CYCLE 5

RESOLUTION DE PROBLEMES PAR UTILISATION DE L'INGENIERIE NUMERIQUE OU L'APPRENTISSAGE AUTOMATISE

TP1 - PSI

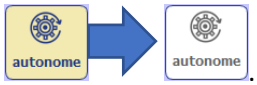
TP 1.2

ROBOT ALPHAI - RENFORCEMENT

L'objectif de ce TP est d'approfondir l'IA à travers l'algorithme des K plus proches voisins et de la méthode de renforcement.

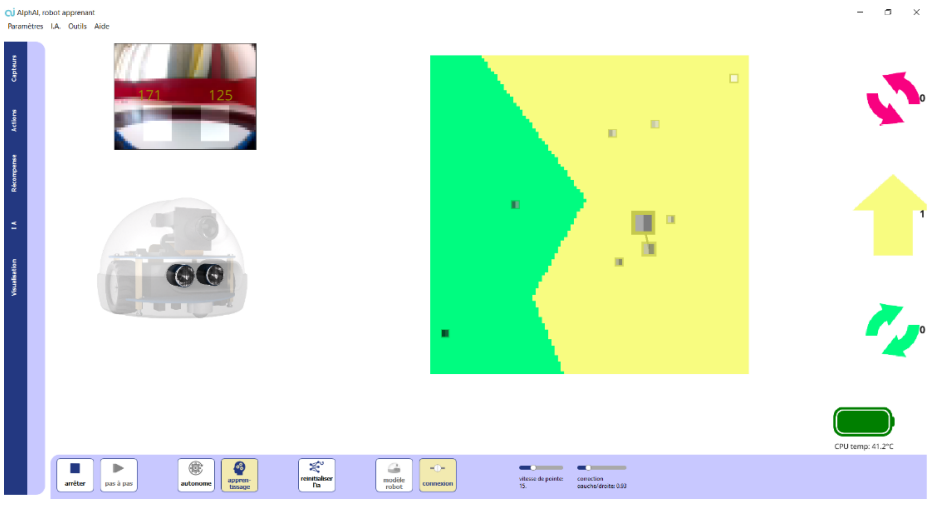
1 KNN

1.1 Découvrir l'algorithme des K plus proches voisins en exécutant la version incluse dans le logiciel.

- Dans le menu Paramètres / Charger les paramètres d'exemple, chargez la configuration « Apprentissage supervisé – KNN caméra ». Désactivez le mode autonome du robot en cliquant sur l'icône 

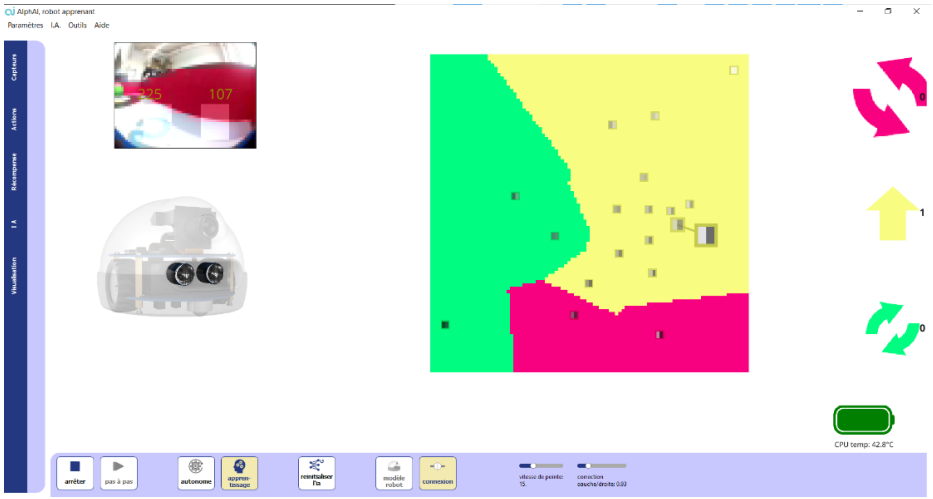
.

- Dirigez le robot en cliquant sur les flèches à droite de l'écran, où grâce aux touches « flèches » du clavier. Tournez quand le robot est proche des murs et allez tout droit sinon. *Vous pouvez aussi déplacer le robot à la main pour le mettre dans de nouvelles situations.*



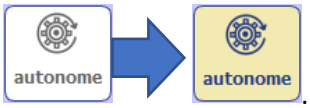
Après chaque mouvement un nouveau point apparaît. La position de chaque point est déterminée par ses coordonnées (x ; y), l'abscisse x représentant la luminosité dans la zone de gauche et l'ordonnée y la luminosité dans la zone de droite. La couleur autour de chaque point est la couleur associée à l'action choisie (vert pour pivoter à droite, jaune pour en avant et rouge pour pivoter à gauche). Ces points sont les données d'entraînement. Cette phase est primordiale pour que l'IA fonctionne correctement. Si les données d'entraînement contiennent trop d'erreurs ou approximations, l'IA ne permettra pas d'avoir un résultat satisfaisant.

