Wildfire-UAVSim: An Exemplar for Evaluation of Adaptive Cyber-Physical Systems in Partially-Observable Environments

Enrique Vilchez, Javier Troya, and Javier Cámara
{enriquev,jtroya,jcamara}@uma.es
ITIS Software – Universidad de Málaga
Málaga, Spain

ABSTRACT

Smart Cyber-Physical Systems (sCPS) operate under dynamic, harsh conditions and limited observability of their operation environment. To facilitate the evaluation of distributed CPS where adaptation is affected by partial observability, this paper contributes the formalization of a wildfire tracking problem in forest areas with unmanned aerial vehicles (UAV), as well as a customizable simulator (Wildfire-UAVSim) that enables the evaluation of diverse adaptation strategies both under full and partial observability conditions.

CCS CONCEPTS

• **Software and its engineering** → *Extra-functional properties*;

KEYWORDS

wildfire monitoring, artificial intelligence, UAVs

ACM Reference Format:
Enrique Vilchez, Javier Troya, and Javier Cámara. 2024. Wildfire-UAVSim: An Exemplar for Evaluation of Adaptive Cyber-Physical Systems in Partially-Observable Environments. In *19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24), April 15–16, 2024, Lisbon, AA, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3643915.3644109

1 INTRODUCTION

Motivated by the necessity for Cyber-Physical Systems (CPS) to operate in challenging domains, where continual adjustments are essential to contend with diverse sources of uncertainty, smart CPS (sCPS) are equipped with adaptive capabilities. These systems frequently find themselves in situations demanding pivotal decisions based on information that is either incomplete or imperfect. For example, autonomous vehicles operate in environments where predicting the actions of other drivers is not always possible, and smart grids must efficiently manage energy distribution despite unpredictable fluctuations in supply and demand, often lacking complete visibility of the network's status.

sCPS can be of great help for addressing extreme and recurrent problems in our society caused by the impending climate change, such as wildfires. Due to the evident risks in tracking a wildfire by overflying it with piloted aircraft, counting on unmanned aerial vehicles (UAV) able to undertake autonomous decisions is of key importance. This is specially critical when factors such as wind and smoke limit fire observability, which can be only partial sometimes. A major limitation in this field is that experimentation in a real scenario is not an option—we cannot simply set a fire to study its behavior and test our UAV, and while feasible, emulating a wildfire (e.g., with LED lights) and partial observability conditions in a controlled lab environment is complex and expensive.

Therefore, we need to be able to simulate wildfire scenarios where to test and train our UAV. This means that the simulators must not only simulate the evolution of a wildfire and other conditions such as partial observation of the environment (e.g., due to smoke), but they also need to provide the means to include agents such as UAV in the simulation so that they can monitor the fire evolution.

There are excellent exemplar problems and simulation artifacts to exercise and evaluate the adaptation capabilities of sCPS [1–3, 5, 6, 8, 11] that could in principle be used to experiment with problems analogous to some extent to wildfire tracking with UAV. In the area of distributed sCPS formed by unmanned air vehicles (UAV), DartSim [8] provides a high-level simulation of a team of UAV performing a reconnaissance mission in a hostile and unknown environment, whereas Dragonfly [6] simulates the behaviours of individual and collections of UAV and evaluates the satisfaction of mission requirements both under normal and exceptional situations. Moreover, DEECo [1] is a component system (model and runtime platform) that can be used to experiment with self-adaptive systems where the physical distribution and mobility of nodes as well as the limited data availability play an important role. UNDERSEA [2] provides simulation for unmanned underwater vehicles (UUV) that can execute long missions with challenging conditions such as oceanic changes and scarce opportunities for communication. Despite their usefulness, none of these artifacts is tailored to simulate UAV in wildfire adaptation scenarios, and more generally, to evaluate alternative adaptation strategies in situations where a partially observable environment plays a key role in the satisfaction of mission requirements.

Hence, in this paper we contribute with Wildfire-UAVSim, a customizable wildfire tracking simulator. Among its many configuration parameters, we can customize the forest area with different densities of vegetation as well as fire and smoke dispersion patterns that are affected by factors such as wind. The configuration options of our simulator also allow to place a team of UAV in charge of tracking the fire over the forest area. We explain how the behavior of these UAV can be customized in our simulator to test different adaptation strategies, providing a simple illustrative example that moves UAV randomly over the map. The contribution of this paper

SEAMS '24, April 15–16, 2024, Lisbon, AA, Portugal

ACM ISBN 979-8-4007-0585-4/24/04…$15.00
https://doi.org/10.1145/3643915.3644109

is twofold, including both a formalization of the problem (that incorporates the forest area, wildfire propagation, wind, smoke, UAV and their mission requirements), as well as the simulator.

