

# **IFT 615 – Intelligence artificielle**

## **Processus de décision markoviens (MDP)**

Hugo Larochelle

Département d'informatique

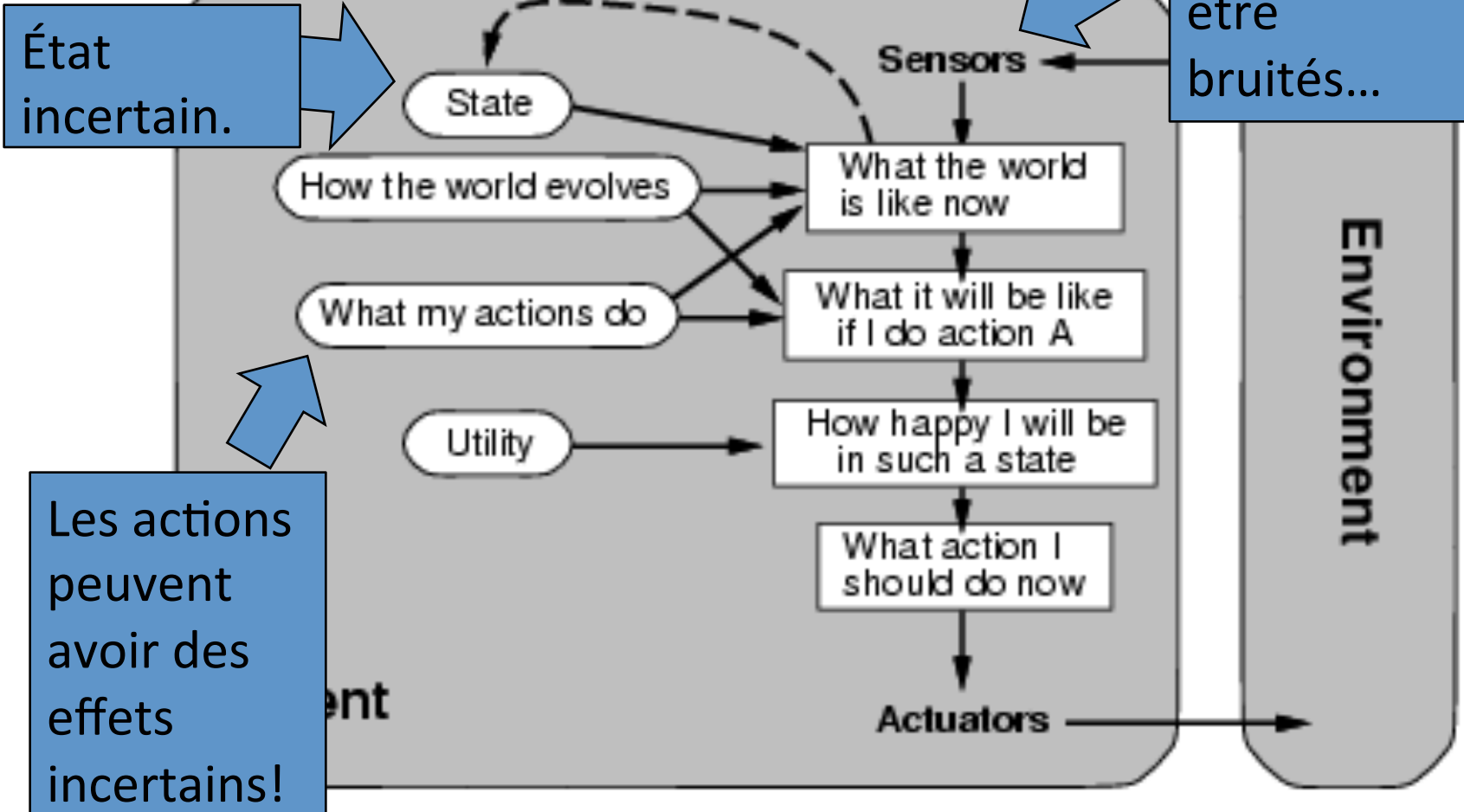
Université de Sherbrooke

<http://www.dmi.usherb.ca/~larocheh/cours/ift615.html>

# Sujets couverts

- Processus de décision de Markov (MDP)
- Algorithme d'itération par valeurs (*value-iteration*)
- Algorithme d'itération par politiques (*policy-iteration*)

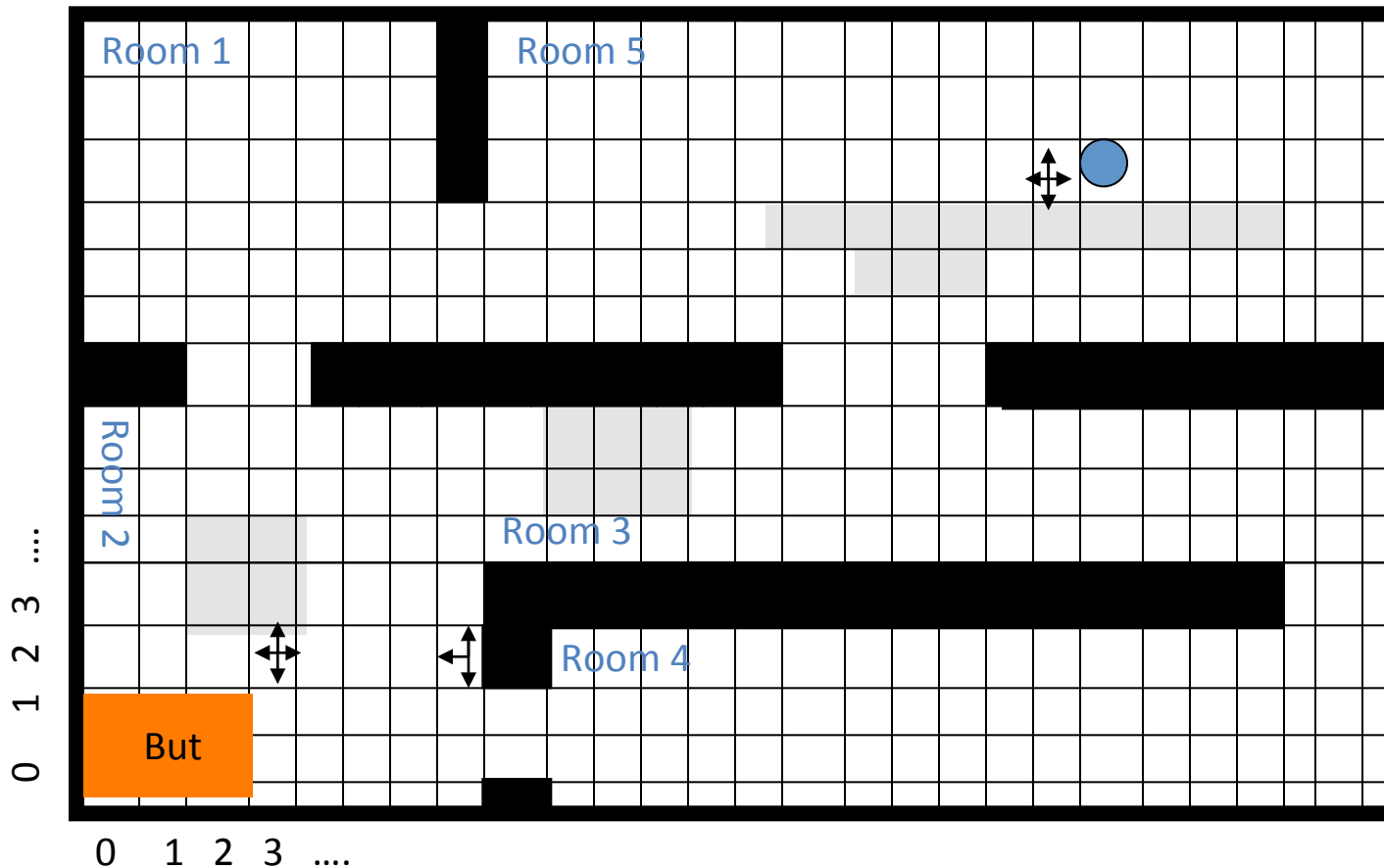
## Rappel: Utility-based agent



# Incertitude dans le contexte d'une décision

- Soit  $A_t$  l'action d'aller à l'aéroport  $t$  minutes avant le départ de l'avion
- $A_t$  me permettra-t-il d'arriver à temps?
- Problèmes:
  - ◆ observabilité partielle (conditions routières, etc.)
  - ◆ senseurs bruités (annonces du trafic, etc.)
  - ◆ incertitude dans l'effet des actions (crevaisons, pannes, etc.)
  - ◆ immense complexité pour modéliser les actions et le trafic

