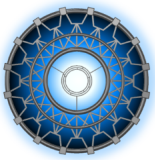


FireCommander: An Interactive, Probabilistic Multi-agent Environment for Joint Perception-Action Tasks

Esmaeil Seraj[†,*], Xiyang Wu[†] and Matthew C. Gombolay[†]

Georgia Institute of Technology, Atlanta (GA), United States

Institute for Robotics & Intelligent Machines

CORE Robotics Lab

[*] **Corresponding author:** Correspondences shall be forwarded to: *electronic mail:* eseraj3@gatech.edu

[†]E. Seraj and M. Gombolay are with the Institute for Robotics & Intelligent Machines (IRIM). X. Wu is with the School of Electrical and Computer Engineering. All authors are affiliates of the Georgia Institute of Technology, Atlanta (GA), United States.

[1]Version: User Guide Version 1.1

November 3, 2020

Abstract

The purpose of this tutorial is to help individuals use the FireCommander game environment for research applications. The FireCommander is an interactive, probabilistic joint perception-action reconnaissance environment in which a *composite* team of agents (e.g., robots) cooperate to fight dynamic, propagating firespots (e.g., targets). In FireCommander game, a team of agents must be tasked to optimally deal with a wildfire situation in an environment with propagating fire areas and some facilities such as houses, hospitals, power stations, etc. The team of agents can accomplish their mission by first sensing (e.g., estimating fire states), communicating the sensed fire-information among each other and then taking action to put the firespots out based on the sensed information (e.g., dropping water on estimated fire locations). The FireCommander environment can be useful for research topics spanning a wide range of applications from Reinforcement Learning (RL) and Learning from Demonstration (LfD), to Coordination, Psychology, Human-Robot Interaction (HRI) and Teaming. There are four important facets of the FireCommander environment that overall, create a non-trivial game:

1. **Complex Objectives:** Multi-objective Stochastic Environment.
2. **Probabilistic Environment:** Agents' actions result in probabilistic performance.
3. **Hidden Targets:** Partially Observable Environment.
4. **Uni-task Robots:** Perception-only and Action-only agents.

The FireCommander environment is first-of-its-kind in terms of including Perception-only and Action-only agents for coordination. It is a general multi-purpose game that can be useful in a variety of combinatorial optimization problems and stochastic games, such as applications of Reinforcement Learning (RL), Learning from Demonstration (LfD) and Inverse RL (iRL) to the following list of problems (see Introduction for details):

1. Multi-agent Coordination/Cooperation and Communication Learning
2. Multi-agent Learning from Heterogeneous Demonstrations (MA-LfHD)
3. Multi-robot Planning/Scheduling and Task Assignment (MRTA)
4. Human-Robot Interaction (HRI), Human-Robot Teaming and Psychology

FireCommander is open-source at[1] [1]:

`https://github.com/EsiSeraj/FireCommander2020`

Video tutorial and PowerPoint documentations of FireCommander can be found at:

`https://youtu.be/UQsWPh9c3eM`

Key-words: FireCommander, Multi-agent Coordination, Cooperation Learning, Joint Perception-Action, Python Environment, Interactive Game, Human-Robot Interaction, Reinforcement Learning, Multi-robot Task Assignment, Learning from Demonstration, Imitation Learning, Wireless Sensor and Actor Networks, Perception Robots, Perception-Action Communication Networks, Manipulator Robots, Joint Perception and Action Tasks, Manual, Tutorial

[1]Distributed under the terms of the GNU GENERAL PUBLIC LICENSE as a set of Python functions.

Contents

1 Introduction

The purpose of this tutorial is to help individuals use the FireCommander game environment for research applications. The FireCommander (Figure 1) is an interactive, probabilistic joint perception-action reconnaissance environment in which a *composite* team of heterogeneous agents (e.g., specific types of simulated robots) cooperate to fight dynamic, propagating firespots (e.g., targets). In FireCommander game, a team of agents must be tasked to optimally deal with a wildfire situation in an environment with propagating fire areas and some facilities such as houses, hospitals, power stations, etc. The team of agents can accomplish their mission by first sensing (e.g., estimating fire states), communicating the sensed fire-information among each other and then taking action to put the firespots out based on the sensed information (e.g., dropping water on estimated fire locations). The FireCommander environment can be useful for research topics spanning a wide range of applications from Reinforcement Learning (RL) and Learning from Demonstration (LfD), to Coordination, Psychology, Human-Robot Interaction (HRI) and Teaming. There are four important facets of the FireCommander environment that overall, create a non-trivial game:

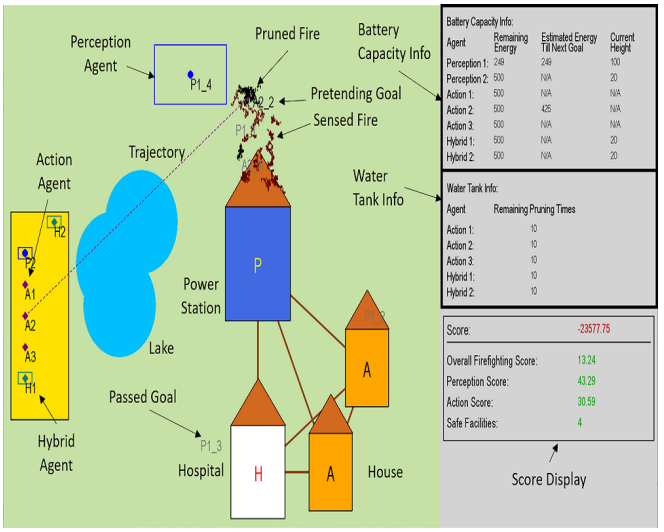


Figure 1: An example screenshot from FireCommander game Environment.

1. **Complex, Multi-Objective Game:** The environment is generally designed as a multi-purpose game in which robots (e.g., Perception-agents which are only capable of sensing and Action-agents which can manipulate but cannot sense) are tasked by a user to collaboratively fight multiple propagating wildfires and protect facilities (e.g, power stations, hospital, houses, etc.) from the fire. Agents are constrained by limited resources, such as battery-life and tanker capacity as well as motion restrictions, communication constraints and time limitations.

2. **Stochastic and Probabilistic Environment:** There are three sources of stochasticity in the FireCommander environment. All agents' actions change the states of the environment stochastically such that: (1) Action-agents can put out the fires in their field-of-view (FOV) according to a random probability distribution, which is designed through a confidence level coefficient. (2) Perception uncertainty varies with respective Perception-agent's altitude. Altitude has direct and reversed relation with observable area (e.g., FOV) and sensing quality (uncertainty), respectively. We model a perception agent's altitude-dependent sensing quality (e.g., the lower the altitude, the higher the quality of estimation) such that, perception-agents can sense (e.g., detect/locate) the fires within their FOV according to a random probability distribution which depends on their altitude. A perception agent at its lowest safe-altitude can sense 100% of the firespots in its FOV while a perception agent at its highest allowable altitude can only sense 40% of the firespots within its FOV. (3) The third stochasticity is associated with the fire behavior. Fire can appear at anytime during the game, anywhere on the map. Moreover, fire propagates according to a stochastic mathematical model (e.g., see FARSITE in Section 2.3.2).

3. **Hidden Targets:** Initially, no firespot is visible on the screen. Users will only see a raw map, including the UAV base (e.g., robot depot) and the various facilities on the map. Firespots can only be seen through Perception-agents. Fires can also appear at anytime anywhere around the map.

Moreover, since the firespots are dynamic, once the Perception agent leaves the fire location, the new location of the firespot becomes uncertain, until a Perception agent is summoned to the area again for an updated observation.

4. **Composite Robot Team:** We define a composite robot team as a group of agents that perform different tasks according to their respective capabilities while their tasks are co-dependent on accomplishing an overarching mission. In FireCommander environment, the robot team is composed of (1) perception-only and (2) action-only agents. As such, we introduce *Perception-agents* and *Action-agents* in this environment, which together, form a composite robot team. As such, the designed coordination policies must take into account the Perception-Action hierarchy and communication problems (e.g., firespots must first be observed by a perception agent, the sensed information must be communicated to an action agent and then, the firespots are put out by the action agent.)

The FireCommander environment is first-of-its-kind in terms of including Perception-only and Action-only agents for coordination. Considering such composite agents helps to represent a clearer picture of coordination and efficient communication problems in heterogeneous teams. FireCommander is a handcrafted environment than can be modified to cover a wide range of game complexities from simple, single-objective games with only one (or two in heterogeneous agent cases) agents, to complex, multi-objective stochastic games with numerous agents and constraints. Many of the existing multi-agent environments, such as the OpenAI multi-agent particle environments[2] are not inherently designed to include heterogeneous agents with different capabilities and are often heavily modified to pose a meaningful heterogeneous communication or coordination problem. Moreover, other environments such as StarCraft II are often too complex and hard, if not impossible, to be modified to match a specific scenario, while FireCommander is open-source and the game logistics can be easily modified to adapt a desirable setting. Additionally, FireCommander is a general multi-purpose game that is designed to be readily leveraged in a variety of combinatorial optimization problems [2, 3, 4, 5], and stochastic games [6, 7], such as applications of Reinforcement Learning (RL) [8], Learning from Demonstration (LfD) [9] and Inverse RL (iRL) [10] to the following list of problems:

1. **Multi-agent Coordination/Cooperation and Communication Learning:** Due to inherently heterogeneous agents (e.g., agents with different capabilities), the FireCommander environment poses an interesting multi-agent coordination problem for Perception and Action-agents to learn how to cooperate and how to efficiently communicate. Figure. 2 represents the FireCommander environment logic and various multi-agent coordination and communication problems it covers. We have developed various ready-to-use versions of the FireCommander which can be directly leveraged to test multi-agent reinforcement learning (MARL) algorithms. We also developed a graphical user interface (GUI) which can be used to record expert data for learning from demonstration (LfD) and human-robot interaction (HRI) studies dealing with multi-agent, heterogeneous coordination/cooperation and communication Learning, similar to the works in [11, 12, 13, 14, 15, 16, 17, 18]. Please refer to Section 2.2.7 for more details.
2. **Multi-agent Learning from Heterogeneous Demonstrations (MA-LfHD):** FireCommander inherently includes two categories of heterogeneous robots: (1) Perception (e.g., Sensing) agents and (2) Action (e.g., Manipulator) agents, and is a multi-objective game. Missions and games in FireCommander do not have unique solutions and each category of agents must learn both a local team-objective (e.g., either perform sensing or acting) as well as a global composite-objective (e.g., fight the fire and protect the facilities). Accordingly, the game objectives can be interpreted and interacted with in various different ways by a user and thus, FireCommander can be a perfect environment for developing learning from *heterogeneous demonstrations* algorithms [20, 21, 22, 23, 24], particularly for multi-agent coordination.

[2]Available Online: https://github.com/openai/multiagent-particle-envs

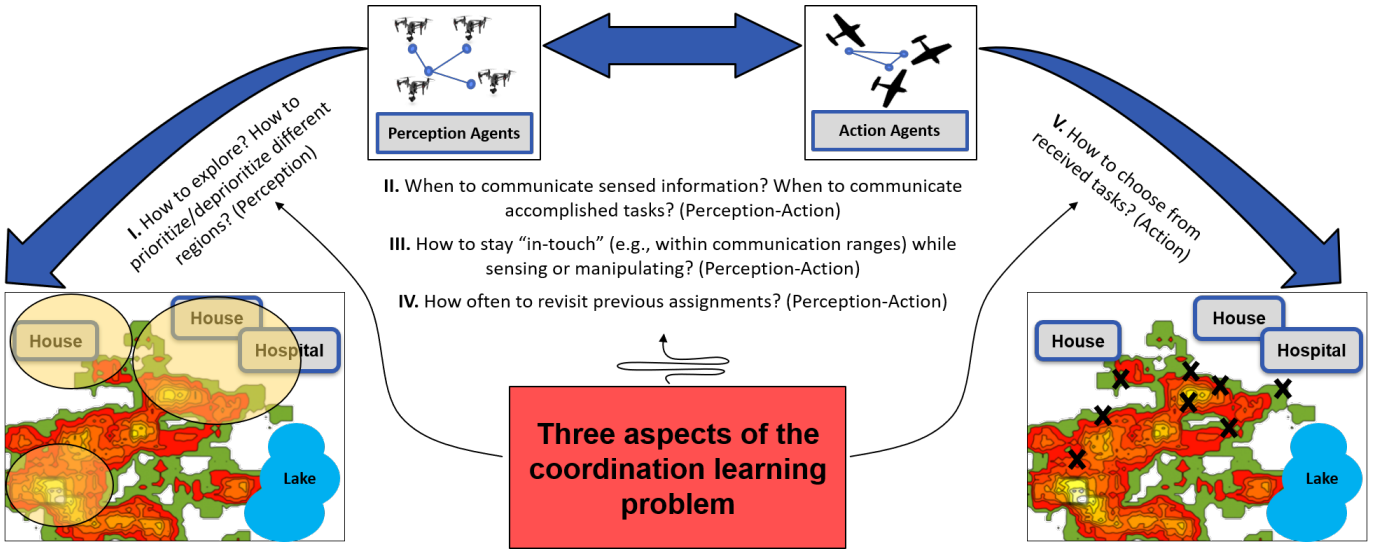


Figure 2: The FireCommander environment logic and various multi-agent coordination problems it covers. The environment generally includes three aspects of the multi-agent coordination problem: (1) coordination among Perception-agents and how they optimize their performance in the environment, (2) coordination among Action-agents and how they optimize their performance in the environment, and (3) coordination among Perception and Action (e.g., Perception-Action) agents and how they optimize their communication for improved collective behavior in the environment [19].

3. **Multi-robot Planning/Scheduling and Task Assignment (MRTA):** As described in Section 2.2.2, in FireCommander, Perception and Action agents must efficiently coordinate to cooperatively fight propagating fires. Doing such however, requires solving various combinatorial optimization problems (COPs) such as "*How should agents choose where to go?*", or "*How should agents divide up the tasks?*", or "*How should agents be distributed among tasks, areas of the map, etc., given we have limited resources?*", and more. Solving for these questions falls within the realms of combinatorial optimization (see Section 2.2.2). Our FireCommander environment can be a complex environment for many challenging COPs such as dynamically learning scheduling policies [25, 26], and multi-agent task assignment [27].

4. **Human-Robot Interaction (HRI), Human-Robot Teaming and Psychology:** FireCommander includes an interactive game-based graphical user interface (GUI) which can be of great use in HRI researches. As an instance, the FireCommander game can be leveraged to design environments with heavy/light workload and then test how an expert's policy design efficiency and quality is affected under situational stress. Various other HRI topics can be similarly modeled to leverage FireCommander as their test-bed, be such as trust and accountability [28, 29], anthropomorphism [30, 31, 28], human-robot co-adaptation [32], human-guided optimization [33, 34, 35], cognitive BCI [36, 37, 38, 39, 40] and many more [41, 42].

5. **Wireless Sensor and Actor Networks (WSAN):** Wireless sensor and actor networks refer to a group of sensors and actors linked by wireless medium to perform distributed sensing and acting tasks [43]. FireCommander environment is by definition a WSAN, in which Perception-agents play the role of wireless sensors and Action-agents reflect the wireless actors. Asu such, FireCommander is a perfectly fit environment for developing coordination algorithms for WSANs [44, 45, 43]. Moreover, WSANs might include static sensors and actors, static sensors and dynamic actors or vice versa. FireCommander environment can easily be modified to match these settings while other existing games and environments lack such flexibility.

1.1 License - No Warranty

FireCommander: An Interactive, Probabilistic Multi-agent Environment for Joint Perception-Action Tasks.

1.2 Citations: Code-Base, Tutorials and Documentations

Within the limits of the GNU GENERAL PUBLIC LICENSE, you can use the toolbox as you please; however, if you use the toolbox in a work of your own that you wish to publish, please make sure to cite the code-base on GitHub [1] this user manual [46] properly, as shown below. This way you will contribute to helping other scholars find these items.

- Esmaeil Seraj, Xiyang Wu and Matthew C. Gombolay, "FireCommander", (2020), GitHub Repository, Release 1.1, [Online] https://github.com/EsiSeraj/FireCommander2020[3].
- Esmaeil Seraj, Xiyang Wu and Matthew C. Gombolay, "FireCommander: An Interactive, Probabilistic Multi-agent Environment for Joint Perception-Action Tasks" arXiv Preprint, 2020 [Online].

1.3 Download and Utilization

The latest version of the "FireCommander" software can be downloaded directly from from the public repository on GitHub, at [1]:

https://github.com/EsiSeraj/FireCommander2020

1.4 Getting Help

The codes are standardized and commented as frequently as possible. Moreover, the current user manual tutorial, a PPT tutorial, a video tutorial and a project publication, all are provided to help

[3]
```
@misc{seraj2020FireCommander,
  author = Seraj, Esmaeil and Wu, Xiyang, and Gombolay, Matthew,
  title = FireCommander,
  volume = Release 1.1,
  year = 2020,
  publisher = GitHub,
  journal = GitHub Repository,
  howpublished = https://github.com/EsiSeraj/FireCommander2020
}
```

individuals utilize this code in their research studies. Links to the mentioned blog-posts, tutorials and documentations are presented below:

- **Code & GitHub Blog:** [1] https://github.com/EsiSeraj/FireCommander2020
- **User Manual Tutorial:** [46] https://arxiv.org/pdf/1907.02862.pdf
- **PowerPoint Tutorial:** [19] https://arxiv.org/pdf/1907.02862.pdf
- **Video Tutorial:** https://youtu.be/UQsWPh9c3eM
- **Project Publication:** https://arxiv.org/pdf/1907.02862.pdf

You can also contact the corresponding author[4] directly to ask any related questions or discuss possible difficulties or errors you might encounter. Please feel free to contact in either case.

2 Environment User Guide

2.1 FireCommander Overview

The FireCommander is an interactive, probabilistic joint perception-action reconnaissance environment in which a *composite* team of heterogeneous agents (e.g., specific types of simulated robots) cooperate to fight dynamic, propagating firespots (e.g., targets). In FireCommander game, a team of agents must be tasked to optimally deal with a wildfire situation in an environment with propagating fire areas and some facilities such as houses, hospitals, power stations, etc. The team of agents can accomplish their mission by first sensing (e.g., estimating fire states), communicating the sensed fire-information among each other and then taking action to put the firespots out based on the sensed information (e.g., dropping water on estimated fire locations). The FireCommander environment can be useful for research topics spanning a wide range of applications from Reinforcement Learning (RL) and Learning from Demonstration (LfD), to Coordination, Psychology, Human-Robot Interaction (HRI) and Teaming. There are four important facets of the FireCommander environment that overall, create a non-trivial game:

1. **Complex, Multi-Objective Game:** The environment is generally designed as a multi-purpose game in which robots (e.g., Perception-agents which are only capable of sensing and Action-agents which can manipulate but cannot sense) are tasked by a user to collaboratively fight multiple propagating wildfires and protect facilities (e.g, power stations, hospital, houses, etc.) from the fire. Agents are constrained by limited resources, such as battery-life and tanker capacity as well as motion restrictions, communication constraints and time limitations.

2. **Stochastic and Probabilistic Environment:** There are three sources of stochasticity in the FireCommander environment. All agents' actions change the states of the environment stochastically such that: (1) Action-agents can put out the fires in their field-of-view (FOV) according to a random probability distribution, which is designed through a confidence level coefficient. (2) Perception uncertainty varies with respective Perception-agent's altitude. Altitude has direct and reversed relation with observable area (e.g., FOV) and sensing quality (uncertainty), respectively. We model a perception agent's altitude-dependent sensing quality (e.g., the lower the altitude, the higher the quality of estimation) such that, perception-agents can sense (e.g., detect/locate) the fires within their FOV according to a random probability distribution which depends on their altitude. A perception agent at its lowest safe-altitude can sense 100% of the firespots in its FOV while a perception agent at its highest allowable altitude can only sense 40% of the firespots within its FOV.

