Projet A4MA

Hugo GILBERT et Stephen LARROQUE

année 2013 2014

PROJET A4MA, PROBLÈME DU WUMPUS
PROFESSEUR : V.CORRUBLE

1 Introduction au problème

Le problème considéré est celui du wumpus (Russel & Norvig,1996, Gregory Yob, 1972) dans lequel un agent chasseur de trésor essaye de trouver un trésor dans un labyrinthe où se trouvent des puits sans fond dans lesquels il peut tomber et un terrible monstre (le wumpus) qui peut dévorer l'agent. Le but est de trouver et de ramener le trésor.

Les perceptions possibles de l'agent sont les suivantes :

– La brise à proximité d'un puit dans lequel peut tomber l'agent.
– L'odeur du monstre.
– La brillance du trésor.

L'agent peut :
– Se déplacer dans une case adjacente.
– Tirer son unique flèche pour tenter de tuer le wumpus.

2 Fonctions de récompenses et de pénalités

Nous avons proposé la fonction de récompense suivante :

Action	Récompense
Trouver le trésor	10
Se faire manger	-5
Tomber dans un puit	-2
Arriver dans une case avec un danger à proximité	0
Tuer le monstre	5
Visiter une même case plus de 2 fois utilisé seulement en Neural Q-Learning	-0.1
Arriver dans une case sans dangers à proximité	0
Arriver dans une case non visitée jusqu'à présent utilisé seulement en Neural Q-Learning	1
Pénalité pour chaque pas effectué utilisé seulement en Neural Q-Learning doit favoriser les chemins court	-0,1

On remarque que les récompenses données après avoir trouvé le trésor ou tué le Wumpus sont très élevés en valeur absolue par rapport aux autres récompenses ce qui encourage fortement ces actions. De même, s'approcher du danger n'est pas trop pénalisant par rapport au fait de tomber dans un trou ou de se faire manger. Cela est important pour ne pas dissuader toute exploration.

3 Notre approche de modélisation du problème Wumpus

Si les positions des différents agents ne changent pas, alors chaque case du labyrinthe peut alors représenter un état. Si n est la longueur du coté du labyrinthe, il y a donc n^2 états à considérer ce qui n'est pas trop élevé pour des petits labyrinthes mais cela peut devenir trop important pour de grands labyrinthes. Un algorithme de td-learning ou de Q-learning, avec la fonction de récompense précisée au dessus, réalise alors l'apprentissage. Si le monde change à chaque tentative du chasseur, alors la modélisation précédente ne tient plus. Un état ne sera plus une case mais un mélange entre la perception actuelle et les perceptions passées de l'agent. Le problème a été formalisé en Netlogo.

Nous commencerons par présenter l'interface graphique réalisée avec les différentes options qu'elle comporte. Ensuite, nous décrirons les différents algorithmes implantés pour réaliser l'apprentissage. Enfin nous discuterons des résultats obtenus.

3.1 Présentation de l'interface

L'interface graphique est composée d'un certains nombre de boutons et de sliders qui nous permettent d'effectuer des actions sur le jeu ou de régler des paramètres. Tous les paramètres sont modifiables dans l'interface : alpha, gamma, epsilon, l'algorithme d'apprentissage, le nombre de couches et de neurones cachés pour le réseau de neurones, etc.. Par ailleurs la plupart peuvent être modifiés pendant la simulation. Par exemple il est possible de changer l'algorithme d'apprentissage. Par exemple apprendre avec TD-Learning puis Q-Learning puis revenir à TD-Learning, les apprentissages ne sont pas réinitialisés. Cela permet de comparer

facilement les différents algorithmes. Par contre, il n'est pas possible de modifier les valeurs des récompenses et des pénalités à partir de l'interphase. En effet, ces paramètres sont extrêmement sensibles et peuvent facilement faire crasher la simulation. Ils restent néanmoins modifiables en tête de la procédure setup-globals.
L'interphase comporte un écran sur lequel est réalisé le jeu. Enfin quelques graphiques renseignent sur l'apprentissage effectué. Latéralement, nous pouvons découper l'interphase en 4 grandes parties :

