

IFT 615 – Intelligence Artificielle

Processus de décision markoviens

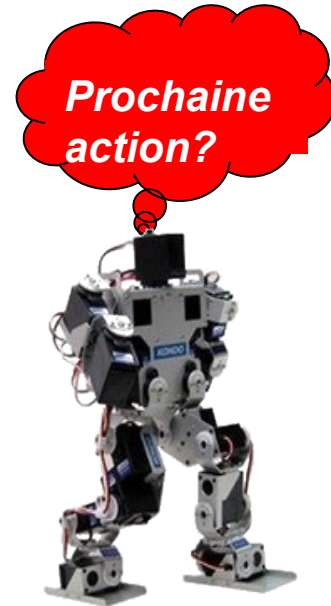
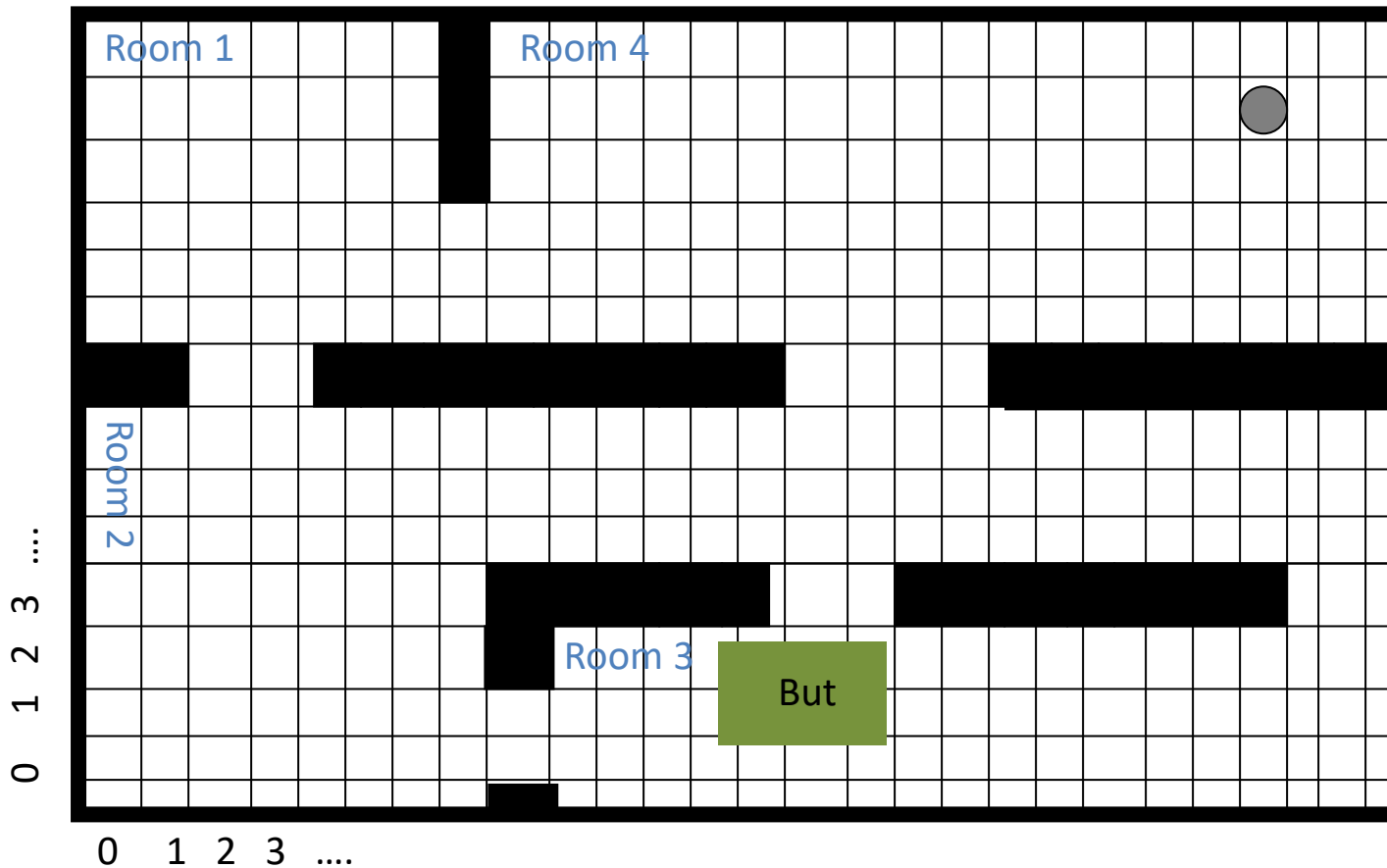
Professeur: Froduald Kabanza


Assistants: D'Jeff Nkashama

Sujets couverts

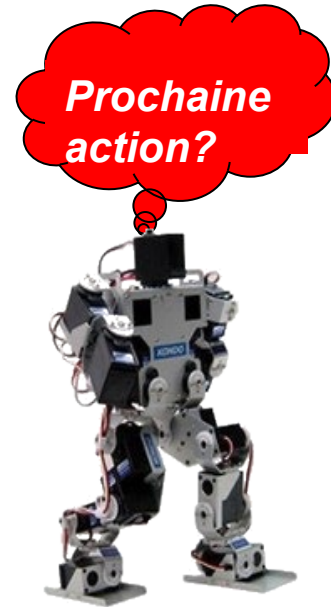
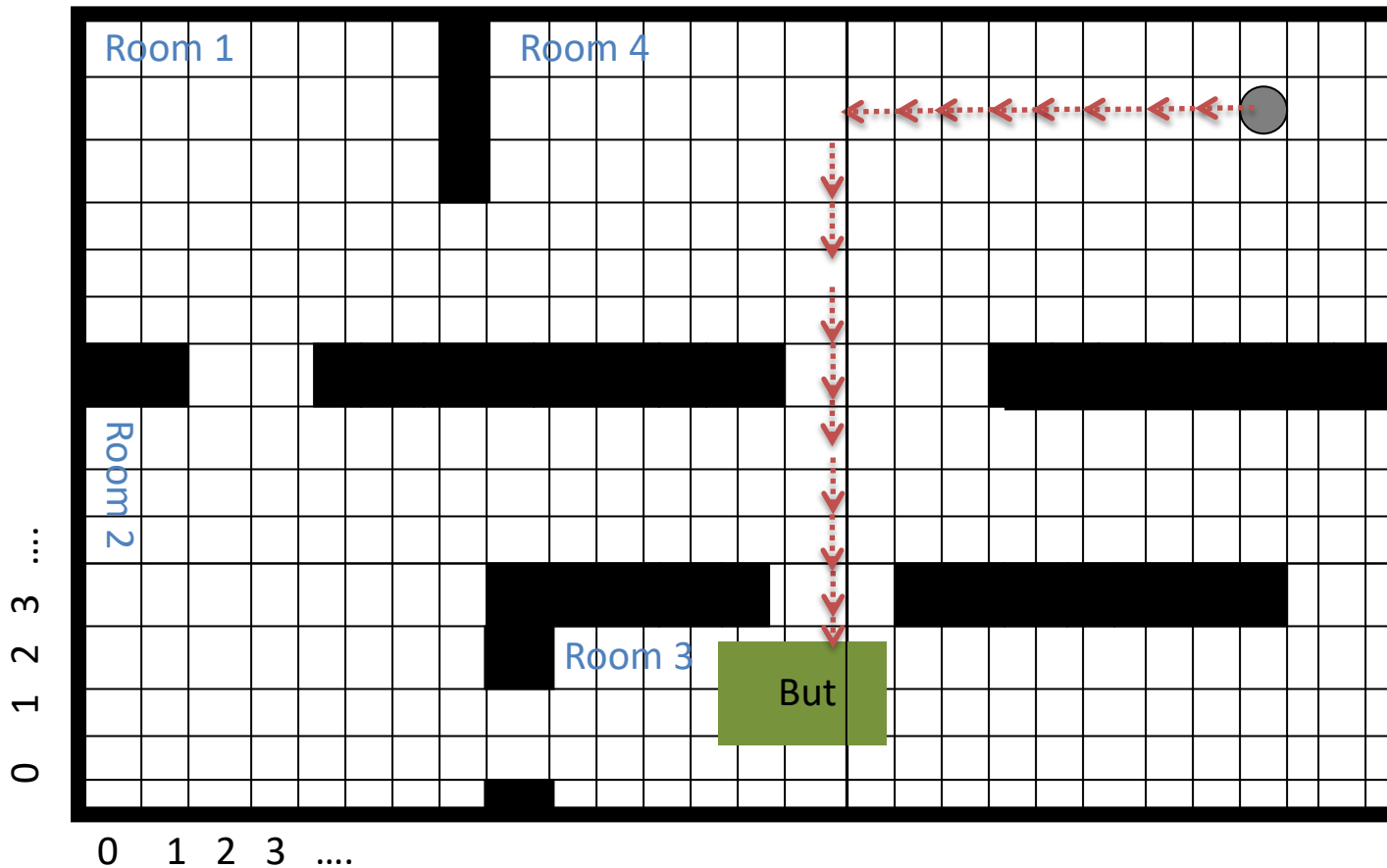
- *MDP*
- *Reward*
- *Policy*
- *Value function*
- Équations de Bellman
- *Q function* (state-action pairs)
- Programmation dynamique
 - ◆ *Value iteration*
 - ◆ *Policy Iteration*
- *Asynchronous Policy Iteration*


