Appendix for Adversarial Reinforcement Learning for Enhanced Decision-Making of Rescue Robots in Intelligent Fire Scenarios

I. Appendix A: Simulation Environment

Our simulation environment is shown in Figure 1. The simulated building consists of three floors, each containing different levels of architectural structures. The minimum unit of the building is a 1x1x1 cube, and the overhead view size of the main part of each floor is 40x40. Each floor is divided into several rooms by walls, and the rooms are interconnected by ”doors” represented by blue cuboids. The floors are connected by two separate stairwell structures, and the platforms of each floor's stairwell are interconnected with the staircases through ”staircase entrances” represented by purple cuboids. Figure 2 illustrates an overview of the walkable areas in the NavMesh navigation system of the entire building.

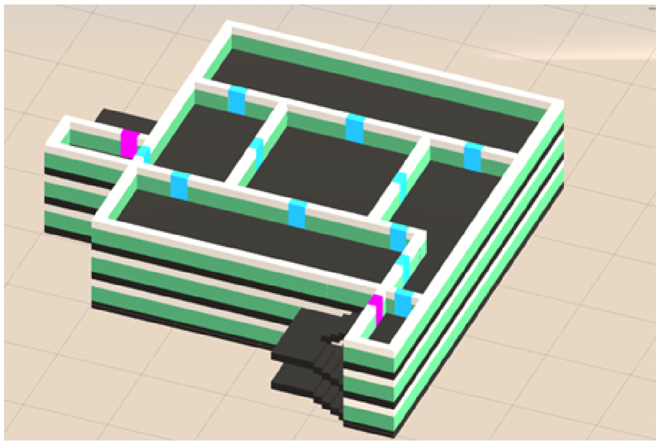

Fig. 1: Overhead view of the virtual environment of the building from an oblique angle.

The door-like structures are responsible for connecting general rooms within each floor plane of the virtual environment. The three architectural elements have no collision volume and do not affect the NavMesh navigation system. Their common role is to mark the connecting passages between small areas in the building, and trigger event functions to produce corresponding interactions when other units in the building come into contact with them. For example, when a human passes through the exit, they will be deregistered from the environment, and flames will destroy regular doors when in contact(see Figure 3).

To simulate multi-story complex buildings, the layout of rooms varies on each floor in the simulated evacuation environment, as illustrated in Figure 4, Figure 5 and Figure 6. The number and complexity of rooms decrease as the number of floors increases.

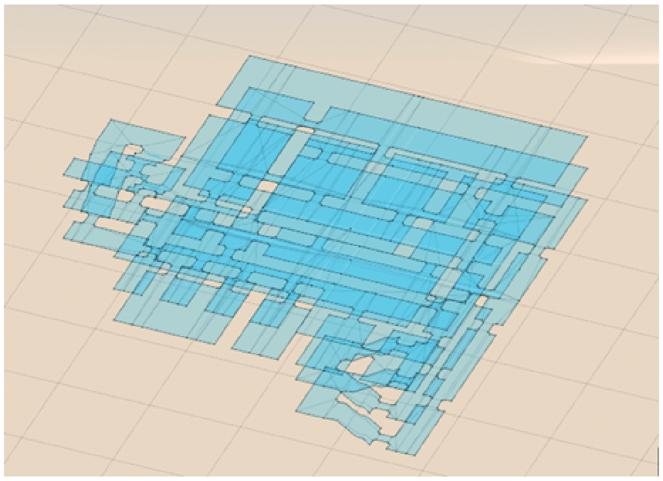

Fig. 2: Overview of the NavMesh navigation system of the building from an oblique overhead view.

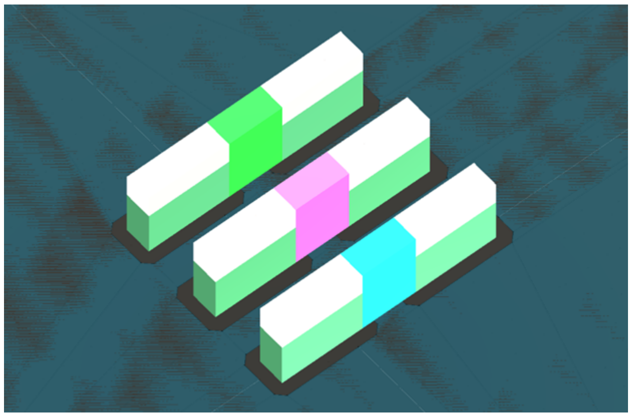

Fig. 3: As shown in Figure, the exit (green), stairwell entrances (purple), and regular doors (blue) are static passable elements in the environment.

