

# Apprentissage par renforcement

Mouhamed Kiram

Lucy Attwood

Mai, 2023



Projet M1 Mathématiques  
Sous la direction de M. Espinasse  
Master Mathématiques Appliquées et Statistique

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Algorithmes de programmation dynamique pour les systèmes simples</b>	<b>5</b>
2.1	Un jeu simple . . . . .	5
2.2	Processus de décision markovien . . . . .	6
2.2.1	Définition . . . . .	6
2.2.2	Exemple Chat Souris Fromage . . . . .	6
2.2.3	Index temporel, Règle de décision et Return . . . . .	8
2.3	Value Functions . . . . .	8
2.3.1	Politique et «processus de Reward de Markov» . . . . .	8
2.3.2	Value Functions et de «Action-Valeur» . . . . .	10
2.3.3	Les equations de Bellman . . . . .	10
2.3.4	La Value Function optimale . . . . .	12
2.4	Dynamic Programming Algorithms for solving MDPs . . . . .	13
<b>3</b>	<b>Temporal Difference Learning dans des espaces finis</b>	<b>16</b>
3.1	Motivation . . . . .	16
3.2	Monte Carlo Method . . . . .	16
3.3	Algorithme TD(0) . . . . .	17
3.4	TD( $\lambda$ ) . . . . .	19
<b>4</b>	<b>Pour aller plus loin, algorithmes pour grands espaces d'états</b>	<b>21</b>

# 1 Introduction

L'apprentissage automatique (Machine Learning) se divise en 3 grandes catégories :

- L'apprentissage supervisé : l'algorithme va apprendre à partir d'un jeu de données étiquetté
- L'apprentissage non supervisé : cette fois-ci l'algorithme n'a pas accès aux étiquettes et va donc par lui-même trouver la structure sous-jacente de ces données
- L'apprentissage par renforcement (Reinforcement Learning) : un agent est entraîné à prendre des décisions qui vont maximiser les récompenses lorsqu'il interagit avec son environnement. Il va donc effectuer des actions sur son environnement, et en retour reçoit une récompense (bonne ou mauvaise) et un nouvel état. L'agent va donc apprendre de ses erreurs et trouver une politique de décision qui maximisera sa récompense sur le long terme.

Nous allons nous focaliser sur ce dernier point **Reinforcement Learning**. Pour mieux comprendre l'idée derrière cette sous-branche, prenons un exemple plus parlant.

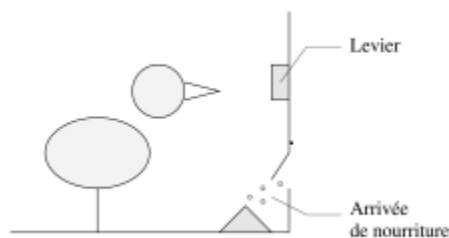


FIGURE 1 – La boîte de Skinner

La boîte de Skinner est un dispositif développé par le psychologue Burrhus F. Skinner dans les années 1930 et est souvent utilisé pour observer le comportement des pigeons, des rats, etc. Le principe est simple : l'animal est laissé dans la boîte et a la possibilité d'explorer et d'interagir avec son environnement.

Comme vous l'aurez compris, le pigeon va apprendre à appuyer sur le levier pour que de la nourriture soit délivrée. Le Reinforcement Learning va donc se baser sur l'apprentissage par essais/erreurs d'une action éventuellement conditionnée par une situation :

- Dès qu'il presse le bouton, le pigeon est récompensé.
- La récompense renforce cette action
- Le pigeon apprend à presser le bouton de plus en plus souvent.
- Modifier la récompense conduit à modifier le comportement que l'animal apprend.

Tout l'enjeu va donc être de reproduire artificiellement le conditionnement de l'animal et faire apprendre un comportement à une machine en lui distribuant des récompenses.

L'apprentissage par renforcement est utilisé dans divers contextes, de la recherche des meilleures actions à entreprendre pour gagner un jeu de morpion aux programmation des voitures autonomes.

Pour introduire le concept de l'apprentissage par renforcement, nous allons commencer par

considérer un système simple que nous cherchons à contrôler. Afin d'optimiser le système, nous allons créer une sortie de récompense chaque fois qu'une action est effectuée sur le système, reflétant ainsi l'impact immédiat de cette action. Nous cherchons ensuite à maximiser cette récompense à long terme afin de contrôler efficacement le système. Pour modéliser ces systèmes, une méthode couramment utilisée dans le cadre mathématiques est le "Processus de Décision Markovien" (MDP). Cette approche nous permet de déterminer la fonction de récompense à l'aide d'algorithmes de Dynamic Programming et donc de trouver la meilleure façon de contrôler le système. Cependant nous découvrirons que dans certains systèmes plus complexes et chaotiques la prédiction de ces fonctions de récompense devient difficile. Dans les Sections 3 et 4, nous explorerons des stratégies alternatives pour nous adapter à ces problèmes.

Nous allons d'abord étudier les MDP à états simples et les algorithmes pour mieux comprendre ces processus de décision avant de nous tourner vers des problèmes plus complexes.

## 2 Algorithmes de programmation dynamique pour les systèmes simples

L'ensemble des définitions et preuves de cette section proviennent de [1].

### 2.1 Un jeu simple

En tant que contrôleur, votre rôle est de jouer au jeu "Chat, Souris et Fromage". Le jeu se déroule dans un ensemble d'états, qui peut être représenté par une grille, par exemple 4x3, où chaque carré ou état correspond à une pièce avec une porte sur chaque mur. Les portes sont situées au nord, au sud, à l'est ou à l'ouest. Dans une pièce se trouve un chat, dans une autre se trouve du fromage, et dans une troisième se trouve votre souris. Votre objectif est d'aider la souris à trouver le fromage tout en évitant de rencontrer le chat.

À chaque tour, la souris change de pièce. Pour ce faire, vous avez le choix entre deux actions.

Action 1 : Vous lancez une pièce de monnaie. Si c'est face, la souris se déplace vers le haut, si c'est pile, la souris se déplace vers la droite (flèches bleues)

Action 2 : Vous lancez une pièce de monnaie. Si c'est face, la souris se déplace vers le bas, si c'est pile, la souris se déplace vers la gauche (flèches rouges)

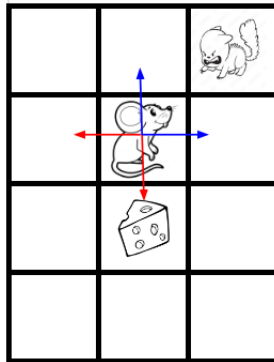


FIGURE 2 – Le choix de l'action dans le jeu 'Chat, Souris et Fromage'

Si vous rencontrez le chat dans une pièce voisine, vous perdez la partie, mais si vous trouvez le fromage, vous gagnez !