# Grille (occupancy grid)



## Actions:

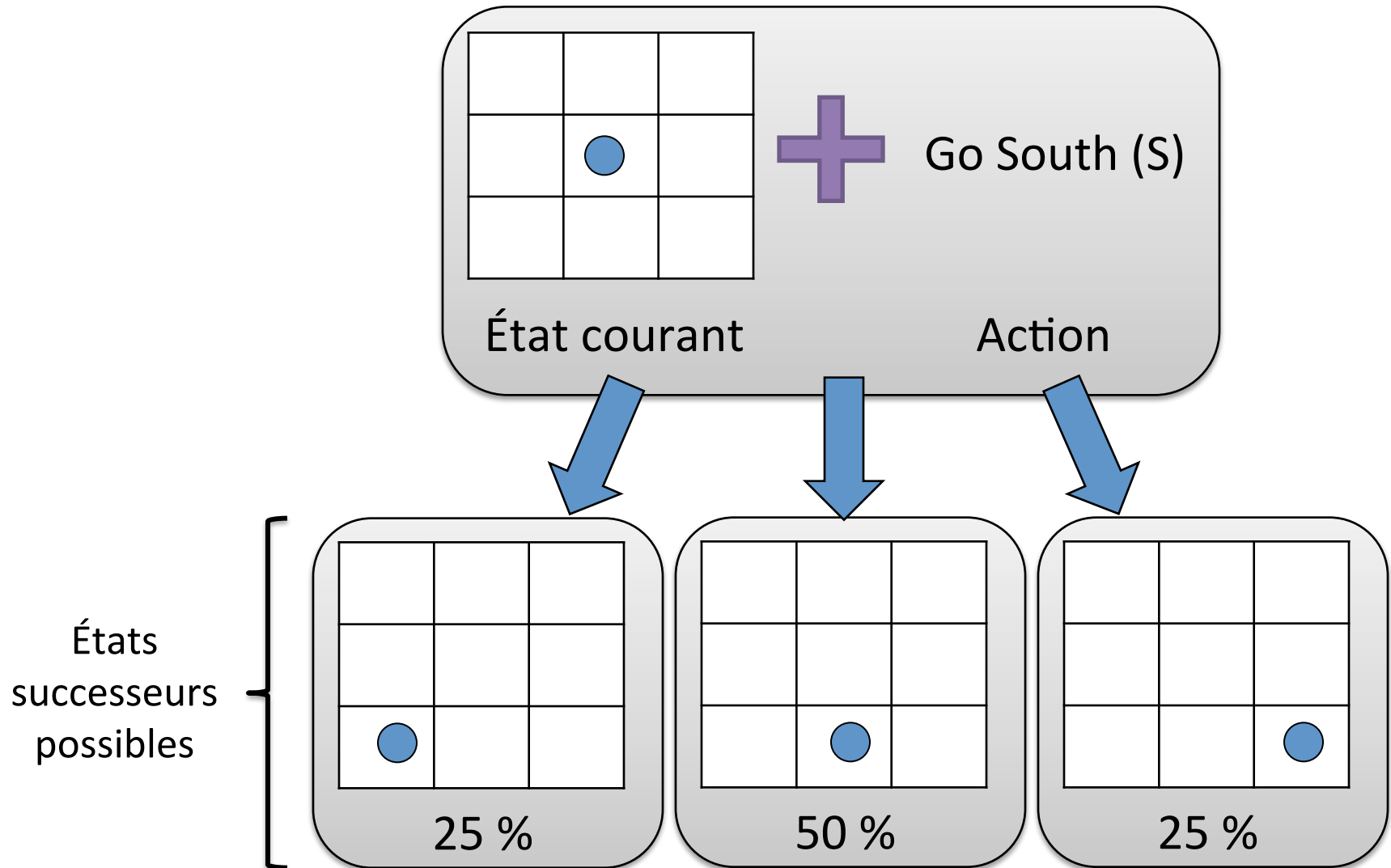
- E: Go east
- W: Go west
- S: Go south
- N: Go north

