PIR : Mixed-initiative mission

(Final report)

Vagneron Thomas*, Caroline Carvalho Chanel†,Raphaelle N. Roy† and Nicolas Drougard†
*Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: {thomas.vagneron}@student.isae-supaero.fr
†Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: {caroline.chanel,raphaelle.roy,nicolas.drougard}@isae-supaero.fr

***Abstract*—Missions involving robots are prone to be more and more present, as the use of machines and their autonomy increase rapidly. In Mixed-Initiative missions, *i.e.* missions benefiting from the skills of both a robot and a human operator, the latter is often thought as omniscient in the literature. However many examples show that human factors are involved in mission failures. Our project is thus to improve human-robot missions by taking into account potential weaknesses of human operators instead of considering them as always reliable. For this end we propose to use Probabilistic Planning (MOMDP) and Reinforcement Learning in order to compute a strategy meant to optimally inform the operator during the mission and allocate tasks between human and robot. The proposed proof of concept mission concerns a human-robot team which has to extinguish fires. This report is dedicated to the robot actions computation given to human behavior. Simulation environments representing the firefighter mission have been developed and agents have been trained on them. The general project is first described to highlight the problem as well as the proposed solution and tools. Then we introduce Reinforcement Learning and give some details about the algorithm TD(λ), improving strategies after some simulations of any environment. Finally the developed simulation environments are described along with the scores obtained by a TD(λ) agent on it for different parameters.**

I. Context

The increasing autonomy of robots led to make use of them in a more and more large range of operations and activities, for instance in rough places for surveillance or inspection of contaminated sites. Because the machines cannot be held responsible in case of failure and do not possess the human creativity to improvise, those activities require the assistance of humans. In a mission led by an autonomous robot under the surveillance of a human operator or pilot, the latter is often viewed as a providential agent, capable of resolving any problem and responding to a failure of the robot instantly. But many factors can also lead to a failure of the mission, regarding the human intervention, such as a misinterpretation of the information given to the pilot (due to e.g. a poor designed user interface) or the tunneling of the human operator on a certain task or point of the mission [2]

Regarding the state of the art, the "Fitts list" [1], which is a compilation of the general strengths and weaknesses of machines and humans, was traditionally used to determine the function allocation between human and machines. The allocation function is the function that defines the actions that humans and machines are allowed to do, i.e. their role during the mission. In order to determine the level of human versus autonomy involvement, it is important to know the individual strengths of each actor. On the one hand, humans are capable of complex social interactions, moral judgment, operate well in dynamics environments and are highly flexible. On the other hand, the machines are capable of higher computational performance, produce repeatable results, can handle multiple tasks simultaneously and are not prone to fatigue or boredom [10].

A problem still lies in the fact that the operator is considered as an external help, capable of resolving all the problems encountered by the machine. In fact, the human operator should no be considered like this, but as a part of a team. Because of the previously mentioned issues : the human operator is not always reliable. He is subject to boredom, fatigue, stress and many others factors that can contribute to lessen its efficiency during the mission. Considering this fact, the operator should be monitored to consider if he is capable of making accurate decisions, and allow the use of countermeasures (visual stimuli or alarm).

Our idea is to use Probabilistic Planning in order to decide when the system produce countermeasures for the human-operator, and distribute the tasks between human and robot.

While we know all the information about the machine and its states, we know nothing about the states of the human operator. It has been shown in [4] that the use of a Mixed Observability Markov Decision Process (MOMDP) model [6] was relevant to compute a useful policy taking into account the partially observable cognitive state of the operator. MOMDP are a planning framework for computing optimal sequence of actions over time. The action sequence is called policy.

The cognitive ability of the human operator can be inferred via an eye-tracker device [9], which shows areas where the operator focuses its attention. Also, the operator's heart beat states, observable via an electrocardiogram (ECG), are successful indicators of cognitive load [11]. The ECG can be coupled with an Eye-Tracker (ET) in order to detect if the operator is in a state of tunneling, and able to take accurate decisions [5]. These data define the observations of the MOMDP.

