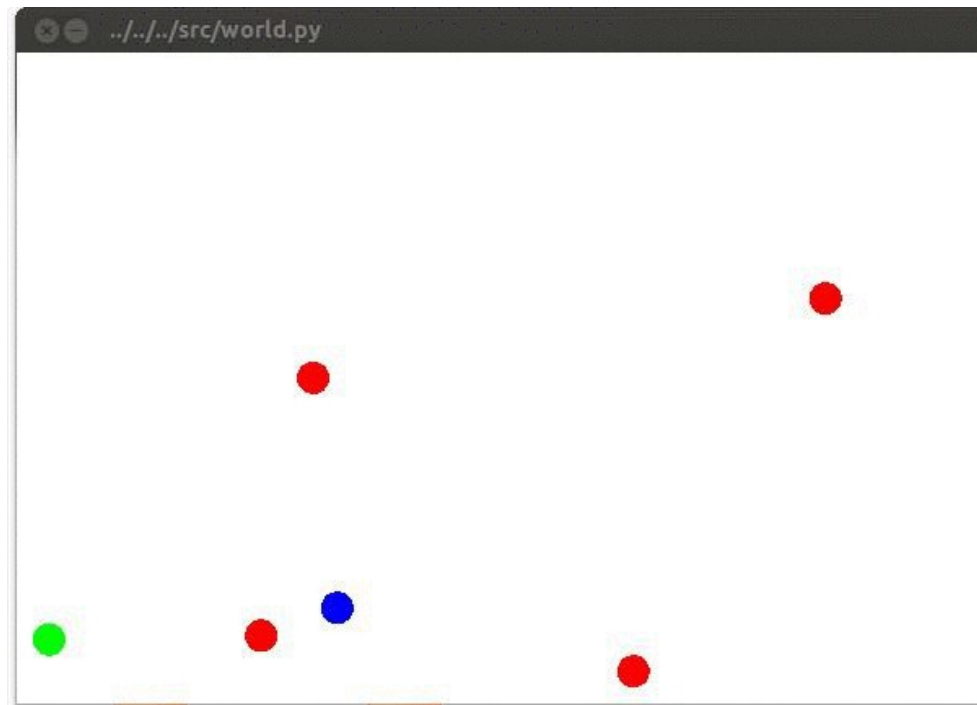


# Étude sur l'apprentissage par renforcement dans les jeux

## Projet de Bachelor 2018 - hepia

Étudiant : Federico Pfeiffer    Professeur : Guido Bologna



h e p i a

Haute école du paysage, d'ingénierie  
et d'architecture de Genève

Hes·SO

Haute Ecole Spécialisée  
de Suisse occidentale  
Fachhochschule Westschweiz  
University of Applied Sciences and Arts  
Western Switzerland

# Plan :

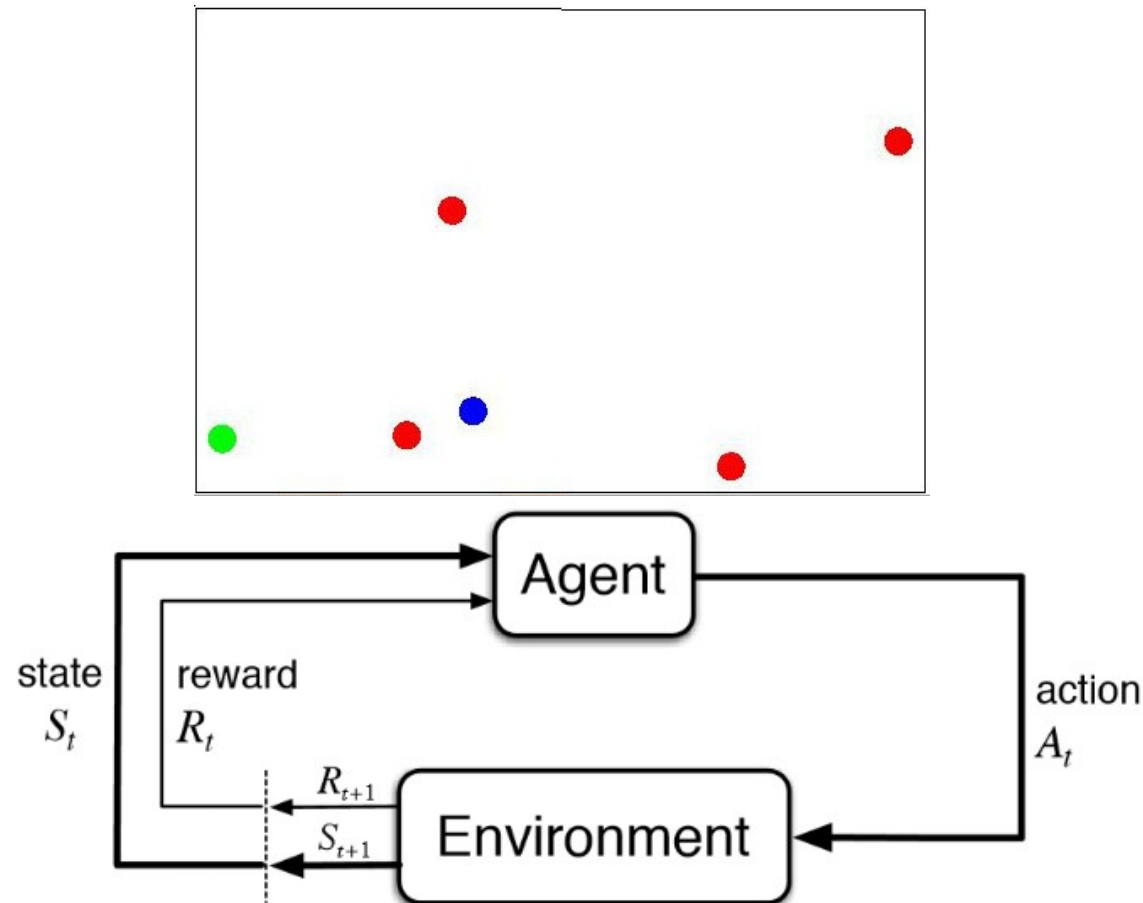
- Cahier des charges
- Apprentissage par renforcement
- Réseaux neuronaux
- Implémentation
- Analyse des résultats
- Conclusion

