Université de Mons
Faculté des Sciences
Département des mathématiques

Rapport du projet de méthode formelle

Blackjack sur Gymnasium

Professeur :
Mickaël RANDOUR
Pierre VANDENHOVE

Auteurs :
Loïc DUPONT

Année académique 2022-2023

Table des matières

1 Introduction.

Dans le cadre du projet de méthode formelle, il a été demandé d'analyser et de pouvoir créer un agent capable de jouer au jeu du Blackjack de la librairie Gymnasium. Celui-ci devra avoir un bon taux de réussite dans le jeu (c'est-à-dire avoir un bon taux de partie gagnante par rapport à la limite théorique que nous laisse le jeu). En effet, si notre agent affiche un taux de 15% de réussite alors que la limite se situe vers les 45%, nous pourrons en déduire que cet agent n'est pas performant.

Les prochaines sections concerneront les différentes manières envisagées pour à la fois calculer cette limite théorique mais aussi dans la confection de notre agent. Enfin, une dernière section discutera sur les problèmes rencontrés mais également différentes pistes pour améliorer nos agents.

2 Découverte de l'environnement.

Avant de se lancer dans la confection de notre agent ou d'un quelconque calcul, regardons l'environnement avec lequel nous travaillons. L'espace où nous travaillons est une représentation du jeu du Blackjack. Celui-ci possède 3 observations discrètes qui sont la somme de notre joueur, la carte visible du croupier et si notre joueur possède un as dont il peut baisser la valeur (c'est-à-dire changer la valeur de 11 à 1). Ensuite, nous avons 2 actions discrètes possibles qui sont de piocher une nouvelle carte ou de s'arrêter. Le but du jeu est d'être le plus proche de 21 par rapport au croupier sans pour autant dépasser le 21. On dit que le joueur/croupier fait un blackjack si la somme de ses cartes forme 21. Le jeu se déroule avec le joueur qui commence. Celui-ci a le choix de piocher des cartes (autant qu'il le souhaite) ou de s'arrêter avec les cartes qu'il possède à ce moment-là. S'il a dépassé 21, il a automatiquement perdu, par contre, dans le cas contraire, c'est au tour du croupier de jouer et il doit continuer de piocher jusqu'à ce qu'il obtienne au minimum 17. S'il dépasse à son tour le 21, le croupier perd mais s'il se retrouve entre 17 et 21, alors on regarde la personne qui possède la plus grande somme et il sera déclaré gagnant.

Avec ces explications sur le déroulement d'une partie du jeu, nous pouvons donc déjà tirer plusieurs points sur des états initiaux et aussi finaux de notre jeu. Tout d'abord, le fait que le joueur peut débuter une partie avec comme minimum 4 car 2 est la plus petite valeur du jeu et nous avons 2 cartes au début de la partie mais aussi le maximum 21 car on peut avoir un as (qui vaut 11 points) et une carte valant 10 points comme le 10 ou l'une des 3 cartes imagées (valet, dame ou roi). Le croupier va démarrer de la même manière que le joueur sauf que l'on pourra observer que sa première carte. Du point de vue du joueur, il ne pourra alors se situer au départ qu'entre 2 et 11. Ceci liste donc les états qui seront les points de départ de notre modèle. Concernant les états finaux, nous aurons donc toutes les situations où le joueur dépasse 21 (quelle que soit la valeur de la carte visible du croupier) mais aussi toutes les situations où le croupier dépasse 21 (de manière similaire avec la somme du joueur). De plus, avec ces états finaux viennent s'ajouter tous les états où le croupier à une somme dont la valeur est au minimum 17 si le joueur est en dessous de 21.

3 Notre modèle du blackjack.

Pour construire notre modèle via la librairie stormpy, nous allons devoir regarder les états initiaux et finaux parlés précédemment, des états de notre jeu et enfin, des différentes transitions pour passer d'un état à un autre. Concernant les états de notre modèle, nous allons les scinder en 3 catégories :

- Les états initiaux. C'est-à-dire celui dont la somme du joueur est entre 4 et 21 et la somme du croupier est entre 2 et 11.
- Les états finaux. Autrement dit, les états qui permettent de décider si le joueur ou le croupier est gagnant comme expliqué avant.
- Les états de transition pour notre jeu. Les états qui sont donc ni initiaux ni finaux.

