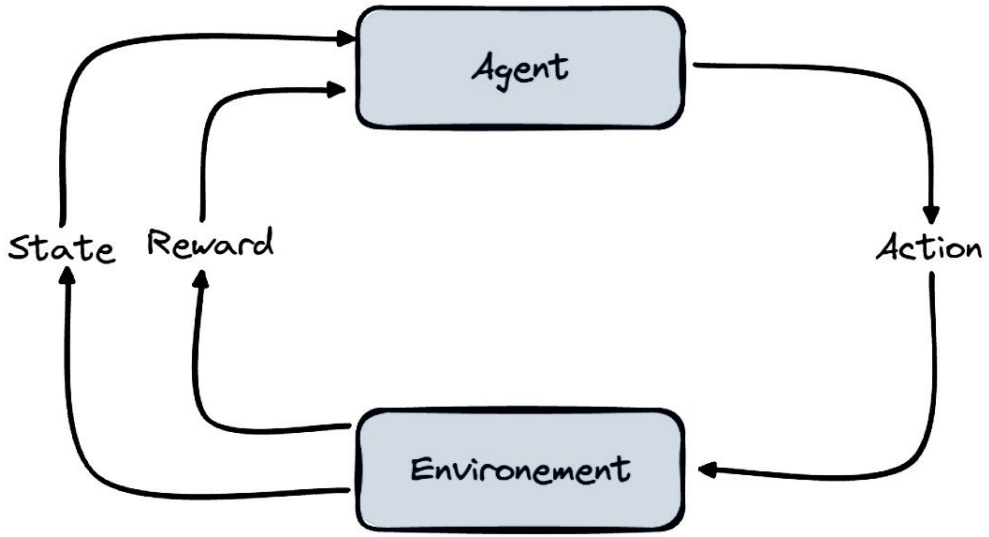
Apprentissage par renforcement

Dans ce chapitre, vous allez apprendre à développer des modèles d'apprentissage par renforcement. En machine learning, l'apprentissage par renforcement est une branche très différente de celles de l'apprentissage supervisé et non supervisé. En effet, dans cette discipline, on ne fournit pas de données (X, y) à notre machine, mais on la programme afin qu'elle puisse générer ses propres expériences et apprendre de ses erreurs.

L'idée centrale est donc de créer un **agent**, libre d'entreprendre des **actions** dans un **environnement**. Ces actions modifient l'**état** (*state*) de l'agent, et ce changement d'état s'accompagne d'une **récompense** (*reward*). Pour l'agent, le but du jeu est de maximiser ses récompenses, ce qui le pousse à apprendre quelles actions effectuer pour obtenir le plus de récompenses.



Cette discipline est très utilisée pour les applications suivantes :

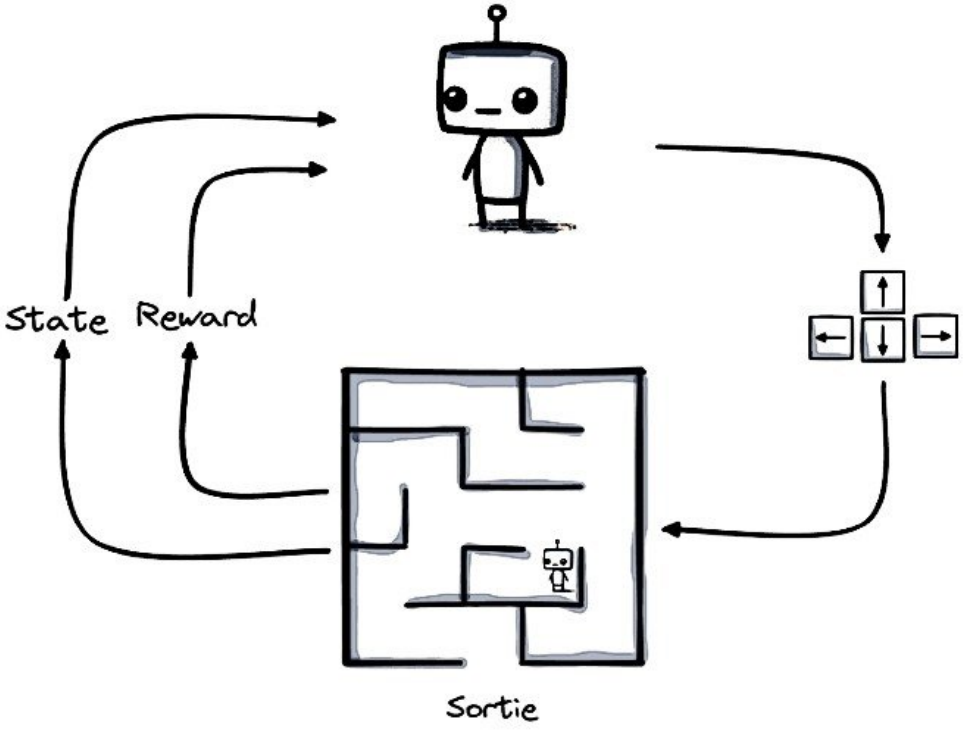
- Robotique
- Véhicules autonomes (voitures, drones)
- Trading
- Algorithmes de prise de décision

Mise en situation

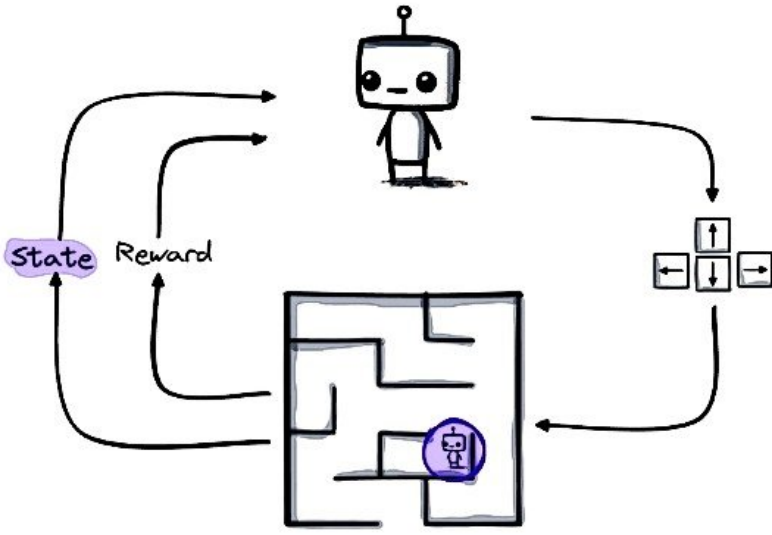
Imaginez : vous programmez un petit robot pour qu'il apprenne à sortir d'un labyrinthe le plus rapidement possible. Pour rendre les choses plus compliquées, le labyrinthe est généré aléatoirement.