Graphe obtenu après un bon entrainement du robot :



La luminosité diminue lorsque le robot s'approche des murs. Normalement, si la zone de gauche est la plus sombre, il faut tourner à droite, et inversement.

Lorsque vous aurez trois régions bien claires sur le graphique, arrêtez la phase d'entraînement et réactivez l'autonomie pour passer à la phase de test.

- Après un court apprentissage, réactivez l'autonomie 

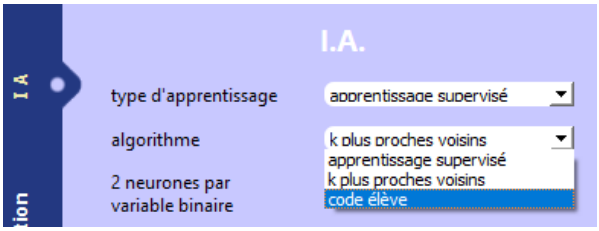
.

Si le robot est bien entraîné, il saura éviter les bords par lui-même. Si au contraire il est mal entraîné ou la luminosité dans l'arène n'est pas homogène, les zones de couleur sur le graphique auront des contours plus irréguliers et le robot fera plus d'actions inappropriées.

1.2 Programmer soi-même l'algorithme

→ Désactivez le bouton « Autonome ».

→ Dans l'onglet IA, sélectionnez Algorithme : code élève, cela ouvrira une nouvelle fenêtre vous demandant de nommer le fichier.



→ Après lui avoir donné un nom, le fichier apparaît dans l'explorateur Windows : ouvrez-le dans votre éditeur de code préféré.

Ce code présente 3 fonctions déjà existantes : *init*, *learn* et *take_decision*. Ces fonctions seront appelées par le logiciel principal. Seule la fonction *take_decision* est importante pour ce TP.

→ Remarquez que quand vous vous déconnectez, un mini-robot simulé apparaît en bas à droite. Si vous le voulez, vous pouvez faire les questions ci-dessous avec ce mini-robot et revenir au vrai robot à la

fin du TP ; mais vous pouvez aussi faire tout le TP en restant connecté au vrai robot : comme vous préférez !

PROGRAMMER LE ROBOT DIRECTEMENT

Avant de programmer une intelligence artificielle, commençons par comprendre le principe de la fonction *take_decision*. Vous allez pouvoir modifier cette fonction pour changer le comportement du robot : après chaque modification, sauvegardez votre code et cliquez sur le bouton « Réinitialiser l'IA ». Cette étape est à répéter pour chaque modification du code : **sauver puis recharger le code**.

La fonction *take_decision* accepte un paramètre *sensors* de type *list[float]* et sa valeur de retour doit être un nombre entier (*int*).

La liste *sensors* contient **deux** nombres décimaux qui représentent la vision du robot. Le premier nombre représente l'**intensité lumineuse** dans la partie gauche de l'image et le deuxième nombre représente l'intensité lumineuse dans la partie droite. La valeur de retour de la fonction doit être un nombre entier correspondant à une action possible pour le robot (tourner à gauche, tourner à droite, ou aller tout droit).

Q1. Ajoutez l'instruction print(sensors) à l'intérieur de la fonction. Sauvegardez, cliquez « Réinitialiser l'IA » dans le logiciel, et démarrez le robot. La valeur de sensors s'affiche maintenant dans la console (cliquer dans la barre des tâches sur l'icône pour afficher la console). Quelle est à peu près la valeur de sensors quand le robot est face à un mur ? quand il n'est pas face à un mur ? quand il y a un mur à sa gauche ou à sa droite ? Dans quel intervalle se trouvent donc les valeurs de sensors ?

Q2. Pour l'instant, la fonction take_decision renvoie la valeur 0. Quelle est l'action correspondant à ce nombre ? En modifiant cette valeur, trouver les nombres correspondants aux actions aller tout droit et tourner à droite.

Q3. Utilisez le paramètre sensors pour programmer un comportement cohérent du robot : « s'il n'y a pas de mur, je vais tout droit ; s'il y a un mur, je me tourne du côté le plus lumineux. »

PROGRAMMATION DE L'ALGORITHME

1.4.1 Calcul de distance

Programmez une nouvelle fonction *distance(a,b)*. Ses paramètres a et b sont deux listes de deux nombres décimaux (*float*), représentant les coordonnées (x ; y) d'un point du graphique. Sa valeur de retour doit être la **distance euclidienne entre ces deux points** (de type *float*).

Pour rappel, si $a = [a_1, a_2]$ *et* $b = [b_1, b_2]$

Alors la distance euclidienne entre ces deux points est : $\boxed{d = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}}$.