[4]Esmaeil (Esi) Seraj: eseraj3@gatech.edu

(3) The third stochasticity is associated with the fire behavior. Fire can appear at anytime during the game, anywhere on the map. Moreover, fire propagates according to a stochastic mathematical model (e.g., see FARSITE in Section 2.3.2).

3. **Hidden Targets:** Initially, no firespot is visible on the screen. Users will only see a raw map, including the UAV base (e.g., robot depot) and the various facilities on the map. Firespots can only be seen through Perception-agents. Fires can also appear at anytime anywhere around the map. Moreover, since the firespots are dynamic, once the Perception agent leaves the fire location, the new location of the firespot becomes uncertain, until a Perception agent is summoned to the area again for an updated observation.

4. **Composite Robot Team:** We define a composite robot team as a group of agents that perform different tasks according to their respective capabilities while their tasks are co-dependent on accomplishing an overarching mission. In FireCommander environment, the robot team is composed of (1) perception-only and (2) action-only agents. As such, we introduce *Perception-agents* and *Action-agents* in this environment, which together, form a composite robot team. As such, the designed coordination policies must take into account the Perception-Action hierarchy and communication problems (e.g., firespots must first be observed by a perception agent, the sensed information must be communicated to an action agent and then, the firespots are put out by the action agent.)

2.2 Fundamentals and General Applicability

2.2.1 Multi-agent Systems

In general, an agent is an autonomous physical (e.g., robot) or virtual (e.g., simulation) entity that can act, perceive its environment (e.g., fully or partially) and communicate, and has skills to achieve a desired goals. In a multi-agent system, multiple agents *interact* in an environment and on objects to accomplish some local or global objectives. Interactions can be defined as relations between all the entities, a set of operations that can be performed by the entities and the changes in the environment over the time and due to agents' actions [47]. Agents in a multi-agent system can be of similar type and with similar dynamics (e.g., homogeneous agents) or otherwise (e.g., heterogeneous agents).

Multi-agent teams are able to execute time-sensitive, complex missions by cooperatively leveraging their unique capabilities and design [48, 49, 27]. Heterogeneity in robots' design characteristics and their roles are introduced into these multi-robot systems to (1) leverage the relative merits of the different agents and their capabilities [48, 49, 27] and, (2) deal with the dynamic and unpredictable nature of the real-world for which designing homogeneous, versatile robot teams that can effectively adjust to all circumstances is difficult and costly [48, 49, 50].

A set of operations/policies taken by agents in a multi-agent system forms interaction. Interacting agents in a multi-agent systems create a notion of Markov Games (MG) [7], which are a special case of Stochastic Games (SG) [6]. Interactions among a set of N agents in an SG can generally take three forms: (1) Cooperative [51, 52, 53], (2) Non-cooperative [53, 54, 55] and (3) Competitive [56, 57]. Cooperative agents have the same reward structure and the goal of the system is to *collectively* maximize a common discounted return [48, 51, 52, 53]. Non-cooperative agents also try to achieve a common goal; however, they do not behave collectively and each agent is only trying to maximize their own local rewards (which leads to maximizing the total return as well) [53, 54, 55]. Competitive agents are similar to non-cooperative agents in only caring about their own local rewards; however, in a competitive SG setup, maximizing each agent's local return does not result in global return maximization [56, 57]. An SG can also be a combination of the aforementioned settings, for instance in the game of soccer where agents are cooperative and competitive at the same time [58, 59, 60].

We classify FireCommander as a cooperative stochastic game. In the FireCommander environment, there are two categories of agents: (1) Perception (e.g., sensing or fire-observing) and (2) Action (e.g., manipulator or firefighting) agents. We note that these agents are uni-task and can only perform their specific task. The objective of the game is to maximize the firefighting reward by cooperatively putting out all of the dynamic, propagating firespots and keep the facilities (e.g., house, hospital, etc.) on the map safe from the fire. Nevertheless, firespots are initially invisible. To fight the fire, one must first find it (e.g., the spots of fire) on the map. We note that, Action-agents cannot fight the fire, unless the fire is first sensed by a Perception-agent, and Perception-agent cannot put out the fires without an Action-agent fighting the fire after it is sensed. As such, the FireCommander game environment is cooperative, since agents must cooperate to accomplish a common, complex mission. We note that, we introduce the term *composite teams* to refer to our FireCommander environment in which agents are co-dependent on accomplishing an overarching mission [49].

2.2.2 Combinatorial Optimization Problems

An optimization problems (e.g., the problem of finding the best solution to a maximization or minimization problem) can generally be divided into two categories: (1) continuous variable optimization and, (2) discrete variable optimization problem (OP). A discrete variable optimization is also known as a *combinatorial* optimization problems (COP) [2, 3, 4]. In the continuous OPs, solutions are a set of real numbers, while in a COP, solutions are objects, such as integers, permutations, graphs, etc., from a large finite set (or countable infinite) [5, 2, 3, 4]. Two simple but well-known examples of typical COPs are the traveling salesman problem (TSP) and the mixed linear integer programming (MILP) [2, 61, 3, 4, 62]:

- **TSP:** given the (x, y) positions of N different cities, find the shortest possible path that visits each city exactly once.
- **MILP:** maximize a specified linear combination of a set of integers $\{x_1, x_2, \cdots, x_n\}$ subject to a set of linear constraints of the form $a_1x_1 + \cdots + a_nx_n \leq c$

Many multi-agent planning, scheduling, resource allocation and task assignment problems are categorized as COPs, since the solution to these problems typically consists of finding an optimal object from a finite set of objects [2, 3, 4]. In many COPs, exhaustive search (e.g., brute force) is not tractable as the space of possible solutions is typically too large. In some cases, problems can be solved exactly using Branch and Bound techniques [63, 2]. However, in other cases no exact algorithms are feasible, and randomized search algorithms must be employed [2, 3, 4].

In FireCommander , Perception and Action agents must be coordinated to cooperatively fight propagating fires. Doing such however, rises various questions such as:

- *How perception agents should select where to go?*
- *How action agents should divide up the tasks received?*
- *Which perception agent goes to explore somewhere in a map that has not been explored yet?*
- *Which action agent should start a task, given a set of constraints such as battery-life, distance, tasks on the queue, etc?*
- *How perception agents should balance between exploring new ares of the map versus exploiting the firespots they already found (e.g., spend time to communicate the sensed information to an action agent)?*
- *How often should perception agents re-visit previously detected firespots to update their information?*

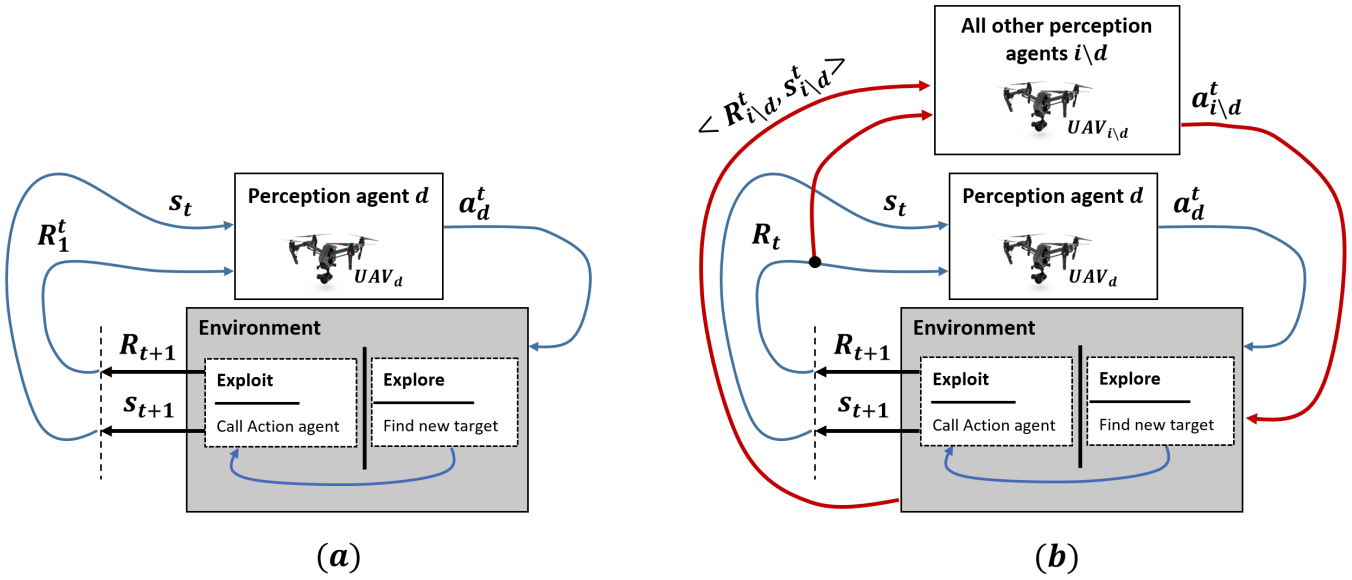


Figure 3: Left-side figure represents the single-agent reinforcement learning (RL) problem in which an agent interacts with an environment by taking an action a at state s and receiving a reward R. Right-side figure presents the multi-agent reinforcement learning (MARL) problem in which a group of agents interact with the environment. Note that here, the reward of taking a rewarding action by one agent is also received by all other agents that did not take that action, since the agents are cooperating and the environment state-change occurs for all agents, regardless of the actor. The objective in both RL and MARL is to maximize a cumulative discounted return.

- *What is an optimal path for an agent to take towards a goal/task?*
- *How agents should be distributed between tasks, areas of the map, etc., given we have limited resources?*

All of the above questions, to some extent, define a constraint satisfaction problem (CSP), solving which falls within the realms of COPs. As such, our FireCommander environment can be a complex testbed environment for many challenging combinatorial optimization problems.

2.2.3 Multi-agent Reinforcement Learning (RL)

Figure 3 represents the single-agent reinforcement learning (RL) and the multi-agent reinforcement learning (MARL) problems. Left-side figure in Figure 3, represents the single-agent RL in which an agent interacts with an environment by taking an action a at state s and receiving a reward R. Right-side figure in Figure 3, presents the MARL problem in which a group of agents interact with the environment. Note that here, the reward of taking a rewarding action by one agent is also received by all other agents that did not take that action, since the agents are cooperating and the environment state-change occurs for all agents, regardless of the actor. The objective in both RL and MARL is to maximize a cumulative discounted return.

In FireCommander, we are dealing with a MARL problem. However, due to the unique characteristics of the FireCommander environment, such as the dynamicity of the environment (e.g., constantly state-changing firespots) and a set of initially invisible firespots, the problem resembles *restless decision making* problem [64]. In our restless decision-making problem, regardless of the collective decisions of the perception-action team, the states of all observed or unobserved targets are constantly evolving through time. Moreover, as long as a fire spots is not being observed by a perception agent, the team does not have any information about the state of the fire and the uncertainty about

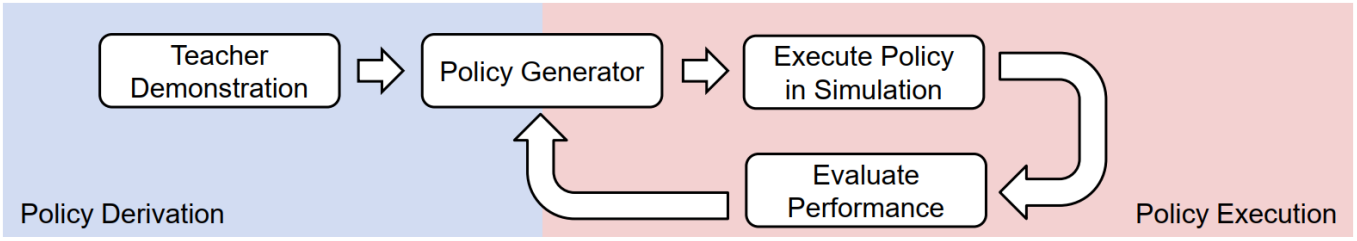


Figure 4: This figure represents the Learning from Demonstration (LfD) framework.

its states keep increasing.

This problem can be considered as a variant of POMDP. Accordingly, our objective is to find an optimal policy, π^*, over all admissible policies in set, $\pi \in \Pi$, that maximizes the expected, time-discounted reward collected by all perception agents over an infinite horizon, as in Equation 1 [49].

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \left(r_t^{P1} + \cdots + r_t^{PN_P} \right) \right] \tag{1}$$

2.2.4 Multi-agent Learning from Demonstrations: Inverse RL

An optimal (or near-optimal) policy, enables a robot to select a wise action based upon the current states of the world which leads to a maximized reward. The development of such policies by hand is often very challenging and as a result machine learning techniques such as RL have been applied to policy development [9, 65]. RL works based on agent's experience through interacting with the environment and often, the more experience-data an agents achieves, the better policy it can generate. Nevertheless, acquiring enough experience data for an RL algorithm to perform well is a limiting factor due to its need for an often prohibitively large number of environment samples before the agent reaches a desirable level of performance [9, 65]. As such, LfD algorithms have been proposed to solve the agent's need for exhaustive search over the state-action space [66, 9, 67].

Figure 4 represents the Learning from Demonstration (LfD) framework. In LfD, a policy is learned from demonstrations (e.g., sample trajectories), provided by a supposed expert teacher [68]. Sample trajectories are defined as sequences of state–action pairs that are recorded during the expert teacher's demonstration of the desired behavior. LfD algorithms utilize produced the trajectory dataset to derive a policy that reproduces the demonstrated behavior (Figure 4). Obtaining a policy in this way is in contrast to RL techniques in which a policy is learned from exploration and experience [9, 65]. LfD is specifically of interest when the reward function structure is unknown.

- **Multi-agent Learning from Heterogeneous Demonstrations (MA-LfHD)**– FireCommander inherently includes two categories of heterogeneous robots: (1) Perception (e.g., Sensing) agents and (2) Action (e.g., Manipulator) agents, and is a multi-objective game. Missions and games in FireCommander do not have unique solutions and thus, can be interpreted and interacted with in various different ways by a user and thus, can be a perfect test-bed for developing learning from *heterogeneous demonstrations* algorithms [69, 20, 70, 21, 22, 23, 24, 71].

2.2.5 Perception-Action Networks

Robot intelligence requires a real-time connection between sensing and action [72]. This fact directly extends to multi-agent systems, such as the FireCommander environment in which in order to achieve *team intelligence*, the perception and action robot teams must communicate to accomplish

a complex overarching mission. Communication channels between agents form a network and thus, communicating perception and action teams, forms a Perception-Action Network.

Perception-Action Networks (PAN) can be static or dynamic [73, 74]. One famous instance of dynamic PANs are Mobile Sensor Networks (MSN) which consist of a group of perception agents (or sometimes even hybrid sensing-acting agents) that can move around an environment to search for their targets of interest (e.g., firespots in wildfire fighting) [75, 73, 76]. MSNs are specifically useful for covering large environments with limited number of agents and for field coverage problems in which the environment is dynamic (e.g., dynamic field coverage) [75, 77, 78, 79, 49].

2.2.6 Wireless Sensor and Actor Networks

Wireless sensor and actor networks (WSAN) refer to a group of communicating perception and action agents (e.g., PANs) which are used to perform distributed sensing and acting tasks [43]. In WSANs, sensors gather information about the physical world (e.g., similar to Perception agents in FireCommander environment), while actors take decisions and then perform appropriate actions upon the environment (e.g., similar to Action agents in FireCommander). Similar to the FireCommander game in which users remotely use perception and action agents to fight the fire, WSANs allow a user to effectively sense and act from a distance [43]. In order to provide effective sensing and acting, coordination mechanisms are required among sensors and actors [43, 44, 45].

An important note to be considered in WSANs is that, to perform correct and timely actions, sensing data must be valid at the time of acting [43]. This is particularly important in dynamic environments such as the FireCommander environment. This is because firespots are propagating (e.g., moving) and their position can only be seen when they are within the field-of-view (FOV) of a Perception agent. As such, when a perception agent observes a firespot, simply passing the spot's current position to an action agent and then leaving to find another firespot will not work, since while the action agent is on its way the firespot will move to a new location and also, action agents cannot sense and thus, cannot see the firespot's new location. Accordingly, the exploration/exploitation dilemma becomes a more challenging problem for effectively coordinating the Perception-Action agents in a composite team [49]. Moreover, WSANs might include static sensors and actors, static sensors and dynamic actors or vice versa. FireCommander environment can easily be modified to match these settings while other existing games and environments lack such flexibility.

2.2.7 Learning to Cooperate: Coordination and Communication Learning

As discussed earlier in Sections 2.2.1 and 2.2.6, in a multi-agent system, in order to accomplish the complex mission (or maximize a time-discounted final return) agents must collaborate. Effective collaboration/cooperation requires a *robust*, *accurate*, *efficient* and *scalable* coordination mechanism and, such coordination among agents, requires communications, which is constrained by many limiting factors. As a solution to all of the mentioned coordination and communication problems, RL, specifically when combined with deep learning (Deep RL[5]) and function approximation, has been widely studied in recent years [11, 12, 13, 14, 15, 16, 17, 18, 83]. These frameworks intend to enable the agents in a team to "*learn how to cooperate*".

Multi-agent reinforcement learning (MARL) provenly performs better when there is communication among participating agents allowing them to coordinate their actions for maximising their shared utility. However, information sharing among miscellaneous agents (of different types and capabilities)

[5]Despite neural networks (NN) being large and requiring massive computational power, Deep RL algorithms are in fact becoming more and more popular in online real-world robotics through applying neural network pruning and deep compression and quantization algorithms such as [80, 81, 82]

can lead to heterogeneous communication protocols. When communication protocols include such miscellaneous information, agents of different types might not be able to differentiate the heterogeneity in the globally sent and received massages and extract valuable information that helps cooperative decision making. Therefore, communication may become unhelpful, and could even deteriorate the multi-agent cooperation learning. FireCommander game not only covers almost all of the mentioned coordination and communication problems, it also adds some challenging twists such as:

- Complex, multi-objective environment
- Partially observable and dynamic environment
- Constrained Communication
- Uni-task robots

As such, FireCommander can be considered as a good, challenging testbed for learning multi-agent cooperation and developing end-to-end differentiable communication protocols for heterogeneous and composite teams [49].

2.2.8 Multi-robot Planning/Scheduling and Task Assignment (MRTA)

Many multi-agent planning, scheduling, resource allocation and task assignment problems are categorized as COPs, since the solution to these problems typically consists of finding an optimal object from a finite set of objects [2, 3, 4]. In many COPs, exhaustive search (e.g., brute force) is not tractable as the space of possible solutions is typically too large. In some cases, problems can be solved exactly using Branch and Bound techniques [63, 2]. However, in other cases no exact algorithms are feasible, and randomized search algorithms must be employed [2, 3, 4].

Nevertheless, search algorithms are time-consuming, require significant computational power and resources and can easily get intractable when the problem dimensions increases. Accordingly, similar to many other optimization problems, COPs, such as dynamic scheduling, have also been tried to be solved by RL algorithms combined with deep learning (Deep RL) and Graph Neural Networks (GNN) [25, 16, 26]. Here, learning frameworks intend to enable the agents in a team to "*learn the scheduling or planning policies*" [25, 84].

As described in Section 2.2.2, in FireCommander, Perception and Action agents must efficiently coordinate to cooperatively fight propagating fires. Doing such however, requires solving various combinatorial optimization problems (COPs) such as "*How should agents choose where to go?*", or "*How should agents divide up the tasks?*", or "*How should agents be distributed among tasks, areas of the map, etc., given we have limited resources?*", and more. Solving for these questions falls within the realms of combinatorial optimization (see Section 2.2.2). Our FireCommander environment can be a complex environment for many challenging COPs such as dynamically learning scheduling policies [25, 26], and multi-agent task assignment [27].