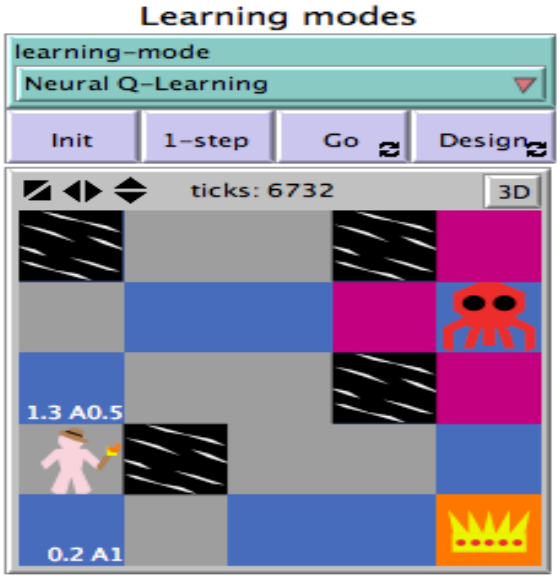
- Une partie centrale où nous pouvons visualiser le jeu et choisir le l'algorithme d'apprentissage.
- Une deuxième partie permet de régler les paramètres de l'algorithme de l'apprentissage sélectionné.
- Une troisième partie permet l'utilisation du Play Mode où l'on peut contrôler soit même l'agent explorateur.
- Une dernière partie est réservé aux paramètres de l'algorithme Neural Q-Learning.

Le jeu se déroule dans une grille 5×5. Avec notre modélisation, on peut noter que plus le monde est grand, plus la convergence car il y a plus d'états à explorer.

- L'explorateur est représenté par un petit bonhomme avec un flambeau et un chapeau d'aventurier.
- Les puits sont représentés par des blocks noire rayés de blanc.
- Le Wumpus est représenté par un horrible monstre violet.
- Le trésor est représenté par une couronne.

On peut visualiser sur les cases de nombreuses informations.

- Une case bleu ne comporte aucun danger à proximité.
- Une case grise est adjacente à un puit et représente le sens "vent".
- Une case rouge est adjacente au Wumpus et représente le sens "odeur".
- Une case magenta est adjacente et à un puit et au Wumpus et représente les sens "vent" et "odeur".
- Avec l'algorithme TD-Learning et le switch reinf-visu activé, chaque case est renseignée de la valeur de la case qu'elle représente. Un dégradé de bleu, du plus clair pour le meilleur état au plus foncé pour le moins bon, renseigne également sur la valeur des états.
- Avec les algorithmes Q-Learning et Sequential Q-Learning et le switch reinf-visu activé, chaque case est renseignée de l'action qui possède la meilleure qualité sur cette case ainsi que de la valeur de qualité correspondante.
- Avec le switch show-threats-scores activé, chaque case est renseignée des 2 scores de menaces respectivement dus au monstre et aux puits. En Neural Q-Learning, on voit également autour de l'agent les q-val prédites pour chaque prochaine action.



L'Utilisateur peut sélectionner ici plusieurs algorithmes d'apprentissage :

- TD-Learning
- Q-Learning
- Sequential Q-Learning
- Neural Q-Learning
- Chiken Heuristic
- Chiken Search

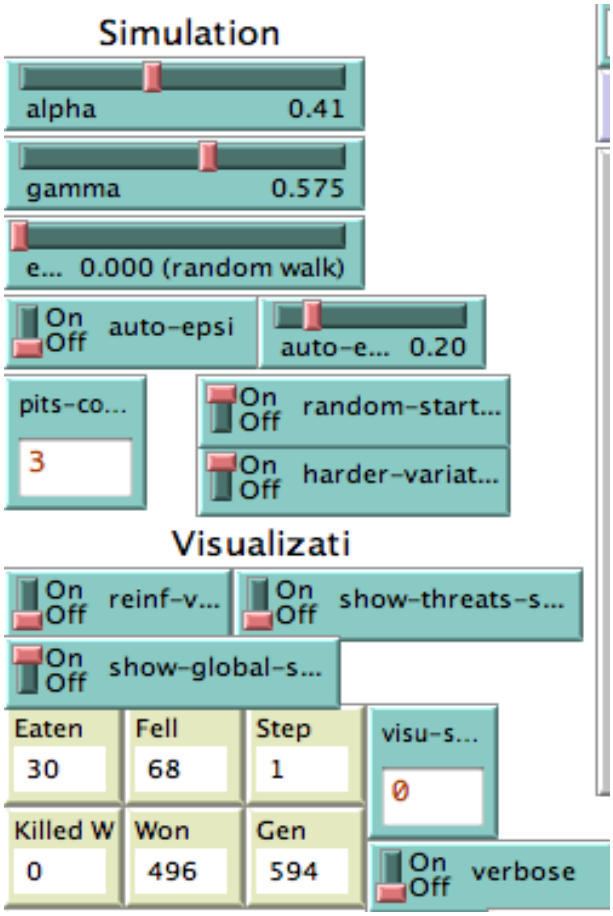
Un algorithme random est également fourni pour comparer.