As mentionned previously, our assumption is that the human operator is not reliable. Our goal is to optimize the mission of a human-robot team using observations from this system. As

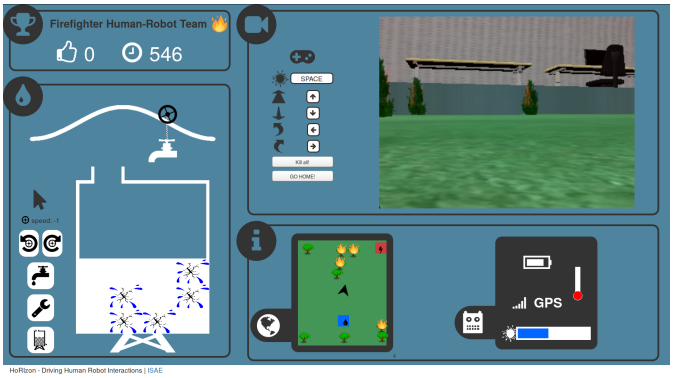

Fig. 1: View of the user interface

a proof of concept, we defined a mission of type "search and fight", which consists in extinguishing fire that will spread over trees in a area with the help of a robot and a user interface.

The number of fires, the battery level or water level are stressing elements. The navigation is a shared task. The ultimate goal of the mission is the optimization of the number of extinguish fire. The MOMDP model will be built using data from a website (i.e. the user interface) with a simulation of the environment. The work developed in this report concerns the optimization of robot's actions given a human behavior. We proposed to compute accurate strategies for the robot by using Reinforcement Learning (RL).

II. Theory

Firstly, let us give a little bit of history about RL : the birth of RL came from the encounter in the late 70s between the computational neurosciences (reinforcement of the synaptic weights of neuronal transmission) and the experimental psychology (animal conditioning models : reinforcement of the behavior leading to satisfaction, *e.g.* Pavlov's dogs experiment) with an adequate mathematical environment : dynamic programming.

A link with experimental psychology is the law of effects of Thorndike (1911) [3] : "Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur : those which accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond".

And so, what is Reinforcement Learning? It is an approach, here computational, to decision-making, understanding and goal-directed learning. The individual learns from direct contact with the environment. It only needs the definition of the interactions between a learning agent and its environment, in terms of states, actions and rewards. The states represent the description of the environment. The actions are the possible agent's actions. Finally the rewards define the goal of the mission given to the agent.

The value and value functions are concepts that are key features of RL methods. Value functions are essential for an efficient search in the space of policies. The value functions are defined over the state space :
for $x \in X, V(x)$ is the mean of the sum of the rewards over time for a process starting in x.

$$V^{\pi}(x) = \mathbb{E}[\sum_{t \geqslant 0} r(x_t, \pi(x_t))|x_0 = x]$$

Continuing on that point, the objectives of RL are : the automatized acquisition of skills for decision making in a complex and uncertain environment and learning by "experience" a behavioral strategy (named policy) reflecting the failures and success (reinforcements or rewards).

Some examples of achievements of Reinforcement learning are :*cartpole* (inverted pendulum on a controlled cart), the production of the best player of backgammon, chess and more recently go (algorithm *alphago*) [7], etc.

To begin with the theory, we need first to understand what are a Markov chain, a Markov Decision Process (MDP) and dynamic programming [3].

A Markov chain is a dynamic system with discrete time $(x_t)_{t \in \mathbb{N}} \in X$, where X is space of states such as:

$$\mathbb{P}(x_{t+1} = x|x_t, x_{t-1}, ..., x_0) = \mathbb{P}(x_{t+1} = x|x_t).$$

Indeed the Markov property states that all the useful information for the future prediction is in the present state. A Markov chain in X is defined by a initial state x_0 and the probability of transition $p(y|x) = \mathbb{P}(x_{t+1} = y|x_t = x)$.

Next come the MDP, which is defined by (X, A, p, r), where:

- X space of states
- A space of actions
- $p(y|x, a)$: probability of transition from a state $x \in X$ to $y \in X$ when the action $a \in A$ is chosen:

$$p(y|x, a) = \mathbb{P}(x_{t+1} = y|x_t = x, a_t = a),$$

- $r(x, a, y)$: reward when passing from x to y using the action a.

