# RAPPORT INTERMÉDIAIRE DE PFE

*“ Traçage d'empreintes de navigateur en utilisant des techniques avancées d'apprentissage automatique”*

*Tuteurs : Walter Rudametkin / Romain Rouvoy*
*Responsable de projets IMA5 : Thomas Vantroys*

# Sommaire

**Présentation du contexte** **3**
Les bots informatiques 4
Reinforcement learning 5
Cahier des charges du projet 7
**Avancement du projet** **8**
**Travail restant et calendrier prévisionnel** **17**

# Introduction

Dans le cadre du projet de fin d'études (PFE), je réalise un projet au sein de l'INRIA, portant sur le traçage d'empreintes de navigateur en utilisant des techniques avancés d'apprentissage automatique. Le sujet étant vaste, il a été convenu que le projet porterait essentiellement sur les bots informatiques et leur techniques d'évasion.

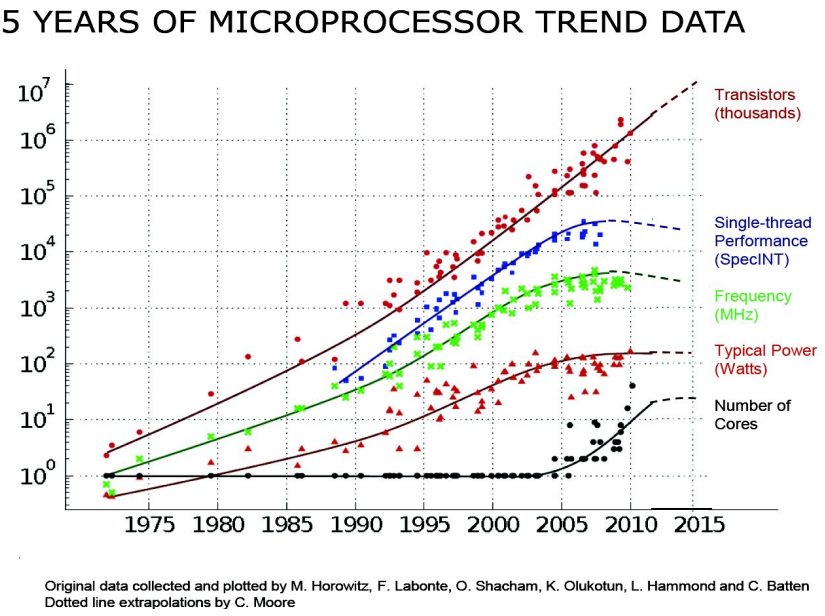
Les bots informatiques sont omniprésents sur le web, et disposent d'autant de points positifs que négatifs. Cependant, les bots considérés comme bons se retrouvent souvent considérés comme mauvais malgré une conception dans un but académique. Ainsi, l'utilisation du machine learning, et plus précisément, du reinforcement learning, qui est une branche du machine learning aux côtés du supervised learning et de l'unsupervised learning, pourra se révéler concluante dans le but de permettre à ces bots de réaliser leurs actions sans se faire blacklister.
Le travail réalisé est ainsi réalisé dans un cadre de recherche, pouvant se révéler être concluante ou non, et est encadré par Mr. Rudametkin et Mr. Rouvoy.

Je présenterai dans un premier temps le contexte du projet, en introduisant les notions principales, pour ensuite décrire l'avancement du projet et finalement terminer sur la présentation du calendrier prévisionnel et des travaux restants.

# I. Présentation du contexte

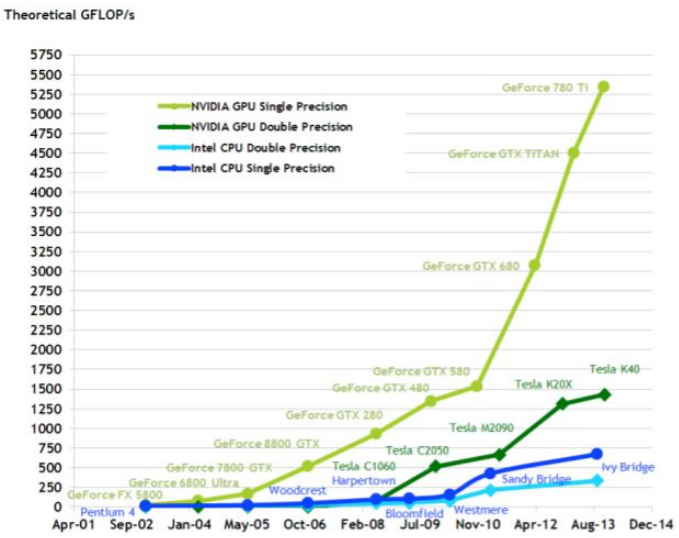
Les dernières années ont mis en relief un mouvement de plus en plus généralisé en raison des avancements constatés dans le domaine. Le secteur du machine learning se retrouve ainsi en position de force auprès du grand public en raison de ses capacités à résoudre des problématiques que les chercheurs considéraient impossible quelques années auparavant.

Le domaine de la reconnaissance faciale, la segmentation d'image, le "speech-to-text", la traduction instantanée, sont tout autant des domaines que la recherche considérait bloqués avant la fin de la première décennie des années 2000. Tous ces domaines ont observé un avancement sans précédent dès lors du début des années 2010 en raison de percées dans le domaine du *machine learning*. Ces avancements peuvent être mis au crédit d'une multiplication constante de la puissance de calcul des ordinateurs personnels, suivant la loi de Moore. Le graphique suivant nous permet d'obtenir une idée du taux d'augmentation de la puissance de calcul des différents processeurs en observant le nombre de transistors composant le micro-processeur.




*http://www-ljk.imag.fr/membres/Jean-Baptiste.Keck/teachings/intro_coprocesseurs.pdf*

Ces avancées n'auraient également pas été possibles sans la démocratisation des GPUs dans les ordinateurs pour particuliers, et notamment les ordinateurs portables, qui disposent, en 2018, pour la majorité d'entre eux d'un GPU dédié, dont la puissance de calcul est suffisante pour permettre le développement et l'entraînement de modèles utilisés dans le deep learning, qui carbure les différents secteurs qui parlent au grand public, tels que la traduction instantanée, qui s'inscrit dans le sous domaine du NLP (*Natural Language Processing*). Les GPUs permettent ainsi, dans le domaine du machine learning, d'améliorer très considérablement les performances, et de diminuer le temps d'exécution, qui était l'un des principaux obstacles à l'avancée du machine learning.




*https://www.researchgate.net/figure/1-Floating-Point-Operations-per-Second-for-the-CPU-and-GPU-source-CUDA-Programming_fig1_268216668*

Le domaine du *machine learning* dispose à ce jour de trois branches :

- *Supervised Learning*, qui utilise des données préalablement traitées par un humain afin de performer de la classification ou de la régression.
- *Unsupervised Learning*, qui utilise des données non labellisées, et sollicite ainsi l'algorithme utilisé pour pouvoir isoler des features des données utilisées.
- Le *reinforcement learning* : c'est le domaine le moins connu du *machine learning*, et celui qui nous intéresse dans le cadre de notre projet. Le reinforcement learning permet un apprentissage sans la nécessité de données.

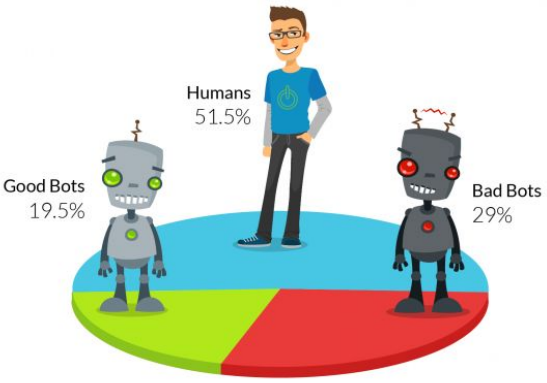
Le succès du *machine learning* est également à mettre au crédit de la multiplication et l'abondance du nombre de données sur internet. Cependant, à ce jour, bien que le web regorge d'informations pouvant être utilisées pour améliorer les performances de modèles actuels, l'obtention de ces données à la main se révèle bien trop lente et bien trop impratique. Les bots informatiques permettent de résoudre ce problème en automatisant la récupération de ces données et en multipliant le taux de données récupérées. De nombreux sites web bloquent néanmoins les bots informatiques, empêchant les chercheurs d'avoir accès aux données nécessaires.