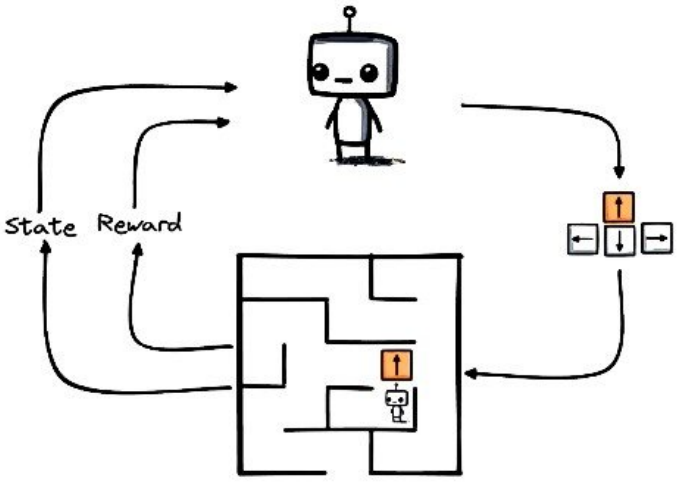
Bien sûr, nous pourrions pour cela construire un algorithme de recherche du chemin le plus court basé sur la théorie des graphes, mais cela reviendrait à coder explicitement le comportement de la machine. Et c'est là tout l'inverse du machine learning. À la place, nous voulons programmer la machine pour qu'elle développe elle-même la stratégie lui permettant de quitter le labyrinthe.

Nous allons donc utiliser une approche d'apprentissage par renforcement. Notre agent sera le petit robot, et le labyrinthe l'environnement dans lequel il évolue.



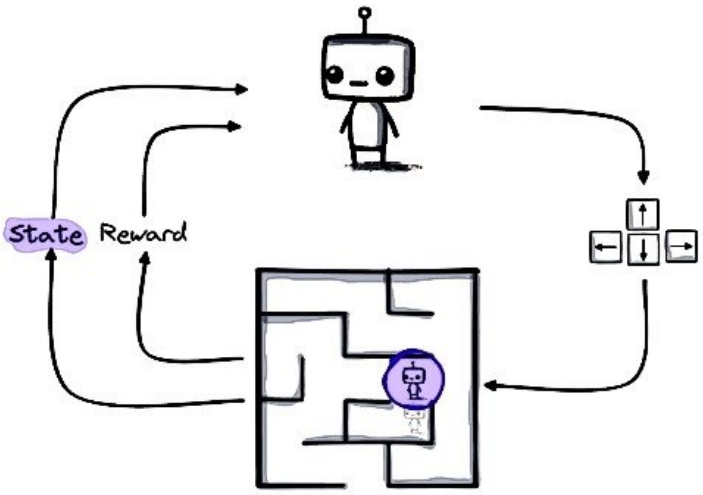
L'état (*State*) correspond à la position de l'agent dans le labyrinthe, et les actions à ses déplacements possibles (haut, bas, droite, gauche)



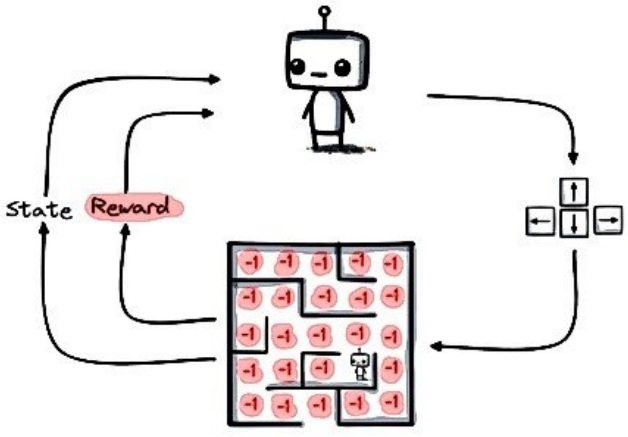


Depuis cet état, l'agent peut choisir une action, par exemple "aller en haut".

En effectuant cette action, l'agent change son état, ce qui s'accompagne d'une récompense. Ici, nous attribuons à la machine un score de -1 point.



Quoi ?! Mais l'agent s'est pourtant rapproché de la sortie, pourquoi le sanctionner avec un score négatif ? C'est vrai, cependant, rappelez-vous de notre objectif : "trouver la sortie le plus vite possible".



Ainsi, nous infligeons un point négatif chaque fois que la machine effectue un déplacement. Son but étant de maximiser son score final, elle cherchera donc le moyen le plus rapide de rejoindre la sortie, préférant marquer -10 points plutôt que -15. Ne vous en faites pas, la machine ne sera pas triste ☺

En résumé, notre objectif est d'apprendre à la machine, *quelle action effectuer lorsqu'elle se situe dans un état donné*. Autrement dit, nous cherchons à développer une fonction $f(s) = a$ qui prend en entrée l'état s et produit une action a. Cette fonction est couramment nommée ***politique d'action***.