We will note here V the value function and π (with $\pi : X \to A$) the policy.

Dynamic programming possess two types of iterations : iterations on the values and iterations on the policies. In each of these algorithms a policy sequence is built, beginning with a given initial policy π_0, and every step k, we evaluate the policy π_k and calculate V^{π_k}, then we improve the policy by calculating π_{k+1} given by V^{π_k} :

$$\pi_{k+1}(x) \in \text{argmax}_{a \in A}[r(x, a) + \gamma \sum_{y} p(y|x, a)V^{\pi_k}(y)]$$

We stop when $V^{\pi_k} = V^{\pi_{k+1}}$. It's important to note that the sequence $(V^{\pi_k})_k$ is a increasing sequence. Therefore, because

there is a finite number of possible policies, the termination criteria is obligatory satisfied with k. So we have $V^{\pi_k} = V^*$ where V^* is the optimal value function, therefor π_k is the optimal policy. Some methods of solving are :

- **Direct solving** of the linear system $(I - \gamma P^\pi)V^\pi = r^\pi$. That's the Gauss elimination method, but it have a complexity of $O(N^3)$
- **Iteration on the values for a permanent policy** : we iterate on the operator T^π (Bellman operator defined by $T^\pi V(x) = r(x, \pi(x)) + \sum_{x'} p(x'|x, a)V(x')$). Considering a given value function V_0, and the recursion $V_{n+1} = T^\pi V_n$. We have then convergence of V_n toward V^π. The problem is that the convergence is asymptotic, but the advantage is that it as a lower complexity than the direct solving method ($O(N^2 \frac{log(1/\epsilon)}{log(1/\gamma)})$ for an approximation of ϵ (interesting if γ is not too close to 1)).
- **Monte-Carlo** : we simulate n trajectories $((x_t^i)_{t \geqslant 0})_{1 \leqslant i \leqslant n}$, starting from x and following the policy π : $x_{t+1}^i \ p(.|x_t^i, \pi(x_t^i))$, so :
$$V^\pi(x) \approx \frac{1}{n} \sum_{i=1}^{n} \sum_{t \geqslant 0} \gamma^t r(x_t^i, \pi(x_t^i)).$$
this method is interesting if we want to evaluate a unique state. It has an approximation error of order $O(1/\sqrt{n})$
- **Temporal differences TD(λ)** : This is a smart method for using the trajectories to evaluate all the states crossed by those trajectories, by evaluating the value of a state x, by the sum of the observed temporal differences $r_t + \gamma V(x_{t+1}) - V(x_t)$ at the future instants t, weighted by a "trace" λ. The algorithm of TD(λ) will be treated later.

During the experiments (in section III, as our work will focus on it), we used some algorithms (Cross Entropy Method (CEM), REINFORCE and TD(λ)). We will focus here on the TD(λ) method, which we found giving the most reliable results during the experiments. We used a random agent, i.e. selecting actions randomly, in order to test the environment. It is important to note that the Random method was used to test the environment, and the debugging of it.

Temporal Differences TD(λ)

Algorithm TD(λ) [8]: After the observation of a trajectory $(x_0, x_1, ..., x_K = 0)$, we update V_n to the states $(x_k)_0 \leqslant k < K$ following :

$$V_{n+1}(x_k) = V_n(x_k) + \eta_n(x_k) \sum_{l=k}^{K-1} \lambda^{l-k} d_l,$$
$$\text{where } d_l = r^\pi(x_l) + V_n(x_{l+1}) - V_n(x_l).$$

with η_n the learning step, typically $\frac{1}{n}$.

