Apprentissage évolutif de comportements éthiques

Mémoire de Master

Rémy Chaput

Université Claude Bernard Lyon 1, France
Encadré par Salima Hassas (LIRIS) et Olivier Boissier (LabHC, Mines Saint-Etienne)

Résumé L'accroissement de l'usage des algorithmes d'Intelligence Artificielle (IA) dans des applications impactant des utilisateurs et acteurs humains nécessite de considérer la question de l'adoption de comportements éthiques par ces algorithmes, afin de garantir leur acceptabilité. Plusieurs approches existent déjà, mais elles ne permettent pas l'adaptation face à des situations nouvelles. Pour répondre à ce problème, nous proposons une approche nouvelle utilisant l'apprentissage par renforcement. Nous développons également un simulateur, de conception générique, que nous instancions dans le cas du problème de la gestion intelligente de l'énergie dans les grilles électriques intelligentes (Smart Grid). Nous utilisons cette application pour évaluer notre approche en étudiant l'impact des différentes récompenses sur le comportement éthique des agents.

Mots-clés: Intelligence Artificielle, Éthique, Systèmes Multi-Agents, Apprentissage par Renforcement, Cartes Auto-Organisatrices, Réseaux de distribution intelligents.

Abstract. The increase in the use of Artificial Intelligence (AI) algorithms in applications impacting human users and actors has, as a direct consequence, the need for endowing these AI systems by ethical behaviors. Several approaches already exist, but they fail to allow adaptability when facing new situations. To tackle this problem, we propose a new approach using Reinforcement Learning. We also propose a generic multi-agents simulator, that we adapt to the case of intelligent management of energy distribution in Smart Grids, and that we use in order to evaluate our implementation of this proposed approach. We then present the results using multiple rewards that reflects ethical behaviors.

Keywords: Artificial Intelligence, Ethics, Multi-Agent Systems, Reinforcement Learning, Self-Organizing Maps, Smart Grids.

1 Introduction

Ce travail s'inscrit dans le cadre du projet *Ethics.AI*[1].

Les récents progrès de l'Intelligence Artificielle (IA) ont mené à une rapide croissance de l'utilisation d'algorithmes ayant un impact potentiel sur les humains ; citons par exemple le *trading* automatique, la conduite assistée (voire autonome) de véhicules, l'allocation de ressources. Nous nous intéresserons à ce dernier domaine, et plus spécifiquement la gestion de l'énergie dans les réseaux de distribution intelligents. Nous détaillerons cet exemple en tant que cas d'étude pour évaluer notre approche dans la section 2. Mettant de côté l'usage de ces algorithmes (opéré par les humains, qui peuvent donc introduire une "mauvaise" utilisation), comment peut-on intégrer un aspect éthique intrinsèque à ces algorithmes ? En d'autres termes, est-il possible de concevoir des systèmes intelligents dont le comportement serait acceptable selon des valeurs humaines (e.g. un sous-ensemble des valeurs définies par Schwartz [23], voire l'annexe A) ?

Ces questions sont partagées par une partie de la communauté (scientifique ou non) ; par exemple, l'organisme de standardisation IEEE[2] a ouvert un groupe de travail dédié à l'éthique dans les systèmes autonomes et intelligents (A/IS)[3]. D'autres groupes sont également consacrés à ces questions, tel que le *High Level Expert Group*[4] de l'Union Européenne ou le rapport de la CNIL[5] (qui offre le point de vue d'un débat public, contrairement aux deux groupes cités précédemment).

Notons que nous ne nous intéressons pas, dans le cadre de ce travail, à la moralité de la conception et l'utilisation de robots (un champ de recherche nommé *Roboethics*), e.g. "Est-il moral d'asservir un robot conçu pour imiter une intelligence humaine ?", mais plutôt à la capacité des agents artificiels à exhiber des comportements moraux (un autre champ de recherche, nommé *Machine Ethics* [3]).

Il est à noter que tous les sous-domaines de l'IA sont potentiellement visés par cette question éthique : algorithmes d'apprentissage (tel que le *Deep Learning*), Robotique Développementale, Agents Autonomes, etc. Les domaines d'application sont également nombreux, on peut citer par exemple l'Intelligence Ambiante, et par extension les *Smart Environments*.

Plusieurs travaux (e.g. [2,7,22]) existent déjà sur l'éthique dans les systèmes d'IA ; toutefois, un inconvénient commun à ces travaux est le manque d'adaptabilité (ou d'évolutivité) de leurs modèles. Bien que les approches soient généralement génériques et permettent de représenter la plupart des cas d'études, le modèle en lui-même ne permet pas à un agent de s'adapter face à une situation nouvelle

1. *Artificial constructivist agents that learn ETHICS in humAn-Involved co-construction*
2. *Institute of Electrical and Electronics Engineers* `https://www.ieee.org/`
3. *Working Group* P7000™ et supérieurs, `https://ethicsinaction.ieee.org/`
4. `https://ec.europa.eu/digital-single-market/en/high-level-expert-group-artificial-intelligence`
5. Commission Nationale de l'Informatique et des Libertés ; `https://www.cnil.fr/sites/default/files/atoms/files/cnil_rapport_garder_la_main_web.pdf`

(comme nous le verrons plus en détails dans la section 3.1).

À partir des observations ci-dessus, nous formulons la problématique suivante : "Comment concevoir un modèle d'agents artificiels intelligents, capables d'apprendre des comportements que l'on considère comme acceptables (par rapport à des valeurs humaines telle que l'équité, ou le respect de l'autorité) et capables de s'adapter au fil de l'exécution, à partir de mesures objectives et calculables de leur environnement ?"

Cette problématique soulève les questions de recherche suivantes :

QR-1. Comment permettre aux agents artificiels de construire leur propre représentation de l'environnement à partir de mesures objectives et calculables ?

QR-2. Comment permettre à un agent d'apprendre un comportement compatible avec des valeurs à partir de l'état de l'environnement ?

QR-3. Comment définir la récompense de telle façon que chaque agent puisse apprendre selon sa propre contribution une ou plusieurs valeurs éthiques ?

Nous évaluons nos propositions et leur mise en oeuvre en définissant plusieurs fonctions de récompense, dont certaines ciblent particulièrement le problème de l'adaptation (voir la section 5.2) ; nous définissons également plusieurs scénarios de simulation (e.g. avec / sans apprentissage) qui seront comparés entre eux pour chaque récompense (voir la section 5.4).

2 Cas d'étude

Le cas d'étude retenu pour expérimenter a été proposé par l'entreprise Ubiant, partenaire du projet Ethics.AI ; il s'agit du problème de la répartition de l'énergie dans les réseaux électriques intelligents (*Smart Grids*, voir l'annexe B). Dans de tels réseaux, la production d'énergie est décentralisée en de multiples grilles (que l'on peut associer à un quartier pour se faire une représentation). Chaque grille dispose d'une "mini-centrale" dédiée ; de plus, les consommateurs d'énergie deviennent des *prosumers* (producteurs-consommateurs). Par exemple, les bâtiments s'équipent de panneaux photo-voltaïques et autres équipements relatifs aux énergies renouvelables, afin d'augmenter leur autonomie (i.e. diminuer leur consommation énergétique externe) voire éventuellement de devenir des bâtiments à énergie positive (c'est-à-dire que sur une longue période, la production du bâtiment a excédé sa consommation externe). Considérant que la demande et la production peuvent fluctuer sur de courtes périodes, et qu'il est difficile de stocker une grande quantité d'énergie, de tels bâtiments s'organisent en grilles afin d'échanger l'énergie en trop. L'échange à l'échelle locale plutôt que nationale permet de limiter les pertes en lignes ; mais cela nécessite de la coopération entre les différents *prosumers*. De manière similaire, lorsqu'il y a une trop forte sollicitation de la grille, les utilisateurs doivent faire des efforts et restreindre leur consommation (au moins de manière temporaire). Cela pousse les utilisateurs à réduire leur confort, pour le bien de l'ensemble des individus afin d'éviter

des situations de *blackout* ou de coupures de courant répétées. Divers mécanismes destinés à optimiser ces flux d'énergie et à contrôler les variations (tels que l'effacement) existent et sont actuellement étudiés [17].

Dans le cadre de ce stage, nous nous intéressons à l'aspect coopératif des agents *prosumers*, qui doivent maximiser leur confort tout en respectant l'intérêt de la grille dans son ensemble. Cette opposition de valeurs pose un cadre intéressant pour l'étude de l'éthique (on pourra y retrouver e.g. des notions d'équité, respect des autres, préservation de la nature). Pour cela, on considère un ensemble de plusieurs bâtiments, de deux types : habitation ou hôpital. Cet ensemble hétérogène se partage une quantité d'énergie disponible à l'intérieur d'une grille ; pour cela, ils prennent (à intervalle régulier) des décisions sur leur consommation énergétique. Ces choix ont une influence sur l'état de la grille et par conséquent l'ensemble des utilisateurs. On détermine un certain nombre de mesures objectives sur cette grille à partir de son état, et on en déduit le comportement éthique (ou non) des utilisateurs.

Par exemple, si tous les utilisateurs consomment le maximum d'énergie possible (pour assurer leur confort), cela force une importante dépendance envers le réseau national, ce qui est contraire aux principes des *Smart Grids*. On pourra donc considérer que le comportement n'est pas acceptable par rapport à cet objectif. Autre exemple, si la plupart des utilisateurs restreignent leur consommation (pour ne pas forcer la grille à utiliser le réseau national), sauf un, qui consomme beaucoup plus, cette situation n'est pas très équitable. On peut éventuellement envisager que cet utilisateur en ait le droit, s'il est prioritaire (e.g. si c'est un hôpital).

Il semble, à partir de ces quelques exemples, qu'il soit difficile d'expliciter clairement l'état que nous recherchons (ou les choix que les utilisateurs doivent faire) ; on préférera donc définir des fonctions qui évaluent la valeur de satisfaction d'un état. En d'autres termes, nous préférons attribuer une quantification à un état du monde (que nous pouvons appeler récompense) plutôt qu'attribuer une étiquette "correct" ou "incorrect" à chaque action.

3 État de l'art

Les questions définies précédemment sont liées à divers champs de recherche, dont nous allons faire un rapide tour d'horizon, afin d'une part d'expliquer le problème sous-jacent et d'autre part de présenter les solutions proposées dans la littérature et les motivations des choix que nous avons fait pour proposer notre contribution, en lien avec les questions de recherche (QR-1, QR-2 et QR-3) définies précédemment. En particulier, nous présenterons les approches existantes et le manque d'adaptabilité qui nous a poussé à proposer une nouvelle approche (dans la section 3.1) ; l'apprentissage constructiviste comme inspiration pour la façon d'apprendre (en accord avec la QR-1, dans la section 3.2) ; l'apprentissage par renforcement (par rapport à la QR-2, dans la section 3.3) ; la quantification vectorielle, afin que l'agent puisse se représenter des états et actions dans un espace continu à haute dimension (ce qui correspond à une par-

tie de la QR-1, dans la section 3.4) ; et enfin, l'affectation de la récompense (la QR-3, dans la section 3.5).

3.1 Éthique et IA (*Machine Ethics*)

Cointe [6] classe les différentes approches visant à traiter de l'éthique dans les programmes d'IA en trois catégories : éthique par conception, par casuistique et par modélisation logique.

Éthique par conception Cette approche vise à implémenter "en dur" le comportement attendu du programme. C'est-à-dire que les concepteurs du programme doivent modéliser les situations qui nécessitent une réflexion éthique, déterminer le comportement à avoir, et implémenter des règles de décision en situation. Par exemple, nous pouvons citer le cas des drones militaires [9], qui sont conçus pour tirer (ou non), selon la reconnaissance des personnes qui seraient touchées par le tir (combien de militaires ennemis, militaires alliés, et civils). Le comportement est donc programmé sous forme d'un seuil de décision, l'éthique n'est pas explicitement représentée.

Le manque d'adaptabilité de cette approche découle de son principe même : comme le concepteur doit choisir un seuil de décision pour chaque situation pouvant survenir, si une situation n'est pas prise en compte, le comportement de l'agent ne pourra pas être optimal dans cette situation.

Éthique par casuistique Cette approche vise à réunir des jugements d'experts, déterminer le "bon" comportement pour différents cas, puis faire fonctionner un algorithme d'apprentissage pour déterminer les règles de décision. Cet algorithme d'apprentissage peut avoir un degré variable de transparence (on peut imaginer du *Deep Learning* ou des clauses de Horn). Ce type d'approche requiert une masse importante des données [13] (les *features* qui représentent une situation), qui doit être annotée par les experts (le *label*, qui indique si l'action proposée est correcte ou non).

On peut citer dans cette approche les travaux d'Anderson sur la conduite autonome [2], et notamment le programme *GenEth* (*General Ethical Dilemma Analyzer*).

Éthique par modélisation logique Cette approche consiste à modéliser et programmer des principes éthiques, qui servent à déterminer la "bonne" action, comme des règles logiques. Les principes éthiques sont un ensemble de règles, conçues par des philosophes, qui permettent de résoudre des situations de "dilemme éthique", c'est-à-dire des situations où aucun choix ne semble optimal. L'un des exemples les plus connus est celui du Dilemme du Tramway [10]. Un des principes éthiques utilisable pour choisir une action est la doctrine du Double Effet (proposée par Thomas d'Aquin).

Ces règles sont formalisées dans un programme afin qu'il puisse raisonner dessus à partir d'une situation donnée. Certaines approches (e.g. [5] n'utilisent

qu'un seul principe éthique à la fois (se pose alors la question de la pertinence de ce principe éthique ? Dans quelles situations est-il adapté ? Dans quelles situations ne l'est-il pas ?), tandis que d'autres considèrent un ensemble de principes éthiques, augmenté d'une relation de priorité. C'est notamment le cas du projet ETHICAA [1] (voir e.g. les travaux de Cointe sur le jugement éthique dans le cas des marchés boursiers [7]).

Cette approche générique permet de formaliser n'importe quel principe éthique, et de l'appliquer à n'importe quel problème. Elle ne peut cependant pas adapter au fil du temps les connaissances morales et éthiques à la disposition de l'agent. Si une situation nouvelle apparaît, les connaissances correspondantes doivent être intégrées à l'agent.

Cet état de l'art des *Machine Ethics* nous permet de confirmer le manque d'adaptabilité que nous avions introduit précédemment.