After this introduction, Section 2 presents an overview of the wildfire monitoring scenario together with a formalizacion of the problem. Then, Section 3 offers an explanation of the implementation driven by the many variables that can be customized, both for the fire simulation and for the UAV. Section 4 explains and discusses the simulator execution. Finally, Section 5 concludes the paper and enumerates some lines of future work.

2 ADAPTATION PROBLEM

In this section, we first provide a high-level overview of the wildfire tracking scenario and follow with a detailed formalization of the problem embedded into Wildfire-UAVSim.

2.1 Scenario Overview

A team of UAV is tasked with tracking the evolution of a wildfire as it spreads over a forest area. The spread of the fire must be tracked as accurately as possible by maximizing the amount of up-to-date information monitored of the fire front (i.e., areas that have already burnt or are not close to the fire yet are not relevant). Each UAV is equipped with a downwards-facing camera that captures information about the forest surface. UAV also have to conduct the mission in a safe manner by minimizing the chance of collisions among them, and are equipped with sensors that enable them to estimate the position of other UAV. Moreover, UAV may have to operate under challenging conditions that involve strong wind, as well as smoke, which limits their ability to perceive the environment (i.e., their surrounding forest area, as well as other UAV).

2.2 Problem Formalization

2.2.1 Forest Area and Wildfire Propagation. A forest area is represented as a grid S of $N \times N$ cells, where each cell $s \in S$ incorporates two time-dependent variables that represent: (i) the amount of burnable fuel in the cell $F : S \rightarrow \mathbb{N}$; and (ii) whether the cell is burning ($B : S \rightarrow \{0, 1\}$). We assume a discrete notion of time, and for simplicity, we represent the fuel value of a cell s at time instant t as $F_t(s)$ ($B_t(s)$ for B, respectively).

Each cell can experience the following changes when one time unit elapses: (a) part of its fuel is consumed (according to a burning rate β – cf. Expression 1) if the cell is burning, (b) when $F_t(s) = 0$, then the fire is extinguished in that cell ($B_{t+1}(s) = 0$), and (c) when there is fuel remaining in a cell that is not burning ($F_t(s) > 0 \wedge B_t(s) = 0$), we define a probability $p_t(s)$ (Expression 2) of the cell getting ignited based on the proximity to other burning cells.

$$F_{t+1}(s) = \begin{cases} max(0, F_t(s) - \beta) & \text{if } B_t(s) \\ F_t(s) & \text{otherwise} \end{cases} \tag{1}$$

$$p_{t+\mho}(s) = \begin{cases} 1 - \prod_{s' \in S_A(s,d)} (1 - dist(s, s')^{-2} \, B_t(s')) & \text{if } F_t(s) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In Expression 2, $p_{t+1}(s)$ depends on the state of cells adjacent to s, designated by the set $S_A(s, d) = \{s' \in S \mid dist(s, s') \leq d\}$, where $dist : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is a distance function, and $d \in \mathbb{R}^+$ is a distance threshold. In the expression, the term $dist(s, s')^{-2}$ captures an inversely proportional relation, which allows to give more importance to the state of cells that are closer to s. Moreover, we define a fire spread rate parameter $\mho$ that controls the speed at which fire propagates from cell to cell (i.e., $\forall t_i$, with $t \leq t_i < t + \mho \bullet p_{t_i} = 0$).

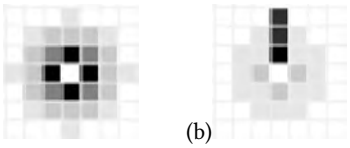

Figure 1: Fire spread probability for a 7×7 grid: (a) no wind, (b) $w = north$ and $\mu = 0.75$. Darker means higher probability.

2.2.2 Wind. We model wind as a bias on each cell's igniting probability and characterize it by its direction $w \in \{north, south, east, west\}$ and strength $\mu \in [0, 1]$:

$$p_{t+1}(s') = \begin{cases} p_t(s') + \mu(1 - p_t(s')) & \text{if } s' \in S_\mu(s, w, d) \\ p_t(s') - \mu p_t(s') & \text{otherwise} \end{cases} \tag{3}$$

In Expression 3, $S_\mu(s, w, d) \subseteq S_A(s, d)$ is the set of cells that are in direction w with respect to cell s. Figure 1 illustrates wildfire spread probabilities for a sample grid with and without wind.