## a. Les bots informatiques

Les bots informatiques sont des scripts écrits dans le but de réaliser des actions prédéfinies sur des pages web sans interaction humaine. Des exemples d'applications de bots possible sont :

- le *web scraping*, qui consiste au parcours de pages internet afin d'en récupérer des données ciblées.
- Les *chatbots*, qui permettent de mettre en place une communication avec les utilisateurs, avec une interaction préalablement définie. Par exemple, des chatbots existent sous facebook et twitter, et permettent de faire la publicité de produits.
- Les *bots sociaux* regroupent les chatbots mais implémentent également d'autres aspects, tels que la mimique des comportements humains afin de se faire passer pour un utilisateur lambda sur un réseau social.

Les *bots* informatiques sont beaucoup plus présents sur internet que l'on ne peut le penser : ils représentent près de 20% du trafic internet pour les bots non malicieux, et près de 30% pour les bots représentants une menace. Cela représente près de 50% du trafic internet total d'après Incapsula[1].




*https://www.incapsula.com/blog/bot-traffic-report-2015.html*

Les bons bots informatiques se contentent de récupérer des informations publiques, et respectent à la lettre le fichier robot.txt, qui spécifie les droits et restrictions des bots sur le site internet en question. La question se pose cependant concernant le respect de la confidentialité : bien que les données soient publiques, est-il moral de récupérer et de stocker chez un particulier des informations qui ont été avancées pour un site web en particulier ? D'un autre côté, la présence de bots informatique entraine nécessairement la mise en place de bots malicieux. Des exemples de ce type de bots sont les *spambots*, qui se chargent de récupérer des listes d'emails dans le but de procéder au *spamming*. Les attaques DDoS sont également réalisés par des bots qui se chargent de réaliser une multitude d'attaque afin de saturer le serveur hébergeant le site internet. Les botnets sont également en vogue en ce moment, et consistent à prendre le contrôle de plusieurs serveurs afin de mener des attaques ciblées.

Les bots informatiques sont implémentés de plusieurs façosn. Ils peuvent soit être écrits de zéro, en utilisant les librairies de *networking* natifs du langage utilisé, soit implémentés en utilisant des librairies écrites dans ce but. CasperJS et PhantomJs sont deux librairies permettant de mettre en place des *bots* informatiques. Leur site web les décrit comme des utilitaires de tests automatisés en majorité. Ces deux librairies étaient prédominantes avant 2016, mais laissent depuis peu à peu la main à des librairies plus performantes: en effet, il est trivial de pouvoir détecter des bots basés sur PhantomJS en se basant simplement sur quelques variables internes à la librairie.

Plus récemment, Google a mis en place son utilitaire de tests automatisés basé sur Chrome Headless, qui n'est rien moins qu'une version sans interface graphique de Chromium, contrôlé par la librairie Puppeteer, qui permet de réaliser la très grande majorité des actions qu'un humain puisse réaliser. Chrome Headless est difficilement détectable avec les méthodes classiques car son empreinte est similaire à celle d'un navigateur utilisé par une personne lambda. Cependant, que Chrome Headless soit difficilement détectable ne veut pas dire qu'il n'est pas détectable, car les méthodes de blocage évoluent en continu et de plus en plus de sites web implémentent des mesures de sécurité contre ces nouveaux types de bots.

[1] "2015 Bot Traffic Report: Humans Take Back the Web, Bad ... - Incapsula." 9 déc.. 2015, https://www.incapsula.com/blog/bot-traffic-report-2015.html.

## *b. Reinforcement learning*

Le *reinforcement learning* est une branche du machine learning qui se centre sur l'apprentissage par l'expérience : soit, l'apprentissage en continu, à la manière dont un humain apprend. Considérons cette analogie : lorsqu'un enfant découvre un plat qu'il ne connaît pas, son instinct serait de le goûter. Si le plat est apprécié, l'enfant aura tendance à vouloir choisir le même plat dans le futur, tandis que si le plat n'est pas à ses goûts, l'enfant refusera ce plat.

Le *reinforcement learning* fonctionne selon les même principes : l'enfant correspond à l'agent, le fait de manger ou pas correspond à une action et le fait que l'enfant apprécie ou non le plat correspond à la récompense. Le *reinforcement learning* est basé sur un *Markov Decision Process* (MDP), qui n'est pas nécessairement parfaitement défini. Ce dernier détail fait la force du *reinforcement learning*, car les algorithmes classiques utilisant les MDPs exigent une représentation parfaite de l'environnement où chaque état est Markov, ce qui est difficilement réalisable. Dans le cas où cette condition est réalisée, le nombre d'états est extrêmement élevé et l'agent ne peut ainsi explorer tous ces états dans un temps raisonnable. Les éléments constituant un modèle de reinforcement learning sont les suivants :

- L'agent : l'agent dans notre cas est le bot informatique. Dans d'autres cas, il pourrait représenter une voiture autonome ou un robot quelconque.
- Les états: les états correspondent aux situations dans lesquelles l'agent se retrouve. Dans le cas d'une partie d'échec, chaque état sera représenté par l'allure actuelle du tableau d'échec.
- Les actions: ils correspondent à ce que l'agent décide de faire. Dans notre cas, "visiter un site web" peut être une action. Dans l'exemple du jeu d'échec, le mouvement d'un pion peut être une action.
- Les récompenses : ce sont les retours sur l'action de l'agent. Si l'action est concluante, la récompense est positive, sinon, la récompense est négative. Si un pion se fait avoir dans le jeu d'échec, l'action est considérée comme non-concluante et l'agent est sanctionné.
- La politique : c'est le courant des actions que notre bot va privilégier. Celui que nous voulons maximiser. La politique dans le cas d'une partie d'échec peut être de gagner la partie.
- L'épisode: un épisode correspond à une phase d'apprentissage et de mise à jour des paramètres. Un épisode contient plusieurs étapes.

Les algorithmes utilisés en *reinforcement learning* sont variés, et se divisent en plusieurs catégories :

- Les algorithmes basés sur le *dynamic programming*,
- Les algorithmes basés sur les méthodes de *Monte-Carlo*,
- Les algorithmes basés sur le *temporal difference learning* (TD).
- Les algorithmes basés sur les méthodes du *Policy Gradient*.

Au sein même de ces catégories, nous retrouvons diverses variations, tels que les mêmes algorithmes utilisant des fonctions d'approximation ou non, utilisant des réseaux de neuronnes profond ou non, le fait que l'algorithme utilise les eligibility traces ou non, et bien d'autres variations.

Les algorithmes de *reinforcement learning* ont cependant tous le même but : maximiser le Q-Value (ou le *Value function*), qui correspondent respectivement à l'action value function et le state value function, représentant le retour estimé en partant d'un état particulier (pour le *state value*), en

suivant une politique définie, et en prenant en plus une action particulière dans le cas de l'action value.

