FrozenLake

Sujet difficile

1 Présentation

1.1 Le Jeu

L'objectif du jeu est d'amener votre personnage jusqu'à l'endroit où son Frisbee s'est égaré. L'action se déroule sur un lac gelé en hiver et le parcours ne vas pas s'avérer facile. En effet, lorsque vous déplacez votre personnage vers la droite par exemple, celui-ci glisse et sa trajectoire peut finalement dévier vers le haut ou vers le bas et ceci de manière aléatoire. Pour corser le jeu, à certain endroit du lac, la glace est fragile et si votre personnage arrive sur une de ces cases, il tombe à l'eau et il recommence le jeu à la position de départ. Pour augmenter encore la difficulté, la glace s'avère de plus en plus glissante en se rapprochant de l'objectif ce qui complique les déplacements en les rendant de plus en plus aléatoires.

1.2 Début du projet

Pour vous familiariser avec le jeu, nous vous conseillons de jouer à plusieurs parties afin de bien assimiler l'aspect aléatoire des déplacements. Pour cela, lancez le fichier .py qui déclenche directement une partie au clavier. Utilisez les touches q, z, d et s pour vous déplacer.

1.3 Une nouvelle approche : le Q-Learning

Habituellement, lorsque nous avons créé une IA pour un jeu, nous connaissions les règles du jeu et nous savions à quoi servait chaque action. En effet, dans Pac-Man, vous savez que la manette permet de se déplacer dans un labyrinthe délimité par des murs. Vous savez aussi que les Pacgums servent à manger les fantômes. Dans FrozenLake, nous allons construire un algorithme de Q-Learning qui ignore tout du jeu. L'algorithme associé va être très différent de ce que nous avons fait précédemment :

- Dans PacMan, l'IA orientait PacMan pour qu'il mange des Pacgums tout en évitant les fantômes. L'algorithme avait donc une connaissance des règles du jeu et il ne pouvait être utilisé que pour le jeu PacMan.
- Dans Tron, l'algorithme a la capacité de dérouler des simulations, ce qui sous-entend qu'il a une connaissance totale des règles du jeu pour pouvoir simuler des milliers de parties. Certes, l'algorithme de Monté Carlo peut s'adapter à d'autres jeux, mais avec la contrainte d'avoir une connaissance complète des règles et des éléments de jeu.
- Pour l'algorithme Minimax, nous sommes dans le même cas que l'algorithme Monte-Carlo. Cet algorithme est assez générique pour s'adapter à d'autres jeux comme les échecs ou le Go… mais il nécessite que l'ensemble des règles soit connu pour dérouler la récursivité.

L'algorithme de Q-learning, ne requiert pas de connaissance préalable du jeu, on dit qu'il est « model free ». L'algorithme se connecte à un environnement et effectue des actions sans savoir à quoi elles correspondent. Cependant, il reçoit à chaque action une récompense sous forme d'un nombre réel. Par exemple pour un jeu d'arcade, cette récompense peut être vue comme la variation du score entre

deux affichages. C'est uniquement grâce à ces récompenses que l'algorithme va apprendre à faire telle ou telle action car il ne connait rien des règles du jeu :

- L'IA ne sait pas à quoi correspondent les actions possibles. Ainsi, elle n'associe pas l'action de basculer le joystick vers la droite au fait de déplacer le personnage vers la droite. Mais en ne touchant jamais le joystick, l'IA ne va pas recevoir de récompenses, elle va donc préférer les scénarios où elle se déplace.
- L'IA ne connaît pas l'existence des trous dans le sol ainsi que le risque associé. Cependant, en tombant dans un trou, la récompense associée va être négative. Ainsi, à force d'apprentissage, l'IA va chercher à éviter de passer sur ces cases et va, à force d'expérience, chercher à les éviter en prenant des chemins passant loin des trous.
- L'IA, une fois le frisbee trouvé, va avoir tendance à se diriger vers cette case car elle est associée à une forte récompense. Comme nos parties ont un nombre limité de tours, l'IA va indirectement choisir un chemin plus court car il lui permettra de ramasser le frisbee plusieurs fois dans la même partie ce qui augmentera ses récompenses. Ce ne sera pas forcément le chemin le plus court, car l'IA fera aussi un compromis entre un chemin court et un chemin sûr.