Ensuite, concernant les transitions, nous avons le choix entre piocher ou d'en rester là que l'on soit le joueur ou le croupier (avec la subtilité que le croupier devra avoir au minimum 17 pour s'arrêter). Pour ce qui est de rester, c'est de ne prendre aucune carte et de terminer son tour (joueur doit terminer avant que le croupier ne commence à tirer des cartes). Pour le tirage par contre, la carte tirée suit une loi uniforme et donc chaque carte à 1/13 d'être tirée mais ce qui va nous intéresser, c'est leur valeur. On n'aura donc que la somme de nos cartes va augmenter entre 2 et 9 avec une probabilité de 1/10 chacune mais il y aura 2/5 chance que la somme augmente de 10 (car 10, valet, dame et roi ont la même valeur). Au final, notre modèle ressemblera donc à cela :

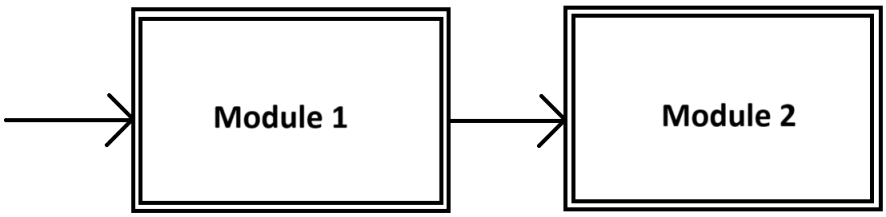


FIGURE 1 – Schémas de notre modèle.

Chaque module représente un ensemble d'états avec des transitions. Notre module 1 sera consacré à l'action du joueur (à savoir tirer une carte) et le passage du module 1 au module 2 sera l'action de s'arrêter pour le joueur. Enfin, le module 2 concerne donc les actions du croupier qui sont de piocher des cartes jusqu'à atteindre le minimum de 17 et ensuite de s'arrêter. Vu que le joueur doit commencer le jeu à chaque partie, le module 2 n'est pas initial mais les 2 modules sont finaux (le fait de dépasser 21 pour le joueur (module 1) et pour le croupier (module 2)).

Avec ce modèle, nous pouvons donc construire notre formule que nous souhaitons vérifier qui est de calculer la probabilité de gagner dans le jeu du blackjack. Cette probabilité sera donc notre limite théorique et donc, la valeur que nos agents devront essayer d'approcher. Pour cela, notre formule devra donc regarder les différents moyens que nous avons pour atteindre les états finaux et ensuite calculer cette probabilité qui est le total d'exécutions gagnantes sur la totalité des exécutions du modèle.

4 Agent naïf

Maintenant que nous connaissons notre limite théorique, essayons de créer notre agent pour s'en rapprocher le plus possible. Pour cela, essayons avec un agent naïf sans grande complexité. Pour rappel, notre agent contrôle uniquement les actions du joueur et le fait qu'il puisse tirer une nouvelle carte ou s'arrêter. Tout d'abord, on va regarder si le joueur est au-dessus de 17, si oui on l'arrête sinon on continue. Ensuite, on regarde si le joueur est en dessous de 11, si oui alors on le fait piocher sinon on continue. Enfin, si on se retrouve dans le cas où le joueur est entre 12 et 16, alors on va regarder la carte visible du croupier. Si elle est en dessous de 6 alors, on ne le fait pas prendre de carte car le croupier a peu de chances d'atteindre le 17 ce qui le ferait s'arrêter et donc il devra reprendre une autre carte. À ce moment-là, les chances qu'il se retrouve au-dessus de 21 sont élevées. Pour terminer, si on rentre dans aucun des cas, nous piochons une carte car les chances que le croupier se retrouve entre 17 et 21 sont très grandes.