3.2 Courant de pensée constructiviste

L'apprentissage constructiviste [12] est inspiré de la théorie du développement cognitif chez l'enfant, développé par Piaget [19]. Nous partons du principe (entre autres) que l'intelligence se manifeste à travers la capacité d'apprentissage d'un système, et que l'apprentissage doit être actif. L'agent (humain ou artificiel) auto-organise ses représentations du monde à travers son expérience d'interactions avec l'environnement. Cet apprentissage se fait par le biais de schémas sensorimoteur, qui représentent un lien entre une action (les moteurs) et une perception (les senseurs). Ce paradigme (du point de vue informatique) trouve ses racines dans le livre de Drescher [8], et est de plus en plus utilisé, notamment dans des domaines tels que la robotique développementale [33], ou l'Intelligence Ambiante. L'une des différences principales avec les autres approches d'apprentissage est que la représentation du monde (la façon dont sont structurées les connaissances que l'agent possède) doit être construite par interaction avec environnement. Ainsi, chaque agent aura sa propre façon de représenter le monde (il s'agit du concept d'*Umwelt*). Bien que l'approche que nous proposons ne soit pas complètement constructiviste, nous nous inspirons de ce courant de pensée, notamment pour l'apprentissage de la représentation.

3.3 Apprentissage par renforcement

L'apprentissage par renforcement (*Reinforcement Learning* - RL) [28] est une forme d'apprentissage qui diffère des autres (e.g. apprentissage supervisé, non supervisé, semi-supervisé) par le fait que l'agent apprend par expériences dans son environnement, en recevant en retour une certaine quantité, appelée la récompense. Cette récompense n'indique pas quelle était l'action correcte (contrairement à l'apprentissage supervisé), mais à quel point l'action effectuée était correcte.

Ce paradigme a été proposé par Sutton [27,28], mais ses origines proviennent des processus décisionnels Markoviens (*Markovian Decision Process* - MDP)

et de l'équation de Bellman [4]. Du point de vue biologique, cela s'inspire de la dopamine et son rôle dans le système motivationnel (d'où l'appellation de "renforcement").

Formellement, on considère un ensemble d'états possibles E et un ensemble d'actions possibles A. Le but du RL est de déterminer une politique de décision $\pi : E \rightarrow A$ (i.e. $\pi(e_i) = a_j$ signifie que a_j est la meilleure action à prendre dans l'état e_i pour optimiser la récompense). Afin de déterminer cette politique, on applique l'algorithme (abstrait) suivant :

```
   // Initialisation
1  On initialise V (e), la valeur attribuée à chaque état e;
2  On initialise π la politique;
3  On initialise e_t l'état de l'environnement;
4  foreach épisode t do
       // Action recommandée
5      a_t ← π (e_t);
       // On observe la récompense
6      r_{t+1} ← Effectuer (a_t);
       // On observe le nouvel état
7      e_{t+1} ← Percevoir ();
       // Mise à jour des valeurs (équation de Bellman)
8      V(e_t) ← V(e_t) + α [r_{t+1} + γV(e_{t+1}) − V(e_t)];
9  end
```

Algorithme 1 : Apprentissage par renforcement (algorithme abstrait).

Dans cet algorithme, α est appelé le *taux d'apprentissage*, et contrôle la "vitesse" de changement des valeurs de la fonction V. γ est appelé le *facteur d'actualisation* (*discount factor*) et permet de contrôler l'horizon des récompenses.

Plusieurs algorithmes spécifiques ont été proposés : *Temporal Difference Learning* (TD-Learning) [27], Q-Learning [29] parmi les premiers, puis des dérivés comme l'algorithme Actor-Critic [15] ou SARSA [21].

L'algorithme *Actor-Critic* [15] consiste en 2 pseudo-agents, un *Actor* qui apprend la politique de décision, et un *Critic* qui apprend à estimer la fonction de valeur V (et qui fournit la récompense à l'*Actor*). Les deux pseudo-agents apprennent en même temps, ainsi la "critique" que fait l'agent *Critic* s'adapte à la politique de l'*Actor* afin de mieux l'orienter. L'inconvénient est la duplication des agents à apprendre (à la fois un algorithme de type *policy-based*, l'*Actor*, et un algorithme de type *value-based*, le *Critic*), ce qui double le nombre de poids à apprendre.

L'algorithme Q-Learning [29] consiste à représenter l'intérêt (la fonction V dans l'algorithme 1) à l'aide d'une table des Q-Valeurs (ou Q-Table), dont les lignes représentent les états et les colonnes représentent les actions. Chaque case contient donc l'intérêt d'effectuer une action dans un état (la récompense attendue). Cette table permet également de calculer la politique de décision π (on peut par exemple prendre l'action d'intérêt maximal ou calculer une probabilité selon l'intérêt). SARSA [21] est un dérivé du Q-Learning dans lequel l'apprentissage se

fait directement sur la politique de décision (*on-policy* : la mise à jour des valeurs se fait en choisissant une action), tandis que le Q-Learning apprend directement la politique optimale (*off-policy* : on met à jour en considérant l'action d'intérêt maximal). On considère généralement que SARSA adopte un apprentissage plus prudent (évite les situations risquées), mais il est plus difficile d'apprendre la politique optimale [6].

Nous avons choisi d'utiliser l'algorithme Q-Learning dans notre modèle, car il permet de représenter facilement l'intérêt, et permet d'apprendre la politique optimale, contrairement à SARSA, et ce en utilisant moins de ressources que l'approche *Actor-Critic*.

Notons que les algorithmes de RL (et en particulier le Q-Learning) nécessitent traditionnellement un ensemble fini d'états et d'actions (afin de simplifier la définition de la fonction V). Il existe un sous-ensemble de ce champ de recherche qui est, au contraire, spécifique à l'utilisation d'espace continus (nommé *Continuous Reinforcement Learning*) ; il est également possible d'appliquer une quantification vectorielle sur les espaces d'états et d'actions afin de représenter chaque vecteur par un scalaire (ce qui revient à avoir un ensemble fini d'états et d'actions).

3.4 Apprentissage de la représentation

Nous voulons utiliser un espace continu de perceptions et d'actions, sous forme de vecteur. Nous voulons également que les agents apprennent leur représentation (i.e. ne pas leur fournir directement de *mapping* entre un vecteur et un identifiant discret) ; cela s'apparente au problème de discrétisation, ou quantification vectorielle.

La quantification vectorielle consiste à assigner un scalaire à tout vecteur de l'espace. On retrouve dans ce champ de recherche des techniques comme le *clustering*, qui consiste à découper l'espace en de multiples groupes (ou *clusters*). Chaque vecteur sera alors représenté par le *cluster* dont il est le plus proche (on associe traditionnellement un identifiant numérique aux *clusters*). Il existe de nombreux algorithmes de *clustering*, tel que K-Means, ou DBSCAN (voir [31] pour une liste plus complète). L'inconvénient est qu'il faut généralement avoir une intuition sur la topologie des données que l'on souhaite apprendre (par exemple si la distribution est circulaire, Gaussienne, ou autre ; cela aide pour choisir l'algorithme), ce que nous n'avons pas dans notre cas.

Les Cartes Auto-Organisatrices (*Self-Organizing Maps* - SOM) de Kohonen [14] sont également des techniques utilisables pour la quantification vectorielle, ainsi que les algorithmes dérivés tels que les Gaz Neuronaux Croissants (*Growing Neural Gas* - GNG) [11], ou encore les Cartes Auto-Organisatrices Dynamiques (*Dynamic-SOM*) [20]. Ces algorithmes utilisent des réseaux de neurones (avec un faible nombre de neurones, en général moins d'une centaine, selon la taille des données à apprendre), dont chaque neurone est associé à un vecteur, que l'on appelle aussi un prototype. Chacun de ces vecteurs prototypes représente

6. Voir par exemple `https://github.com/cvhu/CliffWalking`

une partie de l'espace que l'on souhaite apprendre ; lorsqu'on reçoit un vecteur issu du jeu de données (qui représente donc un point intéressant de l'espace), on déplace les vecteurs de la carte vers ce vecteur. Cela permet un apprentissage de la topologie du jeu de données. La figure 1 montre le déplacement de ces vecteurs lors de la phase d'apprentissage. Les GNG sont similaires aux SOM, mais ont la capacité de créer de nouveaux neurones (pour mieux représenter l'espace des données) ; il s'agit d'une amélioration intéressante mais qui demande l'ajout d'un hyperparamètre pour contrôler cet ajout de neurones. Les Dynamic-SOM ajoutent un paramètre d'élasticité afin de permettre des organisations de neurones plus ou moins resserrées.

L'avantage de ces techniques est l'apprentissage de la topologie des données qui sont fournies. Nous choisissons d'utiliser les SOM de par leur simplicité d'utilisation (et de paramétrage).

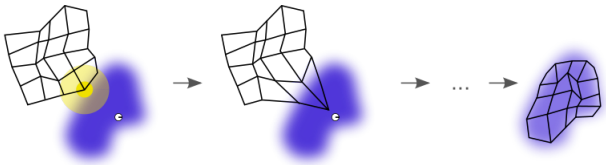

Fig. 1: Etapes d'apprentissage dans une Carte Auto-Organisatrice. Le neurone en jaune est le neurone dont le vecteur est le plus proche de la donnée en entrée, représentée par le disque blanc. Ce neurone est aussi appelé "neurone gagnant" (ou *Best Matching Unit* - BMU). Le BMU et ses voisins sont déplacés vers la donnée en entrée, de manière itérative sur l'ensemble des données, jusqu'à ce que les neurones de la carte représentent l'espace des données. Image extraite de Wikipédia.

3.5 Récompense

La récompense est l'élément principal qui permet aux agents d'adapter leur comportement ; en supposant que l'algorithme d'apprentissage choisi fonctionne correctement, c'est la récompense qui indique le comportement voulu par le concepteur. Il est ainsi primordial que la récompense reçue par un agent corresponde à son propre progrès, sans quoi l'agent apprendrait un comportement non cohérent. Nous nous intéressons à deux problèmes : premièrement, l'affectation d'une récompense à un agent parmi tous les agents (que l'on appelle le *Multi-Agent Credit Assignment Problem*), et deuxièmement l'utilisation de plusieurs objectifs dans une récompense (que l'on appelle le *Multi-Objective Reward*).

Multi-Agent Credit Assignment Problem Supposons que l'on souhaite apprendre aux agents la valeur d'équité, et que tous les agents aient un comportement équitable, sauf un ; si la récompense ne correspond pas exactement à la

contribution des agents, cet agent "déviant" pourrait être récompensé (car globalement, le comportement de tous les agents est équitable). Inversement, les autres agents auraient pu obtenir une meilleure récompense car leur comportement est équitable, mais globalement ils sont légèrement pénalisés par l'agent "déviant". Cette situation illustre un problème connu en RL : l'assignation de la récompense (*Credit Assignment Problem* - CAP), et plus particulièrement le *Multi-Agent Credit Assignment Problem*.

Panait et Luke [18] présentent de multiples façons de calculer des récompenses dans un environnement multi-agents, selon le problème à résoudre. Afin de comparer ces méthodes, définissons $V : Action^n \rightarrow [0, 1]$ la fonction qui attribue une valence à un ensemble d'actions. Nous utilisons V pour comparer les trois méthodes suivantes :

Récompense Globale

$$G = V(\{Action(i)|i \in Agents\})$$

Cette récompense considère l'ensemble des agents dans l'environnement. Il s'agit de la méthode la plus simple à calculer, mais elle échoue quand le comportement de l'agent diffère du comportement majoritaire : il est alors récompensé ou puni, à tort (comme nous l'avons vu dans notre problème d'équité en introduction de cette section).

Récompense Locale

$$L_m = V(\{Action(m)\})$$

La récompense de l'agent m ne dépend ici que de son action. D'une part, cela rend certaines valeurs éthiques difficiles à récompenser (comme l'équité), mais surtout, cela mène à des comportements égoïstes [18] (l'agent ne cherchant qu'à maximiser sa propre récompense, peu importe le reste de la société).

Récompenses Différenciées

$$DR_m = \underbrace{V(\{Action(i)|i \in Agents\})}_{=G} - \underbrace{V(\{Action(i)|i \in Agents \backslash \{m\}\})}_{\text{Monde hypothétique sans l'agent } m}$$

Pour déterminer la récompense de l'agent m, on compare la valence de l'environnement (selon la méthode "globale") et on soustrait la valence d'un environnement hypothétique où l'agent m n'aurait pas agi. L'idée est de calculer à quel point l'action de l'agent m change cette valence. Si l'environnement hypothétique a une valence plus élevée que l'environnement réel, c'est que l'agent m a effectué une mauvaise action (par rapport à la fonction de valence V), et donc il obtiendra une récompense $DR_m < 0$. À l'inverse, si l'environnement hypothétique a une valence plus faible que l'environnement réel, c'est que l'agent m a effectué une bonne action, et obtiendra une récompense $DR_m > 0$. On peut remarquer que la définition de fonction de récompense se retrouve déplacée vers la fonction de valence V : c'est en effet cette fonction qui va définir quels sont les comportements attendus (équité, etc.). Nous choisissons ainsi cette approche (les *Difference Rewards*) pour calculer la contribution des agents.

Multi-Objective Reward D'autre part, nous aimerions pouvoir faire apprendre plusieurs valeurs éthiques. Il s'agit du problème des récompenses multi-objectifs (*Multi-Objective Reward*), qui correspond à calculer le front de Pareto, i.e. l'ensemble des actions dont les récompenses sont incomparables, mais dont on sait qu'elles ne sont pas dominées (au sens de Pareto) par une autre action.

Yliniemi et Tumer [32] proposent les *Multi-Objective Difference Rewards*, qui permettent d'associer les récompenses multi-objectifs aux récompenses différenciées, ce qui correspond à ce dont nous avions besoin. Formellement, cela revient à considérer la récompense comme un vecteur dont chaque composante correspond à un sous-objectif $o \in O$ parmi un ensemble d'objectifs. La majorité des algorithmes de RL utilisent une récompense scalaire, on ne peut donc pas utiliser ce vecteur tel quel ; ainsi, l'idée est de transformer ce vecteur en un scalaire, que nous utiliserons comme récompense finale. Notons ce vecteur f, les deux méthodes suivantes sont proposées pour ce faire :

1. $R_+ = \sum_{o \in O} p_o f_o$ où p_o est le poids associé à l'objectif o
2. $R_\lambda = \pi_{o \in O} f_o$

On remarque que la première méthode (R_+) permet un contrôle plus fin, puisque nous utilisons des poids pour chaque sous-fonction objectif. Cela permet de prioriser un objectif par rapport aux autres. Dans la deuxième méthode (R_λ), les sous-objectifs doivent être normalisés entre 0 et 1 (sans quoi on pourrait multiplier deux termes < 0 et obtenir ainsi une récompense > 0). Dans les deux méthodes, les sous-objectifs peuvent être des *Difference Rewards*, ce qui nous permet de cumuler les deux approches.