Dans notre cas, nous avons sélectionnés plusieurs algorithmes sous lesquels nous pourrions exécuter notre modèle :

- **Q-Learning**[2] : il s'agit d'un algorithme off-policy, soit, qui utilise une politique différente de celle énoncée afin d'améliorer celle-ci. Le but du Q-Learning est d'optimiser la *Q-value*. Le Q-learning s'inscrit dans la lignée des algorithmes basés sur le TD (temporal difference).
- **SARSA**[3] : il s'agit d'une variante du Q-Learning à l'exception que l'algorithme Sarsa est on-policy et se contente ainsi de suivre la politique énoncée.
- **Sarsa(λ)**[4] : il s'agit d'une version améliorée de l'algorithme Sarsa utilisant les eligibility traces, qui permettent d'introduire la notion de temps dans l'algorithme. Les eligibility traces permettent aux actions antérieures d'influer sur les actions futures. La contrainte de cet algorithme est que le nombre de calculs requis est bien plus élevé, car nous devons itérer à travers toutes les Q-values à chaque étape.
- **N-step Sarsa**[5] : Le n-step sarsa permet de mettre à jour les paramètres de l'algorithmes à chaque n étapes, au lieu de chaque fin d'épisode et permet ainsi de généraliser au cours de l'épisode même.
- **Action-Critic**[6] : cet algorithme appartient à la catégorie des policy-gradient, qui va chercher à retrouver la meilleure politique à travers des séquences d'expérimentation. Cet algorithme est généralement le plus performant.

Dans tous les algorithmes précédents, l'exploration est effectué selon une méthode ε-greedy, qui permet de choisir une action au hasard selon une probabilité ε.

## c. Cahier des charges du projet

Le but du projet est ainsi d'utiliser des techniques du *reinforcement learning* afin de permettre à nos bots informatiques de réaliser les actions pour lesquelles ils sont programmés sans se faire bloquer par les sites web. Pour ce faire, le cahier des charges du projet est subdivisé en plusieurs points :

- Recherche et maîtrise de la théorie du reinforcement learning : ce point consiste à se documenter et comprendre l'aspect mathématique et algorithmique derrière le reinforcement learning, afin de pouvoir choisir les algorithmes les plus appropriés et d'implémenter les modèles de manière efficace.
- Mise en place de premiers modèles et implémentation dans un environnement de test dans un premier temps.
- Mise en place l'environnement de test sous OpenAI et observer les performances pour chaque modèle.
- Apprentissage de l'écriture de bots informatiques sous nodeJS en utilisant Puppeteer et Chrome Headless.

---

[2] "Technical Note Q,-Learning." http://www.gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf.
[3] "Q-Learning and SARSA: A Comparison between Two Intelligent ...." 17 juin. 2015, https://www.ssrn.com/abstract=2617630.
[4] "Experimental analysis on Sarsa(lambda) and Q(lambda) with different ...." https://www.researchgate.net/publication/220256799_Experimental_analysis_on_Sarsalambda_and_Qlambda_with_different_eligibility_traces_strategies
[5] "RL — Reinforcement Learning Algorithms Quick Overview - Medium." 29 août. 2018, https://medium.com/@jonathan_hui/rl-reinforcement-learning-algorithms-quick-overview-6bf69736694d.
[6] "Actor-Critic Algorithms." http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_5_actor_critic_pdf.pdf.

- Apprentissage du Javascript et dans une moindre mesure, du Python, dans le cas d'une programmation appliquée à la recherche (numpy, sickit-learn, etc).
- Mise en place d'un environnement réel contrôlant un bot dans un premier temps.
- Extension à plusieurs bots de l'environnement précédent.

OpenAI Gym est un *toolkit* permettant le développement et le test d'algorithmes de *reinforcement learning*. Il a été mis en place dans le but de standardiser les environnements de test, afin de simplifier l'implémentation des algorithmes issues de papiers de recherche.

Il convient également de noter que je ne suis pas familier avec les langages utilisés, ce qui nécessite un temps d'adaptation. Les langages choisis ne peuvent pas être changés, en raison du fait que les librairies avec lesquelles nous travaillons sont implémentés en utilisant ces même langages de programmation.

L'apprentissage du *reinforcement learning* est réalisé en majorité à l'aide de la seconde édition du livre de Sutton & Barto, qui est considéré comme la bible du reinforcement learning. De nombreuses autres ressources sont utilisés, mais dans une moindre mesure.

# II. Avancement du projet

Avant de pouvoir commencer le projet en lui-même, un grand travail de documentation à dû être réalisé. Ayant des connaissances théoriques en *machine learning* au préalable, je disposais ainsi d'une base avant de commencer l'apprentissage du *reinforcement learning*. J'ai pu ainsi, intensivement durant deux semaines, lire le livre sur le *reinforcement learning* de Sutton & Barto[7], la nouvelle édition ayant fait sa sortie dans le milieu de l'année précédente. Le livre est ainsi à jour concernant les avancées récentes du reinforcement learning. J'ai également puisé dans les ressources d'universités américaines, telles que UCL, Berkeley, qui proposaient leur cours gratuitement sous format vidéo ou sous forme de slides. Ainsi, au fur et à mesur, j'ai pu me forger une connaissance du domaine du *reinforcement learning*, en comprenant l'origine théorique des algorithmes utilisés, les subtiles différences entre certains algorithmes, dont uniquement la fonction de mise à jour de la Q-Value change.J'ai cependant dû passer sur quelques sujets que je considérais ne pas être utile pour le projet, tels que le *deep reinforcement learning*, qui utilise des réseaux de neurones profonds afin de représenter les états.

J'ai également dû me documenter sur les *bots* informatiques et leur implémentation. J'ai ainsi pu me pencher en profondeur sur les moyens qui permettent de mettre en place les bots informatiques, qui correspondent aux librairies évoqués précédemment (PhantomJS, Puppeteer…). Nous avons ainsi, après discussion, décidé d'utiliser uniquement Puppeteer dans notre projet, car PhantomJS devenait peu à peu obsolète. Il est cependant possible, après l'obtention des résultats de nos modèles, de tenter de de lancer notre modèle avec comme agent, un contrôleur d'un bot basé sur PhantomJS.
Mes recherches dans ce domaine ont également portés en grande partie sur les manières de bloquer les bots, soit, sur les différentes techniques utilisés par les sites webs actuels :

- **La durée de la session** : une session est définie du moment qu'un agent accède au site internet (détecté par son IP, ou son empreinte) jusqu'à l'inactivité de celui-ci;
- **Le nombre de requêtes réalisés** : un utilisateur normal tend à réaliser un nombre de requête significativement inférieur au nombre de requêtes réalisés par un bot en raison du fait que le bot

[7] "Reinforcement Learning: An Introduction - Stanford University." https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

informatique tend à maximiser les pages explorés dans le but de récupérer le plus de données possibles;

- **Le nombre de requêtes sans la présence du header HTTP Referer** : le header referer contient l'adresse de la page précédente visitée, menant à la page actuelle (par la présence d'un lien). Cependant, cet en-tête n'est pas défini dans le cas où la ressource précédente est représentée par une URI de type data ou si un différent de protocoles de sécurité existe entre les pages(de HTTPS à HTTP). Un pourcentage important de bots tendent à ne pas définir cet en-tête, ou à ne pas le définir correctement. A noter que les humains peuvent également désactiver cet en-tête;
- **Le pourcentage d'images et de ressources CSS chargés par le bot**. Un bot, afin d'optimiser le temps passé sur chaque page, tend à ne charger que l'essentiel de la page, soit uniquement les données textuelles;
- **Pourcentage d'erreurs 4xx** : Les bots tendent à disposer de ce pourcentage plus élevé, en raison du fait qu'un bot tend à tenter d'accéder à des ressources sans en connaître le contenu, et tend à cliquer sur des liens n'existant plus.
- **Demande du robots.txt** : un humain ne demande pas le robots.txt;
- **Pourcentage de requêtes HTTP séquentielles** : les humains tendent à disposer d'un pourcentage plus élevés en raison d'une demande moindre pour des ressources de type script, images ou document, tandis que les bots accèdent à des liens donnant directement sur ce type de ressources et cassent ainsi le parcours "séquentiel";
- **Le fingerprinting** : les empreintes provenant d'une activité douteuse sont sauvegardés. Si une activité similaire est détectée de nouveau, le fingerprinting permet de savoir si elle provient de la même source;
- **Durée entre deux visites de pages au sein d'un même site douteuse**, car soit elle est trop rapide, soit elle est trop régulières pour correspondre à une activité humaine;
- **Adresse IP** : Une utilisation intensive d'une adresse IP disposant des précédents attributs est douteuse;
- **User-agent** : l'user agent d'un utilisateur classique est différent de celui utilisé par chrome headless (sans modification). Ils ressemblent respectivement à :

`Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36` et

`Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/60.0.3112.50 Safari/537.36`

Cette liste n'est ainsi pas fixée et peut être amenée à évoluer fortement tout au long du projet. Ainsi, nous désirons que notre bot informatique puisse éviter de se faire détecter pour les raisons précédentes, en mimant un comportement presque humain tout en gardant un temps d'exécution qui soit élevé.