2.2.9 Human-Robot Interaction (HRI), Human-Robot Teaming and Psychology

FireCommander includes an interactive game-based graphical user interface (GUI) which can be of great use in HRI researches. As an instance, the FireCommander game can be leveraged to design environments with heavy/light workload and then test how an expert's policy design efficiency and quality is affected under situational stress. Various other HRI topics can be similarly modeled to leverage FireCommander as their test-bed, be such as trust and accountability [28, 29], anthropomorphism [30, 31, 28], human-robot co-adaptation [32, 85], human-guided optimization [86, 33, 87, 34, 35, 88] and many more [41, 42].

2.3 FireCommander: Game Structure and Dependencies

2.3.1 Functions, Python Packages and Dependencies

Once you have cloned/downloaded the toolbox, 10 files and folder as represented in following Table shall be appeared in your chosen directory. Table (1) represents all Python files (i.e. core functions and internal computational functions) that come with this toolset and a short descriptions of each. Additionally, a copy of both GNU general public license and this user manual are also included.

Table 1: FireCommander Python files and assets (MARL and LfD/HRI library packages)

** Multi-agent LfD and HRI Package **		
row	**name (function)**	**description**
1	`FireCommander.py`	Main script to run the FireCommander GUI
2	`Utilities.py`	Inner functions and assets required to run the environment
3	`WildFire_Model.py`	Script for FARSITE wildfire propagation model
4	`Scenario_Mode_Params.py`	Script including the Scenario Mode Parameters
5	`Demo_Visualization.py`	Script for deploying an existing teacher demo (data)
row	**name (folder)**	**description**
6	`Open_World_Data`	Contains recorded user-data for Open-World Mode
7	`Scenario_Data`	Contains recorded user-data for Scenario Mode
8	`Images`	Contains images required to run the GUI
** Multi-agent RL Package **		
1	`FireCommander_Base.py`	Homogeneous version of FireCommander for MARL
2	`FireCommander_Cmplx1.py`	Heterogeneous version of FireCommander (Cooperative)
3	`FireCommander_Cmplx2.py`	Full Heterogeneous version of FireCommander for MARL
4	`Utilities.py`	Inner functions and assets required to run the environments
5	`WildFire_Model.py`	Script for FARSITE wildfire propagation model

The codes are written in Python 3.6.4 and PyGame 1.9.6 is used for designing the GUI. The current versions have been tested and as a routine, any later version of these softwares should also be compatible with our code. Using older versions of Python such as Python 2.7 (or other earlier versions) are not recommended.

2.3.2 FARSITE: Wildfire Propagation Mathematical Model

We leverage the simplified Fire Area Simulator (FARSITE) wildfire propagation model [89, 78, 79, 49, 90], where q_t^i indicates how firespot i propagates according to Eq. 2 along the X-Y coordinates and thus, q is a 2-dimensional vector of size 1×2 [49].

$$q_t^i = q_{t-1}^i + \dot{q}_{t-1}^i \delta t \tag{2}$$

The above equation represents the fire motion model in which, $\dot{q}_t^i = dq_t^i/dt$ is the motion dynamics and identifies a fire's growth rate (i.e., fire propagation velocity) and is a function of fuel and vegetation coefficient (R_t), wind speed (U_t), and wind azimuth (θ_t). The first-order firespot dynamics, $\dot{q}_t$, can be estimated for each propagating spot by Equation 3, where $\mathcal{D} = \sin(\theta_t)$ and $\mathcal{D} = \cos(\theta_t)$ along X and Y axes, respectively [89].

$$\dot{q}_t = C(R_t, U_t)\mathcal{D}(\theta_t) \tag{3}$$

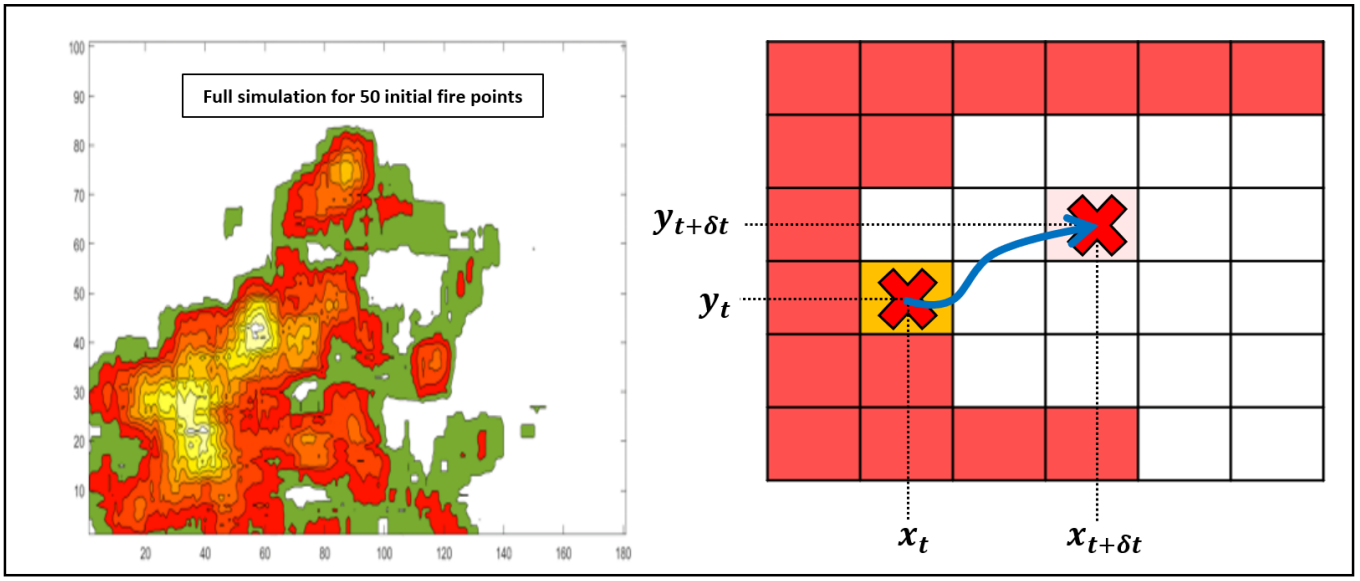


Figure 5: Wildfire simulations using FARSITE [89] fire propagation model. Left: simulation with 50 initial fire points. Right: simple grid visualization of the fire propagation based on the Equation 2.

In Equation 3, the spread rate $C(R_t, U_t)$ can be calculated as in Equation 4, in which $LB(U_t) = 0.936e^{0.256U_t} + 0.461e^{-0.154U_t} - 0.397$ and $GB(U_t) = LB(U_t)^2 - 1$.

$$C(R_t, U_t) = R_t \left(1 - \frac{LB(U_t)}{LB(U_t) + \sqrt{GB(U_t)}}\right) \tag{4}$$

Moreover, to account for the fire intensity (measured in kilo-watts per meter, $[\frac{KW}{m}]$), we leverage the method proposed by [91] which utilizes flame height h_t^q (measured in meters) and flame tilt angle α_t^q with respect to vertical horizon line (measured in degrees) for each fire-spot q at time t to calculate the fire intensity I_t^q, as follows in Eq. 5 [49].

$$I_t^q = 259.833 \left(\frac{h_t^q}{\cos\left(\alpha_t^q\right)}\right)^{2.174} \tag{5}$$

While fire is propagating, we assume each fire-spot radiates heat according to a Gaussian distribution and the intensities of adjacent points are linearly summed if the two points are within a predefined radiation range, which is in accordance with prior studies [78, 79, 49]. Figure 7 demonstrates the wildfire simulations using FARSITE [89] mathematical fire propagation model. Furthermore, due to the fuel exhaustion, we model the intensity decay of a fire spot during its ignition time δt_q, as a dynamic exponential decay rate λ over time, presented below in Equation 6 [49].

$$I_{t+\delta t}^q = I_t^q \left(e^{-\lambda \frac{\delta t_q}{R_t}}\right) \tag{6}$$

Finally, when a firefighting UAV drops the extinguisher fluid over an area of fire, we cut the fire intensities of the respective fire spots according to a predefined extinguisher fluid coefficient. A fire point is pruned from the fire-map if its intensity falls below a threshold value, leaving a burnt spot on the terrain map which cannot catch fire anymore [49].

2.3.3 Game Structure

As mentioned earlier there are two main library packages of FireCommander provided. First, we developed various ready-to-use versions of the Fire Commander which can be directly leveraged to

test multi-agent reinforcement learning (MARL) algorithms. Second, we also developed a graphical user interface (GUI) which can be used to record expert data for learning from demonstration (LfD) and human-robot interaction (HRI) studies dealing with multi-agent, heterogeneous coordination/cooperation and communication Learning.

We classify FireCommander as a cooperative stochastic game (see Section 2.2.1) in which all agents share a common objective and reward function, although the full complex version can also be considered as a non-cooperative Markov game which includes different objectives and rewards for different groups od agents. The environment includes uni-task and multi-task agents. The objective of the game is, given a limited time and number of heterogeneous agents, maximize the firefighting reward by (1) cooperatively putting out all propagating firespots and (2) keeping the targets (e.g., facilities such as house, hospital, etc.) on the map safe from the fire. Firespots are initially invisible. To fight the fire, one must first find the firespots on the map. We note that Action-agents cannot fight the fire unless the fire is first sensed by a Perception-agent, and Perception-agent cannot put out the fires without an Action-agent fighting the fire after it is sensed. As such, the FireCommander game environment is cooperative, since agents must cooperate to accomplish a common, complex mission. We introduce the term *composite teams* to refer to our FireCommander environment in which agents are co-dependent on accomplishing an overarching mission [49]. Accordingly, the FireCommander game environment and objectives can be summarized as follows:

- **Game environment:**
  - *Types of Agents:* Generally, In the FireCommander environment, there are three categories of agents: (1) Perception-agents (e.g., sensing or fire-observing), (2) Action-agents (e.g., manipulator or firefighting) and, (3) Hybrid-agents (e.g., agents that can both sense and manipulate). Most of the designed scenarios in the game (see Section 3.3) are focused on coordinating between uni-task Perception and Action agents, however Hybrid agents that can do both tasks are also provided for generality of the environment. Agents can be homogeneous or heterogeneous in their parameters and motion characteristics such as velocity, safe altitudes, etc. See Section 3.4 for more details.
  - *Types of Targets/Facilities:* There are six different types of targets and facilities in the game: (1) agent depot, (2) power station, (3) hospital, (4) house, (5) road and (6) lakes. Fire can propagate to any of these targets which results in an increased rate of negative-reward received, respective to the type of the target. Each target has a pre-designed size on the map. See Section 3.4 for more details.
  - *Firespots Details:* Firespots are initially invisible to the user and must be discovered by a Perception agent (e.g., or a Hybrid agents). Fire is initialized in multiple spots and propagates through time and space according to the introduced FARSITE model (see Section 2.3.2). Moreover, fire can start at anytime during the game and anywhere on the map (but not on any of the existing targets). See Section 3.4 for more details.
  - *Stochasticity in Environment:* All agents actions change the states of the environment stochastically such that: (1) Action-agents can put out the fires in their FOV according to a random probability distribution, which is designed through a confidence level coefficient. (2) We model a perception agent's altitude-dependent sensing quality (e.g., the lowest the altitude, the highest the quality of estimation) such that, perception-agents can sense (e.g., detect/locate) the fires within their FOV according to a random probability distribution which depends on their altitude. A perception agent at its lowest safe-altitude can sense 100% of the firespots in its FOV while a perception agent at its highest allowable altitude can only sense 40% of the firespots within its FOV.

  - *Battery-Life, Tanker-capacity and Motion Restrictions:* All agents have limited battery-life, upon exhaustion of which agents retreat to base depot for recharge. Moreover, Action and Hybrid agents have a limited capacity for dumping water on fire (e.g., tanker capacity). If the tanker is empty, agents will retreat to base depot for refuel. Moreover, in order to distinguish between the motion characteristics of Perception and Action agents, we assume that action agents are large, expensive devices that cannot be held inactive within the environment. As such, after driving an action agent to a location, the agent can only remain there for three seconds; afterwards, it will either retreat to base depot or will move to a new location specified by the user within the 3-second deadline. Perception agents can remain still in a location until their battery-life reaches zero.

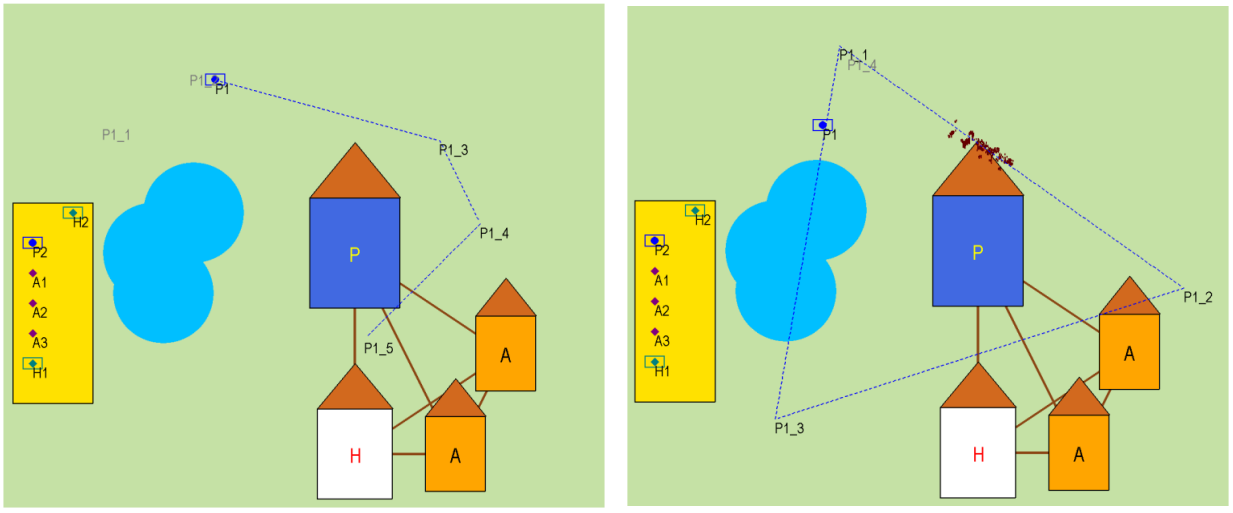
- **Game Objectives and Priorities:**
  - *Game Objectives:* The objective of the game is, given a limited time and number of heterogeneous agents, maximize the firefighting reward by (1) cooperatively putting out all propagating firespots and (2) keep the targets (e.g., facilities such as house, hospital, etc.) on the map safe from the fire.
  - *Target Priorities:* If fire propagates on any of the targets, it results in an increased rate of negative-reward received, respective to the type of the target. As such, targets have different pre-defined priorities which can be sorted as follows: (1) agent depot, (2) power station, (3) hospital, (4) house, (5) road and (6) lakes, in which agent depot and power stations are the most important targets with -5 reward per firespot per time-step that an active fire is on the target. Moreover, priorities of lakes and roads are set to zero.

2.3.4 Interacting with the Game: Inputs & Outputs

FireCommander is an interactive environment, designed such that it can be readily used for user-data acquisition in LfD research. The game environment can be interacted with and the resulting behaviours are saved on the computer in data-files, described as follows:

- **Environment Inputs:** A user must select between multiple existing agents for sensing and firefighting tasks and then pan a trajectory to drive them to desired locations or areas. As such, the environment inputs can be summarized as follows:
  - *Agent Switch:* For this purpose digit keys on the keyboard are used. Agents are indexed $1-9$ each of which can be called by pressing the corresponding key number.
  - *Planar Motion Trajectory Planning:* A computer mouse is used for this purpose by which a User can click on any point on the map to drive the selected agent to that location. Planar trajectories in FireCommander have two types: (1) Normal trajectories (Figure 6a) and, (2) Patrolling trajectories (Figure 6b). Normal trajectories simply drive an agent to a desired location while in a patrolling trajectory, an agent keeps following that trajectory to "*patrol*" the specified area until it is told otherwise or its battery-life reaches zero.
  - *Vertical Motions:* Perception and Hybrid agents can increase their altitudes to gain more information (e.g., to see a larger area), or decrease their altitude to improve their information quality (e.g., to reduce estimation errors). As such, vertical motions are also required. To this end, "*Up*" and "*Down*" arrow keys on the keyboard are used.
- **Environment Outputs:** When one round of game is finished, 13 files with **.pkl*[6] format are saved on the computer. The data-files include a comprehensive set of information regarding

[6]A PKL file (e.g., **.pkl*) is a file created by pickle, a well-known Python module.



(a) An example normal trajectory

(b) An example patrolling trajectory

Figure 6: Types of Planar trajectories in FireCommander.

the game that was played, ranging from agents trajectories, fire information, target information and etc. The recorded information are "*complete*", in the sense that a replay video of the exact game played by the user can be reconstructed from these information. See Section 3.7 and 3.6 for details of data formats and structures as well as the replay video reconstruction process.

2.3.5 Game Feedback: Score Policy & Performance Evaluation

Users receive an online score while playing the game and a general performance evaluation at the end of the game[7]. Below, descriptions of both the online score policy and the performance evaluation metrics are provided (for more details and equations please see Section 3.5):

- **Online Score System:** The online score is shown on the information box during the game. The online score consists of various Negative and positive rewards, listed as follows:
  - *Negative Rewards:* User receives two types of negative rewards: (1) a constant, cumulative reward of -0.1 (negative reward) for each fire spot that is generated, per time-step, which is designed to encourage users to act fast. (2) When a fire propagates on any of the targets, it results in an increased rate of negative-reward received, respective to the target's priority. As such, targets have different pre-defined penalty coefficients (PC) sorted as follows: (1) agent depot (PC $= -5$), (2) power station (PC $= -5$), (3) hospital (PC $= -2$), (4) house (PC $= -1$), (5) road (PC $= 0$) and (6) lakes (PC $= 0$). The penalty coefficients are applied per firespot per time-step that an active fire is on the target[8].
  - *Positive Rewards:* User also receives three different positive rewards: (1) finding firespots (e.g., Perception Score), (2) putting out firespots (e.g., Action Score) and (3) keeping targets/facilities safe (e.g., Facility Protection Score).
- **Final Performance Evaluation:** When a round of game ends, a final performance evaluation

[7]Please note that most of the score policies and performance evaluation metrics are subject to change based on the underlaying application. For instance for some applications the online score might not be required, or a metric might be interesting in a setup that is not provided here, or the verbal performance evaluation might be needed to a fully quantitative system. As such, while we provided a somewhat general score and evaluation policy, users are encouraged to modify the online score and final performance evaluation policies according to their application of interest.

[8]The penalty coefficients can be changed if required.

Grade	Final Score
Failed	< 50
Fair	50 - 60
Almost There!	60 - 80
Well Done	80 - 90
Excellent	>90

Figure 7: The "*General Evaluation*" verbal feedback system.

page is displayed (see Figure 24 as an example) to the user. In this page, an "*Overall Firefighting Performance Score*" is reported as the ratio of number of firespots that have been put out over the total number of firespots in the map (observed and unobserved). Moreover, the perception and action scores as well as the facility protection score and the total negative reward from the online score panel are also reported on this page. Eventually, a "*General Evaluation*" is reported as a verbal feedback, which is designed

2.4 How-Tos and Tips

2.4.1 How to Generate a Coordination Policy?

A coordination policy can take various forms based on the underlying application. However, generally speaking, in a multi-agent system a policy should include actions of all agents during the time of the game, given the states of the environment and the agents. In FireCommander environment we record all of the environment and agents' states as PKL files. As such, in order to generate a coordination policy the comprehensive set of recorded data can be used. See Section 3.7 for details.

2.4.2 How to Deploy an Existing Coordination Policy?

When the coordination policy, as defined in the previous section, exists, it means that no interaction is required with the environment and we only wish to deploy the existing policy (as a set of data) to visualize the environments and the agents' behaviors. In FireCommander an existing teacher policy (as a set of data) can be deployed on the environment (e.g., visualized) through the function "`Demo_Visualization.py`". See Section 3.6 for details of this process.