Sur la page Wikipedia, vous trouverez comme description : <<Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards.>> Le caractère model-free signifie que l'algorithme ne connaît pas les règles du jeu, il le gère comme une boite noire.

2 Principe

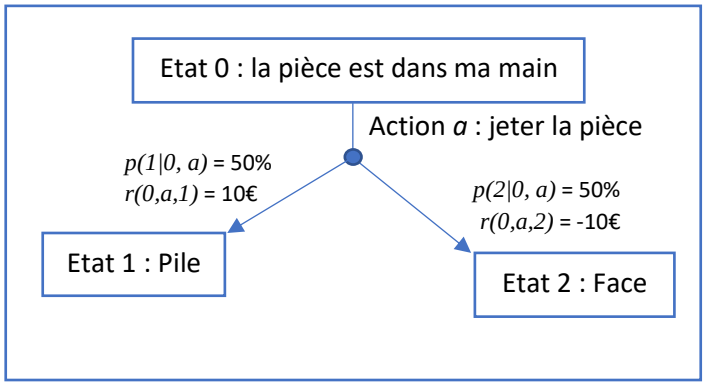
2.1 Modélisation

Nous décrivons un jeu comme un **ensemble d'états** $\mathbb{E}$ et un **ensemble d'actions** $\mathbb{A}$. Chaque action $a \in \mathbb{A}$ amène d'un état courant $e \in \mathbb{E}$ vers un nouvel état $e' \in \mathbb{E}$, on appelle **transition** un triplet (e, a, e'). Les transitions sont stochastiques car tout couple d'état/action n'amène pas toujours vers le même état, il y a une part de hasard. Pour modéliser cela, nous associons à chaque transition une probabilité de réalisation notée $p(e'|e, a)$ qui se lit : probabilité d'arriver à l'état e' depuis l'état e en effectuant l'action a. Ces probabilités définissent une distribution pour chaque couple état/action :

$$\forall e \in \mathbb{E}, \forall a \in \mathbb{A} : \sum_{e' \in \mathbb{E}} p(e'|e, a) = 1$$

Ce qui se lit : la somme des probabilités des nouveaux états atteints depuis un état e et une action a est égale à 1. Nous associons une **récompense** qui associe à toute transition (e, a, e') un nombre réel noté $\boldsymbol{r}(\boldsymbol{e}, \boldsymbol{a}, \boldsymbol{e}')$. La récompense est variable. Cependant, pour simplifier les formules, on considère la récompense moyenne notée $\bar{\boldsymbol{r}}(\boldsymbol{e}, \boldsymbol{a}, \boldsymbol{e}')$.

Habituellement, dans les jeux d'arcade, une action provoque toujours le même résultat. Mais, la vie de tous les jours n'est pas toujours aussi simple. Par exemple, lancer une pièce représente une action et le résultat est stochastique : 50% de change d'obtenir pile et 50% d'obtenir face. Pour le jeu du pile ou face, les transitions sont donc stochastiques. Nous pouvons modéliser cette réalité avec un graph des transitions qui indique pour chaque action les probabilités d'arriver dans tel ou tel état :

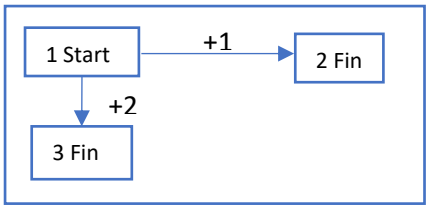


Dans le jeu FrozenLake :

- Les états correspondent à la position du joueur sur la grille de jeu.
- Il existe 4 actions correspondant aux 4 touches du clavier utilisées pour le déplacement.
- Une transition (e,a,e') correspond au déplacement de la position e vers la position e' en ayant appuyé sur la touche a.
- La probabilité $p(e'|e,a)$ correspond à la probabilité d'arriver à la position e' en partant de la position e et en ayant appuyé sur la touche a.