La grille est considérée comme une boucle, ce qui signifie que prendre la porte vers le nord lorsque vous êtes dans la pièce en haut à gauche vous déplacera dans la pièce en bas à gauche. Dans ce jeu simple, certaines pièces sont évidentes quant à l'action qui maximisera les chances de gagner pour le contrôleur, comme illustré dans la Figure 2. Si le contrôleur effectue l'action 2, il a une chance de 50% de gagner, tandis que l'action 1 déplacera définitivement la souris dans une pièce voisine du chat, augmentant le risque de perdre lors de sa prochaine action. Cependant, dans d'autres pièces, la meilleure action pour gagner peut être moins évidente.

En utilisant le cadre mathématique du "Processus de décision markovien", nous pouvons déterminer pour chaque pièce quelle action est préférable, confirmant ainsi l'intuition du

contrôleur et fournissant une réponse dans les cas où il n'est pas certain de la meilleure action à effectuer.

## 2.2 Processus de décision markovien

### 2.2.1 Définition

Dans ce contexte de «jeu simple», nous restreindrons le processus de décision de markovien (MDP) au MDP dénombrable. Un **MDP dénombrable** est défini comme un triplet  $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{P}_0)$  où

- $\mathcal{X}$  est l'ensemble des états, dénombrable et non vide.
- $\mathcal{A}$  est l'ensemble des actions, dénombrable et non vide.
- $\mathcal{P}_0$  la probabilité de transition assignée à chaque paire État et Action  $(x, a)$ .

L'autre élément clé d'un MDP est la **fonction de récompense immédiate**. Cette fonction attribue à chaque paire État et Action  $(x, a)$  l'esperance de récompense immédiate reçue lorsque l'action  $a$  est entreprise dans l'état  $x$ .

$$r(x, a) = \mathbb{E}[R_{(x,a)}]$$

### 2.2.2 Exemple Chat Souris Fromage

En appliquant ces définitions au jeu Chat Souris Fromage nous avons :

- $\mathcal{X}$ , une grille de pièces, dont certaines contiennent soit le fromage soit la souris , soit le chat. On numérote chaque pièce dans la grille comme suit

$$\begin{bmatrix} 0 & 4 & 8 \\ 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \end{bmatrix}$$

et on note que dans la exemple de Figure 2 les postions de le chat, souris et fromage sont les pieces 8,5 et 6 respectivement.

- $\mathcal{A}$ , l'ensemble des actions, dans pour notre exemple l'action 1,  $a_1$ , et l'action 2,  $a_2$ , sont les seules choix dans tous les etats.

$$\mathcal{A} = \{a_1, a_2\}$$

- $\mathcal{P}_0$ , en prenant encore l'état de la souris dans Figure 2,  $x_{mouse} = 5$ , pour action  $a_1$  la probabilite que le souris arrive sur la pièce  $y \in \mathcal{X}$  est

$$\mathbb{P}_{a_1}(x_n = x_{mouse}, x_{n+1} = y) = \begin{cases} \frac{1}{2} & \text{si } y = 4 \\ \frac{1}{2} & \text{si } y = 9 \\ 0 & \text{sinon} \end{cases}$$

et de la même maniere pour action  $a_2$

$$\mathbb{P}_{a_2}(x_n = x_{mouse}, x_{n+1} = y) = \begin{cases} \frac{1}{2} & \text{si } y = 1 \\ \frac{1}{2} & \text{si } y = 6 \\ 0 & \text{sinon} \end{cases}$$

On peut également représenter ces probabilités avec des matrices.

$$\begin{array}{cc} \text{Action 1} & \text{Action 2} \\ \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

Notez qu'il s'agit de la probabilité de se déplacer vers chacun des autres états à partir de l'état actuel de la souris dans la Figure 2 et **non pas** une matrice de transition de probabilité (mais nous l'utiliserons plus tard).

Afin d'étudier la fonction de récompense immédiate dans cet exemple, nous attribuons une valeur  $R_{Win}$  lorsque la souris trouve le fromage,  $R_{Lose}$ , lorsque la souris trouve le chat. Avec  $x_{mouse} = 5$  comme dans la Figure 2, on a

$$r(5, a_1) = \mathbb{E}[R_{(5, a_1)}] = \sum_{y=0}^{11} \mathbb{P}_{a_1}(x_n = 5, x_{n+1} = y) \times (\text{Reward } y) = \frac{1}{2} \times 0 + \frac{1}{2} \times 0 = 0$$

$$r(5, a_2) = \mathbb{E}[R_{(5, a_2)}] = \sum_{y=0}^{11} \mathbb{P}_{a_2}(x_n = 5, x_{n+1} = y) \times (\text{Reward } y) = \frac{1}{2} \times R_{Win} + \frac{1}{2} \times 0 = \frac{R_{Win}}{2}$$

On remarque donc bien que l'action 2 a une récompense imediate beaucoup plus forte que l'action 1. Notre fonction de récompense nous permettra donc d'attribuer un poids pour chaque action prise.

Voici un petit schémas récapitulatif de ce que nous avons vu pour le moment :



FIGURE 3 – Schéma récapitulatif : Apprentissage par renforcement

### 2.2.3 Index temporel, Règle de décision et Return

Afin d'utiliser le MDP, nous devons introduire d'autres concepts utiles. Dans l'exemple de Chat Souris Fromage, mais aussi de manière générale, la façon dont le contrôleur interagit avec le jeu est séquentielle et il est donc utile d'indexer les états et les actions à l'aide d'un index temporel. A l'instant  $t \in \mathbb{N}$  l'état courant sera noté  $X_t \in \mathcal{X}$  et l'action entreprise à cet instant  $A_t \in \mathcal{A}$ . Une fois qu'une action  $A_t$  est sélectionnée à l'instant  $t$ , le système effectue la transition suivante :

$$(X_{t+1}, R_{t+1}) \sim \mathcal{P}_0(\cdot | X_t, A_t).$$

$R_{t+1}$  est la récompense immédiate reçue pour la transition de  $X_t$  à  $X_{t+1}$ . Une règle qui décrit comment le contrôleur choisit ses actions s'appelle une **règle de décision**. Lorsque le contrôleur interagit avec MDP en utilisant cette règle de décision choisie, il collecte ces récompenses  $R_t$ . Nous définissons le **return** sous-jacent à une règle de décision comme la somme actualisée de ces réponses immédiates.

$$\mathcal{R} = \sum_{t=0}^{\infty} \gamma^t R_{t+1}$$

Le fait que la somme soit « actualisée » peut être vu dans la formule par la constante  $\gamma$ . Si  $\gamma < 1$  alors la récompense reçue à l'avenir vaut beaucoup moins que celle reçue dans les premières étapes. Cependant, lorsque  $\gamma = 1$ , la récompense  $R_1$  et  $R_{100}$  ont un poids égal pour le return. Nous pouvons maintenant décrire le but du contrôleur en utilisant ces concepts, il veut trouver la règle de décision qui maximisera le return. Dans le jeu Chat Souris Fromage avec  $\gamma < 1$ , il veut savoir quelle action prendre dans chaque pièce pour atteindre le fromage le plus vite possible et à tout pris éviter le chat au risque d'être mangé !