2.2.3 Smoke. We characterize smoke as a function that maps each cell to three time-dependent variables $\kappa : S \rightarrow \{0, 1\} \times \mathbb{N} \times \mathbb{N}$. Hence, for each cell $s \in S$, we have a tuple $(\gamma, \gamma_\alpha, \gamma_\omega)$, where $\kappa_t(s).\gamma$ indicates whether the cell contains smoke or not, $\kappa_t(s).\gamma_\alpha$ is a counter that keeps track of the time between the instant in which a cell ignites and smoke appears, and $\kappa_t(s).\gamma_\omega$ is a counter that keeps track of the time elapsed between the appearance of the smoke in the cell, and its dissipation. To avoid situations in which smoke dissipates before the end of the cell's burning process, we impose the constraint $\forall s \in S, \kappa_{t_i}(s).\gamma_\omega \geq \lceil F_{t_i}(s)/\beta \rceil$, where t_i designates the time instant in which cell s ignites. Please note that we do not consider different levels of smoke density in our model, which is just either present in a cell or not.

2.2.4 Unmanned Aerial Vehicles (UAV). We model a UAV as a tuple $u = (s, A, \delta^O)$, where $s \in S$ is the position of the UAV in the forest area grid, A is the set of possible actions for the UAV, and $\delta^O \in \mathbb{N}$ is a distance threshold that determines the set of observable cells in the forest area for u. The set of UAV is designated by U, and for simplicity, we assume that the set of actions for all UAV is the same and consists in moving to adjacent cells in four directions, i.e., $\forall u \in U \bullet u.A = \{north, south, east, west\}$. Every UAV is equipped with a down-facing camera that is able to monitor a set of cells centered around its position $S^O(u) = \{s' \in S_A(u.s, u.\delta^O)\}$.

2.2.5 Mission Requirements. During operation, the team of UAV should satisfy the set of requirements captured in Table 1. Requirement R1 captures the effective tracking of the wildfire and consists in maximizing the informative value of the set of monitored cells during system operation (metric M_{R1}). In the expression, $\nu : S \rightarrow \mathbb{R}_{\geq 0}$ is a function that assigns a constant informative (i.e., utility) value $C \in \mathbb{R}^+$ to a cell s in the grid if it is burning and not

Table 1: Mission requirements for UAV wildfire tracking.

Id	Description
R1	**Effective wildfire monitoring.** UAV should maximize the informative value of the monitored area in which the wildfire is active during the mission, which is designated by $M_{R1} = \sum_{t=0}^{T} \sum_{s \in \biguplus_{u \in U}(S_t^O(u))} \nu_t(s)$.
R2	**Collision risk avoidance.** The system should minimize the number of events in which UAV are in close proximity (i.e., closer than safety distance threshold δ_s), which is designated by $M_{R2} = \sum_{t=0}^{T} \lvert e_t(U) \rvert / 2$, where $e_t(U) = \{(t, u, u') \in [0, T] \times U \times U \mid u \neq u' \wedge dist(u.s_t, u'.s_t) \leq \delta_s\}$.

covered by smoke (i.e., $\nu_t(s) = C$ if $B_t(s) = 1 \wedge \kappa_t(s).\gamma = 0$), and zero, otherwise. Note that the use of $\biguplus$ in the expression avoids accruing value for the same cell more than once when it is monitored by more than one UAV. Requirement R2 targets safety during operation by defining metric M_{R2}, which counts the number of instances in which two UAV are closer than a safety distance threshold δ_s.

3 IMPLEMENTATION AND CUSTOMIZATION

The user of our simulator will interact with files *main.py* and *common_fixed_variables.py*. While the former executes the simulator itself, the latter is mainly used for setting different parameters for the simulation. There are other Python classes that implement different features of the simulator. All code is documented for a better understandability[1]. In the following, we describe some relevant features and different customization possibilities of the simulator and the UAV.

3.1 Simulation customization

The graphical interface of the simulator is built on the Mesa framework [7], which allows to show on a web page, using JavaScript, a grid with cells of different colors. Class common_fixed_variables.py allows to customize several variables for having different simulation experiences. Table 2 gives an overview of most of these variables—some variable names have been shortened in the table for visualization purposes.