2.4.3 How to Design a New Scenario?

We have created 24 pre-designed scenarios in three categories of easy, moderate and hard. These scenarios are designed based on the requirements in our own multi-agent learning from heterogeneous demonstrations study [RF[9]] which can also be used in specific HRI and human-robot teaming experiments. As such, one might need to modify these scenarios based on the specific requirements in their own study. For this purpose, the design-parameters for these scenarios are gathered in a separate function, named "`Scenario_Mode_Params.py`", and can be accessed easily by users for further modifications. See Section 3.3 for more details on the pre-designed scenarios.

[9]Reference to be added.

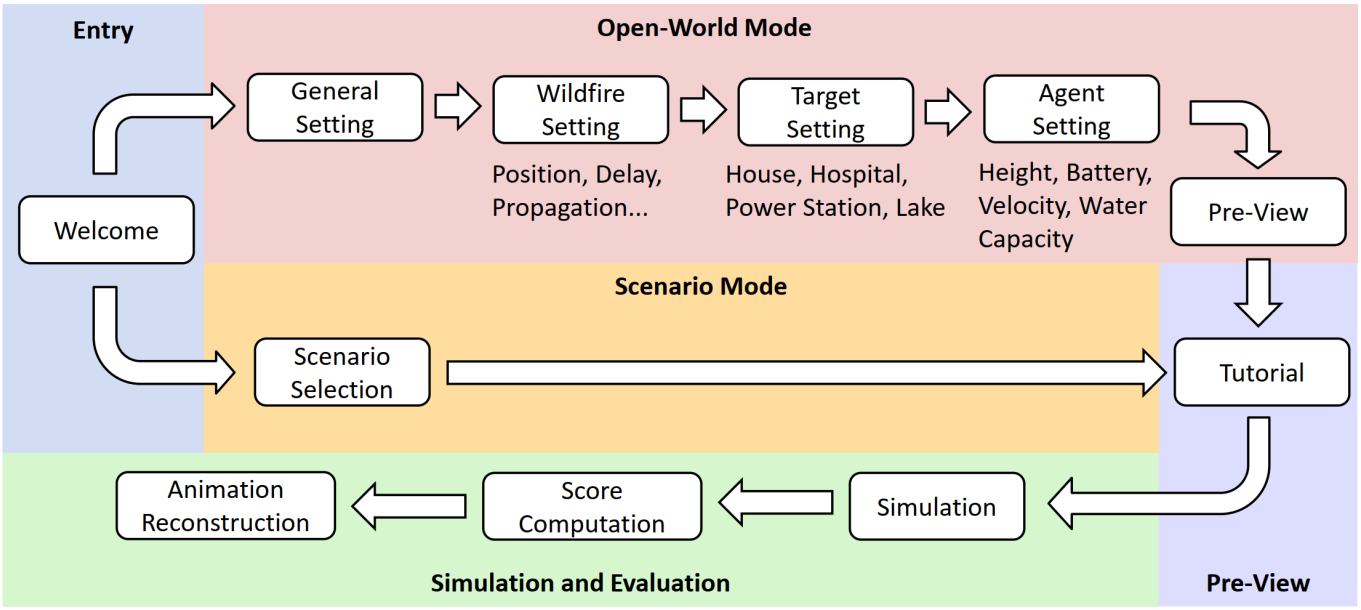


Figure 8: The Graphical User Interface (GUI) structure.

2.4.4 How to Generate Expert Data for LfD Problems?

A user's use of clicks and keyboard buttons represents their "*expert*" actions. These actions can be put-together with the environment and the agents' states through time to generate a list of expert-actions and environment states. These state-action pairs in the created list form an example trajectory (e.g., demonstration), which is the input data to the LfD algorithm.

3 GUI Reference Manual

The basic motivation for designing a graphical user interface (GUI) is to offer users a convenient way to enter existing scenarios or design their own scenarios. Moreover, in related HRI and Psychology studies, subjects need to interact with the game environment and apply changes online. To make the game design concise and interactive, our GUI helps the users design scenarios step by step, from the general number of each targets, to the their exact locations and orientations, as well as the operating parameters for some specific elements of the environment like agents and wildfire areas.

Moreover, another purpose for the GUI is to easily collect subject (e.g., player) data. The pre-determined scenarios in the Scenario Mode are divided into three genres: **Easy**, **Moderate** and **Hard**, based on the overall complexity of the scenario. We also provide a simple practice scenario to help the novice player get accustomed to the GUI and controlling agents.

3.1 GUI General Structure

Figure 8 shows the general structure of the GUI. As the first page in the GUI, the welcome page offers a choice between the open-world mode, which allows the users to design a new scenario, and the scenario mode in which users play an existing scenario (e.g., similar to a mission in other common strategic games). For the open-world mode, the design process covers the following parts:

- **General Setting:** Number of each environment element (e.g., agents, firespots, facilities, etc.)
- **Wildfire Setting:** Locations and propagation parameters of each fire region.

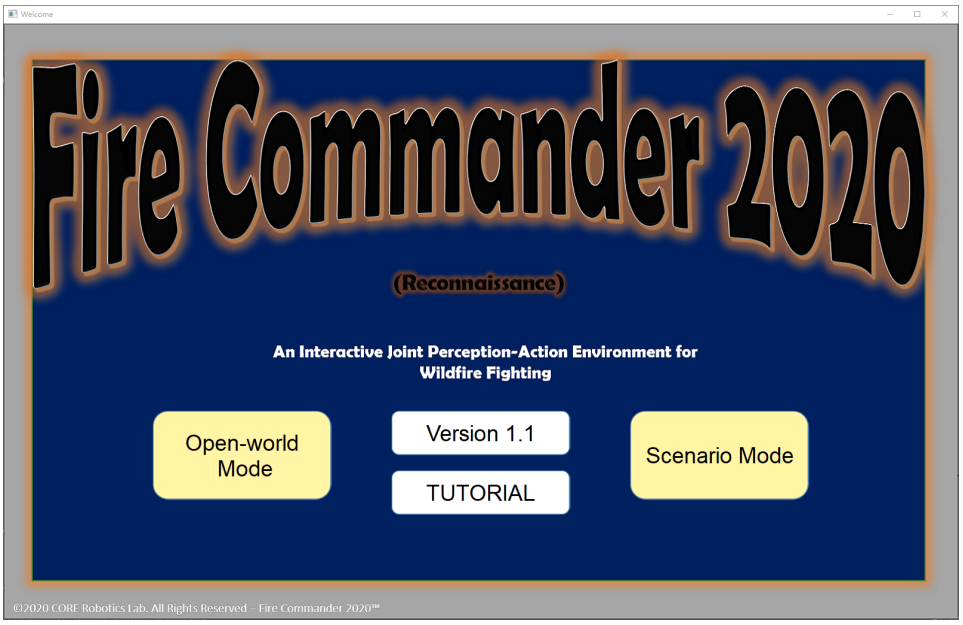


Figure 9: The welcome screen.

- **Target/Facilities Setting:** Locations and orientation (only for agent base) for each of facilities and targets, including the agent base, house, hospital, power station and lake.
- **Agent setting:** Agents specific parameters and characteristics such as the flight height, battery, velocity and water capacity.
- **Preview:** Visualizing a static version of the designed scenario with the current parameters to user for final confirmation before starting the game.

Users don't need to define parameters manually in the scenario mode. Players and subjects of the scenario mode can enter the game by choosing a scenario among a list of possible 24 pre-designed scenarios. After a game ends (either in open-world mode or scenario mode), a score-board of each trial is represented as a report of the user's performance in successfully accomplishing game's objectives. Users could choose to exit the game, to reconstruct a replay animation of their most recent game played as a video data, or to return to the welcome or scenario mode page.

3.2 Welcome Screen

Figure 9 represents the welcome screen in FireCommander GUI. the welcome page offers a choice between the Open-World Mode, which allows the users to design a new scenario, and the Scenario Mode in which users play an existing scenario (e.g., similar to a mission in other common strategic games). Here are the details of the available options on the welcome screen:

- **Scenario Mode:** Users will start the game directly with the existing parameters. The available scenarios include the 24 pre-designed scenarios, categorised from easy to hard, and a practice scenario. Player's data generated during playing any of the scenarios will be recorded.
- **Open-world Mode:** Users can design their own scenarios by specifying necessary parameters for elements of the environment in a step-by-step procedure. Player's data generated during playing a user-designed scenario will also be recorded.
- **Tutorial:** The tutorial page serves as a list of instructions on how to work with the GUI and play the game.

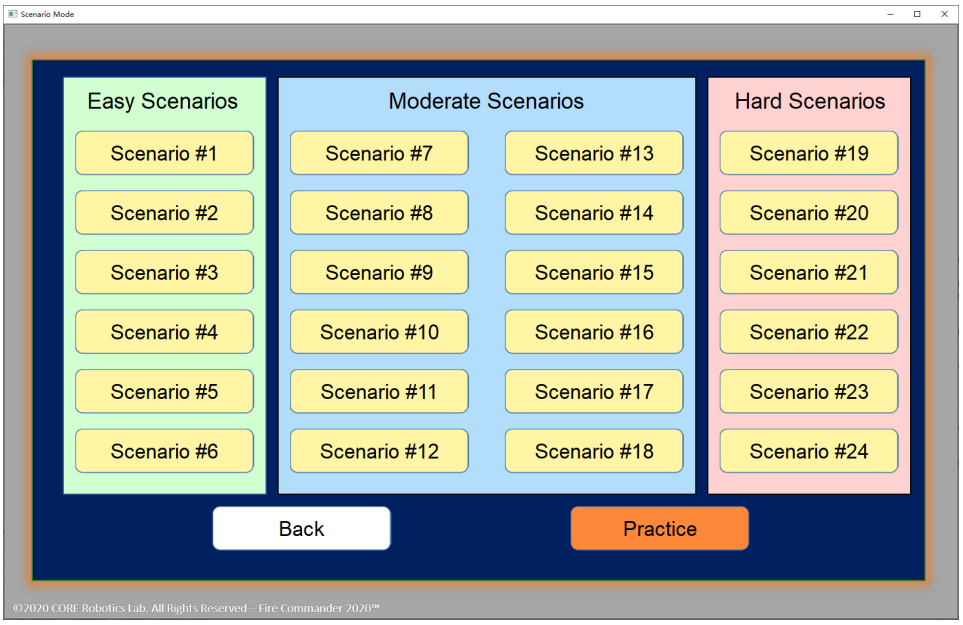


Figure 10: The scenario mode screen.

3.3 Scenario Mode

Figure 10 shows the scenario mode screen. In the scenario mode, all the formal scenarios are divided into three difficulty groups based on the complexity of the scenario. The pre-designed scenarios are designed be diverse while also sharing some aspects in order to achieve both generalization and discrimination. Moreover, to help users practice and get used to the environments and game controls, we also provided a simple practice scenario. User's data while playing the practice scenario will not be recorded. Details of each of these 24 scenarios, as well as the practice scenario is presented in the following sub-sections.

Practice Scenario: The purpose of the practice scenario is to help the novice users familiarize with the environment and game controls before attempting to play the pre-designed scenarios. The practice scenario incorporates the online score and final performance display, while the data storage function is disabled for this tutorial scenario. Once the game ends, users will return to the scenario mode page and their data and score for this trial will not be stored. Table 2 and 3 present the target, agent and wildfire setting in the practice scenario.

Scenario Index	**# of Targets**				**# of Agents**	
	House	Hospital	Power Station	Lake	Perception	Action
Practice	1	1	1	1	2	2

Table 2: The Targets and Agent Setting in the Practice Scenario

Scenario Index	**Region-wise # of Fire Front**	**Fire Delay (Sec)**	**Fuel Coefficient**	**Wind Speed**	**Wind Direction**
Practice	15	0	10	5	45

Table 3: The Fire Setting in the Practice Scenario

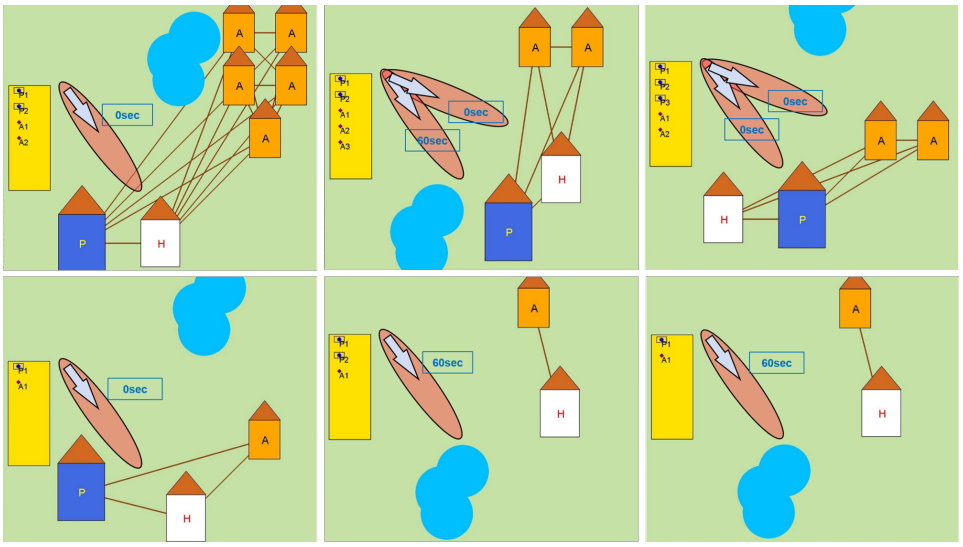


Figure 11: The Easy Scenarios (Scenario #1 – #6)

Formal Pre-designed Scenario Categories: The formal scenarios are divided into easy, moderate and hard scenarios, based on the complexity of the scenario setting. Scenario categories are divided based on number of targets/facilities, number of agents and wildfire model parameters such as the firefront spots, fire delay, fuel coefficient, wind speed and wind direction[10]. Below, we elaborate the details for each scenario:

- **Easy Scenarios (#1 – #6):** Easy cenarios include the Scenario #1 – #6, and their respective settings are presented in Figure 11. Tables 4 and 5 present the target/agent and the fire settings for the easy scenarios. For the cases that multiple fire regions exist in the same scenario, parameters defining different fire regions are separated by a comma.

Scenario Index	# of Targets				# of Agents	
	House	Hospital	Power Station	Lake	Perception	Action
#1	5	1	1	1	2	2
#2	1	1	0	1	2	1
#3	1	1	1	1	1	1
#4	1	1	0	1	1	1
#5	2	1	1	1	3	2
#6	2	1	1	1	2	3

Table 4: The Targets and Agent Setting in the Easy Scenario (Scenario #1 – #6)

- **Moderate Scenarios (#7 – #18):** Moderate scenarios include the Scenario #7 – #18, and their respective settings are presented in Figure 12 and Figure 13. Table 6 presents the facility/target and agent setting and Table 7 presents the fire setting for the moderate scenarios. For the cases that multiple fire regions exist in the same scenario, parameters defining different fire regions are separated by a comma.
- **Hard Scenarios (#19 – #24):** Hard scenarios include the Scenario #19 – #24, and their respective settings are presented in Figure 14. Table 8 presents the facility/target and agent setting and Table 9 presents the fire setting for the hard scenarios. For the cases that multiple

[10]Wind direction is the counter-clockwise angle between the fire's moving direction and the vertical axis.

Scenario Index	Region-wise # of Fire Front	Fire Delay (Sec)	Fuel Coefficient	Wind Speed	Wind Direction
#1	10	0	10	5	45
#2	15	60	15	3	45
#3	5	0	15	5	45
#4	12	60	5	3	45
#5	3, 8	0, 0	5, 10	5, 3	45, 15
#6	5, 3	60, 0	10, 10	5, 5	45, 15

Table 5: The Fire Setting in the Easy Scenario (Scenario #1 – #6)

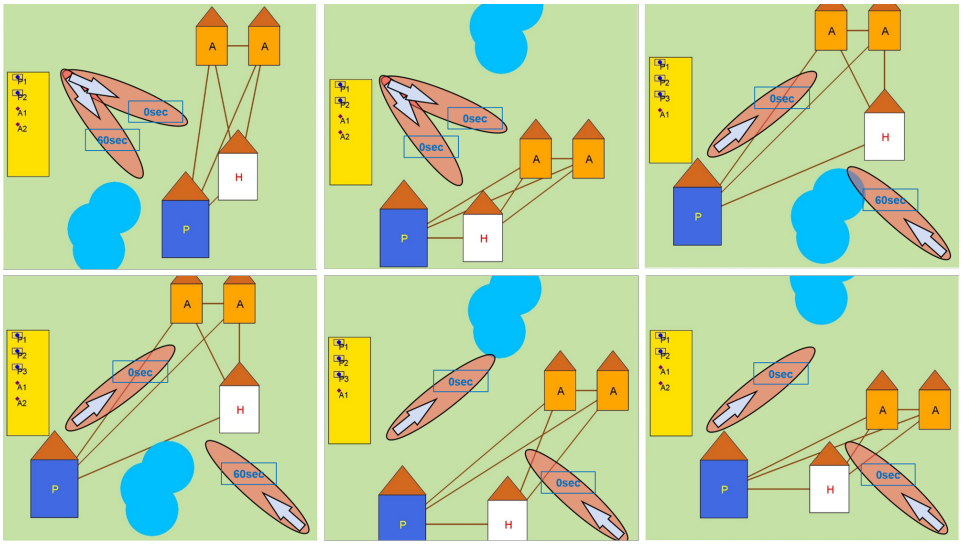


Figure 12: The first batch of Moderate Scenarios (Scenario #7 – #12)

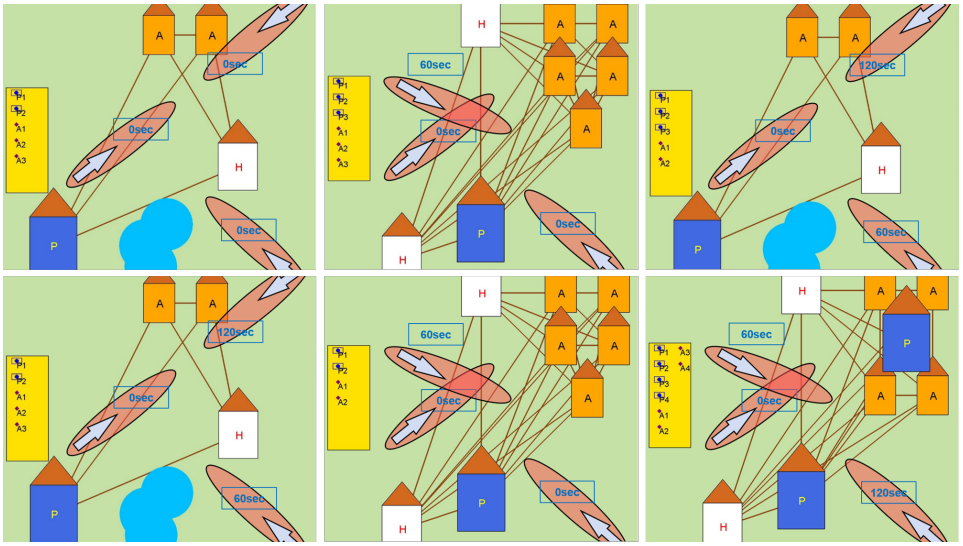


Figure 13: The second batch of Moderate Scenarios (Scenario #13 – #18)

fire regions exist in the same scenario, parameters defining different fire regions are separated by a comma.