## 2.3 Value Functions

Afin de déterminer la règle de décision optimale, il est important de pouvoir quantifier le rendement de cette dernière. Ceci est fait en utilisant « Value Functions ». Pour rendre cela explicite, nous devons d'abord introduire un type spécifique de règle de décision (des politiques) et un nouveau processus.

### 2.3.1 Politique et « processus de Reward de Markov »

Lors de l'étude des règles de décisions, nous ne considérerons qu'une classe spéciale appelée « politique stationnaire déterministe ou stochastique ». Une politique  $\pi$  attribue chaque état à une distribution sur l'espace d'action.

Lorsque nous nous référons à une telle politique  $\pi$ , nous utiliserons  $\pi(a|x)$  pour dénoter la probabilité que l'action  $a_k$  soit sélectionnée par la politique  $\pi$  dans l'état  $x$ .

Dans le jeu Chat, Souris, Fromage, voici trois exemples de politiques.

Politique 1  $\pi_1$ , qui attribue chaque état à l'Action 1 avec probabilité 1 et à l'Action 2 avec probabilité 0.

$$\pi_1(a_1, x) = 1 \quad \forall x \in \mathcal{X} \quad \pi_1(a_2, x) = 0 \quad \forall x \in \mathcal{X}$$



Politique 2  $\pi_2$ , qui attribue chaque état pair à l'Action 1 avec probabilité 1 et chaque état impair à l'Action 2 avec probabilité 1

$$\pi_2(a_1, x) = \begin{cases} 1 & \forall x \in \mathcal{X} \text{ avec } x = 2n \\ 0 & \forall x \in \mathcal{X} \text{ avec } x = 2n + 1 \end{cases} \quad \pi_2(a_2, x) = \begin{cases} 0 & \forall x \in \mathcal{X} \text{ avec } x = 2n \\ 1 & \forall x \in \mathcal{X} \text{ avec } x = 2n + 1 \end{cases}$$

Politique 3  $\pi_3$ , qui attribue chaque état à l'Action 1 avec probabilité  $\frac{1}{2}$  et à l'Action 2 avec probabilité  $\frac{1}{2}$ .

$$\pi_3(a_1, x) = \frac{1}{2} \quad \forall x \in \mathcal{X} \quad \pi_3(a_2, x) = \frac{1}{2} \quad \forall x \in \mathcal{X}$$

Notez que avec la combinaison d'une politique avec un MDP, nous nous retrouvons avec un nouveau processus où les actions ne sont plus choisies. Nous appelons ce processus un «processus de Reward de Markov» (MRP)  $\mathcal{M} = (\mathcal{X}, \mathcal{P}_0)$  où  $\mathcal{P}_0$  attribue à chaque état une mesure de probabilité sur  $\mathcal{X} \times \mathbb{R}$ .

Afin de mieux visualiser et comprendre ces nouvelles notions nous avons choisi d'effectuer une petite simulation simple de ce MPR pour Chat, Souris, Fromage en prenant notre notation de la section 2.2.2 et la comme choix de politique :  $\pi_2$ .

1.  $X_0 = 5$ 
  - Comme 5 est impair on a que  $\pi_2(a_1, 5) = 0$  et  $\pi_2(a_2, 5) = 1$  alors le contronleur prende action 2.
  - Il lance sa pièce et elle atterrit sur pile, alors il se déplace d'une pièce vers la gauche jusqu'au numéro 1. Comme il n'y a ni chat ni fromage dans la pièce 1, la récompense immédiate est de 0.
2.  $X_1 = 1, \mathcal{R}_1 = 0$ 
  - Comme 1 est impair on a que  $\pi_2(a_1, 1) = 0$  et  $\pi_2(a_2, 1) = 1$  alors le contronleur prende action 2.
  - Il lance sa pièce et elle atterrit sur face, il se déplace d'une pièce vers le haut jusqu'au numéro 2. Comme il n'y a ni chat ni fromage dans la pièce 1, la récompense immédiate est de 0.
3.  $X_2 = 2, \mathcal{R}_2 = 0$ 
  - Comme 2 est pair  $\pi_2(a_1, 2) = 1$  et  $\pi_2(a_2, 2) = 0$  le contronleur prend l'action 1.
  - Il lance sa pièce et elle atterrit sur face, il se déplace d'une pièce vers la droite jusqu'au numéro 6. Ici, il trouve le fromage et reçoit donc une récompense immédiate de  $R_{Win}$ . Il a gagné la partie.
4.  $X_3 = 6, \mathcal{R}_3 = R_{Win}$

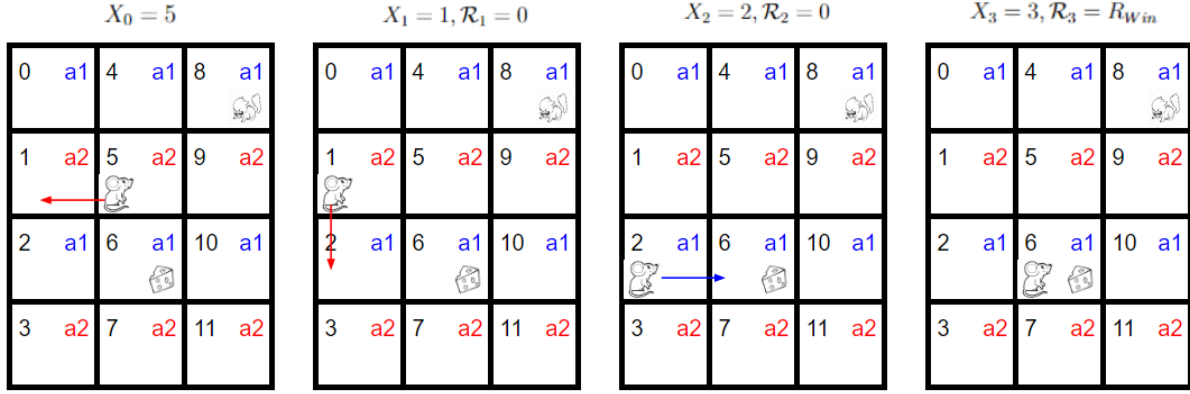


FIGURE 4 – Simulation de MRP pour le jeu Chat, Souris Fromage avec la politique  $\pi_2$ . En haut à gauche dans chaque case, numéro de pièce. En haut à droite, action effectuée dans ce carré pour la politique  $\pi_2$ .