# Cahier des charges

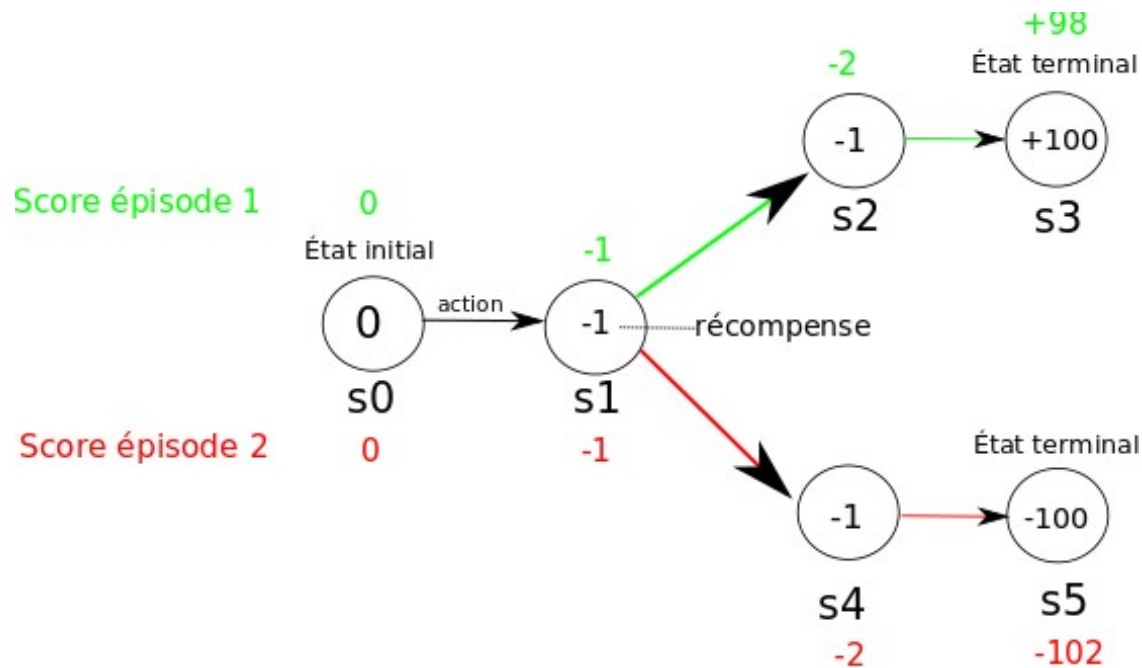
- Comprendre les modèles de réseaux de neurones artificiels standards
- Comprendre les modèles de réseaux de neurones artificiels profonds
- Comprendre les concepts de l'apprentissage par renforcement
- Construire un jeu avec un agent qui apprend par renforcement
- Analyse des résultats
- Rédaction du rapport

# Apprentissage par renforcement

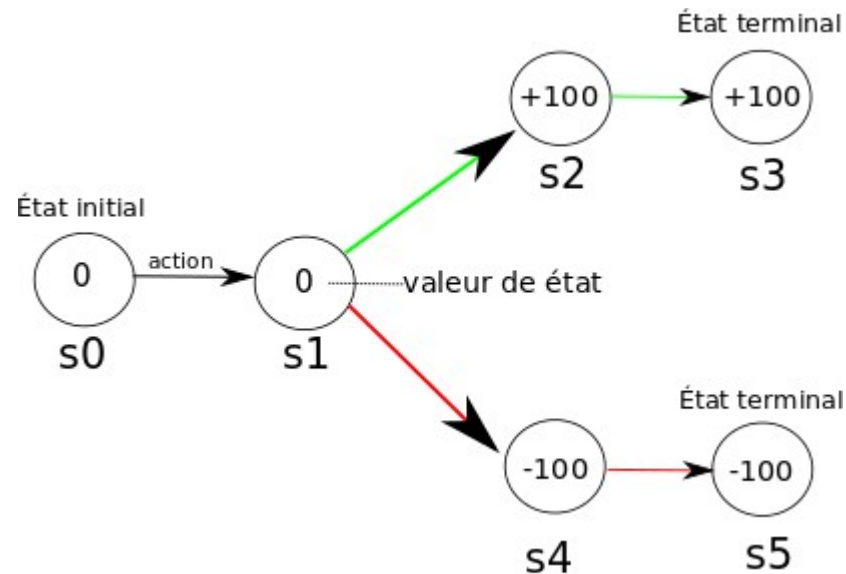
Agent ↔ Environnement



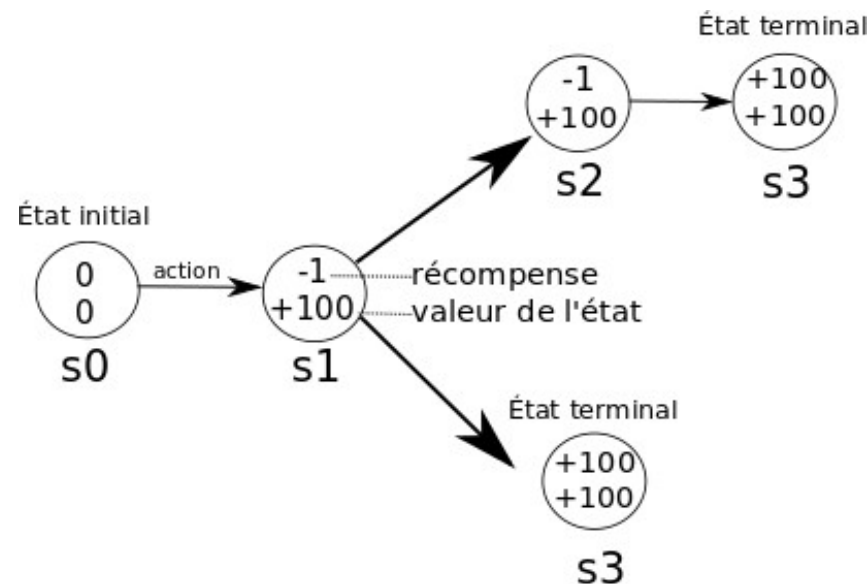
# Choix d'une action: valeur d'un état (1)