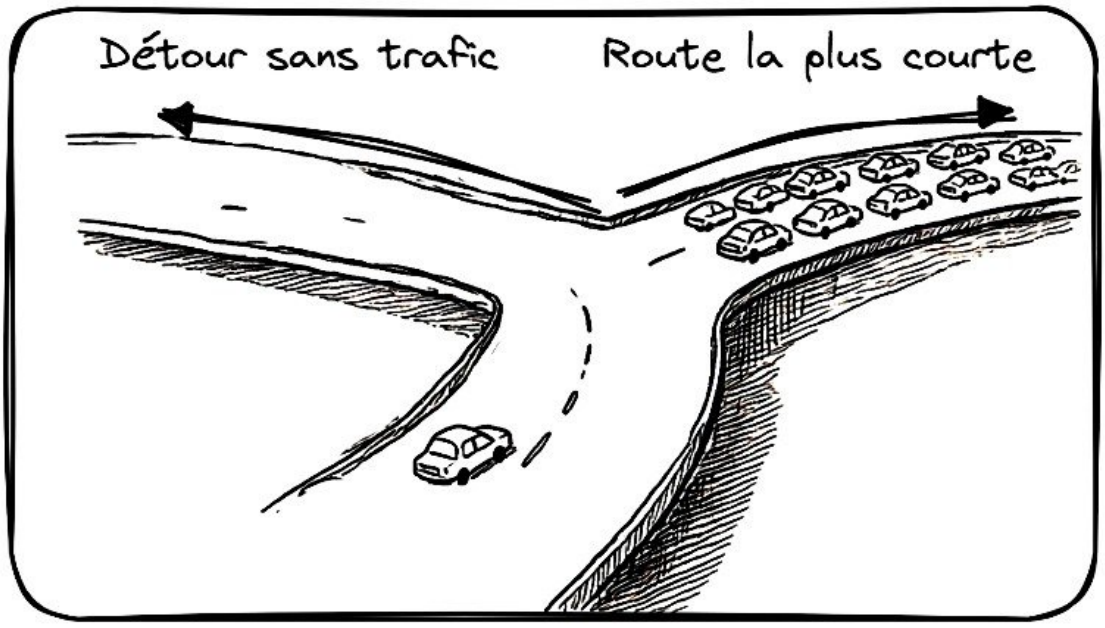
En apprentissage par renforcement, il existe plusieurs méthodes permettant d'apprendre à développer cette politique d'action. Celle que nous allons voir ici est l'approche appelée **Q-Learning**.

Le fonctionnement du Q-Learning

Le Q-Learning consiste à apprendre quelle est la **valeur** d'une action A dans un état S donné.

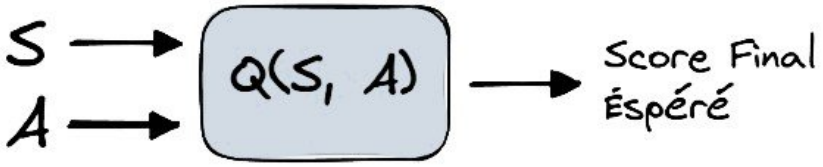
Par exemple, je mets habituellement 30 minutes en voiture pour aller à mon travail. Aujourd'hui, cette route est encombrée par le trafic. J'aperçois cependant qu'une autre route, un peu plus longue, est également possible. Ainsi, deux choix s'offrent à moi : l'action de tourner à gauche ou l'action de tourner à droite.

Pour choisir laquelle emprunter, je vais évaluer la valeur de chacune. À mon avis, quel sera le temps de trajet total si je tourne à gauche ? Et si je tourne à droite ?

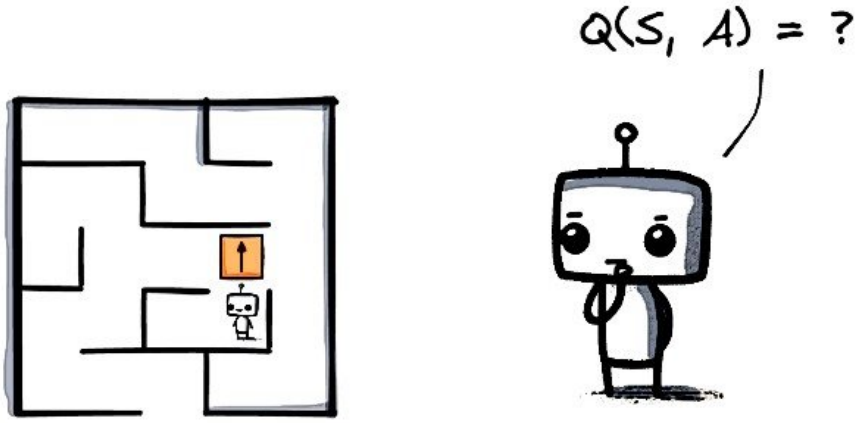


Voilà le but du Q-Learning : apprendre à prédire la valeur de mes actions, en tenant compte de la situation dans laquelle je me trouve.

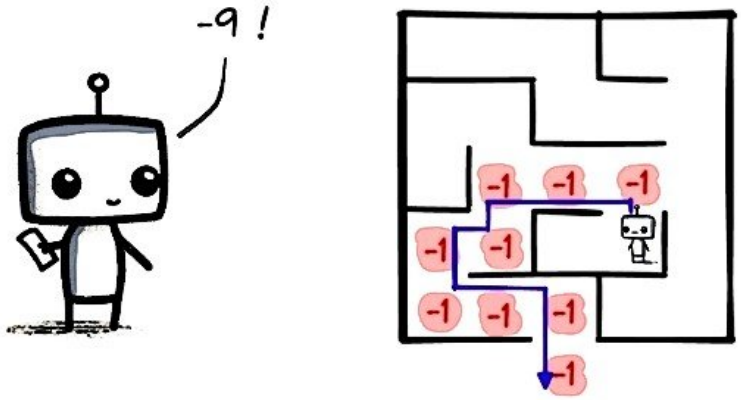
Plus formellement, il s'agit donc d'apprendre une fonction Q qui prend en entrée un état S et une action A, et qui prédit le **score final** que l'on peut espérer obtenir si tout se passe pour le mieux par la suite.



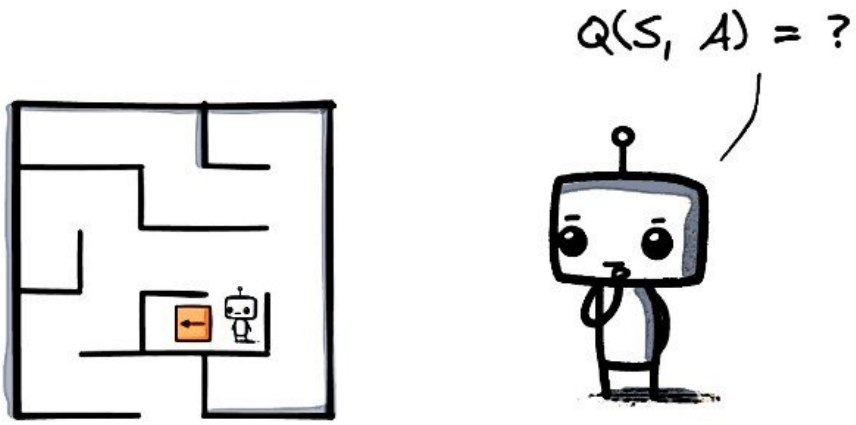
Prenons un exemple : dans la position actuelle, quel est le meilleur score que la machine puisse espérer obtenir si elle choisit l'action "allez en haut"?