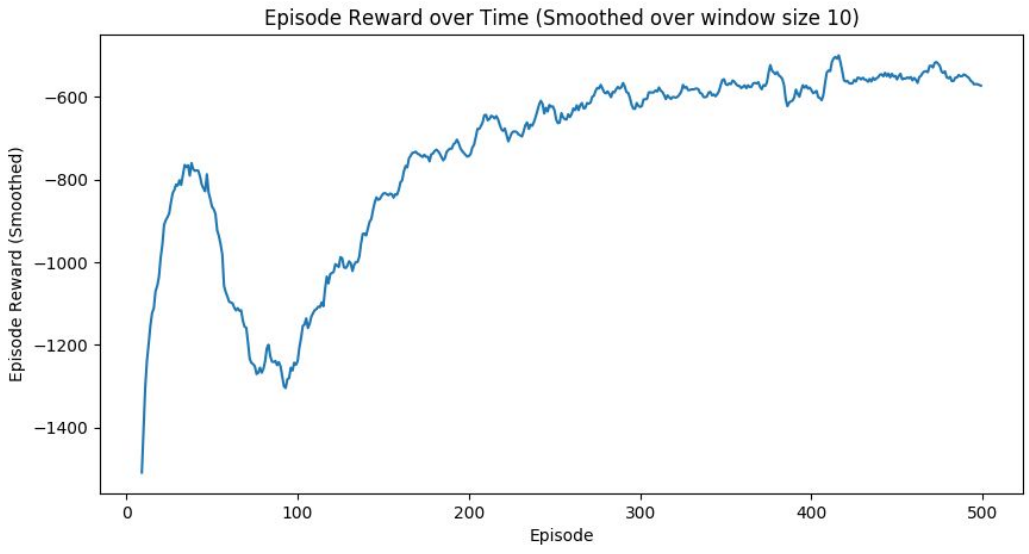
Ayant ainsi identifié les contraintes du projet, et ayant reprise la théorie du *reinforcement learning*, j'étais ainsi prêt à me mettre à définir mon premier modèle.
Après discussions, nous avons défini le modèle suivant :

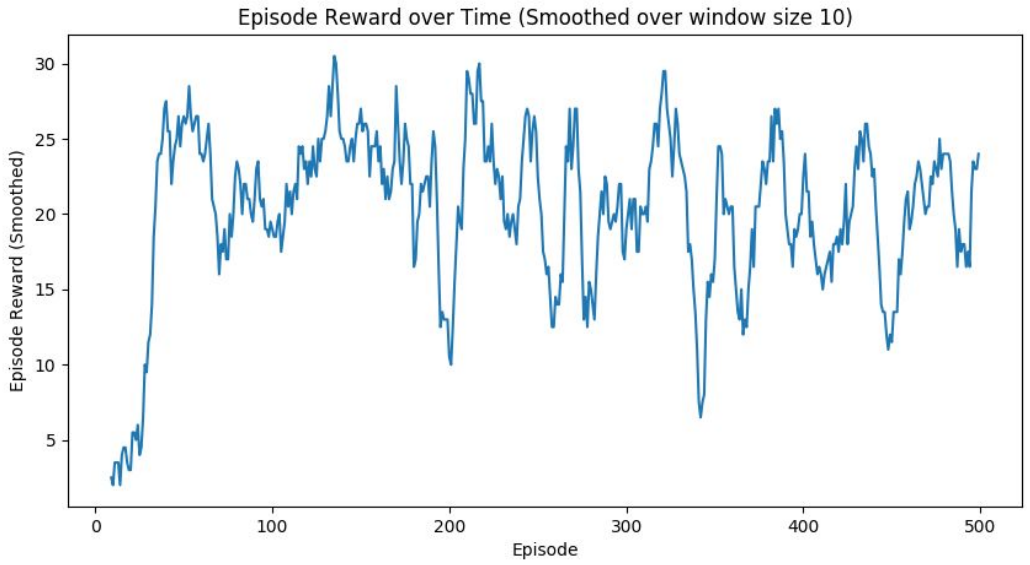
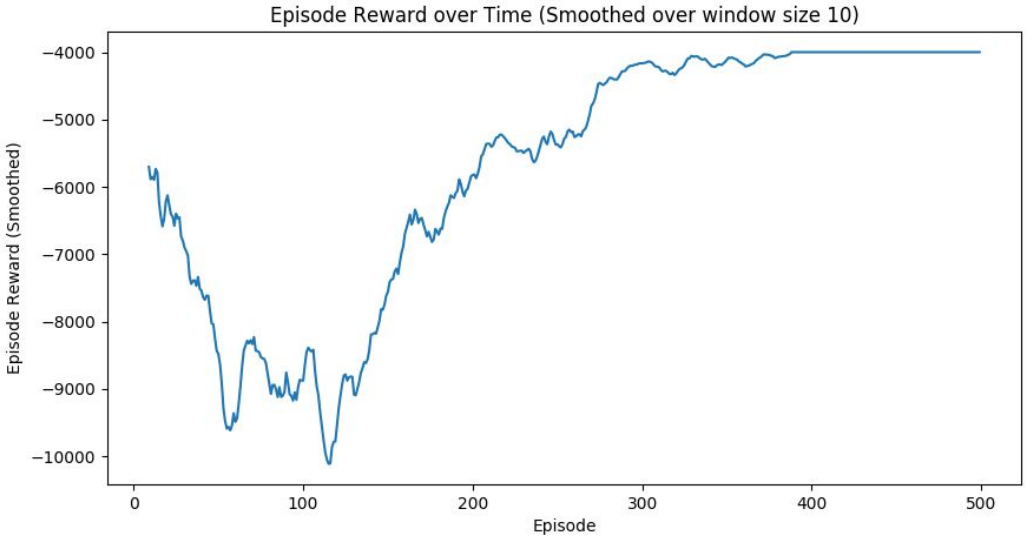
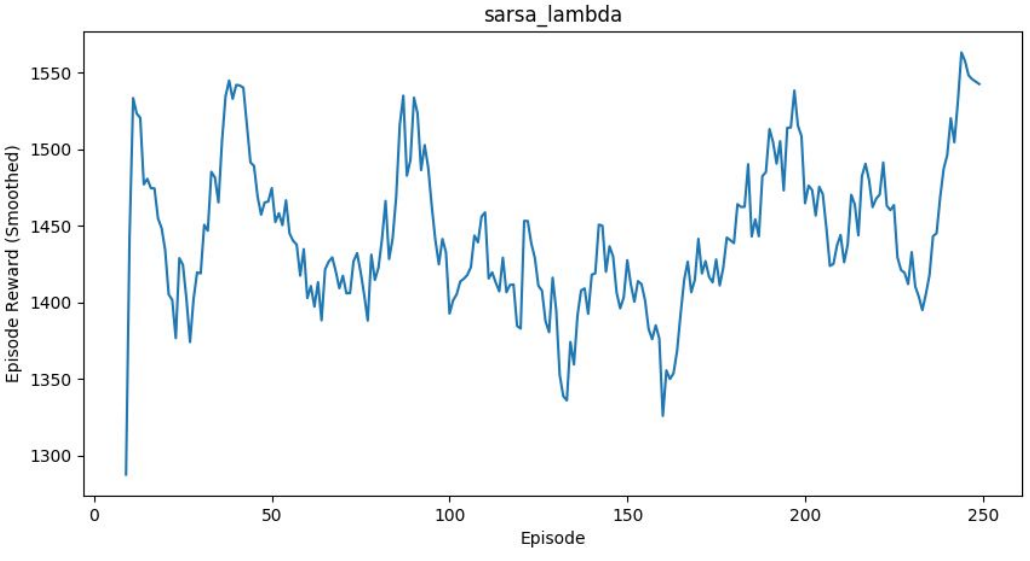
- **États** : les états sont définis par les features exploitables des site webs. Dans notre cas, et dans un premier temps, nous proposons d'utiliser uniquement le fait que le site bloque les bots ou non, le fait que le site dispose d'un security provider ou non, le fait que le site web ait été visité un certain nombre de fois, ou qu'un security provider ait été visité un certain nombre de fois, et le fait qu'il utilise le fingerprinting ou non.
- **Actions** : les actions sont définis par la visite d'un état particulier, le changement de l'adresse IP et le changement de l'User-Agent.
- **L'agent** correspond à un contrôleur du bot. Il donne l'ordre de lancer le crawl ou non.

