Intelligence Artificielle

TP

TP1

2 — Environnement déterministe ou stochastique

Question 1

Exemple de modification :

```
GridworldMDP gmdp = GridworldMDP.getBridgeGrid();
```

Question 2

Dans le cas où le joueur décide de contrôler l'agent vers l'objectif de façon optimale, nous savons que lorsque le bruit est égal à 0, le comportement est déterministe car le chemin est optimal. Cependant, lorsque $x > 0$, les mouvements de l'agent deviennent aléatoires, et ne suivent pas forcément les actions du joueur, malgré sa façon de jouer. Ainsi, les chemins que l'agent doit prendre pour accéder à l'objectif ou le piège sont multiples, et chacun d'entre eux sont lié à une probabilité d'être exécuté. C'est donc un scénario stochastique.

3 — Agent aléatoire

Question 1

Le cœur du code de la classe **`AgentRandom`** réside dans la fonction **`AgentRandom.getAction(Etat)`**, car c'est cette fonction qui définit le comportement (ici aléatoire) de l'agent.

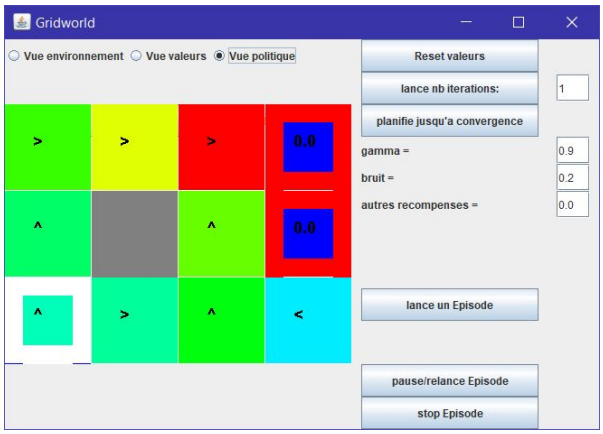
4 — Algorithme value iteration

Question 1

Voir code source dans **`ValueIterationAgent.java`**.

Note : Dans le fichier `ValueIterationAgent.java`, une partie du code dans `updateV()` et `getPolitique()` a été "*factorisé*" par la création de la fonction `double computeVActions(HashMap<Action, Double>, Etat)`, une fonction qui renvoie `V(e)` en calculant la somme pour toutes les actions possibles et pour tout état dans la `HashMap`.

