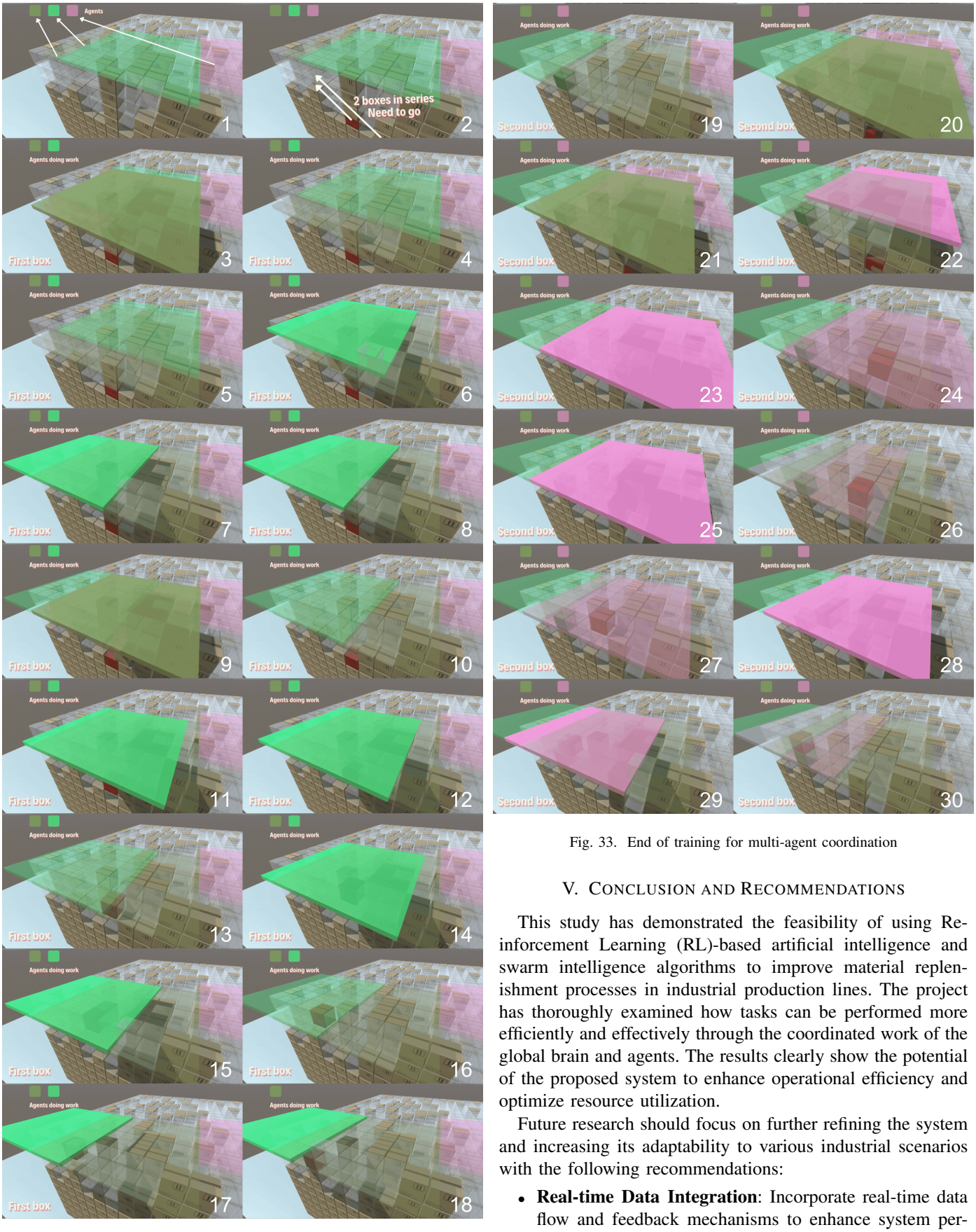


Fig. 33. End of training for multi-agent coordination

V. Conclusion and Recommendations

This study has demonstrated the feasibility of using Reinforcement Learning (RL)-based artificial intelligence and swarm intelligence algorithms to improve material replenishment processes in industrial production lines. The project has thoroughly examined how tasks can be performed more efficiently and effectively through the coordinated work of the global brain and agents. The results clearly show the potential of the proposed system to enhance operational efficiency and optimize resource utilization.

Future research should focus on further refining the system and increasing its adaptability to various industrial scenarios with the following recommendations:

- **Real-time Data Integration**: Incorporate real-time data flow and feedback mechanisms to enhance system performance.

- **Advanced Algorithm Optimization**: Employ more complex task optimizations and innovative learning techniques for algorithms.
- **Physical Robot Applications**: Implement and test the successes achieved in simulation environments on real-world robotic systems to demonstrate the system's practical effectiveness.
- **Comprehensive Scenario Testing**: Evaluate and optimize system performance across different industrial scenarios to increase overall compatibility and flexibility.

A. Application Areas of the Study

The findings of this study can provide significant benefits across many industrial application areas. Specifically, in sectors where material management is critical, such as automotive, electronics, and consumer goods production, the deployment of the proposed system can offer substantial advantages in terms of efficiency and cost savings. The system's applicability is also highly relevant to warehouse management and logistics.