To begin with, variables WIDTH and HEIGHT set the grid size, in number of cells, while BATCH_SIZE is used to establish how long the simulation will run, in number of time steps.

Each cell's color and shape at all times depend on a function called agent_portrayal(), which is in charge of drawing elements such as UAV, fire, smoke, and forest cells. Several variables can be customized for defining color palettes for different types of elements. We define color palettes based on 12 colors, which gives a certain aspect of reality to the simulation according to what is happening in the data model. As for the vegetation colors, they are set by VEGETATION_COLORS, whose default values are displayed in Listing 1. While the first value is used for depicting the absence of vegetation, the following values represent an ascending gradient of green shade, which represent an ascending vegetation density, i.e., amount of burnable fuel. If the data model contains more than 12 different values for the vegetation density, these values are normalized so that they can be graphically represented with the 12-color palette—normalize_fuel_values() function is in charge of this.

[1]The package with code and documentation for Wildfire-UAVSim can be downloaded from: https://github.com/atenearesearchgroup/Wildfire-UAVSim

Table 2: Simulator's customizable parameters

Variable	Type	Measure
WIDTH, HEIGHT	Int	Grid size
BATCH_SIZE	Int	Simulation duration (# timesteps)
VEGETAT_COL	List	12-color palette for depicting vegetation
FUEL_UP_LIMIT	Int	Maximum amount of fuel in a cell
FUEL_BOT_LIMIT	Int	Minimum amount of fuel in a cell
DENSITY_PROB	Double	Overall vegetation density
BURNING_RATE	Int	Fuel burning speed
FIRE_SPR_SPEED	Int	Fire spreading speed
FIRE_COLORS	List	12-color palette for depicting fire
ACTIVATE_WIND	Bool	Wind affects the wildfire spread
FIXED_WIND	Bool	Wind blows one of the 4 cardinal points
WIND_DIRECT	Enum	Only direction the wind blows
FIRST_DIR	Enum	One of the directions the wind blows
SECOND_DIR	Enum	The other direction the wind blows
FIRST_DIR_PROB	Double	Wind's first direction predominance
MU	Double	Wind strength
ACTIV_SMOKE	Bool	Fire generates smoke
SMKE_PD_CT	Int	Smoke pre-dispelling counter
SMOKE_COLORS	List	12-color palette for depicting smoke
PROB_MAP	Bool	Fire spread probabilities are displayed
BW_COLORS	List	12-color palette for spread probabilities

```
VEGETATION_COLORS = ["#414141", "#9eff89", "#85e370", "#72d05c",
                     "#62c14c", "#459f30", "#389023", "#2f831b",
                     "#236f11", "#1c630b", "#175808", "#124b05"]
```

Listing 1: Customizing vegetation color

There is a set of variables for setting the vegetation over the grid. For initializing the vegetation density in each cell, i.e., the burnable fuel amount ($F : S \rightarrow \mathbb{N}$, cf. Section 2.2.1), FUEL_UPPER_LIMIT and FUEL_BOTTOM_LIMIT variables establish the maximum and minimum amount of burnable fuel present in each cell, respectively. Regarding the overall vegetation distribution over the grid, DENSITY_PROB is a value in the range [0, 1] that establishes the percentage of the grid covered by vegetation, i.e., this variable sets the percentage of cells that will have vegetation. Therefore, these three variables determine how vegetation is set all over the grid, as well as its density in each cell, every time the simulator is initialized—different vegetation distributions can appear in different simulations.

Two variables customize how fire evolves. BURNING_RATE (β in our formalization, cf. Section 2.2.1) sets the fuel decay speed in terms of time steps, while FIRE_SPREAD_SPEED ($\mho$ as explained in Section 2.2.1) sets how fast fire spreads to other cells, also in terms