Pour le déroulement de la partie, nous allons juste exécuter cet algorithme en boucle jusqu'à ce que la partie se termine puis recommencer. À chaque partie, nous notons si le joueur gagne, perd ou fait une égalité avec le croupier. Enfin, nous pourrons diviser le taux de victoire avec le nombre total de parties et regarder si on est proche ou non de notre limite précédente. Au final, sur 1000000 de parties, nous obtenons :

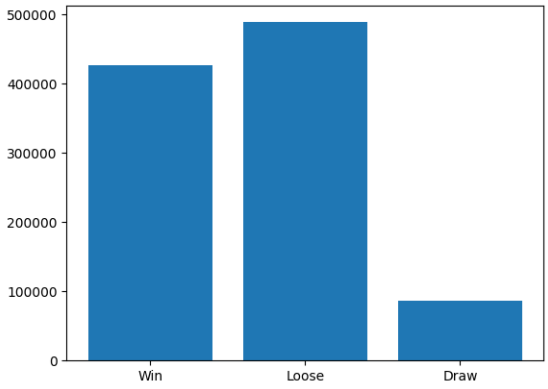


FIGURE 2 – Résultats sur 1 000 000 de parties de l'agent naïf.

Avec cette méthode, nous pouvons voir que nous avons 42.58% de parties gagnantes sur notre total de parties. Nous vérons que par la suite, c'est agent (malgré qu'il soit naïf) minimise les risques à comme conduite de minimiser les risques sur ces parties. Cette manière de procéder lui donne ainsi de très bons résultats.

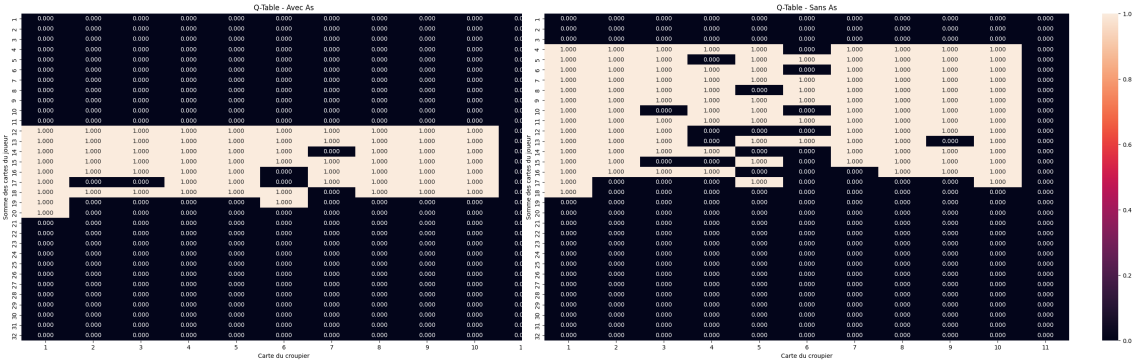
5 Agent Q-Learning.

Regardons avec une technique plus complexe qui utilise une table de Q-Learning. Avec cette technique, notre agent va être accompagné d'une table qui l'aidera au fur et à mesure pour prendre des décisions. Pour construire cette table, nous allons utiliser notre agent avec une valeur qui va décroitre avec l'avancement des parties et suivant cette valeur, l'agent sera soit dans une phase d'exploration et donc prendre une action aléatoire soit dans une phase d'exploitation et l'agent va utiliser la table pour prendre sa décision. Cette table va surtout nous aider car à chaque fois que nous allons faire une action, nous allons changer une valeur dans la table suivante si l'action a été bénéfique pour l'agent ou non. Ce changement est calculé de la manière suivante :

$$Q^{new}(s_t, a_t) = (1 - \alpha) * Q(s_t, a_t) + \alpha(r_t + \gamma * \max_a Q(s_{t+1}, a))$$

où α est le taux d'apprentissage, $Q(s_t, a_t)$ la valeur actuelle dans la table, r_t la récompense de notre action et γ notre facteur d'escompte.