### 2.3.2 Value Functions et de «Action-Valeur»

La **fonction valeur**  $V^\pi(x) : X \rightarrow \mathbb{R}$  pour un MRP, composé d'un MDP et d'une politique  $\pi$ , est définie par l'esperance du return total d'une suite  $(X, \mathcal{R}) = ((X_0), (X_1, \mathcal{R}_1), \dots, (X_t, \mathcal{R}_t) \dots)$

qui commence à  $X_0 = x$ . Cette fonction va attribuer une valeur à chaque état de notre système. Dans notre jeu, les cases proches du fromage auront une "valeur" plus grande que les cases proches du chats. Car une fois sur ces cases nos chances de gagner sont plus importantes. Une autre fonction utile est la **fonction action-valeur**  $Q^\pi(x) : \mathbb{X} \times \mathcal{A} \rightarrow \mathbb{R}$  pour la politique  $\pi$ . Nous supposons que la première action  $A_0$  est choisie au hasard mais à toutes les étapes suivantes l'action est choisie comme pour un MRP utilisant la politique  $\pi$ . Nous avons que

$$Q^\pi(x, a) = \mathbb{E} \left[ \mathcal{R} = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1} \middle| X_0 = x, A_0 = a \right] : x \in \mathcal{X}, a \in \mathcal{A}$$

La tâche pour déterminer ces valeurs  $V^\pi(x)$  et  $Q^\pi(x, a)$  est simplifiée grâce à un ensemble de résultats «les équations de Bellman.»

### 2.3.3 Les equations de Bellman

Un MDP  $(\mathcal{M} = \mathcal{X}, \mathcal{A}, \mathcal{P}_0)$  avec un facteur actualisé  $\gamma$  et une politique  $\pi$  qui est deterministe satisfait le système d'équations suivant :

$$V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{y \in \mathcal{X}} \mathbb{P}(x, \pi(x), y) V^\pi(y), \quad x \in \mathcal{X}$$

avec  $\mathbb{P}(x, \pi(x), y)$  la probabilité de passer de l'état  $x$  à  $y$  avec l'action  $\pi(x)$  déterminée par la politique. [1]

Notez qu'il s'agit d'un système linéaire d'équations et nous pouvons donc définir l'«Opérateur

de Bellman»  $T^\pi : \mathbb{R}^\mathcal{X} \rightarrow \mathbb{R}^\mathcal{X}$  par

$$(T^\pi V)(x) = r(x, \pi(x)) + \gamma \sum_{y \in \mathcal{X}} \mathbb{P}(x, \pi(x), y) V(y), \quad x \in \mathcal{X}$$

Cela nous permet de représenter le système sous une forme compacte.

$$T^\pi V^\pi = V^\pi$$

Nous verrons qu'un opérateur similaire sera également utile pour définir la valeur fonction optimale dans la section 2.3.4 et jouera donc un rôle important dans les algorithmes de programmation dynamique dans la section 2.4.

Cependant en restant dans le cas de la value function pour une politique déterministe, en prenant  $V^\pi$  le vecteur avec  $(V^\pi)_i = V(i)$ ,  $r^\pi$  le vecteur avec  $(r^\pi)_i = r(i, \pi(i))$  et  $P^\pi \in \mathbb{R}^{|\mathcal{X}|}$  (avec  $|\mathcal{X}|$  le cardinal de  $\mathcal{X}$ ) la matrice avec  $(P^\pi)_{ij} = \mathbb{P}(i, \pi(i), j)$ , le système d'équations peut s'écrire sous la forme

$$\begin{aligned} V^\pi &= r^\pi + \gamma P^\pi V^\pi \\ (I - \gamma P^\pi) V^\pi &= r^\pi \end{aligned}$$




donc si  $(I - \gamma P^\pi)$  est inversible on peut déterminer  $V^\pi$  directement

$$V^\pi = (I - \gamma P^\pi)^{-1} r^\pi$$

Dans l'exemple de Chat, Souris, Fromage utilisant la politique  $\pi_2$  comme ci-dessus et avec  $\gamma = 1/2$ ,  $R_{Win} = 100$  et  $R_{Lose} = -200$ . On ajoute également la condition que si la souris commence dans la même pièce que le chat ou le fromage, elle reste dans cette pièce avec une probabilité de 1.

$$r_{\pi_2} = \begin{bmatrix} 0 \\ 0 \\ 50 \\ 0 \\ -100 \\ 50 \\ 100 \\ 0 \\ -200 \\ 0 \\ 0 \\ -100 \end{bmatrix} \quad P^{\pi_2} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \end{bmatrix} \implies V^{\pi_2} = \begin{bmatrix} -73 \\ 36 \\ 109 \\ -72 \\ -218 \\ 109 \\ 200 \\ -73 \\ -400 \\ 36 \\ 36 \\ -218 \end{bmatrix}$$

En retraduisant  $V^\pi$  sur la grille, nous voyons en Figure 5 que ces valeurs sont assez intuitives. Pour la salle 2, le contrôleur effectue l'action 1 et a donc 50% de chances de trouver le fromage dans sa prochaine salle; donc une valeur attendue élevée pour le retour pour  $X_0 = 2$ . Inversement pour la pièce 11, le contrôleur prendra l'action 1 en le déplaçant dans la pièce avec le chat avec une probabilité de 50% et donc une faible valeur attendue de retour pour  $X_0 = 11$ .

0	a1	4	a1	8	a1
-73		-218		-40	
1	a2	5	a2	9	a2
36		109		36	
2	a1	6	a1	10	a1
109		200		36	
3	a2	7	a2	11	a2
-73		-73		-218	

**Légende**

x	a
	vf

x    indice d'état  
a    action  
vf    value for f

FIGURE 5 – Pour chaque pièce ; en haut à droite est le numéro de la pièce, en haut à gauche l'action pour cette salle avec la politique  $\pi_2$  et en bas à droite la valeur de la fonction score  $V^\pi$ .

### 2.3.4 La Value Function optimale

Pour résoudre le problème de la recherche de la règle de décision optimale, nous nous limitons aux politiques déterministes. Pour trouver cette politique optimale, il peut sembler tentant de lister toutes les règles de décision possibles, de calculer leur retour et de sélectionner celle avec le retour le plus grand.