The proposed AI and swarm intelligence-based system can be applied not only in production lines but also in large-scale logistics networks and complex material handling processes. This can enable more efficient management of material flow and improvement of operational processes.

In conclusion, this study underscores the potential of artificial intelligence and robotic technologies in industrial applications and provides a solid foundation for future research. It is imperative to evaluate the proposed system in a broader range of applications and continually develop it.

ACKNOWLEDGMENT

I would like to express my gratitude to everyone who contributed to the preparation of this thesis and supported me throughout the process. My heartfelt thanks go to my advisor, Prof. Dr. Ahmet Onat, for his guidance and for sharing his extensive knowledge with me during the course of this thesis. I am also deeply grateful to my research colleagues who assisted me during my thesis work, and to my family for their constant encouragement and motivation.

Special thanks to Istanbul Technical University for their financial and moral support in the realization of this thesis.

I would also like to extend my sincere thanks to my colleagues and mentors at Siemens AG Turkey, Alpaslan Yıldız and Mustafa Tiftikçi, for their continuous moral support and motivation during the project, and for sharing their knowledge and experience in robotics and mechatronics with me.

This thesis represents a significant step towards enhancing efficiency in industrial production processes and reshaping material replenishment operations. I am immensely thankful to everyone who contributed and supported this work.

REFERENCES

[1] D. Shi, W. Fan, Y. Xiao, T. Lin, and C. Xing, "Intelligent scheduling of discrete automated production line via deep reinforcement learning," International Journal of Production Research, vol. 58, pp. 3362–3380, 2020.

[2] R. Kozlica, G. Schafer, S. Hirlaender, and S. Wegenkittl, "A Modular Test Bed for Reinforcement Learning Incorporation into Industrial Applications," ArXiv, 2023.

[3] Z. Viharos and R. Jakab, "Reinforcement Learning for Statistical Process Control in Manufacturing," Measurement, vol. 182, p. 109616, 2021.

[4] R. Nian, J. Liu, and B. Huang, "A review on reinforcement learning: Introduction and applications in industrial process control," Comput. Chem. Eng., vol. 139, p. 106886, 2020.

[5] M. Panzer and B. Bender, "Deep reinforcement learning in production systems: a systematic literature review," International Journal of Production Research, vol. 60, pp. 4316–4341, 2021.

[6] A. Kuhnle, M. May, L. Schäfer, and G. Lanza, "Explainable reinforcement learning in production control of job shop manufacturing system," International Journal of Production Research, vol. 60, pp. 5812–5834, 2021.

[7] S. Chu, H.-C. Huang, J. Roddick, and J.-S. Pan, "Overview of Algorithms for Swarm Intelligence," in Proceedings of the International Conference on Swarm Intelligence, Springer, pp. 28–41, 2011.

[8] W. Li and W. Shen, "Swarm behavior control of mobile multi-robots with wireless sensor networks," Journal of Network and Computer Applications, vol. 34, no. 4, pp. 1398–1407, 2011.

[9] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, "Swarm Robotic Behaviors and Current Applications," Frontiers in Robotics and AI, vol. 7, p. 36, 2020.

[10] R. Wanka, "Swarm intelligence," it - Information Technology, vol. 61, no. 3, pp. 157–158, 2019.

[11] B. Yu and G. Chen, "Swarm intelligence in mechanical engineering," Advances in Mechanical Engineering, vol. 8, 2016.

[12] D. Li, P. Wei, C. Zhao, S. Yang, and Y. Li, "A Mobile Manipulation System for Automated Replenishment in the Field of Unmanned Retail," in 2023 IEEE International Conference on Mechatronics and Automation (ICMA), pp. 644–649, 2023.

[13] Y. H. Alnema, A. K. Musaa, M. N. Ali, and A. O. Muhammad, "Design and Simulation of an Automated Production Plant and Warehouse Management System," in 2023 International Conference on Engineering, Science and Advanced Technology (ICESAT), pp. 101–107, 2023.

[14] M. Hinrichs, V. Grabmaier, I. Stahl, and S. Schneegass, "Designing a Recommendation System for Spare Parts Replenishment," in Proceedings of the 22nd International Conference on Mobile and Ubiquitous Multimedia, 2023.

[15] P. Piliptchak, M. Aksu, F. Proctor, and J. Michaloski, "Physics-Based Simulation of Agile Robotic Systems," in Volume 2B: Advanced Manufacturing, 2019.

[16] M. Aksu, J. Michaloski, and F. Proctor, "Virtual Experimental Investigation for Industrial Robotics in Gazebo Environment," in Volume 2: Advanced Manufacturing, 2018.

[17] E. Peterson, B. Bogosian, and S. Vassigh, "Evaluating an Immersive Learning Environment for Robotics Training," in Training, Education, and Learning Sciences, 2022.

[18] C.J.C.H. Watkins and P. Dayan, "Q-Learning," Machine Learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[19] R.S. Sutton and A.G. Barto, "Reinforcement Learning: An Introduction," MIT Press, 2018.

[20] E. Becer, O. Ulusoy, and K. Turan, "Tasarı: Alışveriş merkezi, havaalanı veya depo gibi geniş kapalı alanlarda çalışacak otonom bir temizlik robotu tasarımı," Master's thesis, İstanbul Teknik Üniversitesi, Makina Fakültesi, 2023.