Motivation – Planification avec des actions ayant des effets incertains



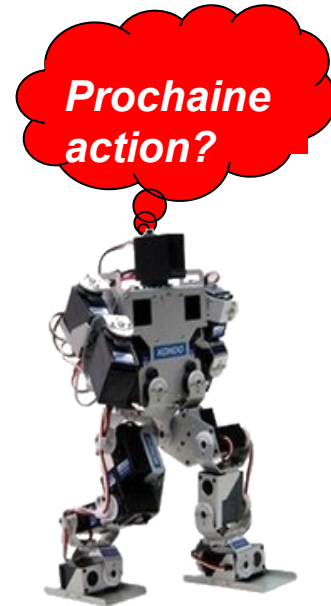
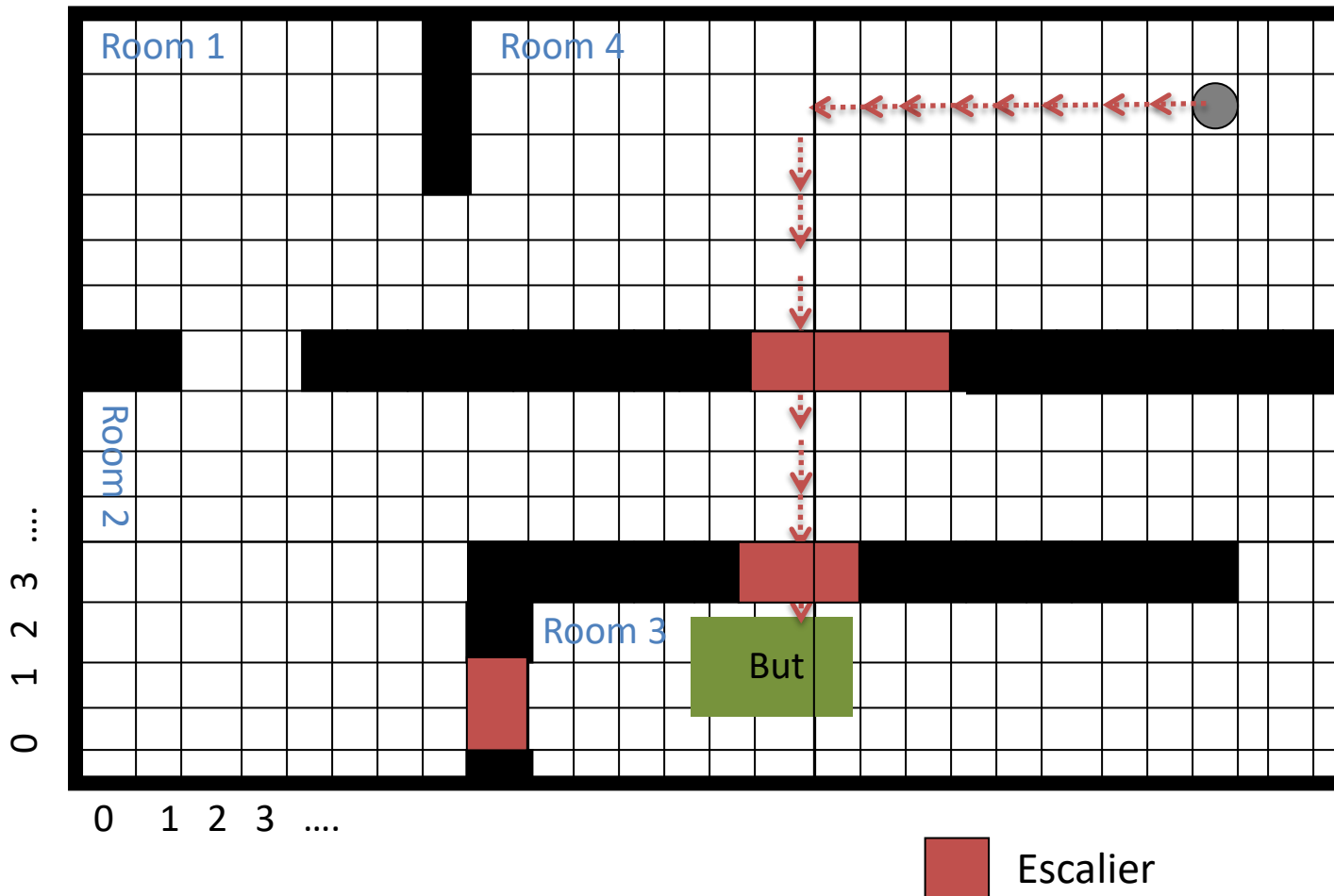
Actions: 
L: Go left
R: Go right
D: Go down
U: Go up