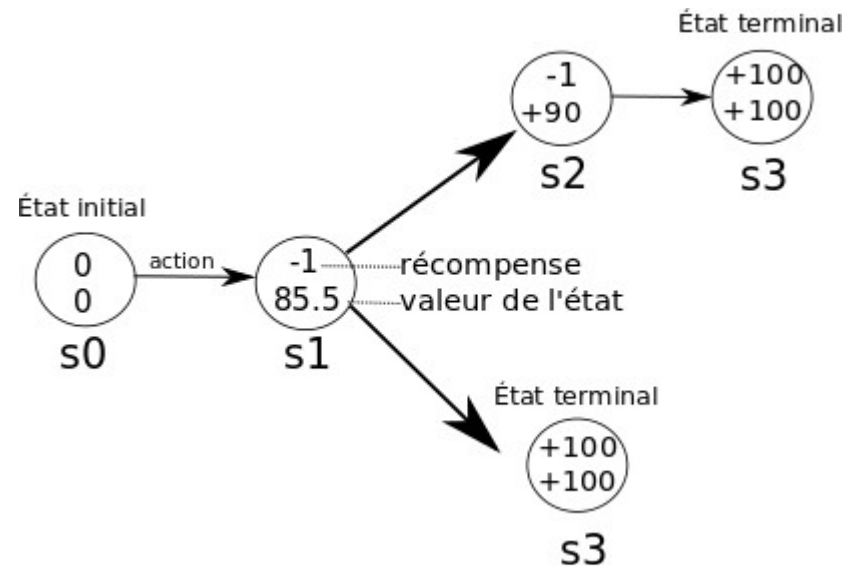
# Choix d'une action: valeur d'un état (2)



# Choix d'une action: valeur d'un état (3)



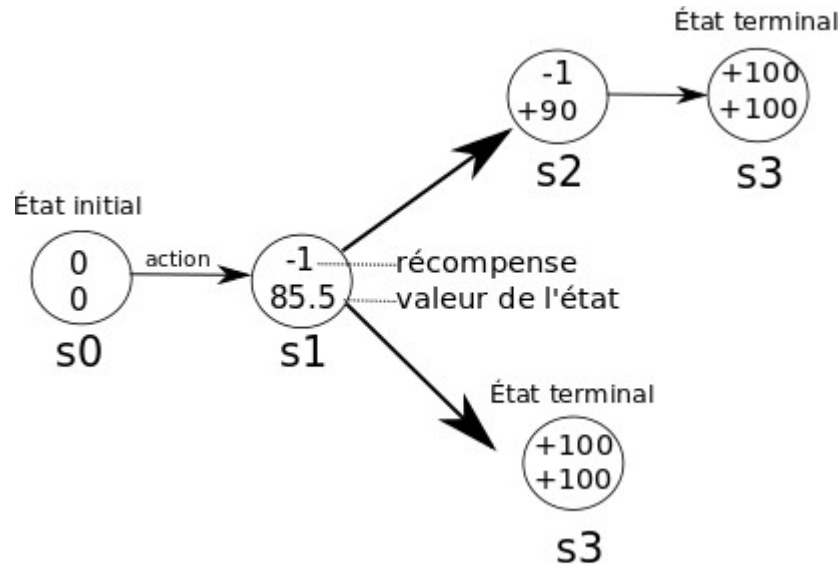
# Choix d'une action: valeur d'un état (3) discount factor ( $\gamma$ )



$$V(s_t) = \gamma V(s_{t+1}) \text{ avec } \gamma \in [0, 1]$$



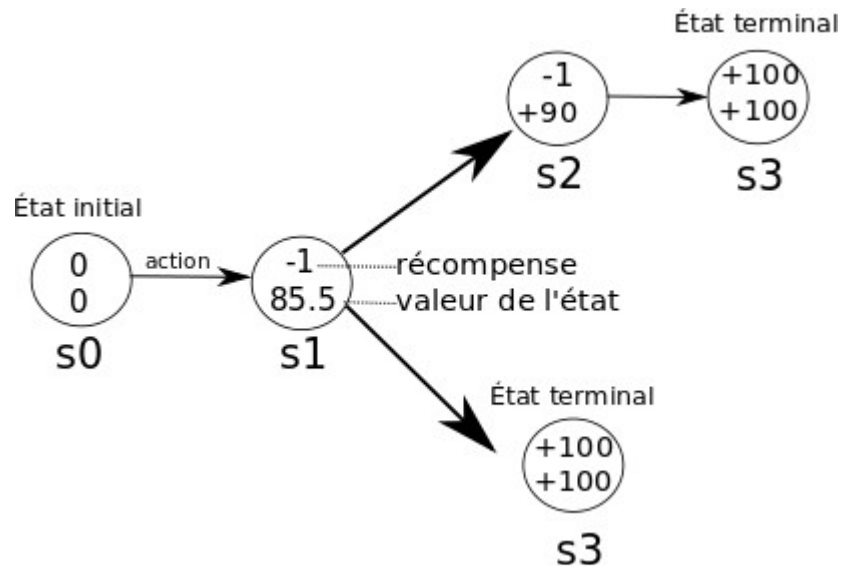
# Choix d'une action: « explore vs exploite » proba aléatoire ( $\varepsilon$ )