Scenario Index	# of Targets				# of Agents	
	House	Hospital	Power Station	Lake	Perception	Action
#7	2	1	1	1	3	1
#8	2	1	1	1	3	1
#9	2	1	1	1	2	2
#10	2	1	1	1	2	2
#11	2	1	1	1	2	2
#12	2	1	1	1	3	2
#13	5	2	1	0	2	2
#14	2	1	1	1	3	2
#15	5	2	1	0	3	3
#16	2	1	1	1	2	3
#17	4	2	2	0	4	4
#18	2	1	1	1	2	3

Table 6: The Targets and Agent Setting in the Moderate Scenario (Scenario #7 – #18)

Scenario Index	Region-wise # of Fire Front	Fire Delay (Sec)	Fuel Coefficient	Wind Speed	Wind Direction
#7	3, 3	0, 0	5, 5	5, 5	135, 225
#8	5, 7	0, 60	3, 3	10, 10	135. 225
#9	5, 5	0, 0	10, 10	5, 5	45, 15
#10	5, 5	60, 0	10, 10	5, 5	45, 15
#11	5, 5	0, 0	5, 5	5, 5	135, 225
#12	3, 10	0, 60	5, 10	5, 10	135, 225
#13	3, 3, 3	60, 0, 0	10, 10, 10	5, 5, 5	75, 135, 225
#14	5, 5, 5	0, 120, 60	10, 10, 10	5, 5, 5	135, 315, 225
#15	3, 5, 7	60, 0, 0	10, 10, 10	3, 5, 10	75, 135, 225
#16	3, 5, 7	0, 120, 60	3, 5, 10	5, 5, 5	135, 315, 225
#17	5, 5, 5	60, 0, 120	10, 10, 10	5, 5, 5	75, 135, 225
#18	3, 3, 5	0, 0, 0	5, 5, 10	3, 3, 5	135, 315, 225

Table 7: The Fire Setting in the Moderate Scenario (Scenario #7 – #18)

Scenario Index	# of Targets				# of Agents	
	House	Hospital	Power Station	Lake	Perception	Action
#19	4	2	2	0	2	2
#20	2	1	1	1	2	1
#21	4	1	2	2	4	4
#22	4	1	2	2	3	3
#23	4	1	2	2	3	2
#24	4	1	2	2	2	2

Table 8: The Targets and Agent Setting in the Hard Scenario (Scenario #19 – #24)

3.4 Open-world Mode

The open-world mode is provided as a tool for users to design their own scenarios, in a step-by-step procedure. In each step, the parameters required for specific objects in the environment (such as

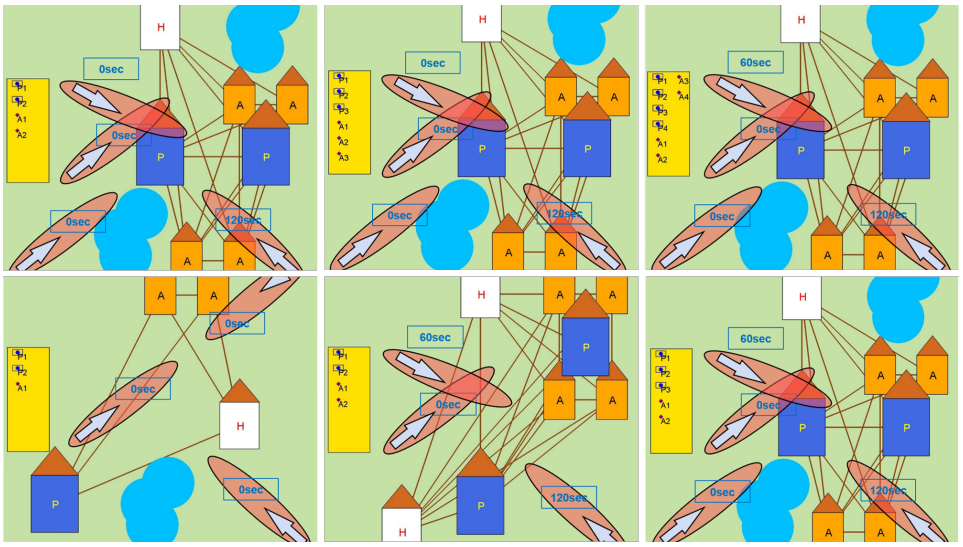


Figure 14: The Hard Scenarios (Scenario #19 – #24)

Scenario Index	Region-wise # of Fire Front	Fire Delay (Sec)	Fuel Coefficient	Wind Speed	Wind Direction
#19	5, 5, 5	60, 0, 120	10, 10, 10	5, 5, 5	75, 135, 225
#20	3, 3, 3	0, 0, 0	5, 5, 5	5, 5, 5	135, 315, 225
#21	5, 5, 5, 5	60, 0, 0, 120	10, 10, 10, 10	5, 5, 5, 5	75, 135, 135, 225
#22	5, 5, 5, 5	0, 0, 0, 120	5, 5, 10, 10	5, 5, 5, 5	75, 135, 135, 225
#23	3, 3, 5, 7	60, 0, 0, 120	3, 5, 3, 10	5, 5, 5, 5	75, 135, 135, 225
#24	3, 5, 7, 8	0, 0, 0, 120	8, 8, 8, 8	3, 3, 3, 3	75, 135, 135, 225

Table 9: The Fire Setting in the Hard Scenario (Scenario #19 – #24)

agents, targets/facilities and even wildfire) are asked to be inserted by the user, through a delicate, user-friendly GUI. Each section of the Open-World Mode is elaborated in the following Sections.

3.4.1 General Setting Page

Represented in Figure 15, the general setting page serves as the entry page of the open-world mode and defines the basic parameters for the user-designed scenario, such as the number of each element in the environment. These elements can be specified on the left-side of the general setting page. The environment setup section include parameters such as the world size, the scenario duration and the number of each kind of targets, including the fire's initial areas, houses, hospitals, power stations and lake. Table. 10 presents the constraints and default values on these parameters. The world size offers three choices: small (800×800 grid world), medium (1000×1000 grid world) and large (1200×1200 grid world). The duration of the scenario offers three choices as well: short game (60 Seconds), medium-duration game (120 Seconds) and long game (180 Seconds). To avoid extremely dense environments, the number of each element cannot be more than five, while their minimum numbers can be set to zero (expect for the number of fire areas which has a minimum of one region allowed). On the right-side of the general setting page a brief instruction on how to use the open-world mode is provided and on the bottom of the page, there is the Back button to return to the welcome page, the Reset button to clear all the inputs and the Next button to proceed to next page.

The robot team setup section include parameters such as the number of each type of agents, e.g.,

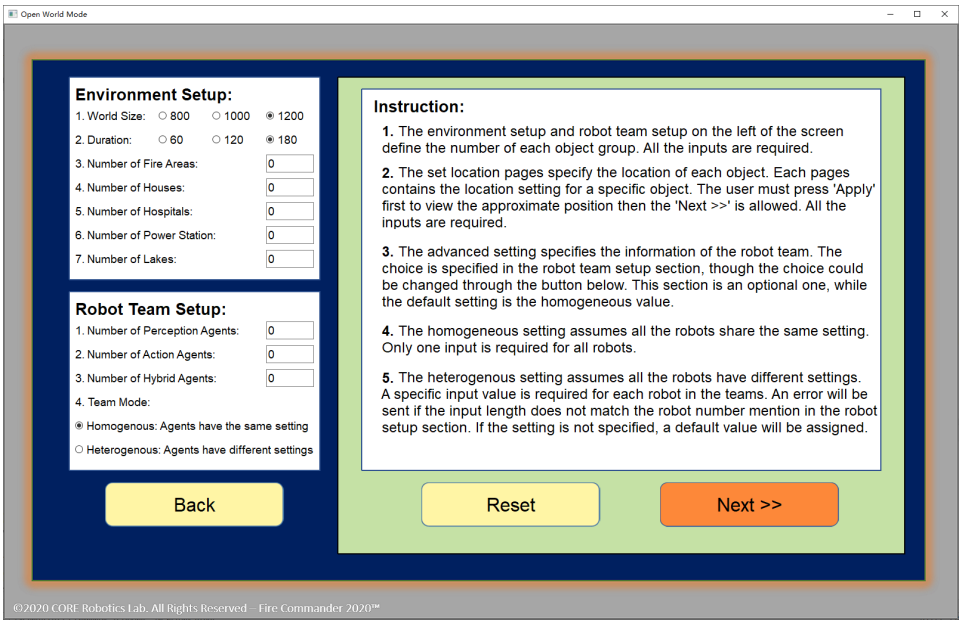


Figure 15: The General Setting Page.

Perception, Action and Hybrid agent, as well as a choice of homogeneous or heterogeneous modes for agents of the same type (e.g., agents of the same type have similar or different characteristics such as the velocity, battery capacity, etc.). The constraints on the robot team composition are as follow:

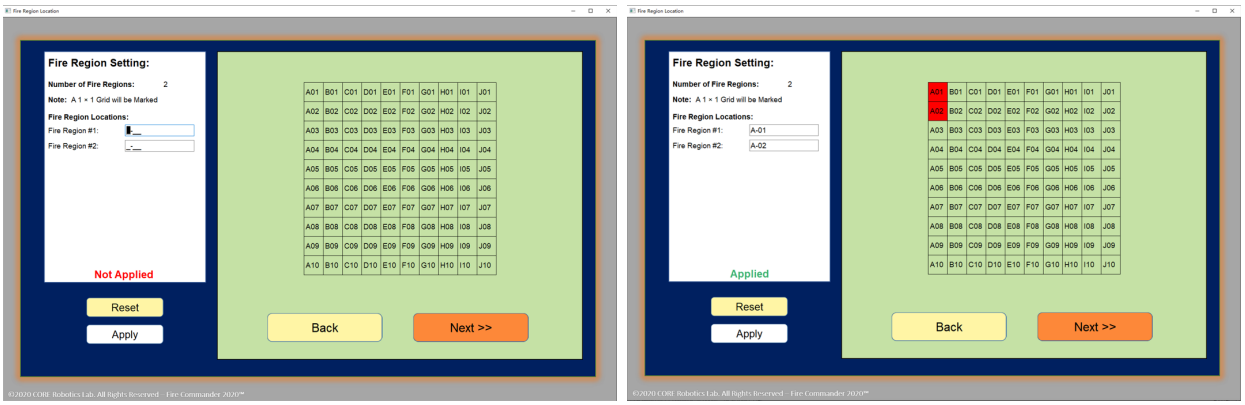
- Since switching between the agents is made by pressing 1-9 digit keys on the keyboard, the sum of all agents cannot exceed nine. This restriction is only for the GUI version and the MARL package does not enforce any restriction on the number of agents.
- The composition of the robot team should include at least one agent that can perform perception and at least one agent for taking action. As such, when the number of Hybrid agents is set to zero, the minimum number of Perception and Action agents will be one.

Target Description	Limitations (Range or Acceptable Value)	Default Value
World Size	800/1000/1200 (grid)	1200
Duration	60/120/180 (Sec)	180
Fire Region	1 – 5 (grid)	0
Agent Base	1 (#)	1
House	1 – 5 (#)	0
Hospital	1 – 5 (#)	0
Power Station	1 – 5 (#)	0
Lake	1 – 5 (#)	0

Table 10: The constraints and limitations on map elements and objects.

3.4.2 Wildfire Setting Page

Wildfire settings are divided into two separate pages. First, users need to define the initial location of the fire regions on the map, and then, specify the details of the fire propagation model parameters and penalty coefficients on a separate page. Figures 16a and 16b present the structure of the fire region location setting page. On the left-side of the page user can specify the location of each fire area, by choosing one of the grids depicted on the right-side panel, which follows a chess-like naming



(a) Before Apply (b) After Apply

Figure 16: The Fire Region Setting Page

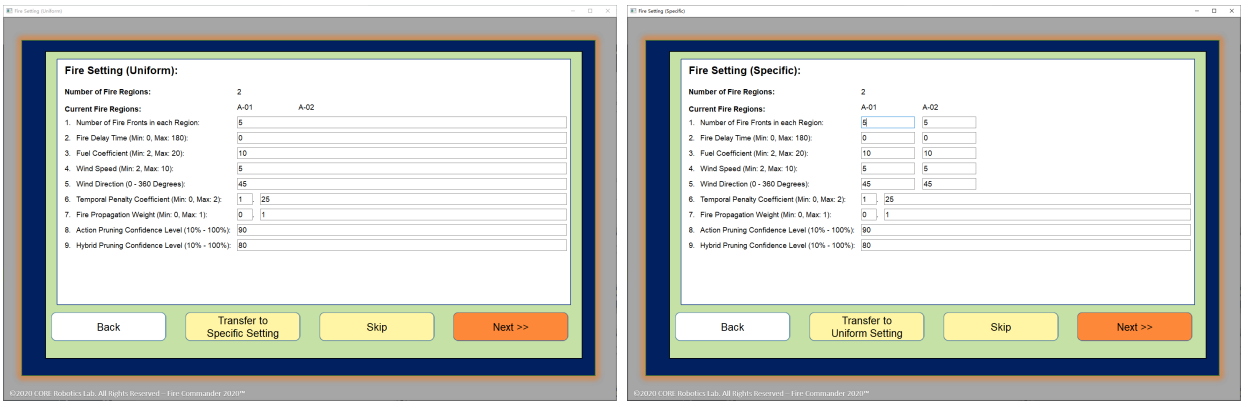
convention. Each initial fire region occupies a 1×1 grid on the right-side grid-map. As an instance, the fire region on 'D-05' represents a fire area in the actual environment that centers at (350, 450). Note that, while specifying the initial fire areas, two or more fire regions could overlap (e.g., start at the same location).

After all the fire region locations have been set, users need to press the Apply button to visualize the intended fire region on the grid map. The initiated fire regions will be marked with red squares on the selected grid. Once the apply button is pressed and all the regulations on the fire regions have been satisfied, the "**Not Applied**" flag will turn into "**Applied**". If users are not satisfied with the current fire region, they could press the Reset button to restart the design.

The grid-map on the right-side of the screen will include an approximate visualization of all facilities/targets as well so that users can track their scenario design. Different types of facilities/targets are marked with different colors. The colors used to represent different targets and their respective sizes are listed below.

- **Green**: Grassland, the default element on the map
- **Red**: Fire region, 1×1 Grid
- **Yellow**: Agent base, 4×2 Grid (Vertical) or 2×4 Grid (Horizontal)
- **Orange**: House, 2×2 Grid
- **White**: Hospital, 2×2 Grid
- **Dark Blue**: Power Station, 2×2 Grid
- **Light Blue**: Lake, 4×3 Grid

The next step in the wildfire setting is to specify the fire propagation model parameters as well as some penalty coefficients. In our GUI, we offer two options: (1) the uniform setting in which all fire regions use the same model parameters and, (2) the specific setting in which fire regions use separate model parameters, each defined individually. The penalty coefficients, which related to agents' performance and how their rewards are going to be calculated, are always defined globally. In fire setting page, there are five inputs for the fire propagation model parameters and two for the penalty coefficients. In the specific setting, users only need to fill in such fields for each separate fire region. Tables 11 and 12 present the parameters in this page (fire model parameters and the two score-related penalty coefficients) and their constraints.



(a) Uniform Fire Setting (b) Specific Fire Setting

Figure 17: The Fire Setting Page

Parameter Description	Limitations (Range or Acceptable Value)	Default
Region-wise # of Firefronts	1 – 30 (#)	10
Fire Delay Time	0 – 180 (Sec)	0
Fuel Coefficient	2 (unit)	10
Wind Speed	2 – 10 (m/s)	5
Wind direction	0 – 360 (θ)	45

Table 11: The constraint and limitations on wildfire model parameters.

Description	Limitations (Range or Acceptable Value)	Default
Temporal Penalty Coefficient	0 - 2	1.25
Fire Propagation Weight	0 - 1	0.1

Table 12: The Constraint on Wildfire Propagation Score

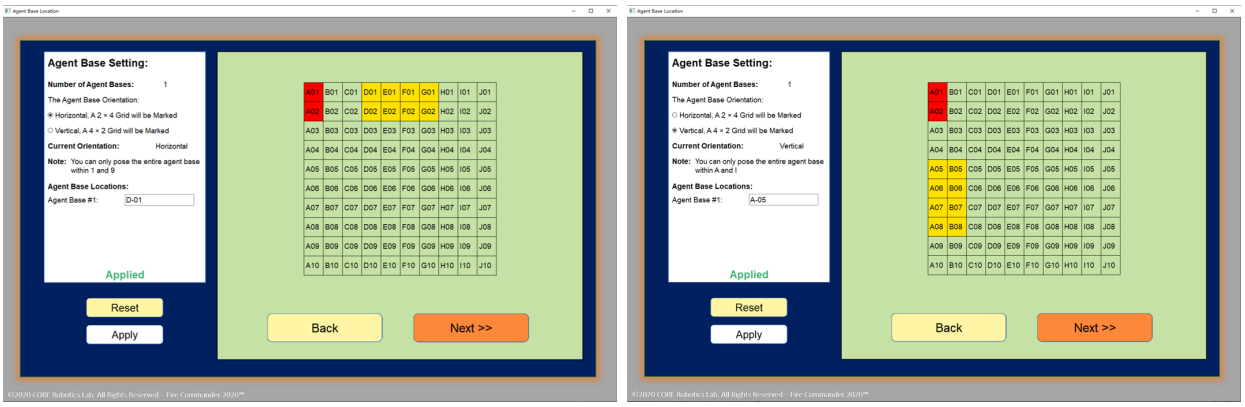
We also incorporated in this page, the pruning confidence level for the Action and Hybrid agents, which relates to the stochasticity in the environment, as introduced in Section 2.2.3, and is fixed during the game. Table 13 presents the limitations and default value for the pruning confidence level. The pruning confidence level in all the pre-designed scenarios (in Scenario Mode) is set to 90% for Action agents and 80% for Hybrid agents, while users could choose any value between 10% and 100% for the Action and Hybrid agents in the Open-World Mode. Users can switch between the uniform and specific fire setting pages by choosing either Transfer to Specific Setting or Transfer to Uniform Setting, or choose Skip and use the pre-determined values for parameters.

