Wildfire Monitoring System

Alexander Hoare 20102303 Giorgos Lysandrou 20106559 Mohak Narang 20088508
Shivam Gandhi 20078225 Yuhao Li 1824562

December 2020

1 Purpose of platform

Wildfires can spread incredibly quickly, causing irreparable damage to properties, and are difficult to control and prevent. The 2020 California wildfire destroyed around ten thousand buildings in the area, costing over $1bn USD and resulting in 30 deaths in total [1]. The causes for wildfires can be due to multiple reasons, such as the increase of human activities, climate change, drought conditions and tropical storms, in particular human activities have become a significant cause for wildfires in recent years.

Monitoring wildfires manually is a challenging task, the most obvious reason would be the extensiveness of forests. For example, the Peak District national park in the UK, has an area of 1440 km^2 [2], making it almost impossible to monitor the entire area. Another reason would be the lack of infrastructure such as signal towers, making information exchange difficult and rapid response to the fire unfeasible.

In this coursework, we propose a wildfire monitoring system using Unmanned Aerial Vehicle (UAV) drones and Multi-Agent theory. The drones will patrol an area looking for a fire. Once found, it will ask other survey drones to converge on its location to double-check if it is indeed a fire. Firefighting drones are then dispatched to the location to douse the flames. In the meantime, maintenance drones will be used to ensure that all agents in the system remain fully operational.

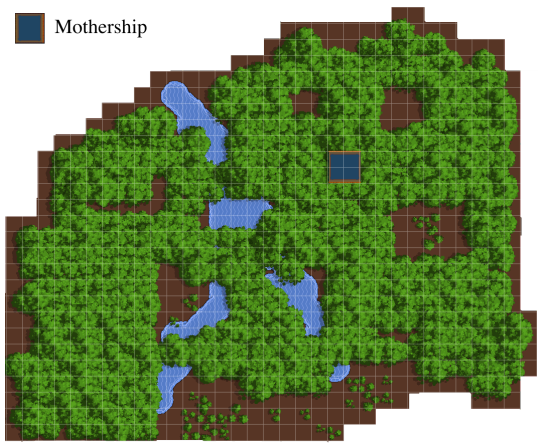

Figure 1: An example forest made with DeepNight [3]

The **Domain** will be considered as a cell world, as shown on Figure 1, each grid represent an area that could be monitored by a single drone. The drones are first launched from a *Mothership* which is a base station capable of storing, maintaining and recharging the drones, they will then explore and patrol the forest.

2 High level design

There will be three types of agents, each correspond to one of the drones:

Agent notation	Drone	Responsibility
D_s	Survey Drone	Explore, patrol and survey cells, alarm when fire is detected
D_f	Firefighting Drone	Extinguish fire
D_m	Maintenance Drone	Repair and recharge drones at request

Figure 2: Drone Notation

Instead of having separate architecture for each agent, we use an universal architecture for all of them while specifying the authorised users to different functional modules. In this way, the architecture is open for future extension but closed for modification within the modules, thus reducing the chance of malicious attacks which could be lethal for a multi-agent application.