En conséquence du moyen que nous venons de mettre en place pour permettre à notre agent de prendre des bonnes décisions, nous n'allons pas tout de suite aller tester notre agent, il nous faut d'abord l'entrainer. Pour cela, nous allons mettre en place 2 parties, une qui nous servira pour construire la table avec des parties d'entrainements et enfin, une deuxième qui nous servira de test pour cette table et nous donnera notre pourcentage de victoires. Pour le graphique qui va suivre, les paramètres suivants ont été décidés sur base de plusieurs tests et en sont sortis comme les meilleurs. Sur 100 000 épisodes, le meilleur duo de paramètre pour le taux d'apprentissage et le facteur d'escompte sont 0.2 et 0.3. Avec ces paramètres, nous avons entrainé notre table qui prendra les décisions suivantes :



(a) Q-table avec un as utilisable.

(b) Q-table sans as utilisable.

La valeur 1 est pour indiquer la pioche et la valeur 0 pour s'arrêter Et donc, avec cette Q-table remise en une seule (à l'aide d'un booléen pour savoir dans quelle sous table on se trouve), notre agent a passé la partie test pour savoir s'il gagne souvent ou non. Nous avons donc :

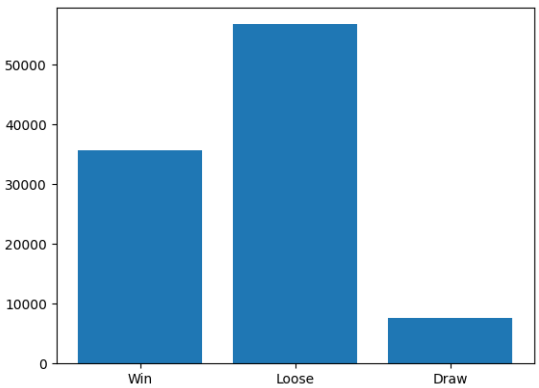
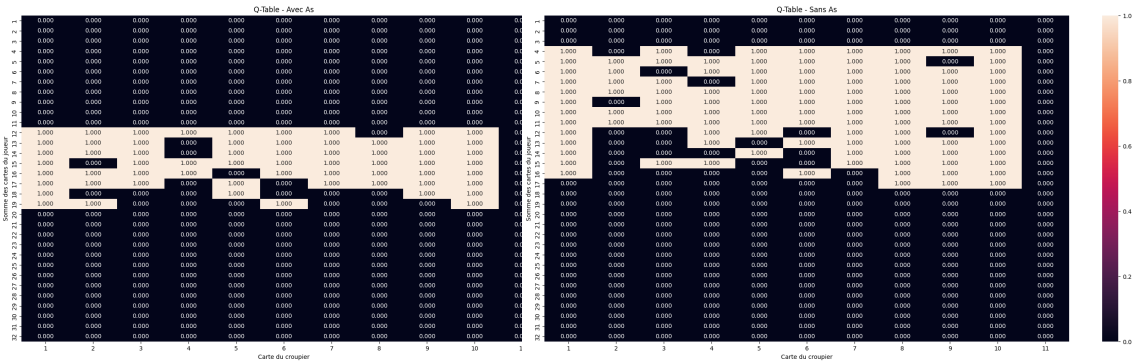


FIGURE 4 – Résultats sur 1 000 000 de parties de l'agent Q-Learning avec Q-table.

Il peut paraître surprenant que nous obtenions des valeurs plus petites que notre agent naïf mais il faut pour cela regarder nos Q-tables pour comprendre. En effet, certaines de nos actions consistées à ne pas piocher alors que nous avons de grandes probabilités de perte mais surtout une pioche sûre car nous ne dépasserons pas le 21. Pour remédier à cela, nous allons essayer d'aider notre Q-table pour une partie des décisions pour que nous puissions l'aider dans le comportement à atteindre.

6 Agent Q-Learning avec bouclier.

Le principe de cet agent reste identique à celui expliqué précédemment sauf sur un point : sa prise de décision. En effet, nous avons remarqué que pour certaines cases, il aurait été plus convenable de prendre une autre action par rapport à celle proposée par la table. De ce fait, nous allons modifier sa fonction d'action pour qu'il choisisse entre notre Q-table (que ce soit la phase d'exploration ou d'exploitation) ou bien notre algorithme naïf. Grâce à cela, nous allons pouvoir le forcer à explorer de nouvelle possibilité mais également l'aider pour des actions où l'agent aurait un peu plus de mal à se décider sur laquelle à entreprendre. Au final, nous obtenons comme Q-table :



(a) Q-table avec un as utilisable.