Notre analyse de l'état de l'art en apprentissage nous a permis d'identifier plusieurs techniques nécessaires à l'apprentissage, tel que le *Q-Learning*, les Cartes Auto-Organisatrices de Kohonen pour la représentation, et enfin les *Multi-Objective Multi-Agent Difference Rewards*.

4 Contributions

Nos contributions se découpent en 2 axes majeurs (liés entre eux, mais que nous préférons séparer pour une meilleure lisibilité) :

— Proposition d'un simulateur suffisamment générique dans sa conception pour permettre de l'appliquer au plus de problèmes possible, mais que nous spécialisons au cas d'étude choisi. Ce simulateur est détaillé dans la section 4.1, et son application au problème des *Smart Grids* dans la section 4.2.

— Proposition d'un processus de décision et d'apprentissage, intégré aux agents du simulateur défini précédemment. Le modèle proposé est détaillé dans la section 4.3.

4.1 Simulateur

Afin de permettre de mettre en place n'importe quel type d'apprentissage, nous proposons un simulateur, multi-agent, avec des pas de temps discrets (*step*

ou étape). Le choix a été fait d'implémenter un nouveau simulateur, plutôt que d'utiliser un système existant (tel que REPAST [16] ou encore NetLogo [30] pour des simulateurs génériques, ou OpenDISCO [26] pour un simulateur spécifique aux *Smart Grids*), afin de créer une approche générique et simplifiée (pas de communication entre les agents par exemple), tout en gardant un contrôle complet sur le simulateur au besoin. Python (version 3) a été choisi comme langage de programmation car il offre une large base de paquets pour l'apprentissage et le calcul mathématique en général (on peut citer par exemple *NumPy*, *Pandas*, *Sompy*, ...). Le simulateur est composé des éléments suivants :

Environnement Structure abstraite avec laquelle les agents interagissent. Cette structure contrôle le temps du simulateur, contient la liste des agents, calcule des propriétés, fournit les perceptions aux agents et dispose d'un historique des actions passées (ainsi que leur résultat). L'environnement contient également les variables partagées (s'il y en a). On considère donc que l'environnement est ici omniscient, mais les agents n'ont pas accès à toutes les informations.

Un composant spécifique, nommé l'instance de régulation, contrôle les actions qui sont effectuées. En particulier, cette instance peut empêcher certaines actions, qui ne devraient pas survenir dans l'environnement. Ces actions peuvent être empêchées soit à cause de raisons que le concepteur impose comme limites infranchissables (exemple : on ne veut pas que l'agent tue un autre agent, ou dans le cas des *Smart Grids*, on ne veut pas que l'agent provoque un *black-out*), ou pour des raisons physiques (e.g. si une action serait impossible à effectuer dans le monde réel). Pour ce faire, l'instance de régulation intervient entre l'agent et l'environnement : elle reçoit les actions des agents et effectue des actions intermédiaires visant à corriger la situation. L'agent apprendra donc un comportement responsable au sein de cet espace défini par des frontières éthiques et physiques.

Agent Une entité nommée du simulateur, qui est présente dans l'environnement et qui interagit avec celui-ci par le biais d'actions. Un agent reçoit un certain nombre de perceptions de la part de l'environnement, détermine à partir de celles-ci une hypothèse d'état (c'est-à-dire l'état dans lequel l'agent pense être), et enfin décide de l'action à effectuer. Un agent peut également avoir un ensemble de variables personnelles. Les agents peuvent être spécialisés (on les appelle alors des "profils d'agent") ; un profil d'agent diffère d'un autre notamment sur les actions qu'il peut effectuer.

Action Une action permet à un agent d'agir sur son environnement, et dans la plupart des cas de le modifier ; afin de permettre des actions continues, on ajoute des paramètres aux actions. L'agent décide de l'action à effectuer, ainsi que des paramètres éventuels. Ces paramètres sont génériques dans ce modèle, ce peut être un nombre entier ou à virgule, des chaînes de caractère, ou même une référence à un autre agent. Les actions implémentées dépendent du problème étudié.

Propriétés Les propriétés sont un ensemble de mesures objectives de l'environnement. Elles sont notamment utilisées pour le caractériser aux yeux d'un opérateur humain (par exemple sous forme de graphique montrant leur évolution selon le temps). Elles sont également fournies aux agents, comme partie intégrante de l'ensemble des perceptions. Comme les actions, les propriétés dépendent du cas d'étude.

Perception Les perceptions sont un ensemble de mesures de l'environnement et/ou de l'agent, aussi appelées "capteurs". Ce sont les informations que chaque agent obtient au début de chaque étape, et qui lui permettent de se représenter l'état actuel de l'environnement (l'état réel n'est pas visible par l'agent, les perceptions apportent un certain point de vue). Les agents ont tous les mêmes capteurs, en revanche ils n'ont pas forcément les mêmes valeurs pour ces capteurs. De même que pour les actions et les propriétés, les perceptions sont à choisir en fonction du problème.

Récompense Afin que les agents puissent apprendre leur comportement, l'environnement calcule une récompense pour chaque agent. Cette récompense est ensuite fournie à l'agent. Le calcul de la récompense, et l'explication des choix pour ce calcul, sont détaillés en section 4.3.

Paramètres de simulation Le simulateur repose sur un ensemble de paramètres, que l'on peut modifier avant chaque simulation afin de changer le comportement des agents par exemple. La plupart des paramètres de simulation sont relatifs au module d'apprentissage (ils sont alors nommés "hyperparamètres" dans la littérature), on notera cependant 2 exceptions : la taille et le délai de la fenêtre temporelle. La fenêtre temporelle est un mécanisme qui agrège les propriétés et perceptions pendant un certain nombre pas de temps. Par exemple, soit D le délai et S la taille, alors au pas de temps t, un agent obtiendra les perceptions agrégées des temps $[t - D, t - D + S]$. Ce mécanisme a été conçu pour les cas où le résultat d'une action serait éloigné dans le temps (par rapport à l'action elle-même). Dans le cas des *Smart Grids*, ce mécanisme est désactivé ($D = 0$, $S = 1$), car on considère que les actions ont des répercussions immédiates.

4.2 Application du simulateur aux *Smart Grids*

Les éléments génériques décrits précédemment sont spécialisés afin de correspondre au cas d'étude choisi.

Environnement L'environnement représente une grille locale, composée de plusieurs bâtiments. La grille dispose d'une certaine quantité d'énergie disponible à chaque étape (que les agents doivent se répartir). Les agents eux-mêmes disposent d'une petite quantité d'énergie dans une "batterie personnelle" (que l'on

```
Data : env: Environnement, agents: Liste d'Agents, nb_etapes: Entier
// Initialisation
etape ← 1;
récompenses ← ∅;
actions ← ∅;
while etape < nb_etapes do
    agents ← Mélanger(agents);
    foreach agent a dans agents do
        p ← env->CalculerPerception (a);
        DonnerPerception (a,p);
        if récompenses ≠ ∅ then
            r ← récompenses[a];
            DonnerRécompense (a,r);
        action ← ChoisirAction (a);
        actions ← actions ∪ {action};
    end
    actions ← Réguler (actions);
    Faire (env,actions);
    récompenses ← CalculerRécompenses (env);
    etape ← etape + 1;
end
```

Algorithme 2 : Déroulement d'une simulation.

nomme "stockage"). Cette batterie est alimentée à chaque étape (cela correspond à l'énergie rendue disponible par les équipements relatifs aux énergies renouvelables sur les bâtiments, tels que les panneaux photo-voltaïques).

L'instance de régulation intervient lorsque les agents utilisent plus d'énergie que la grille n'en possède, en simulant un achat d'énergie de la grille auprès du réseau national (que l'on nomme "surconsommation").

Agent Les agents ont un certain besoin en énergie, en fonction de leur profil et de l'heure de la journée, que l'on note $Besoin_{a,t}$. Ils consomment de l'énergie afin de satisfaire ce besoin, noté $Consommation_{a,t}$. On appelle confort, noté $Confort_{a,t} = f_a(Besoin_{a,t}, Consommation_{a,t})$, ce rapport entre consommation et besoin. Dans la littérature relative aux *Smart Grids*, on parle aussi d'effort ; dans notre cas, $Effort_{a,t} = 1 - Confort_{a,t}$. Il est à noter que le confort n'est pas maximal quand $Consommation_{a,t} = Besoin_{a,t}$, mais sera plutôt de 50% en général (les agents se sentent confortables quand leur consommation dépasse leur besoin). La fonction f_a dépend de l'agent, car certains sont plus prioritaires, aussi leur confort ne sera pas le même considérant le même ratio $\frac{consommation}{besoin}$. Cette fonction f_a permet de contrôler cette courbe (dans notre cas, la priorité se réduit simplement au profil, i.e. la fonction f_a est la même pour tous les agents ayant le même profil). Voir la figure 5 pour une représentation graphique de ces courbes.

Action Les deux profils d'agents ont accès à une unique action, constituée de multiples paramètres (en revanche, l'intervalle de valeurs autorisées pour les deux profils n'est pas le même). Le choix de l'action à chaque étape se traduit en un choix de valeur pour chacun des paramètres suivants :

— `Consommer grille` Consomme une quantité d'énergie q issue de la grille.

— `Consommer stock` Consomme une quantité d'énergie p issue de la batterie (le stock personnel).

— `Donner` Transfère une quantité d'énergie s du stockage vers la grille.

— `Stocker` Extrait une quantité d'énergie u de la grille pour la stocker dans la batterie.

— `Acheter` Achète une quantité d'énergie r depuis le réseau national pour la stocker dans la batterie.

— `Vendre` Vend au réseau national une quantité d'énergie z issue de la batterie.

Ce choix constitue un vecteur $\in \mathbb{R}^6 = [q, p, s, u, r, z]$. Notons que l'énergie placée dans le stockage au temps t peut être utilisée au même temps t.

Propriétés Les mesures de l'environnement que nous utilisons sont définies dans la table 1.

Nom	Description	Formule
Équité	Mesure statistique de dispersion (Gini) des conforts	$1 - Gini(Conforts)$
Perte en énergie	Énergie disponible non utilisée (gaspillée)	$\frac{Disponible - Surconsomme - \sum_a (s_a - q_a - u_a)}{Disponible + Surconsomme + \sum_a s_a}$
Autonomie	Non-interaction avec le réseau national (mode îlot)	$\frac{\sum_a Consommation_{a,t}}{\lvert Agents \rvert \times \sum_a (r_a + z_a)}$
Exclusion	Proportion d'agents dont effort $> 150\%$ médiane	$\frac{\lvert\{e \lvert e > 1.5 \times mediane(Efforts), e \in Efforts\}\rvert}{\lvert Agents \rvert}$
Bien-Être	Médiane du confort des agents	$mediane(Conforts)$
Surconsommation	Quantité d'énergie prise en excès	$\frac{Surconsomme}{\sum_a (q_a + u_a)}$

Table 1: Définition des propriétés (mesures objectives de l'environnement).

Perception Les perceptions que les agents reçoivent sont récapitulées dans la table 2 ; en particulier, notons que l'agent dispose de perceptions issues de l'environnement (heure, énergie disponible) et issues de lui-même (auxquelles les autres agents n'ont pas accès : stockage personnel, confort, profit). De plus, les propriétés de l'environnement font partie intégrante de ces perceptions.

4.3 Apprentissage et décision

Vue d'ensemble Le modèle d'apprentissage que nous proposons doit permettre aux agents d'apprendre à partir d'une récompense, des comportements (représentés par des actions avec de multiples paramètres continus) dans un espace d'états

	Nom	Description
Env.	Heure	Heure de la journée
	Énergie disponible	Qté. d'énergie disponible dans la SG
	Propriétés	L'ensemble des 6 propriétés de l'env. définies
Soi	Stockage	Batterie de l'agent
	Confort	Confort au pas de temps précédent
	Profit	Énergie vendue - énergie achetée

Table 2: Description des perceptions que chaque agent reçoit (capteurs de l'environnement et sur soi-même) à chaque étape. Notons que les perceptions relatives à "soi" ne sont pas connues des autres agents.

également continu. Pour cela, en accord avec les contraintes identifiées et l'état de l'art, nous proposons un modèle, inspiré de Smith [24], basé sur le Q-Learning [29] et les Cartes Auto-Organisatrices (SOM [14]). Ce modèle mémorise l'intérêt d'une action dans un état grâce à la table des Q-Valeurs (QR2), en utilisant deux SOM pour la correspondance entre les états (resp. actions) discrets et les vecteurs de perception (resp. paramètres d'actions) continus (QR1).

La figure 2 schématise ce modèle et montre les étapes du processus de décision et apprentissage :

1. L'environnement envoie des perceptions à l'agent à partir de l'état actuel, ainsi qu'une récompense (dans le cas d'une étape $t > 1$).
2. L'agent calcule une discrétisation de l'état à partir de la perception (bloc "Discrétiser état" sur la figure), en s'appuyant sur la **carte des entrées** (*Input SOM*). Cette carte apprend ce faisant la typologie des vecteurs de perception qu'elle reçoit (au fil des étapes).
3. L'agent utilise cette discrétisation (hypothèse d'état) associée à la récompense pour apprendre si sa précédente action était intéressante (bloc "Apprendre").
4. L'agent décide, toujours en fonction de l'hypothèse d'état, de la prochaine action à effectuer, en s'appuyant sur la **Q-Table** et la **carte des actions** (*Action SOM*) (bloc "Décider").
5. L'action est envoyée à l'environnement. L'instance de régulation prend éventuellement des mesures pour empêcher des actions interdites par le concepteur de l'environnement.
6. L'environnement traite l'ensemble des actions faites (bloc "Calcul nouvel état"), en résulte un nouvel état.
7. Cet état est utilisé par l'environnement pour calculer une valeur de satisfaction (la récompense), ainsi que des perceptions qui représentent l'état pour chaque agent (blocs "Calcul perception" et "Calcul récompense"). Le cycle de décision/apprentissage recommence ainsi.