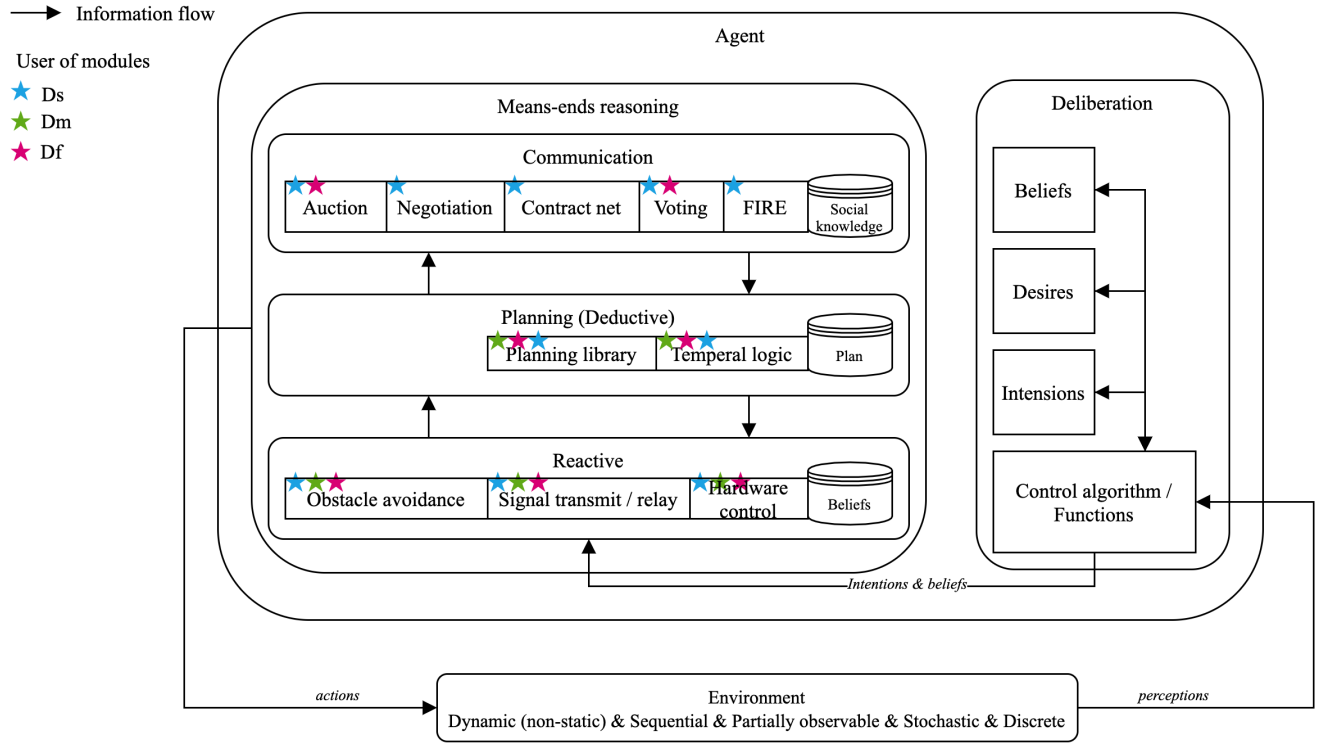

Figure 3: Agent architecture

The architecture combines BDI [4] and InteRRaP [5]. The deliberation with beliefs, desires and intentions provides flexibility for an agent to determine and change it's goal, also the similarity to human thinking means it is explainable and easy to understand. However, in a non-static environment, the planning of an agent might not be sound due to changes of the environment during the planning period. So InteRRaP is introduced to allow switching between simple reactions, deductive reasoning and also communication when an agent alone is not capable of accomplishing the goal.

3 BDI Deliberation

Originally devised for modelling human behaviour[6], the Beliefs, Intentions and Desires (BDI) model allows for a much clearer reasoning about the actions taken by an agent in a system, and creates a system more easily managed and expanded upon.

Each drone type comes complete with its own set of beliefs, intentions and desires, as well as sharing common ones that allow the drones to ensure they remain functional and aware of the environment they are currently in.

3.1 Shared Beliefs

The following beliefs and desires are shared by all drones in the system:

Belief	Summary	Predicate
BatteryLevel	Current battery level of the drone as a percentage	Float BatteryLevel (0-1)
CurrentStatus	A list of broken parts (if any) that the drone needs repaired	List(Part) BrokenList
Tasks	A list of tasks for the agent	List(Task) TaskList
ForestBounds	An array that holds the forest data/bounds	Forest[X][Y]

Figure 4: Universal Drone Intentions

3.2 Shared Desires

"!ϕ denotes "achieve ϕ", "?ϕ" denotes "query ϕ". [7]

Desire	Rationale
!(BatteryLevel ≥ 0.2)	Has enough battery
?BrokenParts	A check for broken parts
!(Empty(BrokenList))	Desire to remain fully functional
!(Inside(ForestBounds))	Desire to remain in the bounds of the forest
!(Complete(Task in TaskList))	Desire to complete its tasks