La racine carrée peut être utilisée en Python en important la bibliothèque math (ou numpy) en haut du fichier :

```
import math
```

et s'écrit ainsi :

```
y = math.sqrt(x)
```

L'opérateur carré peut être fait de la façon suivante :

```
y = x**2
```

Q4. Programmez la nouvelle fonction distance. Puis pour la tester, écrivez en-dessous :

```
# compute distance
a = [0, 0]
b = [1, 2]
print("distance", distance(a, b))
```

Sauvez, chargez le code avec « Réinitialiser l'IA » et notez le résultat qui s'affiche dans la console.

1.4.2 Calcul de toutes les distances

Maintenant que vous disposez d'une fonction pour calculer une distance, écrivez une nouvelle fonction *all_distances* qui prend en paramètres un point *a* et une liste de points *train_sensors*, qui représente les données d'entrainement du robot. Cette fonction *all_distances* doit renvoyer la **liste des distances** entre le point *a* et chacun des points contenus dans la liste *train_sensors*. Pensez à utiliser la fonction *distance* pour calculer une distance entre deux points.

Si la liste *train_sensors* est de longueur *n = len(train_sensors)*, alors la valeur de retour doit aussi être une liste de longueur n.

Q5. Programmez la fonction all_distances, puis testez-la en ajoutant le code suivant en bas du fichier :

```
# compute all distances
a = [0.4, 0.6]
train_sensors = [[0, 0], [0, 1], [1, 0]]
distance_list = all_distances(a, train_sensors)
print('distances to data', distance_list)
```

Rechargez le code et recopiez votre résultat.

1.4.3 Trouver le plus petit element d'un tableau

Créez une fonction s'appelant *find_minimum* prenant comme unique paramètre une liste de nombres *dlist* (de type *list[float]*) et renvoyant **l'indice du premier plus petit élément** (de type *int*).

Q6. Programmez et testez find_minimum avec le code suivant :

```
# minimum in a list
idx_min = find_minimum(distance_list)
print('index of minimum', idx_min)
```

Notez le résultat.

1.4.4 Le plus proche voisin

Maintenant que nous avons toutes les fonctions nécessaires, nous allons pouvoir créer la fonction *nearest_neighbor_decision* qui accepte 3 paramètres : *train_sensors*, une liste de points représentant les données d'entraînement du robot ; *train_decisions*, une liste d'entiers représentant les actions associées à chacun des points de *train_sensors* ; et un point *a*, représentant les valeurs de luminosité fournies par la caméra. On rappelle qu'un point est représenté par une liste de deux nombres décimaux qui sont ses coordonnées.

La fonction *nearest_neighbor_decision* doit (en 3 lignes) :

1) Calculer les distances entre le point *a* et chacun des points de la liste *train_sensors*.
2) Trouver l'indice de la plus petite de ces distances.
3) Renvoyer l'action correspondant au point d'entrainement le plus proche du point a. Sa valeur de retour est donc un entier correspondant à un code d'action (tout droit, à droite ou à gauche).

Q7. Programmez et testez nearest_neighbor_decision avec le code suivant :

```
# KNN
a = [0.4, 0.6]
```

```
train_sensors = [[0, 0], [0, 1], [1, 0]]
train_decisions = [1, 2, 0]
decision = nearest_neighbor_decision(train_sensors, train_decisions, a)
print('KNN', decision)
```

Recopiez le résultat.

1.4.5 UTILISATION DE VOTRE ALGORITHME AVEC LE ROBOT

Félicitations, vous avez programmé vous-mêmes l'algorithme des K plus proches voisins dans le cas K=1. Il ne vous reste plus qu'à l'utiliser dans le programme pour entraîner le robot.

Pour cela, recopiez les lignes ci-dessous pour reprogrammer la fonction *take_decision* pour prendre les bonnes décisions, mais également *learn* pour se rappeler des données d'entraînement *train_sensors* et *train_decisions* qui seront créées au fur et à mesure dans le programme principal :

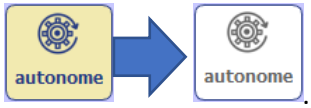
```
train_sensors = train_decisions = None

def learn(X_train, y_train):
    global train_sensors, train_decisions
    train_sensors, train_decisions = X_train, y_train
    loss = 0
    return loss

def take_decision(sensors):
    if train_sensors is None:
        return 0
    return nearest_neighbor_decision(train_sensors, train_decisions, sensors)
```

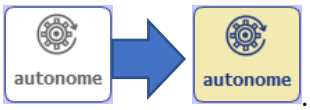
Et à présent, vous pouvez entraîner votre robot !

1. Rechargez votre code 

.
2. Désactivez l'autonomie du robot en cliquant sur l'icône 

.
3. Dirigez le robot en utilisant les flèches du clavier. Tournez quand le robot est proche des murs et allez tout droit sinon.

Les tableaux *train_sensors* et *train_decisions* sont remplis automatiquement par le logiciel à chaque fois que vous donnez un ordre au robot en appuyant sur une flèche.