The impact of the temporal differences of future transitions on the estimation of the current state value is controled by λ. TD(λ) is a compromise between :

- **TD**(0) (to estimate the fixed point of the operator of Bellman T^π):
$$V_{n+1}(x_k) = V_n(x_k) + \eta_n(x_k) d_k.$$
- **TD(1)** (to estimate the mean):
$$V_{n+1}(x_k) = V_n(x_k) + \eta_n(x_k) \sum_{l \geqslant k} d_l.$$

the choice of λ :

- $\lambda < 1$ allows to reduce the variance of the estimators compared with $\lambda = 1$.
- $\lambda > 0$ allows to propagate faster the rewards compared with $\lambda = 0$.

in the actuated case: We can define the actuated value function when, at each time step, the process is stopped with probability $1 - \gamma$ with $0 < \gamma < 1$:

$$V^\pi(x) = \mathbb{E}[\sum_{t \geqslant 0} \gamma^t r(x_t, \pi(x_t))|x_0 = x].$$

Algorithm TD(λ) becomes:

$$V_{n+1}(x_k) = V_n(x_k) + \eta_n(x_k) \sum_{t \geqslant k} (\gamma\lambda)^{l-k} d_l.$$

Here was a little introduction to the theory of reinforcement learning and a little zoom on the TD(λ) algorithm that we used during the experimentations of our environment.

III. Experiment

We now arrive at the core of this project, which was to develop a learning environment describing the firefighting task in order to improve the policy of the robot and improve its autonomy. For this purpose, we used the toolkit OpenAI Gym, which is a toolkit for developing and comparing reinforcement learning algoriths. We also used the open resource which you can find in the github of torch-twrl (you can follow the link : https://github.com/twitter/torch-twrl). With this resource, you can run already existing environments such as a cartpole environment, an acrobot, board games, etc.

We used those resources to implement and develop our own environment which was the firefighting task.

The mission that we needed to simulate is a mission of search and fight, here representing a firefighter robot. The mission of the robot is to find trees that are on fire, and extinguish them by spreading water on it.

Development of the environment

The first task developed was the most simple one: no moving robot, the only task was to detect if a target was on fire, and then proceed to extinguish it.

This environment was first developed as deterministic. It was an environment where we knew when and where the target will fire up, and see if the robot was able to learn and take measures against. It is important to note that we used the agent/algorithm cited earlier : TD(λ), REINFORCE, Random agent and CEM.

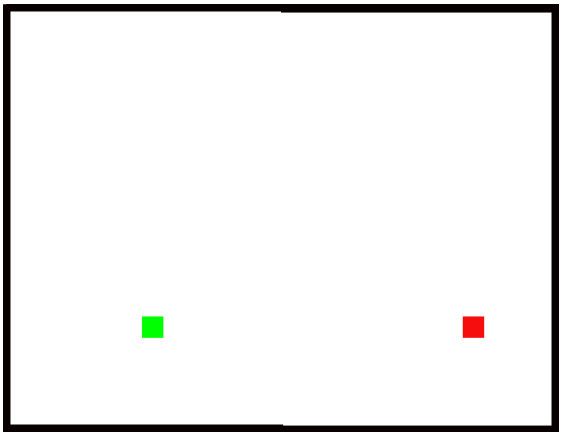

Fig. 2: View of the simulation of the simple determinist environment

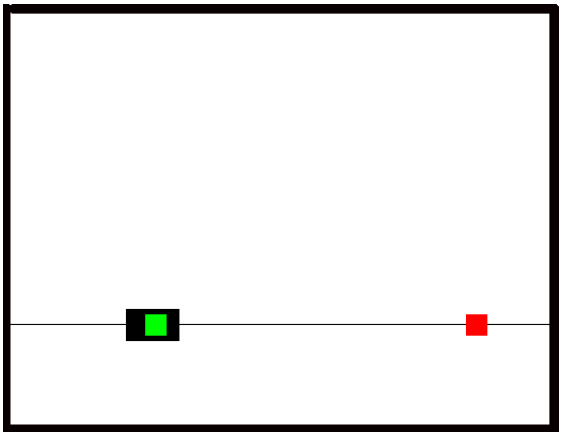

Fig. 3: View of the simulation of the 1D environment

The next step was to turn the deterministic environment into a random one, to see if the agents were able to adapt.

After studying the adaptability of the agents to the simplest environment possible for our task, the goal was to develop an environment where there was not only the action of extinguishing the target, but also moving to it.