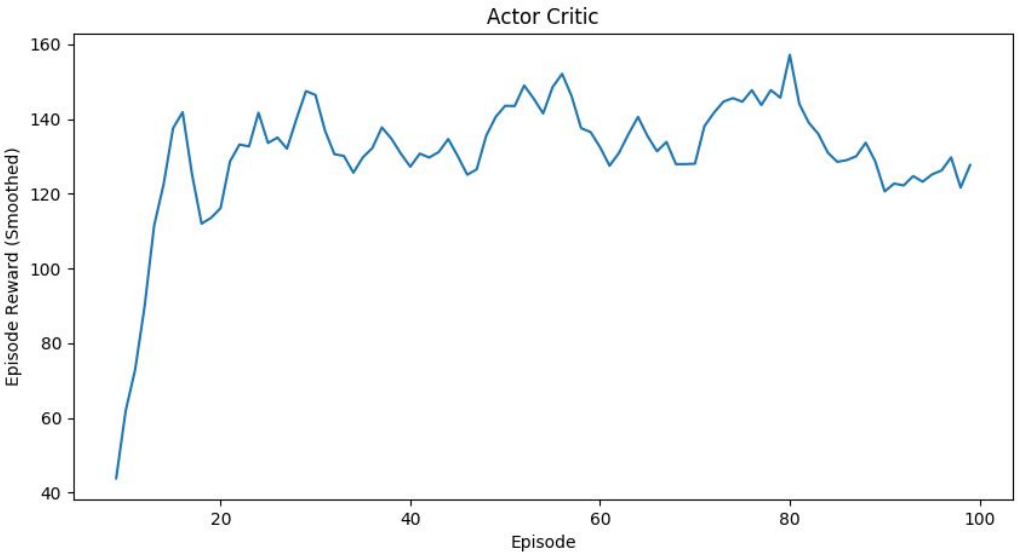
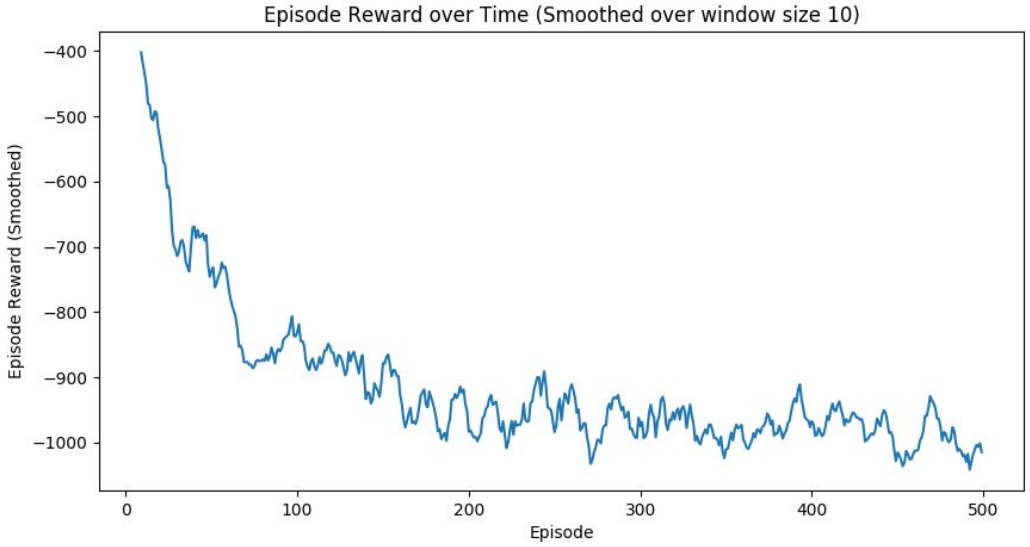
- **Récompenses** : Le bot reçoit une récompense négative dans le cas où le site web le détecte et le bloque, une récompense négative très faible dans le cas où l'état ne dispose d'aucun site web à crawler, et une récompense positive dans le cas d'un crawl à succès

Dans ce modèle, un épisode correspond au *crawl* d'un nombre de sites web prédéfini. Les sites web sont mis à jour à chaque étapes afin d'être attribués à chaque état, en fonction de leur taux de visites et du taux de visites de leur security provider. Un *security provider* est simplement une entreprise s'occupant de la sécurité de plusieurs sites webs.

Ainsi, dans un premier temps, nous décidons de tester tous les modèles qui nous paraîtrons intéressants dans un environnement de test que nous programmerons. Cette décision est motivée par la volonté de ne pas blacklister nos IPs par les sites webs avant la mise en place d'un modèle stable, et une nécessité d'avoir un environnement s'exécutant rapidement : un crawl simulé d'un site web est quasi-instantané tandis qu'un crawl réel est lent et dépend de plusieurs facteurs. Pour cela, nous programmons notre environnement en tant qu'environnement OpenAI, en utilisant le langage Python. Il a ainsi fallu revoir la documentation d'OpenAI afin de savoir comment implémenter un environnement sous ce framework. Il m'a également fallu me familiariser avec les librairies scientifiques de Python tel que numpy, mais aussi la syntaxe du langage. Après une dizaine de jours, j'ai pu tester mon modèle dans l'environnement de test : les sites web bloquent le bot de façon probabiliste en fonction des caractéristiques qui leurs ont été attribués. Les security providers bloquent également en fonction du nombre de visites et de leur efficacité, qui est représentée par une note attribuée (de 0 à 5). Les UA et les IPs entraînent une plus grande chance de blocage dans le cas où ils sont récurrents et suivis dans le temps. Les résultats sont les suivants (à noter que nous avons dû adapter les récompenses en fonction des algorithmes):

| **Algorithme** | **Rewards** |
|---|---|
| SARSA |  |

| | |
|---|---|
| Q-Learning |  |
| n-step SARSA (avec 8000 étapes par épisode) |  |
| Sarsa(λ) |  |