## Degré de désirabilité

0    -0.4

**But** : +1

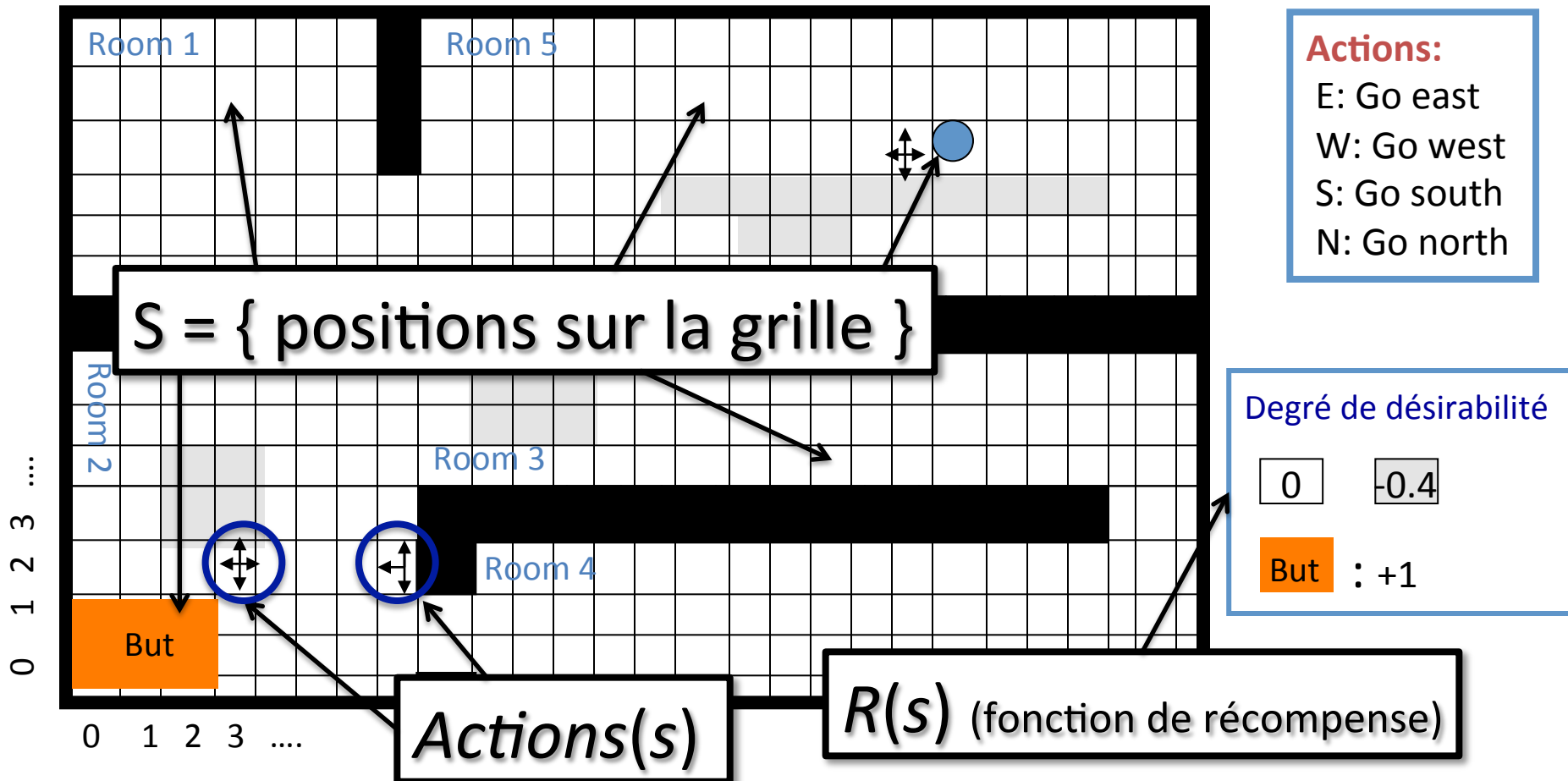
# Actions aux effets incertains



# Processus de décision markovien

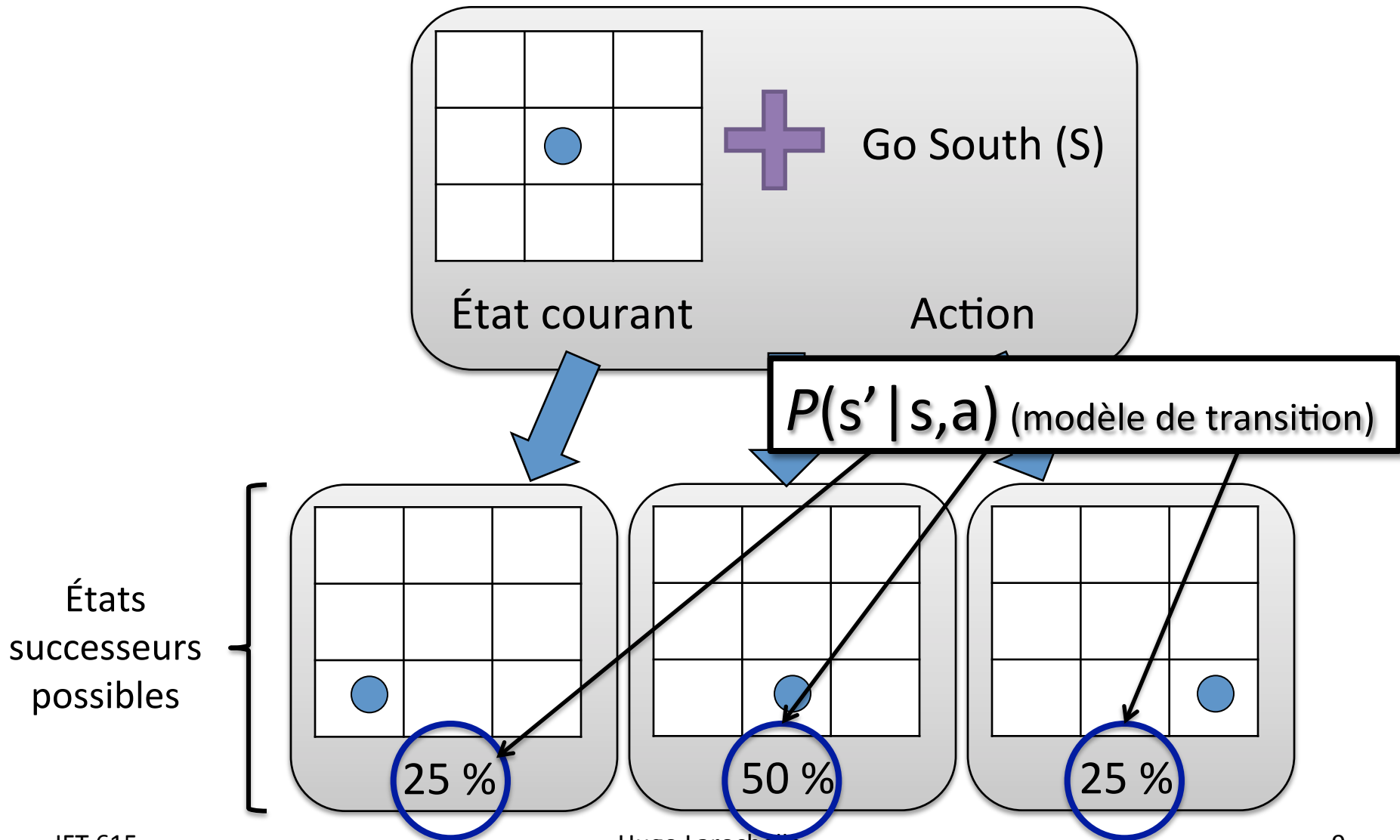
- Un **processus de décision markovien** (*Markov decision process*, ou **MDP**) est défini par:
  - ◆ un **ensemble d'états**  $S$  (incluant un état initial  $s_0$ )
  - ◆ un **ensemble d'actions** possibles  $Actions(s)$  lorsque je me trouve à l'état  $s$
  - ◆ un **modèle de transition**  $P(s'|s, a)$ , où  $a \in A(s)$
  - ◆ une **fonction de récompense**  $R(s)$  (utilité d'être dans l'état  $s$ )
- Un MDP est donc un modèle général pour un environnement stochastique dans lequel un agent peut prendre des décisions et reçoit des récompenses
- On y fait une supposition markovienne (de premier ordre) sur la distribution des états visités
- Requière qu'on décrive un objectif à atteindre à partir d'une fonction de récompense basée seulement sur l'état courant

# Grille (occupancy grid)





# Actions aux effets incertains



# Décision

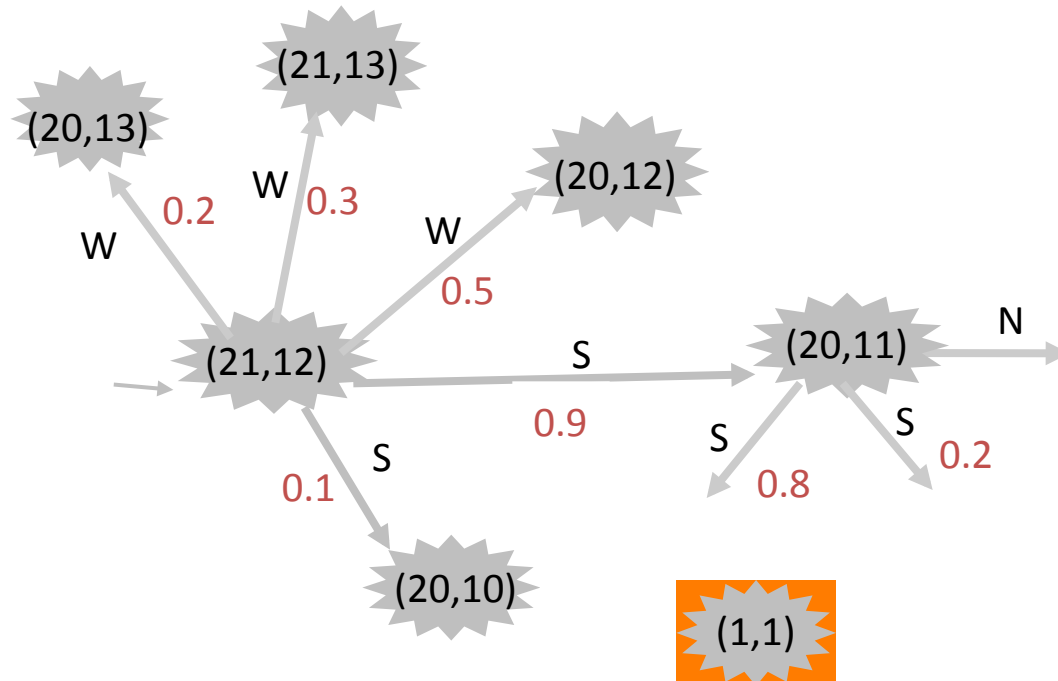
- Une **décision** est un choix d'une action dans un état
  - ♦ c'est une règle « *if state then action* »

## Exemples:

$(21,12) \rightarrow W$

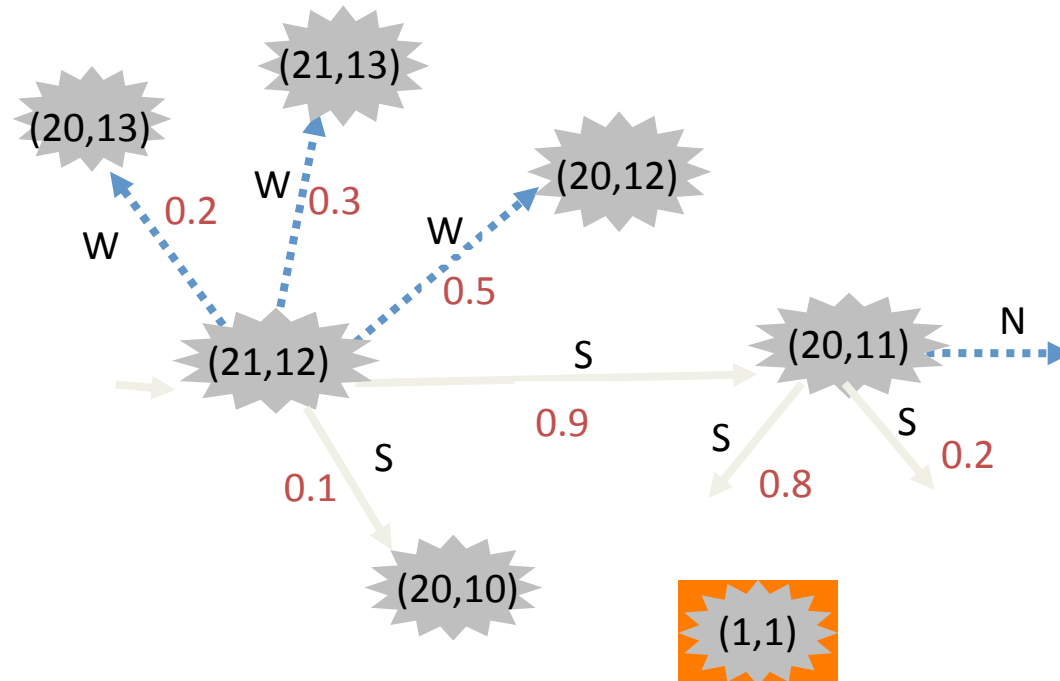
ou

$(21,12) \rightarrow E$



# Plan (politique)

- Un **plan** est une stratégie: choix d'une action (décision) **pour chaque état**
  - ◆ un plan est également appelé une politique (*policy*)
  - ◆ c'est un **ensemble** de règles *if state then action*



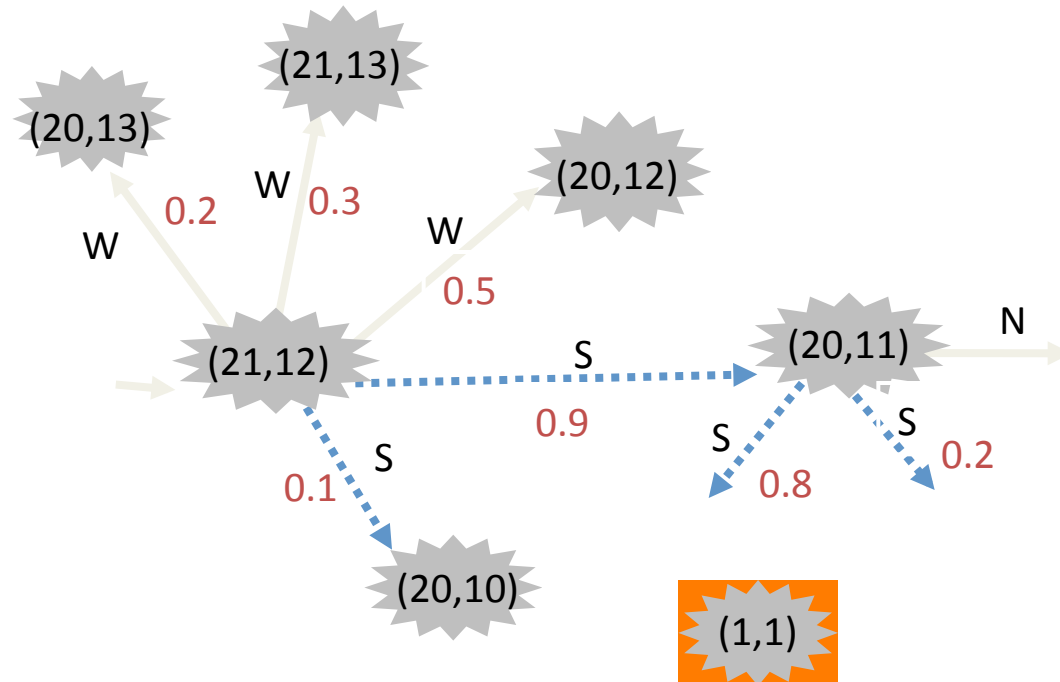
## Exemples:

### Plan $\pi_1$

$\{ (21,12) \rightarrow W,$   
 $(20,13) \rightarrow S,$   
 $(21,13) \rightarrow S,$   
 $(20,11) \rightarrow N,$   
 $\dots$   
 $\}$

# Plan (politique)

- Un **plan** est une stratégie: choix d'une action (décision) **pour chaque état**
  - ◆ un plan est également appelé une politique (*policy*)
  - ◆ c'est un **ensemble** de règles *if state then action*



## Exemples:

### Plan $\pi_1$

$\{ (21,12) \rightarrow W,$   
 $(20,13) \rightarrow S,$   
 $(21,13) \rightarrow S,$   
 $(20,11) \rightarrow N,$   
 $\dots$   
 $\}$

### Plan $\pi_2$

$\{ (21,12) \rightarrow S,$   
 $(20,11) \rightarrow S,$   
 $(21,10) \rightarrow E,$   
 $\dots \}$

# Exécution d'un plan (politique)

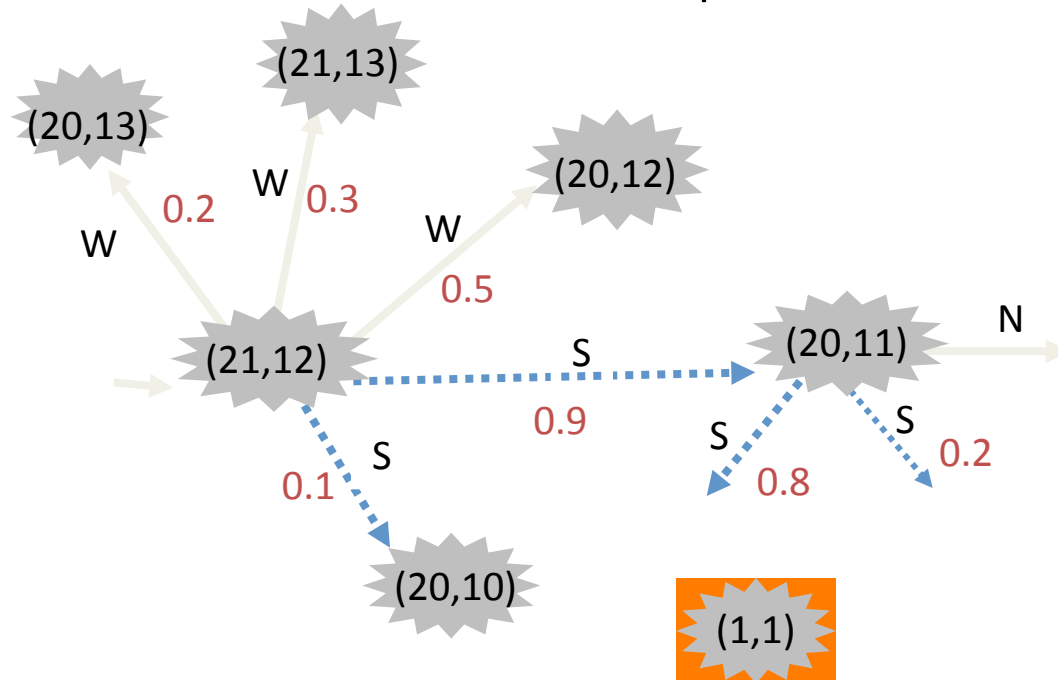
- Un plan est un ensemble de règles *if state then action*
- Notons  $\pi(s)$  l'action désignée par le plan  $\pi$  dans l'état  $s$
- Voici un algorithme d'exécution ou d'application d'un plan

```
While (1)
{
  1.  $s$  = état courant du système;
  2.  $a = \pi(s)$ ;
  3. execute  $a$ ;
}
```

- L'étape 1 peut impliquer de la détection (*sensing*) et de la localisation
- L'état résultant de l'exécution de l'action à l'étape 3 est imprévisible

# Interprétation/application d'un plan

- L'application d'un plan dans un MDP **résulte en une chaîne de Markov** sur les états, avec une matrice de transition dont les entrées sont données par  $P(s'|s, \pi(s))$
- La chaîne se déroule en un arbre potentiellement infini



## Exemples:

### Plan $\pi_1$

$\{ (21,12) \rightarrow W,$   
 $(20,13) \rightarrow S,$   
 $(21,13) \rightarrow S,$   
 $(20,11) \rightarrow N,$   
 $\dots$   
 $\}$

### Plan $\pi_2$

$\{ (21,12) \rightarrow S,$   
 $(20,11) \rightarrow S,$   
 $(21,10) \rightarrow E,$   
 $\dots \}$

# Valeur d'un plan

- $R(s)$ : **récompense** pour l'état  $s$ , c-à-d. l'utilité de l'état  $s$
- $V(\pi, s)$ : **valeur** du plan  $\pi$  à l'état  $s$

$$V(\pi, s) = \underbrace{R(s)}_{\text{récompense actuelle}} + \gamma \underbrace{\sum_{s' \in S} P(s' | s, \pi(s)) V(\pi, s')}_{\text{somme des récompenses futures espérée}}$$

- ◆  $\gamma$ : facteur d'escompte ( $0 < \gamma \leq 1$ ), indique l'importance relative des récompenses futures par rapport à la récompense actuelle
- ◆  $S$ : espace d'états
- ◆  $\pi(s)$ : action du plan à l'état  $s$
- ◆  $P(s' | s, \pi(s))$ : probabilité de la transition du MDP

# Plan optimal

- Un plan  $\pi$  **domine** un plan  $\pi'$  si les deux conditions suivantes sont réunies:
  - ◆  $V(\pi, s) \geq V(\pi', s)$  pour tout état  $s$
  - ◆  $V(\pi, s) > V(\pi', s)$  pour au moins un  $s$
- Un plan est **optimal** s'il n'est pas dominé par un autre
  - ◆ il peut y avoir plusieurs plans optimaux, mais ils ont tous la même valeur
  - ◆ on peut avoir deux plans **incomparables** (aucun ne domine l'autre)
    - » la dominance induit une fonction d'ordre partiel sur les plans
- Deux algorithmes différents pour le calcul du plan optimal:
  - ◆ **itération par valeurs** (*value iteration*)
  - ◆ **itération par politiques** (*policy iteration*)



# Équations de Bellman pour la valeur optimale

- Les **équations de Bellman** nous donnent une condition qui est garantie par la valeur  $V^*$  des plans optimaux

$$V^*(s) = R(s) + \max_a \gamma \sum_{s' \in S} P(s' | s, a) V^*(s') \quad \forall s \in S$$

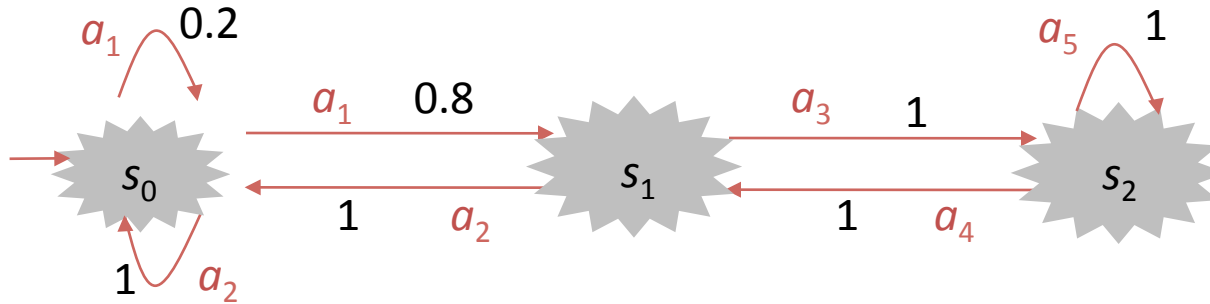
- Si nous pouvons calculer  $V^*$ , nous pourrions calculer un plan optimal aisément:
  - ◆ il suffit de choisir dans chaque état  $s$  l'action qui maximise  $V^*(s)$  (c.-à-d. le argmax)