II. Appendix B: Details of the Human Behavior Model

Humans use the RayCast method to simulate a real human's field of view, if a ray hits a visible entity, information about that entity is returned. In this way, all architectural elements in the simulation are tagged with corresponding labels, allowing the human agents to judge the entity type in front based on the information returned by RayCast. Meanwhile, to simulate the low visibility in fire scenarios, each human's ray detection distance is limited to a certain range. Directions without ray feedback are marked by the human as currently unexplored unknown directions. Humans

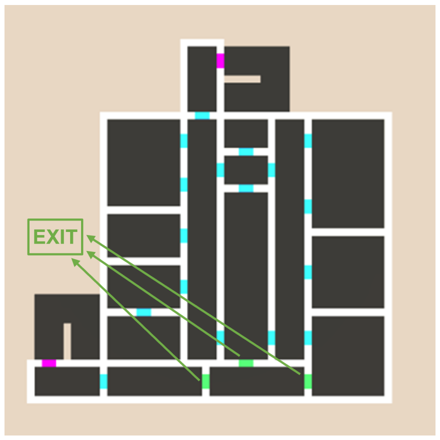

Fig. 4: Top view of the **FIRST** floor environment

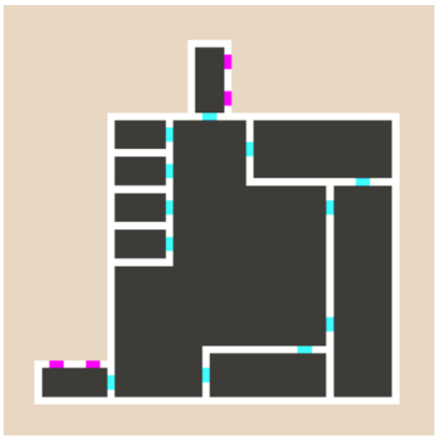

Fig. 5: Top view of the **SECOND** floor environment

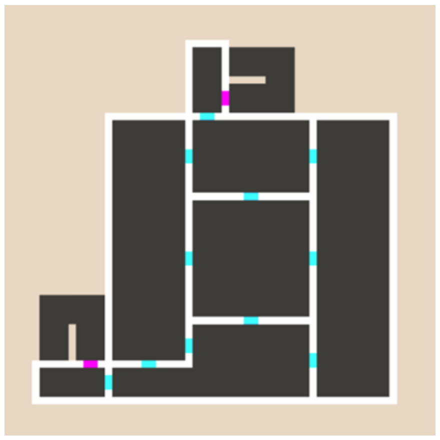

Fig. 6: Top view of the **THIRD** floor environment

Algorithm 1: Human follower

```
   Input: input parameters observe
   Output: output setoff
 1 while not at exit do
 2     if not stuck and no leader then
 3         switch mode
 4     end
 5     if leader within line of sight then
 6         move towards leader;
 7         if leader is a robot then
 8             act according to robot's instructions
 9         end
10     else
11         disconnect and switch mode
12     end
13 end
```

decide their next action based on the detection results from these rays.

Our human behavior model mainly has two modes, leader mode and follower mode. Under certain conditions, the two modes switch to achieve the best simulation effect. Algorithm 1 shows the behavior logic for the human follower, while Algorithm 2 represents the behavior logic for the human leader.

III. Appendix C: Additional Details on Evacuation Robots

The reward function for our evacuation robots is presented in the main text. This section mainly describes the observation inputs and action space for the evacuation robot agents.

A. Observations for Evacuation Robots

Observations represent the current state of the environment and serve as inputs to the agent. Observation values can include various data types, such as visual images, numeric values, or discrete categories, depending on the nature of the environment. In this work, agents on each floor have the same observations to maximize sharing of information within the environment. Evacuation robot agents have two sources of observations: vector observations and grid sensor observations.

Vector observations are the traditional observation method for reinforcement learning agents, with each corresponding observation value fed directly as input to the agent's network. The vector observations implemented for our evacuation robots include: 3D position coordinates of all humans in the environment, 3D position coordinates of all evacuation robots in the environment, the number of robots in the evacuation team on this floor, the elapsed time since evacuation began, and 3D position coordinates of all fire locations as well as the specific number of fires on each floor. This forms a 167-dimensional vector observation.