Motivation – Planification avec des actions ayant des effets incertains



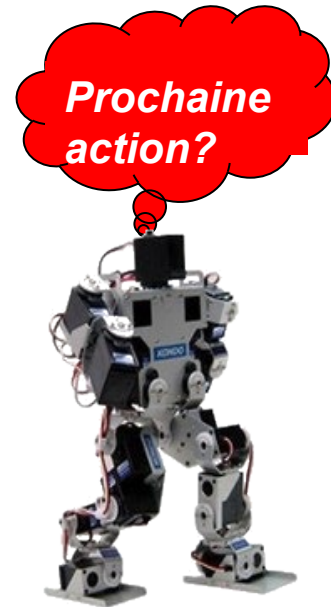
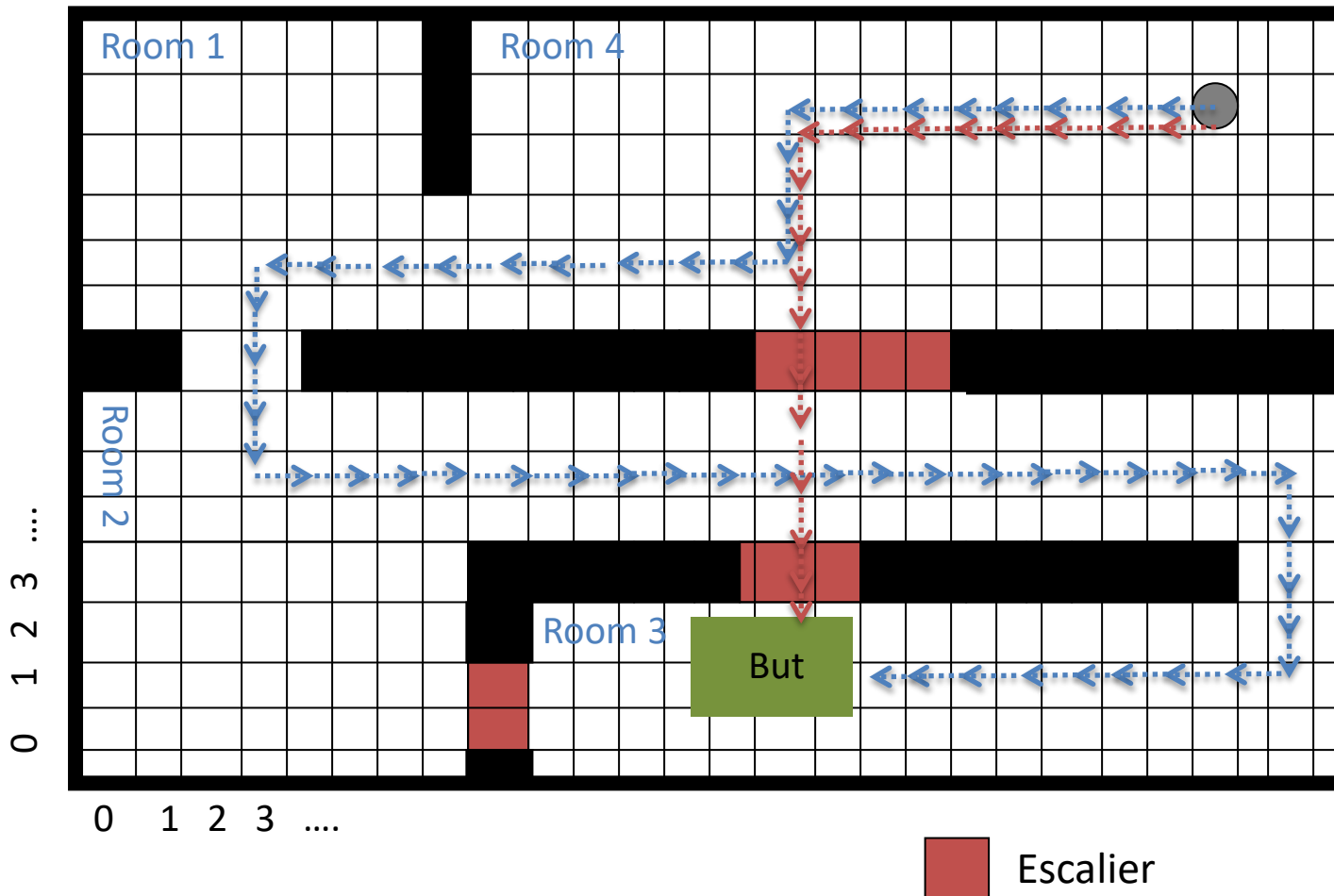
Actions: 
L: Go left
R: Go right
D: Go down
U: Go up

Motivation – Planification avec des actions ayant des effets incertains



Actions: 
L: Go left
R: Go right
D: Go down
U: Go up

Motivation – Planification avec des actions ayant des effets incertains



Actions:

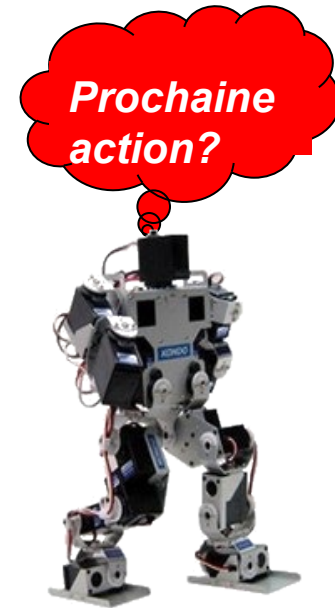
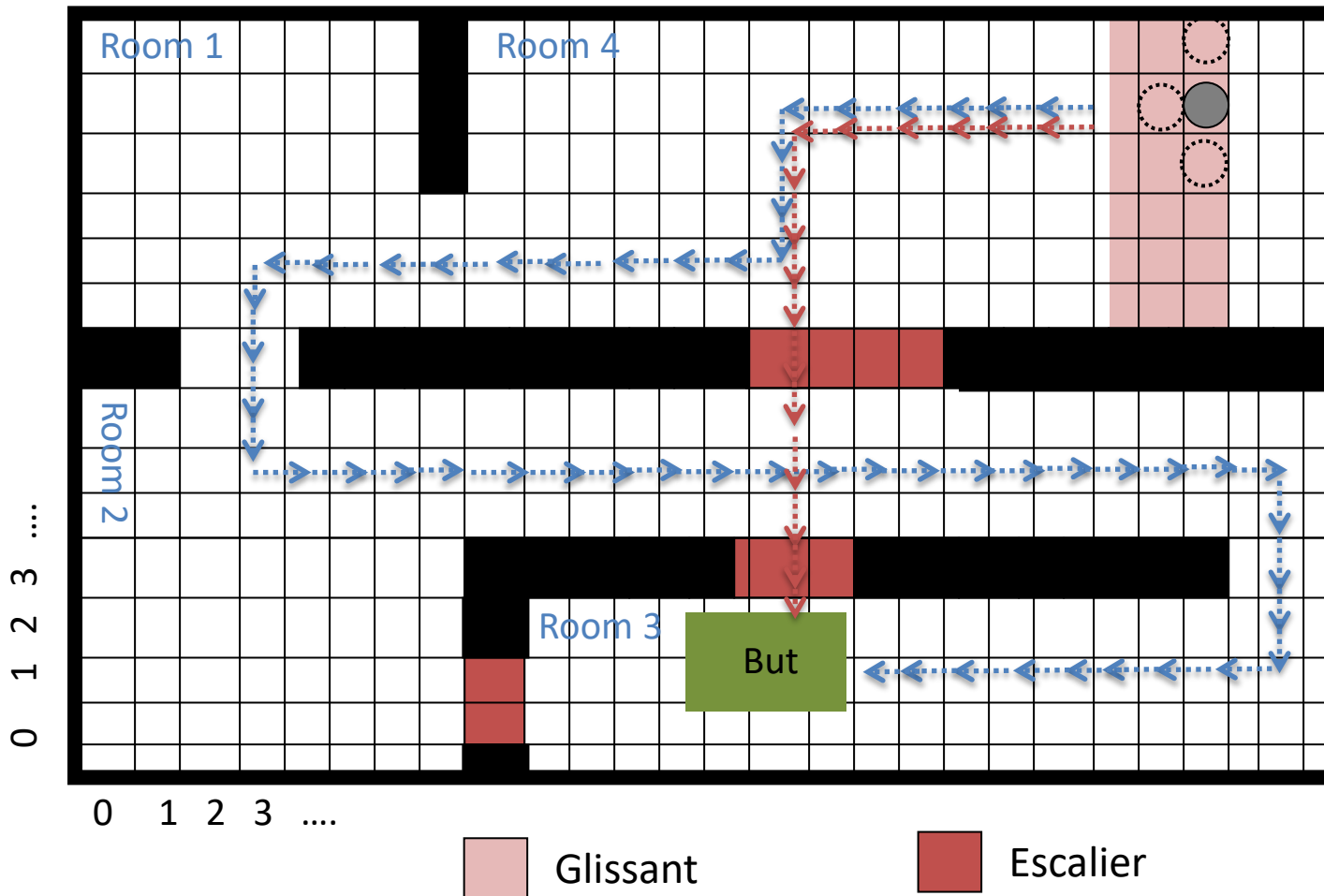
L: Go left


R: Go right

D: Go down

U: Go up

Motivation – Planification avec des actions ayant des effets incertains



Actions: 

L: Go left

R: Go right

D: Go down

U: Go up

Type d'environnement

Statique vs. Dynamique

Prévisible vs. Imprévisible

Environment

Actions

Discrètes

vs.
Continues

Déterministes
vs.