# Algorithme *Value Iteration*

1. Initialiser  $V(s)$  à 0 pour chaque état  $s$ .
2. Répéter (jusqu'à ce que le changement en  $V$  soit négligeable).
  - I. pour chaque état  $s$  calculer:
$$V'(s) = R(s) + \max_a \gamma \sum_{s' \in S} P(s'|s,a) V(s')$$
  - II. si  $\sum_{s \in S} |V(s) - V'(s)| \leq \text{tolérance}$ , quitter
  - III.  $V \leftarrow V'$
3. Dériver le plan optimal en choisissant l'**action  $a$  ayant la meilleure récompense future espérée**, pour chaque état  $s$ 
  - I.  $\pi(s) = \operatorname{argmax}_a \sum_{s' \in S} P(s'|s,a) V(s')$

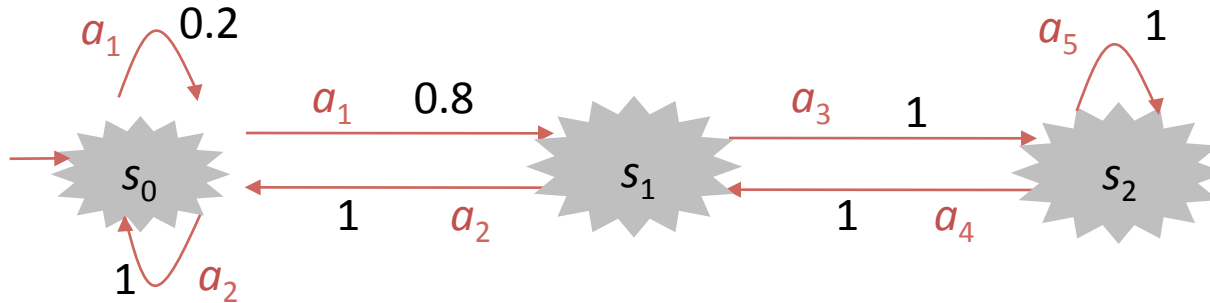
- En mots, on choisit l'action qui **maximise l'espérance** des sommes de récompenses futures
- Converge au plan optimal en  $O(|S|^3)$

# Exemple de MDP



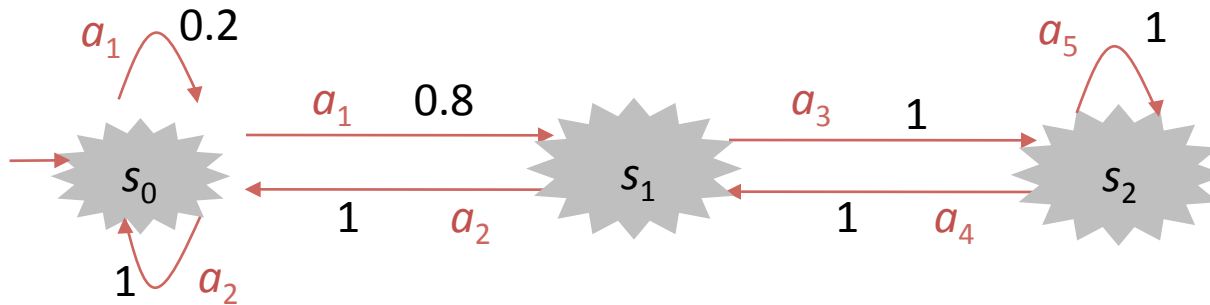
- MDP à 3 états:  $S = \{s_0, s_1, s_2\}$
- But:  $s_2$

# Exemple de MDP



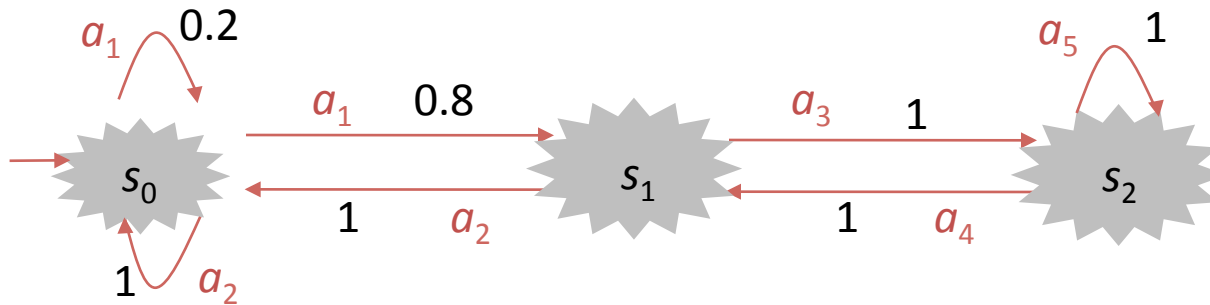
- MDP à 3 états:  $S = \{s_0, s_1, s_2\}$
- Le but (atteindre  $s_2$ ) est exprimé par une fonction de récompense:
  - ◆  $R(s_0) = 0, R(s_1) = 0, R(s_2) = 1$
- Le facteur d'escompte est  $\gamma=0.5$

# Exemple de MDP



- $V(\pi, s)$ : valeur du plan  $\pi$  dans l'état  $s$ 
  - ◆  $V(\pi, s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V(\pi, s')$
- Notons  $r_i = R(s_i)$  et  $v_i = V(\pi, s_i)$ :
  - ◆  $v_i = r_i + \gamma \sum_j P(s_j|s_i, \pi(s_i)) v_j$

# Value iteration: initialisation



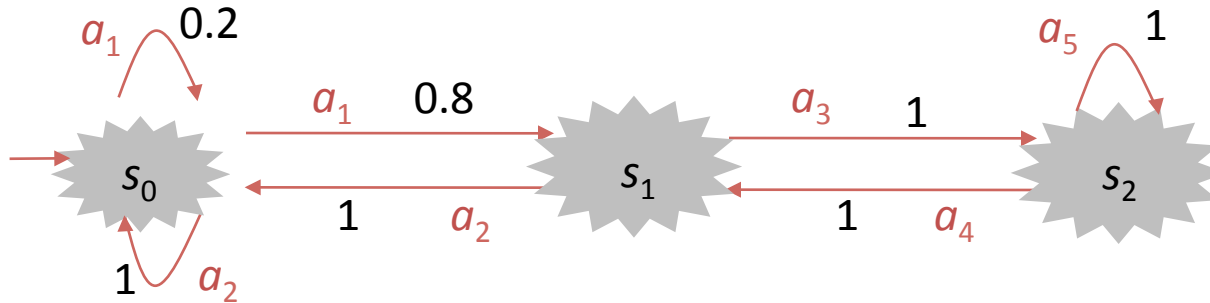
- Valeurs initiales fixées à 0:

$$v_0 = 0$$

$$v_1 = 0$$

$$v_2 = 0$$

# Value iteration: itération #1



- Mise à jour droite-gauche des valeurs

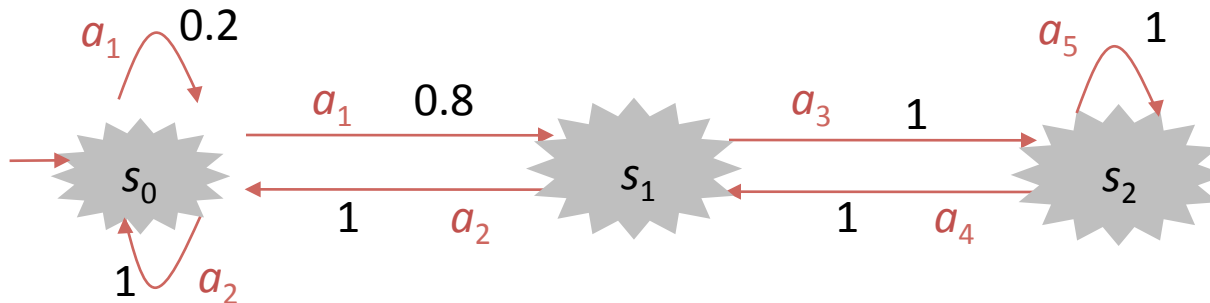
$$v_0 \leftarrow 0 + 0.5 \max\{ 0.2 v_0 + 0.8 v_1, v_0 \} = 0 + 0.5 \max\{ 0, 0 \} = 0$$

$$v_1 \leftarrow 0 + 0.5 \max\{ v_0, v_2 \} = 0 + 0.5 \max\{ 0, 0 \} = 0$$

$$v_2 \leftarrow 1 + 0.5 \max\{ v_1, v_2 \} = 1 + 0.5 \max\{ 0, 0 \} = 1$$

- Les nouvelles valeurs sont  $v_0 = 0$ ,  $v_1 = 0$ ,  $v_2 = 1$

## Value iteration: itération #2



- Mise à jour droite-gauche des valeurs

$$v_0 \leftarrow 0 + 0.5 \max\{ 0.2 v_0 + 0.8 v_1, v_0 \} = 0 + 0.5 \max\{ 0, 0 \} = 0$$

