RoboCup Rescue 2025
TDP Agent Simulation
3Rakshak (India)

Varshitha Konireddy, Sahaya Shaanal Veda Rokkash, Renu Sree Vyshnavi Bakka, Kamalakar Karlapalem, and Lini Thomas

International Institute of Information Technology, Hyderabad, India
sahaya.shaanal@students.iiit.ac.in

Abstract. 3Rakshak presents a comprehensive multi-agent strategy for optimizing civilian rescue and dynamic path clearing in the RoboRescue simulation. Our approach focuses on coordinated civilian extrication and transport by Fire Brigade and Ambulance agents, and on intelligent, impact-driven path clearance by Police agents. We introduce a tunable priority scoring system for both civilian rescue and blockade clearance, and demonstrate significant improvements in rescue rates, agent utilization, and congestion avoidance compared to conventional heuristics.

1 Introduction

In the RoboRescue simulation, agents have dedicated roles: Fire Brigade evacuates stuck civilians, Ambulance Teams carry civilians to shelters, and Police remove obstructions. Conventional methods tend to use greedy heuristics like nearest-neighbor search [2], shortest unblocked path choice [1], or first-come-first-serve task assignment [3], which optimize immediate rewards like least distance [4] or quickest completion time [5] without regard for system-wide efficiency. Our solution suggests employing simplest techniques, real-time data hash maps enabled with semaphores and advanced formulae to facilitate data-driven decision-making for resource allocation and prioritization. This work explains how individual agents use these approaches to promote improved coordination and efficiency.

2 Modules: Civilian Priority Score for Rescue Target Selection

In our system, the **civilian priority score** is a unified metric used to determine the urgency of rescuing each civilian. This score is now directly used by both Fire Brigade and Ambulance Team agents to select which civilians to prioritize for rescue, ensuring that the most critical cases are addressed first.

The priority score is computed based on the civilian's health points (HP), damage, and distance from the agent, reflecting both survivability and accessibility. The formula is:

$$\text{Survival\_Time} = \frac{\text{Current\_HP}}{\text{Damage}} \tag{1}$$

$$\text{Priority} = \frac{1000}{\text{Survival\_Time}} \times \frac{1}{d+1} \tag{2}$$

where:

- **Survival_Time**: Estimated simulation ticks until civilian death if not rescued
- **Current_HP**: Civilian's current health points
- **Damage**: HP loss in a timestep
- **d**: Euclidean distance in simulation units

This approach ensures that agents focus on civilians who are both in critical condition and reachable in time.

This score is now consistently used by all rescue agents for target selection, resulting in improved rescue rates and more coordinated agent behavior.

Note: This priority score was originally developed for clustering purposes, but its effectiveness has led to its adoption as the primary criterion for agent prioritization.

Purpose: Selecting civilians and agents based on proximity and urgency to reduce redundant efforts and delays in rescue operations.

Available Information: Each agent maintains a distributed knowledge base consisting of:

- Local civilian database (agent-discovered and communicated discoveries)
- Agent position updates (when communication infrastructure permits)
- Partial map knowledge (limited to explored areas only and line of signt)

Civilian Priority Implementation: Agents utilize a dynamic priority score for each civilian, calculated using available health, damage, and distance information. This score enables agents to select and coordinate rescue targets efficiently, ensuring that the most critical and accessible civilians are prioritized for rescue actions. The priority score is updated as new information becomes available, allowing agents to adapt to the evolving environment and maximize overall rescue effectiveness.

3 Modules: Path Planning

Key Funtionalities of the Dynamic Dijkstra Path Planning:

To address the issue of agents getting stuck due to dynamic blockades, we have implemented a **Dynamic Dijkstra Path Planning** module. This module being an extension of the default Dijkstra Algorithm for Path Planning, recalculates paths in real-time, adapting to changes in the environment such as newly discovered blockades or cleared roads.

3.1 Stuck and Loop Detection

To prevent agents from remaining idle due to repeated failed movements, such as those caused by undetected blockades or map inconsistencies, the algorithm maintains a short history of the agent's recent positions. By analyzing this history, the system can detect if the agent is stuck in place or oscillating between a small set of locations (a loop). If such a condition is detected, it is flagged as a "stuck" or "looping" state. This enables the agent to trigger replanning or alternative strategies to escape the situation, rather than wasting cycles on ineffective movement.

3.2 Police Proximity Check for Blockade Clearance

When a blockade is encountered, the algorithm assesses whether any police agents are nearby and capable of clearing the blockade. This is achieved by maintaining an up-to-date record of all police agent positions and calculating their proximity to the blockade, and the record is called `Police Agent Tracker`, updation of which happens in the Police Path Planning. If police agents are within a certain distance threshold, the agent can wait or coordinate for the blockade to be cleared. If no police are available, the agent will attempt to avoid the blockade by replanning its path.