4. Après un court apprentissage, réactivez l'autonomie 

.
5. Le robot navigue seul dans l'arène. Évite-t-il les murs par lui-même ?

Pour aller plus loin

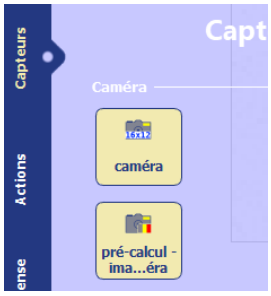
UTILISER UNE VRAIE IMAGE CAMERA

Pour pouvoir utiliser une image de meilleure résolution, il faut modifier notre fonction *distance*. En effet, celle-ci permet actuellement de calculer la distance entre deux points ayant chacun 2 coordonnées (puisque nous travaillions avec des images de 2 pixels). Il faut donc maintenant pouvoir calculer la distance entre deux

images ayant un nombre plus grand de pixels (ce nombre de pixels devant être le même pour les deux images). Pour cela, on peut par exemple utiliser la fonction suivante :

```
def distance(image1, image2):
    nombre_pixels = len(image1)
    somme_carres = 0
    for i in range(nombre_pixels):
        somme_carres = somme_carres + (image1[i] - image2[i])**2
    return math.sqrt(somme_carres)
```

Prendre le temps de bien lire et comprendre cette fonction. Quel est son prototype ?

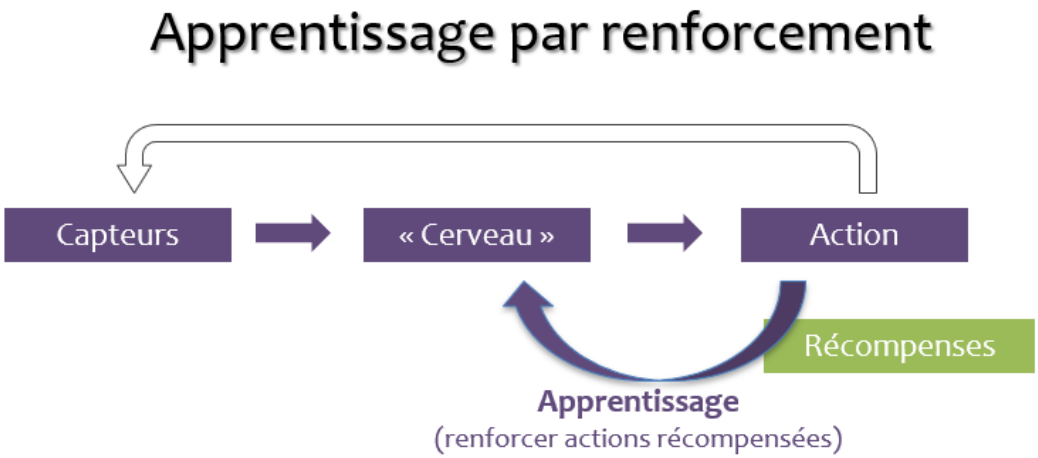


Grâce à cette fonction, vous pouvez maintenant organiser une course de robots en utilisant leur caméras ! (Choisissez par exemple dans l'onglet Capteurs la résolution de caméra 16x12 et le précalcul « Luminance – Jaune/Bleu – Vert/Rouge »).

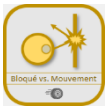
L'algorithme que vous avez écrit utilise le plus proche voisin pour prendre sa décision. Modifiez votre code pour qu'il prenne en compte la majorité des décisions prises entre un nombre $k > 1, k \in \mathbb{N}$. La difficulté réside dans la modification de la fonction trouvant le minimum. Il faut maintenant qu'elle en trouve *k* et non juste 1.

2 RENFORCEMENT

Nous allons découvrir en particulier l'**apprentissage par renforcement** et les **réseaux de neurones artificiels**. L'apprentissage par renforcement consiste à laisser le robot apprendre de ses expériences grâce à un système de récompenses et de pénalités.



2.1 Démarrage

1. Dans le menu **paramètres** (en haut à gauche de l'écran), cliquez sur **charger des paramètres d'exemple**, puis sélectionnez **Edition manuelle – Bloqué vs. Mouvement**. 

2. Maintenant commençons ! C'est vous qui allez créer les connexions dans le réseau de neurones du robot. Désactivez **apprentissage** 