The first step was still a deterministic environment, where we knew when and where the targets will light up.
The second step was to turn the deterministic environment to a random one. The final step was to develop the 2D environment. The first 2D-environment developed was a simple one, with only the targets and the robots. It was used to see if the added actions where working accordingly to the real robot.

Then, we developed a more complex environment to make it as close as possible as reality.

To ensure that the environment was close to reality, we discussed the different drawbacks that we needed to implement:

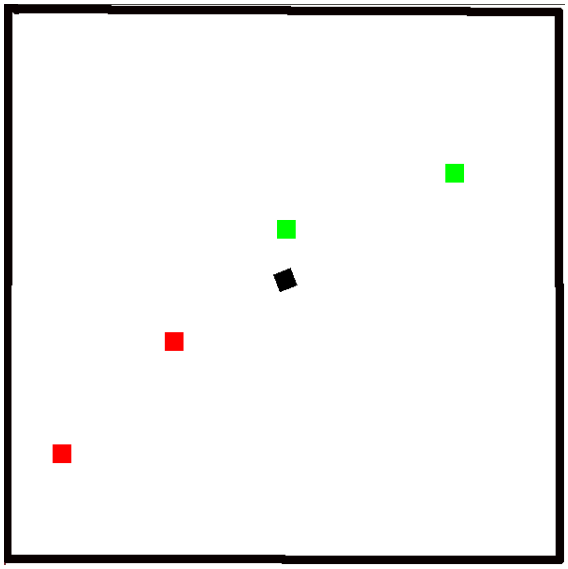

Fig. 4: View of the simulation of the 2D environment

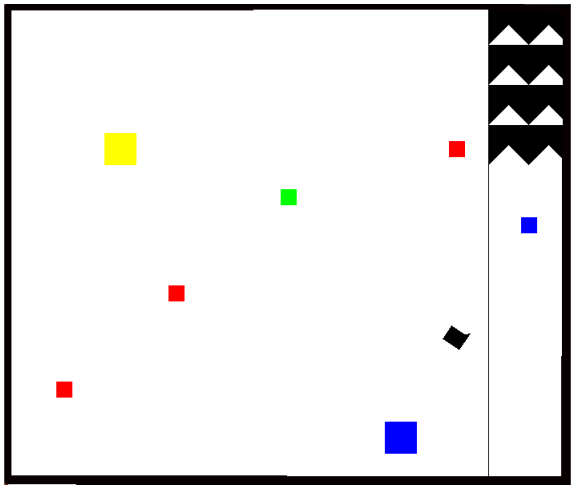

Fig. 5: View of the simulation of the complex 2D environment

- **the battery** : the energy available for the robot is not infinite, for this, a decrementing counter was implemented, which was decrementing at every action carried out by the robot. There was also a battery zone where the robot could refill up its energy. The battery zone is modeled by the yellow square.

- **the robot's water reserve** : as for the battery, the water that the robot can carry is not unlimited. The water was implemented as a counter which was decremented every time the agent used the action to spill water. This water could be refilled in the water zone which was represented as a blue square.

- **the refilling water zone** : represented as a blue square, the water refilling zone could see its water level decreasing (which was viewed by the fading blue

color) due to water leaks (which are part of the operator actions.). Those leaks are represented by the square in the right side of the window. Each square represent a level of leaks, and the higher the level is, the more water leak.

- **the human operator** : The operator was designed as a random agent with a fixed random policy.

- **the temperature** : The temperature needed also to be modeled, as the robot could not stay to long close to a fire. It was modeled as a counter that was increasing when the robot was in a range from the fire, and decreasing when it was out of this range.

- **the fire** : The fires modeled in the environment could be extinguished with only one application of water. We discussed the fact that we needed more than one action to take care of it, but it was not implemented ultimately. But the idea still lies.

You can find the environments and videos of the simulations following this address : https://github.com/TVagneron/Reinforcement-learning

Results

In this part, we will display the results of the experiment on the parameters λ, γ and ϵ which affect the learning update or/and the policy. The following results have been run for a number of 1000 episodes, each episode describing 500 steps. The deterministic environment was tuned to fire one target at a time, every target was lighted up 50 step after the previous one was extinguished.

As we can see with fig.6, for a simple environment, changing λ does not affect the reward outcome.

