SAPIENZA
UNIVERSITÀ DI ROMA

The report on a project
for Artificial Intelligence course

Multi Agent Systems (MAS)

"Fire Control"

i) Task allocation strategy:
Contract Net Protocol (CNP) - Decentralized Contract Net Protocol

ii) Inside-task exploration strategy:
Random Walks (RW) - Selection of valid cells

Project and report are done by: Nazgul Mamasheva
Supervisors: Prof. Daniele Nardi, Albani Dario

Task allocation strategy:

Decentralized CNP: many managers, where each manager with UAVs in communication range compose one entity. Each entity applies Centralized CNP for its auctions.
Let’s assume there is a manager named K, it has two UAVs (KN_1, KN_2) in communication range. K can make bids and proposes to itself without sending packets to itself.

K can make auctions in two ways (winner utilities are denoted by green):

Table 1: Auctions for each task one by one

Auctioneer K	Task_1	Task_2	Task_3
KN_1	0.3	0.8	0.06
KN_2	0.2	0.1	0.09
K	0.5	0.9	0.7
Winners	KN_2	KN_1	K

Table 2: Auctions for all tasks at once

Auctioneer K	Task_1	Task_2	Task_3
KN_1	0.3	0.8	0.06
KN_2	0.2	0.1	0.09
K	0.5	0.9	0.7
Winners	K	KN_2	KN_1

In first scenario, auctioneer will send bids for specific task to all bidders. Then auctioneer will wait for all proposes for that task and will choose a winner. As shown in Table 1 above, for Task_1 the winner will be KN_2, it has the smallest utility result. Then for second auction, i.e. for Task_2 the bidder with the smallest utility (0.1) is KN_2, but this bidder already got a task, then winner will be the next available bidder with smallest utility propose, it is KN_1 with 0.8. And last winner will be K itself.
The problem here is that KN_1 instead of moving to its nearest task Task_3 will move to the farthest task Task_2. And KN_2 instead of moving to its nearest task Task_3 will move to its farthest task Task_1.

The second scenario, Table 2 can be a good option to solve such issue which was implemented in this program. The auctioneer will make auctions for all tasks at once. Then it will wait for all proposes. After getting all proposes, it will choose the smallest utility among all proposes, here it is 0.06. The first winner will be KN_1 for Task_3. The second smallest utility is 0.09, but auctioneer can’t assign two bidders to the same task. It this example, each task can get only one winner. So the auctioneer will search for next smallest utility and it is 0.1, auctioneer will check does this task already have a winner, in this example - no, then the winner for Task_2 will be KN_2. The next smallest utilities are 0.2 and 0.3, but their owners are already winners. The auctioneer will skip them and stop on 0.5, its owner is available, then the winner will be K, for Task_1.

As results show, the second option is better. The utilities for two scenarios remain the same, but results of two scenarios show that winners can be completely different from each other for each task.

Utility function: an UAV needs to calculate the cost of its performance according to a task. When an agent gets a bid, it will call this function to calculate the utility and send the result to auctioneer as a propose.

u(Agent, Task) = 0.7 * (numMoves / maxMoves) +

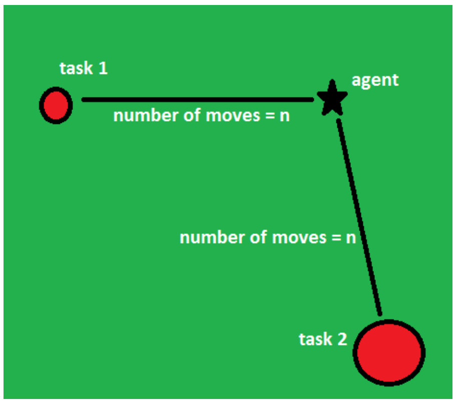
0.2 * (1 – taskSize / maxTaskSize) +

0.1 * (1 – taskRadius / maxTaskRadius)

0.7, 0.2 and 0.1 are weights, where 0.7 + 0.2 + 0.1 = 1.0

0 <= u(Agent, Task) <= 1

The number of moves from agent's current position to task's centroid position is the most important criteria for utility, therefore it has 0.7 weight out of 1.0.