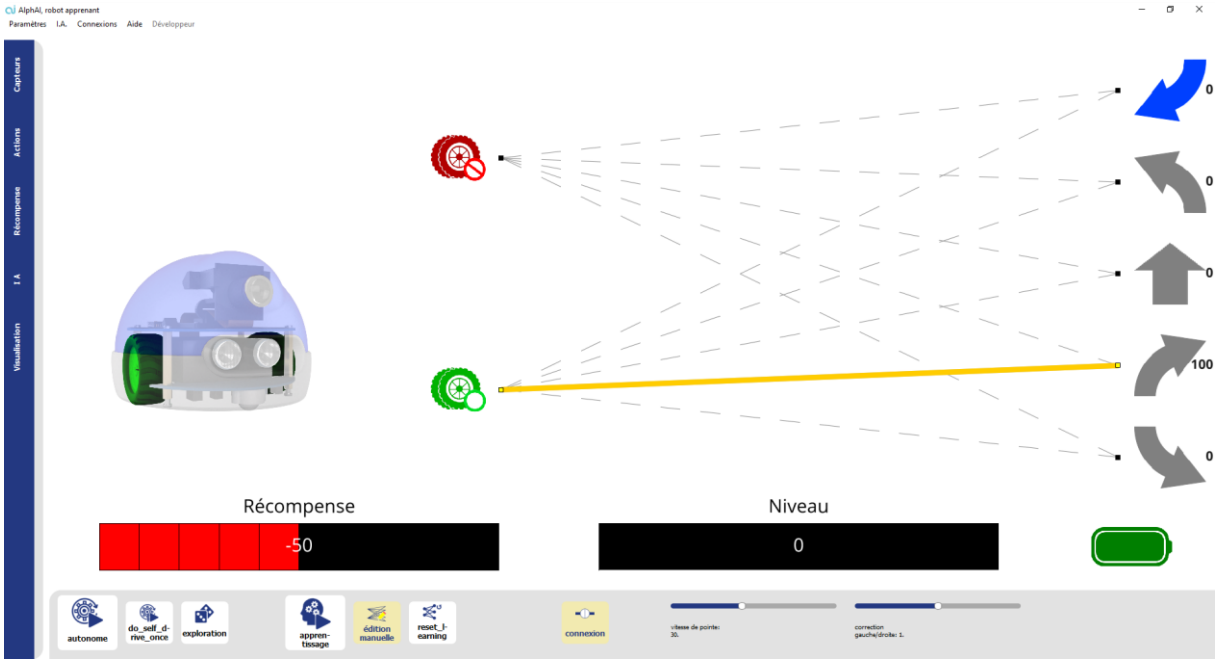
et **exploration** 

, activez **édition manuelle** 

et cliquez sur **réinitialiser l'IA** 

.

3. Vous devriez avoir à l'écran une représentation des connexions dans le réseau de neurones artificiels. Ce réseau a 7 neurones : 2 neurones d'entrée (capteurs) et 5 neurones de sorties (actions).



4. Si vous survolez avec la souris les connexions en pointillés, vous pourrez alors cliquer afin de **créer une nouvelle connexion**. Ces connexions vont transmettre l'activité des neurones d'entrée vers les neurones de sortie ; elles correspondent donc à la sélection d'une action (sur la droite) lorsque le robot se trouve dans un état en particulier (sur la gauche). Dans l'exemple ci-dessus, la connexion créée, va permettre au robot de choisir l'action **avancer vers la droite** lorsqu'il est dans l'état **non bloqué**.

5. Activez le mode autonome 

: le neurone vert « je ne suis pas bloqué » s'allume, et allumera une action dès lors que vous aurez créé une connexion : A vous de jouer donc, créez les bonnes connexions pour permettre au robot **d'obtenir le maximum de récompenses**. Un indice : les récompenses sont positives lorsque le robot avance, mais négatives s'il est bloqué ou recule.

6. Observez la valeur du niveau : il indique la moyenne des récompenses reçues au cours de la dernière minute. Notez la valeur qu'il atteint : nous allons comparer avec le niveau que le robot peut atteindre lorsqu'il apprend par lui-même !

2.2 APPRENTISSAGE AUTOMATIQUE : BLOQUÉ / MOUVEMENT

« Robot piloté »

1. Désactivez **édition manuelle** 

et à la place activez **apprentissage** 

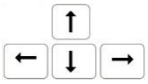
, puis cliquez sur **réinitialiser l'IA** 

: cela initialise des connexions au hasard.

2. Dans l'onglet **Visualisation**, activez l'affichage **valeurs des connexions** 

, de petits numéros s'affichent au-dessus des connexions. On voit que l'épaisseur des traits symbolise la « force » des connexions. Expliquez pourquoi les connexions affichées ne sont pas « bonnes » et devront être « améliorées » durant l'apprentissage ?

3. Pour commencer c'est *vous* qui allez piloter le robot, ce n'est pas lui qui va choisir ses actions, il va seulement *apprendre*. Pour cela désactivez **autonome** 

.
4. Le robot attend que vous le pilotiez ! Vous pouvez le faire en cliquant sur les flèches en sortie du réseau sur l'interface ou en utilisant les flèches du clavier 
.

5. Observez attentivement l'épaisseur des connexions et les chiffres qui se situent à côté des actions possibles ; à votre avis, à quoi sert ce chiffre ? que signifie-t-il ? pourquoi évolue-t-il ? comment évolue-t-il ? Comment est-il calculé ?