$$V(s_t) = \gamma V(s_{t+1}) \text{ avec } \gamma \in [0, 1]$$

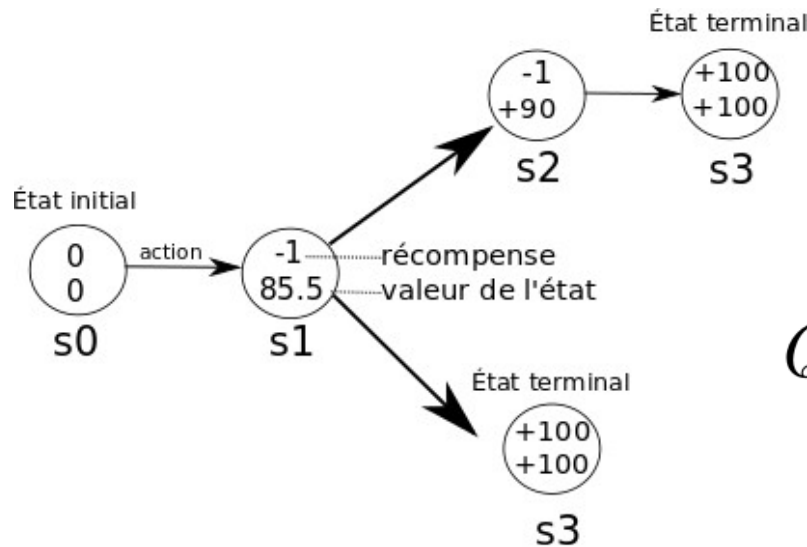
action aléatoire avec  $p = \varepsilon \in [0, 1]$

# Valeur d'un état: équation de Bellman (simplifiée)



$$V_{\pi}(s) = \sum_a \pi(a|s) \{r + \gamma V_{\pi}(s')\}$$

# Valeur d'un état: Q-Learning



$$V_{\pi}(s) = \sum_a \pi(a|s) \{r + \gamma V_{\pi}(s')\}$$

$$Q_{\pi}(s, a) = \{r + \gamma Q_{\pi}(s', a')\}$$

$$Q(s, a) = Q(s, a) + \alpha(\{r + \gamma Q(s', a')\} - Q(s, a))$$

# Apprentissage par renforcement

## Q-Learning : pseudo-code

```
# phase d'apprentissage
world = new World()
agent = new Agent()
max_episodes = 100000
gamma = 0.9
epsilon = 0.1
alpha = 0.1

for i in range(max_episodes) :
    state = world.reset()
    while not world.game_over() :
        action = agent.choose_action(state, epsilon)
        reward, next_state = world.do_action(state, action)
        a2 = agent.choose_action(next_state, epsilon)

        if world.game_over() :
            agent.Q[state][action] += alpha*(reward)
        else :
            agent.Q[state][action] += alpha*(reward+gamma*agent.Q[next_state][a2])

    state = next_state

# phase une fois l'apprentissage effectué
state = world.reset()
while not world.game_over() :
    action = agent.choose_action(state, epsilon)
    state, _ = world.do_action(action)
```

$$Q(s, a) = Q(s, a) + \alpha(\{r + \gamma Q(s', a')\} - Q(s, a))$$

# Q-Learning

## Nécessité de réseaux neuronaux

$V(s)$

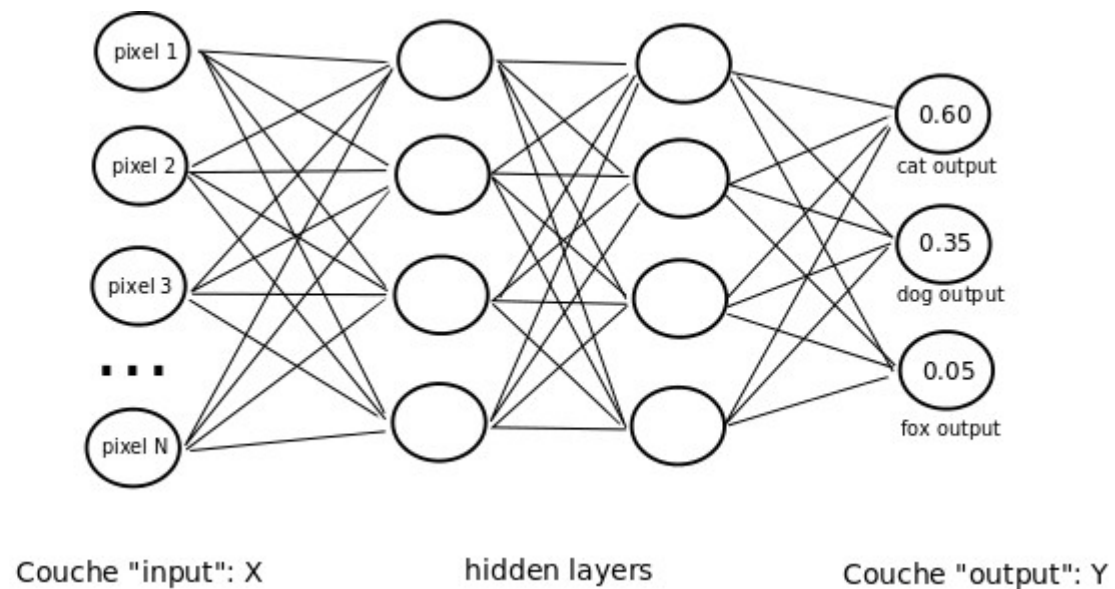
valeur état 1	valeur état 2	valeur état 3	...	valeur état N
------------------	------------------	------------------	-----	------------------

