SAPIENZA
UNIVERSITÀ DI ROMA

Report on a project

for Artificial Intelligence course

Probabilistic Reasoning and Learning

"Fire Control"

Reinforcement Learning (RL)

Dynamic Q-learning with Experience Replay (DQLER)

Nazgul Mamasheva

Introduction

In my previous course project, I worked on a “Fire control” project. This project involved developing a Multi-Agent System that simulates wildfires and unmanned aerial vehicles (UAVs) tasked with extinguishing these wildfires. In the project, when an agent (UAV) approaches a chosen task (wildfire), it has to explore the area before initiating the extinguishing. The primary objective of the project was to implement a Control Network Protocol (CNP) simulation, enabling agents to communicate, conduct auctions for tasks (wildfires), strategically choose tasks, and subsequently extinguish the selected wildfires. The project involved conventional programming components without incorporating any machine learning methods.

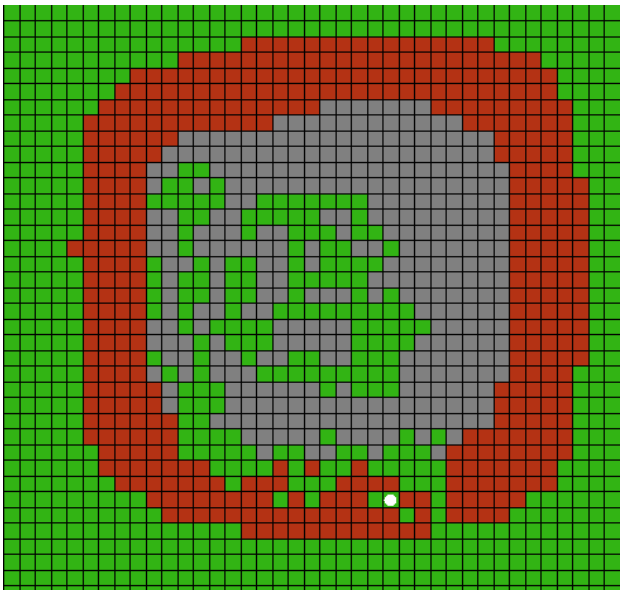
This project experience prompted me to consider the potential benefits of training the agents (UAVs) to intelligently explore and extinguish wildfires. By leveraging machine learning, an agent could learn effective strategies for approaching and addressing wildfires, taking into account their tendency to spread in specific patterns. This could enhance the overall efficiency and effectiveness of wildfire control efforts.

Description of the problem

The world is a two dimensional grid of size w and h. Each cell has the following properties:

- Position, a pair (x,y) identifying the position of the cell
- Type, each cell assumes one of the following types:
    - NORMAL, cell is in a good shape and trees are safe
    - FOAMED, a fire has been previously extinguished and trees are safe
    - FIRE, fire is devastating the cell
    - BURNED, the cell is completely lost

Cells normally start with status NORMAL. After a while, a fire starts to propagate, cells move from NORMAL to FIRE. Cells reached by UAVs in time, are treated with a special foam that can extinguish the fire. After the treatment, a cell cannot be ignited again; cells treated with the special foam move from FIRE to FOAMED. After a certain time, a burning cell moves from FIRE to BURNED and cannot go back to NORMAL status again.



White circle is an UAV; red cells are trees on fire; green cells are either normal trees or foamed (extinguished) trees; gray cells are burned out trees with no possibility to be recovered.

UAV has limited perception range: the camera mounted on UAV has the footprint that coincides with the size of a cell; i.e. it perceives only the status of the cell above which it is located. Agents store their own local knowledge as a collection of known cells. When a cell is inspected from a close range (i.e. at the level of a single cell), the agent automatically updates their knowledge with the information about current location. An agent has to select a cell within the set of available cells coming from those that lie in the area of the agent's current task.

Formal model of the problem

The state space of the problem is S = { s_1, s_2, … , s_k }, where S is a set of states, s_k is the cell on the field with (x, y) coordinates and k is m*n.