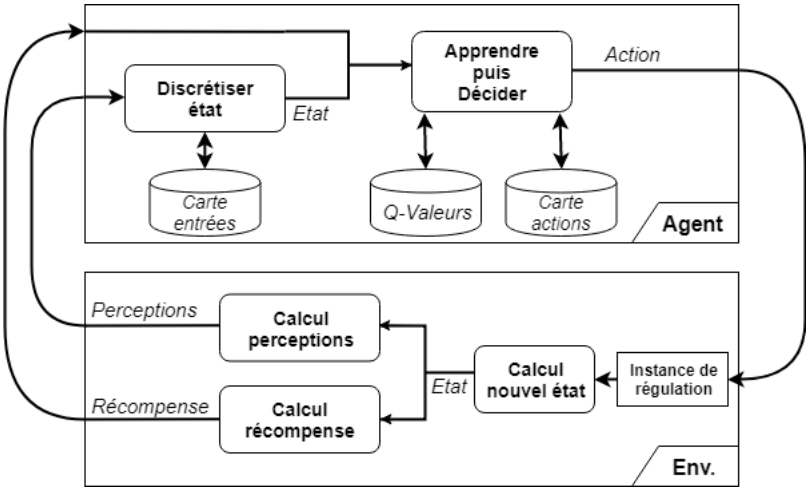

Fig. 2: Schéma du modèle d'apprentissage, centré sur la relation entre un agent donné et l'environnement. En particulier, l'état calculé par l'environnement n'est pas l'état que connaît l'agent.

Détail du processus de décision Le processus de discrétisation puis de décision est détaillé dans la figure 3. Le vecteur de perceptions X reçu par l'agent est comparé avec l'ensemble des neurones U de la **carte des entrées** afin de déterminer le neurone U_i dont le vecteur associé est le plus proche de X (également appelé le *Best Matching Unit*), c'est-à-dire U_i tel que $i = \arg\min ||X - U_i||$. Ce neurone U_i nous donne une hypothèse d'état e_i, c'est-à-dire que l'on regarde dans la Q-Table à la ligne i. Cette ligne contient l'ensemble des Q-Valeurs associées aux actions, par rapport à cette hypothèse d'état e_i (autrement dit, l'intérêt de faire une action, à supposer que l'on soit dans l'état e_i). On utilise ces Q-Valeurs afin de choisir une action a_j, en utilisant la politique dite Boltzmann (*Boltzmann Policy*) :

$$P(a_j) = \frac{e^{\frac{Q_{e_i,a_j}}{\tau}}}{\sum_k e^{\frac{Q_{e_i,a_k}}{\tau}}}$$

Cette formule représente la probabilité de choisir une action a_j, par rapport à sa Q-Valeur et la Q-Valeur des autres actions. Intuitivement, plus une Q-Valeur est forte par rapport aux autres, plus sa probabilité est élevée. τ est un hyperparamètre (aussi appelé la température) utilisé pour contrôler le dilemme exploration-exploitation : plus τ est élevé, plus on explore (i.e. on choisit des actions pour apprendre leur intérêt), à l'inverse quand τ est petit, on exploite (i.e. on choisit des actions qu'on pense intéressantes). On choisit donc l'action a_j selon les probabilités calculées par cette politique ; cela nous indique de regarder le vecteur W_j associé au neurone j dans la carte des actions. On appelle ce vecteur l'action proposée. Afin de permettre d'explorer l'espace des actions, on ajoute un bruit uniforme (paramétré par une constante ϵ) à chacun des poids du vecteur

(le bruit est différent selon chaque composante du vecteur), ce que l'on appelle l'action perturbée. Les poids de cette action perturbée sont les paramètres d'actions émis par le processus de décision (que l'on appelle vecteur perturbé W'_j).

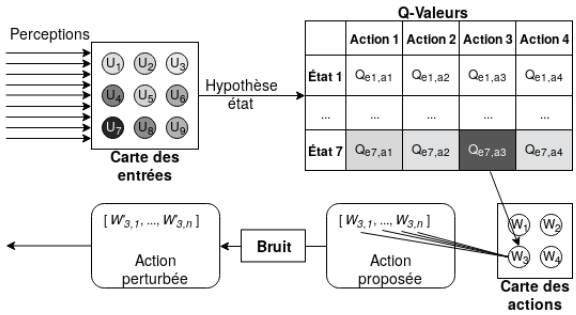

Fig. 3: Processus de décision détaillé, pour 1 agent.

Une fois que chaque agent a déterminé ses paramètres d'action, l'instance de régulation agit si nécessaire (en particulier pour gérer l'énergie sur-consommée), puis chaque agent calcule son confort (en fonction de sa consommation et de son besoin), et en informe l'environnement. Cette centralisation des informations est nécessaire pour calculer certaines propriétés (comme l'équité, puisqu'elle repose sur les conforts), mais seule l'instance de régulation dispose de ces informations (les autres agents n'auront accès qu'à l'information partielle représentée par les propriétés, pas le détail). L'environnement passe dans un nouvel état, puis les perceptions sont envoyées à chaque agent, associées à une récompense, afin que l'agent puisse apprendre si son action était intéressante ou non (i.e. si elle a contribué à améliorer l'environnement du point de vue de la récompense déterminée par le concepteur du système).

Détail du processus d'apprentissage En utilisant les informations reçues à ce nouveau pas de temps, l'agent doit mettre à jour la carte des entrées, la table des Q-Valeurs, et la carte des actions. Pour rappel, l'hypothèse d'état au temps `t` était e_i et l'action proposée par l'agent a_j (les paramètres d'actions étaient représentés par le vecteur perturbé W'_j).

1. On note r la récompense et X' le nouveau vecteur de perceptions. De la même manière que pour le processus de décision, on calcule le neurone le plus proche de ce vecteur ; on obtient une hypothèse d'état que l'on note e'_i.
2. Si $\underbrace{r + \gamma \max_k Q_{e'_i,a_k}}_{\substack{\text{Intérêt du} \\ \text{nouvel état } e'_i}} > \underbrace{Q_{e_i,a_j}}_{\substack{\text{Intérêt de} \\ a_j \text{ dans } e_i}}$, l'action était intéressante.

   Intuitivement, on regarde si la récompense reçue additionnée à la récompense

que l'on s'attend à obtenir dans le nouvel état est supérieure à l'intérêt mémorisé (*via* la table des Q-Valeurs) de l'action a_j dans l'ancien état e_i. Cela signifie que l'action perturbée est meilleure que l'action proposée. Dans ce cas, on met à jour la carte des actions en suivant la règle de mise à jour habituelle (i.e. en déplaçant le vecteur de chaque neurone k vers l'action perturbée, en fonction de la distance entre le neurone k et le neurone de l'action proposée j). Formellement, pour chaque neurone k :

$$W_k := W_k + \alpha_A \psi_A(j, k, \sigma_A)(W'_j - W_k)$$

3. On met à jour la table des Q-Valeurs, en mettant *toutes* les valeurs à jour (contrairement à la méthode habituelle qui consiste à ne mettre à jour que la valeur Q_{e_i,a_j}). Pour cela, on modifie la formule d'apprentissage du Q-Learning (dérivée de l'équation de Bellman) en prenant en compte le voisinage de la carte des entrées et le voisinage de la carte des actions. Formellement, pour chaque état m et chaque action n :

$$Q_{e_m,a_n} := Q_{e_m,a_n} + \alpha_Q \underbrace{\psi_E(i, m, \sigma_E)}_{\substack{\text{Carte des} \\ \text{entrées}}} \underbrace{\psi_A(j, n, \sigma_A)}_{\substack{\text{Carte des} \\ \text{actions}}} (r + \gamma \max_k Q_{e'_i,a_k} - Q_{e_m,a_n})$$

4. Enfin, on met à jour la carte des entrées, en suivant la méthode habituelle, comme pour la carte des actions. Formellement :

$$\forall k \; U_k := U_k + \alpha_E \psi_E(i, k, \sigma_E)(X - U_k)$$

Dans les équations précédentes, différents hyperparamètres interviennent ; Q en indice indique que le paramètre est relatif à la Q-Table, E indique que le paramètre est relatif à la carte des entrées et A indique que le paramètre est relatif à la carte des actions.

γ Le facteur d'actualisation (*Discount Factor*) : permet de contrôler l'horizon. Une valeur proche de 0 implique que l'agent cherche à optimiser ses récompenses à court terme, tandis qu'une valeur proche de 1 implique que l'agent cherche à optimiser ses récompenses à long-terme.

α Le taux d'apprentissage (*Learning Rate*) : permet de contrôler la vitesse de modification des valeurs (Q-Valeurs ou vecteurs) et donc permet de contrôler la convergence. Une valeur proche de 0 implique qu'on modifie très peu les valeurs, cela permet en général une meilleure convergence mais plus lente ; une valeur élevée implique qu'on modifie beaucoup les valeurs, cela permet une apprentissage plus rapide mais cela peut parfois empêcher le modèle de converger.

ψ La fonction de voisinage : cette fonction définit la valeur de voisinage d'un neurone par rapport au neurone "gagnant" (le *Best Matching Unit*). Elle est maximale (= 1) quand on compare le neurone gagnant avec lui-même, et décroît ensuite jusqu'à 0 au plus on s'éloigne de ce neurone. Cela permet de déplacer non seulement le neurone gagnant mais aussi son entourage vers le vecteur à apprendre.

σ Taille du voisinage : cette valeur définit la taille du voisinage dans la fonction de voisinage ; intuitivement, les neurones au-delà de cette distance ont une valeur de voisinage de 0 (et ne sont donc pas mis à jour).

Nous avons ainsi détaillé le processus de décision et d'apprentissage des agents, basé sur l'algorithme *Q-Learning* et utilisant des cartes de Kohonen pour la gestion des perceptions et actions continus, de même que la boucle d'interaction entre l'agent et l'environnement.

5 Résultats

Afin d'expérimenter sur un cas concret et par la suite d'évaluer notre modèle, nous avons dans un premier temps configuré les éléments laissés au concepteur (e.g. les fonctions de besoin, voir la section 5.1) ; dans un second temps, nous avons défini les fonctions de récompense (voir la section 5.2) ; enfin, nous avons déterminé empiriquement les hyperparamètres des algorithmes d'apprentissage à utiliser (voir la section 5.3). Nous présentons les résultats obtenus sur divers scénarios dans la section 5.4.

5.1 Configuration des simulations

Notre modèle de simulateur utilise plusieurs fonctions et valeurs qui sont à définir par le concepteur :

— Fonction de besoin de chaque agent. Nous avons choisi de nous inspirer de données réelles, mais modifiées au niveau de l'échelle (pour considérer 1 agent Hôpital et 10 agents Habitation plutôt que des milliers). La figure 4 récapitule les valeurs utilisées, par heure et par profil.

— Fonction de confort pour chaque agent (voir ci-dessous).

— Capacité des batteries (500 pour Habitation, 2000 pour Hôpital [7]).

— Intervalle de valeurs autorisées pour les paramètres d'action ([0,1000] pour Habitation, [0,4000] pour Hôpital). Cela signifie qu'ils ne peuvent pas consommer e.g. plus de 1000 de la part de la grille, mais peuvent consommer 1000 et acheter 1000. S'ils essaient de mettre plus d'énergie que possible dans le stockage, l'énergie est perdue.

— Fonction d'énergie disponible dans la grille : nous tirons une valeur aléatoire $\in [0, 5000]$ selon une distribution uniforme à chaque étape.

— Fonction d'énergie disponible dans la batterie : nous ajoutons à la quantité disponible à l'étape précédente une valeur aléatoire $\in [0, 100]$, tirée selon une distribution uniforme pour chaque agent.

— Instance de régulation : l'instance empêche les situations où la consommation est supérieure à l'énergie disponible, en simulant un achat de la part de la grille, représenté par la surconsommation.

7. L'unité n'est pas importante ici, mais est la même que le besoin défini précédemment.

Nous avons défini les fonctions de confort en utilisant des courbes logistiques généralisées (ce type de courbe est facilement paramétrable, ce qui nous permet de définir deux fonctions similaires mais dont la croissance est plus ou moins rapide par exemple), voir la figure 5 pour une représentation graphique. La formule de confort du profil "Habitation" est la suivante :

$$Confort_a = \frac{1}{1 + e^{-0.12 \times (Consommation_a - Besoin_a)}} \tag{1}$$

La formule de confort du profil "Hôpital" est la suivante :

$$Confort_a = \begin{cases} \frac{1}{1+e^{-0.004 \times (Consommation_a - Besoin_a)}}, & \text{si } Consommation_a \leq Besoin_a \\ \frac{1}{1+e^{-0.015 \times (Consommation_a - Besoin_a)}}, & \text{si } Consommation_a > Besoin_a \end{cases} \tag{2}$$

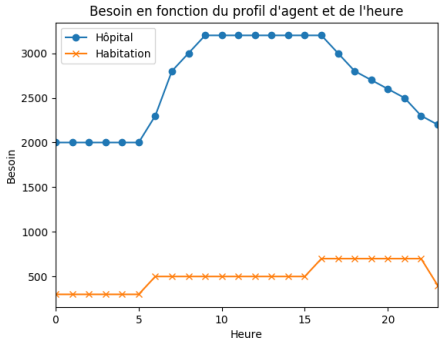

Fig. 4: Besoin en énergie, en fonction du profil de l'agent et de l'heure de la journée.

5.2 Récompenses utilisées dans les expérimentations

Afin d'évaluer la capacité d'apprentissage de notre modèle, nous avons défini plusieurs fonctions de récompenses, chacune utilisant une ou plusieurs propriétés issues de l'environnement (parmi celles décrites dans le tableau 1). Nous avons conçu ces fonctions pour permettre à chaque agent d'apprendre un comportement exhibant une valeur éthique que nous jugeons intéressante par rapport au cas d'étude des *Smart Grids*. Les comportements doivent également être cohérents avec les objectifs intrinsèques de ce problème. Par exemple, nous ne voulons pas que nos agents apprennent à ne pas consommer d'énergie ; bien que cette solution serait acceptable du point de vue de l'équité, elle n'est pas applicable en pratique (ce n'est pas l'objectif des *Smart Grids* que de supprimer entièrement la consommation d'électricité). Ainsi, en utilisant la propriété d'équité,

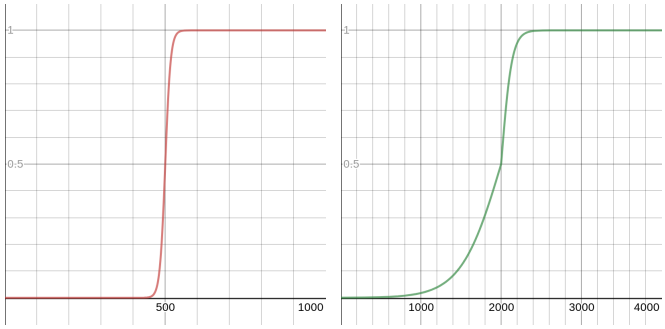

Fig. 5: Confort calculé selon la consommation pour un agent "Habitation" (à gauche) et un agent "Hôpital" (à droite). Le besoin de l'agent "Habitation" est fixé à 500, celui de l'agent "Hôpital" à 2000 (changer le besoin déplace le point d'inflexion de la courbe).

nous nous attendons à ce que les agents apprennent un comportement équitable (sans forcément limiter leur consommation) ; en utilisant la surconsommation, nous nous attendons à ce qu'ils limitent leur consommation (ce que l'on peut associer à une valeur écologique). Définir plusieurs fonctions différentes permet de tester une variété de valeurs (et ainsi prendre du recul sur la possibilité que l'apprentissage d'une valeur en particulier soit due à un biais dans la fonction de récompense).