Figure 5: Universal Drone Beliefs

3.3 Different Beliefs and Desires

Below is a table displaying some examples of **different** Beliefs, Intentions and Desires held by the different drone types:

BDI Type	Drone	Example	Summary
Belief	D_s	PatrolRoute	Which section the drone will patrol.
Belief	D_m	AvailableParts	Replacement parts the maintenance drone has access to.
Belief	D_f	WaterLevel	Current water tank level as a percentage.
Desire/Intention	D_s	!DetectFire($x_{current}$,$y_{current}$)	To detect presence of fire at its position.
Desire/Intention	D_s	!follow(PatrolRoute)	Desire to follow the patrol route.
Desire/Intention	D_m	?MaintenanceRequest	Check maintenance requests.
Desire/Intention	D_f	!(WaterLevel==1)	Remain water tank full.

Figure 6: Drone BDI Examples

4 Behaviour and Reasoning of Agents

4.1 Temporal logic

All the three agents need to reason about and react to the changes in environment based on current or past states or behaviours. We define temporal logic for each agent here on how might they react in different situations.

1. Survey Drones (ask(survey_support), ask(fire_vote)) [give(need_survey_support), give(fire_alert), ask(fire_vote), give(request_maintenance_bids)]
   - ◎ ask(need_survey_support) → action(move_in_requested_drone_area)
   - ◎ fact(fire_confirmed) → ⋄ give(fire_alert) ∧ fact(countdown_start)
   - ◎ fact(unsure_fire) → ⋄ give(fire_vote)
   - ◎ fact(BatteryLevel < 0.2) → ⋄ give(request_maintenance_bids)
   - ◎ fact(notEmpty(BrokenList)) → ⋄ give(request_maintenance_bids) ∧ ⋄ give(need_survey_support)
   - ◎ ask(fire_vote) ∧ fact(adjacent_grid) → ⋄ action(move_in_vote_area) ∧ ¬ fact(adjacent_grid)
   - ◎ ask(fire_vote) ∨ action(move_in_vote_area) ∧ ¬ fact(countdown_end) → ⋄ action(order_voting_preferences)
   - ◎ action(order_voting_preferences) ∧ fact(countdown_end) → ⋄ action(announce_winner)
2. Firefighter Drones (ask(need_backup), ask(fire_alert))[give(request_maintenance_bids), give(need_backup), ask(fire_vote)]
   - ◎ ask(fire_alert) → ⋄ action(reach_alert_coordinates)
   - ◎ ask(need_backup) → ⋄ action(reach_backup_coordinates)
   - ◎ fact(BatteryLevel < 0.2) → ⋄ give(request_maintenance_bids)
   - ◎ fact(water_out) → ⋄ give(water_out_alert) ∧ ⋄ give(need_backup)
   - ◎ fact(notEmpty(BrokenList)) → ⋄ give(request_maintenance_bids) ∧ ⋄ give(need_backup)
   - (fact(fire_alert) $\mathcal{U}$ fact(fire_ceased)) → ⋄ give(fire_out)
   - ◎ action(order_voting_preferences) ∧ fact(countdown_end) → ⋄ action(announce_winner)
   - ◆ fact(fire_alert) ∧ ◆ fact(¬ fire_out) → ⋄ ask(fire_vote)
3. Maintenance Drones (ask(request_maintenance_bids))[give(maintenance_bid)]
   - ◎ ask(request_maintenance_bids) ∧ fact(available) → give(maintenance_bid)
   - ◎ fact(maintenance_completed) → fact(available)
   - ◎ ask(request_maintenance_bids) ∧ fact(¬available) → fact(wait) $\mathcal{W}$ fact(available)
   - ◎ fact(wait) ∧ fact(available) → give(maintenance_bid)