$Q(s,a)$

s1	valeur s1/a1	valeur s1/a2	valeur s1/a3	...	valeur s1/aM
s2	valeur s2/a1	valeur s2/a2	valeur s2/a3	...	valeur s2/aM
s3	valeur s3/a1	valeur s3/a2	valeur s3/a3	...	valeur s3/aM
...					
sN	valeur sN/a1	valeur sN/a2	valeur sN/a3	...	valeur sN/aM

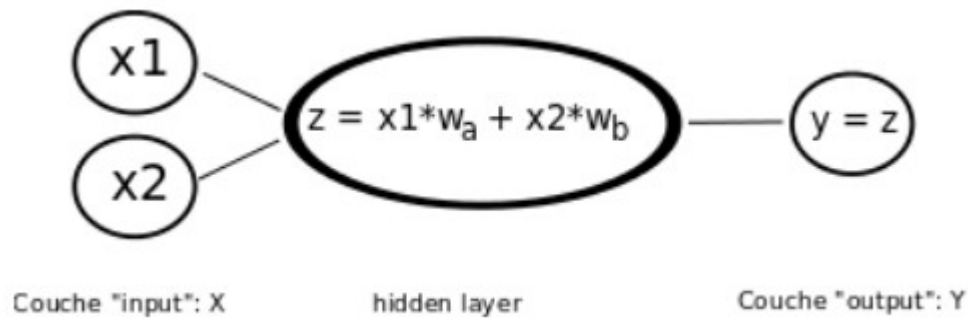
# Réseaux Neuronaux

Utilité : estimer la valeur d'un état  $Q(s,a)$



# Réseau de neurones « standard »

## Prédiction

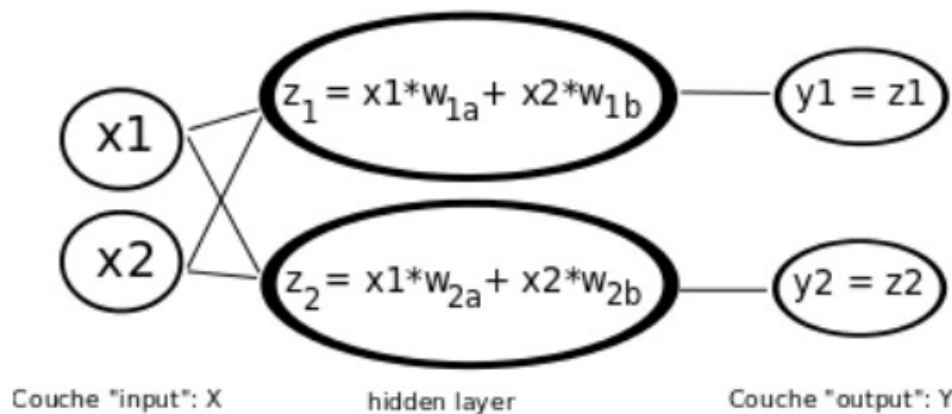


$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$W = (w_a, w_b)$$

$$Z = WX = (w_a, w_b) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = (x_1 w_a + x_2 w_b)$$

$$Y = Z = (x_1 w_a + x_2 w_b)$$



$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

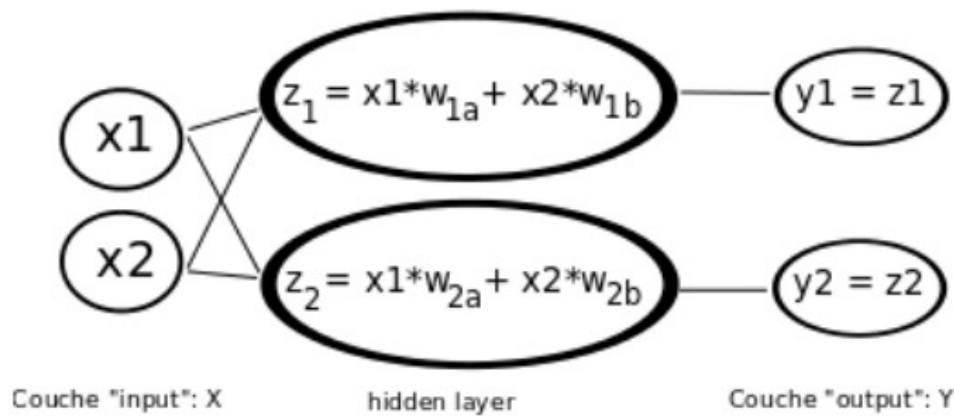
$$W = \begin{pmatrix} w_{1a}, w_{1b} \\ w_{2a}, w_{2b} \end{pmatrix}$$

$$Z = WX = \begin{pmatrix} w_{1a}, w_{1b} \\ w_{2a}, w_{2b} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} w_{1a}x_1 + w_{1b}x_2 \\ w_{2a}x_1 + w_{2b}x_2 \end{pmatrix}$$

$$Y = Z = \begin{pmatrix} w_{1a}x_1 + w_{1b}x_2 \\ w_{2a}x_1 + w_{2b}x_2 \end{pmatrix}$$

# Réseau de neurones « standard »

## Apprentissage (back-propagation)



$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$W = \begin{pmatrix} w_{1a}, w_{1b} \\ w_{2a}, w_{2b} \end{pmatrix}$$

$$Z = WX = \begin{pmatrix} w_{1a}, w_{1b} \\ w_{2a}, w_{2b} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} w_{1a}x_1 + w_{1b}x_2 \\ w_{2a}x_1 + w_{2b}x_2 \end{pmatrix}$$

$$Y = Z = \begin{pmatrix} w_{1a}x_1 + w_{1b}x_2 \\ w_{2a}x_1 + w_{2b}x_2 \end{pmatrix}$$

$$W = W + \alpha \frac{dJ(Y, T)}{dW}$$



# Réseau de neurones profonds

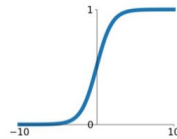
- Rendre les réseaux non-linéaires

$$Z = f(WX + b)$$

## Activation Functions

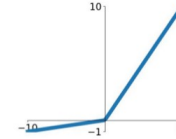
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