3.3 Edge Manipulation and Replanning

If a blockade cannot be cleared immediately and no police support is available, the algorithm temporarily modifies the navigation graph by removing the blocked edge. This prevents the agent from repeatedly attempting to traverse the same blocked route. The agent then recalculates its path from its current or a previous position, ensuring that the new route avoids the known blockade. Once planning is complete, the navigation graph is restored to its original state to maintain consistency for future planning cycles. This dynamic manipulation of the navigation graph allows the agent to adapt to changing conditions in real time and avoid persistent obstacles.

3.4 Purpose

By combining stuck detection, real-time blockade handling, police coordination, and adaptive replanning, the Dynamic Dijkstra Path Planning module provides robust and flexible navigation for agents in complex, dynamic environments. This significantly reduces agent idle time, prevents deadlocks, and improves overall mission efficiency.

Key Funtionalities of the Police Blockade Path Planning:

The `PoliceBlockadePathPlanning` module is designed to enable police agents to efficiently plan and execute paths to the nearest blockades within the environment. Its primary goal is to ensure that police agents can quickly identify and

reach blockades, facilitating the clearing of critical routes for all rescue teams and improving overall mission flow. This module is particularly suited for integration with actions such as `ExtActionClear`, which require precise navigation to the location of a blockade.

3.5 Available Information

The module leverages several sources of information:

- **Police Agent Positions:** Continuously updated using the `PoliceAgentTracker`, which maintains a mapping of each police agent to its current position.
- **Blockade Locations:** A collection of blockade entity IDs provided as potential destinations.
- **Path Planning State:** The last calculated path and the last targeted blockade are stored for reference and further decision-making.

3.6 Implementation Overview

The module extends a standard Dijkstra-based path planning algorithm, specializing it for police agents. At each planning cycle, the following steps are performed:

1. **Police Position Tracking:** The positions of all police agents are updated in the `PoliceAgentTracker`. This ensures that the system has an accurate, real-time view of agent distribution, which can be used for coordination and redundancy avoidance.
2. **Destination Setting:** The set of known blockades is provided as possible destinations. The planner is configured to always select the nearest blockade as the target.
3. **Path Calculation:** The underlying Dijkstra algorithm computes the optimal path from the agent's current position to the selected blockade, ensuring that the blockade itself is included as the final step in the path.
4. **Result Storage and Logging:** The calculated path and the target blockade are stored for later retrieval. Informative logs and console outputs are generated to aid debugging and analysis.

3.7 Usage in the System

This module is invoked whenever a police agent needs to clear a blockade. By always targeting the nearest blockade and maintaining up-to-date knowledge of all police positions, the system can distribute agents efficiently across the map, minimizing overlap and ensuring that blockades are cleared as quickly as possible. The module's results are used directly by the agent's action logic to move and clear blockades.

3.8 Benefits and Impact

- **Efficient Resource Allocation:** By tracking all police positions and always targeting the nearest blockade, the system avoids redundant assignments and ensures that all blockades are addressed promptly.
- **Improved Coordination:** The use of the `PoliceAgentTracker` enables better coordination among police agents, reducing the likelihood of multiple agents converging on the same blockade unnecessarily.
- **Transparency and Debugging:** The module provides detailed logs and console outputs, making it easier to monitor agent behavior and diagnose issues during development and testing.
- **Scalability:** The approach is robust to large maps and many agents, as it dynamically adapts to the current distribution of agents and blockades.

3.9 Purpose

The `PoliceBlockadePathPlanning` module is a specialized, efficient, and well-coordinated solution for police agent navigation in dynamic rescue scenarios. By leveraging real-time agent tracking and optimal path planning, it significantly enhances the system's ability to keep critical routes open and support the broader rescue mission.

4 Modules: Synchronized Civilian Rescue with HashMap and Semaphores

To ensure that only one agent attempts to rescue a civilian at a time, we use a **HashMap** to track civilians currently being rescued, combined with **semaphores** for synchronization. This prevents multiple agents from duplicating rescue efforts and ensures efficient allocation of resources.

Implementation Details:

- The `civilianMap` in `TriRakshakHumanDetector.java` stores the status of each civilian.
- Semaphores are used to lock access when an agent commits to rescuing a civilian.

This mechanism has eliminated race conditions and improved the reliability of rescue operations.

5 Modules: Group Allocation for Multi-Agent Coordination

We have implemented a **group allocation** strategy where agents are assigned to move in coordinated groups, typically consisting of one Police, one Fire, and one Ambulance agent. This ensures that agents can support each other, for example,

Police clearing blockades for Fire and Ambulance, and Fire assisting Ambulance in hazardous areas.