S.no	Survey Drone 1		Survey Drone 2		Survey Drone 3		Survey Drone 4		Firefighting Drone	
	Propositions	Commitment	Propositions	Commitment	Propositions	Commitment	Propositions	Commitment	Propositions	Commitment
1	ask(fire_vote)	⋄action(order voting preferences)		fact(adjacent_grid) W fact(¬adjacent_grid)		fact(adjacent_grid) W fact(¬adjacent_grid)		fact(adjacent_grid) W fact(¬adjacent_grid)		fact(adjacent_grid) W fact(¬adjacent_grid)
2	action(order voting preferences)		ask(fire_vote)	⋄(action(move_in_vote_area) AND fact(¬adjacent_grid))	ask(fire_vote)	⋄(action(move_in_vote_area) AND fact(¬adjacent_grid))	ask(fire_vote)	⋄(action(move_in_vote_area) AND fact(¬adjacent_grid))		
3			action(move_in_vote_area) AND fact(¬adjacent_grid)	⋄action(order voting preferences)	action(move_in_vote_area) AND fact(¬adjacent_grid)	⋄action(order voting preferences)	action(move_in_vote_area) AND fact(¬adjacent_grid)	⋄action(order voting preferences)		
4		⋄action(announce vote results)	action(order voting preferences)	⋄action(announce vote results)	action(order voting preferences)	⋄action(announce vote results)	action(order voting preferences)	⋄action(announce vote results)		
5	YesFire wins	○ give (fire_alert)	YesFire wins		YesFire wins		YesFire wins			
6	fire_alert								fire_alert	⋄action(reach alert coordinates), ⋄fire_vote
7	fire_vote	⋄action(order voting preferences)	fire_vote	⋄action(order voting preferences)	fire_vote	⋄action(order voting preferences)	fire_vote	⋄action(order voting preferences)	ask(fire_vote)	
8	action(order voting preferences)	⋄action(announ-ce vote results)	action(order voting preferences)	⋄action(announ-ce vote results)	action(order voting preferences)	⋄action(announ-ce vote results)	action(order voting preferences)	⋄action(announ-ce vote results)	action(order voting preferences)	⋄action(announce vote results)
9	NoFire wins		NoFire wins		NoFire wins		NoFire wins		NoFire wins, fire out	

5 Cooperation and Trust

5.1 Contract net protocol

Having an unexplored grid world, and a finite set of survey drones, the exploration of the grid could be done using Contract Net Protocol between the Mothership and survey drones, assuming time for signal transfer is negligible. Similar method is described in "Multi-agent-based Auctions for Multi-robot Exploration" [8].

A random survey drone is first assigned the task to scan the initial cell, where the Mothership is. Then the exploration is achieved by iteratively pushing the boundary between explored and unexplored areas until the frontier of the forest is met. Figure 7 shows algorithms and a sequence diagram describing the process.

```
Survey drone:

Scan current cell;
Record landFeatures(x_current, y_current, features);
Broadcast explored(droneID, x_current, y_current, landFeatures);
Derive valid neighbors of the current cell;
For every valid neighbor:
    Broadcast unexplored(x_neighbor, y_neighbor);
Wait for incoming ExploreQuest(x, y), when received:
    Calculate the utility of doing ExploreQuest(x, y);
If the utility is above a threshold:
    Respond back to the Mothership by Propose(droneID, x, y, utility);
Wait for incoming ExploreTask(droneID, x, y), when received:
    if droneID == myID:
        Move to the assigned cell;
Otherwise:
        Stay in current cell;
```

```
Mothership:

Always listen and record explored(droneID, x, y, landFeatures) and
                         unexplored(x, y) and
                         Propose(droneID, x, y, utility);
Always record landFeatures(x, y, features);
While currentTime mod contractInterval == 0:
    Calculate set Boundary = union of all unexplored(x, y);
    For each element in the Boundary set:
        Broadcast ExploreQuest(x, y);
While currentTime mod deadlineInterval == 0:
    Calculate set Proposals = union of all Propose(droneID, x, y, utility);
    For every distinct (x, y) in Proposals:
        Find the proposal with the maximum utility;
        Broadcast ExploreTask(droneID, x, y) with droneID in that proposal;
```