Quatre boutons sont placés directement au dessus de l'écran du jeu :

- Init est le bouton d'initialisation du jeu.
- 1-step permet de réaliser une seule action.
- Go lance la simulation.
- Le bouton Design permet à l'utilisateur de placer les puits en clickant sur la carte.

Alpha, gamma et epsilon sont des paramètres de tous les algorithmes excepté les heuristiques Chiken.

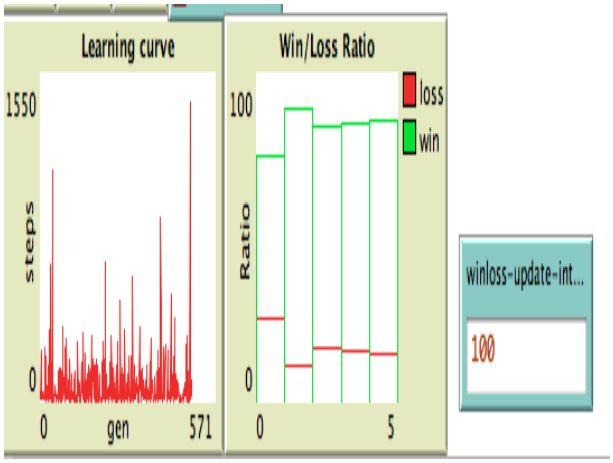
Paramètres de la simulation :



- Alpha est un taux d'apprentissage.
- Gamma représente la propension de l'agent à considérer les récompenses futures.
- Avec une probablité de (1-epsilon) l'agent réalise la meilleure action connue. Avec une probabilité de epsilon, l'agent réalise une action aléatoirement. (stratégie epsilonn-greedy)
- Si le switch auto-epsi est activé, alors epsilon commence à 0.9 et décroit exponentiellement en fonction de auto-epsi-param. auto-epsi aide énormément à converger de manière beaucoup plus rapide, mais seulement pour la version simple du problème.
- Le switch random-startpos fait commencer l'agent explorateur d'une position aléatoire à chaque début de partie. random-startpos permet de générer une politique globale au lieu d'une politique locale. cela permet aussi de converger beaucoup plus vite car cette option aide l'agent à explorer le monde.
- Le switch harder-variable permet de passer à une version plus difficile du problème où les positions des agents changent à chaque partie.

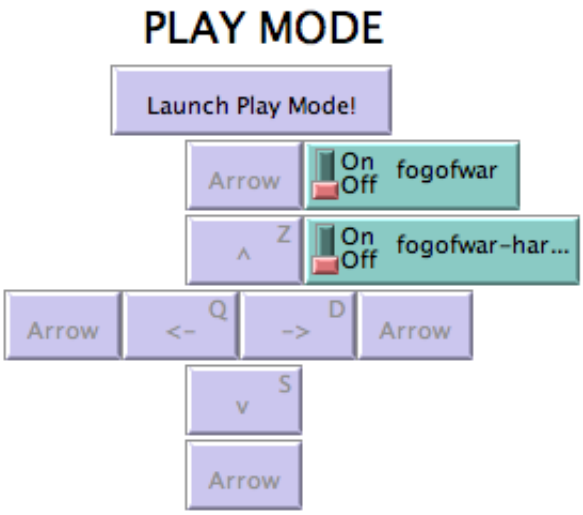
Visualisation

- reinf-visu permet de visualiser toutes les informations de la phase d'apprentissage. pits-count est le nombre de puits.
- Gen est le nombre de parties jouées.
- Step est le nombre d'étapes réalisées dans le jeu actuel.
- Eaten est le nombre de fois où l'agent a été mangé par le Wumpus.
- Fell est le nombre de fois où l'agent est tombé dans un puit.
- Killed W est le nombre de fois où l'agent a tué le monstre.
- Won est le nombre de parties où l'agent a trouvé le trésor.
- Si le switch show-threats-score est activé, il est affiché sur chaque case les scores de menace liés respectivement au monstre et aux puits.
- Si le switch verbose est activé, un certain nombre de messages liés au jeu peuvent s'activer à l'écran.