2.2 Rechercher les meilleures récompenses

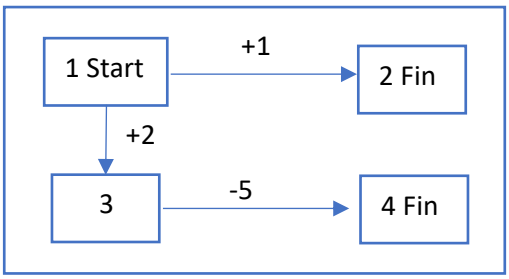
Nous allons considérer un jeu avec un état initial et deux actions possibles. Chaque action amène vers un nouvel état et une récompense est associée à chaque transition. Le jeu se termine après une action. Les deux actions possibles sont : Aller à droite, ce qui amène à l'état 2 et rapporte 1 point et Aller en bas, ce qui amène à l'état 3 et rapporte 2 points.



Graph de transitions du Jeu 1

Si vous deviez jouer à ce jeu, il est évident qu'après quelques parties, vous choisiriez tout le temps d'aller vers le bas car c'est l'option la plus rentable. Il suffit ici de choisir l'action qui rapporte le plus de points.

Mais ce n'est pas toujours aussi évident. Observons un deuxième jeu :



Graph de transitions du Jeu 2

Cette fois, même si aller vers le bas rapporte 2 points, pour terminer la partie, il faut aller jusqu'à l'état 4 en effectuant une action rapportant -5 points. Ainsi ce chemin s'avère peu rentable avec un total de 2-5=-3 points par rapport à l'autre option. Trouver la meilleure stratégie ne se résume donc pas à rechercher les actions associées aux meilleures récompenses.

2.3 La notion de politique

Pour créer notre IA, nous allons lui associer une politique. Une **politique déterministe** indique depuis un état courant l'action à sélectionner. Une **politique stochastique** associe à chaque action disponible depuis l'état courant une probabilité d'être choisie. Supposons par exemple que dans le jeu 1, la politique soit la suivante :

- 60% du temps aller en bas
- 40% du temps aller à droite

Quel serait le gain moyen de cette politique ? Il suffit pour cela d'estimer le gain moyen associé à ces probabilités : 60% * 2 + 40% * 1 = 1,4

Lorsque l'IA doit jouer en respectant une politique donnée, comment choisit-elle une action avec une politique stochastique ? Supposons que nous ayons trois actions possibles et une politique stochastique leur associant les probabilités de 60%, 30% et 10%. Choisir une action peut s'effectuer de la manière suivante : tirer un nombre aléatoire dans l'intervalle [0,100[. Si le tirage se trouve dans l'intervalle [0,60[, alors la première action est choisie. Si le tirage se trouve dans l'intervalle [60,90[, elle sélectionne la deuxième action. Lorsque le tirage se trouve dans l'intervalle [90,100[, la dernière action est choisie.

Remarque : le concept de politique stochastique peut paraître déroutant. Habituellement, une stratégie ou un choix de jeu s'énonce de manière déterministe : entourer l'adversaire ou récolter de l'or et fabriquer des combattants. Pourtant dans certains jeux, la politique optimale est stochastique. Par exemple, dans le jeu Pierre / Feuille / Ciseaux, une politique visant à présenter uniquement la Pierre serait un échec car l'adversaire n'aurait qu'à présenter la Feuille à chaque fois pour gagner. La politique optimale associe à chaque possibilité une probabilité de 1/3. Le joueur doit choisir aléatoirement une de ces options avec cette politique.