Cependant, même pour un jeu simple comme Chat Souris Fromage, si nous considérons l'ensemble des règles de décisions où l'action dans chaque état est fixe, nous avons pour nos 12 cases 2 action possibles donc  $2^{12} = 4096$  comportements possibles dont il faudra calculer le retour pour chacun d'eux. Et en passant simplement à une grille 5x5 on en obtient 33554432 (ce qui fait un peu beaucoup !). Cette méthode est clairement impraticable, surtout lorsque la complexité du jeu augmente.

Dans la section 2.4, nous allons voir comment utiliser une stratégie algorithmique pour trouver cette politique optimale. Nous commencerons par définir la "fonction de valeur optimale"  $V^*$  et la "fonction de valeur d'action"  $Q^*$  que nous chercherons à atteindre avec la politique optimale.

$$Q^*(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x, a, y) V^*(y), \quad x \in \mathcal{X}, a \in \mathcal{A}$$

$$V^*(x) = \sup_{a \in \mathcal{A}} Q^*(x, a), \quad x \in \mathcal{X}$$

Comme pour les fonctions de valeur,  $V^*$  satisfait le système d'équation

$$V^*(x) = \sup_{a \in \mathcal{A}} \left\{ r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x, a, y) V^*(y) \right\}, \quad x \in \mathcal{X}$$

appelées les « Bellman Optimality equations ». Nous définissons «Bellman optimality operator» l'opérateur  $T^* : \mathbb{R}^{\mathcal{X}} \rightarrow \mathbb{R}^{\mathcal{X}}$  par

$$(T^*V)(x) = \sup_{a \in \mathcal{A}} \left\{ r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x, a, y) V(y) \right\}, \quad x \in \mathcal{X}$$

Et aussi pour action-value function  $Q^*$  avec  $T^* : \mathbb{R}^{\mathcal{X} \times \mathcal{A}} \rightarrow \mathbb{R}^{\mathcal{X} \times \mathcal{A}}$  par

$$(T^*Q)(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x, a, y) \sup_{a' \in \mathcal{A}} Q(y, a') \quad (x, a) \in \mathcal{X} \times \mathcal{A}$$

Si une politique atteint la valeur optimale pour l'état, dans tous les états, alors la politique est optimal.

Alors nous avons que  $V^*$  et  $Q^*$  sont des solutions des équations  $T^*V^* = V^*$  et  $T^*Q^* = Q^*$  respectivement, un résultat important de avec nous baserons les «Dyanmic Programming Algorithms» pour trouver la politique optimal.

## 2.4 Dynamic Programming Algorithms for solving MDPs

On définit les suites

$$\begin{aligned} V_{k+1} &= T^*V_k, \quad k \geq 0 \\ Q_{k+1} &= T^*Q_k, \quad k \geq 0 \end{aligned}$$

ou plus explicitement

$$\begin{aligned} V_{k+1} &= \sup_{a \in \mathcal{A}} \left\{ r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x, a, y) (V_k)_y \right\}, \quad x \in \mathcal{X} \quad k \geq 0 \\ Q_{k+1} &= r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathcal{P}(x, a, y) \sup_{a' \in \mathcal{A}} (Q_k)_{y, a'} \quad (x, a) \in \mathcal{X} \times \mathcal{A} \end{aligned}$$

où  $(V_k)_y$  est la composante y du vecteur  $V_k$  qui est égale à la fonction valeur pour  $X_0 = y$  à la itération k et  $(Q_k)_{y, a'}$  l'élément de la ligne y et de la colonne  $a'$  de la matrice  $Q_k$ .  $Q_k$  a une colonne pour chaque action et une ligne pour chaque état et les éléments sont égaux à la fonction de valeur pour cette action, dans cet état à la itération k.

On peut montrer que ces suites  $V_k$  et  $Q_k$  convergent vers  $V^*$  et  $Q^*$  grâce au théorème de Banach. [1]

Pour mieux comprendre cette itération nous allons encore l'appliquer à l'exemple du jeu chat, souris, fromage avec  $\gamma = \frac{1}{2}$ ,  $R_{Win} = 100$  et  $R_{Lose} = -200$ .

On prend pour  $V_0$  un vecteur nul de taille 12 et on applique l'algorithme. En transformant chaque itération en une grille, nous pouvons voir en Figure 6 comment l'impact de la récompense se diffuse. Nous observons que l'algorithme atteint un point fixe à 8 décimales après 40 itérations et nous avons donc  $V^* \approx V_{40}$ . Ainsi, pour une politique optimale  $\pi^*$ , nous

$V_0$	$V_1$	$V_2$	$V_3$																																																
<table><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	0	0	0	0	<table><tr><td>0</td><td>0</td><td>-200</td></tr><tr><td>0</td><td>50</td><td>0</td></tr><tr><td>50</td><td>100</td><td>0</td></tr><tr><td>0</td><td>50</td><td>0</td></tr></table>	0	0	-200	0	50	0	50	100	0	0	50	0	<table><tr><td>0</td><td>13</td><td>-300</td></tr><tr><td>13</td><td>75</td><td>25</td></tr><tr><td>75</td><td>150</td><td>75</td></tr><tr><td>25</td><td>75</td><td>13</td></tr></table>	0	13	-300	13	75	25	75	150	75	25	75	13	<table><tr><td>9</td><td>19</td><td>-350</td></tr><tr><td>25</td><td>91</td><td>38</td></tr><tr><td>91</td><td>175</td><td>91</td></tr><tr><td>38</td><td>91</td><td>25</td></tr></table>	9	19	-350	25	91	38	91	175	91	38	91	25
0	0	0																																																	
0	0	0																																																	
0	0	0																																																	
0	0	0																																																	
0	0	-200																																																	
0	50	0																																																	
50	100	0																																																	
0	50	0																																																	
0	13	-300																																																	
13	75	25																																																	
75	150	75																																																	
25	75	13																																																	
9	19	-350																																																	
25	91	38																																																	
91	175	91																																																	
38	91	25																																																	

FIGURE 6 – Itérations  $V_0, V_1, V_2, V_3$  de Dynamic Programming Algorithm

devrions nous attendre à trouver  $V^{\pi^*} = V^*$  la fonction valeur de  $\pi^*$ . On a

$$V^* \approx V_{40} = \begin{bmatrix} 22 \\ 41 \\ 110 \\ 55 \\ 33 \\ 110 \\ 200 \\ 110 \\ -400 \\ 55 \\ 110 \\ 41 \end{bmatrix}$$

Nous devons encore déterminer la politique optimale  $\pi^*$ . Pour ce faire, nous utiliserons plutôt le Dynamic Programming Algorithm pour  $Q_k$  qui rappelle nous attribue une valeur en prenant en compte la case où la souris se trouve mais aussi l'action qu'elle a choisie, ce qui est très utile pour déterminer la meilleure action à prendre sur chaque case. On prend pour  $Q_0$  une matrice nulle de taille  $12 \times 2$ . La première colonne représentera l'action 1 et la deuxième action 2. La ligne  $i$  représentera l'élément  $(i - 1)$  dans notre espace d'état  $\mathcal{X}$  (étant donné les étiquettes de notre  $x \in \mathcal{X}$  commence à partir de 0).