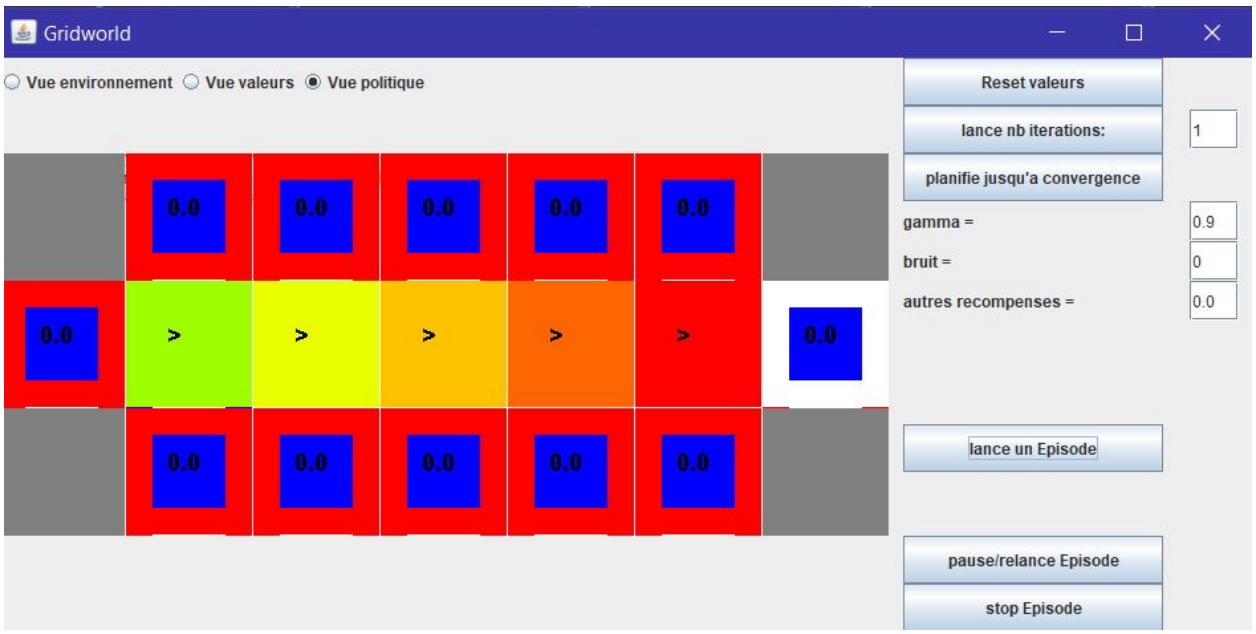
Question 2



5 — Influence des paramètres

5.1 — BridgeGrid

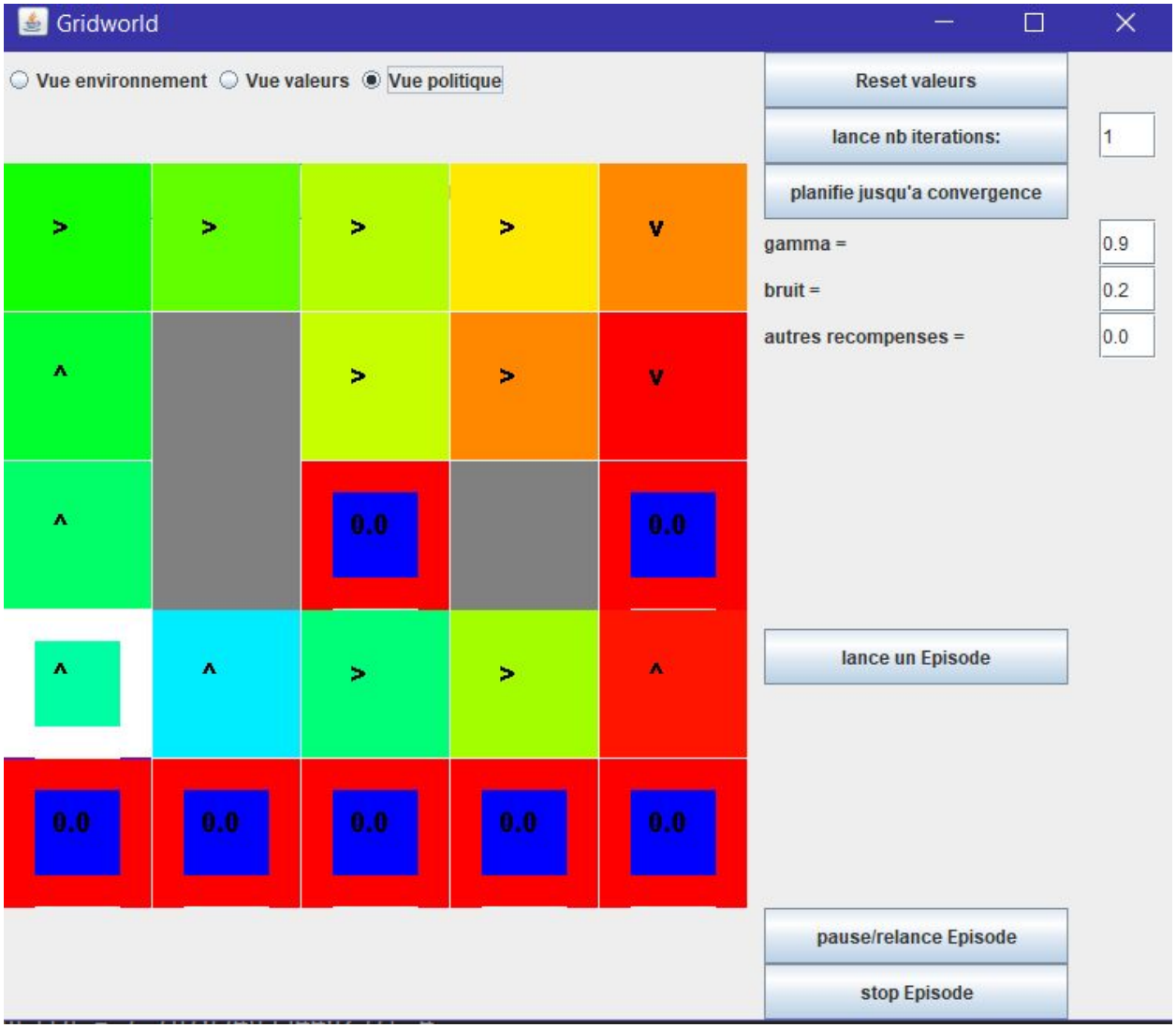
Question 1



Le paramètre à changer est le bruit : En le mettant à `0`, l'agent peut traverser le pont sans risque de "*tomber*" sur les côtés (les cases avec une valeur de -100). Si le bruit est supérieur à `0`, l'algorithme prend conscience du risque de tomber, et décide de ne pas faire traverser l'agent vers l'autre bout du pont.