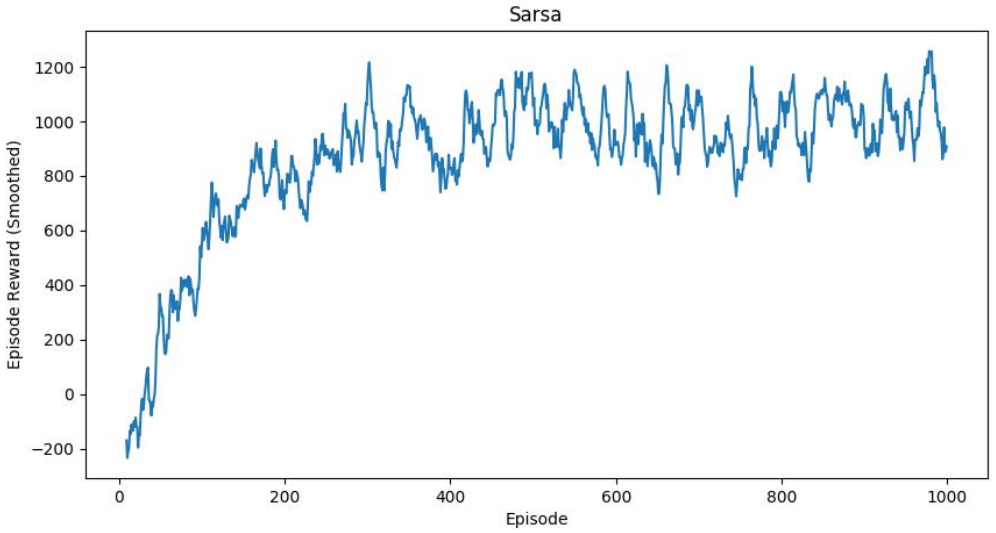
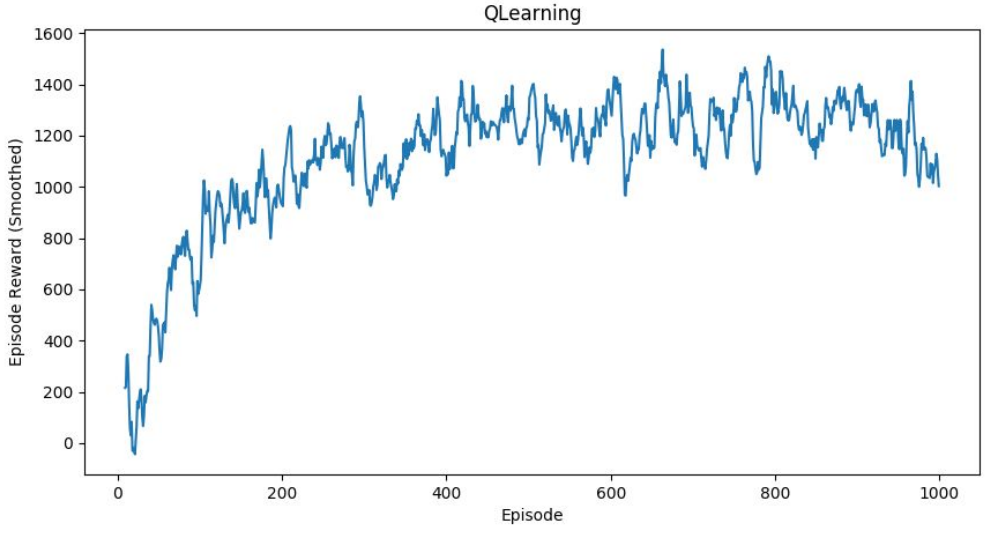
| | |
|---|---|
| Actor-Critic |  |
| hasard (pour comparaison) |  |

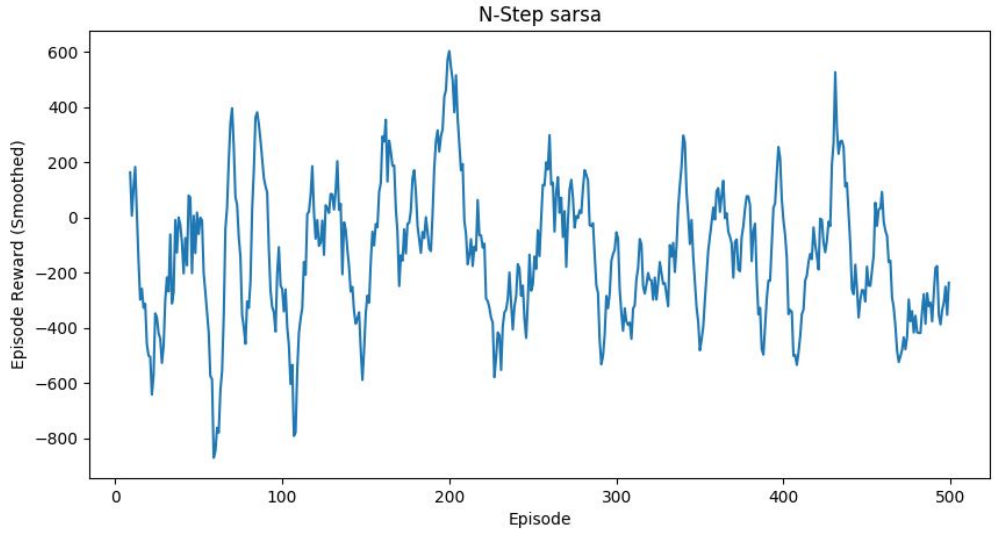
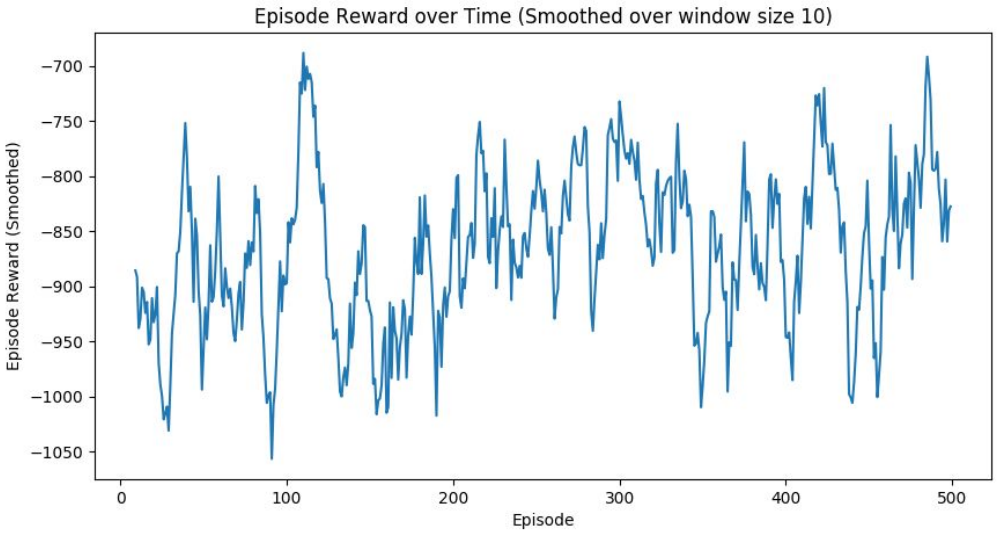
A partir de ce premier modèle, nous avons pu observer que les résultats sont concluants dans un environnement de test, cependant, nous n'atteignons pas un taux de crawl avec succés qui est suffisant : nous tournons aux alentours des 70% de réussite, ce qui est assez faible. Ainsi, il a été question de mettre en place un nouveau modèle, afin de pouvoir comparer avec le précédent. Le nouveau modèle dispose de la représentation suivante :

- **Les états représentent la configuration du bot**, soit, l'user-agent, le proxy utilisé, l'utilisation du header REFERER, l'utilisation du header HEAD, le taux de chargement d'images et le taux de chargement des ressources Css et documents. Le nombre d'états est donc très élevé ( de l'ordre de 10^6) pour ce modèle car par exemple, pour chaque nouvel user-agent ajouté, le nombre d'états est doublé.
- **Les actions** sont représentés par le changement d'user-agent, le changement de proxy, le changement de referer et du header Head, le passage d'un site web à un autre, et l'augmentation ou la diminution du taux de chargement d'images.