Grid sensor observations are a new observation method for agents added recently in Unity. Grid sensors divide a specified planar area within a certain range into grids of a specified size, and detect whether each grid contains a specified type of entity. In this work, each evacuation robot agent utilizes a 40x40 grid sensor to observe walls, fires, and exits on the three floors. The grid sensor encodes all static elements to be observed into an image format as input to a convolutional neural network for feature extraction. Compared to vector observations, grid sensors greatly reduce the complexity of the reinforcement learning environment at the cost of some loss of precision. Our overall observation method combining dynamic and static elements helps the evacuation agents better understand the environmental state.

B. Action Space

The action space of a reinforcement learning agent refers to the set of actions the agent can choose from, defining the different actions it can take at each time step. For our evacuation task, each step's action decision for the evacuation robots contains just two float values, representing the horizontal position coordinates (X_r, Y_r) of their next destination. However, since the north and south stairwells are located

Algorithm 2: Human leader

```
Input: input parameters observe
Output: output setoff
while not at exit do
    if Exit found in view then
        Go to the exit
    end
    if There are doors or stairs in view that desired to go then
        if There are suitable leaders in the view then
            switch mode
        else
            Go to the desired stair or door;
        end
    else
        Look for doors and stairs in view;
        if All doors in view have been or do not exist then
            if There are unexplored directions in the view then
                Move in the unexplored direction;
            else
                Go to the door that passed by last time;
            end
        else
            Go to the desired stair or door;
        end
    end
end
```

outside the main building body, there is some correction between the robot's actual move target and decision result:

$$\left(X_r^{'}, Y_r^{'}\right) = \begin{cases} (X_a, Y_a) & ||(X_r, Y_r) - (X_a, Y_a)|| < 20 \\ (X_b, Y_b) & ||(X_r, Y_r) - (X_b, Y_b)|| < 20 \\ (X_r, Y_r) & others \end{cases} \quad (1)$$

Where (X_a, Y_a) and (X_b, Y_b) represent the center positions of the south and north stairwells on this floor respectively, and $(X_r^{'}, Y_r^{'})$ represents the actual target position of the evacuation robot. In other words, when the evacuation robot sets its destination near a stairwell, it will automatically move inside the stairwell.

IV. Appendix D: Additional Details of Fire Source Agent

The fire source agent is our adversarial opponent, mainly used to increase the difficulty of the experiments.

A. Observations for Fire Source Agent

The observations of the fire source agent are similar to the evacuation robots, including coordinates of all humans and evacuation robots, as well as grid sensors observing the status of each floor of the building.

Algorithm 3: Robot algorithm

```
Input: input parameters observe
Output: output destination
Start;
while There are humans not evacuated in this level do
    Evacuate humans;
    if Model or Algorithm makes a decision then
        Update destination based on the decision;
        Modify instructions based on the destination;
    end
    Move towards the current destination;
end
```

B. Action Space

The action space of the fire source generator consists of one discrete action and two continuous actions. The discrete action has three branch options, representing the selection of which floor to generate fire on. The continuous actions are the same as the evacuation robots, where X represents the horizontal coordinate of the fire source generation position. Due to environmental reasons, the actual generation position of the fire source needs to be adjusted to the center of the grid closest to the decision:

$$X_f' = \begin{cases} \lfloor X_f \rfloor - 0.5 & ||\lfloor X_f \rfloor - X_f + 0.5| > |\lfloor X_f \rfloor - X_f - 0.5| \\ \lfloor X_f \rfloor + 0.5 & ||\lfloor X_f \rfloor - X_f + 0.5| \leq |\lfloor X_f \rfloor - X_f - 0.5| \end{cases} \quad (2)$$

$$Y_f' = \begin{cases} \lfloor Y_f \rfloor - 0.5 & ||\lfloor Y_f \rfloor - X_f + 0.5| > |\lfloor Y_f \rfloor - Y_f - 0.5| \\ \lfloor Y_f \rfloor + 0.5 & ||\lfloor Y_f \rfloor - Z_f + 0.5| \leq |\lfloor Y_f \rfloor - Y_f - 0.5| \end{cases} \quad (3)$$

A represents the actual generation position of the next fire source controlled by the fire source generator. During training, if the area where A is located is occupied by walls or other entities, this fire source will not be generated and a certain reward will be deducted. In non-training mode, if an occupied situation occurs, a fire source will be generated at another random location in the environment.