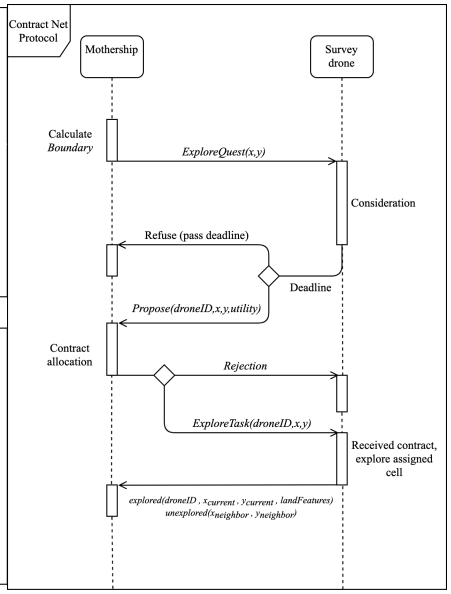

Figure 7: Contract Net Protocol for exploration

In fact, other tasks could be assigned in the some form when central processing is needed, such as patrolling tasks. After exploration, the explored areas are then recursively bisected into equal sizes and assigned to survey drones with Contract Net Protocol.

5.2 FIRE model

When allocating tasks with Contract Net Protocol, ties could happen where two agents are equally willing to do something. In this case, FIRE model with Interaction Trust as a direct source of information is used to decide which agent will be given the task.

The rating of an interaction for an agent, by completing a single task is given by:

$$v = \begin{cases} 4 & 0.9T_E \le T_C \le 1.1T_E \\ 5 & T_C < 0.9T_E \\ 4 - (2 * T_C/T_E) & 1.1T_E < T_C \le 3T_E \\ 0 & otherwise \end{cases}$$

where,
T_C : actual time for the task completion
T_E : estimated time for the task completion

The overall trust value (T) of an agent a to another agent b is updated throughout time, base on ratings of the last 30 transactions represented as tuples:

$$T : (a, b, c, r)$$

a : evaluating agent
b : target agent
c : interaction type
r : average rating

$$r = (r - \frac{r}{30}) + \frac{v}{30}$$

6 Negotiation for Task Redistribution

To account for scenarios where a Survey Agent D_{si} might need to temporarily leave their Patrol Area, for instance for a maintenance absence, a Negotiation Protocol is implemented. Upon the Agent's temporary absence from their Task, their Patrol Area Task is automatically split into 4 equal parts T_i and assigned to the immediate 4 neighbouring Agents whose Patrol Area directly borders Agent's D_{si} Patrol Area in the Grid.

The Agents then taking into account some aspects of their Beliefs, such as their battery level, negotiate between them to redistribute their newly assigned Patrol Areas[1].

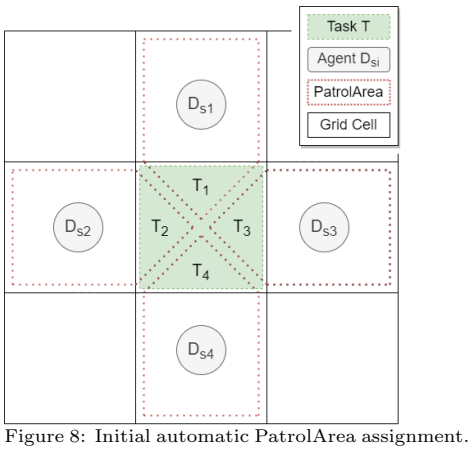

Figure 8: Initial automatic PatrolArea assignment.