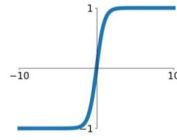
**Leaky ReLU**

$$\max(0.1x, x)$$



**tanh**

$$\tanh(x)$$

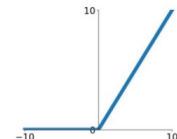


**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

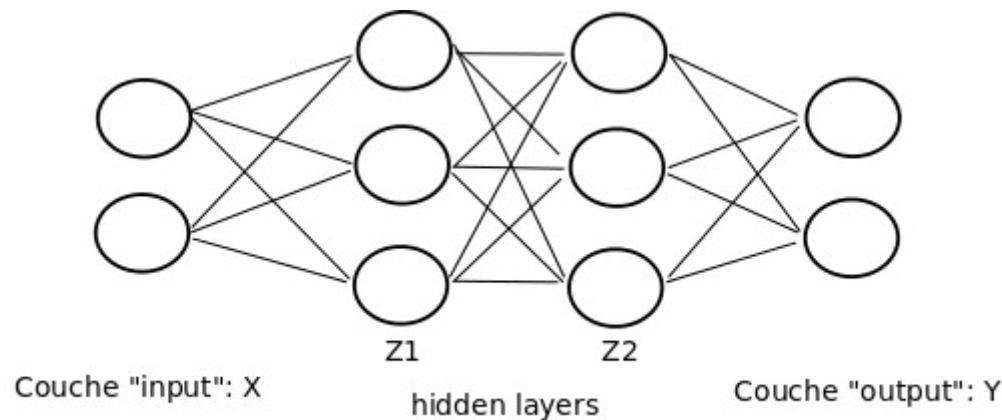
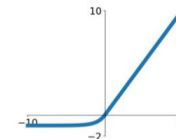
**ReLU**

$$\max(0, x)$$



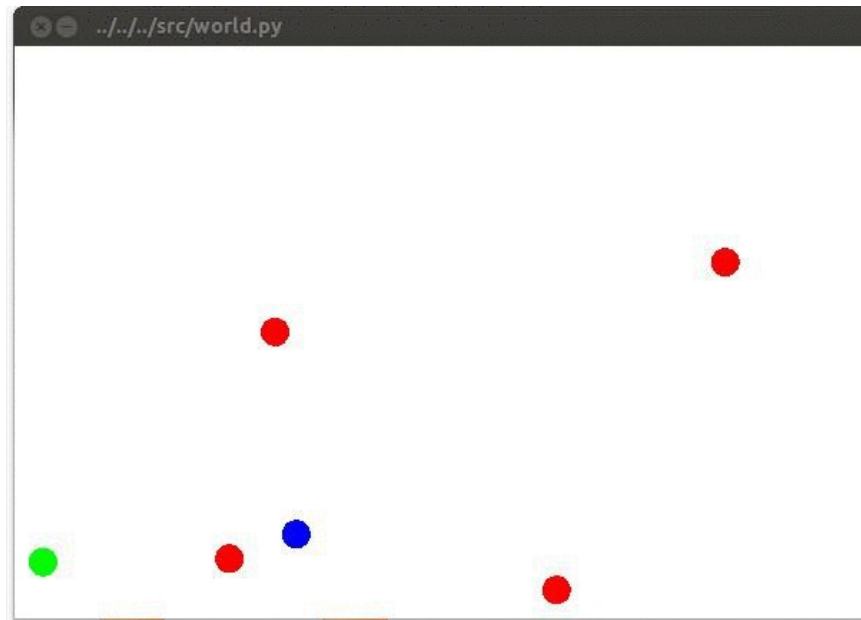
**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# Implémentation :

## Rappel environnement



# Implémentation (2): rappel pseudo-code

```
# phase d'apprentissage
world = new World()
agent = new Agent()
max_episodes = 100000
gamma = 0.9
epsilon = 0.1
alpha = 0.1
```