V. Appendix E: Configuration Details

This section elaborates the implementation details of the environment in our research. The adversarial reinforcement learning framework and virtual environment are built using Unity3D engine version 2021.3.3f1c1. The training is conducted on a server platform equipped with 32GB RAM, Intel i9-12900k CPU, and an Nvidia RTX 3080Ti GPU. Our software configuration is as follows:

- PyTorch version: 1.8.2
- CUDA version: 11.7
- ML-Agents version: 0.29.0
- Python version: 3.8.13

We fine-tuned the hyperparameters for training. For the initial learning rate, to address large fluctuations in the training curve, we slightly reduced the learning rate. For batchsize, to obtain more accurate gradients, we moderately

TABLE I: Configuration for Evacuation Robot Agent Group

trainer type	poca		
hyperparameters	batch size	1024	
	buffer size	20480	
	learning rate	0.0001	
	learning rate schedule	linear	
	beta	0.005	
	beta schedule	constant	
	epsilon	0.2	
	epsilon schedule	linear	
	lambd	0.95	
	num epoch	3	
network settings	vis encode type	simple	
	normalize	true	
	hidden units	256	
	num layers	3	
reward signals	extrinsic	gamma	0.90
		strength	1.0
keep checkpoints	100		
checkpoint interval	1000		
max steps	10000000		
time horizon	64		
summary freq	1000		

TABLE II: Configuration for Fire Source Agent

trainer type	ppo		
hyperparameters	batch size	1024	
	buffer size	10240	
	learning rate	0.0003	
	learning rate schedule	linear	
	beta	0.005	
	beta schedule	constant	
	epsilon	0.2	
	epsilon schedule	linear	
	lambd	0.95	
	num epoch	3	
network settings	vis encode type	simple	
	normalize	false	
	hidden units	256	
	num layers	3	
reward signals	extrinsic	gamma	0.99
		strength	1.0
keep checkpoints	80		
checkpoint interval	500		
max steps	500000		
time horizon	64		
summary freq	500		

increased it within a constrained range. Consistently, we also enlarged the buffersize.

The neural network design is also part of our parameter configuration work. Since the vectors we observe have relatively complex correlations, sufficiently large fully connected layers are needed. After research and discussion, we decide to use 3 hidden layers with 256 units each to decode our observed values. We apply normalization to the vector observation inputs to form residual connections for better inference. Importantly, we also adjusted the extrinsic reward signals from the environment for the MA-POCA algorithm, setting the discount factor gamma of rewards from the future environment to 0.9, meaning how far into the future the agent cares about rewards. We have presented the specific configuration files used for training in the tables provided. Table 1 showcases the configuration for the evacuation robot agent group, while Table 2 displays the configuration for the fire source agent.

The elaborate tuning of these key training hyperparameters aims at more stable and effective adversarial reinforcement learning for the complex fire evacuation scenarios.

VI. Appendix F: Calculation of Evaluation Formula Parameters

Regarding the values of related coefficients, first we isolate the part related to Reward in the evaluation function and set it as y (Formula 4):

$$y = \frac{b^{(\frac{R}{\alpha})}}{c} \tag{4}$$

We set it as a curve similar to the exponential function, first to handle the case of negative Rewards, and second to utilize the properties of exponential functions. According to existing research, currently people generally use the maximized worst-case reward R_w as the metric to evaluate the training performance and robustness of reinforcement learning agents with adversarial input disturbances[2]. Where R_w refers to the reward under the worst possible adversarial attack sequence.

$$R_w = \min_{\|\delta_t\|_p \le \epsilon} \mathbb{E}_\tau[\mathrm{R}(\tau)] \tag{5}$$

$\mathbb{E}$ represents expectation, trajectory $\tau = (s_0, a_0, ..., s_T, a_T)$, where a_t, s_t, r_t are sampled from the perturbed policy of the agent(π), the state transition probabilities of the environment($\mathcal{P}$), and the reward function of the environment($\mathcal{R}$) respectively, and $\mathrm{R}(\tau) = \sum_t r_t$, but this is actually uncomputable.

However, Oikarinen et al further studied on this basis and proposed an alternative evaluation method called greedy worst-case reward (GWC)[1], which approximates the expected R_w and can be efficiently computed with estimable linear complexity.