Deux graphes sont proposés pour visualiser l'apprentissage de l'agent.

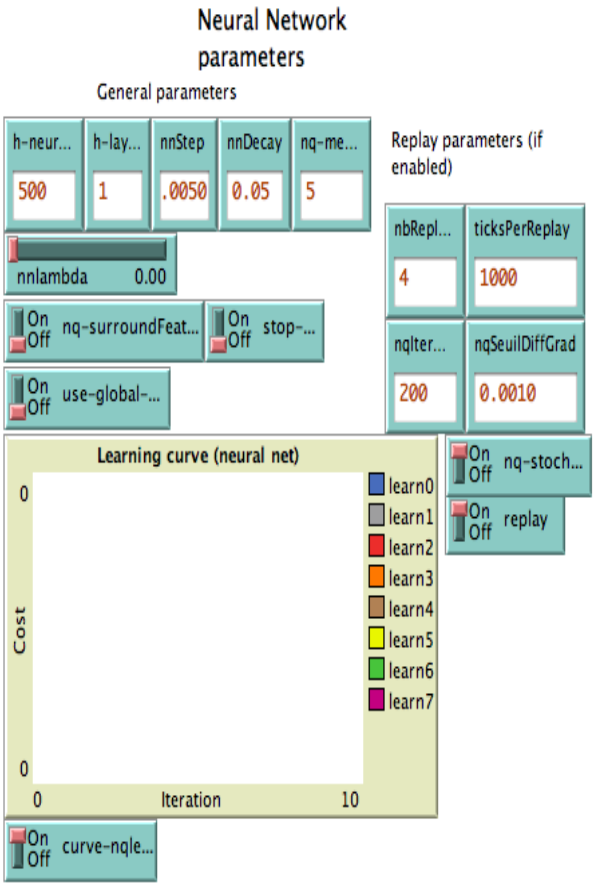
- La courbe learning curbe représente le nombre d'étapes par partie gagnée.
- Le graphique Win, Loss Ratio permet de voir le ratio de parties gagnées et le ratio de parties perdues.



En mode Play Mode, l'utilisateur peut déplacer l'agent à droite, à gauche, vers le haut, vers le bas, ou tirer une flèche dans ces 4 directions.
Avant de commencer si le switch fogofwar n'est pas actionné, le labyrinthe en entier est visible. Cette possibilité peut permettre au joueur de comprendre facilement le jeu. Si le switch fogofwar est actionné, le joueur se retrouve en situation réelle avec comme seul indice, la couleur de la case actuelle et des cases précédemment visitées. Le switch fogofwar-harder permet de passer à une version plus difficile du problème où l'on ne visualise que la case actuelle. Les cases précédemment visitées sont masquées, comme lorsque l'agent n'utilise pas la Mixture de Filtres Particulaires.

Cette dernière partie de l'interface permet de gérer le réseau de neurone qui est utilisé en mode Neural Q-Learning.

- h-neurons est le nombre de neurones par couche cachée
- h-layers est le nombre de couches cachées du réseau de neurone
- nnStep est le pas du gradient ou learning rate
- nnDecay, (learning rate decay) permet de diminuer le gradient à chaque itération pour converger plus rapidement
- nq-memo est le nombre d'expériences passées mémorisées (si 1, alors l'agent n'utilise que les perceptions de l'état en cours, si 2 , l'agent utilise les perceptions de l'état en cours et ceux de l'état précédent, etc.)
- nnlambda permet la régularisation pour le réseau de neurones (évite le sur-apprentissage).
- nq-surroundFeatures permet, quand il est activé, d'utiliser comme perception les informations de toutes les cases voisines.
- use-global-score permet de remplacer, quand il est activé, toutes les informations des voisins par un "global-score" agrégeant toutes les perceptions.
- stop-nqlearn est un bouton stop pour le réseau de neurones pendant la propagation arrière, s'il est trop long par exemple...