$$\textbf{Task } T = \textit{temporary PatrolArea}(G, x)$$
$$\text{where} \quad G = \textit{Grid cell}$$
$$x = \textit{amount of cell Area}, \quad 0 \leq x \leq 1$$

$$\textbf{Sub-Tasks } T_i = \{T_1, T_2, T_3, T_4\}, \quad T_i \subset T$$
$$\text{where} \quad T_i = \textit{PatrolArea}(G, 0.25)$$

$$\textbf{Agents } D_{si} = \{D_{s1}, D_{s2}, D_{s3}, D_{s4}\}$$

$$\textbf{Cost } c_i = (w_i T_i - wp(T(D_{si}, T_i))), \quad \text{for Agent } D_{si}$$
$$\text{where} \quad w_i = \frac{1}{BatteryLevel}, \quad 0 < BatteryLevel \leq 100$$
$$w = \textit{constant}, \quad 0 < w < 1$$
$$p(T(D_{si}, T_i)) = \textit{predicted Trust T of Agent } D_{si} \textit{ after completion of Task } T_i$$

Algorithm 1: Temporary PatrolArea Negotiation Protocol

Agent D_{si} temporarily leaves their PatrolArea;
Tasks T_i are automatically distributed to Agents D_{si};
Cost $cost_i$ is calculated by Agents D_{si} for Task T_i;
Initialisation: **if** *for any* D_{si}, $cost_i > \theta$ **then** (where θ = *arbitrary constant threshold*)
D_{si} broadcast their $cost_i$;
Agents added to Ranked List $R[\,]$ in descending order of their $cost_i$;
while $R[\,]$*.length()* ≥ 2 **do**
  if *Adjacent*$(R[0], R[1])$ == *True* **then**
    while *TerminationCriteria == False* **do**
      Negotiation$(R[0], R[1])$;
    Remove$(R[0])$;
  else
    while *TerminationCriteria == False* **do**
      Negotiation$(R[0], R[2])$;
    Remove$(R[0])$;

Where **Negotiation**(D_{si}, D_{si}) consists of:

$$\textbf{Deal } \delta = \langle d_1, d_2 \rangle, \quad \text{between Agents } D_{s1} \text{ and } D_{s2}$$
$$\text{where} \quad d_i = \textit{PatrolArea}(G, x), \quad d_i \subset T$$
$$\text{where for } D_{si}, \quad cost_i(\delta) = c_i(w_i d_i - wpT(D_{si}, d_i))$$

$$\textbf{utility}_i(\delta) = c_i(w_i T_i - wp(T(D_{si}, T_i))) - c_i(w_i d_i - wp(T(D_{si}, d_i)))$$

Goal: Balance $\frac{utility_1(\delta_1) - utility_1(\delta_2)}{utility_1(\delta_1)}$ and $\frac{utility_2(\delta_2) - utility_2(\delta_1)}{utility_2(\delta_2)}$

TerminationCriteria = both Agents agree on a Proposal $\vee$ timeout t occurs and the initially assigned T_i Task is executed by both Agents instead of any deal δ between them.

[1] Assuming that (1) each Agent knows each other's utility function and (2) each agent knows the range of possible deals.

7 Voting Accounting for Preferences

The desired properties of a voting protocol are:

- **Surjective:** Every contestant should have an equal chance of winning.
- **Resolute:** There should always be a unique winner in the final round of voting.
- **Strategy Proof:** There should be no voter profile such that the voter prefers an outcome that is determined by its false preference.
- **Not a Dictatorship:** The winner should never be always identical to the first choice of a particular voter.
- **Weakly Pareto:** If a particular candidate is not preferred by any voter then it will not be selected as a winner.
- **Independent of Irrelevant Alternatives:** Adding a new candidate will not make a candidate win that was previously not preferred.

According to Gibbard-Satterthwaite Theorem, it is not possible to design a voting protocol that is Resolute, Surjective, Strategy Proof and not a Dictatorship.

Arrow's Theorem states that there cannot be a voting protocol that is Weakly Pareto, Independent of Irrelevant Alternatives and not a Dictatorship.

Since reporting fire is the prime functionality of the survey drones, we desire a voting protocol that is Surjective, Resolute, Strategy proof, Weakly Pareto, Independent of Irrelevant alternatives and not a Dictatorship. To achieve this, we set up restrictions on the order of candidate preference that an agent can have while voting for the presence of fire.