We can observe the same effects for γ and ϵ, but what will happen if we complicate the environments? You will see the results for a more complex environment with the figures 9, 10 and 11.

As we can see in the figure, the value of λ affects the reward outcome during the learning process. The more the value of the parameter go further from the optimum, the less will be the reward. This was expected as λ affects the actuating of the process. We can discuss to what extent λ affect the learning process, but it is not the subject of this report, it is to discuss the fine tuning of the parameters in order to optimize the learning process, and thus the policy. The tuning of λ can be performed mathematically or experimentally. Here we chose to tune it experimentally. That can be carry out by a machine, which will hasten the process. We can take note of the spikes that appear at some points during the learning process. Those spikes illustrate the dilemma of prospection/exploitation. This dilemma is the property that the process should not stop to explore, but it should explore less and less. We can also take note in the

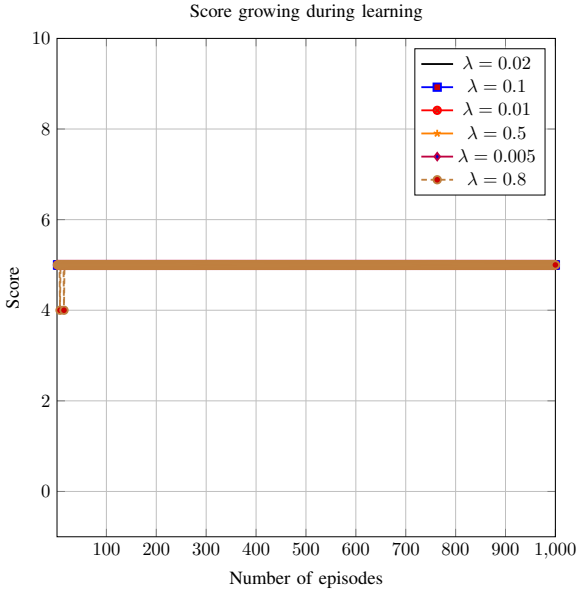

Fig. 6: Score of the simple deterministic environment for λ

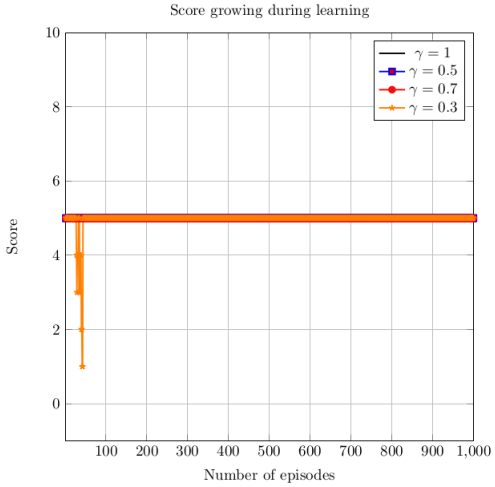

Fig. 7: Score of the simple deterministic environment for γ

difference of reward for a deterministic environment, which is one of the simplest developed here. It brings up the question of fine tuning for the more complex ones, which have more available actions and space.

The addition of γ in the learning process also affects the policy (as it appears in both computer codes). We can see, if γ is different from one, the reward outcome is zero (fig.10) which is different from the results of fig.7. This could be explained by the complexity of the second environment (with have γ^t with a bigger t in fig.10). It was not to be unexpected as γ is a parameter that affect the convergence of the sequence.