If the width and height of the field are m and n respectively, i.e. width = m and height = n, then number of states will be m*n.

A is a set of actions, where $a_t \in A$ is the action executed by the agent at time t.

A = { Upper-Left, Up, Upper-Right, Left, Right, Down-Left, Down, Down-Right }

The max number of possible actions from current state to next state is 8, i.e. neighbor cells of the current cell. The number of possible actions might be less than 8, if some of the neighboring cells are invalid.

(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)

P_a (s, s') is the transition function, performing an action at state s, the agent arrives at state s'. In dynamic environment the transition function is still deterministic, i.e. P^t (s, s') = $P^{t+\Delta t}$ (s, s').

R_a (s, s') is the reward function, indicating the reward agent receives after performing action a at state s leading to s'. r_t indicates the reward received at time t. In dynamic environment the reward function is not always deterministic, i.e. R^t (s, s') ≠ $R^{t+\Delta t}$ (s, s').

Q-function:

- α (alpha), α ∈ [0, 1] is a learning rate.
- γ (gamma), γ ∈ [0, 1] is the discount factor, indicating the priority of immediate rewards compared to later rewards.
- e (epsilon), γ ∈ [0, 1] an exploration probability for an e-greedy method.

Solution algorithm

For this dynamic environment problem as a solution was implemented "Dynamic Q-learning with Experience Replay" (DQLER) algorithm.

Q-learning algorithm: Q-learning uses its last experience to update its policy.

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha[r_t + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)]$$

e-greedy exploration method: e-greedy is used along with Q-learning, with e ∈ [0, 1].

$$a_t = \begin{cases} \arg\max_a Q(s_t,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Experience Replay algorithm: Experience Replay uses the same Q-learning function, but with a different approach for updating policy. Instead of using the last experience to update the policy, it will store experiences in a buffer for later use. Those experiences in the buffer later can be used multiple times for updating the policy.

Input: RL algorithm L, exploration probability ϵ, discount factor γ, learning rate α, number of experiences to replay N, number of replays K, number of iterations before learning Z

1: $Q \leftarrow Q_0$ // Initialize Q-function
2: $M \leftarrow \emptyset$ // Set of experiences to replay
3: **while** not reached stopping criterion **do**
4: $c = 0$ // iteration counter
5: **for** $t = 1, 2, .., T$ **do**
6: $a_t = \begin{cases} \arg\max_a Q(s_t, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
7: perform action a_t
8: observe new state s_{t+1} and reward r_t
9: $E \leftarrow \{s_t, a_t, r_t, s_{t+1}\}$
10: $M \leftarrow update(M, E, N)$
11: **end for**
12: $c = c + 1$
13: **if** $c = Z$ **then**
14: $Q \leftarrow L(Q, M, K, \gamma, \alpha)$
15: $c = 0$ // restart counting
16: **end if**
17: **end while**

on the line 10 in above algorithm, updating M will be executed as algorithm below. Instead of saving all experiences $E = \{ s_t, a_t, r_t, s_{t+1} \}$ to buffer M, it will save last N experiences. If M is full, then the least recent experience will be removed from M and the most recent experience will be added to the buffer as a last element.

Input: Database M, experience E, number of experiences N

1: **if** $size(M) < N$ **then**
2: $M \leftarrow M \cup E$
3: **else**
4: remove least recent element from M
5: $M \leftarrow M \cup E$
6: **end if**

Output: M

Dynamic Q-learning with Experience Replay (DQLER): DQLER is based on Q-learning and Experience Replay with a modified e-greedy exploration strategy for dynamic environments. It uses the Experience Replay algorithm above.

The agent tends to go back to visited locations multiple times. For this project time is crucial, therefore execution should prevent agents from revisiting already visited states multiple times.