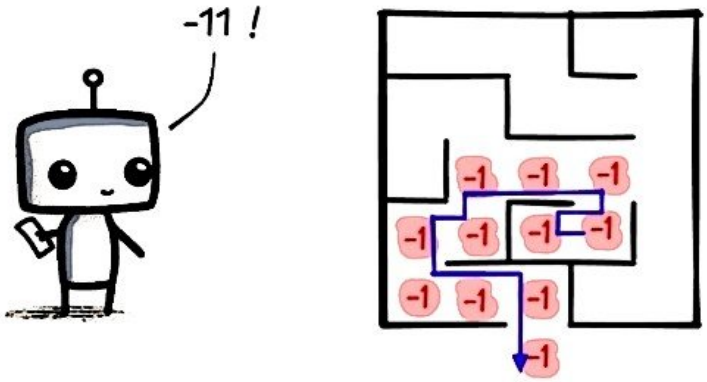
En tenant compte du fait que la machine perd un point pour chaque déplacement, le meilleur score qu'elle puisse espérer obtenir est -9.



Et dans le cas où la machine choisit d'aller à gauche?



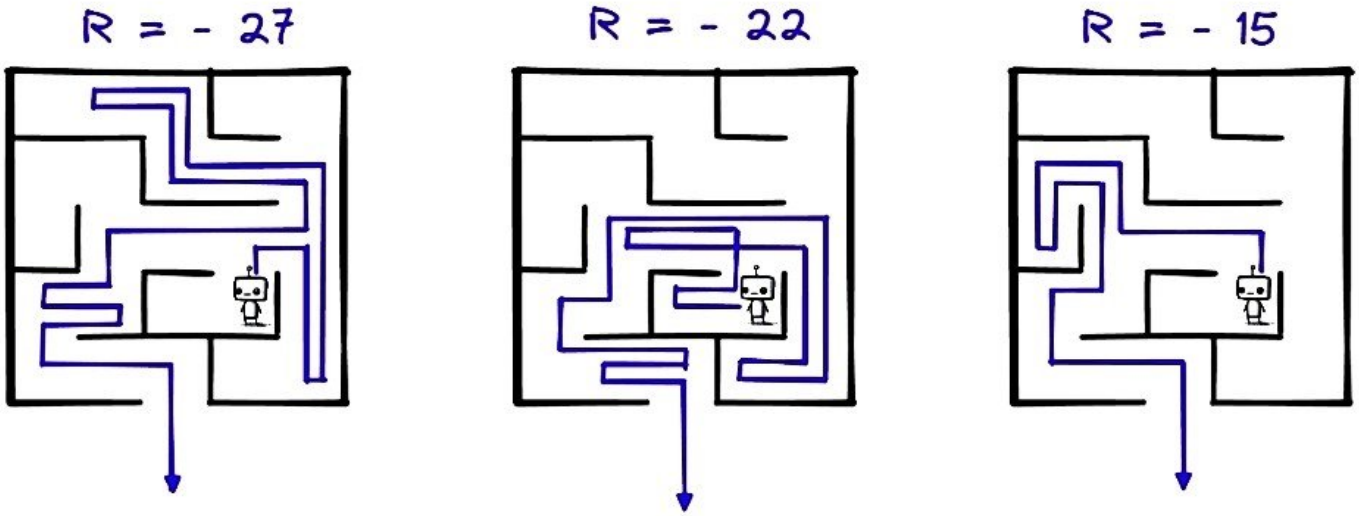
... Cette fois-ci, la réponse est -11, car la machine devra revenir sur son chemin. Ainsi, il est préférable de choisir l'action "aller en haut" car elle donne un score final plus élevé (-9) que l'action "aller à gauche" (-11)



Il reste maintenant une question : Comment apprendre cette fonction Q?