Après 40 itérations, la fonction de valeur optimale  $Q_{40}$  est calculée pour chaque état et action possible dans la grille du jeu. Pour trouver la politique optimale  $\pi^*$ , nous choisissons pour chaque état dans la grille  $\mathcal{X}$  la meilleure action qui maximise la fonction de valeur. Si la valeur de la colonne 1 dans une ligne est supérieure à celle de la colonne 2, alors l'action 1 est préférable, sinon l'action 2. En appliquant ce choix pour chacun des 12 états, nous obtenons

notre politique optimale.

$$Q_{40} = \begin{bmatrix} 22 & -190 \\ 33 & 41 \\ 110 & 41 \\ 55 & 15 \\ -172 & 33 \\ 22 & 110 \\ 200 & 200 \\ 110 & 22 \\ -400 & -400 \\ -190 & 55 \\ 41 & 110 \\ 41 & -172 \end{bmatrix} \Rightarrow \begin{array}{c} \text{Action préférable} \\ \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 2 \\ 2 \\ \text{les deux} \\ 1 \\ \text{les deux} \\ 2 \\ 2 \\ 1 \end{bmatrix} \end{array}$$

Nous pouvons donc définir notre politique optimale  $\pi^*$  par

$$\pi^*(x) = \begin{cases} \text{Action 1} & \forall x \in \{0, 2, 3, 6, 7, 8, 11\} \\ \text{Action 2} & \forall x \in \{1, 4, 5, 9, 10\} \end{cases}$$

Maintenant on vérifie que cette politique est optimale en déterminant  $V^{\pi^*}$  avec la même méthode que précédemment pour déterminer  $V^{\pi^2}$ . On a

$$r_{\pi^*} = \begin{bmatrix} 0 \\ 0 \\ 50 \\ 0 \\ 0 \\ 50 \\ 100 \\ 50 \\ -200 \\ 0 \\ 50 \\ 0 \end{bmatrix} \quad P^{\pi^*} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix} \Rightarrow V^{\pi^*} = \begin{bmatrix} 22 \\ 41 \\ 110 \\ 55 \\ 33 \\ 110 \\ 200 \\ 110 \\ -400 \\ 55 \\ 110 \\ 41 \end{bmatrix}$$

comme  $V^{\pi^*} = (I - \gamma P^{\pi^*})^{-1} r_{\pi^*}$ .

On a que  $V^* = V^{\pi^*}$  qui implique  $\pi^*$  est bien la politique optimal. Le contrôleur maximisera ses chances d'amener la souris vers le fromage tout en évitant le chat s'il effectue l'action dans chaque pièce suggérée par  $\pi^*$ .

## 3 Temporal Difference Learning dans des espaces finis

Toujours en suivant [1].

### 3.1 Motivation

Le défi de trouver la fonction de valeur d'un MRP comme dans la section précédente est un cas spécifique d'un problème de prédiction de valeur. Plus généralement, ces problèmes peuvent inclure la prévision des conditions météorologiques, des marchés boursiers de la probabilité qu'une coupure de courant se produise. Une méthode explicite telle que celle vue dans la section précédente n'est pas toujours disponible et nous utilisons donc à la place une estimation.

Dans le cadre des MRP, la méthode Monte Carlo est une technique d'estimation de la fonction de valeur qui consiste à effectuer plusieurs réalisations à partir d'un état donné et de calculer la moyenne des rendements obtenus. Cette méthode est simple à comprendre et à mettre en œuvre, mais elle présente des limites. Tout d'abord, elle nécessite de pouvoir réinitialiser le système, ce qui peut être impossible dans certains cas. De plus, elle ne permet pas de mettre à jour les estimations de la fonction de valeur en temps réel, ce qui peut rendre l'apprentissage plus lent.

Pour résoudre ces problèmes, la méthode «Temporal Difference Learning» (TDL) a été développée. L'idée est de mettre à jour les estimations de la fonction valeur à chaque étape d'apprentissage, plutôt qu'à la fin de chaque réalisation, comme c'est le cas avec la méthode de Monte Carlo. De cette façon, TDL peut être utilisé pour les systèmes où la réinitialisation est impossible et permet la mise à jour en temps réel des estimations de la fonction de valeur.

### 3.2 Monte Carlo Method

La méthode de Monte Carlo est utile dans les systèmes «épisodiques», comme ils peuvent être redémarrés à partir du même état exact dans les mêmes conditions. Nous calculons une estimation pour la fonction de valeur d'état  $x$  en générant plusieurs MRP qui commencent à partir de  $X_0 = x$  à chaque instant. On calcule le rendement de chacun de ces MRP, puis la moyenne des rendements. Cette moyenne sera alors l'estimation de la fonction de valeur.

Pour observer cet algorithme, nous regardons à nouveau le jeu Chat, Souris, Fromage. Ici, nous pouvons définir un «épisode» comme  $X_0 = x$  pour n'importe quel état  $x$ , jusqu'à  $X_t$  ce que le Chat ou le Fromage soit trouvé. Pour simuler un MRP, nous devons sélectionner une politique, ici nous utiliserons la politique  $\pi_1$  de la Section 2.3.1 pour plus de simplicité.