2.4 La grandeur Q

On peut décrire une simulation comme une série de transitions. Pour cela, on utilise un indice t comme marqueur de temps, indice qui s'incrémente à chaque action. Ainsi, on définit une ***simulation* τ** comme une séquence de paires état/action (e_t, a_t) :

$$\tau = (e_0, a_0)(e_1, a_1) \dots (e_k, a_k)$$

Nous notons $r_t = r(e_t, a_t, e_{t+1})$ la récompense obtenue à l'étape t. De cette façon, nous définissons le **gain** associé à une simulation τ :

$$R(\tau) = \sum_{t=0}^{k} r_t$$

Remarque : r_t n'est pas correctement défini pour $t = k$ car e_{k+1} est inconnu. Ceci n'est pas un réel problème car suivant le problème étudié, ce cas particulier se traite facilement.

Nous définissons la politique maxQ de la manière suivante :

- Depuis un état donné, nous choisissons l'action associé au gain moyen maximal.

Suivant cette politique, nous définissons deux grandeurs importantes :

- La grandeur $\boldsymbol{V_e}$ associée au gain moyen obtenu depuis un état *e* donné.
- La grandeur $\boldsymbol{Q_{e,a}}$ associée au gain moyen obtenu depuis un état *e* en effectuant l'action *a*.
-

Connaissant *Q,* nous pouvons en déduire *V* :

$$V_e = \max_a Q_{e,a}$$

Cette formule s'interprète ainsi : le gain le plus intéressant depuis un état e s'obtient en choisissant l'action associée au gain moyen le plus élevé.

Connaissant *V*, nous pouvons en déduire *Q* :

$$Q_{e,a} = \sum_{e' \in E} p(e'|e,a).[\bar{r}(e,a,e') + V_{e'}]$$

Dans cette formule, l'état e' désigne un état quelconque or tous les états ne sont pas accessibles depuis un état donné. Ce n'est pas très grave, car dans ce cas, il suffit de mettre la probabilité associée à 0. La quantité $\bar{r}(e,a,e') + V_{e'}$ correspond aux gains que l'on peut espérer par la transition (e,a,e'). Pour trouver la valeur *Q*, il suffit donc de pondérer les gains moyens associés de chaque transition (e,a,e') par la probabilité de cette transition.

En combinant les deux formules, nous pouvons retirer la grandeur V pour obtenir une définition utilisant uniquement la grandeur Q :

$$Q_{e,a} = \sum_{e' \in E} p(e'|e,a).\left[\bar{r}(e,a,e') + \max_{a'} Q_{e',a'}\right] \quad (1)$$

3 Mise en place

3.1 Estimer les valeurs Q

N'ayant aucune information sur le jeu, le programme va d'abord simuler plusieurs parties totalement aléatoires. Chaque partie va ainsi générer une série de transitions. Dans le jeu FrozenLake, nous ne savons pas lorsqu'une partie est terminée, en effet, lorsque le joueur meurt en tombant dans un trou, cela amène un malus et le joueur repart à la position initiale, il n'y a pas de comptage de vies restantes. Du point de vue de l'IA, les trous dans la glace pourraient tout aussi bien être des téléporteurs associés à un coût d'utilisation ! Par conséquent, pour chaque simulation, nous allons jouer un nombre constant de tours, ainsi, toutes les simulations auront le même nombre de transitions.

Une fois les simulations générées, nous avons suffisamment d'information sur le jeu pour estimer les grandeurs $p(e'|e,a)$ et $\bar{r}(e,a,e')$. Pour cela, nous comptabilisons :

- Le nombre de fois où nous avons choisi l'action a depuis l'état e : *n(a|e)*
- Le nombre de fois où s'est produit une transition *(e,a,e') : n(e'|a,e)*
- La somme des récompenses obtenues pour les transitions *(e,a,e') : s(e'|a,e)*