Stochastiques

Observabilité

Complète

vs.

Partielle

Vs.

Aucune

Capteurs

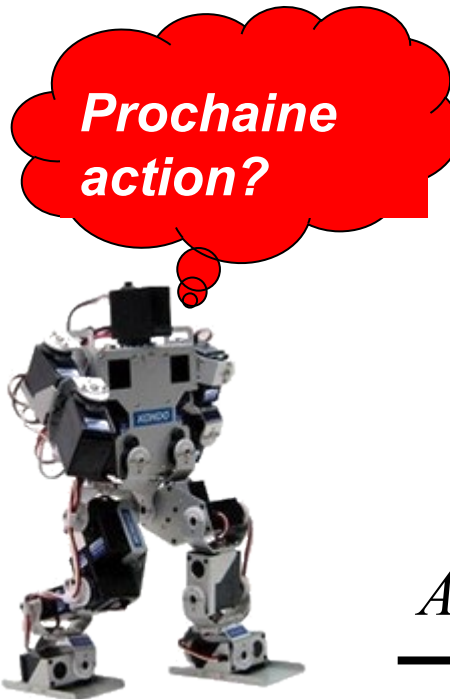
Parfaits

vs.

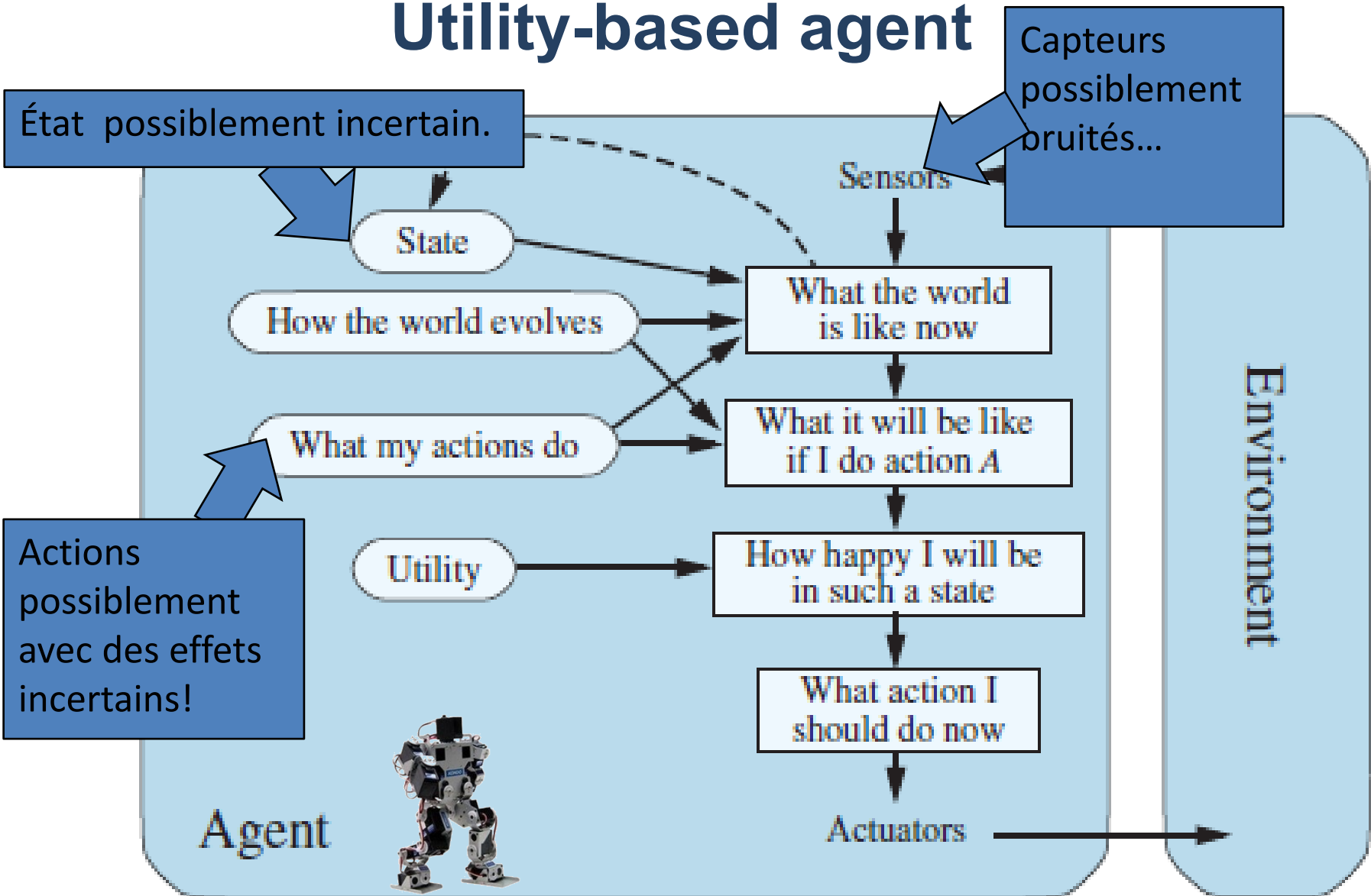
Bruités

Percepts

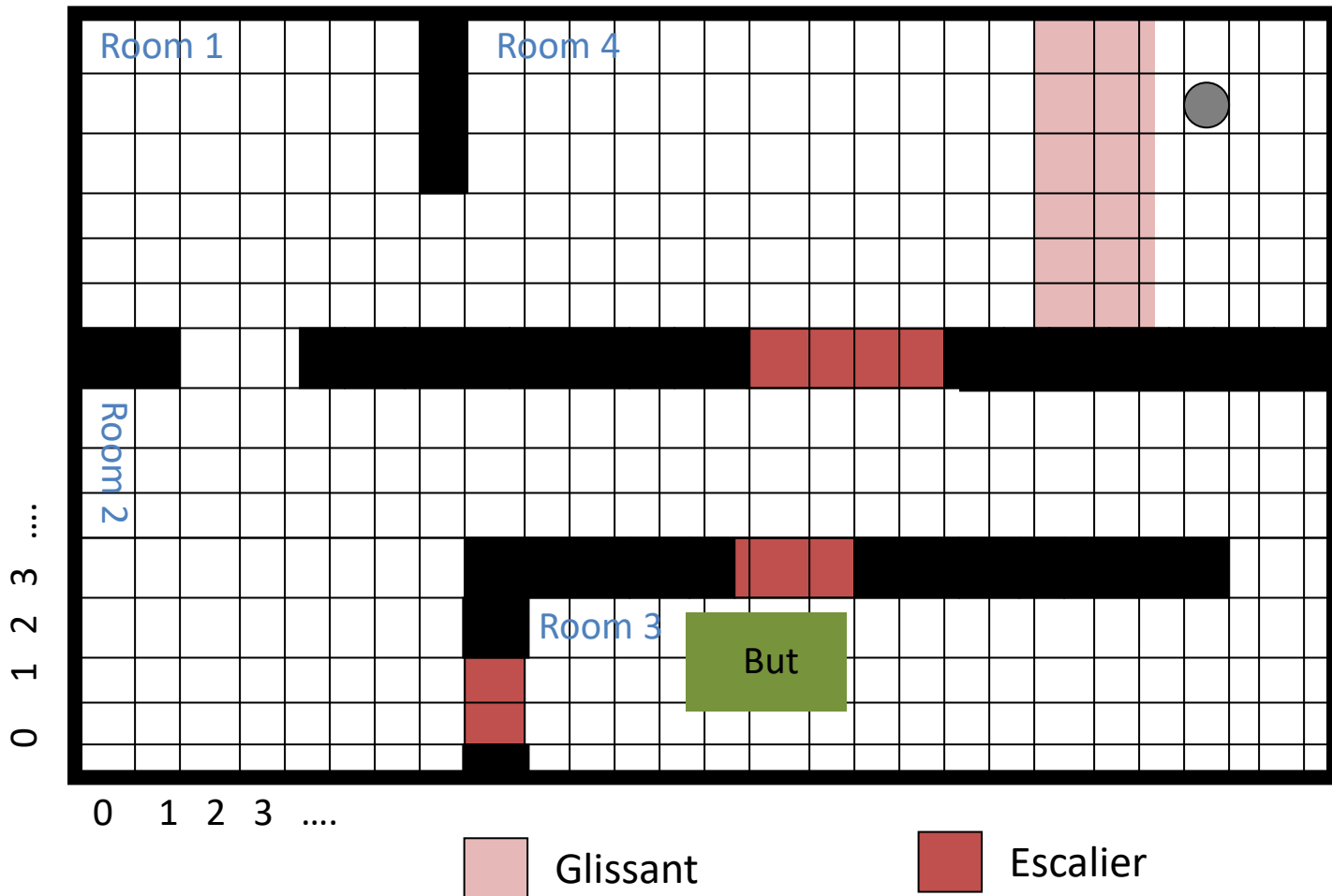
Actions




Utility-based agent



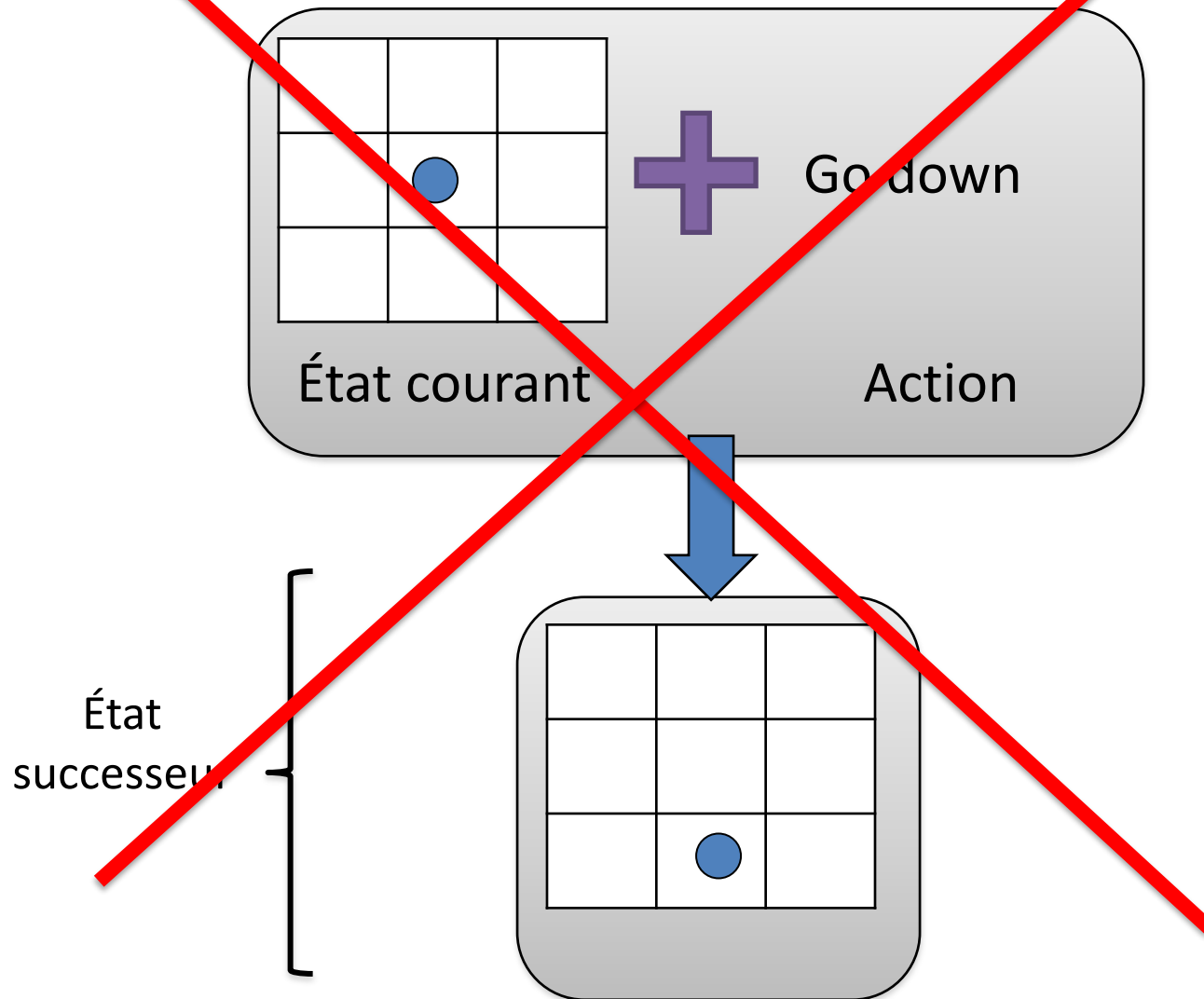
Modèle d'actions



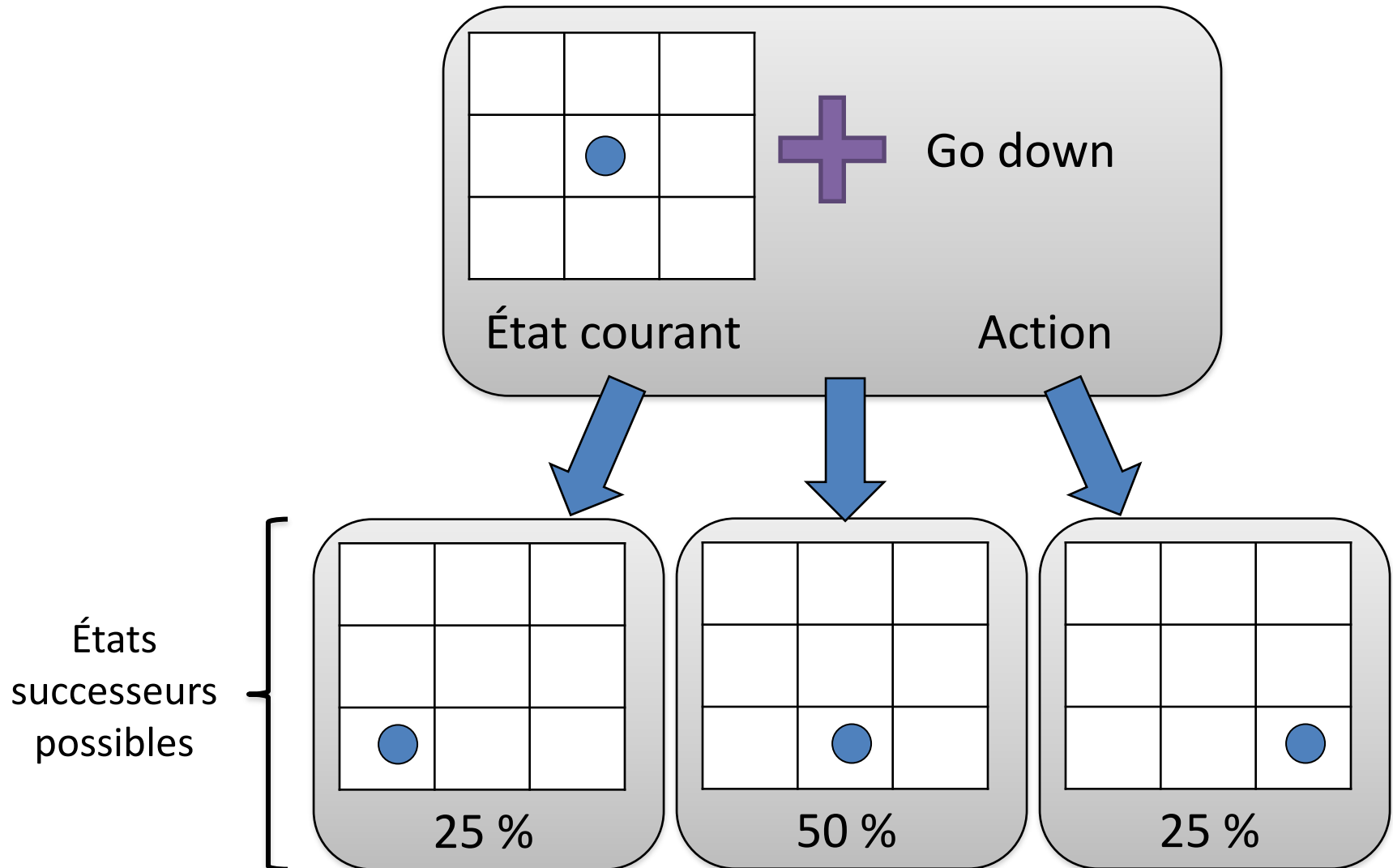
Actions: 

- L: Go left
- R: Go right
- D: Go down
- U: Go up

~~A* : actions déterministes~~