On note que le Return de chaque episode de longueur  $T$  est  $R_{episode}$  égal à

$$R_{episode} = \gamma^{T-1}(\mathbb{1}_{X_T=X_{Chat}}R_{Lose} + \mathbb{1}_{X_T=X_{Fromage}}R_{Win})$$



On simule 5 épisodes qui commencent à partir de  $X_0 = 0$  et on calcul leur return.

$$\begin{aligned}
X_{ep1} &= \begin{bmatrix} 0 \\ 4 \\ 7 \\ 11 \\ 10 \\ 2 \\ 6 \end{bmatrix} & R_{ep1} &= 1.5625 & X_{ep2} &= \begin{bmatrix} 0 \\ 4 \\ 8 \end{bmatrix} & R_{ep2} &= -50 & X_{ep3} &= \begin{bmatrix} 0 \\ 3 \\ 7 \\ 6 \end{bmatrix} & R_{ep3} &= 12.5 \\
X_{ep4} &= \begin{bmatrix} 0 \\ 3 \\ 7 \\ 6 \end{bmatrix} & R_{ep4} &= 12.5 & X_{ep5} &= \begin{bmatrix} 0 \\ 3 \\ 1 \\ 2 \\ 1 \\ 0 \\ 4 \\ 7 \\ 6 \end{bmatrix} & R_{ep5} &= 0.78125
\end{aligned}$$

Dans cette exemple notre estimation pour  $V(5) = \text{mean}(R_{ep1}, R_{ep2}, R_{ep3}, R_{ep4}, R_{ep5}) = 4.5312$ . Avec seulement 5 épisodes, notre estimation est encore loin de la vraie valeur. Si nous prenons 50 épisodes pour chaque état, nous trouvons une estimation plus précise pour la fonction de valeur de notre MRP. Nous avons transformé cette MRP en grille et arrondi les valeurs à l'entier le plus proche pour plus de clarté.

$$V = \begin{bmatrix} 5 & 10 & -200 \\ 9 & 26 & 10 \\ 34 & 100 & 22 \\ 15 & 26 & 11 \end{bmatrix}$$

Comme on vient de le voir la méthode de Monte Carlo permet de calculer la fonction de valeur pour les systèmes épisodiques, où l'on peut redémarrer le processus dans les mêmes conditions à chaque fois. Cependant, cette méthode présente des limites pour les systèmes non-épisodiques où il est impossible de réinitialiser le processus.

Pour résoudre ce problème, nous introduisons la méthode de Temporal Difference (TD), une approche qui permet d'estimer la fonction de valeur en utilisant les transitions d'un seul épisode. Dans la prochaine sections, nous allons explorer les algorithmes de TD pour les processus de décision de Markov et voir comment ils peuvent être utilisés pour estimer la fonction de valeur dans les systèmes non-épisodiques.

### 3.3 Algorithme TD(0)

L'algorithme TD(0) est l'un des algorithmes de Temporal Difference les plus simples qui permet d'estimer la fonction de valeur de certains MRP à partir d'une seule réalisation  $(X_t, \mathcal{R}_{t+1})$ ;  $t \geq 0$ . Initialement, une valeur  $V_0$  est attribuée à chaque état. À chaque itération  $t$ , l'algorithme met à jour uniquement la valeur de l'état  $x$  correspondant à  $X_t = x$  qui vient

d'être visité. Lors du passage de  $X_t$  à  $X_{t+1}$ , l'algorithme collecte la récompense  $R_{t+1}$  associée à la transition ainsi que la valeur  $V(X_{t+1})$  de l'état suivant. La différence entre la somme de la récompense et de la valeur de l'état suivant, et la valeur actuelle de l'état  $X_t$  est l'erreur de différence temporelle  $\delta_{t+1}$ .

$$\begin{aligned}\delta_{t+1} &= \mathcal{R}_{t+1} + \gamma V_t(X_{t+1}) - V_t(X_t) \\ V_{t+1}(x) &= V_t(x) + \alpha_t \delta_{t+1} \mathbb{1}_{\{X_t=x\}} \quad \forall x \in \mathcal{X}\end{aligned}$$

Dans [1] on voit que cet algorithme converge vers un unique  $V^*$ , la fonction valeur du MRP.

Encore une fois, nous appliquerons cet algorithme au jeu Chat, Souris, Fromage, mais cette fois, nous verrons qu'il y a quelques problèmes. Tout d'abord, le jeu se termine une fois que la souris a trouvé le chat ou le fromage, ce qui signifie que le MRP peut être très court. Dans les cas les plus extrêmes, juste  $X_0$ , à partir duquel nous ne sommes pas en mesure de calculer une fonction de valeur significative. Nous constatons également que la fonction de valeur ne sera mise à jour qu'une seule fois, le tour qui trouve le chat ou le fromage signifiant que toute la récompense sera attribuée uniquement à cet état.

Comme solution à cela, nous adapterons Chat, Souris, Fromage afin que notre MRP, qui est une chaîne de Markov, ait les propriétés pratiques d'être irréductible, apériodique et récurrente positive. (Comme ces propriétés sont nécessaires pour la convergence de TD(0)). Nous adaptons le jeu pour chaque propriété par :

- Irréductible - C'est possible avec les actions disponibles mais nous devons nous assurer que la politique choisie le permet, par exemple  $\pi_1$ .
- Appériodique - Nous adapterons les deux actions pour inclure une troisième possibilité, rester dans la même pièce.
- Récurrent Positif - Chaque fois que le chat ou le fromage est trouvé, il réapparaît dans une pièce différente. Nous limiterons le chat aux 6 premières pièces et le fromage au bas. Dans le but de pouvoir observer cette différence de  $V$ .

Pour simplifier le MRP, nous choisirons la politique  $\pi_1$  car il est clair que cela donne la propriété d'irréductibilité. Pour utiliser l'algorithme, nous simulons d'abord le MRP, le jeu utilisant notre politique choisie. Cela nous donne deux vecteurs, un pour  $X_t$ , la pièce dans laquelle se trouve la souris au temps  $t$  et l'autre pour  $\mathcal{R}_{t+1}$  la récompense qu'il reçoit lorsqu'il se déplace dans la prochaine pièce à  $t + 1$ . Cela ressemblera à ce qui suit :

$$X = \begin{pmatrix} 6 \\ 6 \\ 10 \\ 10 \\ 9 \\ 1 \\ 0 \\ 3 \\ \dots \end{pmatrix} \quad \mathcal{R} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 100 \\ 0 \\ \dots \end{pmatrix}$$

Comme  $\mathcal{R}_8 = 100$  nous pouvons déterminer qu'en passant de  $X_7 = 0$  à  $X_8 = 3$  la souris a trouvé le fromage.

Nous appliquons 500 itérations de l'algorithme TD(0) qui produit (arrondi à l'entier le plus proche)

$$V_{500} = \begin{bmatrix} 6 & -10 & 1 \\ -4 & -8 & -10 \\ -2 & 5 & 0 \\ 1 & 1 & 3 \end{bmatrix}$$

Ce processus est stochastique, la souris fait à chaque fois un choix aléatoire de 3 mouvements et la position du chat et du fromage sont également choisis au hasard. Le calcul de cette fonction valeur à chaque simulation ne produira donc pas le même résultat.