5.2 — DiscountGrid

Question 2

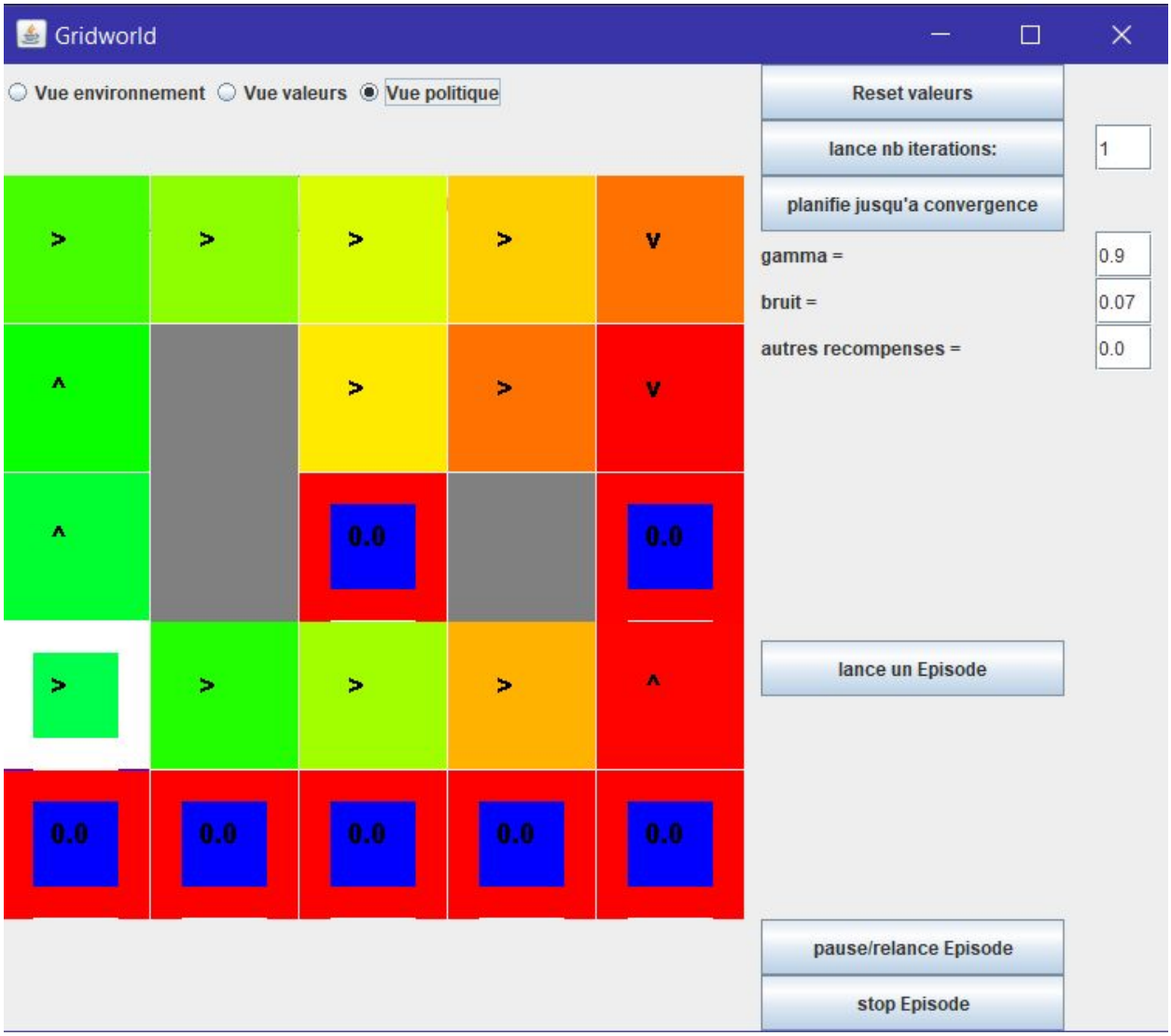


L'agent décide de prendre la récompense à +10 points plutôt que de la récompense à +1.

1. Pour que l'agent prenne un chemin risqué pour arriver à l'état absorbant "+1", il faut que $\gamma = 0.3$ et le bruit à `0`.