(b) Q-table sans as utilisable.

Ces tables sont déjà beaucoup plus remplies que les précédentes ce qui va permettre de prendre de meilleures décisions. Suite à cela, on recommence le test et nous obtenons donc le graphique suivant.

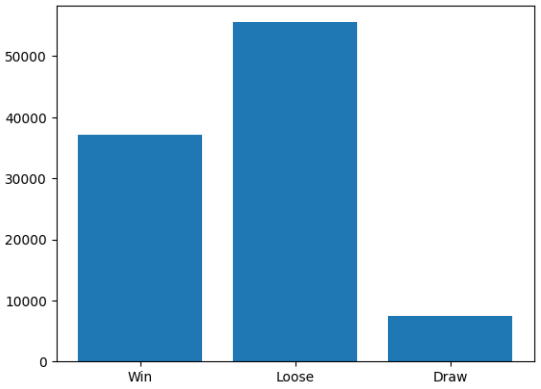


FIGURE 6 – Résultats sur 1 000 000 de parties de l'agent Q-Learning avec Q-table et un bouclier.

On remarque qu'avec ce bouclier en plus, nos performances ont pu augmenter mais sont encore loin de notre agent naïf.

7 Agent réseau de neurone.

Dans cette section, nous allons parler d'un agent encore plus complexe que celui discuté précédemment car nous voulons remplacer la Q-table. En effet, la méthode avec la table fonctionne dans des environnements comme celui-ci car assez petits pour rentrer toutes les données sans avoir une explosion d'états mais dans certains environnements, nous pouvons avoir trop d'états ce qui rend la table impraticable (par exemple, si notre environnement possède une observation ou une action avec une plage de valeur continue, alors on ne peut représenter tous les cas possibles). À cause de cela, nous pouvons recourir à l'utilisation de réseau de neurones qui peuvent contourner ce problème. Dans le cas-ci présent, ce problème n'est pas étant donné que toutes nos observations et notre action sont sous forme discrètes.

N otre réseau de neurones est fait de 3 couches et ensuite d'une sortie comme pour un classifier. Ainsi, il prendra en entrée les observations perçues par l'environnement (somme du joueur, carte visible du croupier et si un as est utilisable ou non) et devra nous retourner l'une des 2 classes finales qui représenteront soit la pioche soit l'arrêt. Comme pour l'agent précédent, il va falloir d'abord passer par une phase d'entrainement pour construire les poids de ce réseau pour la prise de décision et enfin la phase de test pour révéler s'il est performant ou non. C'est ce que nous montre le graphique suivant :

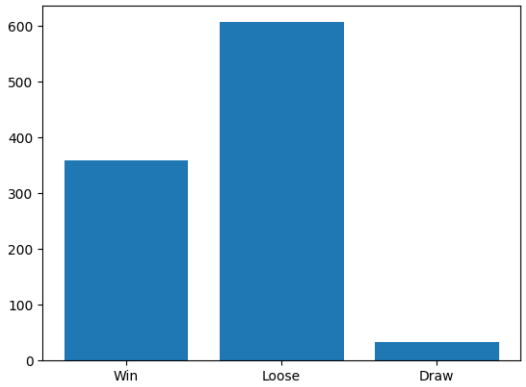


FIGURE 7 – Résultats sur 5 000 parties de l'agent Q-Learning avec Q-table et un bouclier.

On se rend compte que le réseau de neurones est déjà puissant comme il est actuellement avec une atteinte de presque 40%. Pour l'aider comme pour le Q-Learning avec sa table, nous allons mettre également en place un bouclier pour l'aider à prendre certaines décisions.