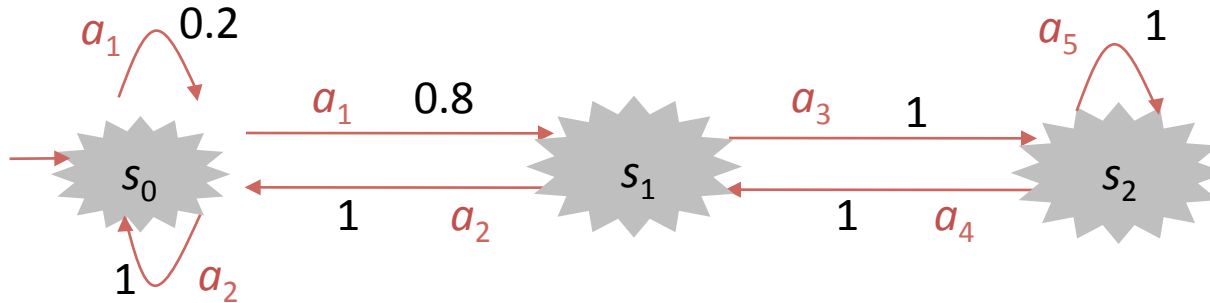
$$v_1 \leftarrow 0 + 0.5 \max\{ v_0, v_2 \} = 0 + 0.5 \max\{ 0, 1 \} = 0.5$$

$$v_2 \leftarrow 1 + 0.5 \max\{ v_1, v_2 \} = 1 + 0.5 \max\{ 0, 1 \} = 1.5$$

- Les nouvelles valeurs sont  $v_0 = 0$ ,  $v_1 = 0.5$ ,  $v_2 = 1.5$



## Value iteration: itération #3



- Mise à jour droite-gauche des valeurs

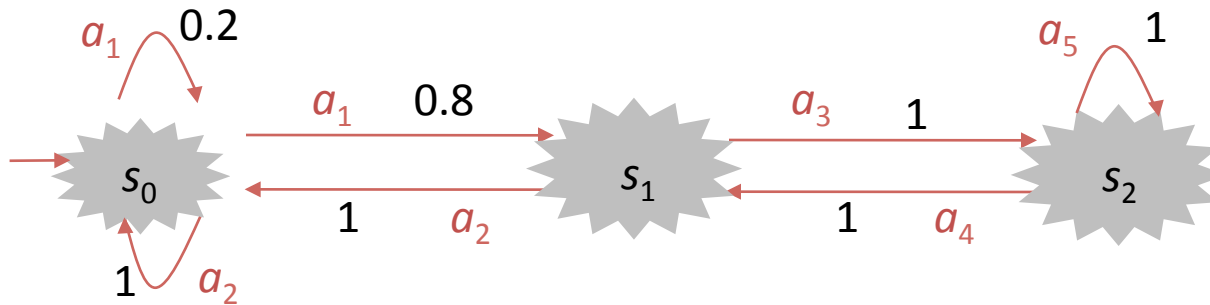
$$v_0 \leftarrow 0 + 0.5 \max\{ 0.2 v_0 + 0.8 v_1, v_0 \} = 0 + 0.5 \max\{ 0.8 * 0.5, 0 \} = 0.2$$

$$v_1 \leftarrow 0 + 0.5 \max\{ v_0, v_2 \} = 0 + 0.5 \max\{ 0, 1.5 \} = 0.75$$

$$v_2 \leftarrow 1 + 0.5 \max\{ v_1, v_2 \} = 1 + 0.5 \max\{ 0.5, 1.5 \} = 1.75$$

- Les nouvelles valeurs sont  $v_0 = 0.2$ ,  $v_1 = 0.75$ ,  $v_2 = 1.75$

## Value iteration: itération #3



- Si on arrêta à la 3<sup>e</sup> itération, le plan retourné serait

$$\pi(s_0) = \operatorname{argmax}\{ 0.2 v_0 + 0.8 v_1, v_0 \} = \operatorname{argmax}\{ 0.2*0.2+0.8*0.75, 0.2 \} = a_1$$

$$\pi(s_1) = \operatorname{argmax}\{ v_0, v_2 \} = \operatorname{argmax}\{ 0.2, 1.75 \} = a_3$$

$$\pi(s_2) = \operatorname{argmax}\{ v_1, v_2 \} = \operatorname{argmax}\{ 0.75, 1.75 \} = a_5$$

- Même si les valeurs n'ont pas tout à fait convergé, on a déjà le plan optimal
  - ◆ ça aurait pu ne pas être le cas, seulement garanti si on boucle infiniment

# Démonstration de *value iteration*

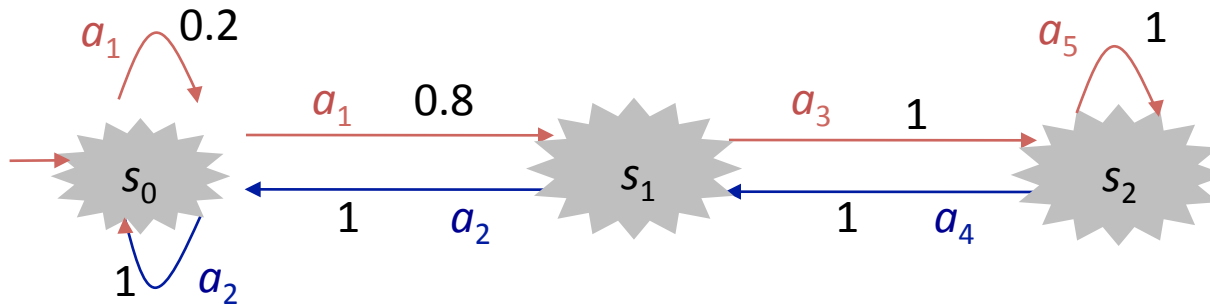
- <http://planiart.usherbrooke.ca/~eric/ift615/demos/vi/>

# Algorithme *Policy Iteration*

1. Choisir un plan arbitraire  $\pi'$
2. Répéter jusqu'à ce que  $\pi$  devienne inchangée:
  - I.  $\pi := \pi'$
  - II. pour tout  $s$  dans  $S$ , calculer  $V(\pi, s)$  en résolvant le système de  $|S|$  équations et  $|S|$  inconnues
$$V(\pi, s) = R(s) + \gamma \sum_{s' \in S} P(s' | s, \pi(s)) V(\pi, s')$$
  - III. pour tout  $s$  dans  $S$ , s'il existe une action  $a$  telle que
$$[ R(s) + \gamma \sum_{s' \in S} P(s' | s, a) V(\pi, s') ] > V(\pi, s)$$
alors  $\pi'(s) := a$  sinon  $\pi'(s) := \pi(s)$
3. Retourne  $\pi$

- Converge au plan optimal en  $O(|S|^3)$

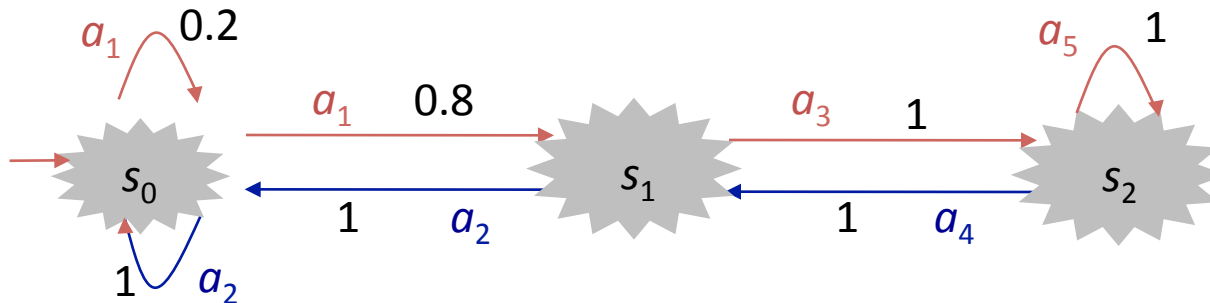
# Policy iteration: initialisation



- Plan initial choisi arbitrairement:

$$\pi' = \{ s_0 \rightarrow a_2, \\ s_1 \rightarrow a_2, \\ s_2 \rightarrow a_4 \}$$

# Policy iteration: itération #1



I.  $\pi = \pi'$

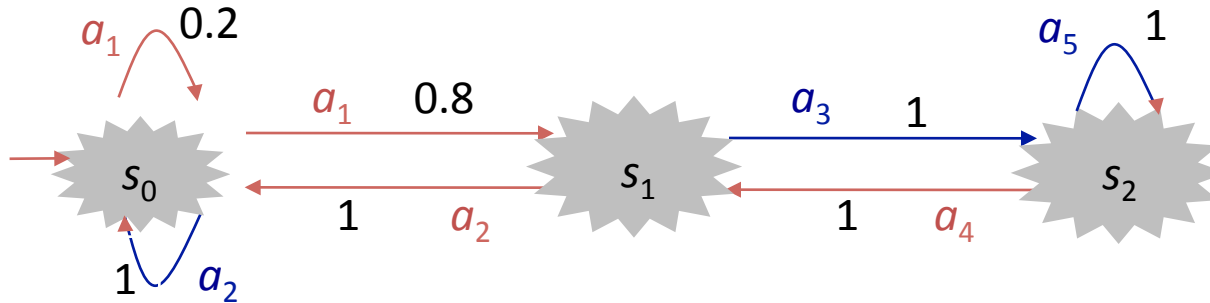
II. Équations:  $v_0 = 0 + 0.5 * (1 * v_0)$ ;  
 $v_1 = 0 + 0.5 * (1 * v_0)$ ;  
 $v_2 = 1 + 0.5 * (1 * v_1)$