6. Lorsque vous pensez que l'apprentissage est terminé, désactivez **apprentissage** pour l'arrêter et réactivez **autonome** pour que le robot se conduise tout seul : se comporte-t-il comme attendu ?
Si ce n'est pas le cas, réactivez l'apprentissage pour réentraîner le robot.

Robot autonome

7. Recommencez l'apprentissage cette fois-ci avec le robot qui choisit ses actions par lui-même (appuyez sur **réinitialiser l'AI** et vérifiez que **autonome** est activé). Qu'observez-vous ? Le robot apprend-il correctement ? Que peut-il manquer ?

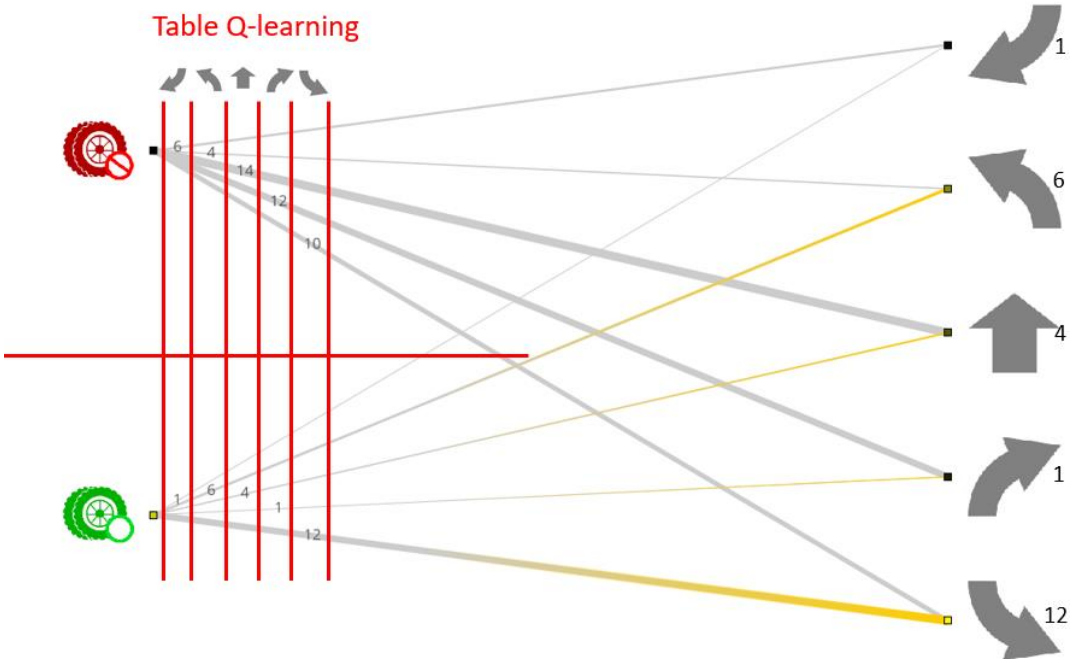
8. A présent activez également **exploration** 

dans la barre inférieure et observez comment se passe l'apprentissage. Que fait ce paramètre ? Appelez-nous pour nous faire part de vos réflexions.

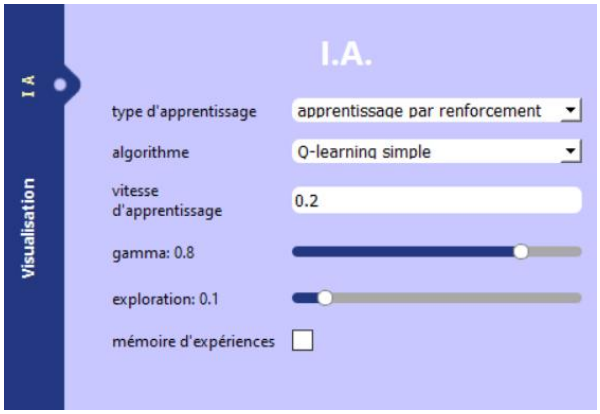
Compréhension des détails mathématiques

Vous avez observé que l'apprentissage modifie les connexions dans le réseau de neurone. Continuez les étapes suivantes pour en comprendre tous les détails et arriver à l'équation. Vous pouvez également aller directement à la partie III pour commencer les apprentissages avec caméra et revenir à cette étape à la fin s'il vous reste du temps.

L'algorithme utilisé ici est le Q-learning qui repose sur une table de valeurs représentée dans la figure ci-dessous. Cette table sert à déterminer à partir d'un calcul mathématique, quelle action sera utilisée à la prochaine étape. Les colonnes représentent les actions possibles et les lignes font référence aux états possibles. Quant aux valeurs, elles correspondent aux poids des connexions entre état et action.



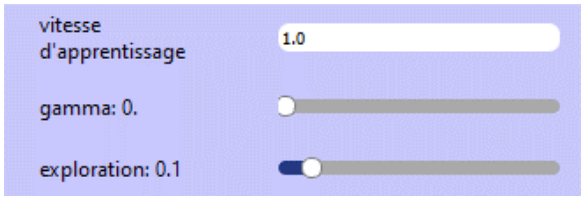
9. Affichez l'onglet « I.A. » : vous voyez apparaître 3 paramètres, *vitesse d'apprentissage*, *le facteur d'actualisation γ (gamma)* et l'*exploration*.