Équité 1

$$r_{equite,a} = equite(Conforts) - equite(Conforts \backslash \{a\})$$

avec $Conforts$ l'ensemble des conforts, et $equite(Conforts) = 1 - gini(Conforts) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}|Conforts_i - Conforts_j|}{2n\sum_{i=1}^{n} Conforts_i}$.

Cette fonction utilise la propriété d'équité définie au niveau de l'environnement (elle-même utilisant le coefficient de Gini, pour rappel un indicateur statistique de dispersion). Elle représente à quel point l'agent a influe sur ce coefficient, i.e. s'il augmente la dispersion des conforts ou la réduit. Après expérimentations, nous avons remarqué que le coefficient de Gini ne semble pas permettre de calculer une contribution qui soit significative à nos yeux.

Équité 2

$$r_{equite2,a} = Conforts_a - moyenne(Conforts \backslash \{a\})$$

avec $Conforts$ l'ensemble des conforts.

Nous avons défini cette fonction pour corriger le problème de la fonction précédente et obtenir des récompenses dont la valeur absolue serait plus grande. On pourrait toutefois objecter qu'elle contient un biais supplémentaire, puisque nous utilisons la valeur de confort de l'agent directement.

Autrement dit, à moyenne des conforts égale, l'agent sera plus récompensé si son confort est plus élevé (ce qui peut le pousser à consommer plus).

Surconsommation

$$r_{surconsommation,a} = Surconsommation - \frac{surconsomme}{\sum_{i \neq a} pris_i}$$

avec $Surconsommation$ la propriété de Surconsommation de l'environnement, on rappelle le vecteur de paramètres d'action [Consommer grille q, Consommer stock p, Donner s, Stocker u, Acheter r, Vendre z], $surconsomme = \sum_{i \neq a}(q_a + u_a - s_a) - Disponible$ la quantité totale d'énergie sur-consommée par les agents, et $pris_i = q_i + u_i$ la quantité d'énergie extraite par l'agent i. Intuitivement, nous comparons la propriété de surconsommation avec ce qu'elle serait sans l'agent a ; nous divisons par la quantité d'énergie prise afin d'obtenir une valeur entre 0 et 1. Cette fonction pousse les agents à réduire la surconsommation et donc par extension leur consommation.

Multi-Objectif Produit (MOP)

$$r_{mop,a} = r_{surconsommation,a} \times Confort_a$$

avec $r_{surconsommation,a}$ la fonction définie précédemment et $Confort_a$ le confort de l'agent a.
L'objectif de cette fonction est d'opposer la valeur de surconsommation au confort personnel de l'agent ; en d'autres termes, nous souhaitons que l'agent apprenne à minimiser sa surconsommation, tout en maximisant son confort.

Multi-Objectif Somme (MOS)

$$r_{mos,a} = 0.8 \times r_{surconsommation,a} + 0.2 \times Confort_a$$

Cette fonction est une alternative à la précédente, en utilisant une somme plutôt qu'un produit (comme nous l'avons vu dans 4.3). De plus, nous ajoutons comme information que la réduction de la surconsommation est plus importante que le confort personnel de l'agent. Les poids ont été choisis à cet effet, et il serait intéressant d'observer leur impact sur le score final.

Adaptabilité 1

$$r_{adaptabilite1,a} = \begin{cases} r_{equite,a} & \text{si étape} < 3000 \\ r_{mos,a} & \text{sinon} \end{cases}$$

avec $r_{equite,a}$et $r_{mos,a}$ les fonctions définies précédemment.
Nous avons défini cette fonction afin d'évaluer la capacité de l'agent à s'adapter lorsque la fonction de récompense est entièrement remplacée (il n'y a, à priori, pas de lien entre l'équité et la fonction MOS que l'agent pourrait utiliser, à la façon du *Transfer Learning*). Le nombre d'étapes avant changement a été déterminé expérimentalement, en observant à partir de quand le module d'apprentissage semblait converger (en particulier, la carte des entrées et la valeur de récompense obtenue). Nous faisons l'hypothèse que l'agent changera (presque) complètement son comportement, puisque la fonction de récompense est différente.

Adaptabilité 2

$$r_{adaptabilite2,a} = \begin{cases} r_{equite,a} & \text{si étape} < 2000 \\ \frac{r_{equite,a}+r_{surconsommation,a}}{2} & \text{sinon} \end{cases}$$

Cette fonction vise à évaluer la capacité d'adaptation de l'agent également, mais contrairement à la précédente nous conservons une partie de la fonction de récompense. Au lieu de la remplacer complètement, nous gardons l'aspect équité, et nous ajoutons la fonction de surconsommation. Ainsi, nous faisons l'hypothèse que l'agent parviendra à conserver ses acquis relatifs à l'équité, et devra adapter une partie seulement de son comportement pour intégrer la surconsommation. Nous diminuons également l'étape à laquelle le changement s'opère (puisque l'agent continuera à apprendre la fonction d'équité, nous considérons qu'il n'est pas nécessaire d'atteindre la même convergence).

Adaptabilité 3

$$r_{adaptabilite3,a} = \begin{cases} r_{equite,a} & \text{si étape} < 2000 \\ \frac{r_{equite,a}+r_{surconsommation,a}}{2} & \text{si étape} < 6000 \\ \frac{r_{equite,a}+r_{surconsommation,a}+Confort_a}{3} & \text{sinon} \end{cases}$$

Cette fonction est semblable à la précédente, en ajoutant une 3ème phase. Ainsi, nous évaluons le nombre de valeurs que l'agent peut apprendre à la suite.

5.3 Hyper-paramètres

Comme nous l'avons vu dans la partie 4.3, le module d'apprentissage et de décision utilise de nombreux hyperparamètres, que nous avons déterminés empiriquement (voir les Annexes B et C) en essayant diverses combinaisons de valeurs pour ces paramètres, et en comparant le score obtenu (moyenné sur 10 simulations). Le tableau 3 présente les paramètres utilisés pour la suite des résultats.

5.4 Résultats

Nous évaluons notre modèle selon deux axes : le premier est celui de l'apprentissage (par rapport à une fonction de récompense donnée), le deuxième est celui des comportements relatifs à l'éthique. Pour ce faire, nous avons défini plusieurs scénarios de simulation : dans chacun d'entre eux, nous utilisons 10 agents de profil "Habitation" et 1 agent de profil "Hôpital". Le score que nous calculons à la fin des simulations est la moyenne des récompenses "globales" (i.e. la fonction de récompense r appliquée à l'environnement entier et pas un agent en particulier). Nous comparons ces scores entre scénarios pour observer la capacité d'apprentissage du modèle (premier axe) ; puis nous calculons l'évolution de l'équité pour chaque fonction de récompense, afin d'observer celle qui permet le plus de se rapprocher des comportements équitables. Les scénarios sont :

Module	Nom du paramètre	Valeur
Carte des entrées	Forme	7x7
	Taux d'apprentissage (α_E)	0.8
	Taille du voisinage (σ_E)	1
Carte des actions	Forme	5x5
	Taux d'apprentissage (α_A)	0.8
	Taille du voisinage (σ_A)	1
Q-Learning	Taux d'apprentissage (α_Q)	0.01
	Facteur d'actualisation (γ)	0.9
	Fonction de perturbation	Epsilon
	Paramètre de bruit (ϵ)	0.08
	Politique d'exploration	Boltzmann
	Paramètre d'exploration (τ)	0.5
	Décroissance de τ	Non

Table 3: Valeurs des hyperparamètres utilisés pour l'ensemble des expérimentations.

Aléatoire Le processus de décision et d'apprentissage est remplacé par une fonction aléatoire uniforme (chaque paramètre d'action est tiré au hasard de manière indépendante). Ce scénario nous sert de comparaison pour les autres scénarios.

SoloLearner Un seul des agents "Habitation" utilise le processus de décision et d'apprentissage ; les autres (y compris l'agent "Hôpital") utilisent une fonction aléatoire comme dans le scénario Aléatoire. Cela nous permet d'évaluer d'une part l'impact d'un unique agent (en comparant le score de ce scénario au score du scénario "Aléatoire") et d'autre part d'évaluer si l'agent est capable d'apprendre de s'adapter à des comportements aléatoires.

Apprentissage Il s'agit du scénario "classique", où tous les agents apprennent.

InitZero Ce scénario est similaire au scénario "Apprentissage", cependant les vecteurs des cartes d'actions sont initialisés à 0 (au lieu d'être aléatoirement initialisés). L'intuition qui motive ce choix est le fait de permettre aux agents de faire tendre les paramètres d'action vers 1 tant que la récompense augmente, et de s'arrêter lorsque la récompense redescend.

Les résultats pour chaque scénario et chaque fonction de récompense sont résumés dans la table 4.

À partir de ces résultats, on peut observer premièrement que le scénario "Apprentissage" a un score plus élevé que le scénario "Aléatoire" dans tous les cas sauf "MultiObjectif Somme". Le scénario "InitZero" a un score plus élevé sur ce cas, donc cette récompense peut être apprise. En revanche, les scores sont très similaires entre les différents scénarios dans le cas de "MultiObjectif Produit" (0.55 - 0.58). On peut se demander si cette récompense peut être correctement apprise.

Le scénario "SoloLearner" a un score supérieur au scénario "Aléatoire", mais de peu (entre 0 et 0.02). Par rapport à notre hypothèse de départ, on peut donc

	Aléatoire	**SoloLearner**	**Apprentissage**	**InitZero**
équité	0.76	0.78	0.99	0.62
équité 2	0.76	0.78	0.99	0.89
surconsommation	0.18	0.19	0.56	0.94
MO-produit	0.57	0.57	0.58	0.55
MO-somme	0.33	0.34	0.32	0.53
adaptabilité 1	0.46	0.47	0.52	0.44
adaptabilité 2	0.53	0.54	0.67	0.58
adaptabilité 3	0.59	0.60	0.72	0.64

Table 4: Score pour chaque fonction de récompense et chaque scénario. Remarque : comme le score dépend directement de la fonction de récompense, la comparaison entre deux lignes n'est pas recommandée (en particulier, un score plus élevé ne signifie pas que la fonction de récompense est meilleure ; elle peut aussi être plus "facile" à satisfaire).

assumer que l'agent parvient à s'adapter aux comportements aléatoires, mais que son impact n'est pas très grand.

Le scénario "InitZero" a un score inférieur à celui de "Apprentissage" dans la plupart des cas, similaire dans le cas "MultiObjectif Produit", et supérieur dans les cas "Surconsommation" et "MultiObjectif Somme". La réussite avec la récompense "surconsommation" s'explique par le fonctionnement de cette récompense. Comme elle est maximale si les agents ne consomment pas, commencer avec un vecteur initialisé à 0 (en particulier le paramètre de consommation) offre un avantage certain. Le choix d'initialiser les vecteurs d'actions à 0 ou non doit donc dépendre de la fonction de récompense.

Nous comparons ensuite l'évolution de la valeur d'équité dans le scénario "Apprentissage", pour chacune des fonctions de récompense. La moyenne de l'équité sur le temps est proche de 1 pour la plupart des récompenses utilisées (*équité* et *équité 2* en tête, ce qui est logique), à l'exception de *surconsommation* (0.26). Cela montre que les récompenses définies permettent, pour la plupart, d'apprendre la valeur d'équité (voir l'annexe J).

Les annexes C à P détaillent ces résultats avec plusieurs graphiques, notamment sur les scores reçus par étape, mais aussi sur les modules d'apprentissage de l'agent (notamment l'activation des cartes de Kohonen), ou encore les graphiques d'évolution de la valeur d'équité.

6 Discussions

6.1 Rappel de la contribution

Nous avons conçu un simulateur multi-agent adapté au problème de la répartition d'énergie dans les *Smart Grids*. Nous avons abordé le problème de l'éthique dans les agents artificiels par l'apprentissage par renforcement (qui, à notre connaissance, n'a pas été exploré jusque là). Le processus d'apprentissage que nous avons défini s'appuie sur un travail existant ([24]), basé sur l'utilisation d'un algorithme de Q-Learning dont les états et actions sont représentés par des neurones (dans des cartes de Kohonen). Nos contributions dans ce processus d'apprentissage sont (par rapport aux travaux existants) :

— Prise en compte d'espace de plus grande dimension : 11 pour les entrées contre 6, 6 pour les actions contre 1 à 2 dimensions.

— Choix des actions par une politique Boltzmann au lieu d'une politique ϵ-cupide ; cela permet plus d'exploration des actions.

— Utilisation de fonctions de récompenses différenciées (*Multi-Agent Credit Assignment Problem*), multi-objectifs, et combinant ces deux propriétés.

De plus, nous avons mis en place ce processus d'apprentissage dans un contexte multi-agent, intégré à notre simulateur. Le module de décision (Q-Table, carte des entrées, carte des actions) est dupliqué pour chaque agent. Nous avons évalué ce modèle sur plusieurs récompenses, en observant une bonne capacité à apprendre dans la plupart des cas, ainsi que des comportements exhibant la valeur d'équité.

6.2 Limitations et perspectives

Nous nous sommes inspirés du paradigme constructiviste dans le sens où les agents construisent leur propre représentation du monde (l'état est discrétisé à partir des perceptions, et ce processus est appris au fur et à mesure de l'interaction entre l'agent et son environnement) ; toutefois, notre processus d'apprentissage de la représentation manque d'une rétro-action de la récompense pour être considéré comme vraiment constructiviste. Intuitivement, le paradigme constructiviste implique que l'apprentissage de la discrétisation est contrôlé par le choix des actions, afin que la représentation construite par l'agent soit utile au processus de décision. En d'autres termes, si l'agent est incapable de choisir une bonne action, cela vient peut-être du fait que l'hypothèse d'état ne lui permet pas de se représenter correctement l'état réel, et donc qu'il faut modifier la façon de calculer cette hypothèse d'état. Dans notre cas, nous n'avons pas de tel mécanisme, la carte des entrées apprend simplement les vecteurs de perceptions reçus, peu importe la récompense associée.