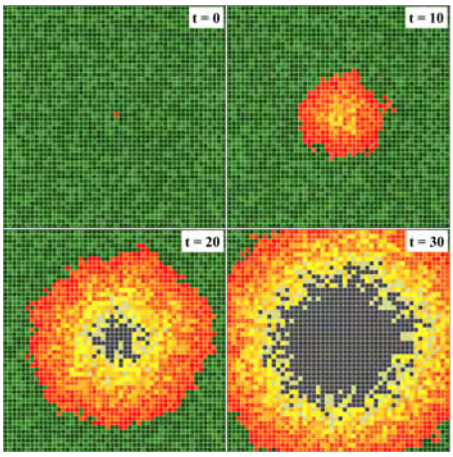

Figure 2: Wildfire propagation in a 50x50 cell grid

of time steps. The formalization of the fire evolution is detailed in Section 2.2.1. As for depicting fire in the grid, variable FIRE_COLORS is used, with a color palette evolving from yellow to red in the default configuration. Figure 2 shows the propagation of a fire where all cells have some vegetation. We can see that different cells have different green shades, representing different vegetation densities. Similarly, fire shows different colors depending on how active it is, while cells where both vegetation and fire have been extinguished are displayed in gray. For showing how the above-mentioned DENSITY_PROB variable affects a simulation, Figure 3 displays a simulation in which the DENSITY_PROB has been set to 0.5. In both simulations, the variables FUEL_UPPER_LIMIT and FUEL_BOTTOM_LIMIT have been set to 7 and 10, respectively.

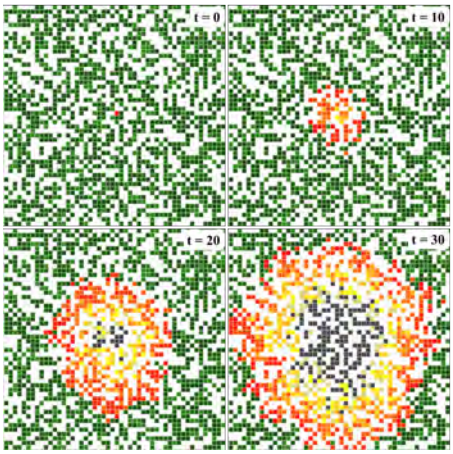

Figure 3: Wildfire propagation in a 50x50 cell grid with DENSITY_PROB = 0.5

Then, there are some variables for customizing wind's behavior. First, ACTIVATE_WIND sets whether the fire spread is influenced by wind. If it is, we can have wind blowing one specific direction out of the 4 cardinal points (for instance, *north*) or it can blow two directions (such as *north-west*). If FIXED_WIND is active, then

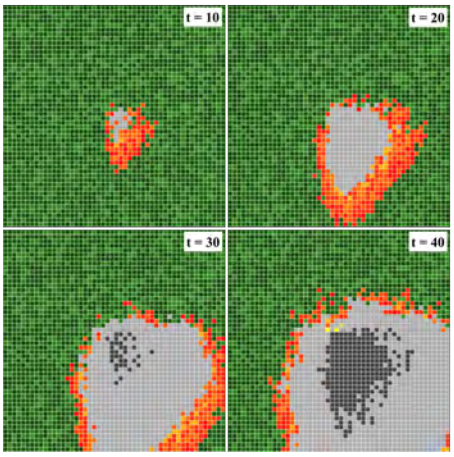

Figure 4: Wildfire propagation in a 50x50 cell grid with wind blowing south-east with FIRST_DIR_PROB = 0.8 and MU = 0.9; and with smoke with SMOKE_PRE_DISPELLING = 2

wind blows in the direction set by WIND_DIRECTION. If it is not, it means wind blows two directions, specified by FIRST_DIR and SECOND_DIR. Since wind can blow a direction stronger than the other one, FIRST_DIR_PROB establishes the wind first direction's predominance. It is a value in the interval $[0, 1]$. This means that wind's predominance in the other direction is 1−FIRST_DIR_PROB. For instance, if wind blows *north-west* and FIRST_DIR_PROB is 0.6, it means that wind is affected by the *north* component in a 60% and by the *east* component in a 40%. MU sets how strong wind blows with a value in the range $[0, 1]$—characterized by μ in the formalization of Section 2.2.2.

Regarding smoke, variable ACTIVATE_SMOKE sets whether smoke will be part of the simulation, and SMOKE_PRE_DISPELLING establishes how fast smoke appears after fire starts in a cell, formalized by $\kappa_t(s).\gamma_\alpha$ in Section 2.2.3. The pace at which smoke dispels depends on the amount of burning fuel ($\kappa_t(s).\gamma_\omega$ in Section 2.2.3), and it is controlled by the dispelling_counter_start_value variable in the Smoke.py class. Variable SMOKE_COLORS is used for representing smoke with an ascending gradient of grey shades, similarly as explained before for VEGETATION_COLORS and FIRE_COLORS. However, in the current implementation, we use only one color for smoke since, as mentioned in Section 2.2.3, smoke in our simulator has a constant density. It is part of our future work to integrate different levels of density for the smoke, which would allow different degrees of partial visibility over the area. Figure 4 displays how the simulation evolves when wind and smoke are present—smoke is represented in light gray, while burnt forest area has a dark gray color. In particular, wind blows south-east with FIRST_DIR_PROB = 0.8 and MU = 0.9. Regarding smoke, SMOKE_PRE_DISPELLING is set to 2.