- **Les épisodes** sont toujours représentés de la même manière, à savoir, un nombre d'étapes fixé par épisode.
- **Les récompenses** ne changement pas également, à l'exception du fait que nous ne pouvons plus avoir d'états qui ne permettent pas de ne visiter aucun site web, ce qui fait que la récompense négative sur la visite d'un état vide est supprimée.

Les résultats pour ce modèle sont les suivants :

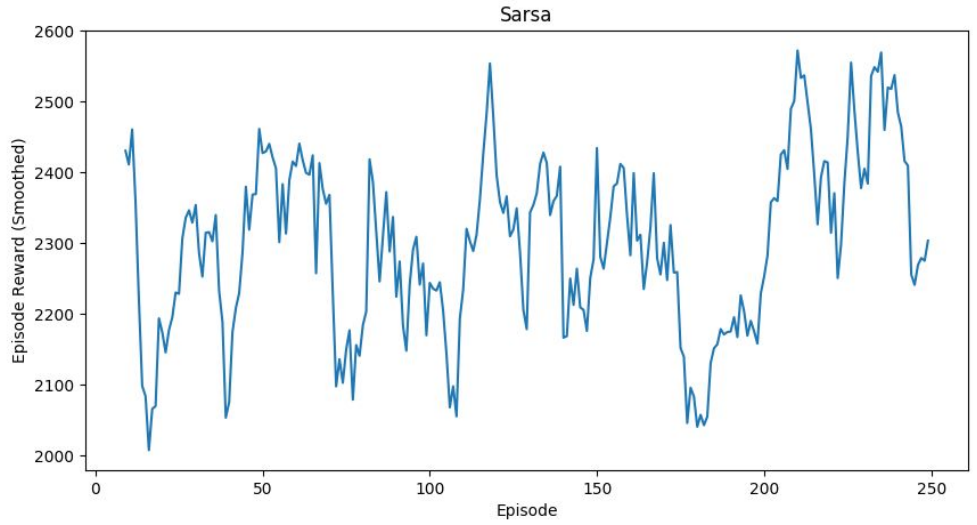
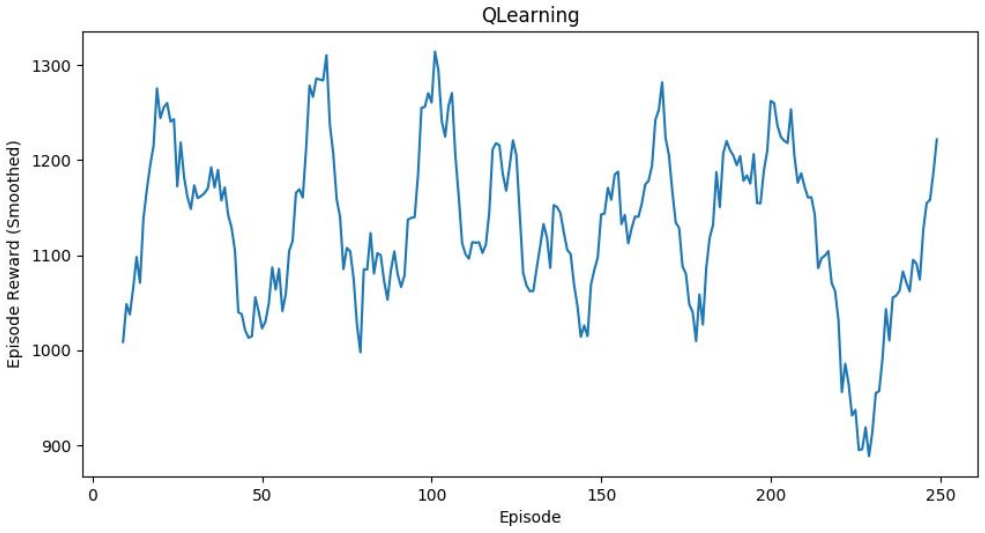
| **Algorithme** | **Rewards** |
|---|---|
| SARSA |  |
| Q-Learning |  |

| | |
|---|---|
| n-step Sarsa |  |
| Random |  |

Nous pouvons observer que ce modèle ne bénéficie pas de la mise à jour des paramètres en plein épisode et fonctionne mieux avec des algorithmes classiques qui ne se mettent à jour qu'en fin d'épisodes. Ainsi, après discussions, nous avons convenus qu'il était peut-être bénéfique de tenter de récupérer le meilleur des deux mondes. Nous mettons ainsi en place un modèle qui utilise des features du site web mais également des caractéristiques du bot :

- **Les etats** : prise en compte des caractéristiques du site web introduits dans le premier modèle, en supprimant le security provider, et en ajoutant le taux d'utilisation d'un user-agent particulier et d'un proxy particulier.
- **Les actions** : consistent à changer d'user-agent, de proxy, de site web, ou de ne rien faire (les actions peuvent être combinés également).
- **Les épisodes:** ils ne changent pas et les récompenses non plus.

Les résultats pour le Q-learning et Sarsa ne se révèlent pas prometteurs :

| **Algorithme** | **Rewards** |
|---|---|
| SARSA |  |
| Q-Learning |  |





Malgrés d'autres tentatives, en jouant sur les hyper-paramètres des algorithmes, tels que le taux d'apprentissage, le discount factor, ainsi que le nombre d'étapes par épisodes, les résultats ne s'améliorent pas. Il convient de noter que nous sommes dans un environnement de test durant toutes les exécutions de ces modèles, soit, un modèle qui s'exécute mal dans un environnement de test peut très bien faire ses preuves dans un environnement réel.

Nous décidons ainsi de passer à l'écriture du code d'exécution en environnement réel. Le langage étant le Javascript sous NodeJS, et n'ayant aucune notion en ce langage, la mise en place fut plutôt longue (de la semaine 7/8 à la semaine 10). J'ai eu quelque difficulté à maîtriser la concurrence en Javascript et les nombreux modules externes que j'utilise. La difficulté résidait également dans le fait que nous ne simulons plus : les différents changements que le bot subit étaient entièrement implémentés, certains étant plus complexes que d'autres, et nécessitent de la documentation. Le modèle retenu fut le suivant :

- Les états répondent aux questions suivantes :