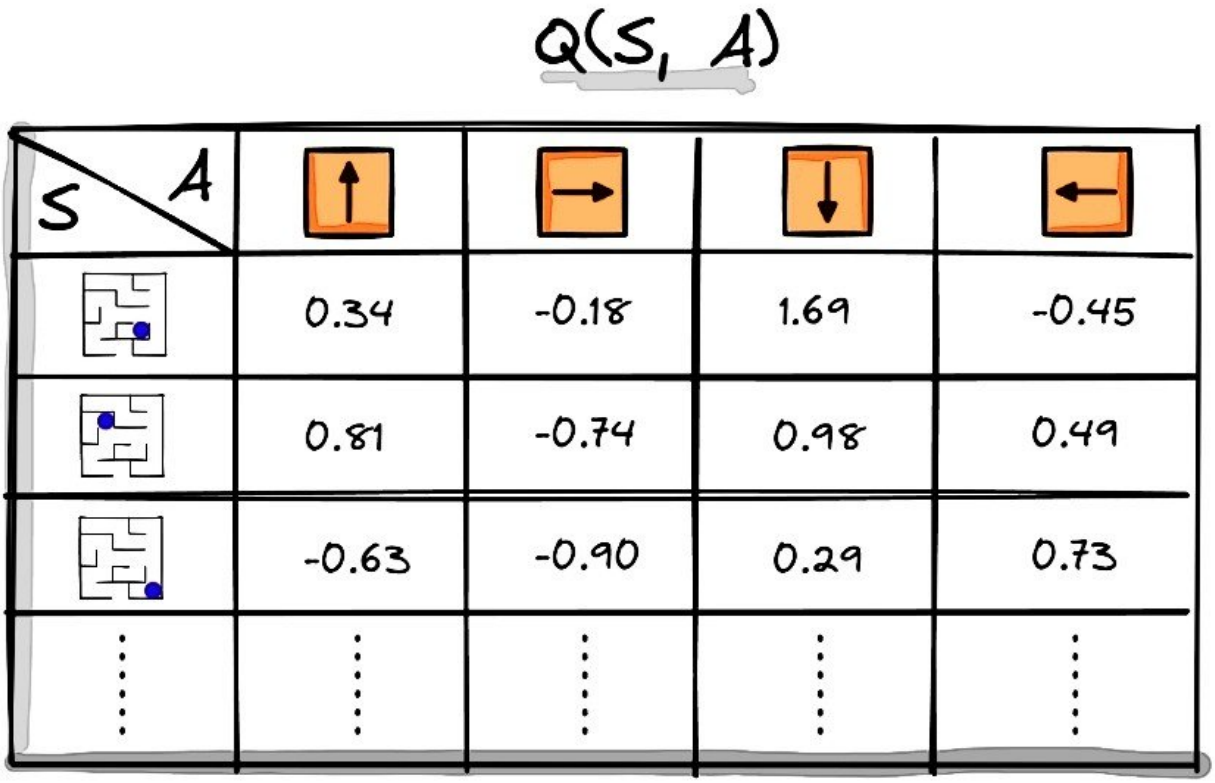
L'apprentissage de la fonction Q

Pour apprendre à prédire la valeur de ses actions, la machine doit explorer son environnement en s'y déplaçant au hasard, afin de générer des données (S, A, R) (état, action, récompense).

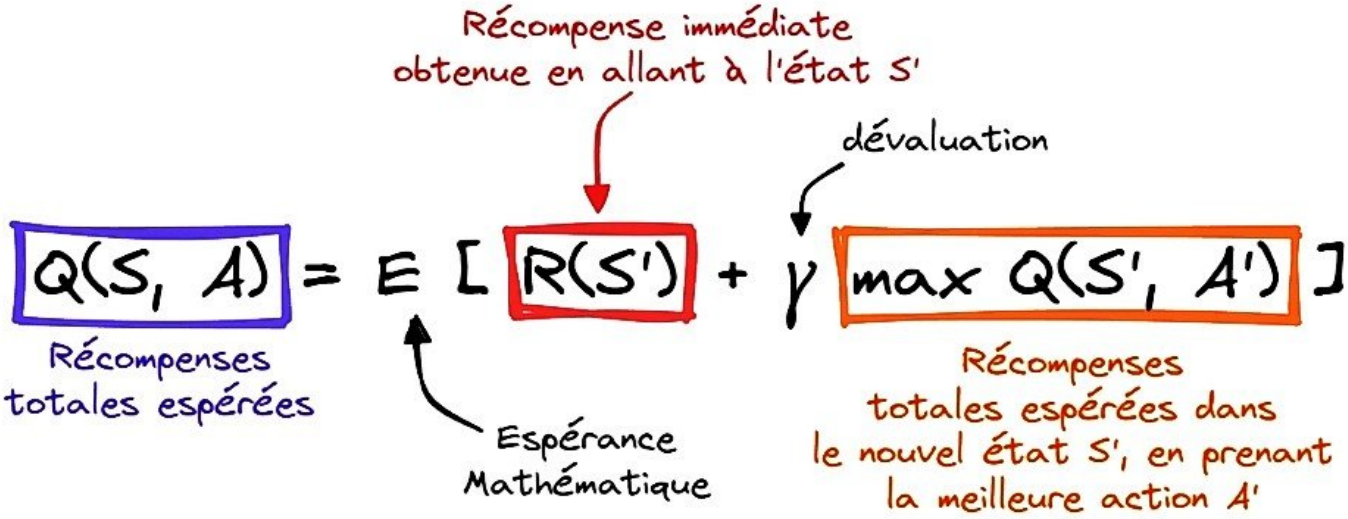


Ces données sont ensuite utilisées pour mettre à jour un tableau Q(S, A) qui informe la machine des récompenses qu'elle est censée obtenir à l'avenir en choisissant telle ou telle action dans l'état présent.

Au début de son apprentissage, ce tableau est rempli de valeurs aléatoires, comme illustré ci-dessous.



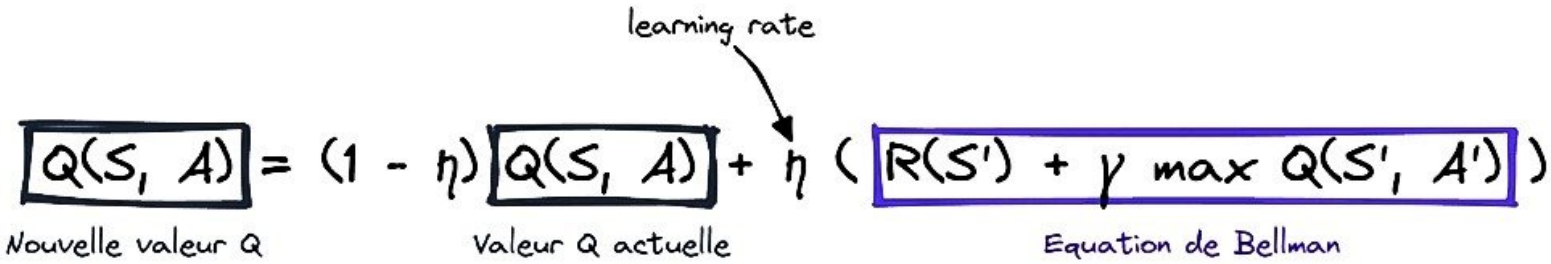
Une formule, connue sous le nom de **l'équation de Bellman**, permet d'exprimer le score total que l'on peut espérer obtenir à l'avenir. Ce score est égal à la récompense obtenue immédiatement dans l'état S, à laquelle on ajoute le même score pour l'état suivant S', si l'on choisit la meilleure action possible dans ce nouvel état.



Pour bien comprendre, reprenons l'analogie avec la voiture.

L'équation de Bellman vous permet d'estimer que la valeur de tourner à gauche vous donne la récompense immédiate d'éviter le trafic, en plus de la valeur que vous allez obtenir dans votre prochain état. Si vous tombez sur des bouchons 1 km plus loin, le fait de tourner à gauche n'était peut-être pas la meilleure option.

À chaque fois que la machine réalise une action, elle peut mettre à jour sa table Q, car elle reçoit une récompense associée à son changement d'état. D'après la formule de Bellman, cette récompense R(S') peut être utilisée pour recalculer la valeur de Q(S,A) en se basant sur sa valeur actuelle et cette nouvelle donnée :



Exploration et exploitation

Cependant, dès que la table-Q commence à donner de bons résultats, la machine peut tomber dans le piège de suivre une politique d'action limitée, ignorant au passage les autres possibilités encore inexplorées.