$$Q(s, a) = Q(s, a) + \alpha(\{r + \gamma Q(s', a')\} - Q(s, a))$$

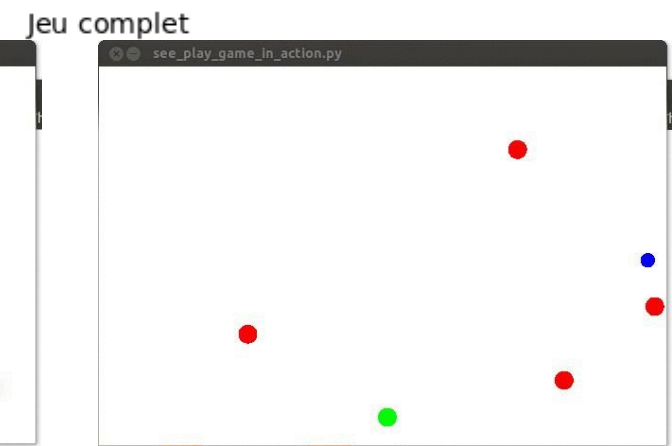
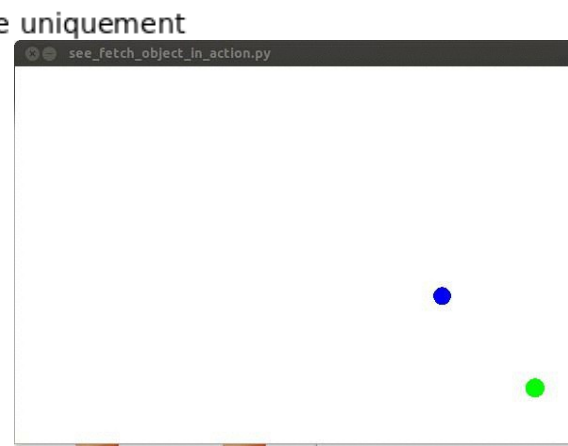
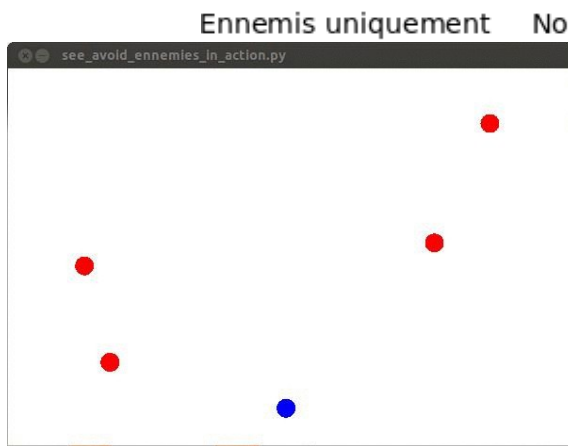
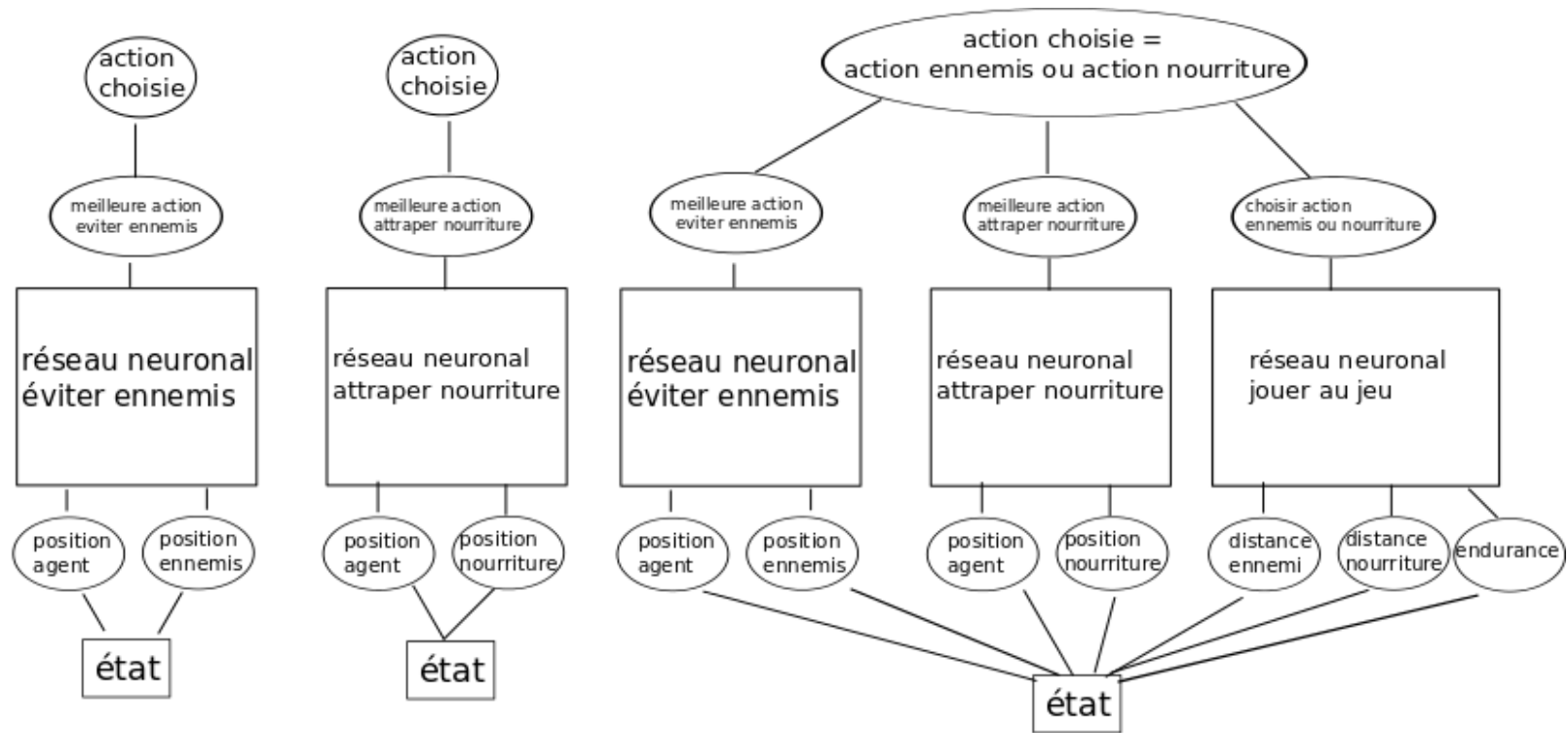
```
for i in range(max_episodes) :
    state = world.reset()
    while not world.game_over() :
        action = agent.choose_action(state, epsilon)
        reward, next_state = world.do_action(state, action)
        a2 = agent.choose_action(next_state, epsilon)

        if world.game_over() :
            agent.Q[state][action] += alpha*(reward)
        else :
            agent.Q[state][action] += alpha*(reward+gamma*agent.Q[next_state][a2])

    state = next_state

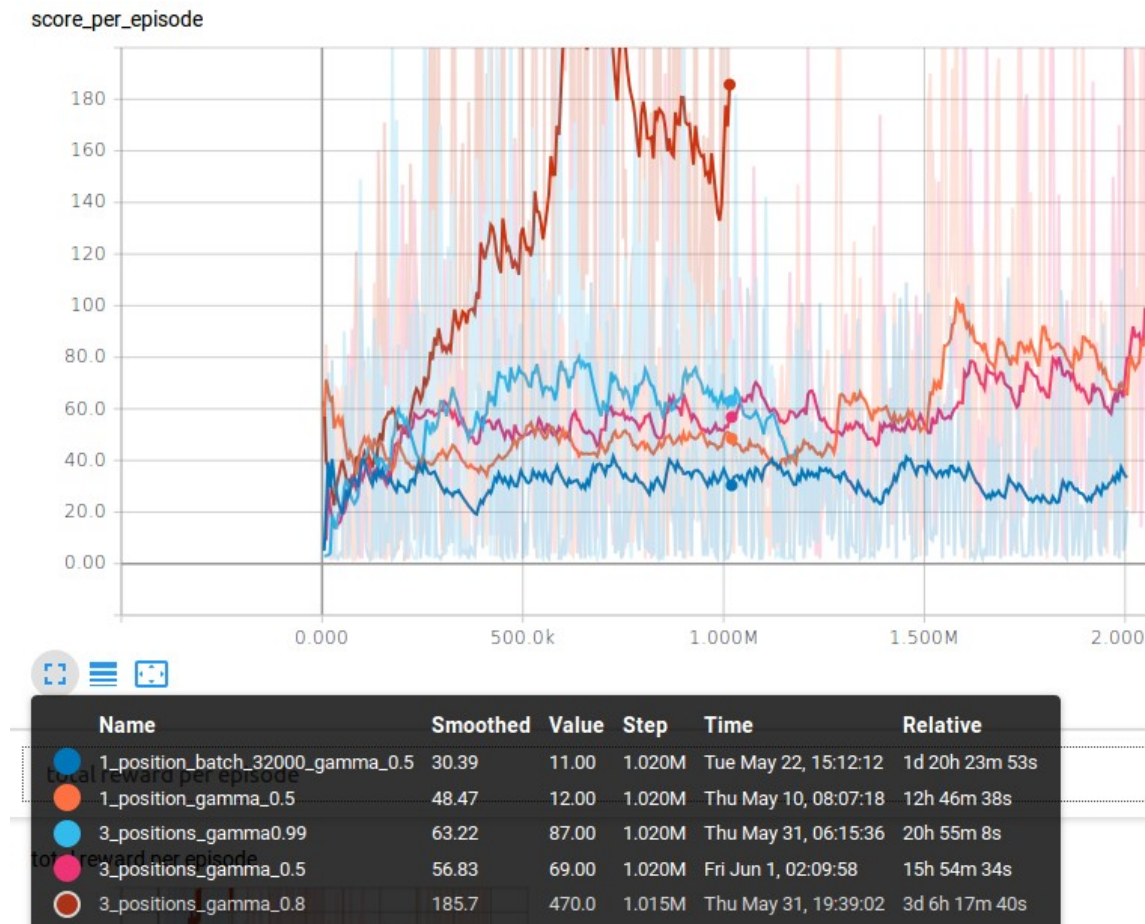
# phase une fois l'apprentissage effectué
state = world.reset()
while not world.game_over() :
    action = agent.choose_action(state, epsilon)
    state, _ = world.do_action(action)
```