The Utility Function could consist of only one part (number of moves). But size and radius of task are also should be taken into account. There can be such scenario, where agent has to choose one of two tasks, where number of moves from agent to two tasks are the same. Then we can see that decisive factors will be the task's size and radius. In such situation, agent should choose the task with big size and large radius, because such task(fire) is more dangerous, it can ignite more cells.

Here we can see that size and radius are also important for calculating an utility of agent to a specific task. Therefore, these two criteria also included in utility function calculation.

Tasks priorities: this is a solution to a question "how many agents should go for a task?" from project specification. The function defines a heuristic that allows to deploy the right number of agents to every task.

First, we need to define the priority of each task according to its size and radius. The task with largest size and radius has the highest priority and so on. Then it makes some calculation:

w = numUAVs / numTasks (numUAVS - number of all UAVs)
r = numUAVs % numTasks (numTasks - number of all Tasks)

Each task has to get a minimum number of UAVs, so each task will get “w” number of UAVs. Then the “r” tasks with highest priorities will get by “1” UAVs. For example, numUAVS = 23, numTasks = 5

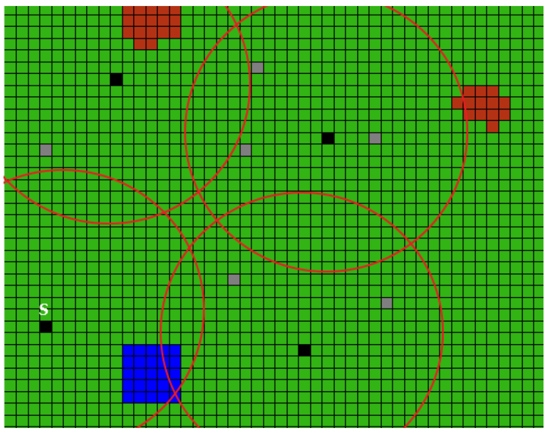
int w = 23 / 5 4

int r = 23 % 5 3

Tasks order (by priorities)	Task_1	Task_2	Task_3	Task_4	Task_5
Number of UAVS	4 + 1	4 + 1	4 + 1	4	4

Each manager will apply this function in order to assign the right number of UAVs to each task.

Defining managers: when program starts it will first define managers. The managers are those agents (UAVs) with maximum number of neighbors. First, for each agent there will be an iteration of all agents, where it will count the number of neighbors in communication range. Then the results of agents will be compared and program will choose the agent with maximum number of neighbors, this will be assigned with “manager” status, all agents which are in list of this manager and they could not be managers in that area. It will continue until no agents will left in list.



Some agents can be a manager, even if it does not have any neighbors, because no other managers can send bids to this agent. In program code this kind of managers will be assigned with “_manager” status. It will bid and propose to itself.

Example for _manager with no neighbors, denoted by “S”

Inside-task exploration strategy:

Selection cells: after choosing a task, UAVs will choose valid cells, which will increase efficiency of extinguishing process. The centroid of a task will be a very first position of each agent. This position will be saved in a local knowledge of an agent. In next step agent will choose the nearest cell of the last saved cell in its local knowledge. It will be one of 8 positions.