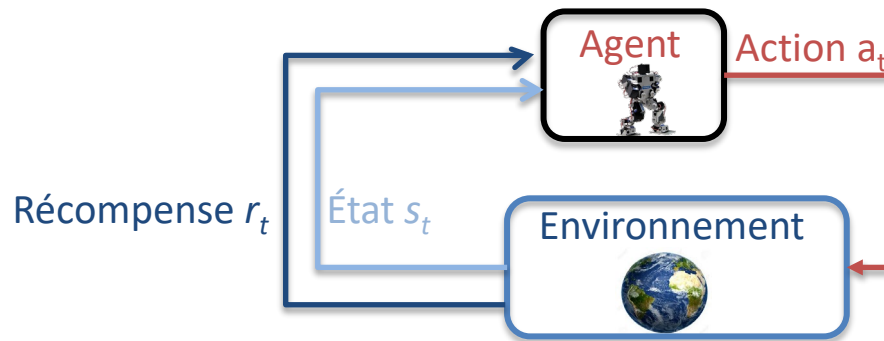
MDP : actions non déterministes



Processus de décision markovien

Idée de base

- La planification par les **processus de décision Markoviens** s'intéresse au cas où un agent doit **décider comment agir** en tenant compte d'une fonction d'utilité **exprimée sous forme des récompenses ou renforcements**



- Exécution (comportement) de l'agent: $s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$