Finally, users can also decide whether to visualize the probability of the fire to spread to each cell at all times. This is a special visualization mode that can be set by activating PROBABILITY_MAP variable. In this mode, only black, gray and white colors are displayed, according to the color palette specified in a variable named

Table 3: UAV's default parameters

Variable	Measure	Default value
NUM_AGENTS	Number of UAV	3
N_ACTIONS	UAV's possible movements	{n,e,w,s}
OBS_RADIUS	UAV's observation radius	8
side	Side size of observed area	(OBS_RADIUS * 2) + 1
N_OBSERV	Number of cells observed	$side^2$
SEC_DIST	Security dist among UAV	10

BLACK_AND_WHITE_COLOR. Such a visualization mode is like the one shown in Figure 1.

3.2 UAV customization

In Table 3 we can see the different features for the UAV that can be set in common_fixed_variables.py file—some variable names have been shortened in the table. While NUM_AGENTS establishes the amount of UAV that will fly over the forest area (zero indicates the simulator will simulate only the wildfire spread), N_ACTIONS specifies the number of possible actions each UAV can take when deciding on a move, which we set as [north, east, west, south]. Variable UAV_OBSERVATION_RADIUS sets the observation radius—technically it is not really a radius, since observed areas have square shapes. In particular, this measure is used for computing the side of the observation square. Variable N_OBSERVATIONS uses the side to define the number of cells in the observation area of each UAV through their downwards-facing camera, in our case a square area. Finally, SECURITY_DISTANCE establishes the minimum distance that UAV should be separated from each other for avoiding collisions.

As for the way to obtain each UAV's partial observation, we have the state() function, as depicted in Listing 2. The first for loop obtains the amount of burning cells for each agent, through surrounding_states() method. The obtained observation lists contain ones for cells that are burning, and zeros for cells that are not. If an UAV is near an edge, the observed number of cells will be lower—since some observed cells would be out of the grid. However, the Mesa framework forces to constantly have the same number of observed cells. Therefore, the second for loop adds a zero in those positions of the list that would correspond to cells that cannot be observed.

```
1  def state(self):
2      states = []
3      for agent in self.schedule.agents:
4          if type(agent) is agents.UAV:
5              surrounding_states = agent.surrounding_states()
6              states.append(surrounding_states)
7
8      for st, _ in enumerate(states):
9          counter = len(states[st])
10         for i in range(counter, N_OBSERVATIONS):
11             states[st].append(0)
12     return states
```

Listing 2: Function to obtain each UAV partial observation

4 SIMULATOR EXECUTION

4.1 Main loop

For building the graphical interface, the Mesa framework requires instantiating the CanvasGrid class, which holds the simulator logic. This class overwrites the step() method, which is executed every simulation time step, and allows to introduce any behaviour that must be executed repeatedly. This means that each type of agent (such as UAV and Fire agents in our case) implements a step() function for expressing its own logic, which is called from the CanvasGrid step() function. There is also an important Mesa class instantiated as scheduler, which is in charge of executing the main and the agents inner loops, and can be used for checking simulation elements states.

```
1  def step(self):
2      self.datacollector.collect(self)
3
4      # check if simulation ended, if so print MR1 and MR2 overall metrics,
5      # and finish loop. Otherwise, keep executing.
6      if BATCH_SIZE == self.evaluation_timesteps_counter - 1:
7          print(" --- MR1 --- ")
8          print(self.MR1_LIST)
9          print(" --- MR2 --- ")
10         print(self.MR2_VALUE)
11         sys.exit(0)
12
13     if sum(isinstance(i, agents.UAV) for i in self.schedule.agents) > 0:
14         state = self.state()  # s_t
15         # self.new_direction is used to execute previous obtained a_t
16         self.new_direction = [SYSTEM_RANDOM.choice(range(0, N_ACTIONS))
17                               for i in range(0, self.NUM_AGENTS)]  # a_t
18
19         # TODO: algorithm/s calculation with partial state
20         # reward = self.algorithm(state) # r_t+1
21
22         # TODO: an EXAMPLE of analytics extraction can be seen.
23         # However, your own implementations can be applied as well.
24         self.MR1(state)
25         self.MR2()
26
27         self.set_drone_dirs()
28
29     self.evaluation_timesteps_counter += 1
30     self.schedule.step()
```