# Implémentation (3): phases d'apprentissage



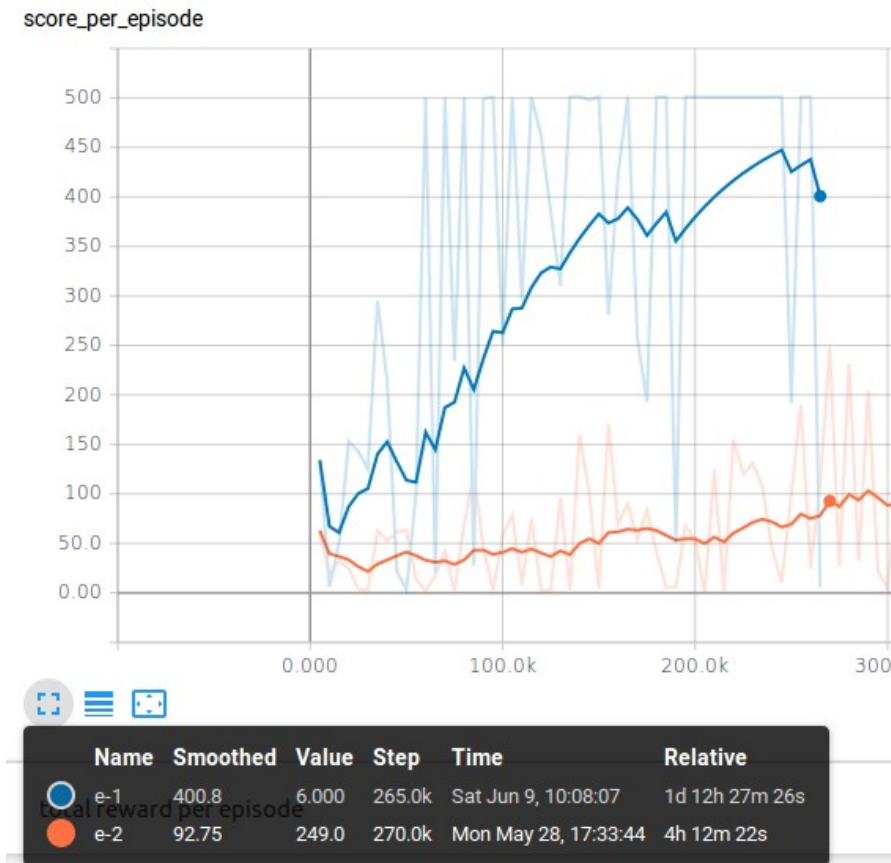
# Résultats

- Choisir adéquatement la manière de représenter l'environnement
- Choix du discount factor



# Résultats (2)

→ Learning rate ( $\alpha$ )  $Q(s, a) = Q(s, a) + \alpha(\{r + \gamma Q(s', a')\} - Q(s, a))$



# Conclusion

- Domaine pas enseigné à l'hepia
- Théorie éloignée de la pratique
- Domaine évolue sans cesse
- Architecture pas adaptée au jeu construit