 - ◆ L'agent **agit** sur son environnement
 - ◆ Reçoit une retro-action sous-forme de **récompense (renforcement)**
 - ◆ Son **but** est de maximiser la somme des récompenses espérés

- Problème: **Calculer la politique (plan) qui** maximiser la somme des récompenses

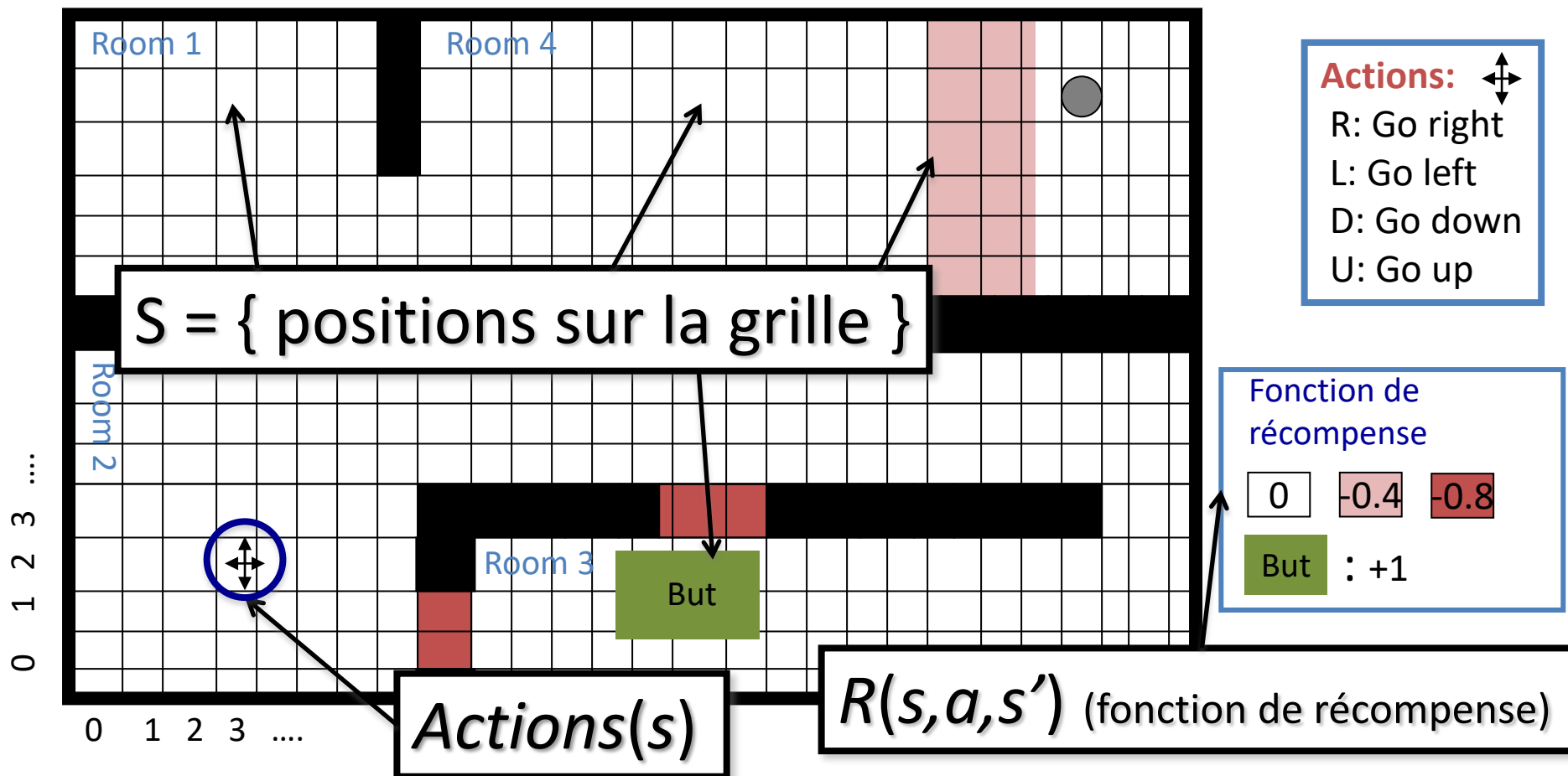
$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots, \quad \text{avec } 0 \leq \gamma \leq 1 \quad \text{et } r_i < R_{max}$$