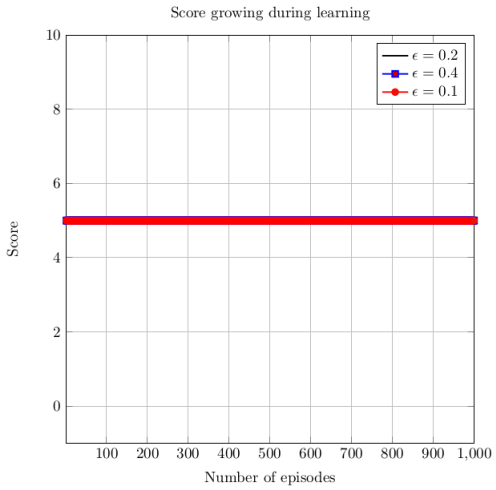

Fig. 8: Score of the simple deterministic environment for ϵ

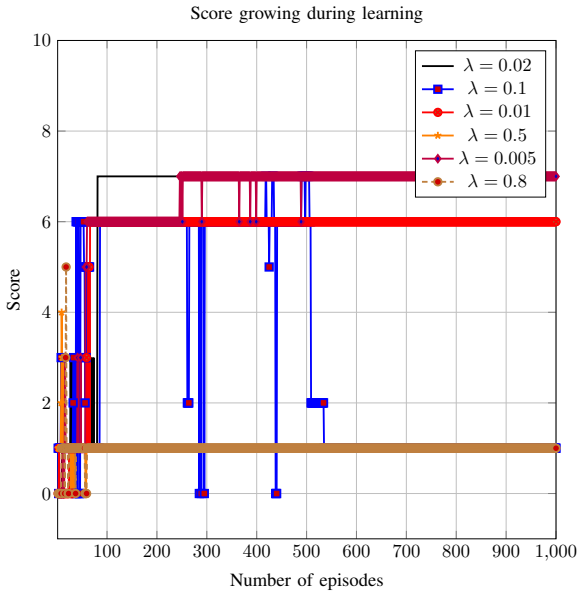

Fig. 9: Score of the deterministic 1D environment for λ

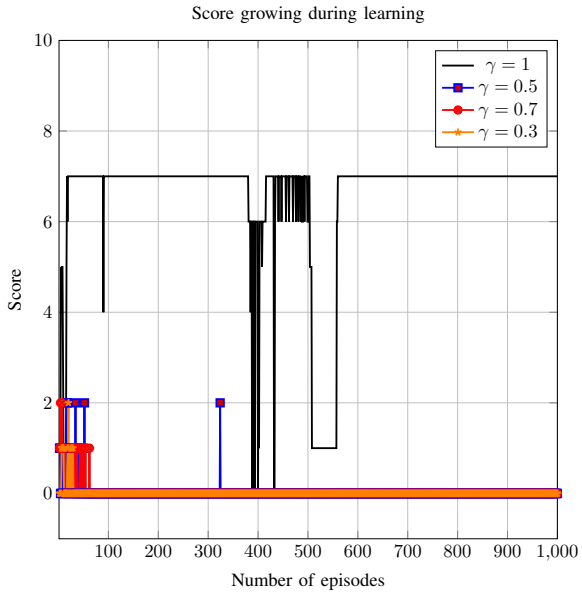

Fig. 10: Score of the deterministic 1D environment for γ

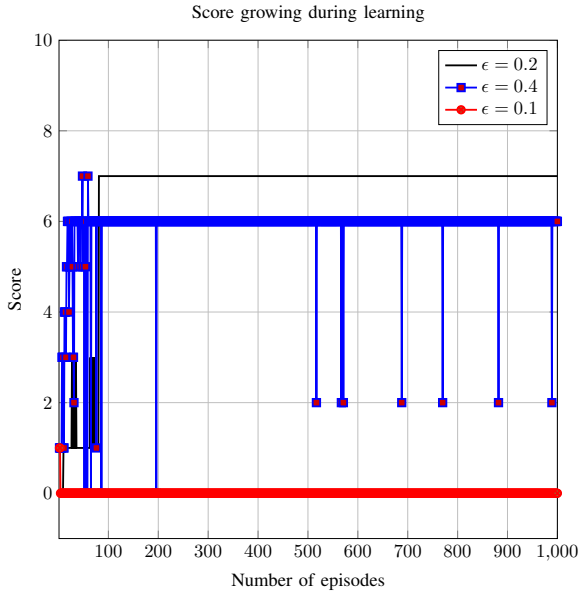

Fig. 11: Score of the deterministic 1D environment for ϵ

The variable ϵ ($1 - \epsilon$ is the probability to exploit, and ϵ the probability to explore) affects the policy: if it is too small, the policy cannot be actualized, resulting in a non-learning behavior, whereas if the value is too high, the policy will be actualized, but to a different policy than the optimum.