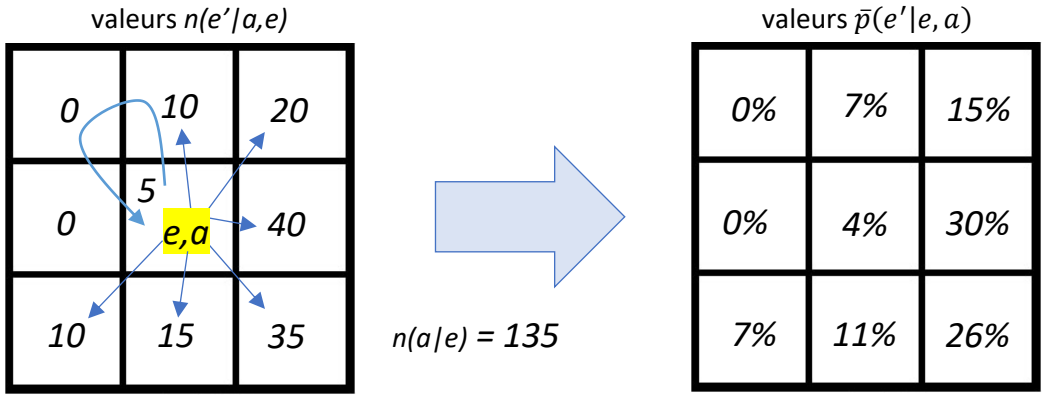
Ces informations peuvent être conservées d'un run à l'autre car ces observations sont indépendantes de la politique choisie. En les conservant, on augmente donc notre connaissance de l'environnement. Cependant, les simulations peuvent être oubliées car toute l'information nécessaire est stockée dans ces trois grandeurs : *n(a|e), n(e'|a,e)* et *s(e'|a,e).*

Nous estimons maintenant les valeurs $p(e'|e,a)$ et $\bar{r}(e,a,e')$. La notation $\bar{r}$ prend déjà en compte que nous ne connaissons pas exactement la récompense moyenne, mais que nous avons juste une estimation. Nous utilisons ainsi la notation $\bar{p}(e'|e,a)$ pour désigner l'estimation de $p(e'|e,a)$:

$$\bar{p}(e'|e,a) \sim \frac{n(e'|a,e)}{n(a|e)}$$

$$\bar{r}(e,a,e') \sim \frac{s(e,a,e')}{n(e'|a,e)}$$

Voici un exemple ci-dessous :



Il nous reste maintenant à construire les différentes valeurs Q. Pour cela, nous utilisons un tableau de taille : nombre d'états x nombre d'actions. Pour notre problème, sa taille est acceptable. Nous allons utiliser l'approche suivante :

> Initialisez les valeurs $Q_{e,a}$ à zéro
>
> Refaire un certain nombre de fois :
>
> Appliquez la formule (1) une fois pour chaque case du tableau

Bien sûr, vous n'allez pas effectuer une boucle infinie. Il faut donc trouver un critère d'arrêt judicieux permettant de détecter la convergence et la stabilisation des valeurs *Q*. Pour cela, on peut par exemple comparer le tableau Q de l'itération courante avec le tableau Q de l'itération précédente. Si la variation de chaque case est inférieure à 0.01, on peut considérer que les valeurs *Q* sont maintenant stables.

3.2 Mise en place d'une politique de jeu

On peut maintenant mettre en place la politique de jeu MaxQ. Pour cela, il suffit lorsque le joueur se trouve sur une certaine case correspondant à l'état *e* de choisir l'action offrant une espérance de gain maximale :

$$\max_a \ Q_{e,a}$$

Examinez des simulations utilisant cette politique en les affichants à l'écran. Cela vous permet de faire un contrôle des performances actuelles de l'IA.

3.3 On améliore !

Il se peut que le résultat de votre premier run soit peu satisfaisant. En effet, si lors des simulations aléatoires, la case du Frisbee n'a jamais été trouvée, l'IA n'est pas en mesure d'aller dans sa direction. Il se peut que votre joueur IA joue alors la sécurité en tournant en rond dans un coin.