Processus de décision markovien

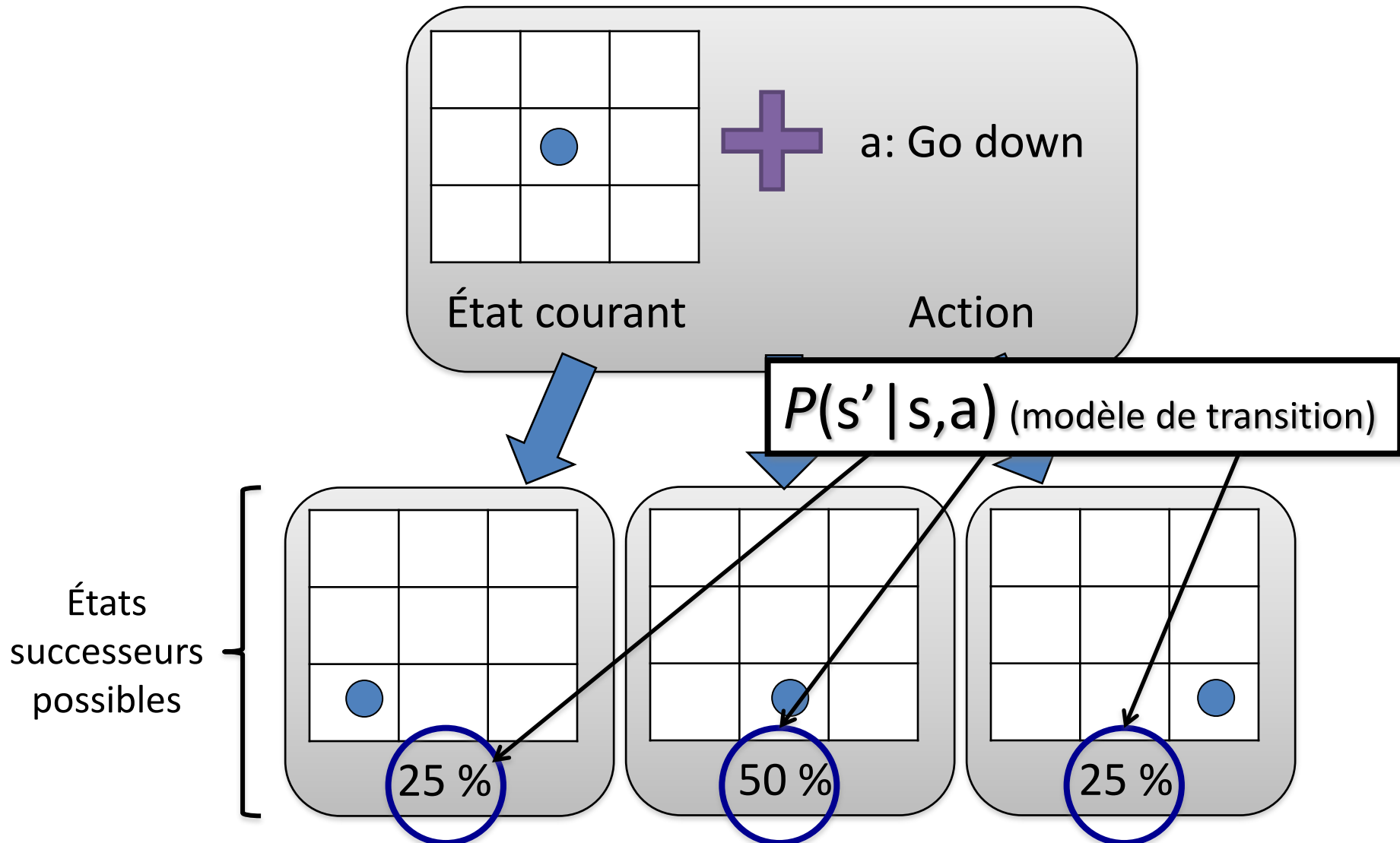
Définition Formelle

- 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 $A(s)$ 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
- Requièrè qu'on décrive un objectif à atteindre à partir d'une fonction de récompense basée seulement sur l'état courant