Solution:  $v_0 = 0, v_1 = 0, v_2 = 1$

III.  $s_0 \rightarrow a_1: 0 + 0.5 * (0.2 * 0 + 0.8 * 0) = 0$ ;  
 $s_1 \rightarrow a_3: 0 + 0.5 * (1 * 1) = 0.5 > 0$ ;  
 $s_2 \rightarrow a_5: 1 + 0.5 * (1 * 1) = 1.5 > 1$ ;  
 $\pi' = \{ s_0 \rightarrow a_2, s_1 \rightarrow a_3, s_2 \rightarrow a_5 \}$

ne change pas  
**change**  
**change**

# Policy iteration: itération #2



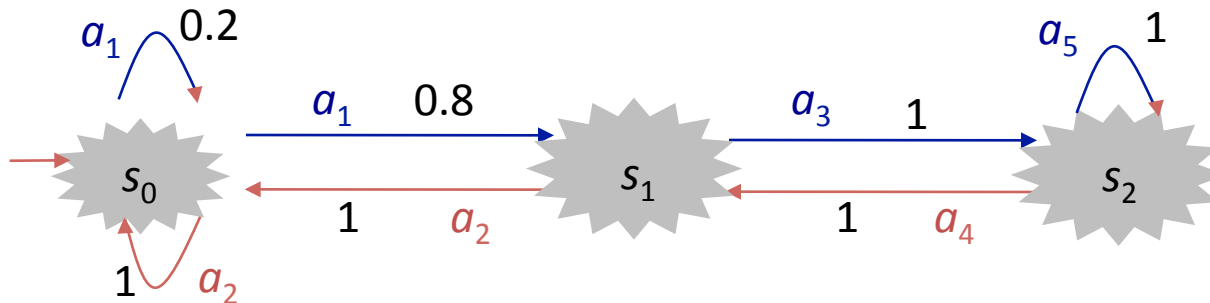
I.  $\pi = \pi'$

II. Équations:  $v_0 = 0 + 0.5 \cdot (1 \cdot v_0)$ ;  
 $v_1 = 0 + 0.5 \cdot (1 \cdot v_2)$ ;  
 $v_2 = 1 + 0.5 \cdot (1 \cdot v_2)$

Solution:  $v_0 = 0$ ,  $v_1 = 1$ ,  $v_2 = 2$

III.  $s_0 \rightarrow a_1$ :  $0 + 0.5(0.2 \cdot 0 + 0.8 \cdot 1) = 0.4 > 0$ ; **change**  
 $s_1 \rightarrow a_2$ :  $0 + 0.5(1 \cdot 0) = 0 < 1$ ; ne change pas  
 $s_2 \rightarrow a_4$ :  $1 + 0.5(1 \cdot 1) = 1.5 < 2$ ; ne change pas  
 $\pi' = \{ s_0 \rightarrow a_1, s_1 \rightarrow a_3, s_2 \rightarrow a_5 \}$

# Policy iteration: itération #3



I.  $\pi = \pi'$

II. Équations:  $v_0 = 0 + 0.5 \cdot (0.2 \cdot v_0 + 0.8 \cdot v_1)$ ;  
 $v_1 = 0 + 0.5 \cdot (1 \cdot v_2)$ ;  
 $v_2 = 1 + 0.5 \cdot (1 \cdot v_2)$

Solution:  $v_0 = 4/9$ ,  $v_1 = 1$ ,  $v_2 = 2$

III.  $s_0 \rightarrow a_2$ :  $0 + 0.5(1 \cdot 0.4) = 0.2 < 4/9$ ; ne change pas  
 $s_1 \rightarrow a_2$ :  $0 + 0.5(1 \cdot 0.4) = 0.2 < 1$ ; ne change pas  
 $s_2 \rightarrow a_4$ :  $1 + 0.5(1 \cdot 1) = 1.5 < 2$ ; ne change pas  
 $\pi' = \{ s_0 \rightarrow a_1, s_1 \rightarrow a_3, s_2 \rightarrow a_5 \}$ , c-à-d.  $\pi$

• Solution finale:  $\pi$



# Rappel: systèmes d'équations linéaires

- Soit le système d'équations:  
$$v_0 = 0 + 0.5 * (0.2*v_0 + 0.8*v_1);$$
$$v_1 = 0 + 0.5 * (1*v_2);$$
$$v_2 = 1 + 0.5 * (1*v_2)$$
- En mettant toutes les variables à droite, on peut l'écrire sous la forme:  
$$0 = -0.9 v_0 + 0.4 v_1 \quad (1)$$
$$0 = -v_1 + 0.5 v_2 \quad (2)$$
$$-1 = -0.5 v_2 \quad (3)$$
- De l'équation (3), on conclut que  $v_2 = -1 / -0.5 = 2$
- De l'équation (2), on conclut que  $v_1 = 0.5 v_2 = 1$
- De l'équation (1), on conclut que  $v_0 = 0.4 v_1 / 0.9 = 4/9$

# Rappel: systèmes d'équations linéaires

- Soit le système d'équations:

$$v_0 = 0 + 0.5 * (0.2*v_0 + 0.8*v_1);$$

$$v_1 = 0 + 0.5 * (1*v_2);$$

$$v_2 = 1 + 0.5 * (1*v_2)$$

- En mettant toutes les variables à droite, on peut l'écrire sous la forme:

$$0 = -0.9 v_0 + 0.4 v_1 \quad (1)$$

$$0 = -v_1 + 0.5 v_2 \quad (2)$$

$$-1 = -0.5 v_2 \quad (3)$$

- Approche alternative: on écrit sous forme matricielle  $b = A v$ , où

$$A = \begin{pmatrix} -0.9 & 0.4 & 0 \\ 0 & -1 & 0.5 \\ 0 & 0 & -0.5 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \quad v = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix}$$

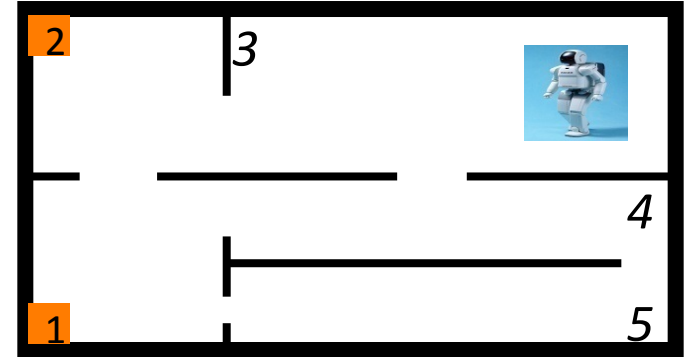
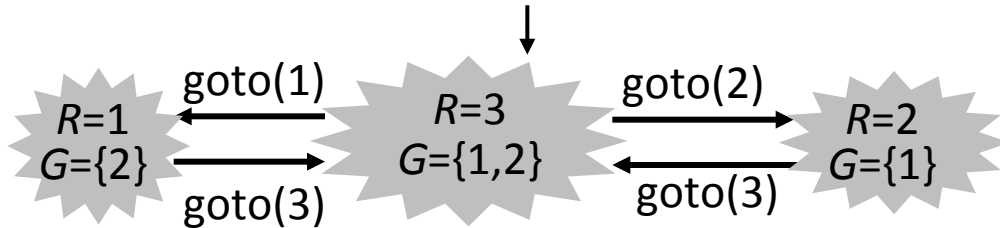
# Rappel: systèmes d'équations linéaires

- Soit le système d'équations:  
$$v_0 = 0 + 0.5 * (0.2*v_0 + 0.8*v_1);$$
$$v_1 = 0 + 0.5 * (1*v_2);$$
$$v_2 = 1 + 0.5 * (1*v_2)$$
- En mettant toutes les variables à droite, on peut l'écrire sous la forme:  
$$0 = -0.9 v_0 + 0.4 v_1 \quad (1)$$
$$0 = -v_1 + 0.5 v_2 \quad (2)$$
$$-1 = -0.5 v_2 \quad (3)$$
- Suffit alors d'inverser A pour obtenir  $v = A^{-1} b$

- ◆ on peut utiliser une librairie d'algèbre linéaire (ex.: Numpy en Python):

```
>>> A = numpy.array([[ -0.9, 0.4, 0], [0, -1, 0.5], [0, 0, -0.5]])
>>> b = numpy.array([0, 0, -1])
>>> Ainv = numpy.linalg.inv(A)
>>> v = numpy.dot(Ainv, b)
>>> print v
[ 0.44444444  1.          2.          ]
```

# Fonctions de récompenses complexes



- Notons:
  - ◆  $R = i$  le fait que le robot est dans le local numéro  $i$
  - ◆  $G=\{i,...,k\}$  le but spécifiant que le robot doit visiter les locaux  $\{1, ..., k\}$
- Ainsi  $G=\{1,2\}$  signifie que le robot doit visiter le local 1 (c-à-d.,  $R=1$ ) **et** visiter le local 2 (c-à-d.,  $R=2$ )
- Ce genre de but nécessite d'étendre au préalable l'espace d'états de manière à attribuer des récompenses à des comportements (pas seulement à un état)
- Une façon élégante de le faire est d'attribuer les récompenses à des formules de logique temporelle satisfaisant les comportements désirés [Thiébaux et al., JAIR 2006]