Paramètres de replay (si activé)

- replay : active le mode replay, les exemples ne seront pas appris tout de suite mais seront mémorisé pour n ticks et seront ensuite appris un certain nombre de fois en batch. Ce processus permet d'évoluer par pallier de manière cohérente (les valeurs nqval sauteront d'un coup mais de manière cohérente pour tous les exemples puisqu'ils auront tous été appris avec la même police). Sinon, si le mode replay est désactivé, l'agent apprendra tout de suite à chaque action (il mettra à jour par backpropagation le réseau de neurone, comme le Q-Learning le fait avec la fonction Q(lambda)).
- nq-stochastic : À utiliser en mode replay. L'apprentissage ne sera pas en batch mais stochastique, en partant du dernier exemple ajouté au premier (conseillé par [Lin, 92]).
- nbReplays est le nombre de phase d'apprentissage effectué en mode replay.
- ticksPerReplay est le nombre de ticks à attendre avant de lancer le prochain replay/apprentissage.
- nqIterMax : 1er critère d'arrêt de la propagation arrière. Sa valeur représente le nombre d'itération maximum pour la propagation arrière
- nqSeuilDiffGrad : 2eme critère d'arrêt. Sa valeur représente le seuil de différence entre deux gradients consécutif. Si la différence entre deux gradients consécutifs est en-dessous de ce seuil, l'apprentissage s'arrêtera là (mais pourra continuer lors du prochain replay).

Pour le graphique :

- curve-nqlearn : active ou désactive l'affichage du plot "Learning curve (neural net)" (En effet, cet affichage ralentit de manière significative l'apprentissage).
- les learnX : X est un numéro entre 0 et 7 et représente l'identifieur de l'action que le réseau est en train d'apprendre, dans l'ordre (suivant le fameux code SEGA) :
- 0 = haut
- 1 = bas
- 2 = gauche
- 3 = droite
- 4 = tir haut
- 5 = tir bas
- 6 = tir gauche
- 7 = tir droite

3.2 Les différents algorithmes d'apprentissage

Nous décrivons ici chaque algorithme :

TD-Learning et Q-Learning sont des algorithmes classiques d'apprentissage par renforcement pour trouver la politique optimale.

TD-Learning

Initialisation de $V(s)$ à 10 pour encourager l'exploration
Itérer
a<-action donner par la politique π pour l'état s
faire a ; observer la récompense obtenue r, et le nouvel état s'
$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
$s < -s'$
jusqu'à atteindre l'état optimal

Q-Learning

Initialisation de $Q(s, a)$ à 10 pour encourager l'exploration
Itérer
a<-action donner par la politique π pour l'état s
faire a ; observer la récompense obtenue r, le nouvel état s' et la future nouvelle action a'
$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
$s < -s'$
jusqu'à atteindre l'état optimal

Sequential Q-Learning

Sequential Q-Learning, est un Q-Learning un peu modifié que nous avons proposé pour permettre de tuer le Wumpus de manière plus fiable.
Cet algorithme permet d'avoir une séquence d'actions sur une seule case. Plus formellement on a :
$Q(s', a', 0) = Q(s, a, t) + \alpha[r + \gamma Q(s', a', 0) - Q(s, a, t)]$ si $s! = s'$
$Q(s', a', t + 1) = Q(s, a, t) + \alpha[r + \gamma Q(s', a', t + 1) - Q(s, a, t)]$ si $s = s'$

Le t est le nombre d'action effectué en une séquence sur la case s. Si l'on change de case alors t est réinitialisé à 0.
Pour chaque chaque case et action, nous n'avons pas une Q-val, mais un array de Q-val pour chaque tick où l'agent est resté sur cette case. Par exemple si l'agent tire, il va rester sur la même case, mais maintenant il n'est plus à t = 0 mais t = 1. Ainsi, plus tard, il saura qu'à t=0 il doit tirer pour tuer le wumpus, et seulement ensuite avancer quand t=1. Cette technique équivaut à définir une séquence temporelle pour chaque Q(s,a).

Neural Q-Learning

Neural Q-Learning apprend un réseau de neurones multicouches. Ce choix a été motivé par le fait que dans la version difficile du problème, on aimerait pouvoir inférer des comportement en fonction de perceptions passées proches de celles rencontrées.
Le Neural Q-Learning a été implanté selon les conseils de [Lin, 1992] et de http ://computing.dcu.ie/ humphrys/Notes/RL/q.neural.html, avec plusieurs approches :

- stochastique (replay off)
- avec replay et batch (replay on, nq-stochastic off)
- avec replay et stochastique arrière du dernier exemple au premier selon [Lin, 1992] (replay on, nq-stochastic on)
- pleins de features différentes et plusieurs configurations pour ces feautures. Des tests on été réalisé avec des features brutes ou avec des scores globaux d'aggrégation.(nq-surroundFeatures, nq-global-score). Nous avons également joué avec la fonction de récompense en mettant des pénalités pour le fait de revenir trop de fois sur une case précédente et des récompenses pour favoriser l'exploration, etc..
- Dans une des possibilité testée, le réseau de neurones utilise les n précédentes expériences comme features avec comme paramètre nq-memo pour savoir combien d'expériences doivent être mémorisées. (nq-memo)

Chicken Search Chicken Heuristic fait simplement aller l'agent à la case voisine avec le max global-score.