Description	Limitations (Acceptable Range)	Default
Action Pruning Confidence Level	10% - 100%	90%
Hybrid Pruning Confidence Level	10% - 100%	80%

Table 13: The Constraint on Pruning Confidence Level for Action and Hybrid Agents

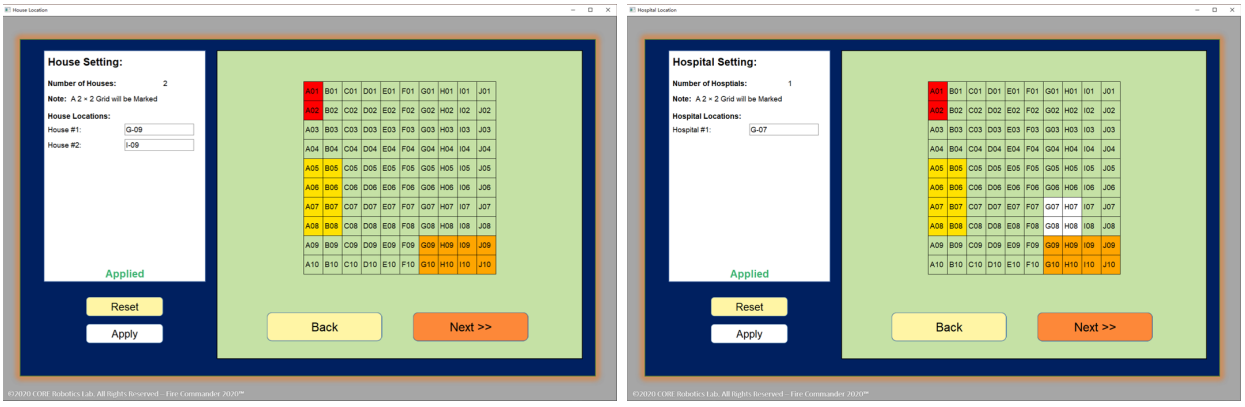
3.4.3 Facilities/Targets Setting Page

In the facility/target location design process, the first step is to specify the location and orientation of the agent base. Figure 18 presents the agent base setting page. Agent base is unique in all the pre-designed scenarios (in Scenario Mode), while its orientation and location could be changed in

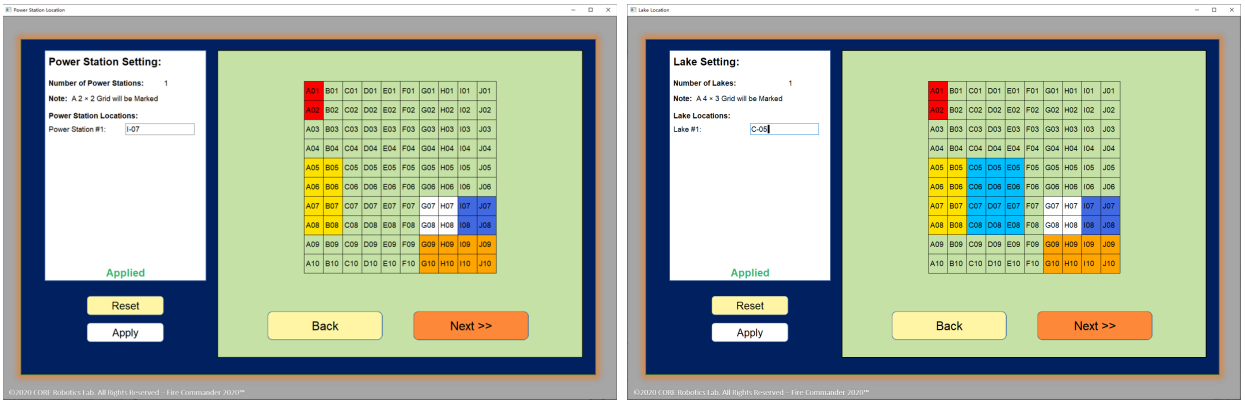


(a) Horizontal (b) Vertical

Figure 18: Agent Base (e.g., Depot) Setting Page



(a) House (b) Hospital

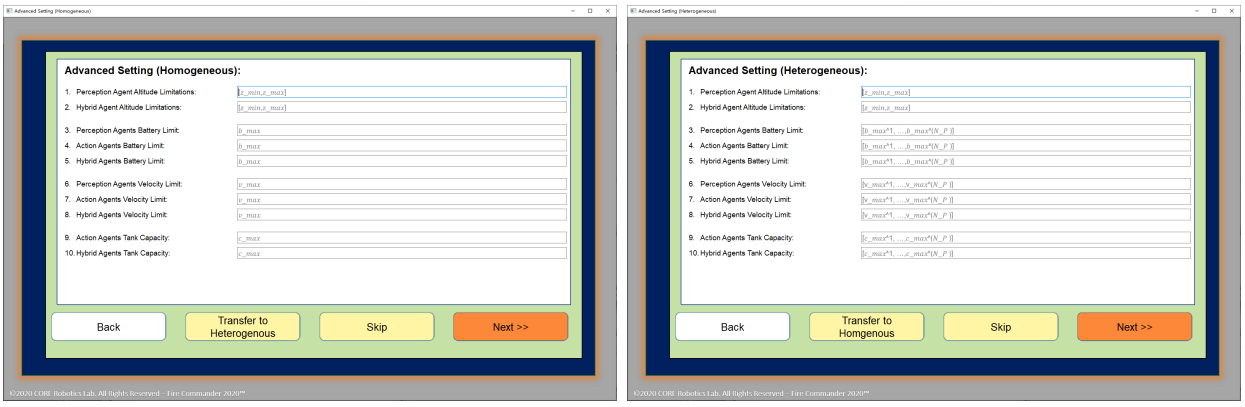


(c) Power Station (d) Lake

Figure 19: Target Setting Page

the Open-World Mode. Note that, the top-left grid of the agent base must be selected such that the entire base fits inside the map and it does not overlap with any of the existing facilities/targets. Two options are allowed for the base orientation:

- **Horizontal:** The agent base occupies 2×4 on the map. In this case, the agent base must be located either on top or bottom edge of the map.
- **Vertical:** The agent base occupies 4×2 on the map. In this case, the agent base must be located either on left or right edge of the map. Vertical is the default option.



(a) Homogeneous Agent Setting Page (b) Heterogeneous Agent Setting Page

Figure 20: Agent Setting Page

After the agent base setting page, the rest of the facility setting pages, such as house, hospital, power station and lake, will be shown to the user with relatively similar structures. The sizes of these facilities on the grid-map are 2×2 for house, hospital and power station, and 4×3 for lake. Similar constraints such as no overlapping facilities and fitting inside the map apply to all facilities.

3.4.4 Agent Setting Page

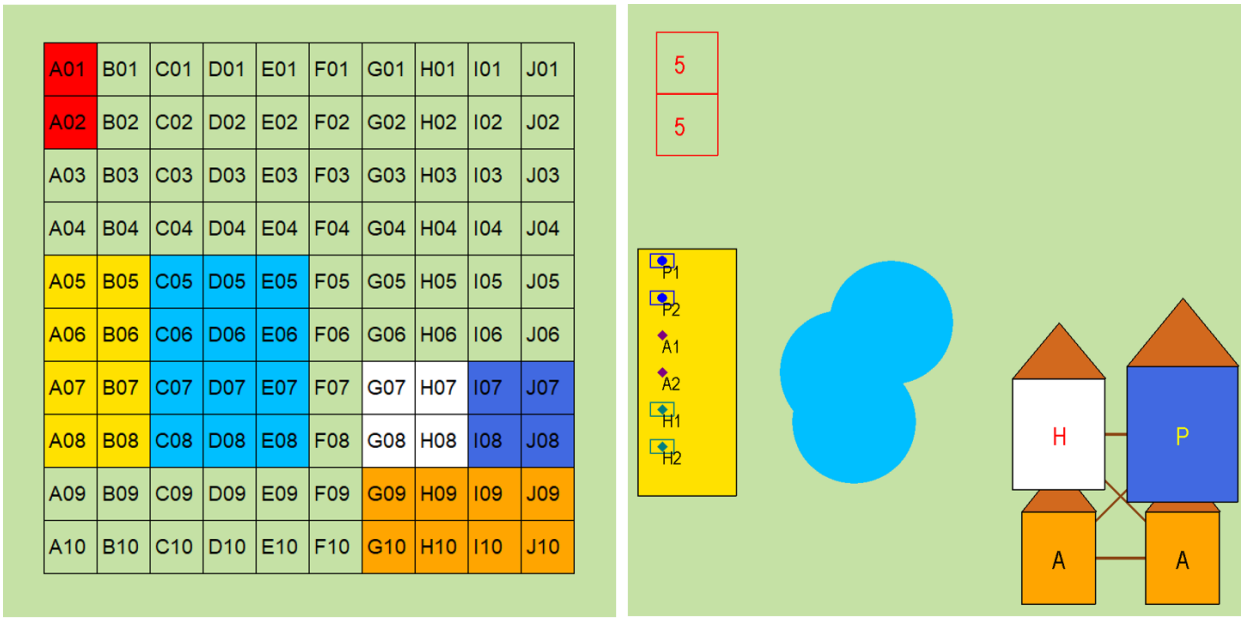
Figure 20 presents the agent setting page that specifies the agents' characteristics and control parameters. The agents' parameters include:

- **Altitude Limit:** The upper and lower bound of the agents' flight altitude, which is an available option for Perception and Hybrid agents only. The flight height for the Action agents is fixed.
- **Battery-Life Limit:** The battery capacity that determines the maximum flight duration.
- **Velocity Limit:** The maximum velocity for each agent, which is the default step size of the respective agent that builds its trajectory.
- **Tanker Capacity Limit:** The number of pruning times that an Action or a Hybrid agent could could attempt during a single flight.

There are two parameter modes for the agents: (1) homogeneous parameters and (2) heterogeneous parameters. Homogeneous mode applies a similar set of parameters to all agents of the same type (i.e., all Perception agents), while the heterogeneous mode allows heterogeneity in aforementioned characteristics of agents of the same type. In homogeneous case, parameters can be entered as a single value or tuple (as shown with gray text inside the boxes) while in the heterogeneous case, parameters are inputted as lists, each element of which corresponds to one agent. Table. 14 presents the default values for the described parameters. Note that, the heterogeneous mode will automatically switch to homogeneous mode, if the parameter boxes are left empty.

Description	Limitations (Range or Acceptable Value)	Default Value
Agent Altitude	10 – 100 (m)	10 – 100
Battery Capacity	200 – 800 (time steps)	500
Velocity	10 – 30 (m/s)	20
Water Tank Capacity	1 – 15 (#)	10

Table 14: The Constraint on Agent Setting



(a) Grid Map (b) Preview Page

Figure 21: Preview Page

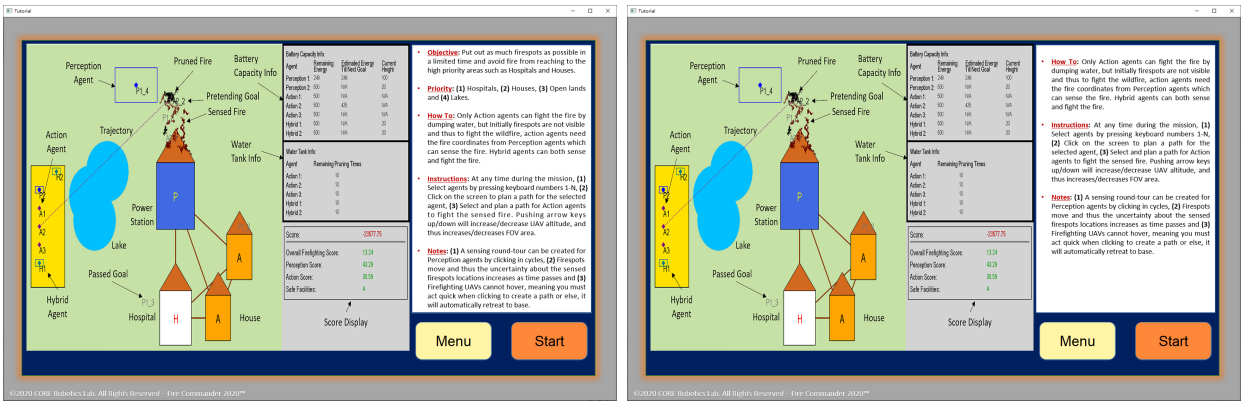
Users can define all agents parameters and then proceed to preview their design by pressing Next, or use default values for these parameters by pressing Skip. Moreover, users are allowed to leave a few inputs blank (i.e., only specify altitude and battery limits and leave velocity and tanker capacity limits blank), in which case, the GUI will automatically fill up these inputs with default values.

3.5 Preview, Tutorial, Information and Score Display Pages

After all the parameters have been specified in the previous steps, users should be prepared to start a new game. In the Open-World Mode, users are first presented with a visualization of the scenario they just designed in the Preview Page, and then, a Tutorial Page appears to help users' with a better understandings of the game logistics and objectives. During a game, an online Information Display Screen presents some information regarding the environment and the agents as well the game score at current time of the game to the player and eventually, after finishing a game, the player's performance in different aspects of the game will be calculated and reported in the Performance Evaluation Screen. Details of each aforementioned page is presented in the following.

Preview Page: The preview page appears when users choose to proceed with their design after specifying all environment parameters. Figure 21 provides the mapping from the grid map in which users designed their desired scenario and the actual map in the game environment. Note that preview page only shows a static image of the game environment for user's confirmation of a satisfactory design. For agents, the preview page marks their ID and initial positions on the agent base. For wildfire regions, the preview page marks their position and number of initial firespots in each region. Once the user closes the preview page, a dialog box pops up with the question: "*Do you like to proceed with this environment design?*". Users could either choose to proceed or return to the previous design pages.

Tutorial Page: If users confirm the design scenario in the preview page, the GUI proceeds to a tutorial page. Figure 22a shows the tutorial page for the Open-World Mode, while Figure 22b shows the tutorial page for the Scenario Mode. The difference between the two tutorial pages is that in



(a) Tutorial for Open-world Mode (b) Tutorial for Scenario Mode

Figure 22: Tutorial Page

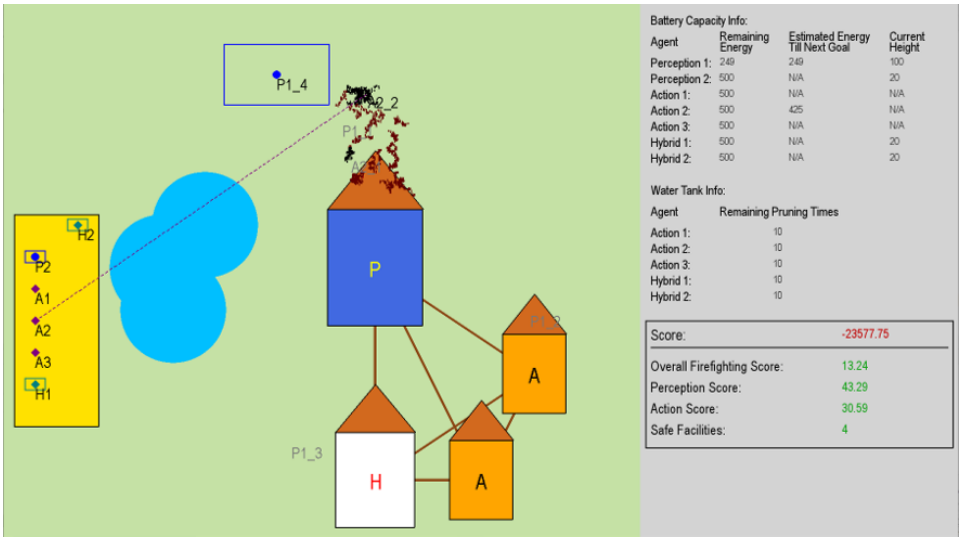


Figure 23: The Information Display.

scenario mode, we avoid giving direct objective instructions to the users and we expect the LfD framework to determine these objectives automatically from user behaviors.

Information Display: When a game is running, the necessary information are presented to the player on an information display screen on the right-side of the environment. Figure 23 presents the layout of the information display during a game. On the top of the display bar the battery capacity information for each agent at current time step is presented. The battery capacity info table includes the following information:

- **The current remaining energy in the battery:** Once the agent departs from the base, its battery begins to drain. The energy consumption rate when the agent is running is currently set to 0.1 per unit length, while the consumption rate is 0.05 for each iteration.
- **The estimated remaining energy for agent when at goal:** This value shows the estimated remaining battery energy for the last node the player planned for the current agent. This value helps users plan and choose between agents strategically so that none of agents runs out of energy mid-flight during a mission (e.g., a planned trajectory).
- **Current flight height:** Current flight altitude of an agent has a direct relation with the size of its FOV, and thus, is presented to players for planning.

Under the battery capacity info table, the water tank capacity information table is presented which show the remaining tanker capacity (e.g., remaining number of times an agent can prune firespots) for the Action and Hybrid agents. Next, on the bottom of the information display screen, the online performance measures are presented to the player. The first score shown is the current total negative reward, which is always a negative value, determined by how much time has passed and how much fire has grown since the beginning of the game, as show in Equation 7

$$\begin{aligned} Total\ Negative\ Reward = 0.1 \times Number\ of\ Active\ Firespots \\ + Penalty\ Coef + Number\ of\ Firespots\ in\ Targets \end{aligned} \tag{7}$$

Elements in Equation 7 are described below:

- *Penalty Coefficients:* 0.1 per new firespot, 1 per House, 2 per Hospital, 5 per Power Station and 5 per Agent Base. These values can be changed in the open-source code, if required.
- *Active Firespots:* All firespots that have not been pruned yet (e.g., both sensed and not-sensed firespots are counted and the pruned firespots are excluded).
- *Firespots in Targets:* The active firespots (sensed or not-sensed) in each facility/target.

Under the total negative reward, there are four more performance metrics, which cover the following:

1. **Overall Firefighting Score:** The ratio between the number of the active firespots and the total number of firespots that have been generated so far in the game (sensed or not-sensed):

$$Overall\ Firefighting\ Score = \frac{Number\ of\ Pruned\ Firespots}{Total\ Number\ of\ Firespots} \times 100\% \tag{8}$$

2. **Perception Score:** The ratio between the number of sensed firespots and the total number of firespots that have been generated so far in the game (sensed or not-sensed):

$$Perception\ Score = \frac{Number\ of\ Sensed\ Firespots}{Total\ Number\ of\ Firespots} \times 100\% \tag{9}$$

3. **Action Score:** The ratio between the number of pruned firespots and the total number of sensed firespots (e.g., numerator in Equation 9).

$$Action\ Score = \frac{Number\ of\ Pruned\ Firespots}{Number\ of\ Sensed\ Firespots} \times 100\% \tag{10}$$

4. **Facility Protection Score:** Percentage of the number of facilities in the scenario that have been safe so far.

$$Facility\ Protection\ Score = \frac{Number\ of\ Facilities\ Never\ on\ Fire}{Total\ Number\ of\ Facilities} \times 100\% \tag{11}$$

Performance Evaluation Page: The performance evaluation page appears when a game ends (both pre-designed and user-designed scenarios). A performance evaluation page is presented in Figure 24. On top, a verbal evaluation is reported to the player according to some pre-defined metrics, which are described below. Under the general verbal evaluation bar, the final values of the online scores introduced in Equations 7–11 are again reported to the player. To generate the general verbal evaluation, we first calculate a final score as shown in Equation 12:

$$\begin{aligned} Final\ Score = Perception\ Score + Action\ Score + Facility\ Protection\ Score \\ - 3 \times Negative\ Reward\ Ratio \end{aligned} \tag{12}$$

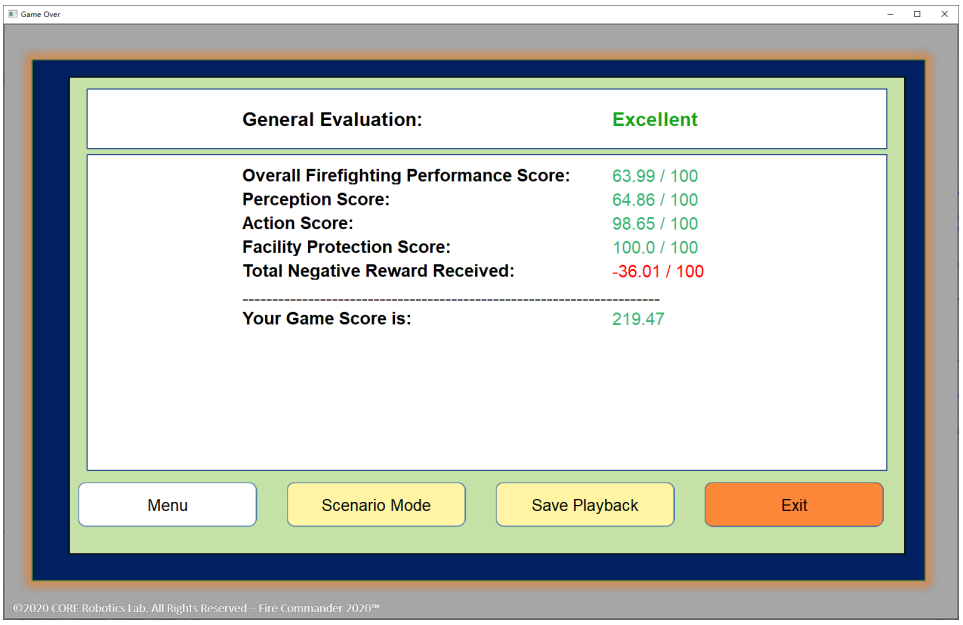


Figure 24: The Score Display.