Listing 3: Simulator main loop function

An example of the main step() method can be seen in Listing 3. This example contains pieces of code that serve as a guide for implementing own functionalities. Lines 2 and 30 are used for the scheduler management, which needs to be inside the main step() method. In line 29, the evaluation counter is incremented to update the current simulation time step. Line 13 uses the scheduler for checking the amount of existing UAV in the simulation. If there are no UAV, the simulation keeps executing. Otherwise, line 14 is used to call the previously described state() method. Line 19 would allow to obtain and execute new actions, based on each UAV partial observation. Since, ideally, the observation of a team of UAV should comply with the mission requirements explained in Section 2.2.5, line 20 is an example of how rewards from the UAV's

performance could be collected for obtaining statistics about the algorithm performance, as well as other desired metrics.

As a default example of how an UAV team could make and execute actions, a random selection example is shown in line 16. In particular, it sets a list of actions for moving the UAV team, which in this case is randomized, i.e., each UAV decides its next action randomly, without taking into account the fire. Also, set_drone_dirs() method (line 27) is used to manage the directions obtained from the new_direction attribute, and make the UAV team move over the forest area. An example of how a setting such as this one with three drones (NUM_AGENTS = 3) would evolve in time is displayed in Figure 5. We can see that the UAV are displayed as black cells, and their monitored area is envisioned with dashed squares.

4.2 Obtaining simulation results

In the main loop it is also included how to compute the simulation results in terms of the effectiveness of the UAV's observation. For this, the mission requirements described in Section 2.2.5 and Table 1 are used. In Listing 3, the calculation of the metrics described in the table is performed in lines 24 and 25. Line 24 executes the self.MR1() method, allowing to obtain the effective wildfire monitoring metric (M_{R1}), in each time step, passing state as a parameter. Similarly, line 25 executes self.MR2() method for obtaining the collision risk avoidance metric (M_{R2}), for which the SECURITY_DISTANCE variable is used (cf. Section 3.2). This parameter keeps track of how often UAV overpass their security areas with respect to the other UAV. Finally, lines 6-11 compose an if statement for printing the final metrics calculated over all simulation time steps, and finishing the program when the counter reaches the end of the simulation.

4.3 Discussion

Please note that the UAV's random movements explained above represent a very simple and not realistic setting, but it serves for demonstration purposes. More sophisticated algorithms can be added to our simulator for complying with the mission requirements explained in Section 2.2.5. For instance, we have developed two different algorithms for managing the UAV (described in [10]). One of them is an extension and adapted version of a state-of-the-art reactive approach based on Deep-Q-Networks (DQN) that solves an analogous adaptation problem [4]. The other algorithm is our own implementation based on Predictive Coordinate Descent (PCD) [9], which is used primarily in machine learning and data analysis. Our simulator allows to visualize how the UAV behave in different scenarios depending on the chosen algorithm. Besides, thanks to the rewards obtained by both algorithms, it is also possible to evaluate which algorithm is more effective in different scenarios with different observability conditions. In particular, we performed experiments in which both PCD and DQN were evaluated under four scenarios: (i) normal operating conditions (no wind, no smoke), (ii) harsh operating conditions (wind, no smoke), (iii) partial observability conditions (smoke, no wind), and (iv) harsh and partial observability conditions (wind and smoke).

Finally, it is worth mentioning that the MESA framework allows to perform simulations without a graphical interface. Evidently, this saves computation time, so it is very useful when we want to quickly obtain the results of the simulation. For instance, this was very handy when we compared the performance our PCD and DQN implementations, since we needed to perform the average execution time of more than 30 simulations so that times were representative. This is also of use when comparing the effectiveness of both algorithms because, again, we are only interested in their rewards at the end of the executions.