# Un peu plus loin ...

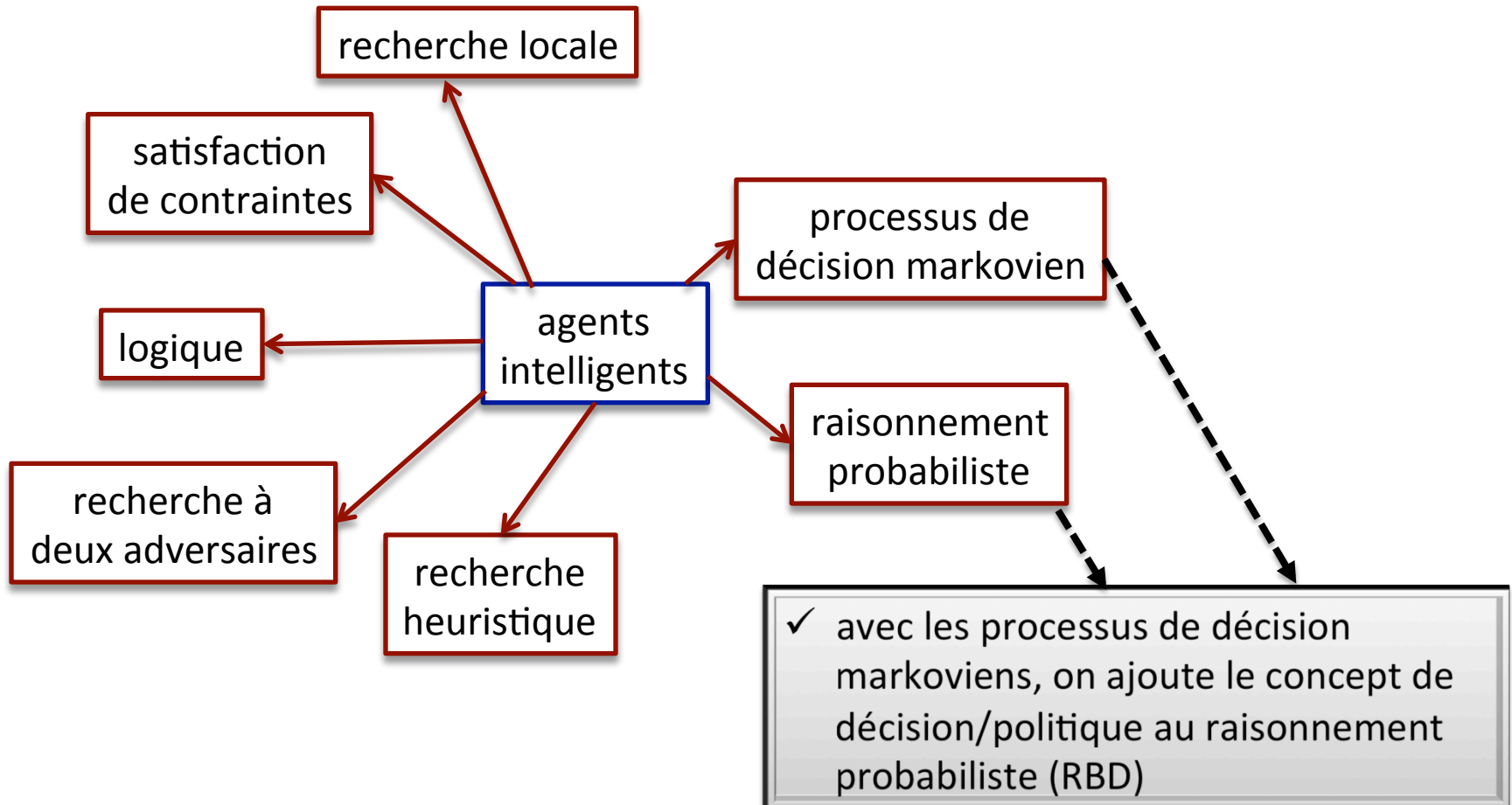
- Les algorithmes *value-iteration* et *policy-iteration* sont lents sur des grands espaces d'état
  - ◆ Améliorations:
    - » *Real-Time Dynamic Programming* (RTPD)
    - » *Labeled RTDP*
- Les MDP supposent une observation complète
  - ◆ *Partially Observable MDP* (PoMDP)
- Les MDP sont limités à des décisions séquentielles
  - ◆ pour des actions simultanées:
    - » *Concurrent MDP* (CoMPD)
    - » *Concurrent Probabilistic Temporal Planning* (CPTP)

# Résumé

- L'approche Markovienne est très attrayante parce qu'elle combine raisonnement probabiliste et optimisation avec élégance
- C'est une des approches les plus étudiées actuellement pour:
  - ◆ la planification (cours IFT 702)
  - ◆ l'apprentissage par renforcement (qu'on verra bientôt)
- Elle est notamment populaire dans les applications de robots mobiles

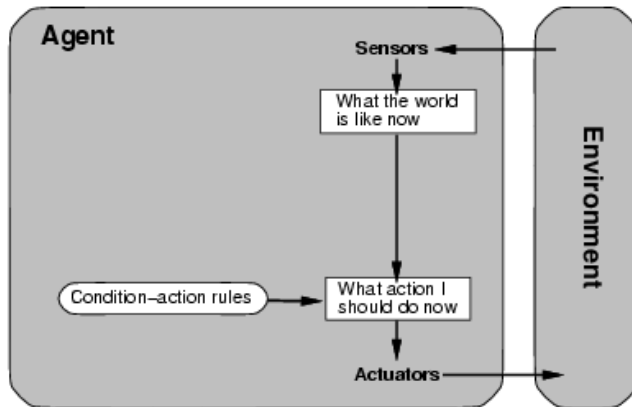
# Objectifs du cours

## Algorithmes et concepts

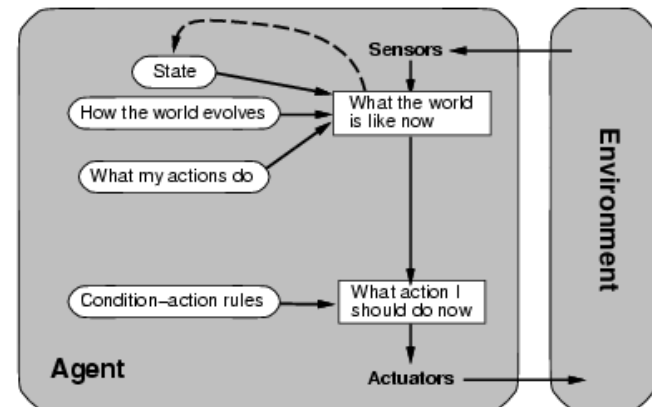


# Recherche pour jeux à deux adversaires : pour quel type d'agent?

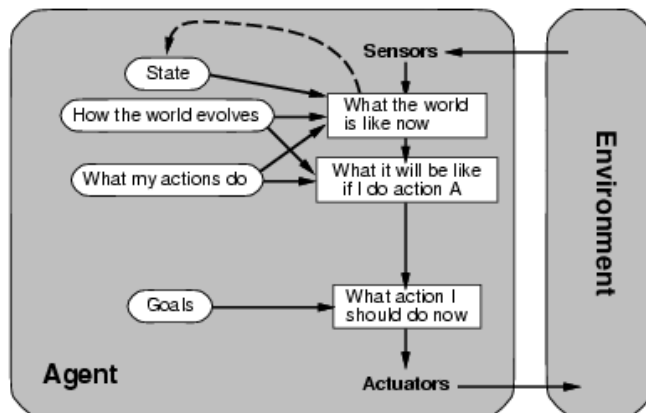
## Simple reflex



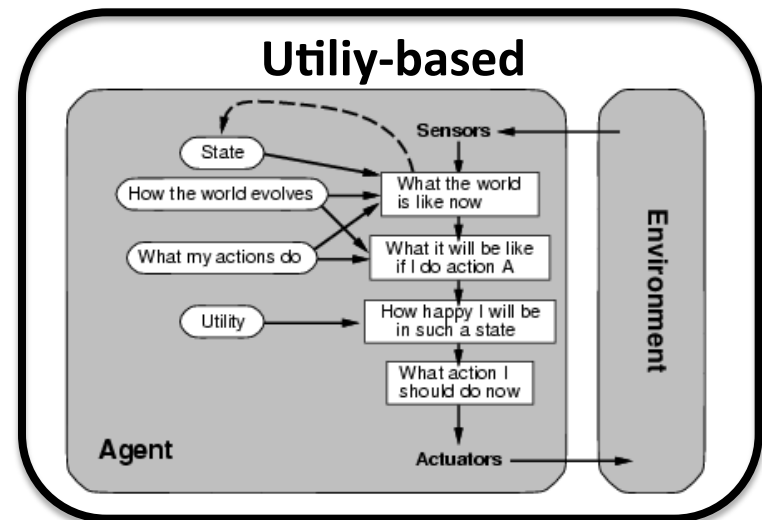
## Model-based reflex



## Goal-based



## Utility-based





# Vous devriez être capable de...

- Donner la définition d'un processus de décision markovien
  - ◆ espace d'état
  - ◆ action
  - ◆ modèle de transition
  - ◆ fonction de récompense
  - ◆ décision
  - ◆ plan/politique
- Simuler *value iteration*
- Expliquer le fonctionnement de *policy iteration*