Une des limitations bien connue des algorithmes d'apprentissage par renforcement est l'importance de la fonction de récompense ; en effet, certaines fonctions (notamment les non continues ou non différentiables) ne permettent pas un apprentissage que l'on pourrait considérer comme "de qualité". Dans notre cas,

nous avons remarqué que la fonction de récompense Multi-Objectif Produit en particulier échouait. Il serait intéressant, pour aller plus loin et vérifier la capacité d'apprentissage de notre modèle, d'essayer des fonctions connues pour être difficiles (e.g. la fonction de Rastrigin [8]).

L'exploration des actions est entièrement aléatoire : à partir de vecteurs initialisés aléatoirement, on ajoute un bruit sur chacune des composantes. Si cette idée semble fonctionner dans des espaces de faible dimension (comme par exemple la recherche d'un minima dans un espace de dimension 2), il semble plus difficile d'explorer efficacement dans un espace avec une dimension plus élevée (dans notre cas, 6). En effet, si une perturbation d'une composante est intéressante, il paraît probable qu'au moins une des autres composantes sera modifiée, et peut-être d'une façon qui annule cet intérêt. Une piste qui nous semble intéressante pour éviter ce problème serait d'effectuer une perturbation sur une seule des composantes à la fois (en choisissant cette composante au hasard). Cela pourrait permettre une convergence plus lente, et donc éventuellement de trouver des actions plus intéressantes, mais au prix d'un temps nécessaire beaucoup plus élevé (puisque les vecteurs ne seraient déplacés que dans une seule dimension à la fois, les "sauts" dans l'espace des actions seraient moins chaotiques mais également plus courts).

Il serait également intéressant d'implémenter des mécanismes de diminution de certains hyperparamètres sur le temps (*decay*), notamment le voisinage des cartes de Kohonen σ, la température de la politique Boltzman τ ou encore le taux d'apprentissage α. Cela demande en particulier de tester correctement ces mécanismes et de déterminer (empiriquement) la formule optimale pour la diminution (e.g. rapide ou lente).

Enfin, nous nous étions fixés la capacité d'adaptation des agents comme objectif, notamment par rapport aux travaux existants ; la fonction de récompense Adaptabilité 2 semble montrer des résultats allant dans ce sens, mais les autres fonctions montrent au contraire un échec de l'agent face au changement. On peut se demander ainsi si cette capacité d'adaptabilité est une simple coïncidence dans le cas d'Adaptabilité 2. Nous n'avons pas de piste "simple" pour améliorer cet aspect, d'autres approches fondamentalement différentes tel que l'apprentissage par options [25] pourraient donner de meilleurs résultats.

Remerciements

Ce travail a été financé par la région Auvergne-Rhône Alpes, dans le cadre du projet *Ethics.AI*.

References

1. Ethique et agents autonomes `https://ethicaa.greyc.fr/media/files/ethicaa.white.paper.pdf`

8. `https://en.wikipedia.org/wiki/Test_functions_for_optimization`

2. Anderson, M., Anderson, S.L.: Toward ensuring ethical behavior from autonomous systems: a case-supported principle-based paradigm 42(4), 324–331, http://www.emeraldinsight.com/doi/10.1108/IR-12-2014-0434
3. Anderson, M., Anderson, S.L.: Guest editors' introduction: Machine ethics. IEEE Intelligent Systems 21(4), 10–11 (2006), https://doi.org/10.1109/MIS.2006.70
4. Bellman, R.: A markovian decision process. Journal of Mathematics and Mechanics pp. 679–684 (1957)
5. Berreby, F., Bourgne, G., Ganascia, J.: A declarative modular framework for representing and applying ethical principles. In: Larson, K., Winikoff, M., Das, S., Durfee, E.H. (eds.) Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017. pp. 96–104. ACM (2017), http://dl.acm.org/citation.cfm?id=3091145
6. Cointe, N.: Ethical Judgment for decision and cooperation in multiagent systems. Theses, Université de Lyon (Dec 2017), https://tel.archives-ouvertes.fr/tel-01851485
7. Cointe, N., Bonnet, G., Boissier, O.: Multi-agent based ethical asset management. In: Bonnet, G., Harbers, M., Hindriks, K.V., Katell, M., Tessier, C. (eds.) Proceedings of the 1st Workshop on Ethics in the Design of Intelligent Agents, The Hague, The Netherlands, August 30, 2016. CEUR Workshop Proceedings, vol. 1668, pp. 52–57. CEUR-WS.org (2016), http://ceur-ws.org/Vol-1668/paper9.pdf
8. Drescher, G.L.: Made-up minds: a constructivist approach to artificial intelligence. MIT press (1991)
9. Dyndal, G.L., Berntsen, T.A., Redse-Johansen, S.: Autonomous military drones-no longer science fiction-. Romanian Military Thinking (2) (2017)
10. Foot, P.: The problem of abortion and the doctrine of double effect. Oxford Review 5, 5–15 (1967)
11. Fritzke, B.: A growing neural gas network learns topologies. In: Advances in neural information processing systems. pp. 625–632 (1995)
12. Guerin, F.: Constructivism in AI: prospects, progress and challenges. In: Guerin, F., Vasconcelos, W.W. (eds.) AISB 2008 Convention: Communication, Interaction and Social Intelligence, 1st-4th April 2008, University of Aberdeen, UK. Computing & Philosophy, vol. 12, pp. 20–27. AISB (2008), http://www.aisb.org.uk/convention/aisb08/proc/proceedings/12%20Computing%20and%20Philosophy/04.pdf
13. Kalayeh, H.M., Landgrebe, D.A.: Predicting the required number of training samples. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-5(6), 664–667 (Nov 1983)
14. Kohonen, T.: The self-organizing map. Proceedings of the IEEE 78(9), 1464–1480 (Sep 1990)
15. Konda, V.R., Tsitsiklis, J.N.: Actor-critic algorithms. In: Advances in neural information processing systems. pp. 1008–1014 (2000)
16. North, M.J., Collier, N.T., Ozik, J., Tatara, E.R., Macal, C.M., Bragen, M.J., Sydelko, P.: Complex adaptive systems modeling with repast simphony. CASM 1, 3 (2013), https://doi.org/10.1186/2194-3206-1-3
17. Palensky, P., Dietrich, D.: Demand side management: Demand response, intelligent energy systems, and smart loads. IEEE transactions on industrial informatics 7(3), 381–388 (2011)

18. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Autonomous Agents and Multi-Agent Systems 11(3), 387–434 (2005), https://doi.org/10.1007/s10458-005-2631-2
19. Piaget, J.: The construction of reality in the child. Routledge (2013)
20. Rougier, N.P., Boniface, Y.: Dynamic Self-Organising Map. Neurocomputing 74(11), 1840–1847 (2011), https://hal.inria.fr/inria-00495827
21. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems, vol. 37. University of Cambridge, Department of Engineering Cambridge, England (1994)
22. Salge, C., Polani, D.: Empowerment as replacement for the three laws of robotics. Front. Robotics and AI 2017 (2017), https://doi.org/10.3389/frobt.2017.00025
23. Schwartz, S.H.: Universals in the content and structure of values: Theoretical advances and empirical tests in 20 countries. In: Advances in experimental social psychology, vol. 25, pp. 1–65. Elsevier (1992)
24. Smith, A.J.: Applications of the self-organising map to reinforcement learning. Neural Networks 15(8-9), 1107–1124 (2002), https://doi.org/10.1016/S0893-6080(02)00083-7
25. Stolle, M., Precup, D.: Learning options in reinforcement learning. In: International Symposium on abstraction, reformulation, and approximation. pp. 212–223. Springer (2002)
26. Stübs, M., Köster, K.: OpenDISCO – open simulation framework for distributed smart grid control 1(1), 32, https://doi.org/10.1186/s42162-018-0044-0
27. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning 3, 9–44 (1988), https://doi.org/10.1007/BF00115009
28. Sutton, R.S., Barto, A.G.: Reinforcement learning - an introduction. Adaptive computation and machine learning, MIT Press (1998), http://www.worldcat.org/oclc/37293240
29. Watkins, C.J.C.H., Dayan, P.: Q-learning 8(3), 279–292
30. Wilensky, U.: Netlogo. http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL (1999), http://ccl.northwestern.edu/netlogo/
31. Xu, D., Tian, Y.: A comprehensive survey of clustering algorithms. Annals of Data Science 2(2), 165–193 (Jun 2015), https://doi.org/10.1007/s40745-015-0040-1
32. Yliniemi, L.M., Tumer, K.: Multi-objective multiagent credit assignment through difference rewards in reinforcement learning. In: Dick, G., Browne, W.N., Whigham, P.A., Zhang, M., Bui, L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (eds.) Simulated Evolution and Learning - 10th International Conference, SEAL 2014, Dunedin, New Zealand, December 15-18, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8886, pp. 407–418. Springer (2014), https://doi.org/10.1007/978-3-319-13563-2_35
33. Ziemke, T.: The construction of 'reality'in the robot: Constructivist perspectives on situated artificial intelligence and adaptive robotics. Foundations of Science 6(1-3), 163–233 (2001)

A Valeurs de Schwartz

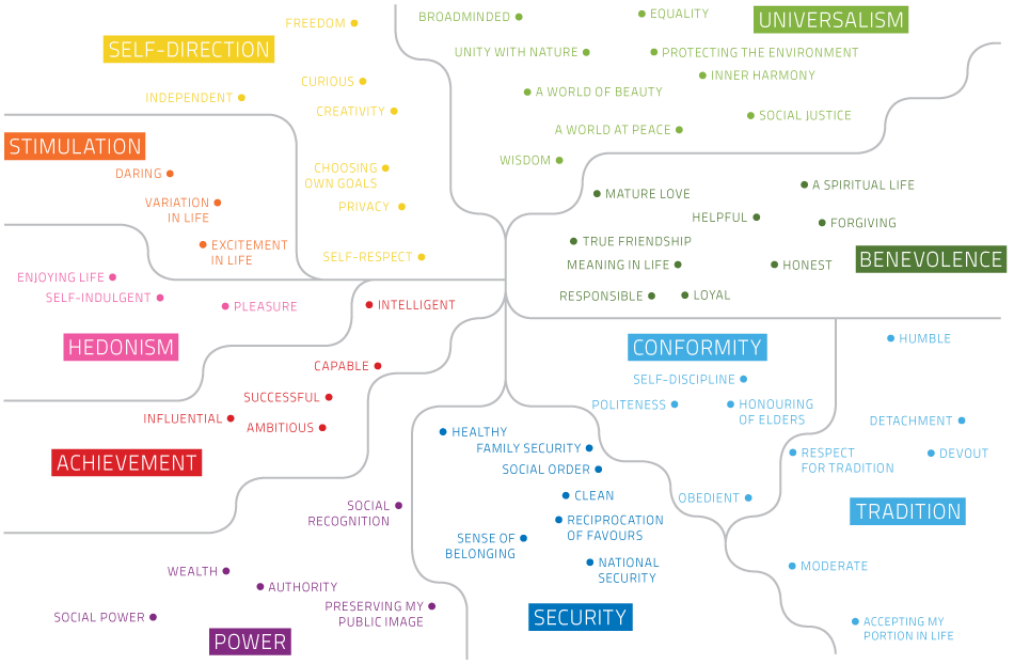

Fig. 6: Valeurs de Schwartz, regroupées par thème commun. On peut voir que les thèmes s'opposent entre eux, e.g. il est difficile de concilier "conformité" et "auto-détermination", ou "pouvoir" et "bienveillance".

B Réseaux de distribution intelligents

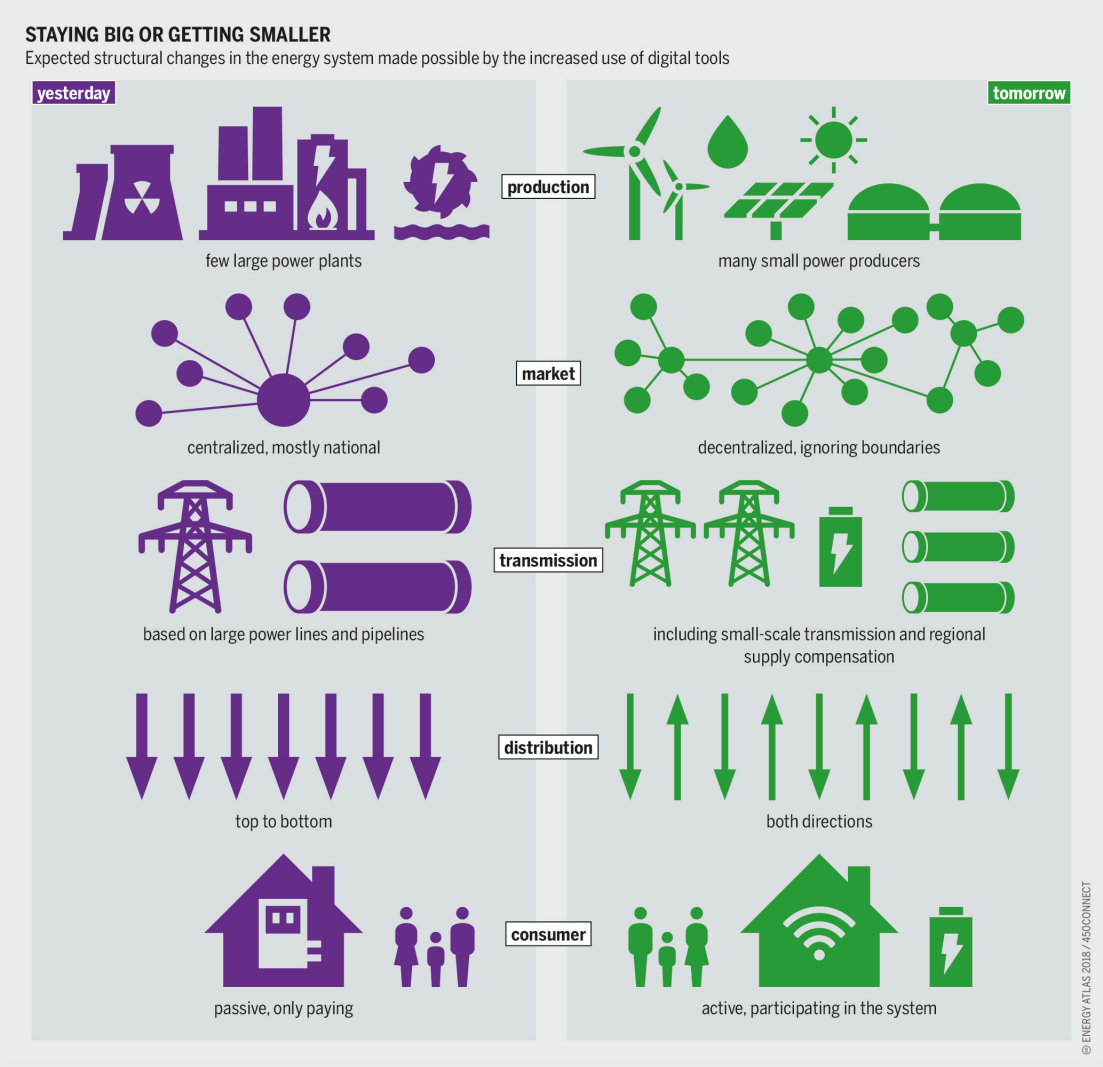

Fig. 7: Comparaison entre les *Smart Grids* (*tomorrow*) et les réseaux de distribution actuels *yesterday*). En particulier, les *Smart Grids* utilisent une production locale, décentralisée, avec des échanges à faible échelle, dans lesquels les *prosumers* participent activement.
Image extraite de Wikipédia.