Specifically, the idea of GWC is to avoid evaluating the exponential number of trajectories, and use a simple heuristic approach to approximate R_w. The method is to greedily choose the action with the lowest reward at each state. For our scenario, the worst-case outcome of the worst action at each state ideally leads to the death of all humans. Therefore, we can use the cumulative human death penalties set to represent our R_w (equation (6)).

$$R_w = \sum_{i=1}^{n} (\textit{-Fixed Pen}) \tag{6}$$

We refer to the ideas of Oikarinen et al[1] and use it to estimate the worst case in our scenario, finally obtaining the estimated worst reward case. After rounding, $R_w \approx -12000$. On the other hand, since our scenario is not that complex, the best reward case in our scenario is also estimable. The theoretical best reward R_b is estimated by the following formula:

$$R_b = \sum_{i=1}^{n} \left(\sum_{j=1}^{3} \left(hp - \frac{Avg[Min(len_j)/speed]}{framesp \times rate} \right) + \textit{FixReward} \right) \tag{7}$$

Where hp represents the $health$ of humans in the scenario, initially set to 100, *Fixed Reward* represents the fixed additional reward, in addition to Health, settled when humans evacuate, len_j is the distance from the human on floor j to the exit or stairwell entrance, the Min function is used to represent the nearest distance to exit or stairwell since there are two stairwells in our scenario, $speed$ is the human speed, $framesp$ is the frame rate, we use frame rate instead of time concept since the time used for frame updates differs under different hardware conditions, $rate$ is the health deduction rate of humans, set as 0.01 in our scenario, n is the number of humans, n=40. Finally we obtain the theoretical optimal reward, round it and get $R_b \approx 15000$. Note that the group reward of the agent group is used in our experiments and calculations.

As we know, if reinforcement learning has learned the correct policy, its obtained reward curve will optimistically rise, but as training time increases, the rise will gradually decrease until almost stopping rising in late training. We believe that even small rises in reward curves that tend to stabilize indicate major breakthroughs for the agent. Therefore, when agent's Reward exceeds a certain value, we want to amplify the impact of such slight improvements on the overall evaluation formula E.

Utilizing the properties of the exponential function y, when R is below a certain value, the entire y function becomes very small, indicating the overall performance of the agent group is extremely poor. We call this Critical Point 1, when we believe the agent's policy cannot impact the environment, $R \approx R_w$. Conversely, when R exceeds a certain value, the entire y function explodes exponentially and affects the balance of the entire evaluation function. We want to set the other Critical Point 2 at the start of exponential explosion, and R should be close to R_b at this point. Our expectation is to leverage the exponential explosion effect to increase the weights of improvements obtained by late agent training in a disguised way. We calculate the needed b and c through the two set Critical Points.

We have conducted in-depth research and analysis on the impact of y values on the evaluation function. When designing the difficulty function, we have already considered the upper and lower limit factors of difficulty, and further explored this basis. Based on the comprehensive grasp of the upper and lower limits of the difficulty function, we finally chose two suitable parameters $y1$=0.1 and $y2$=1000. This gives us Critical Point 1 ($R1$=-12000, $y1$=0.1) and Critical Point 2 ($R2$=14000, $y2$=1000). By calculation we get the base coefficient that meets our expectations b=1.4251, control coefficient c= 1/7.017. At this point when R=15000 the slope of y is about 0.5, the slope of Critical Point 1 is about 0.00004, and the slope of Critical Point 2 is about 0.4.

Based on the data we observed during the experiments, we find that when the scaling coefficient α takes the value of 1000, it can produce good practical effects on data processing and help improve the reliability of the experiments.

Based on the existing coefficient values, we can calculate a formula without considering β:

$$e' = \left(\frac{b^{\left(\frac{R}{\alpha}\right)}}{c} + \frac{h^2}{k\alpha^2} \right) \tag{8}$$

In this case, we will set $\beta = e'$ and use it as a baseline to measure other cases. In other words, the evaluation formula with coefficients other than k is like a ruler - we just need to calibrate it using the agents' average performance to mark the scale, so it can be used to estimate efficiency.

Through multiple experiments, we selected some data with relatively high rewards and low difficulties from the average performance of MA-POCA baseline, and compared them with data of relatively low rewards and high difficulties. We believe the performance of these two cases is similar(as the scale), and bring their reward values R and difficulty values h into the formula to obtain the proportionality coefficient k as 0.009, which indicates the weight distribution of the two parts of the evaluation function. Finally, we obtain all relevant coefficients and $\beta = e' = 1649.7$, at which point $E = 1$.

In summary, through comprehensive analysis we determine appropriate values for the key coefficients involved in the evaluation formula, enabling it to effectively assess the training effects under changing difficulties. The coefficients are grounded in observed data patterns and extensive experiments.

REFERENCES

[1] Tuomas Oikarinen et al. "Robust deep reinforcement learning through adversarial loss". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 26156–26167.

[2] Huan Zhang et al. "Robust deep reinforcement learning against adversarial perturbations on state observations". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21024–21037.