Implementation:

- Group assignments are managed in the target allocator modules.
- Each group is assigned a common target area, and agents synchronize their movement and actions.

This approach has led to a significant reduction in agents being blocked and improved the overall mission success rate.

6 Modules: Agent Spreading in Search

To maximize coverage and avoid redundant searching, we have implemented **agent spreading** in the Search module. Agents are assigned to search different regions, minimizing overlap and ensuring that all areas are explored efficiently.

Implementation:

- The search space is partitioned and each agent is assigned a unique region.
- This is additionally to the original KMeans Clustering performed before the simulation runs. Dynamic reassignment occurs if agents complete their region or if new information indicates a need to rebalance.

This has resulted in faster discovery of civilians and hazards, and more efficient use of agent resources.

7 Strategies

7.1 Police Force Strategy

Purpose: The primary goal of the police force is to clear blockades and maintain open routes throughout the city, enabling fire brigades, ambulance teams, and civilians to move efficiently and safely. By ensuring critical pathways remain accessible, police agents play a foundational role in the overall success of rescue operations.

1. **Information Used:** Police agents operate using a combination of real-time and precomputed information. They continuously monitor the locations and status of all known blockades and roads, leveraging the `PoliceAgentTracker` to keep an updated map of all police agent positions. Additionally, group assignments are established at the start of the simulation, pairing police agents with fire and ambulance teams to maximize area coverage and facilitate coordinated action.
2. **Target Allocation:** At the beginning of each scenario, agents are grouped and assigned to specific sectors or high-priority roads using a group allocation mechanism. This ensures that all critical areas are covered and that agents can support each other's objectives. The allocation process considers both the distribution of agents and the locations of blockades, aiming to minimize overlap and maximize efficiency.

3. **Path Planning:** Each police agent utilizes a specialized path planning module that always selects the nearest blockade as its target if it did not receive an instruction target. The agent computes the shortest path to the blockade and updates this path dynamically as the environment changes—such as when blockades are cleared or new ones appear. This approach ensures rapid response and optimal use of available resources.
4. **Redundancy Avoidance:** To prevent multiple agents from converging on the same blockade, the system tracks and shares the real-time positions of all police agents. Before committing to a target, each agent checks the current assignments and positions of its peers, ensuring that efforts are distributed across all blockades and that no area is neglected or over-served.
5. **Fallback Behavior:** If a police agent finds no blockades within its assigned area or sector, it will automatically seek out the next closest blockade or reposition itself to a strategic location for future tasks. This fallback mechanism ensures that agents remain productive and ready to respond as new obstacles emerge.
6. **Guiding Principle:** The core decision rule for police agents can be summarized as:
$$\text{Target} = \arg \min_{b \in \text{Blockades}} \text{Distance}(\text{Agent}, b)$$
where each agent selects the blockade that is closest to its current position, subject to group assignments and current agent distribution.
7. **Summary:** Through real-time coordination, dynamic path planning, and intelligent target allocation, the police force ensures that the city's transportation network remains functional, directly supporting the rescue and evacuation efforts of all other emergency teams.

7.2 Ambulance Team Strategy

Purpose: To maximize civilian survival and minimize rescue redundancy by ensuring optimal coordination between ambulance agents in selecting, rescuing, and transporting civilians to the most suitable refuges.

1. **Civilian-Centric Coordination and Allocation**
Our system uses a **shared, synchronized hash map** where each civilian is represented by the following attributes:
   - `Civilian_ID`
   - `(X, Y)` coordinates
   - `HP` and `Damage`
   - `Assigned_Agent_ID`

   This map is accessible to all agents and protected by **semaphores** to ensure that only one agent can commit to a given civilian at a time.
   When an agent discovers a civilian, it first checks the hash map:
   - If the civilian is not listed, the agent adds a new entry.
   - If already present, it updates the entry with newer observations.

   This mechanism avoids duplication and promotes efficient task distribution.

2. **Civilian Prioritization Strategy**
   After securing a civilian, the agent evaluates **rescue priority** based on the following order:
   (a) **Survival Time:** Civilians with critically low HP and high damage are prioritized, as they are most likely to die soon.
   (b) **Distance to Refuge:** Civilians closer to accessible refuges are preferred, as they can be transported quickly and are more likely to survive.
   (c) **Distance from Agent:** Among equally urgent civilians, preference is given to those closer to the agent to reduce approach time.