De la même manière qu'après avoir trouvé un bon restaurant dans votre ville, vous choisissez systématiquement d'aller dîner dans ce restaurant car vous pensez qu'il n'y en a pas de meilleurs.

Pour éviter ce phénomène, on définit dans l'algorithme du Q-learning une probabilité **epsilon** qui pousse la machine à continuer **d'explorer** l'environnement des possibles, plutôt que de simplement **exploiter** sa politique d'action déjà apprise.

C'est là le dernier élément qui compose l'algorithme du Q-Learning.

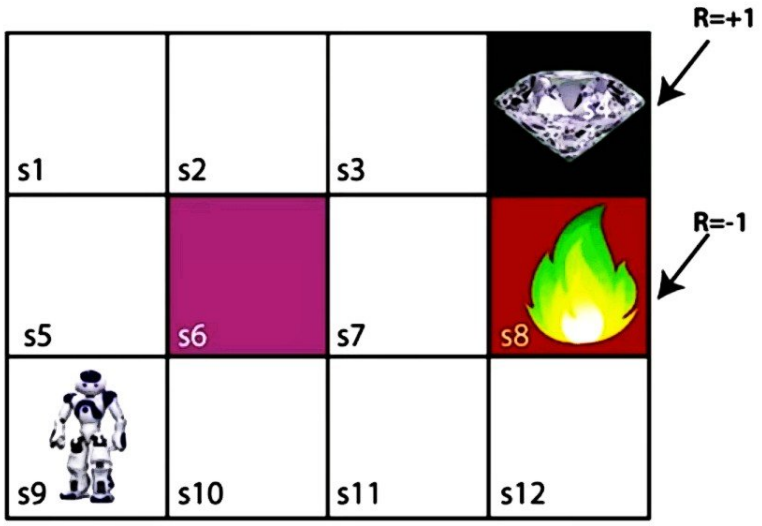
Pour finir ce chapitre, je vous propose donc de passer à l'action avec une implémentation des plus amusantes ☺

The Bellman Equation

- So now, using the Bellman equation, we will find value at each state of the given environment.

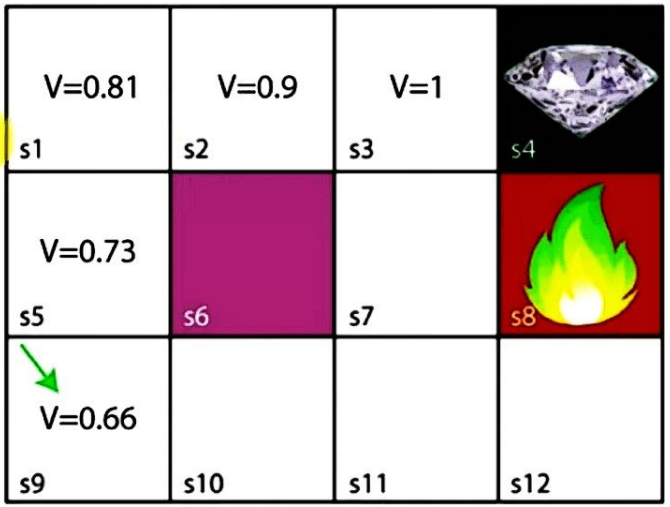
- We will start from the block, which is next to the target block.

- **For S3 block:**
- $V(s3) = \max[R(s,a) + \gamma V(s`)]$,



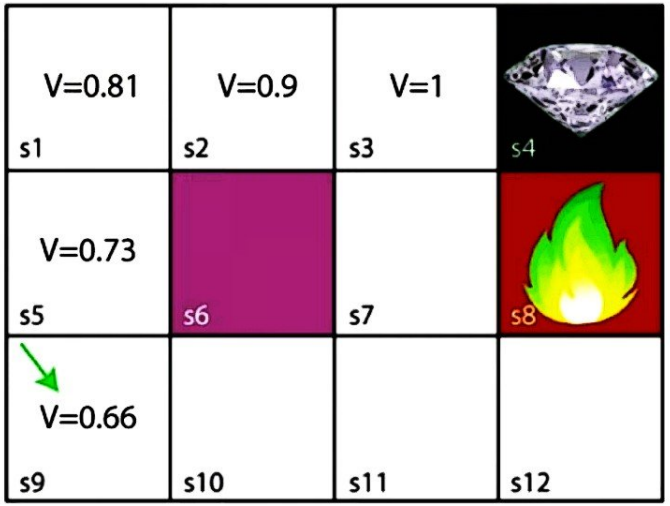
The Bellman Equation

- **For S2 block:**
- $V(s2) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 1, and\ R(s,a) = 0$, because there is no reward at this state.
- $V(s2) = \max[0.9(1)] => V(s2) = \max[0.9] => V(s2) = 0.9$



The Bellman Equation

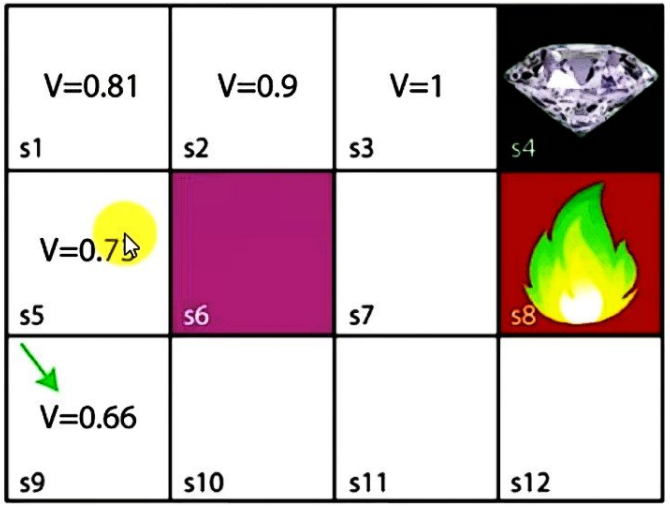
- **For S1 block:**
- $V(s1) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9(lets), V(s') = 0.9, and\ R(s,a) = 0$, because there is no reward at this state also.
- $V(s1) = \max[0.9(0.9)] => V(s1) = \max[0.81] => V(s1) = 0.81$