Cependant, nous pouvons voir que même dans ce petit exemple, les valeurs suivent une certaine intuition. Dans la deuxième rangée de pièces, quelle que soit l'action entreprise par la souris, elle restera dans la région où se trouve le chat (pour rappel, nous avons limité le chat aux 6 pièces du haut et la souris aux 6 du bas). Il n'est donc pas surprenant que dans cette ligne nous observions les valeurs les plus basses. De même, dans la rangée du bas, l'action déplacera également la souris vers une pièce qui se trouve toujours dans la région du fromage et on s'attend donc à voir des valeurs plus élevées ici. Pour les première et troisième rangées, l'action 1 peut soit déplacer la souris dans l'une ou l'autre zone. Dans le premier il a deux fois plus de chances de rester dans la zone du chat et dans le troisième celle du fromage d'où l'on trouve les valeurs du rang 1 inférieures à celle du rang 3.

### 3.4 TD( $\lambda$ )

Ces deux méthodes Monte Carlo et TD ont toutes deux leur utilité, mais il existe des systèmes où l'une est préférable à l'autre (comme indiqué ci-dessus). Une troisième option pour calculer une fonction de valeur est l'algorithme TD( $\lambda$ ) qui peut être considéré comme une combinaison de Monte Carlo et TD(0).  $\lambda \in [0, 1]$  avec TD( $\lambda = 0$ ) l'algorithme TD(0) vu dans la Section 3.3 et TD( $\lambda = 1$ ) un method de Monte Carlo.

Pour illustrer cette étude, les étapes d'une itération de l'algorithme de plus près.

$$\begin{aligned} \delta_{t+1} &= \mathcal{R}_{t+1} + \gamma V_t(X_{t+1}) - V_t(X_t), \\ z_{t+1}(x) &= \mathbb{1}_{\{x=X_t\}} + \gamma \lambda z_{t+1}(x) \\ V_{t+1}(x) &= V_t(x) + \alpha_t \delta_{t+1} z_{t+1}(x) \\ z_0(x) &= 0, \\ &\forall x \in \mathcal{X} \end{aligned}$$

$\delta_{t+1}$  est l'erreur de temporal difference comme précédemment pour TD(0). On voit que si  $\lambda = 0$ , ce nouveau terme  $z_{t+1}$  devient l'indicateur  $\mathbb{1}_{\{x=X_t\}}$  et on a le même algorithme que TD(0). Lorsque  $\lambda \neq 0$ , le  $z_{t+1}$  qui ressemble à l'étape d'une méthode de Monte Carlo influencera également l'algorithme. En faisant varier la valeur de  $\lambda$ , il est possible de trouver la manière la plus efficace de combiner les deux algorithmes en termes de vitesse de convergence (vois [1]). Nous pouvons appliquer TD( $\lambda$ ) à la deuxième version, plus chaotique, de Chat, Souris, Fromage comme voir dans la section 3.3. Notez que la méthode de Monte Carlo seule ne

pouvait pas être utilisée sur ce problème, mais grâce au nouvel algorithme, nous pouvons maintenant utiliser la stratégie qui la sous-tend. On utilise la même politique (qui prend l'action 1 partout) et on calcule  $V$  pour notre MRP avec l'algorithme  $TD(\frac{1}{2})$ . Nous trouvons (arrondi à l'entier le plus proche et transformé en grille)

$$V = \begin{bmatrix} 13 & 34 & 8 \\ -59 & -38 & -109 \\ -7 & -20 & 38 \\ 15 & 62 & 19 \end{bmatrix}$$

On observe comme à la section précédente que les valeurs les plus élevées sont concentrées dans la ligne du bas, où l'action 1 maintiendra la souris dans la zone du fromage et les valeurs les plus basses dans la ligne 3 où l'action 1 maintiendra la souris dans la zone du chat .

## 4 Pour aller plus loin, algorithmes pour grands espaces d'états

Alors que le deuxième exemple du jeu Chat, Souris, Fromage est beaucoup plus chaotique, il est toujours joué dans un espace d'états fini (et très petit), ce qui signifie que le stockage d'une valeur pour chaque état est possible facilement dans une mémoire d'ordinateur. Ce sont des algorithmes tabulaires :

$Q(s, a)$	$s_1$	$s_2$	$\dots$	$s_n$
$a_1$	0.1	0.3	$\dots$	0.2
$a_2$	0.4	0.1	$\dots$	0.4
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$a_m$	0.3	0.2	$\dots$	0.2

FIGURE 7 – Représentation tabulaire des algorithmes vu pour l'instant avec  $S$  l'ensemble d'état et  $A$  l'ensemble des actions.

Cependant, les problèmes où une fonction de valeur doit être trouvée ne sont clairement pas limités aux systèmes à espaces discret ou finis (ou très grand). Des méthodes distinctes paramétriques et non paramétriques, sont nécessaires pour la prédiction de valeur dans de tels systèmes. L'apprentissage de la fonction valeur devient un problème d'approximation de fonction.

Des exemples de méthodes non paramétriques comprennent les méthodes k-plus proches voisins et le lissage non paramétrique du noyau. Ces méthodes sont plus facilement adaptables mais peuvent être beaucoup plus complexes en termes de calcul.

Dans le cas paramétrique comme on ne veut pas trouver directement la valeur de  $x$  on étudie plutôt certaines «features» de  $x$  et on met une valeur sur ces features. Pour représenter cela, nous définissons

$$V_\theta(x) = \theta^\top \phi(x), \quad x \in \mathcal{X}$$

où  $V_\theta$  est la valeur à estimer,  $\theta \in \mathbb{R}^d$  un vecteur de paramètres et  $\phi : \mathcal{X} \rightarrow \mathbb{R}^d$  un mappage des états sur un vecteur d-dimensionnel. Ce  $(\phi(x))_i$  représente une caractéristique ' $i$ ' de  $x$ . Des algorithmes de similarité comme dans le cas des espaces finis, comme TD( $\lambda$ ), sont ensuite utilisés pour estimer la valeur de ces caractéristiques de  $x$ .

Pour approfondir notre compréhension des algorithmes d'apprentissage par renforcement, nous aimerions étudier ces techniques plus en détail et les appliquer à un certain système avec des espaces infinis.

## Références

- [1] Csaba Szepesvári ; Algorithms for Reinforcement Learning ; Morgan & Claypool Publishers : 2010
- [2] Ahilan, Sanjeevan ; A Succinct Summary of Reinforcement Learning ; January 2023
- [3] Pierre Gérard : Apprentissage par Renforcement Apprentissage Numérique :January 2023

$$\pi_1(A1, S0) = \pi_1(A2, S0) = \pi_1(3, S0) = \frac{1}{3}$$