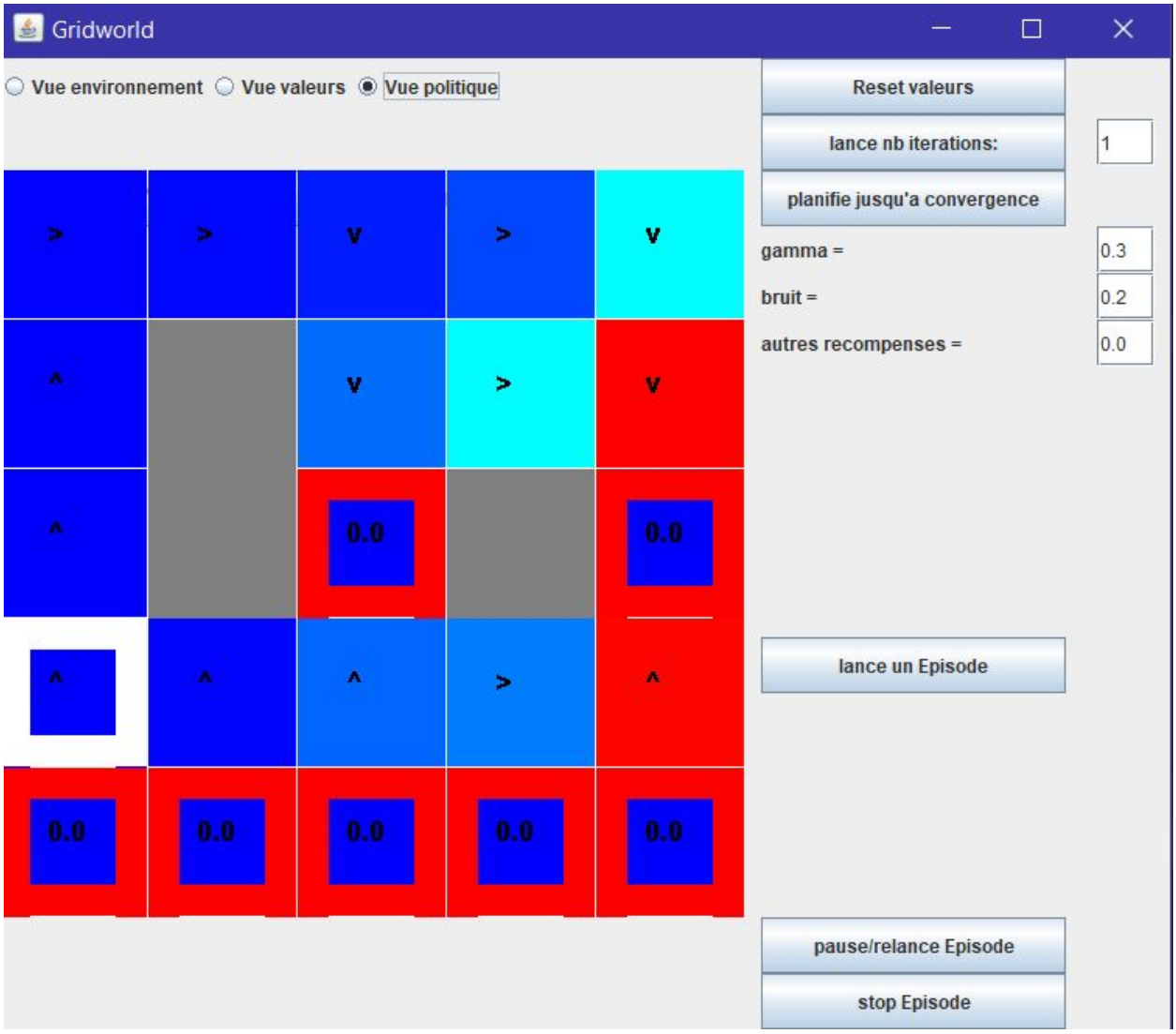
2. Pour que l'agent prenne un chemin risqué pour arriver à l'état absorbant "+10", il faut que le bruit soit inférieur ou égale à `0.07`. Ainsi, l'algorithme est apte à prendre des risque (car γ est grand) et le faible risque réduit fortement la probabilité de "tomber"

dans une case à valeur négative.



3. Pour que l'agent prenne un chemin sûr pour arriver à l'état absorbant "+1", il faut que $\gamma = 0.3$. Cela est dû au fait que plus γ est petit, moins l'algorithme va prendre des risques. C'est pour cela que l'agent prend le chemin le plus sûr vers la récompense la

plus proche (en l'occurence, +1).