Median voter rule:

- Is an exception to Gibbard-Satterthwaite Theorem and Arrow's Theorem.
- It does not follow the universal domain assumption.
- It puts restrictions on order of candidate preference.

Ω = {YesFire, MaybeFire, NoFire} such that, **YesFire >> MaybeFire >> NoFire**

Voting scenarios:

- When a survey agent is unsure about the presence of fire (i.e. sensing heat levels below the alert threshold), it sends a voting signal to invite other survey drones from adjacent grids. The agents submit their preference order and appropriate action is taken after a winner is decided.
- When a firefighting drone has put out fire in a grid, it sends a voting signal to invite survey drones from adjacent grids. The winner of the voting determines the next actions.

Voting in action:
Since an ordering restriction exists on preferences, agents can order the candidates in the following ways:

$YesFire > MaybeFire > NoFire$

$NoFire > MaybeFire > YesFire$

$MaybeFire > YesFire > NoFire$

$MaybeFire > NoFire > YesFire$

Possible Outcomes:

- YesFire wins: Fire alert is raised.
- NoFire wins: Fire alert is not raised.
- MaybeFire wins: Fire is reported as a precautionary measure.

8 Auction

For the Wildfire Monitoring System, we present a modification on a well-known auction methodology, in order to ensure that the system picks the best drone for a particular task when the situation arises, depending on limited resources such as battery and fuel. The system is based on the English auction system, in which a 'price' starts low, and climbs as additional bids are made.

We utilise this auction system and modify it to create an 'inverse-synchronous sealed-bid English auction'. This modification can be simplified below:

Notation	Meaning
Inverse	Instead of the 'best price' being as high as possible, the winning bid is instead allocated to the agent which can produce the lowest score.
Synchronous	All agents involved in the auction will submit their bids at the same time.
Sealed Bid	Agents do not know what other agents have bid.
English	The auction will start at a particular best score, then will increase (in this case, decrease) until an agent which makes the best bid is found.

Figure 9: Agent Auctions

8.1 Drone Allocation

There are two scenarios in which an auction will occur. First, when deciding which of the available D_fs will be drafted to tackle a wildfire that is occurring. Secondly, when the system must decide which D_ss will be sent in order to help other D_ss analyse and vote on whether a detected wildfire is indeed happening.

For a firefighting drone, in order to decide, when a wildfire is potentially detected, if it is a fire, multiple D_ss will visit the location and conduct a voting protocol to confirm. However, an auction must occur when deciding which D_s drones to send out. Using the same inverse-synchronous English auction system, a D_s will calculate its 'score'.

Upon confirmation of a fire, the system must allocate n D_fs, where n is the number of drones required to successfully tackle the blaze. The score generated by a D_f is calculated by taking into account factors such as battery level, current levels of water stored on-board and distance from the fire.

$$S(D_s) = (1 - B) + D$$

$$S(D_f) = (1 - B) + (1 - W) + D$$

Notation	Meaning
B	Percentage battery remaining (0-1)
W	Percentage of water tank full (0-1)
D	Distance of agent from fire

Figure 10: D_f Score Formula

8.2 Combinatorial Auction

In the event that there are multiple fires, requiring multiple D_fs to go to each, a combinatorial auction will take place using a VCG mechanism. In this, each fire is assigned a ranking, based on the size of the fire (in m^2). Then different firefighting drones are assigned to different fires, in order to minimise the total score of all of the accepted bids. This is done in order to ensure that all blazes are tackled with an equal amount of suitable drones, rather than just prioritising the largest flame exclusively.

9 Other

9.1 Firefighting drone as embedded agent

Here we will define the behaviour of the firefighting agents during the task of ceasing the fire.

Following are the possible environment states, actions and the probabilities associated with them and associated conclusive utility function.

1. E = $\{e_0, \ldots, e_m, \ldots e_n\}$
   Where each e_i in E describes of decreasing order of extent of fire (%).
   Hence, the transitions between environments are ordered sequentially but can be in either direction.