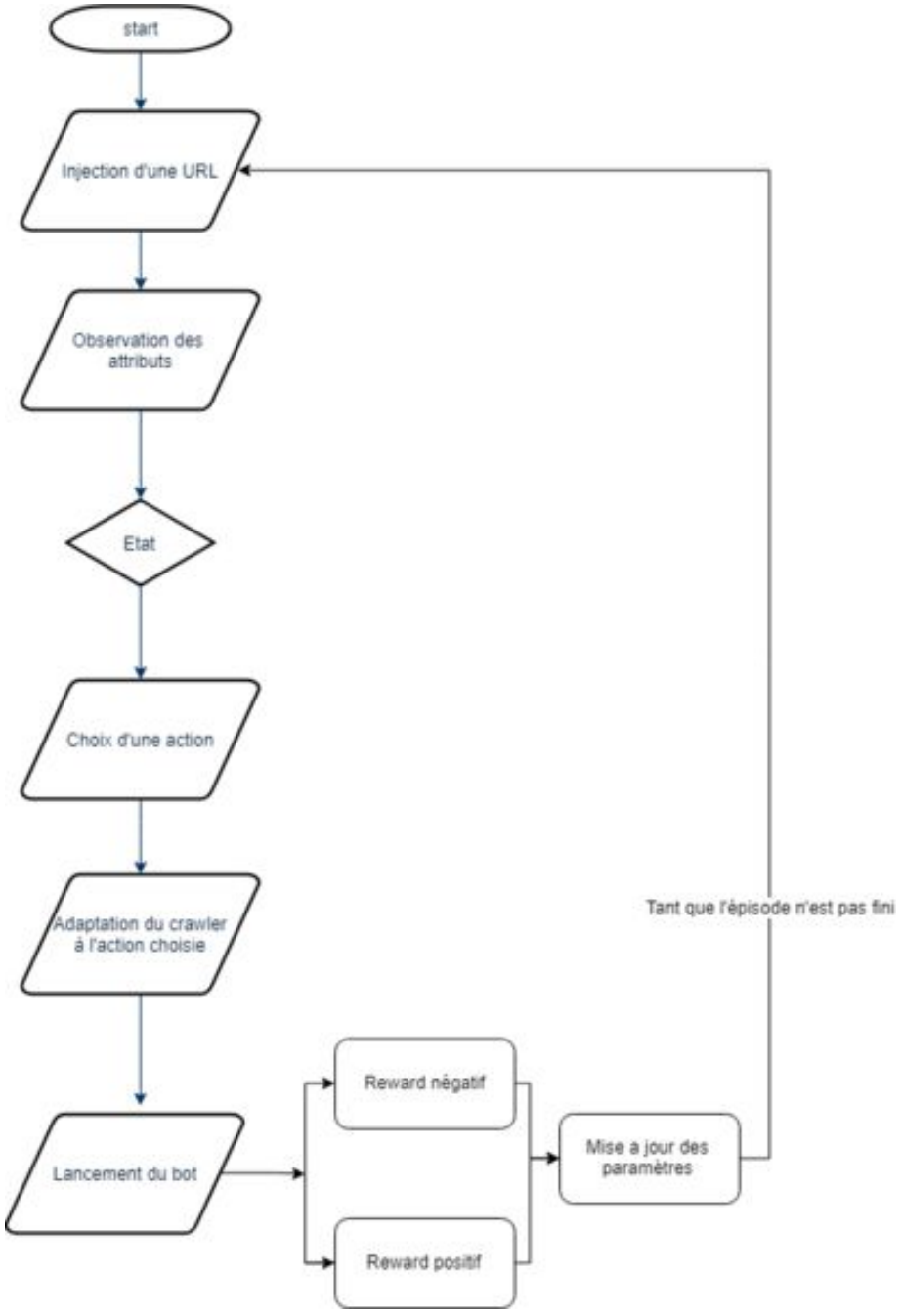
    - Le site vérifie-t-il le taux de chargement de documents ?
    - Le site utilise-t-il le fingerprinting ?
    - Le site vérifie-t-il la taille de l'écran ?
    - Nombre de visites du site ?
    - Le site bloque-t-il les bots ?
    - Le site vérifie-t-il les plugins ?
    - Le site analyse le pattern de la souris/clavier ?
    - Le site vérifie les IPs/UAs ?
    - Nombre de fois que notre IP à été utilisée
    - Nombre de fois que notre UA à été utilisée
- Les actions consistent en :
    - Changement d'UA/IP (présence de plusieurs bots dans une prochaine version)
    - Utilisation d'un proxy
    - Augmentation/Diminution de taux de chargement de ressources non-nécessaires
    - Changement de la taille de la fenêtre
    - Utilisation de plugins
- Les épisodes présentent un changement : un épisode correspond à un crawl d'une série de page appartenant à un seul et même domaine.

Les états sont peuplés grâce à des données recueillis par Antoine Vastel, qui travaille sur un projet nécessitant ce genre d'information. Il a ainsi pu partager avec moi sa base de donnée, sous MongoDb. Je récupère ainsi les informations pour peupler les caractéristiques des sites web ainsi que des états à travers la base de données.

Afin d'éviter des taux de chargements trop long, je sérialise ces données qui ne changent que rarement, tels que la représentation des sites webs et les récupère à chaque utilisation. Cette fonction de sérialisation est toujours en cours d'écriture à ce jour.

Le bot est ainsi implémenté sous Puppeteer, et nous réécrivons les fonctions internes au navigateur intégré (Chromium) qui retournent des informations sur l'user-agent, les plugins, le taux de chargement des documents, et autres.

Le processus du programme principal est le suivant :




*Processus d'exécution du programme en conditions réelles*

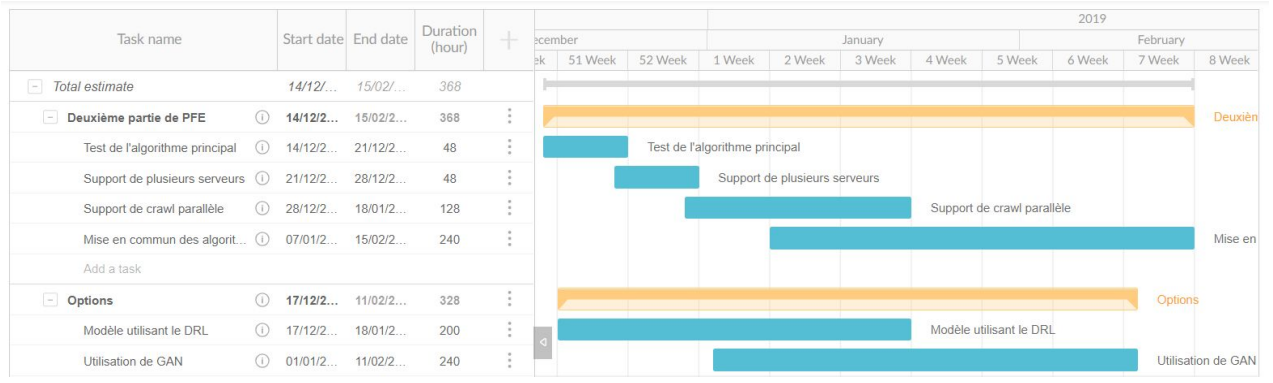
L'écriture du programme principal est terminé à 95% à la semaine 10, ne manquant que la fonction de sérialisation et désérialisation avant de pouvoir lancer le programme.

# III. Travail restant et calendrier prévisionnel

Pour le reste du projet, il faudrait pouvoir tester le programme mis en place en environnement réel, analyser les données recueillis, et procéder à des améliorations du modèle. Le modèle mis en place n'est pas encore très complet et ne prend pas en compte plusieurs causes de blocage : c'est volontaire, dans un soucis de simplification du problème, avant de le complexifier. Il faudrait ainsi ajouter certaines caractéristiques exploitables des sites web et du bot et observer les résultats.
Après avoir perfectionné le modèle principal, il me faudra développer une API permettant de lancer les bots à partir de serveurs différents afin de pouvoir obtenir une adresse IP différente pour chaque bot.
Il me faudra également trouver un moyen de lancer plusieurs bots en parallèle, et de profiter des rewards des bots ayant déjà terminés leurs crawl.

Nous avons également discuté de la possibilité d'utiliser le *deep reinforcement learning*, pour observer une image de la page à visiter, et ainsi performer des actions en fonction des observations graphiques. Cette technique peut être utile pour déterminer quels sont les bons boutons sur lesquels cliquer, et faire en sorte de varier les motifs de mouvements de la souris, qui peut être une action déterminante dans l'évasion des bots.
La possibilité d'utiliser un *Generative Adversarial Network* est également discutée pour implémenter cette fonctionnalité.

Le diagramme GANTT suivant permet d'organiser les travaux restants sur la période restante :




*GANTT de deuxième partie du PFE*

# Conclusion

J’ai ainsi pu consacrer cette première partie de projet à tester différents modèles et en évaluer les performances. J’ai donc dû réaliser un travail de recherche continu, me permettant d’en apprendre toujours plus sur le reinforcement learning et les bots informatiques.
La sélection du meilleur modèle à exécuter dans un environnement réel à été réalisée sur la base des résultats en environnement de tests, et son implémentation est en cours.
La deuxième partie du projet permettra de continuer sur cette lancée, d’évaluer les paramètres en environnement réel et ainsi de déterminer des améliorations. Elle permettra également, dans le mesure ou le temps le permet, l’implémentation de plusieurs idées plus complexes, mais pouvant améliorer les performances du modèle.