Dans la mise en place d'un apprentissage, il existe un équilibre entre « l'exploitation » des bonnes stratégies connues et « l'exploration » de nouvelles approches. Jouer uniquement la carte de l'exploitation en rejouant des scénarios similaires aux meilleurs scénarios est un choix confortable car il vous garantit que vous obtiendrez des performances proches des meilleures performances connues. Par contre, il ne faut pas compter sur l'exploitation pour trouver un passage secret ou un super bonus. En effet, c'est le travail de la stratégie « d'exploration » qui consiste à tester de nouveaux chemins et de nouvelles options. Cette stratégie rapporte peu car la plupart des séquences générées vont être sans grande valeur, cependant, elle nous permettra de trouver de nouvelles approches qui vont nous permettre d'aller plus loin dans le jeu.

La stratégie mise en place par le *max Q* est une stratégie d'exploitation. En début d'apprentissage, cette stratégie ne permet de revisiter que les endroits du jeu déjà visités. Il faut donc intégrer une astuce qui permette aussi l'exploration durant nos simulations.

Encourager l'exploration

Cela peut, indépendamment du jeu, être mis en place en fournissant une récompense pour tout nouvel état exploré durant une partie. On peut par exemple fixer cette récompense à +1.

Equilibrer exploitation et exploration

Choisissez une valeur ε entre 0 et 1. A chaque tour de jeu durant les simulations, vous pouvez choisir la politique optimale avec une probabilité ε ou un choix aléatoire avec une probabilité 1-ε.

Laisser une chance aux choix moins rentables

Nous pouvons choisir une politique de jeu moins stricte que celle du *maxQ* qui nous force à toujours choisir la même action depuis un état donné. Nous pourrions par exemple préférer une politique qui choisissent l'action aléatoirement mais proportionnellement au gain espéré. Ainsi, deux actions associées à un gain moyen de 100 et 200, pourrait être associées à une politique de 1/3 2/3. Cependant, cette règle de proportionnalité ne marche pas correctement lorsque nous avons des valeurs négatives dans les récompenses. Nous allons dans ce cas plutôt utiliser la fonction SoftMax. La fonction Softmax permet de construire des probabilités à partir d'une liste de valeurs en offrant les propriétés suivantes :

- Plus la valeur est haute, plus la probabilité est haute
- Deux valeurs proches ont des probabilités proches
- Si l'on décale les valeurs en entrée d'une certaine constante, les probabilités construites restent inchangées. Ainsi, les valeurs -1 2 et 3 produisent les mêmes probabilités en sortie que les valeurs 1 4 et 5.

Nous allons donc utiliser les valeurs des gains associées à chaque action et la fonction SoftMax pour construire la probabilité associée à chaque action. Ainsi, pour *m* actions associée chacune à un certain gain noté g_i. Nous allons définir la probabilité de choisir l'action *i* de la manière suivante :

$$p_i = \frac{exp(g_i)}{\sum_{j=0}^{m-1} exp(g_j)}$$

Vous pouvez remarquer que la valeur du dénominateur est identique pour toutes les probabilités p_i. Voici un exemple :

Action	g_i	$exp(g_i)$	$\sum_{j=0}^{m-1} exp(g_j)$	p_i
0	3	20,0	21,41	20,0/21,41 = 93,4%
1	0.2	1,22		1,22/21,41 = 5.70%
2	-2	0,13		0,13/21,41 = 0.61%
3	-5	0,06		0,06/21,41 = 0.28%

La politique MaxQ aurait choisi uniquement l'action 0, car elle est associée au meilleur gain : 3. La politique associée à la fonction Softmax va donner 93.4% de sélectionner l'action 0 mais aussi 5.7% de sélectionner l'action 1. Les autres actions ont des probabilités faibles, mais cependant non nulles, elles ne sont donc pas abandonnées comme dans la politique du MaxQ.