   This strategy enables time-efficient, life-critical decision-making under uncertainty.
3. **Real-Time Synchronization and Updates**
   The civilian hash map is dynamically updated as agents discover or receive new information. This ensures all ambulance agents operate on a consistent and evolving shared view, facilitating decentralized coordination.
4. **Refuge Selection Strategy**
   Once a civilian is selected, the agent evaluates available refuges using a weighted scoring system based on availability, queue, and travel distance:

$$S = w_1 \cdot \frac{\text{Available Beds}}{\text{Total Beds}} + w_2 \cdot \left(1 - \frac{\text{Queue Length}}{\text{Max Queue Capacity}}\right) + w_3 \cdot (1 - \text{Normalized Distance}) \tag{3}$$

   where the weights are $w_1 = 50$, $w_2 = 30$, $w_3 = 20$, and:

$$\text{Normalized Distance} = \frac{\text{Distance to refuge}}{\text{Map diagonal length}} \tag{4}$$

5. **Example**
   Consider an ambulance at $(1000, 1500)$ on a map with diagonal length 5000:

7.3 Fire Brigade Strategy

Purpose: Rescue civilians from buildings and support their evacuation, while coordinating with other agents.

Explicit Prioritization Mechanism:

1. **Target Assessment:** At every simulation tick, the fire brigade agents systematically reassess all known civilians in hazardous areas. Using up-to-date information from the shared civilian map, agents recalculate the priority of each civilian based on their current health, damage, and distance to refuge. This ensures that the most vulnerable but more accessible civilians are always at the top of the rescue list.

2. **Resource Coordination:** To maximize efficiency and avoid redundant rescue attempts, fire brigade agents coordinate with each other and with nearby ambulance teams. This is achieved through the use of synchronized data structures (the hash map with semaphores as shown in section 4), which ensure that once a civilian is assigned to a specific agent, others are prevented from targeting the same individual.

3. **Dynamic Adaptation:** The strategy is highly adaptive: new blockades appear, or civilians' conditions change, agents continuously update their priorities and reassign themselves as needed. This dynamic adaptation is supported by real-time updates to the civilian map and ongoing communication between agents, enabling the fire brigade to respond effectively to the evolving disaster scenario and maximize the number of successful rescues.

This multi-faceted prioritization mechanism ensures that fire brigade agents act with both urgency and coordination, dynamically balancing risk, resource availability, and the changing environment to achieve optimal rescue outcomes.

Application of the Priority Score in Fire Brigade Operations
The priority score previously introduced in Section 2 of this TDP,

$$\text{Priority} = \frac{1000}{\frac{\text{Current\_HP}}{\text{Damage}}} \times \frac{1}{d+1} \tag{5}$$

is directly utilized by the fire brigade agents for civilian rescue decision-making.

Implementation Details: Within the `TriRakshakHumanDetector` module, each civilian's health, damage, and distance from the agent are dynamically tracked and updated. The priority score is computed for all valid civilians, and agents select targets based on the highest calculated priority. This mechanism is synchronized using a hash map and per-civilian semaphores, ensuring that only one agent commits to a civilian at a time and preventing redundant rescue attempts.

This approach ensures that the fire brigade's rescue efforts are both targeted and efficient, maximizing the number of civilians saved under dynamic and hazardous conditions.

Also this approach has led to a measurable reduction in agent deadlocks and improved overall mission efficiency.

8 Results:

The above enhancements have been fully integrated into the `TriRakshak_2025` codebase and have demonstrated clear improvements in simulation results, including higher rescue rates, reduced agent idle time, and better overall coordination.

The following table shows results of our implemented code in comparision to the sample agent code. We collected the `paris` map score at timestep 106 and the `berlin` map score at 200.

Team	**Map**	
	paris	**berlin**
3Rakshak Team	90.7	23.5
Sample-agent	87.9	15.2

References

1. Madkour, A., Aref, W., Rehman, F., Rahman, A., Basalamah, S.: A survey of shortest-path algorithms (05 2017). https://doi.org/10.48550/arXiv.1705.02044
2. Park, Y., Park, S., Lee, S.g., Jung, W.: Greedy filtering: A scalable algorithm for k-nearest neighbor graph construction. In: Bhowmick, S.S., Dyreson, C.E., Jensen, C.S., Lee, M.L., Muliantara, A., Thalheim, B. (eds.) Database Systems for Advanced Applications. pp. 327–341. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-05810-$8_2$2
3. Schwiegelshohn, U., Yahyapour, R.: Analysis of first-come-first-serve parallel job scheduling. Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (06 1998). https://doi.org/10.1145/314613.315031
4. Skulimowski, A.M.: Optimality and sensitivity of least-distance and avoidance solutions in multicriteria optimization. In: 2018 23rd International Conference on Methods Models in Automation Robotics (MMAR). pp. 375–380 (2018). https://doi.org/10.1109/MMAR.2018.8486062
5. Zhang, Y.: IEEE Access **PP**, 1–1 (01 2024). https://doi.org/10.1109/ACCESS.2024.3465628