C Paramètres de perturbation

Bruit	Méthode	Nombre étapes	Score
0.01	gaussian	3000	0.889458251987943
0.01	epsilon	3000	0.7105939516333781
0.02	gaussian	3000	0.8693511654776104
0.02	epsilon	3000	0.9047393744252257
0.03	gaussian	3000	0.8203773453760509
0.03	epsilon	3000	0.9383584018525672
0.04	gaussian	3000	0.7947613593831269
0.04	epsilon	3000	0.6841068234634197
0.05	gaussian	3000	0.759312201825191
0.05	epsilon	3000	0.8315343392107387
0.06	gaussian	3000	0.8125285164172874
0.06	epsilon	3000	0.7778613089609762
0.07	gaussian	3000	0.8606255142211846
0.07	epsilon	3000	0.8127568424662798
0.08	gaussian	3000	**0.9165698694838699**
0.08	epsilon	3000	**0.9701651754846973**

Table 5: Comparaison des perturbation "epsilon" et "gaussienne". La méthode "epsilon" consiste à ajouter un bruit suivant une distribution uniforme à chacune des composantes de l'action proposée, afin de créer l'action perturbée. La méthode "gaussienne" consiste à ajouter un bruit suivant une distribution normale à chacune des composantes de l'action proposée. La colonne "bruit" dans cette table représente un paramètre utilisé pour les deux distributions : dans le cas de la distribution uniforme, il s'agit des bornes (i.e. on tire une valeur dans [-Bruit,+Bruit]) ; dans le cas de la distribution normale, il s'agit de la variance (i.e. on utilise la distribution normale de moyenne 0 et d'écart type "bruit"). On observe que les résultats sont relativement proches quelle que soit la méthode utilisée, mais fortement corrélés avec le paramètre "bruit".

D Tests empiriques des hyperparamètres

Forme	Exploration	τ	Décroissance τ	Délai fenêtre	Taille fenêtre	Nombre étapes	Score
(10, 10)	epsgreedy	5.0	Non	5	3	10000	0.6696126779309193
(10, 10)	epsgreedy	5.0	Non	5	3	3000	0.7870137998809693
(10, 10)	epsgreedy	5.0	Non	5	3	3000	0.7317549485662113
(10, 10)	epsgreedy	5.0	Non	0	1	3000	0.9930139388575846
(10, 10)	epsgreedy	5.0	Non	0	1	3000	0.9933601455034079
(10, 10)	epsgreedy	5.0	Non	1	1	10000	0.6163637600994921
(10, 10)	epsgreedy	5.0	Non	0	1	10000	0.9954092289236277
(10, 10)	boltzmann	5.0	Oui	0	1	3000	0.9951351793039636
(10, 10)	boltzmann	5.0	Oui	5	3	3000	0.741695987483498
(10, 10)	boltzmann	1.0	Oui	5	3	3000	0.7404126221674404
(10, 10)	boltzmann	1.0	Non	5	3	3000	0.7949995559013409
(10, 10)	boltzmann	0.5	Non	5	3	3000	0.7119525119297728
(8, 8)	boltzmann	0.5	Non	5	3	3000	0.8096982188404453
(7, 7)	boltzmann	0.5	Non	5	3	3000	0.8118944033727079
(6, 6)	boltzmann	0.5	Non	5	3	3000	0.7207340125522812
(7, 7)	boltzmann	0.5	Non	0	1	3000	0.9905280870033849

Table 6: Quelques combinaisons de paramètres utilisées pour déterminer empiriquement les meilleurs paramètres à utiliser. Par manque de temps, peu de combinaisons ont pu être essayées au final (surtout considérant le nombre de paramètres disponibles). On remarque en particulier que les paramètres relatifs à la fenêtre temporelle ("délai fenêtre" et "taille fenêtre") sont parmi ceux qui impactent le plus le score final. Le paramètre "Forme" réfère à la taille de la carte des entrées (i.e. "(x,y)" réfère à une grille de x par y neurones). Nous n'avons pas eu le temps de tester la forme de la carte des actions, aussi la valeur de "(5,5)" a été choisie. Elle semble donner des résultats corrects en pratique. Le paramètre "Exploration" réfère à la méthode utilisée pour choisir une action (politique ϵ-cupide ou Boltzmann). Dans le cas de la politique ϵ-cupide, nous avons fixé $\epsilon = 0.05$. τ représente la température de la politique Boltzmann (ce paramètre est donc non utilisé lorsque l'exploration est ϵ-cupide).

E Méthodes d'exploration

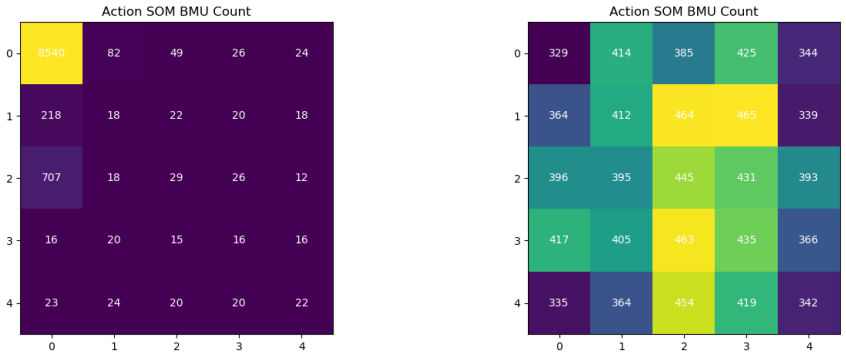

Fig. 8: Nombre de fois que chaque unité de la carte des actions a été considérée comme *Best Matching Unit*, i.e. nombre de fois que l'action a été choisie par l'agent. A gauche, les actions ont été choisies selon la politique ϵ-cupide (a) ; à droite, selon la politique Boltzmann (b). On remarque que les actions suivent une distribution relativement uniforme dans (b), alors qu'une seule action se démarque dans (a). On peut donc se demander si la politique ϵ-cupide ne se précipite pas vers un maximum local. En effet, le fait de choisir une action fait augmenter sa Q-Valeur (sauf si l'action était contre-productive), et donc augmente ses chances d'être choisie la prochaine fois. En particulier, avec 5% de probabilité de choisir une autre action, l'action n°1 peut être choisie suffisamment de fois de suite pour que les autres actions ne puissent espérer dépasser sa Q-Valeur. La politique Boltzmann n'a pas ce biais : les actions dont la Q-Valeur est proche reçoivent des probabilités proches, et donc peuvent être choisies.

F Comparaison des discrétisations des perceptions

Fig. 9: Nombre de fois que chaque unité de la carte des entrées a été considéré comme *Best Matching Unit*, i.e. nombre de fois que l'état a été choisi comme hypothèse d'état. A gauche (a), la carte des entrées d'un agent "Habitation" (similaire à n'importe quel autre agent du même profil) ; à droite (b), la carte des entrées d'un agent "Hôpital". On observe que plus d'états différents ont été fréquemment considérés comme hypothèses dans (b). Nous n'avons pas d'explication définitive à ce phénomène, mais nous formulons l'hypothèse que cela est dû à la capacité d'impact sur l'environnement plus importante des agents de type "Hôpital".

G Récompense - Adaptabilité 1

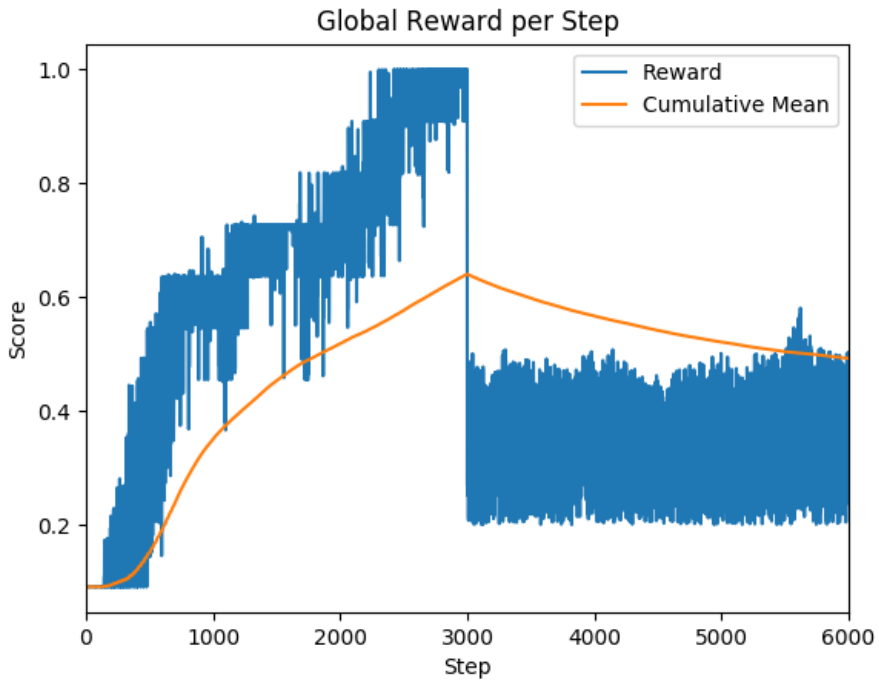

Fig. 10: Score par étape et somme cumulée des scores - Scénario "Apprentissage" - Fonction de récompense "Adaptabilité 1". On observe que les scores changent brutalement lorsque la récompense change, ce qui est logique. En revanche, nous pourrions nous attendre à ce que les scores progressent de la même manière que lors des étapes 0 à 3000, or ils stagnent autour de la même valeur. Nous en déduisons que l'agent n'a pas su s'adapter à une fonction de récompense radicalement différente de celle apprise en premier lieu.

H Récompense - Adaptabilité 2

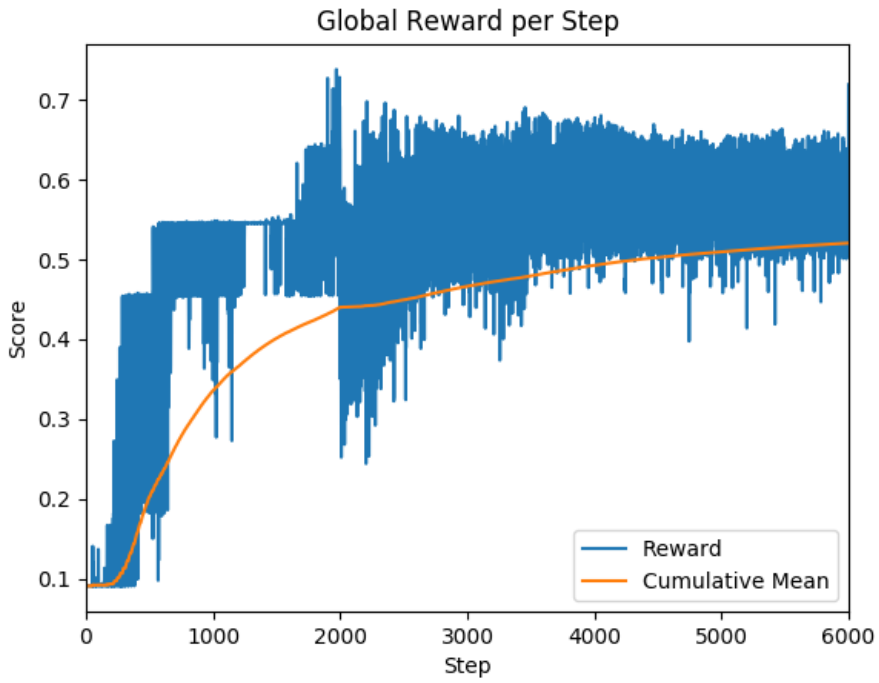

Fig. 11: Score par étape et somme cumulée des scores - Scénario "Apprentissage" - Fonction de récompense "Adaptabilité 2". De même que pour "Adaptabilité 1", les scores changent brutalement lorsque la récompense, mais cette fois on peut observer une augmentation sur le temps des scores reçus après ce changement. Cela est visible particulièrement avec la somme cumulée. L'agent semble donc s'adapter, au moins légèrement, au changement de fonction.

I Récompense - Adaptabilité 3

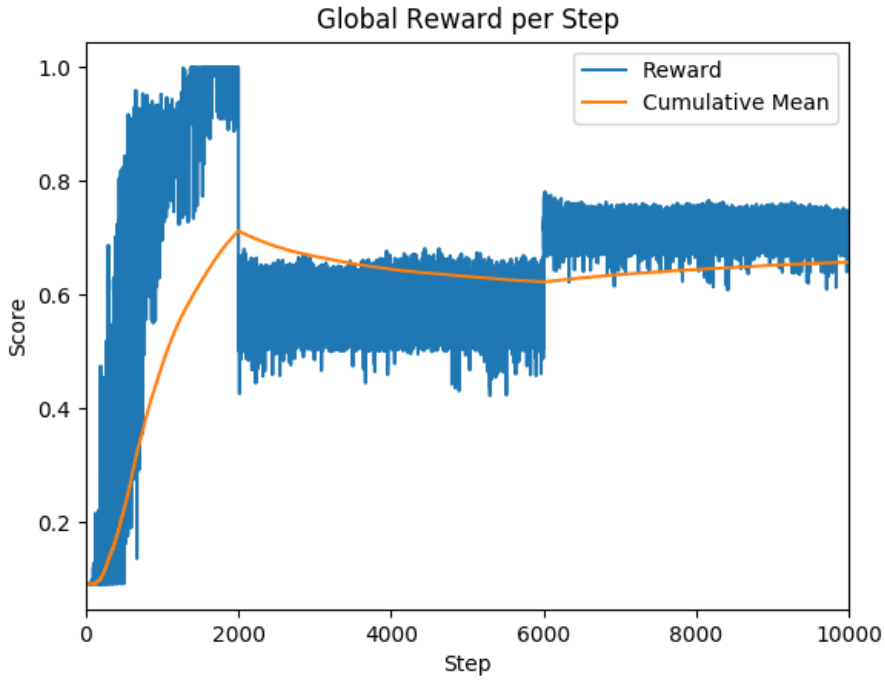

Fig. 12: Score par étape et somme cumulée des scores - Scénario "Apprentissage" - Fonction de récompense "Adaptabilité 3". Comme précédemment, les scores changent lors du changement de fonction, mais l'agent semble ne pas s'adapter à la deuxième fonction (étapes de 2000 à 6000). Le score est légèrement meilleur en utilisant la troisième fonction, mais ne semble pas progresser au fil du temps. L'agent ne semble donc pas capable de s'adapter dans cette situation.