If agent on position (x, y), then nearest cells will be:

(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	(x, y)	(x+1, y)
(x-1, y+1)	(x, y+1)	(x+1, y+1)

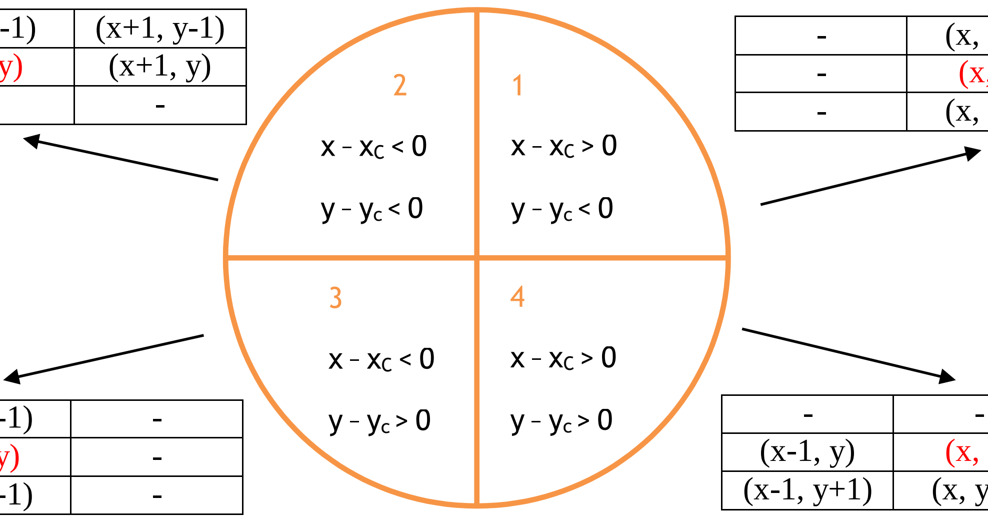
Randomly agent will choose one of 8 options. The program each time will check a new position for not being out of boundary of a field and for being in area of a task. If nearest cells are already extinguished or normal, then program will gradually increase the distance between last cell position and new cell position.

After while, when cells will start burning and agent's next position will coincide with burned cell, the agent will use another strategy:

The task(fire) has a circular kind of expansion:

(x-1, y-1)	(x, y-1)	(x+1, y-1)
(x-1, y)	(x, y)	(x+1, y)
-	-	-

-	(x, y-1)	(x+1, y-1)
-	(x, y)	(x+1, y)
-	(x, y-1)	(x+1, y+1)



(x-1, y-1)	(x, y-1)	-
(x-1, y)	(x, y)	-
(x-1, y+1)	(x, y-1)	-

-	-	-
(x-1, y)	(x, y)	(x+1, y)
(x-1, y+1)	(x, y-1)	(x+1, y+1)

x: x coordination of new position
y: y coordination of new position

x_c: x coordination of centroid position
y_c: y coordination of centroid position

If agent's new position will be a burned cell, then it will calculate $x - x_c$ and $y - y_c$. And then will define to which quadrant it belongs to. For example, for 1st quadrant agent can choose only 5 positions, i.e. agent should move to the north-east side, but not to the west side. Because on the north-west side most probably cells around are already burned and it does not make any sense to move to that side. For 2nd quadrant – to the north-west side, for 3rd quadrant – to the south-west and for 4th quadrant – to the south-east.

Results evaluation:

Number of UAVs	Number of Tasks	Communication Range	Burned Cells
2	3	30	3233
3	3	30	3080
4	3	30	2753
8	3	30	2525
15	3	30	1708
5	2	30	2838
5	3	30	2818
5	4	30	2925
5	8	30	2894
5	15	30	2923
8	3	10	2670
8	3	20	2398
8	3	40	2417
8	3	50	2551
8	3	60	2709

The results also depend on UAVs' and tasks' positions, which are random each time. Therefore, the number of burned cells sometimes differed significantly for each combination of parameters and average results were chosen as demonstration results.

1) As we see, with constant number of tasks (3) and constant communication range (30), if number of UAVs will increase, the number of burned cells will decrease.

2) With a constant number of UAVs (5) and constant communication range (30), if the number of tasks will increase, the number of burned cells will overall increase a little bit.

3) With constant number of UAVs (8) and constant number of tasks (3), if radius of communication range will increase, the number of burned cell will vary.

So, according to the results the number of burned or distinguished cells highly depend on number of UAVs, rather than the communication range.