On the line 6 in Experience Replay algorithm instead of using standard e-greedy method, it will use another strategy below: A_n – actions at state s_t that lead to states which are not visited by agent. A_v – actions at state s_t that lead to states which are already visited by an agent. If the buffer of visited states A_n is not empty, then strategy will choose the action among all actions in A_n which has the maximum Q. If A_n is empty, it will choose action from A_v which has the maximum Q.

$$a_t = \begin{cases} \left.\begin{array}{l} \arg\max_a Q(s_t, a \in A_n) \text{ if } A_n \neq \emptyset \\ \arg\max_a Q(s_t, a \in A_v) \text{ if } A_n = \emptyset \end{array}\right\} & \text{w.p. } 1-\epsilon \\ \text{random action} & \text{w.p. } \epsilon \end{cases}$$

Implementation

Data structure:

If the width and height of the field are m and n respectively, i.e. width = m and height = n, then number of states will be k = m*n. Number of actions from one state to another state is maximum 8. It might be less, if two states are not neighbors by position.

Q matrix	action_1	action_2	…	action_8
state _1				
state _2				
…				
state _k				

There is no reward matrix, because the environment is dynamic, which means the reward of each cell will change over time. After each step iteration the simulation updates each cell's status in the forest field. So, if the program needs to get a reward from the exact cell, it will check the status of that cell at that time and if, for example, the cell's status is fire, then reward will be 1.0. Agents can move only within the area of the wildfire, cells of which can be on fire, extinguished or burned.

Type of the cell	Reward
CellType. FIRE	1.0
CellType.EXTINGUISHED	-0.25
CellType. BURNED	-0.5

Static variables:

N – number of experiences to store in M.

M – buffer for store last N experiences.

Z – number of simulations before updating Q matrix.

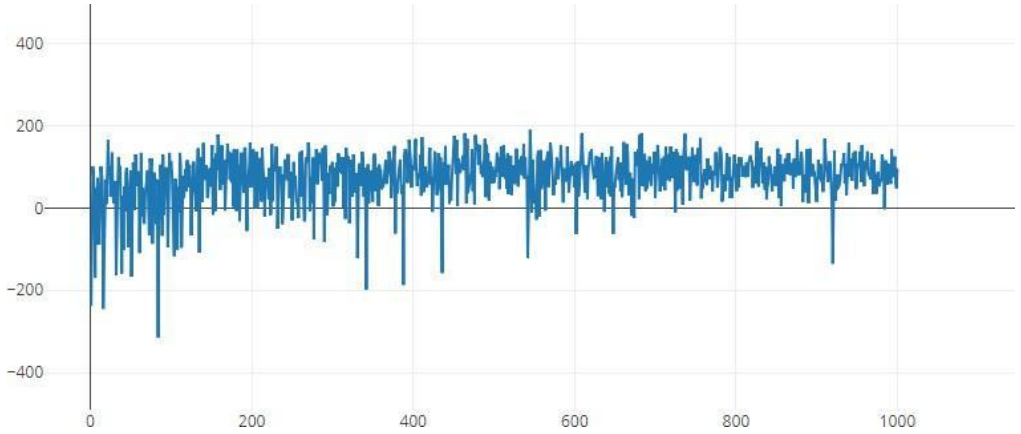
K – mini-batch, after each Z simulation an experience from M will be chosen randomly K times in order to update the Q matrix.

Main implementation details:

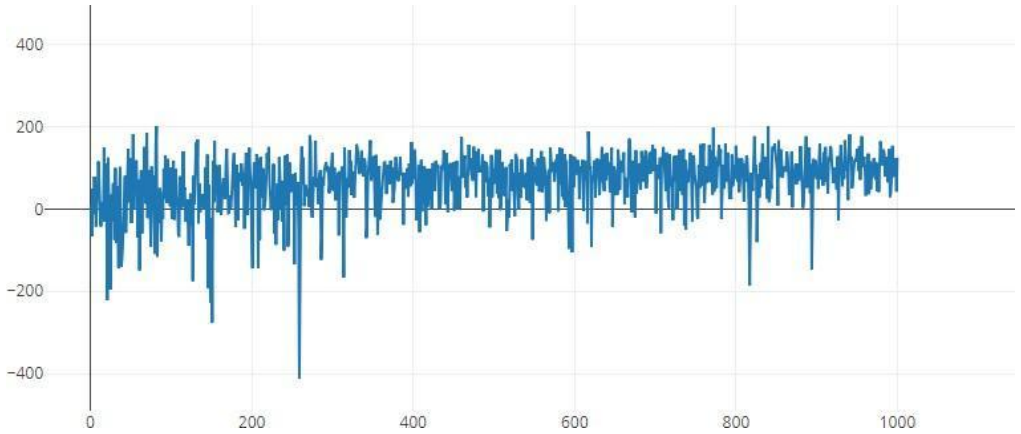
The project was done in the MASON library for simulation, on java 8. The implementation was done by DQLER algorithms which were described above. There was added a new file Buffer.java with simple class for storing experiences and deleted DataPacket.java file from initial files, because for this project we don't need to implement Contract Net Protocol simulation, so we don't need this script. The reinforcement learning components were implemented in two scripts, they are UAV.java and Ignite.java. The other given initial scripts were not changed. The model and graphic parts are strictly separated in MASON. So, if I want to run on the console I will enter one command, if for visual representation another command. For this project only one agent was involved. For each simulation, the agent's first move will be the origin of the wildfire, i.e. centroid position of the task (fire).

Experimental evaluation

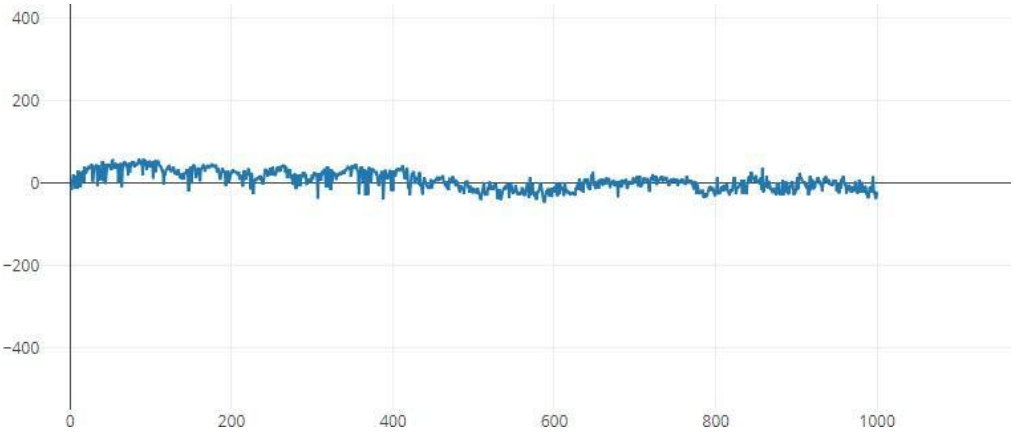
	Graph-1: DQLER	Graph-2: DQLER	Graph-3: Q-Learning
Number of simulations	1000	1000	1000
α (alpha)	1.0	0.1	0.1
γ (gamma)	0.99	0.9	0.9
e (epsilon)	0.1	0.1	0.1
N	50	100	-
Z	1	1	-
K	10	10	-



Graph-1: DQLER



Graph-2: DQLER



Graph-3: Q-Learning

Discussion of the results

1st and 2nd graphs show us there are some learning progresses by using DQLER. The agent doesn't learn fast and it makes some progress gradually. Graph 1 shows better performance than Graph 2. It might be because of the large learning rate $\alpha = 1$ and large discount factor, which shows the importance of the next state's policy. And the 3rd graph shows that Q-learning fails for this problem. Q-learning is not successful in a non-stationary environment.

Conclusion

Working on a project that involved a dynamic system with machine learning components was a captivating experience. I gained a deeper understanding of reinforcement learning methods in machine learning. The project's scope could be expanded by incorporating more agents (UAVs) and facilitating knowledge sharing among them about the explored areas. A thorough examination to identify and address weaknesses in specific scenarios could enhance the implementation. Additionally, exploring the integration of other learning techniques for a dynamic environment would add further depth to the project.