J Valeur d'équité

Fonction de récompense	Moyenne de l'équité
Équité	0.99
Équité 2	0.99
Surconsommation	0.26
Multi-Objectif Somme	0.98
Multi-Objectif Produit	0.90
Adaptabilité 1	0.98
Adaptabilité 2	0.99
Adaptabilité 3	0.99

Table 7: Moyenne de la propriété d'équité sur 10 000 étapes, pour chaque fonction de récompense, dans le scénario "Apprentissage". La valeur est proche de 1 dans presque tous les cas, à part "surconsommation", dans lequel on peut considérer que l'équité n'a pas été apprise. La plupart des fonctions permettent donc d'apprendre un comportement éthique (du moins la valeur d'équité).

K Évolution de l'équité - Fonction de récompense "équité"

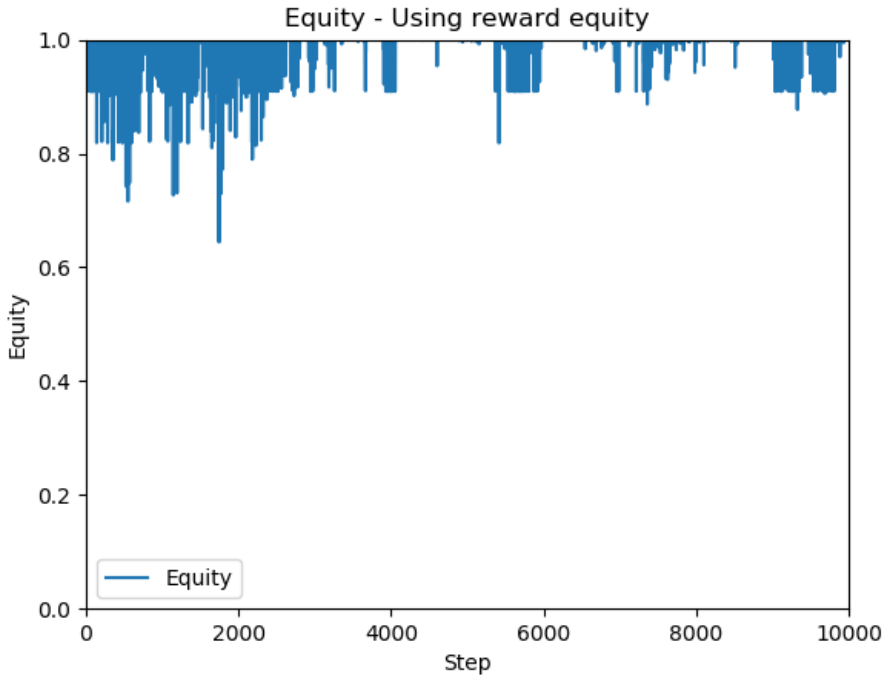

Fig. 13: Évolution de l'équité au fil des étapes - Fonction de récompense "équité". La valeur est, en toute logique, proche de 1 dans la plupart du temps.

L Évolution de l'équité - Fonction de récompense "équité 2"

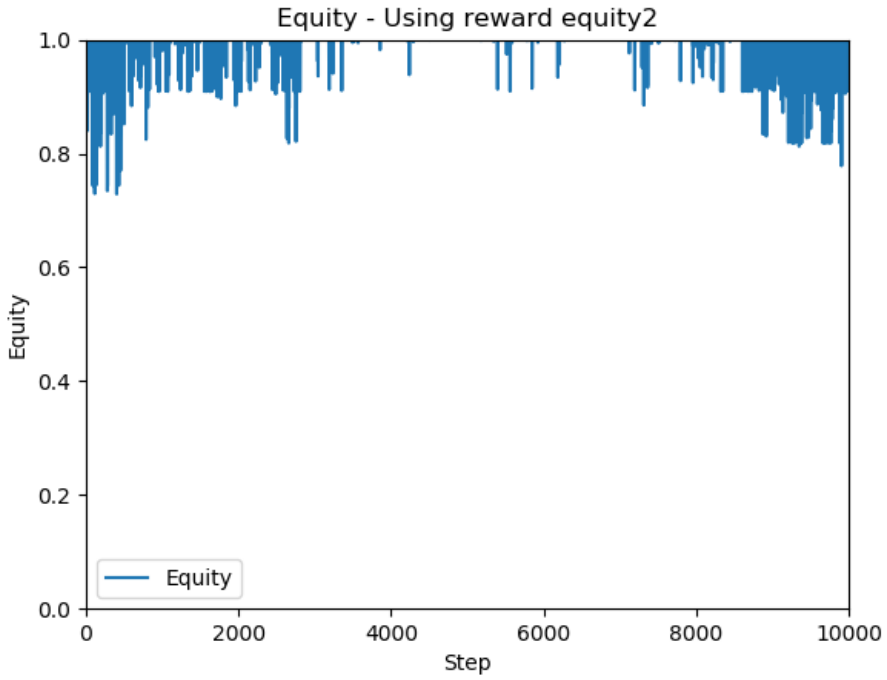

Fig. 14: Évolution de l'équité au fil des étapes - Fonction de récompense "équité 2". La valeur est, en toute logique, proche de 1 dans la plupart du temps.

M Évolution de l'équité - Fonction de récompense "surconsommation"

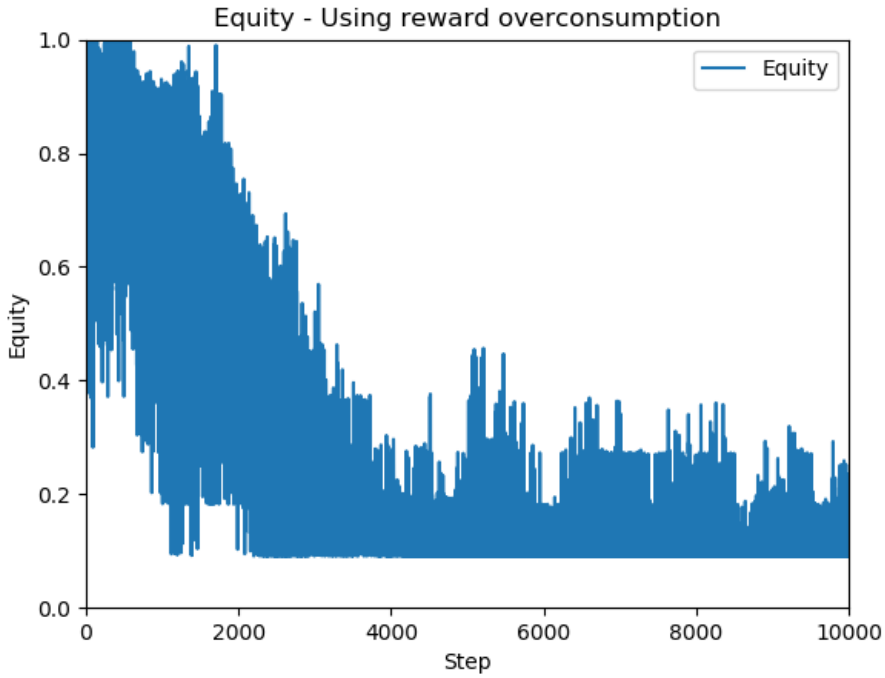

Fig. 15: Évolution de l'équité au fil des étapes - Fonction de récompense "surconsommation". La valeur baisse au fil des étapes, donc il semblerait qu'apprendre à réduire la surconsommation ne permette pas d'apprendre l'équité. On peut supposer qu'à chaque pas de temps, un agent consomme et tous les autres restreignent leur consommation.

N Évolution de l'équité - Fonction de récompense "multi-objectif somme"

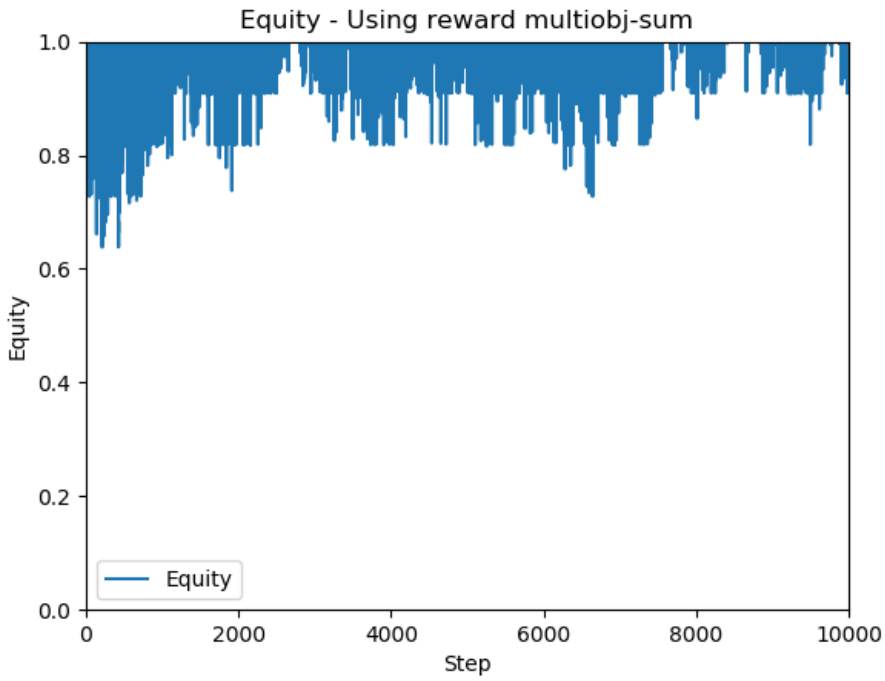

Fig. 16: Évolution de l'équité au fil des étapes - Fonction de récompense "multi-objectif somme". La valeur est généralement entre 0.8 et 1, mais plus souvent proche de 0.8 que dans le cas "équité" ou "équité 2", ce qui semble logique puisque nous n'apprenons pas directement la valeur d'équité dans ce cas. À partir de 8 000 étapes, la valeur semble un peu plus souvent proche de 1, mais cela n'est pas très significatif.

O Évolution de l'équité - Fonction de récompense "multi-objectif produit"

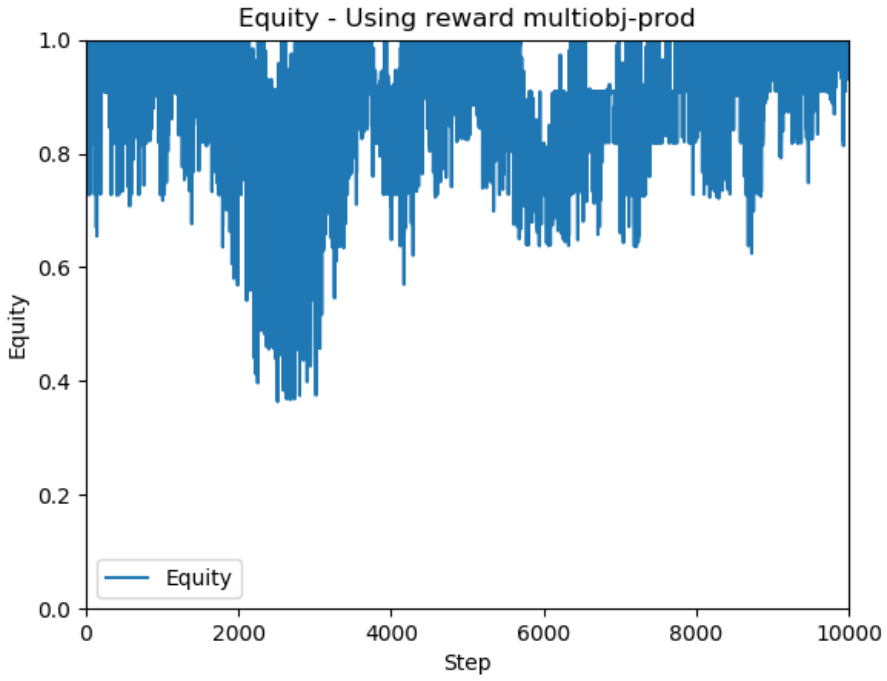

Fig. 17: Évolution de l'équité au fil des étapes - Fonction de récompense "multi-objectif mop". La valeur est varie beaucoup, descend jusqu'à 0.4 puis remonte ; l'apprentissage de la valeur d'équité semble donc moins présent dans ce cas.

P Évolution de l'équité - Fonction de récompense "adaptabilité 1,2,3"

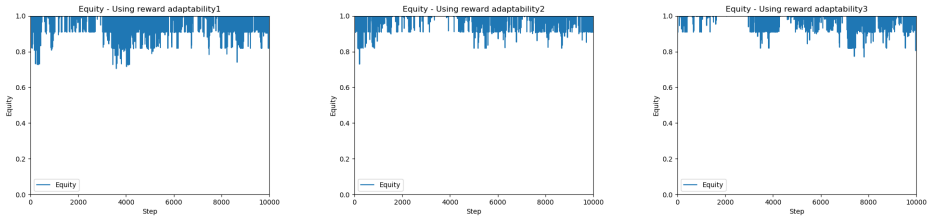

Fig. 18: Évolution de l'équité au fil des étapes - Fonction de récompense "adaptabilité 1, 2 et 3". L'évolution de la valeur est similaire dans les 3 cas : proche de 1, mais légèrement moins que dans le cas de "équité" et "équité 2". Cela semble logique, car les fonctions "adaptabilité" intègrent *en partie* l'équité, alors que les deux autres sont basées dessus. L'apprentissage de l'équité dans ce cas nous semble donc normal.