The *Negative Reward Ratio* in Equation 12 is computed as presented in Equation 13 in which the numerator is the total negative reward calculated in Equation 7. Moreover, since the penalty coefficient for fire propagation (e.g., 0.1), number of initial firespots and the game duration, T, are known to the environment, an expected negative reward can also be calculated as in Equation 14. We call the ratio between this expected negative reward and the actual total negative reward received by the user the *Negative Reward Ratio*. We incorporate *Negative Reward Ratio* to design the objective for protecting the facilities. Note that, according to Equation 12 and Equation 13, as long as none of the facilities/targets are on fire, the final score can be easily high (close to 90%-100%), given a reasonable perception and action performances. However, when firespots enter a facility, the value of negative reward ratio suddenly increases and the final score in Equation 12 drops. Note that the negative reward ratio also enforces agents to get rid of the firespots that have entered a facility as soon as possible, since the longer firespots are inside a facility, the larger will be the negative reward ratio (e.g., according to Equation 7).

$$Negative\ Reward\ Ratio = \frac{Total\ Negative\ Reward}{Expected\ Negative\ Reward} \times 100\% \tag{13}$$

$$Expected\ Negative\ Reward = \sum_{t=1}^{T} 0.1 \times (Initial\ Number\ of\ Firespots \times t) \tag{14}$$

Now, leveraging the final score in Equation 12, the verbal evaluation includes the following five levels: (1) **Failed** (*Final Score* <50), (2) **Fair** (50 <*Final Score*<60), (3) **Almost There!** (60 <*Final Score*<80), (4) **Well Done** (80 <*Final Score*<90) and, (5) **Excellent** (*Final Score* >90).

After the final performance evaluation page, users could either choose to return to the main menu, move to scenario mode, or simply exit the GUI. Additionally, we provided the option that users can save a playback of their game (e.g., their performance) as a video file. For this option, users can select the Save Playback button to call the animation reconstruction function, which creates a video playback of the game from the saved data and stores the video on the computer.

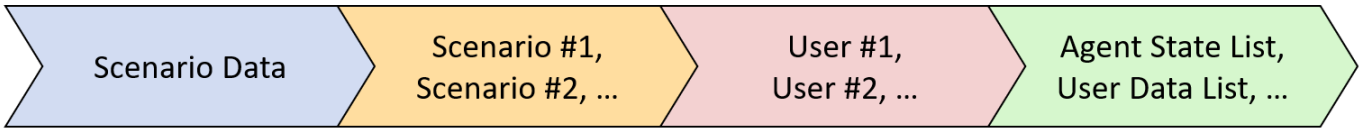


Figure 25: The Data Storage Structure.

3.6 Animation Reconstruction

We provided the option that users can save a playback of the game they just played (e.g., their performance) as a video file. For this option, users can select the Save Playback button on the final performance evaluation page to call the animation reconstruction function, which creates a video playback of the game from the saved data and stores the video on the computer. The playback video can be used in image-based LfD and LfHD algorithms, or other observations for HRI/Psychology studies. During each trial of the game, the GUI writes the user data into a file with **.pkl* format with a frequency of 100Hz. The stored data is then used to reconstruct the video playback.

3.7 Data Formats

All required information from a trial of the game are recorded and automatically stored in a designated folder. The recorded data, e.g., states of the environment, must cover 4 main aspects of the game: (1) agent states, (2) wildfire states, (3) facility/target states and (4) user-data information. We elaborate each category separately in the following sub-sections and show the structure of the stored data for each case. Figure 25 shows the general data storage structure.

Agent Information (States): The agent information list includes all agents' states at all time-steps during the game. The agents states include their positions, velocities, identity and other necessary information for trajectory planning, sensing and pruning tasks. Figure 26 presents the global structure of the stored agents' state information. Note that, inside each individual agent's state list, the elements-order follows the structure depicted in Figure 27. Each Bit is described below:

- **Bit 0 – 2:** Current Pose, (X, Y, Z) coordinates.
- **Bit 3 – 5:** Current Velocity along (X, Y, Z) axes, which determine the agent's next-step pose.
- **Bit 6:** Current time-step.
- **Bit 7:** Current goal index (e.g., in normal trajectory mode).
- **Bit 8 – 9:** Agent's Identity such that, bit 8 shows agent's type: (0) for Perception, (1) for Action and (2) for Hybrid agents. Bit 9 shows the agent's index in its specific category.
- **Bit 10 – 12:** Battery and water tank information, where the three bits represent: (1) current cumulative distance, (2) current cumulative waiting time and, (3) water tank capacity (Action and Hybrid agents), respectively.
- **Bit 13 – 15:** Control flag, where the three bits respectively represent: (1) movement flag, which determines whether to finish the trajectory or to retreat to the base, (2) patrolling flag, which determines the normal or the patrolling trajectory modes and, (3) patrolling goal index for the patrolling trajectory mode.

User Data Information: The user-data information is a list including the player's actions on keyboard and mouse. Such user actions include, selecting an agent and planning trajectories for

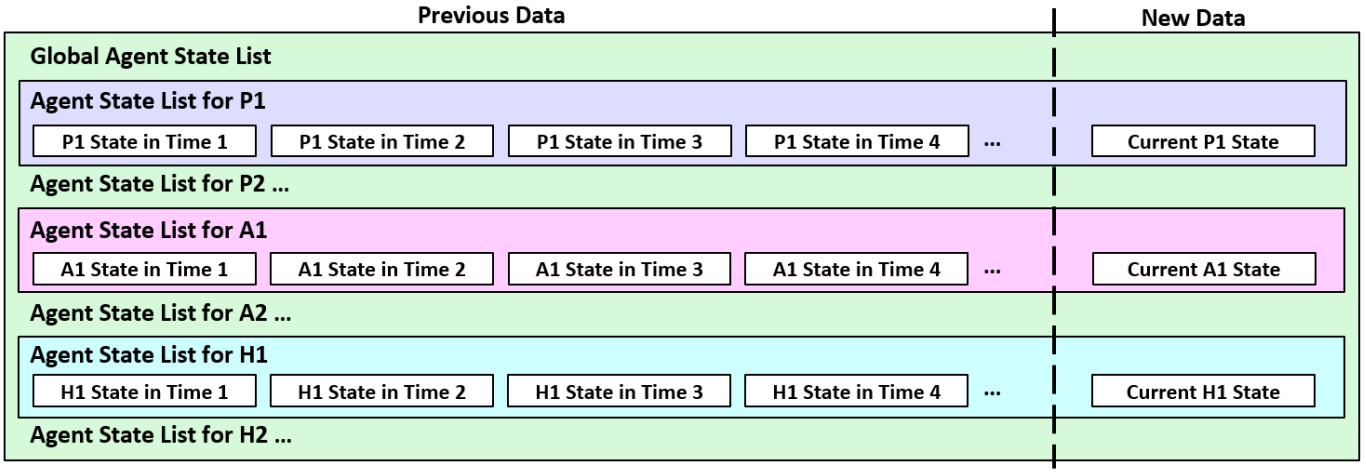


Figure 26: The Structure of the Global Agent State List.

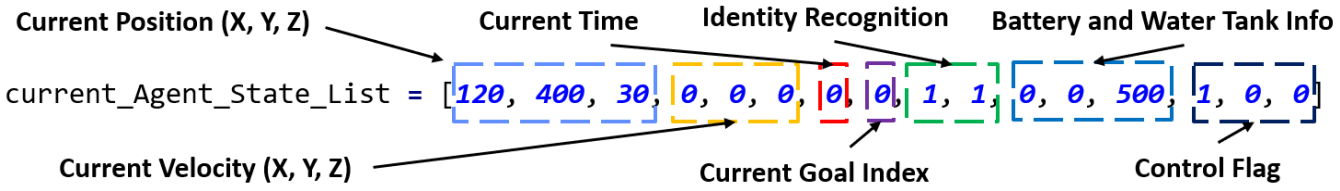


Figure 27: The Agent State List for P1 at Time 0.

agents by adding new goal to the goal buffer, designing a normal or a patrolling trajectory and switching the current agent's flight altitude. Figure 28 presents the general structure of the user-data list. The goal information for all the agents is stored in a single list. Since the agents' lists are stored sequentially, the list recording states for a specific agent can then be easily tracked. Once the new goal is created for the current agent, a unit list called new goal buffer is appended at the end of the user data list. The new goal buffer list stores the goal information for an agent generated at time t. The user-data list is arranged chronologically, in which the keyboard and mouse actions are stored together and are distinguished by the action type flag. Figure 29 presents the detail of the new goal buffer list, the bits of which are described below:

- **Bit 0 – 1:** Goal position, (X, Y) coordinate of the user click on screen. Note that, in case of vertical motion, agent's previous position is repeated for current position. Diagonal motions are not allowed in this environment.
- **Bit 2:** Current time of user's actions (e.g., mouse or keyboard use).
- **Bit 3 – 4:** Keyboard action type, such that, (0) is for a planar motion (mouse click) and, (1) is for vertical motion (arrow keys on keyboard).
- **Bit 5:** Goal index for the planar goal.

Target Loci List: The target loci list is used to store the stats of all facilities/targets in the environment at each moment. Although in our current version of the FireCommander the targets are always static, we provide the recording of targets' states at every time-step which allows further extensions of the environment to include the dynamic target. The target loci data structures are shown in Figures 30–33. The figures are self-descriptive and the notations are mostly similar to the previous notations used for agents' states data and user-data lists. Nevertheless, here are some useful

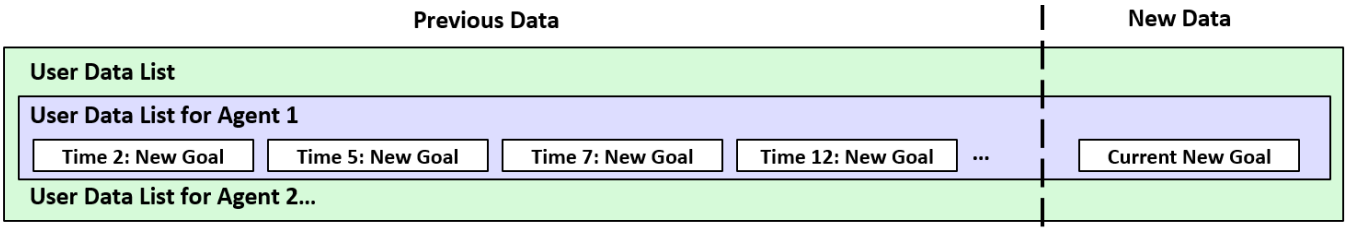


Figure 28: The User Data List.

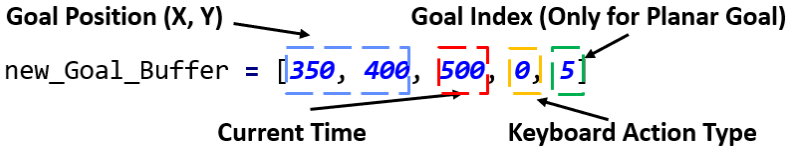


Figure 29: The New Goal Buffer List.

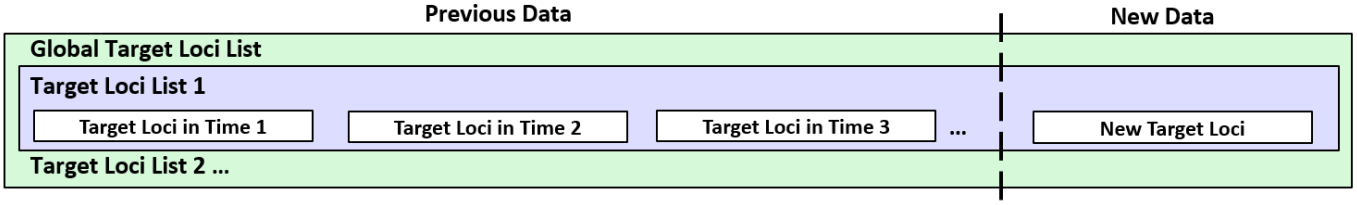


Figure 30: The Target Loci List.

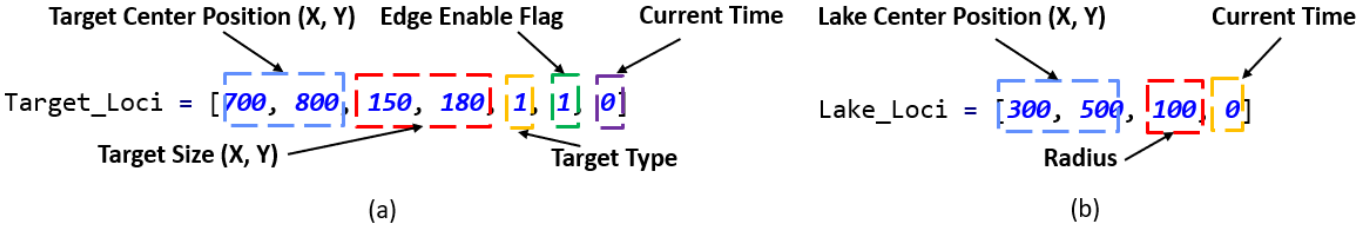


Figure 31: The Target Loci List Unit.

information to help understand these figures better:

- The general target loci list applies to the house, hospital, power station and lake. Although using a similar template, the information for the first three targets, house, hospital and power station, are stored in the same list, while the information for the lake is stored separately. The reason for rises from the fact that the designed lake has a large irregular shape.
- The agent base information is separated and is stored in a different file to make it easier for the users to have access to this data. Having a moving base also poses an interesting robotics problem in which a large, dynamic ground robot is used as a recharge station for small quadcopters. The quadcopters can be considered as Perception and Action agents which now need to coordinate and communicate with the moving base as well as their other teammates.

Wildfire States List: The wildfire list stores the states of firespots, that is the essential information required to track the fire propagation, including firespot coordinates and intensities at each time step. The fire states list consists of three separate types of fire information: (1) new firefronts generated, (2) sensed firespots and (3) pruned firespots. Figure 34 presents the structure of the fire

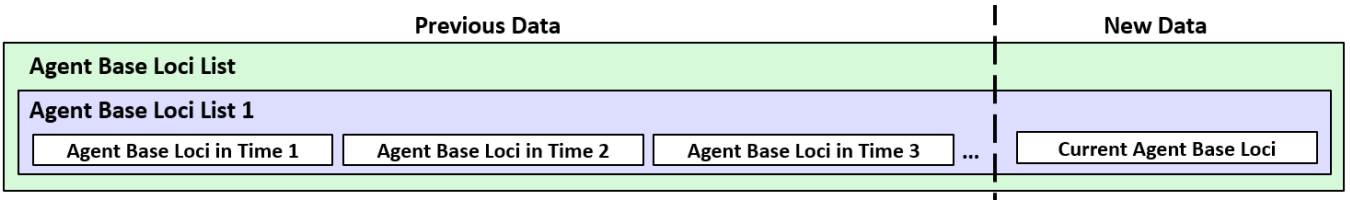


Figure 32: The Agent Base Loci List.

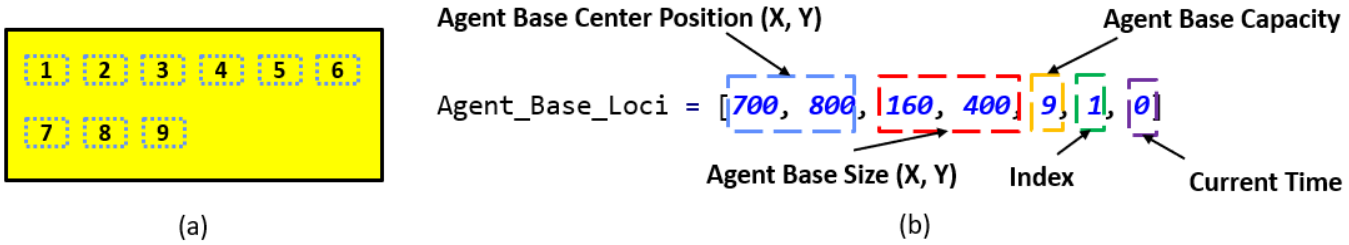


Figure 33: The Agent Base Layout and Its Composition Unit List.

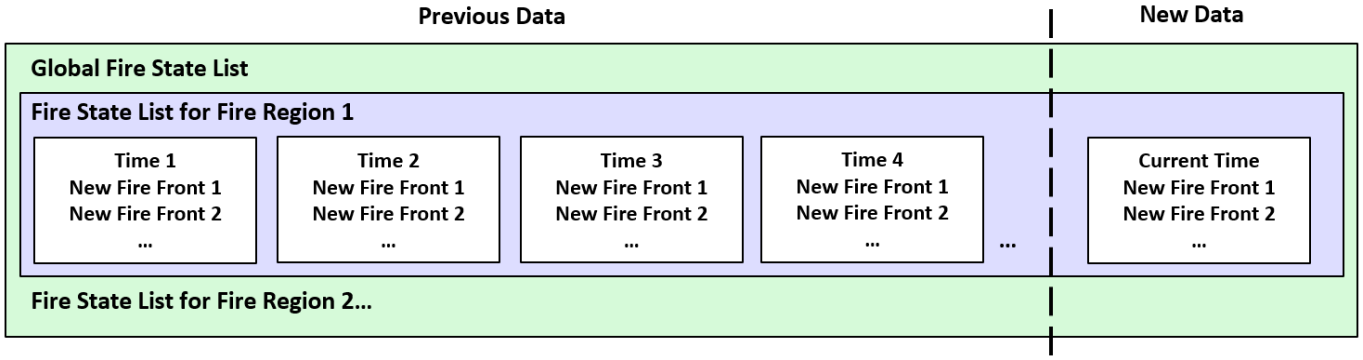


Figure 34: The Fire State List.

state list that stores all of the generated firefronts, separately for each initial fire region and over time. Figure 35 presents the details for the new firefront list that records the position, intensity and generation time for new firefronts. Note that, in the discrete setting of PyGame, the fire coordinates (and all other coordinates as well) are always discrete values, and thus, often it may occur that two firespots propagate to the same location. In such cases, the fire intensity of the new firespot will be the linear summation of the two parent firespots. Figure 36 presents the structure of stored data for the sensed and pruned firespots. The structure of the sensed and pruned fire front list are generally the same, and they both are similar to the general new firefront data list to some extend. The differences are: (1) rather than across fire regions and over time, firespot data here is stored across agents and over time and, (2) The sensed and pruned fires list entry at some time steps may be null. Figure 37(a) presents the structure of the sensed fire front list that records the position, intensity and fire velocity for the sensed fire spots. The fire velocity is necessary to compute the center of mass for the continuous fire regions. Figure 37(b) presents the structure of the pruned fire front list that records the position, intensity for the pruned fire spots.

Storing the Information for the Firespots That Enter Facilities (Target-On-Fire List): We also need to keep track of the information of the firespots that enter facilities/targets (e.g., the target-on-fire list). Figure 38 presents the target-on-fire list structure. This list stores the number of firespots that locate inside each facility/target at each time step. When an Action or Hybrid agent prunes some of the firespots inside a target, the respective firespots are removed from the list for

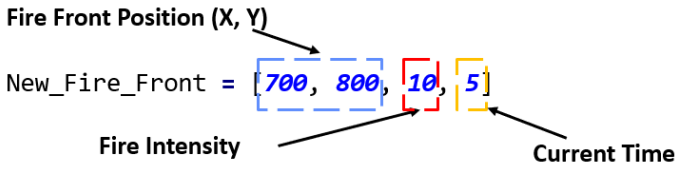


Figure 35: The New Fire Front List.

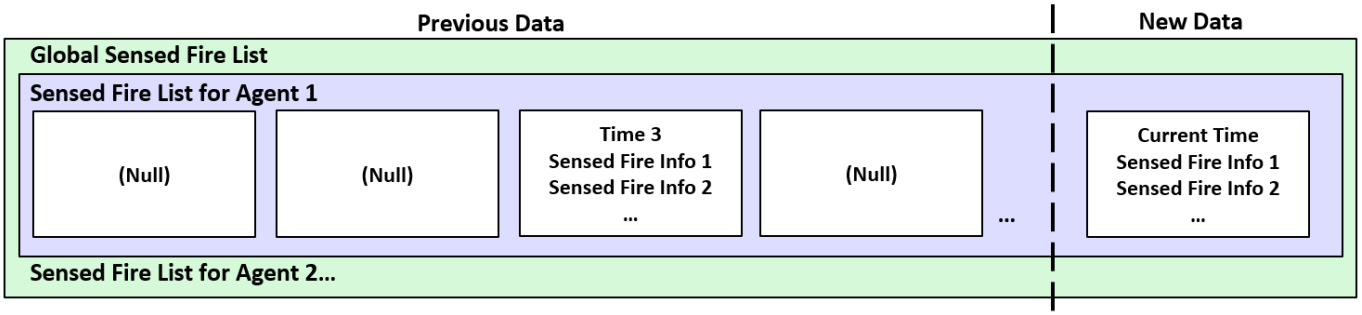


Figure 36: The Structure of the Sensed and Pruned Fire List.