Chicken Heuristic Chicken Search fait une exploration en profondeur du max global-score. L'agent va toujours aller voir en premier la case avec le max global-score même si cette case est à l'autre bout du monde, il calculera le chemin.

Pour les algorithmes Chicken Search et Chicken Heuristic, il faut mettre epsilon à 0 pour avoir le maximum de performances avec ces deux heuristiques.

3.3 Discussion sur les résultats obtenus

Regardons en premier la complexité de la solution proposée. L'augmentation du nombre de cases ne fait qu'augmenter le temps (ticks) de convergence. En effet, il y a plus d'états à explorer. Par contre, la complexité de notre modèle n'est pas en fonction du nombre de case : il est quadratique en nombre de direction, mais pas de case. Le modèle fonctionne donc aussi bien et avec la même rapidité avec 10 cases que 90 cases voire 1000 cases. Cependant, le problème pour un aussi grand nombre de cases réside dans l'exploration de tous les états pour trouver l'optimum global.

Pour la première version du problème, les algorithmes TD-Learning et Q-Learning suffisent à régler le problème. L'agent apprend toujours à trouver le trésor. De plus si epsilon est à 0 ou converge vers 0, TD-Learning et Q-Learning convergent et trouve le chemin optimal pour trouver le trésor. On peut noter ic l'intérêt de auto-epsi qui fait effectivement converger epsilon vers 0. Sequential Q-Learning améliore le résultat en permettant de toujours tuer le Wumpus si ce dernier se trouve sur le chemin de l'agent (ce qui n'est pas le cas avec Q-Learning).

Pour le problème difficile, les algorithmes TD-Learning, Q-Learning, Sequential Q-Learning n'arrivent plus à apprendre de manière convenable. On remarque que les percepts de l'état en cours ne sont plus suffisants pour guider de manière pertinente l'agent. En effet, dans la version difficile du problème, on se retrouve non seulement face à un problème d'apprentissage, mais également à un problème de localisation : si l'agent ne sait pas où se trouvent les dangers, il ne saura pas où aller. Il est donc nécessaire de lui fournir une mémorisation des percepts afin de l'aider à localiser les dangers. Un moyen simple est de mémoriser les n précédentes actions et de les donner directement au réseau de neurone (ce que fait nq-memo). Malheureusement, ce processus limite la mémoire de l'agent.
Nous avons donc implanté une sorte de Mixture de Filtres Particulaires de Vermaak, mais en le modélisant avec des règles logiques au lieu de règles probabilistes : lorsqu'une case est visitée, un score de danger est attribué à ses voisins, voire un score de sûreté s'il n'y a aucun indice de danger. Au début, aucune case n'a de score de danger, mais au fur et à mesure de l'exploration de l'agent, les scores de chaque case est mis à jour et renforcé ou supprimé selon ce qui a été visité. Par exemple, si on visite deux cases voisines au Wumpus, on aura deux percepts d'odeur qui vont s'ajouter sur la case où il y a le Wumpus. En revanche, si on visite une case sans percepts, on sait alors que toutes les cases voisines sont sans danger, et donc on enlève tout score de danger sur ces cases.

Avec ces outils, nous avons testé un certain nombre de méthodes à base de réseau de neurones et utilisant des features et des modèles de mémoire variés et de nombreuses fonctions de récompenses. Nous avons essayé d'implanter une agrégation, un score global pour chaque état. Cependant, aucun de ces essais n'a apporté d'améliorations significatives.

Nous avons alors implanté les heuristiques Chiken pour vérifier que le global score était une bonne heuristique pour solutionner le problème. En effet, les deux dernières heuristiques proposées solutionnent la version difficile du wumpus. Chicken Heuristic gagne environ 75% du temps, et Chicken Search 85% du temps.

Pour solutionner le problème difficile, nous proposons dans de futures travaux d'utiliser le Hierarchical Q-Learning utilisant des fonctions de récompense locales ou W-Learning (Minimize Worst Unhappiness) ("Action Selection methods using Reinforcement Learning", Mark Humphrys, PhD Thesis, 1997.