The Bellman Equation

- **For S1 block:**
- $V(s1) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9(lets), V(s') = 0.9, and\ R(s,a) = 0$, because there is no reward at this state also.
- $V(s1) = \max[0.9(0.9)] => V(s1) = \max[0.81] => V(s1) = 0.81$

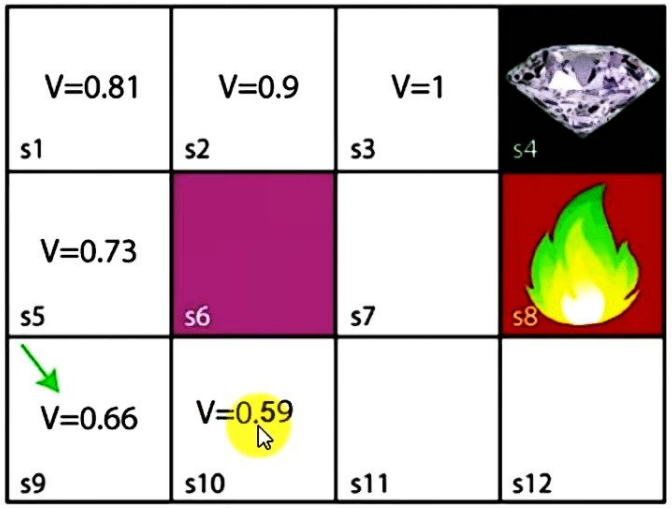
- **For S5 block:**
- $V(s5) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9(lets), V(s') = 0.81, and\ R(s,a) = 0$, because there is no reward at this state also.



The Bellman Equation

- **For S10 block:**

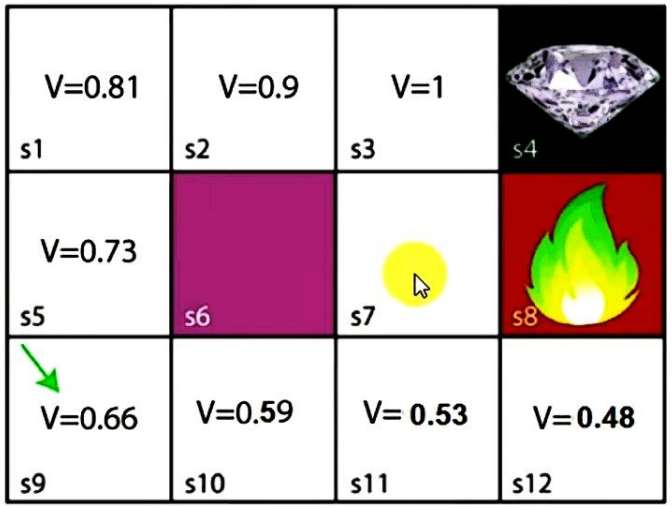
- $V(s10) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 0.66$ *and* $R(s,a) = 0$, because there is no reward at this state also.
- $V(s10) = \max[0.9(0.66)] => V(s10) = \max[0.59] => V(s10) = 0.59$



The Bellman Equation

- **For S12 block:**

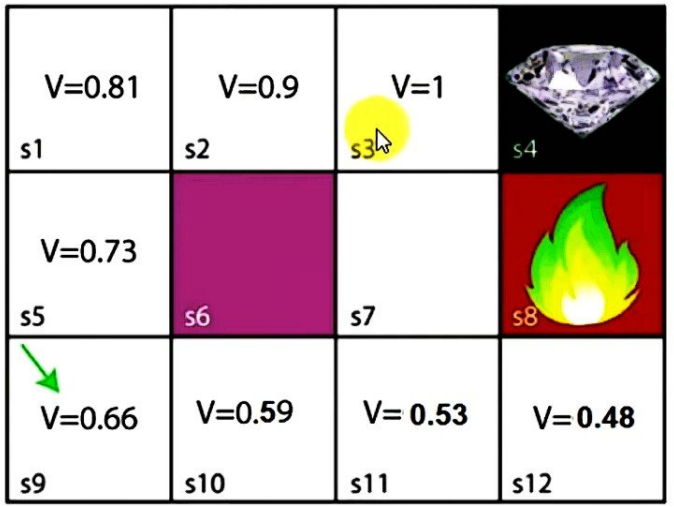
- $V(s12) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 0.59$ *and* $R(s,a) = 0$, because there is no reward at this state also.
- $V(s12) = \max[0.9(0.53)] => V(s12) = \max[0.48] => V(s12) = 0.48$



The Bellman Equation

- **For S7 block:**

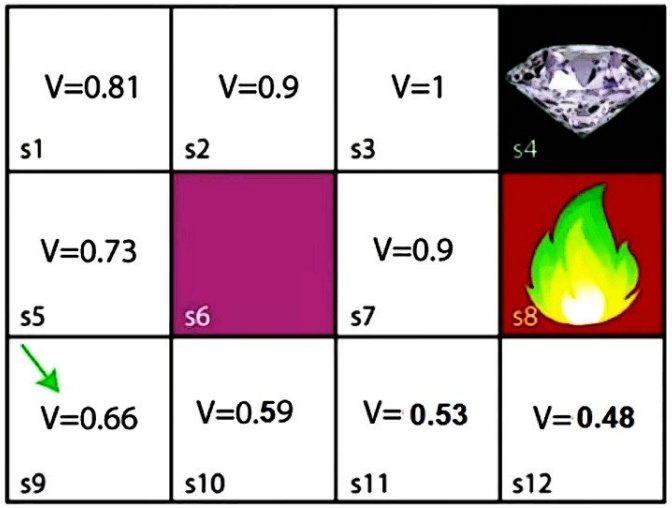
- $V(s7) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 1\ and\ R(s,a) = 0$, because there is no reward at this state also.
- $V(s7) = \max[0.9(1)] => V(s7) = \max[0.9] => V(s7) = 0.9$