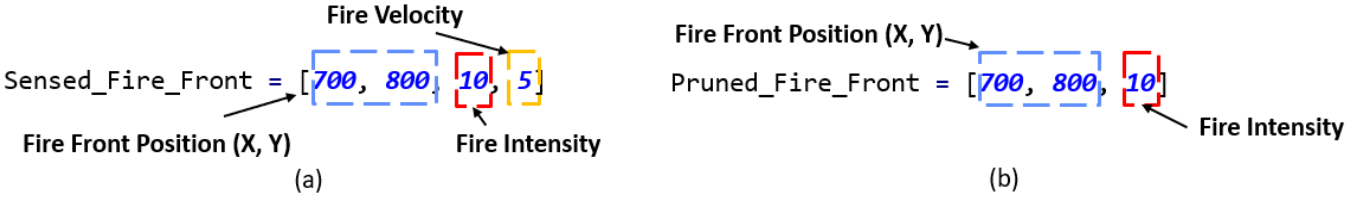


Figure 37: The Sensed and Pruned Fire List Unit

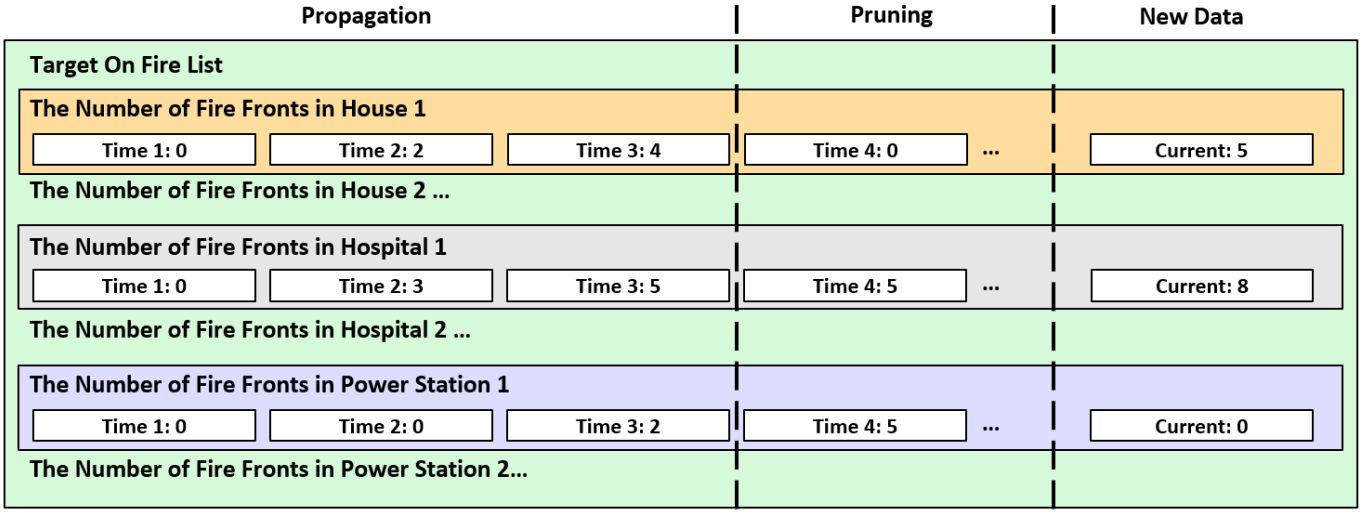


Figure 38: The Target on Fire List.

that time-step onward. Note that, although the user receives the positive reward for removing the firespots inside a target and potentially avoid receiving exponentially growing negative rewards at future time steps, the current target is still counted as non-protected and thus, a negative reward will be counted in the facility protection score in Equation 11.

4 Acknowledgment

The authors would like to thank Letian Chen and Rohan Paleja for their insightful comments and feedback.

5 References

References

[1] E. Seraj, X. Wu, and M. C. Gombolay, "Firecommander 2020," https://github.com/EsiSeraj/FireCommander2020, 2020.

[2] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[3] L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*. John Wiley & Sons, 1999, vol. 55.

[4] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003, vol. 24.

[5] I. Sghir, I. B. Jaafar, and K. Ghédira, "A multi-agent based optimization method for combinatorial optimization problems," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 05, p. 1850021, 2018.

[6] L. S. Shapley, "Stochastic games," *Proceedings of the national academy of sciences*, vol. 39, no. 10, pp. 1095–1100, 1953.

[7] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[9] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[10] Y. Gao, J. Peters, A. Tsourdos, S. Zhifei, and E. M. Joo, "A survey of inverse reinforcement learning techniques," *International Journal of Intelligent Computing and Cybernetics*, 2012.

[11] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in neural information processing systems*, 2016, pp. 2137–2145.

[12] S. Sukhbaatar, R. Fergus *et al.*, "Learning multiagent communication with backpropagation," in *Advances in neural information processing systems*, 2016, pp. 2244–2252.

[13] C. Zhang and V. Lesser, "Coordinating multi-agent reinforcement learning with limited communication," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 2013, pp. 1101–1108.

Other Toolboxes by E. Seraj:

- Essential Motor Cortex Signal Processing: an ERP and functional connectivity MATLAB toolbox [92]
- Cerebral Signal Instantaneous Parameters Estimation MATLAB Toolbox [93]

[14] M. Ghavamzadeh and S. Mahadevan, "Learning to communicate and act using hierarchical reinforcement learning," *Computer Science Department Faculty Publication Series*, p. 172, 2004.

[15] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation," in *Advances in neural information processing systems*, 2018, pp. 7254–7264.

[16] D. Kim, S. Moon, D. Hostallero, W. J. Kang, T. Lee, K. Son, and Y. Yi, "Learning to schedule communication in multi-agent reinforcement learning," *arXiv preprint arXiv:1902.01554*, 2019.

[17] H. Mao, Z. Gong, Y. Ni, and Z. Xiao, "Accnet: Actor-coordinator-critic net for" learning-to-communicate" with deep multi-agent reinforcement learning," *arXiv preprint arXiv:1706.03235*, 2017.

[18] S. Q. Zhang, Q. Zhang, and J. Lin, "Efficient communication in multi-agent reinforcement learning via variance based control," in *Advances in Neural Information Processing Systems*, 2019, pp. 3235–3244.

[19] E. Seraj, X. Wu, and M. C. Gombolay, "Firecommander: An interactive, probabilistic multi-agent environment for joint perception-action tasks - presentation slides," 2020.

[20] L. Chen, R. Paleja, M. Ghuy, and M. Gombolay, "Joint goal and strategy inference across heterogeneous demonstrators via reward network distillation," in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 2020, pp. 659–668.

[21] J. Song, H. Ren, D. Sadigh, and S. Ermon, "Multi-agent generative adversarial imitation learning," in *Advances in neural information processing systems*, 2018, pp. 7461–7472.

[22] L. Yu, J. Song, and S. Ermon, "Multi-agent adversarial inverse reinforcement learning," *arXiv preprint arXiv:1907.13220*, 2019.

[23] A. Šošić, W. R. KhudaBukhsh, A. M. Zoubir, and H. Koeppl, "Inverse reinforcement learning in swarm systems," *arXiv preprint arXiv:1602.05450*, 2016.

[24] R. Paleja and M. Gombolay, "Inferring personalized bayesian embeddings for learning from heterogeneous demonstration," *arXiv preprint arXiv:1903.06047*, 2019.

[25] Z. Wang and M. Gombolay, "Learning to dynamically coordinate multi-robot teams in graph attention networks," *arXiv preprint arXiv:1912.02059*, 2019.

[26] ——, "Learning scheduling policies for multi-robot coordination with graph attention networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4509–4516, 2020.

[27] H. Ravichandar, K. Shaw, and S. Chernova, "Strata: A unified framework for task assignments in large teams of heterogeneous robots," *arXiv preprint arXiv:1903.05149*, 2019.

[28] M. Natarajan and M. Gombolay, "Effects of anthropomorphism and accountability on trust in human robot interaction," in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 2020, pp. 33–42.

[29] C. Heyer, "Human-robot interaction and future industrial robotics applications," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 4749–4754.

[30] J. Złotowski, D. Proudfoot, K. Yogeeswaran, and C. Bartneck, "Anthropomorphism: opportunities and challenges in human–robot interaction," *International journal of social robotics*, vol. 7, no. 3, pp. 347–360, 2015.

[31] J. Fink, "Anthropomorphism and human likeness in the design of robots and human-robot interaction," in *International Conference on Social Robotics*. Springer, 2012, pp. 199–208.

[32] A. Y. Gao, W. Barendregt, and G. Castellano, "Personalised human-robot co-adaptation in instructional settings using reinforcement learning," in *IVA Workshop on Persuasive Embodied Agents for Behavior Change: PEACH 2017, August 27, Stockholm, Sweden*, 2017.

[33] C. Schaff and M. R. Walter, "Residual policy learning for shared autonomy," *arXiv preprint arXiv:2004.05097*, 2020.

[34] A. Jevtić, A. Colomé, G. Alenyà, and C. Torras, "Robot motion adaptation through user intervention and reinforcement learning," *Pattern Recognition Letters*, vol. 105, pp. 67–75, 2018.

[35] M. Hüttenrauch, A. Šošić, and G. Neumann, "Guided deep reinforcement learning for swarm systems," *arXiv preprint arXiv:1709.06011*, 2017.

[36] E. Seraj and R. Sameni, "Robust electroencephalogram phase estimation with applications in brain-computer interface systems," *Physiological measurement*, vol. 38, no. 3, p. 501, 2017.

[37] F. Karimzadeh, R. Boostani, E. Seraj, and R. Sameni, "A distributed classification procedure for automatic sleep stage scoring based on instantaneous electroencephalogram phase and envelope features," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 26, no. 2, pp. 362–370, 2017.

[38] E. Seraj and F. Karimzadeh, "Improved detection rate in motor imagery based bci systems using combination of robust analytic phase and envelope features," in *2017 Iranian Conference on Electrical Engineering (ICEE)*. IEEE, 2017, pp. 24–28.

[39] R. Sameni and E. Seraj, "A robust statistical framework for instantaneous electroencephalogram phase and frequency estimation and analysis," *Physiological measurement*, vol. 38, no. 12, p. 2141, 2017.

[40] E. Seraj, "An investigation on the utility and reliability of electroencephalogram phase signal upon interpreting cognitive responses in the brain: A critical discussion," *Journal of Advanced Medical Sciences and Applied Technologies*, vol. 2, no. 4, pp. 299–312, 2016.

[41] M. A. Goodrich and A. C. Schultz, *Human-robot interaction: a survey*. Now Publishers Inc, 2008.

[42] A. Kolling, K. Sycara, S. Nunnally, and M. Lewis, "Human swarm interaction: An experimental study of two types of interaction with foraging swarms," *Journal of Human-Robot Interaction*, vol. 2, no. 2, 2013.

[43] I. F. Akyildiz and I. H. Kasimoglu, "Wireless sensor and actor networks: research challenges," *Ad hoc networks*, vol. 2, no. 4, pp. 351–367, 2004.

[44] T. Melodia, D. Pompili, V. C. Gungor, and I. F. Akyildiz, "A distributed coordination framework for wireless sensor and actor networks," in *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, 2005, pp. 99–110.

[45] ——, "Communication and coordination in wireless sensor and actor networks," *IEEE transactions on mobile computing*, vol. 6, no. 10, pp. 1116–1129, 2007.

[46] E. Seraj, X. Wu, and M. C. Gombolay, "Firecommander: An interactive, probabilistic multi-agent environment for joint perception-action tasks," 2020.

[47] J. Ferber and G. Weiss, *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley Reading, 1999, vol. 1.

[48] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[49] E. Seraj, L. Chen, and M. C. Gombolay, "A hierarchical coordination framework for joint perception-action tasks in composite robot teams," *IEEE Transactions on Robotics [Manuscript Under Review]*, 2020.

[50] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, p. 29, 2019.

[51] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous agents and multi-agent systems*, vol. 11, no. 3, pp. 387–434, 2005.

[52] S. Kapetanakis and D. Kudenko, "Reinforcement learning of coordination in cooperative multi-agent systems," *AAAI/IAAI*, vol. 2002, pp. 326–331, 2002.

[53] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[54] P. C. Pendharkar, "Game theoretical applications for multi-agent systems," *Expert Systems with Applications*, vol. 39, no. 1, pp. 273–279, 2012.

[55] J. Etesami and C.-N. Straehle, "Non-cooperative multi-agent systems with exploring agents," *arXiv preprint arXiv:2005.12360*, 2020.

[56] R. H. Guttman and P. Maes, "Cooperative vs. competitive multi-agent negotiations in retail electronic commerce," in *International Workshop on Cooperative Information Agents*. Springer, 1998, pp. 135–147.

[57] X. Wang and D. Klabjan, "Competitive multi-agent inverse reinforcement learning with sub-optimal demonstrations," *arXiv preprint arXiv:1801.02124*, 2018.

[58] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for robocup soccer keepaway," *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.

[59] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement learning for robot soccer," *Autonomous Robots*, vol. 27, no. 1, pp. 55–73, 2009.

[60] E. Asali, M. Valipour, N. Zare, A. Afshar, M. Katebzadeh, and G. Dastghaibyfard, "Using machine learning approaches to detect opponent formation," in *2016 Artificial Intelligence and Robotics (IRANOPEN)*. IEEE, 2016, pp. 140–144.

[61] M. Gombolay, R. Wilcox, and J. Shah, "Fast scheduling of multi-robot teams with temporospatial constraints," 2013.

[62] M. Gombolay, "Human-machine collaborative optimization via apprenticeship scheduling," Ph.D. dissertation, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, January 2017a.

[63] R. Marinescu and R. Dechter, "And/or branch-and-bound search for combinatorial optimization in graphical models," *Artificial Intelligence*, vol. 173, no. 16-17, pp. 1457–1491, 2009.

[64] J. Le Ny, M. Dahleh, and E. Feron, "Multi-uav dynamic routing with partial observations using restless bandit allocation indices," in *2008 American Control Conference*. IEEE, 2008, pp. 4220–4225.

[65] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, "Reinforcement learning from demonstration through shaping," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[66] M. Gombolay, R. Jensen, J. Stigile, S.-H. Son, and J. Shah, "Apprenticeship scheduling: Learning to schedule from human experts." AAAI Press/International Joint Conferences on Artificial Intelligence, 2016.

[67] N. Soans, E. Asali, Y. Hong, and P. Doshi, "Sa-net: Robust state-action recognition for learning from observations," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2153–2159.

[68] D. Bertsimas and J. Niño-Mora, "Restless bandits, linear programming relaxations, and a primal-dual index heuristic," *Operations Research*, vol. 48, no. 1, pp. 80–90, 2000.

[69] R. Paleja, A. Silva, L. Chen, and M. Gombolay, "Interpretable apprenticeship learning from heterogeneous decision-making via personalized embeddings," *arXiv preprint arXiv:1906.06397*, 2019.

[70] R. Paleja and M. Gombolay, "Heterogeneous learning from demonstration," in *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2019, pp. 730–732.

[71] L. Chen, R. Paleja, and M. Gombolay, "Learning from suboptimal demonstration via self-supervised reward regression," *arXiv preprint arXiv:2010.11723*, 2020.

[72] S. Lee, "Sensor fusion and planning with perception–action network," *Journal of Intelligent and robotic systems*, vol. 19, no. 3, pp. 271–298, 1997.

[73] N. Sabor, S. Sasaki, M. Abo-Zahhad, and S. M. Ahmed, "A comprehensive survey on hierarchical-based routing protocols for mobile wireless sensor networks: Review, taxonomy, and future directions," *Wireless Communications and Mobile Computing*, vol. 2017, 2017.

[74] V. Ramasamy, "Mobile wireless sensor networks: An overview," in *Wireless Sensor Networks: Insights and Innovations*. InTech, 2017, pp. 3–19.

[75] R. Elhabyan, W. Shi, and M. St-Hilaire, "Coverage protocols for wireless sensor networks: Review and future directions," *Journal of Communications and Networks*, vol. 21, no. 1, pp. 45–60, 2019.

[76] L. Hu and D. Evans, "Localization for mobile sensor networks," in *Proceedings of the 10th annual international conference on Mobile computing and networking*, 2004, pp. 45–57.

[77] S. Poduri and G. S. Sukhatme, "Constrained coverage for mobile sensor networks," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 1. IEEE, 2004, pp. 165–171.

[78] E. Seraj and M. Gombolay, "Coordinated control of uavs for human-centered active sensing of wildfires," in *2020 American Control Conference (ACC)*. IEEE, 2020, pp. 1845–1852.

[79] E. Seraj, A. Silva, and M. Gombolay, "Safe coordination of human-robot firefighting teams," *arXiv preprint arXiv:1903.06847*, 2019.

[80] F. Karimzadeh, N. Cao, B. Crafton, J. Romberg, and A. Raychowdhury, "Hardware-aware pruning of dnns using lfsr-generated pseudo-random indices," *The IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019.

[81] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[82] F. Karimzadeh, N. Cao, B. Crafton, J. Romberg, and A. Raychowdhury, "A hardware-friendly approach towards sparse neural networks based on lfsr-generated pseudo-random sequences," *IEEE Transactions on Circuits and Systems I [Manuscript Under Review]*, 2020.

[83] E. Asali, F. Shenavarmasouleh, F. G. Mohammadi, P. S. Suresh, and H. R. Arabnia, "Deepmsrf: A novel deep multimodal speaker recognition framework with feature selection," *arXiv preprint arXiv:2007.06809*, 2020.

[84] Z. Wang and M. Gombolay, "Heterogeneous graph attention networks for scalable multi-robot scheduling with temporospatial constraints."

[85] M. Gombolay, A. Bair, C. Huang, and J. Shah, "Computational design of mixed-initiative human–robot teaming that considers human factors: situational awareness, workload, and workflow preferences," *The International journal of robotics research*, vol. 36, no. 5-7, pp. 597–617, 2017.

[86] M. Gombolay, R. Jensen, J. Stigile, T. Golen, N. Shah, S.-H. Son, and J. Shah, "Human-machine collaborative optimization via apprenticeship scheduling," *Journal of Artificial Intelligence Research*, vol. 63, pp. 1–49, 2018.

[87] M. C. Gombolay, R. A. Gutierrez, S. G. Clarke, G. F. Sturla, and J. A. Shah, "Decision-making authority, team efficiency and human worker satisfaction in mixed human–robot teams," *Autonomous Robots*, vol. 39, no. 3, pp. 293–312, 2015.

[88] M. C. Gombolay, C. Huang, and J. Shah, "Coordination of human-robot teaming with human task preferences," in *2015 AAAI Fall Symposium Series*, 2015.

[89] M. A. Finney, "Farsite: Fire area simulator-model development and evaluation," *Res. Pap. RMRS-RP-4, Revised 2004. Ogden, UT: US Department of Agriculture, Forest Service, Rocky Mountain Research Station. 47 p.*, vol. 4, 1998.

[90] E. Seraj and M. Gombolay, "Coordinated Control of UAVs for Human-Centered Active Sensing of Wildfires - Presentation Slides," Georgia Institute of Technology, USA, Research Report, Jul. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02904163

[91] M. E. Alexander, "Calculating and interpreting forest fire intensities," *Canadian Journal of Botany*, vol. 60, no. 4, pp. 349–357, 1982.

[92] E. Seraj and K. Mahalingam, "Essential motor cortex signal processing: an erp and functional connectivity matlab toolbox–user guide," *arXiv preprint arXiv:1907.02862*, 2019.

[93] E. Seraj, "Cerebral signal instantaneous parameters estimation matlab toolbox-user guide version 2.3," *arXiv preprint arXiv:1610.02249*, 2016.