8 Agent réseau de neurone avec bouclier.

Comme discuté fin de la section précédente, nous allons garder notre agent comme il est actuellement et modifier uniquement la partie du choix de l'action. Comme cela, nous pourrons en partie aiguiller notre agent sur le choix de l'action et éviter une trop grande partie d'aléatoire. Nous aurons donc les résultats suivants :

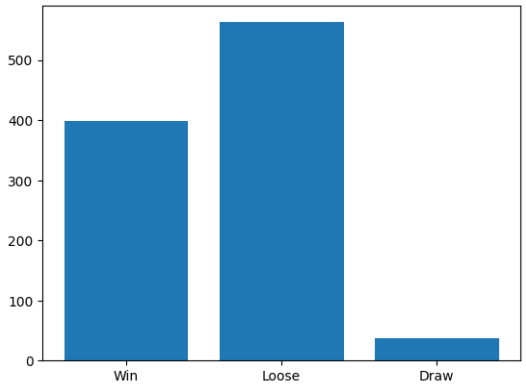


FIGURE 8 – Résultats sur 5 000 parties de l'agent Q-Learning avec Q-table et un bouclier.

Nous remarquons une augmentation de notre pourcentage de réussite suite à l'ajout de cette protection supplémentaire sur nos actions. Cependant, elle est moins importante qu'avec notre Q-table dû au fait que le réseau de neurones avait déjà un bon taux de réussite.

9 Problèmes rencontrés et moyens d'améliorations.

Concernant les problèmes rencontrés lors de l'élaboration du projet, le problème principal que j'ai eu a été de ne pas tomber dans le plagiat. En effet, avec le travail déjà effectué lors de l'année, je voulais éviter que le projet se retrouve à juste faire un copier-coller de ce qui a été fait lors de l'année. Du coup, j'ai décidé de tout reprendre de zéro et de tout recoder par moi-même. La plus grosse difficulté a été du côté de la librairie stormpy. Je n'arrivais pas à bien visualiser comment créer un bon modèle qui représenterait fidèlement l'environnement du blackjack de Gymnasium. La documentation étant pauvre, je me suis beaucoup basé sur des exemples disponibles de la librairie pour m'assister lors de la création de ce code. Pour mieux comprendre le problème que j'ai rencontré, j'ai laissé le code erroné pour plus de compréhension.

Des points pouvant être améliorés sont certainement nombreux mais les 2 points les plus importants selon moi sont :

- De meilleurs choix de paramètres pour le taux d'apprentissage et de facteur d'escompte pour les agents avec de l'apprentissage par renforcement. Cela permettrait d'augmenter les résultats pour que l'entrainement au fil des parties est un réel sens.
- D'autres modèles de réseaux de neurones. En effet, un seul réseau a été traité dans ce projet mais il pourrait être intéressant de regarder si un modèle avec plus ou moins de couches est utile, si un optimiseur différent peut aider et encore d'autre chose concernant les réseaux de neurones.

10 Conclusion.

Choisir des méthodes complexes pour résoudre des environnements simplistes comme celui du blackjack peut-être à la fois couteux en temps (réseau de neurones) mais aussi en mémoire (Q-Learning avec table). Au final, pour essayer de se rapprocher au plus possible d'un maximum de gain, il suffit de jouer avec les mouvements les plus sécurisés possible (ceux qui ne nous font pas perdre avec une probabilité de 1) et de ne pas prendre trop de risques inutiles.

11 Bibliographie

Références

[1] Farama Foundation. Blackjack.

[2] Wikipedia. Q-learning.

[3] Nicolas Vallois. fmsd-project-2023/group2/code_finale/q_learning.py.

[4] Matthias Volk. Storm Tutorial @DisCoTec 2020 - Part 2 : "Introduction to Storm".

[5] Matthias Volk. Storm Tutorial @DisCoTec 2020 - Part 3 : "Advanced Features with Stormpy".

[6] fakemonk1. Reinforcement-Learning-Lunar_Lander/Lunar_Lander.py.

[7] Nicolas Vallois. fmsd-project-2023/group2/code_finale/storm_mdp_maker.py.

[8] Moves RWTH Aachen. Stormpy Documentation.