The Bellman Equation

- **For S11 block:**

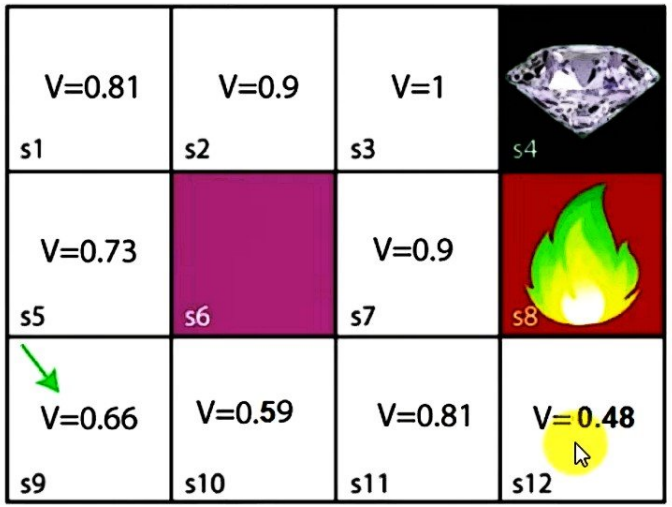
- $V(s11) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 0.9$ *and* $R(s,a) = 0$, because there is no reward at this state also.
- $V(s11) = \max[0.9(0.9)] => V(s11) = \max[0.81] = > V(s11) = 0.81$



The Bellman Equation

- **For S11 block:**

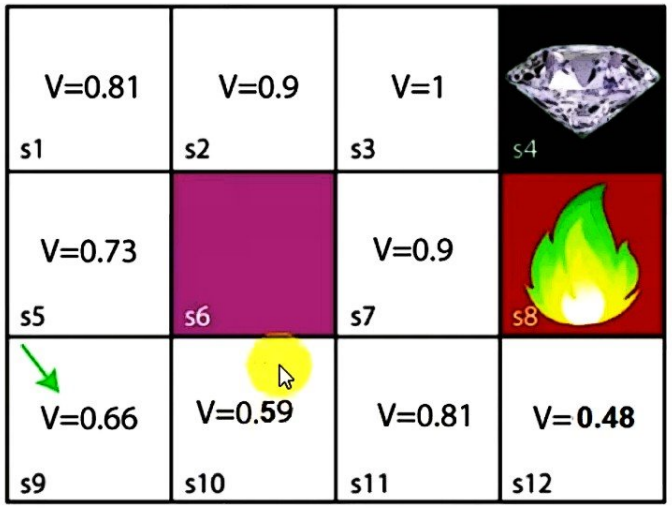
- $V(s11) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 0.9$ *and* $R(s,a) = 0$, because there is no reward at this state also.
- $V(s11) = \max[0.9(0.9)] => V(s11) = \max[0.81] = > V(s11) = 0.81$



The Bellman Equation

- **For S10 block:**

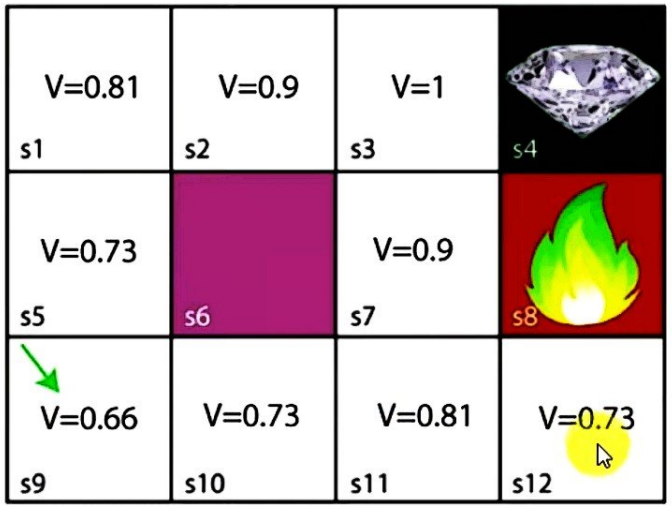
- $V(s10) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 0.81$ *and* $R(s,a) = 0$, because there is no reward at this state also.
- $V(s10) = \max[0.9(0.81)] => V(s10) = \max[0.73] => V(s10) = 0.73$



The Bellman Equation

- **For S12 block:**

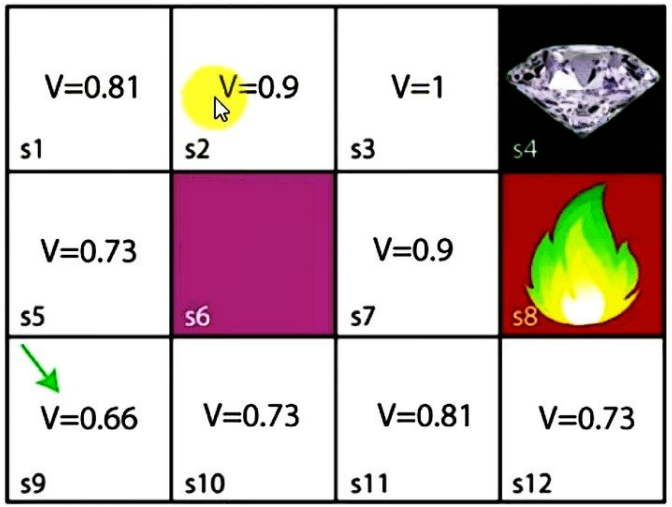
- $V(s12) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 0.81$ *and* $R(s,a) = 0$, because there is no reward at this state also.
- $V(s12) = \max[0.9(0.81)] => V(s12) = \max[0.73] => V(s12) = 0.73$



The Bellman Equation

- **For S12 block:**

- $V(s12) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 0.81$ *and* $R(s,a) = 0$, because there is no reward at this state also.
- $V(s12) = \max[0.9(0.81)] => V(s12) = \max[0.73] => V(s12) = 0.73$



The Bellman Equation

- The Bellman equation can be written as:

$$V(s) = \max[R(s,a) + \gamma V(s')]$$

- Where,
- $V(s) = value\ calculated\ at\ a\ particular\ point.$
- $R(s,a) = Reward\ at\ a\ particular\ state\ s\ by\ performing\ an\ action\ a.$
- $\gamma = Discount\ factor$
- $V(s') = The\ value\ at\ the\ previous\ state.$
- In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

The Bellman Equation

- The key-elements used in Bellman equations are:
  - Action performed by the agent is referred to as "a"
  - State occurred by performing the action is "s."
  - The reward/feedback obtained for each good and bad action is "R."
  - A discount factor is Gamma "γ."

Markov Decision Process (MDP)