2. Possible actions at each environment state include:

   (a) Moving closer/away from target area

   (b) Increasing/Decreasing flow of water

3. $\forall e_i$ in E, accommodating other environment factors that may lead to further increase in fire
   Following are the probabilities defined for three possible transitions and their values are learned dynamically through the process of Reinforcement learning model for that agent.

   (a) $P(e_i + 1)$, after applying action a_i on e_i , if action a_i is more effective than other factors

   (b) $P(e_i - 1)$, after applying action a_i on e_i , if action a_i is less effective than other factors

   (c) $P(e_i)$, after applying action a_i on e_i , if action a_i is equally effective on fire other factors

4. Utility $U(e_i) \forall e_i$ in E = 2 * (% decrease in fire extent) – (% amount of water used)

   (a) This leads to a trade-off between both values to accommodate over utilization of water in case of minor fire.

   (b) Giving (% decrease in fire extent) twice the weight to prioritize ceasing fire over saving water.

10 Reflection

10.1 Implementation Discussion

BDI Deliberation The translation of the BDI model to the agents was a smooth transition, our system was easily able to assign beliefs, desires and intentions to each of the agents, encapsulating much of the real-world environment.

Behaviour and Reasoning of Agents It was very helpful to have the high level behavior logic implemented for various agents through which could easily identify future states based on previous states.

Cooperation and Trust Contract Net Protocol comes in handy, but for real application, the Mothership need to be able to filter outdated information. FIRE model helped in setting up the reputation of an agent for another agent which could be used in resolving many ties while implementing other protocols.

Negotiation for Task Redistribution Realistic implementation of Negotiation would require a less time consuming procedure, such as using a single offer $(\frac{1-\delta_2}{1-\delta_1\delta_2}, \frac{\delta_2(1-\delta_1)}{1-\delta_1,\delta_2})$ and mechanisms in place to prevent deception.

Voting Accounting for Preferences The idea of agents voting to confirm forest fire status transitioned very well in the domain. The left-to-right ordering constraints on candidate preference were very natural and practical.

Auction Transcribing an auction process to the system proved very relevant, but difficult to compress. With auctions carried out by both the D_s and D_f drones as well as a combinatorial auction in the event of multiple fires. Realistically, in a real-world setting - a drones auction score will be affected by a plenitude of hidden variables, such as weather, part degradation and wildlife.

10.2 Limitations & Future Development

Aspects of the implementation that need to be addressed for the platform to reach a realistic functioning solution.

Platform Safety Implement encryption and secure protocols to prevent malicious activity.

Dynamic Environment Agents dynamically adjusting behaviour based on revised beliefs on eg. the weather.

Platform Semantics Universal communication language and protocols to allow for expansion of platform.

References

[1] Joseph Serna and Don Bartletti. "California faces a perilous fire season as coronavirus threatens firefighters". In: (2020). URL: https://www.latimes.com/california/story/2020-05-10/coronavirus-coming-wildfire-season.

[2] Wikipedia contributors. "Peak District". In: (2020). URL: https://en.wikipedia.org/wiki/Peak_District.

[3] *deepnight.net RGP map editor 2*. 2020. URL: https://deepnight.net/tools/rpg-map/.

[4] Anand Rao and Michael Georgeff. "BDI agents: From theory to practice". In: (Nov. 2000).

[5] Jörg Müller and Markus Pischel. "The Agent Architecture InteRRaP: Concept and Application". In: (July 1993).

[6] Michael E. Bratman. "Intention, Plans, and Practical Reason". In: (1987). URL: https://web.stanford.edu/group/cslipublications/cslipublications/site/1575861925.shtml/.

[7] Mark d'Inverno et al. "A formal specification of dMARS". In: vol. 1365. Apr. 2006, pp. 155–176. ISBN: 978-3-540-64162-9. DOI: 10.1007/BFb0026757.

[8] Xia Ma et al. "Multi-agent-based Auctions for Multi-robot Exploration". In: *2006 6th World Congress on Intelligent Control and Automation*. Vol. 2. 2006, pp. 9262–9266. DOI: 10.1109/WCICA.2006.1713793.