10. Le paramètre *exploration* représente la fréquence des explorations : variez-le entre 0 et 1 et observez l'effet sur le comportement du robot. Vous semble-t-il que la valeur initiale de 0.1 était correcte ?

On note a_t l'action effectuée à l'instant t, s_t l'état dans lequel on se trouve à ce même instant t et $Q(s_t, a_t)$ la valeur de la table de Q-learning pour ces 2 paramètres. Enfin, on note r_{t+1} la récompense immédiate reçue après cette action a_t dans l'état s_t.

11. A présent, réglez la vitesse d'apprentissage à 1 et *le facteur d'actualisation γ* à 0.



Dans ce cas, le calcul de la nouvelle valeur de la connexion revient à : $Q(s_t, a_t) = r_{t+1}$. En effet, nous ne tenons pas compte du phénomène d'actualisation.
Essayez un apprentissage avec ces valeurs des paramètres. Que se passe-t-il ?

12. Pour éviter des fluctuations trop fortes des connexions sans jamais converger vers une valeur stable, diminuez la vitesse d'apprentissage. Le calcul de la valeur de la table Q devient alors (α étant le paramètre "vitesse d'apprentissage") :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} - Q(s_t, a_t))$$

Ce qui donne après factorisation :

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha.r_{t+1}$$

Obtient-on de meilleurs apprentissages ? Voyez-vous quelque chose qui manque encore ?

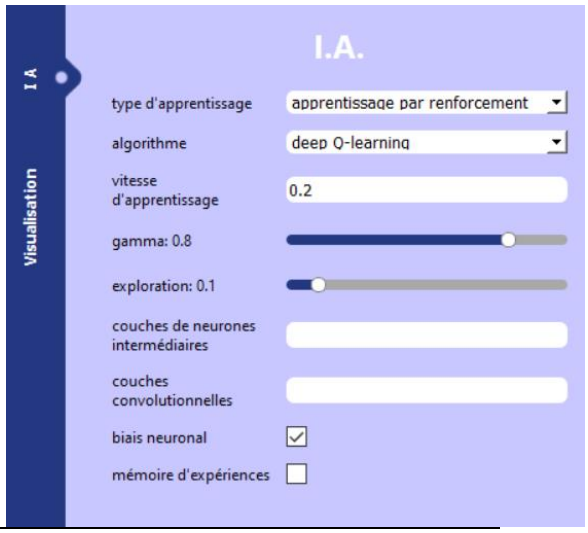
13. Pour permettre au réseau de neurones d'apprendre que quand le robot est bloqué l'action « se retourner » est meilleure que les actions « tout droit » ou « tourner », il faut intégrer dans la valeur des actions la qualité du nouvel état dans lequel on arrive (ici, bloqué ou non bloqué). C'est ce que permet de faire le paramètre d'actualisation gamma. La nouvelle valeur de la table devient :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left((1-\gamma)r_{t+1} + \gamma Q(s_{t+1}, a^*)\right) - Q(s_t, a_t))$$

Ce qui donne après factorisation :

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\left((1-\gamma)r_{t+1} + \gamma Q(s_{t+1}, a^*)\right)$$

Avec a* la meilleure action (celle rapportant la plus grande récompense) une fois arrivé dans le nouvel état s_{t+1}.



2.3 APPRENTISSAGE - ÉVITEMENT D'OBSTACLES AVEC CAMÉRA

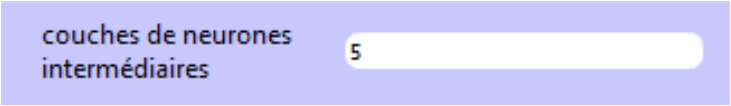
1. Dans l'onglet « I.A. » changez l'algorithme pour mettre « deep Q-learning », cela va permettre d'utiliser des réseaux de neurones multi-couches et d'activer de nouveaux paramètres.

2. Lancez un nouvel apprentissage : vous constaterez que les connexions et les valeurs des actions peuvent cette fois être positives *ou négatives*. Vérifiez qu'un premier apprentissage se déroule bien.

3. Activez l'option « mémoire d'expériences » et relancez l'apprentissage. Normalement il devrait être plus rapide ! En effet l'apprentissage est très accéléré car les connexions sont modifiées de nombreuses fois par seconde, non pas en fonction de la dernière action réalisée seulement, mais *de toutes les actions effectuées et récompenses reçues* jusqu'à présent. Il suffit donc de faire une exploration tout droit pour apprendre que l'action *d'aller tout droit* est meilleure que l'action de *tourner à gauche* ou *à droite*.