As stated previously, those results are only for the deterministic and one dimensional environment. Due to a lack of time, the more complex ones were tested via a random method to verify their functioning. The only tests where we used RL agents were not enough relevant to present them here, but it is important to speak about the optimization of the parameters, and the fact that the development requires to go step by step, as we could let errors slip as we jump over what we could consider as unimportant steps.

We also conducted experimentation on the epsilon decay rate, that affects the processing speed, but the tests were not relevant as they brought up an issue that will be stated in the issue part later in the report.

We opened a repository on github (https://github.com/TVagneron/Reinforcement-learning) where you will find all the developed environments, scripts to compile them and a correction to the qfunction script. You will also find the folder containing videos of the experiments. The first script

will compile the chosen environment for running it with the REINFORCE agent. The second script will compile all the environments and allow you to chose the agent with which you can run them. The qfunction script is a correction of the original qfunction script, where we added print functions to slow down the process (refer to section "Issues" where this issue is developed).

Issues

Some problems occurred, and this section is here to describe some of them and maybe help to resolve them.

One of the first issue that we encountered when developing our environment was the definition of the observation space, where we needed to define the observations that can be seen by the agent, as for this we defined the states of the targets by a strict binary duo, 0 when the target was extinguished, 1 if it was lighted. To follow strictly our mathematical model, we used a Tuple (that,in openaiGym, are a Cartesian product between a discrete space and a continuous space.) to define the target states, but Tuple where not accepted for observation space, so we needed to define them as a continuous space by the use of arrays and boxes.

The second issue was that when we run our environment, there was an error occurring, stating that the state value was nil. It appears that it was related to a process unable to provide the state at the proper time, resulting in a nil value. This issue was solved by slowing down the process (adding prints to track the error origin solved the problem unexpectedly, but it was also slowed down by adding a wait function in the qfunction). This issue has been observed on two PCs using Ubuntu 14.04. (You can find the issue report following : https: //github.com/twitter/torch-twrl/issues/37).

IV. Conclusion

This report proposes a short introduction to the possible issues implied by Mixed-Initiative missions. This kind of missions is prone to grow in the future. We studied in this report the use of Reinforcement Learning to develop a policy for the machine tasks. The simulation environments developed here concern a described firefighter mission involving a human and a robot. As indicated and illustrated in the experiments it is important to fine tune the different parameters as they affect the learning process and the learning speed. More generally, the domain of machine learning is a recent domain, so there is a lot more to discover and improve. Regarding the global Mixed-Initiative missions, an other kind of planning algorithm called MOMDP will be used to infer information about the human operator using the eye-tracker and the electrocardiogram: it will allow to improve the interaction between the machine and the human, and thus the mission achievement.

Acknowledgment

The authors would like to thank Caroline Chanel, because she accepted my participation in this wonderful project.

A huge thank to Nicolas Drougard, for his patience, his support, and all the mishaps we went through.

References

[1] P. M. Fitts. Human engineering for an effective air-navigation and traffic-control system.

[2] L.Dargent. Profilage pilote par apprentissage automatique pour reconfiguration ihm.

[3] R. Munos. Introduction à l'apprentissage par renforcement.

[4] F. Dehais P. E. U. Souza, C. P. Carvalho Chanel. Momdp-based target search mission taking into account the human operator's cognitive state.

[5] Emmanuel Rachelson, François Schnitzler, Louis Wehenkel, and Damien Ernst. Optimal sample selection for batch-mode reinforcement learning. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART 2011)*, 2011.

[6] D. Hsu Wee Sun Lee S. C. W. Ong, Shao Wei Png. The human should be part of the control loop?

[7] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[8] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[9] Mai-Hui Le F. Dehais T. Gateau, C. P. Carvalho Chanel. Considering human's non-deterministic behavior and his availability state when designing a collaborative human-robots system.

[10] S. A. Burden M. K. McCourt J. Williard Curtis W. D. Nothwang, R. M. Robinson. The human should be part of the control loop?

[11] M. M. Wilson. Development of online cardiac feedback for control applications.