4. Pour que l'agent évite les états absorbants, il faut que le bruit soit égale à 1. Ainsi, l'algorithme décide de ne pas faire bouger l'agent à cause du risque de "*tomber*" dans

une case à valeur négative.

TP 2

2.2 — QLearning Tabulaire

Question 1

Un état MDP est défini par deux variables :

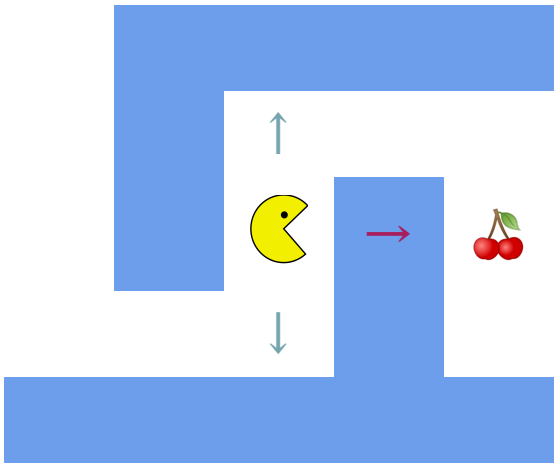
- Une liste de la position relative par rapport à pacman en x et en y de tous les fantômes dans un rayon de 3 cases, en utilisant Manhattan comme calcul de distance. Cette variable est de type "`ArrayList<Pair<Integer, Integer>>`", où

"`Pair<Integer, Integer>`" représente le couple (x, y). Le nom attribué à cette variable est "`distancePacmanGhosts`".

- La direction que le pacman doit prendre vers la nourriture la plus proche, notée "`directionToClosestFood`". Si la direction n'est pas une action valide (pacman se dirige vers un mur s'il prend cette direction), alors elle est remplacée par une des deux directions perpendiculaires aléatoirement.

  Exemple :

  > Pacman souhaite aller à l'est pour pouvoir manger le pacdot le plus proche. Cependant, un mur l'empêche d'y accéder. Le système va choisir aléatoirement soit la direction "*Nord*", soit la direction "*Sud*", de manière à faire bouger Pacman. Si l'une de ces deux directions n'est aussi pas disponible, l'autre est prise. Pour finir, si aucune direction n'est disponible, la valeur "*STOP*" est attribuée.



D'autre attributs sont présents dans la classe, mais ne sont utilisés que pour récupérer certaines informations pour les autres méthodes de la classe, et ne sont pas conséquent non présent dans le hashcode et la fonction `equals`.
Ajouter d'autre attributs comme la distance entre pacman et le pacdot le plus proche engendre un trop grand nombre d'état lors de l'exécution. De même, on ne liste pas la distance relative de tous les fantômes pour avoir un nombre d'état respectable.

2.3 — QLearning Généralisé

Les fonctions caractéristiques de la classe `FeatureFunctionPacman` sont :

1. Le biais (1),
2. Le nombre de fantômes étant à un pas ou moins de pacman à la prochaine itération,
3. La présence d'un pacdot à la prochaine itération (`1` si pacdot, `0` sinon).
4. La distance euclidienne entre pacman et le pacdot le plus proche divisé par la taille totale du labyrinthe.
5. La distance euclidienne entre pacman et le fantôme le plus proche
6. Le nombre de fantôme étant à 3 pas ou moins de pacman à la prochaine itération.

Comparaison des méthodes

On décide de comparer les 3 techniques en fonction de leur performance (pourcentage de réussite) et du temps d'apprentissage. Chaques algorithmes utilisent les valeurs suivantes :

- $\gamma = 0.8$
- $\alpha = 0.1$
- $\epsilon = 0.05$
- nbmean = 3
- nbepisodelearn = 500
- nbepisodegreedy = 300
- mazename = mediumGrid.lay

		Performance	Temps
QLearning Tabulaire		70%	1m54s
QLearning Généralisé	Identité	76%	25s
	Pacman	66%	11m50s

Les algorithmes n'ont été lancés qu'une seule fois. Les résultats indiqués ne sont pas toujours similaires dû à l'aspect aléatoire. La performance est censée être relativement la même pour tous les algorithmes, tournant autour des 70%. Seul le temps les différencie vraiment. D'après ce tableau, on remarque que le QLearning Généralisé utilisant la fonction caractéristique Identité est le meilleur dans ce domaine.