Neurones intermédiaires

4. Dans l'onglet IA, ajoutez une couche de neurones intermédiaires et regardez ces nouveaux neurones et de nouvelles connexions apparaître dans l'interface !

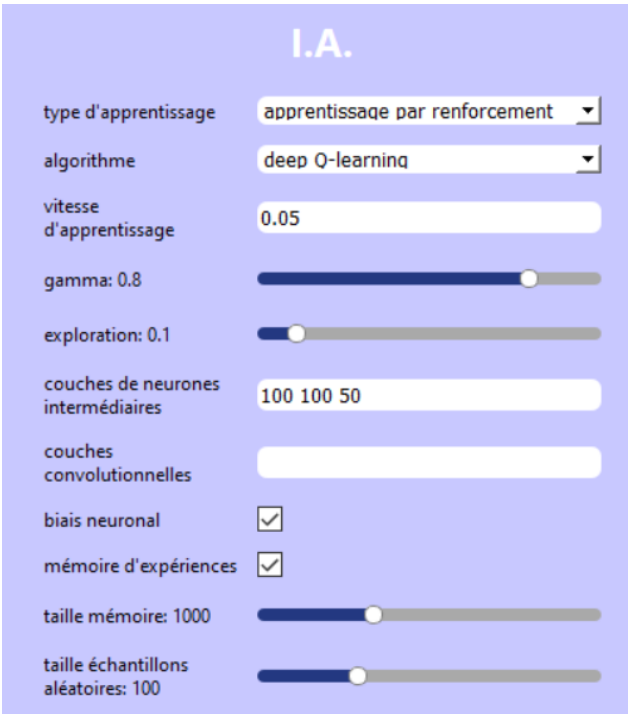


Relancez l'apprentissage. Comme précédemment l'algorithme d'apprentissage apprend les valeurs correctes des actions, mais le calcul de ces valeurs devient beaucoup plus complexe puisqu'il dépend d'un grand nombre de connexions. Appelez-nous pour en discuter avec nous.

Caméra

5. Enfin ! Vous pouvez activer la caméra en chargeant les paramètres d'exemple **Apprentissage par renforcement – Evitement d'obstacles**. 

Voici ci-dessous les réglages que nous vous conseillons pour l'onglet IA : vous pourrez aussi en essayer d'autres !



6. Démarrez le robot et observez comment le robot apprend. Normalement il apprend successivement à aller tout droit, à se retourner quand il se cogne à un obstacle, puis à reconnaître ces obstacles à l'avance et tourner avant de se cogner dedans ! Cet apprentissage prend à peu près 10 minutes. Jouez avec les options de visualisation pour voir séparément les connexions dans le réseau de neurone, ses activités, ses apprentissages… Notez vos observations.

2.4 APPRENTISSAGE - FOOTBALL

1. Pour réaliser cet apprentissage avec les paramètres recommandés, chargez les paramètres d'exemple **Apprentissage par renforcement – Suivi de ballon (vert)**. 

2. Vous pouvez parcourir les paramètres des onglets situés à gauche de l'écran, pour comparer les paramètres de ce scénario avec les précédents. Ce sont les récompenses qui sont maintenant déterminées différemment.

3. Vous êtes prêt ! Mettez un ballon vert dans l'arène et lancez l'apprentissage ! Cet apprentissage prend un peu plus de temps, le robot apprend d'abord à suivre le ballon lorsque celui-ci est proche, il met plus de temps à le reconnaître de loin.

4. Il peut arriver que les pixels verts dans l'image ne soient pas bien détectés. Dans ce cas
   a. Réglez les **actions possibles** dans les paramètres à **stop**.
   b. Positionnez le robot face au ballon vert.
   c. Appelez-nous pour vous aider à régler les paramètres pour la récompense de la couleur.

   Vous êtes arrivés jusqu'ici ? Vous pouvez relancer de nouveaux apprentissages en modifiant les paramètres que vous avez appris !

3 CONCLUSION

Notions apprises :

Ce qu'on appelle « Intelligence Artificielle » est en fait mal défini, plus qu'une *méthode*, c'est un *projet* : celui de reproduire l'intelligence biologique !

Les grands progrès récents concernent l'*Apprentissage Machine* (en anglais, *Machine Learning*) : nous avons découvert cela de près aujourd'hui !

La méthode souvent la plus efficace est l'utilisation de *réseaux de neurones artificiels* qui copient l'activité et les apprentissages des neurones dans notre propre cerveau !! Plus précisément il s'agit d'augmenter et diminuer les bonnes *connexions* entre les neurones artificiels.

Par conséquence certaines idées de ces algorithmes renvoient à nos propres apprentissages ! Essais et erreurs ; Curiosité ; Temps d'apprentissage, Répétition et Apprentissage pendant le repos ; etc.

Tout cela s'appuie sur des équations mathématiques rigoureuses. Les chercheurs en Intelligence Artificielle doivent être particulièrement qualifiés en *statistiques*, mais aussi en *programmation* et en ayant *un bon sens intuitif*.