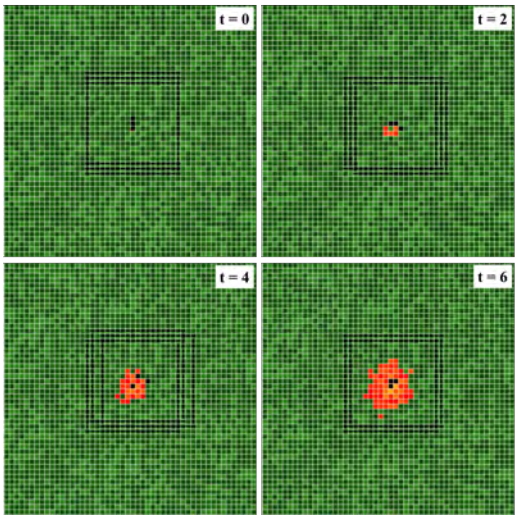

Figure 5: Random action selection for the UAV team

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented Wildfire-UAVSim, a customizable simulator of wildfires over a forest area that can be monitored by unmanned aerial vehicles (UAV). We have formalized the problem of the fire evolution by also considering wind and smoke, and have also formalized the UAV's movements as well as their mission requirements. We have also described the parameters than can be customized in our simulator and the possibilities for enriching the UAV with different behaviors for their monitoring tasks.

We plan to make our wildfire simulation more realistic. At the moment, smoke has a constant density, so we plan to consider different density levels, which means different degrees of visibility. We also plan to include slopes in the forest area that will make fire spread differently in hills and flat areas. Beyond different vegetation densities, we also plan to consider different types of vegetation. For instance, oaks resist fire better than pines, which burn quickly.

ACKNOWLEDGEMENTS

Work partially funded by the Spanish Government (FEDER, Ministerio de Ciencia e Innovación–Agencia Estatal de Investigación) under projects TED2021-130523B-I00 and PID2021-125527NB-I00.

REFERENCES

[1] Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, and Frantisek Plasil. 2013. DEECO: an ensemble-based component system. In *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*. 81–90.

[2] Simos Gerasimou, Radu Calinescu, Stepan Shevtsov, and Danny Weyns. 2017. UNDERSEA: an exemplar for engineering self-adaptive unmanned underwater

vehicles. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 83–89.

[3] Ilias Gerostathopoulos and Evangelos Pournaras. 2019. TRAPPed in traffic?: a self-adaptive framework for decentralized traffic optimization. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Marin Litoiu, Siobhán Clarke, and Kenji Tei (Eds.). ACM, 32–38. https://doi.org/10.1109/SEAMS.2019.00014

[4] Kyle D. Julian and Mykel J. Kochenderfer. 2018. Distributed Wildfire Surveillance with Autonomous Aircraft using Deep Reinforcement Learning. *Journal of Guidance, Control, and Dynamics* 42 (10 2018), 1768–1778. Issue 8. https://arxiv.org/abs/1810.04244v1

[5] Filip Krijt, Zbynek Jirácek, Tomás Bures, Petr Hnetynka, and Ilias Gerostathopoulos. 2017. Intelligent Ensembles - a Declarative Group Description Language and Java Framework (Artifact). *Dagstuhl Artifacts Ser.* 3, 1 (2017), 06:1–06:3. https://doi.org/10.4230/DARTS.3.1.6

[6] Paulo Henrique Maia, Lucas Vieira, Matheus Chagas, Yijun Yu, Andrea Zisman, and Bashar Nuseibeh. 2019. Dragonfly: a Tool for Simulating Self-Adaptive Drone Behaviours. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 107–113. https://doi.org/10.1109/SEAMS.2019.00022

[7] David Masad, Jacqueline Kazil, et al. 2015. MESA: an agent-based modeling framework. In *14th PYTHON in Science Conference*, Vol. 2015. Citeseer, 53–60.

[8] Gabriel A. Moreno, Cody Kinneer, Ashutosh Pandey, and David Garlan. 2019. DARTSim: an exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Marin Litoiu, Siobhán Clarke, and Kenji Tei (Eds.). ACM, 181–187. https://doi.org/10.1109/SEAMS.2019.00031

[9] Yu Nesterov. 2012. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization* 22, 2 (2012), 341–362.

[10] Enrique Vilchez, Javier Troya, and Javier Cámara. 2024. Towards Proactive Decentralized Adaptation of Unmanned Aerial Vehicles for Wildfire Tracking. In *2024 IEEE/ACM 19th Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. To Appear..

[11] Jochen Wuttke, Yuriy Brun, Alessandra Gorla, and Jonathan Ramaswamy. 2012. Traffic routing for evaluating self-adaptation. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 27–32.