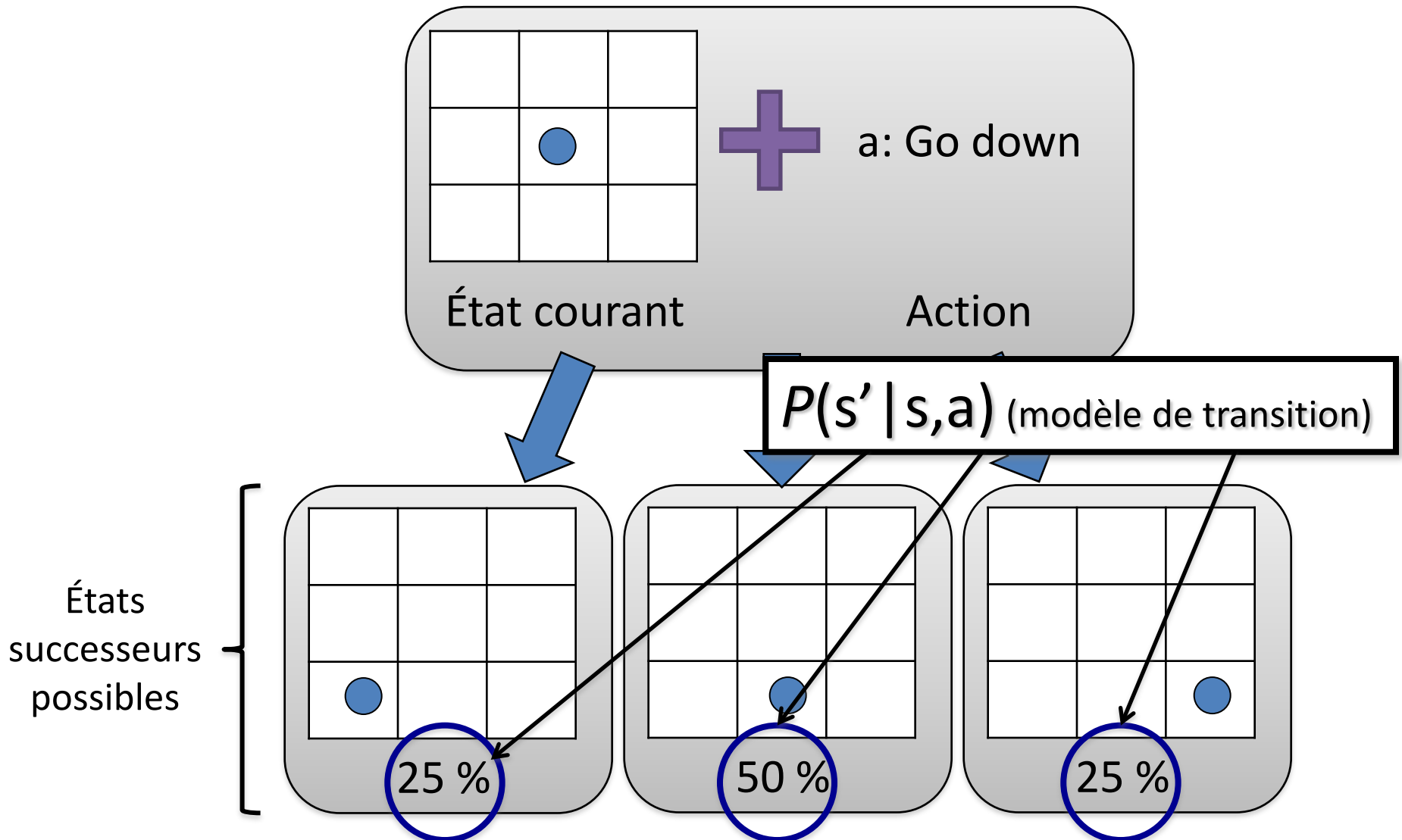
Modèle d'actions



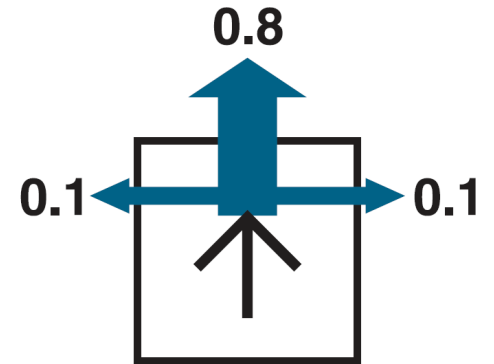
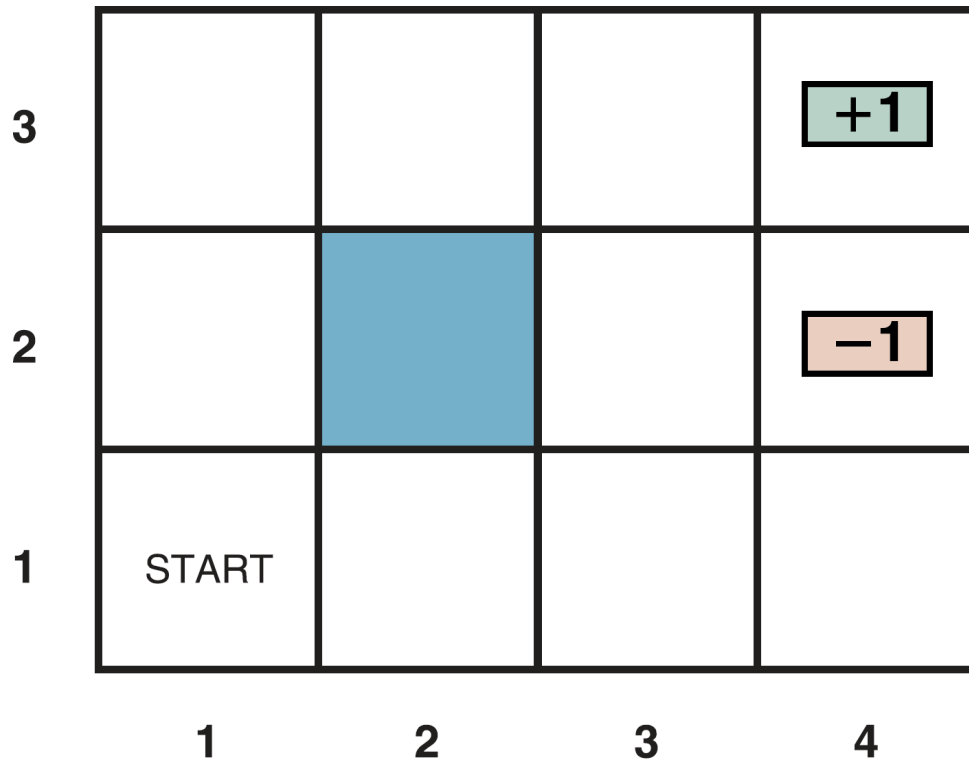
Actions aux effets incertains



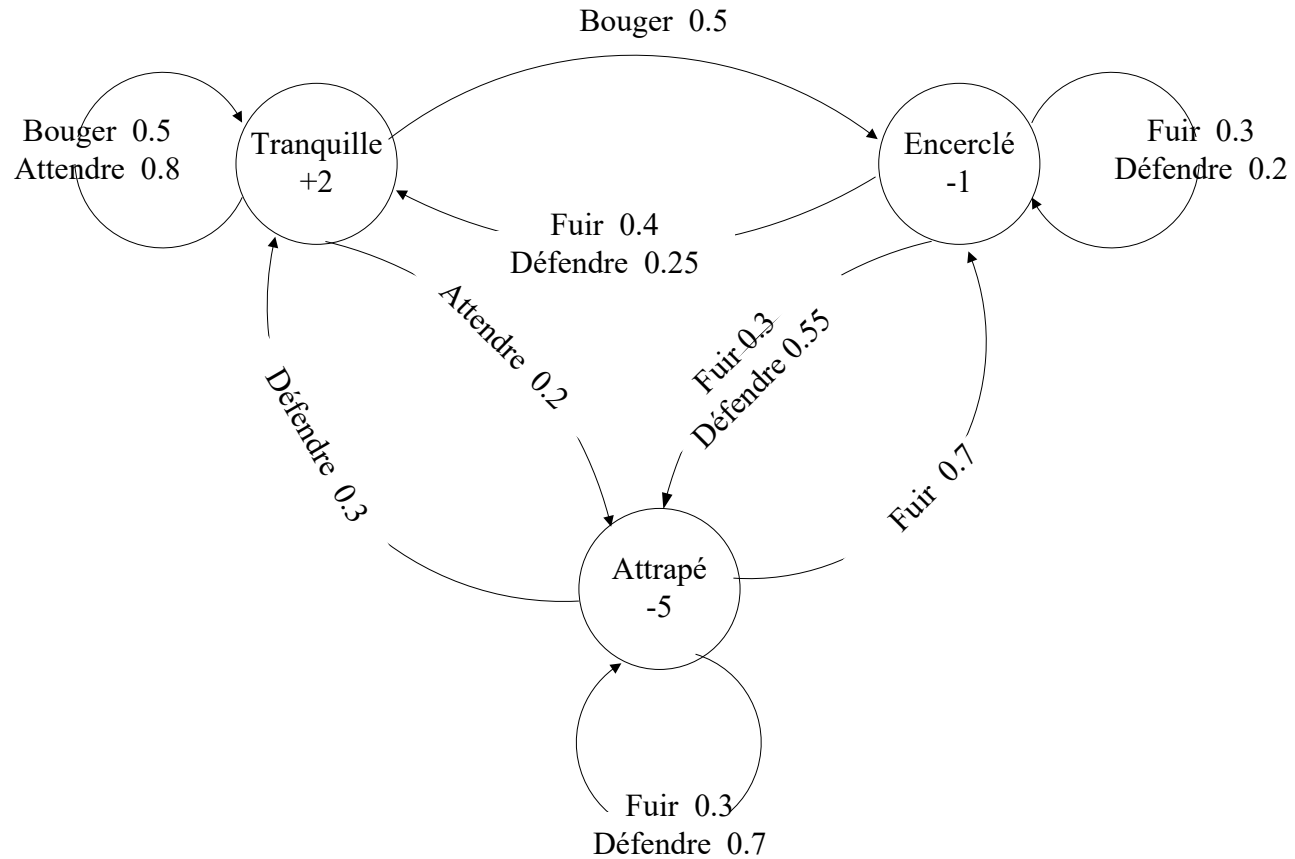
Actions aux effets incertains



Exemple



Exemple



Décision

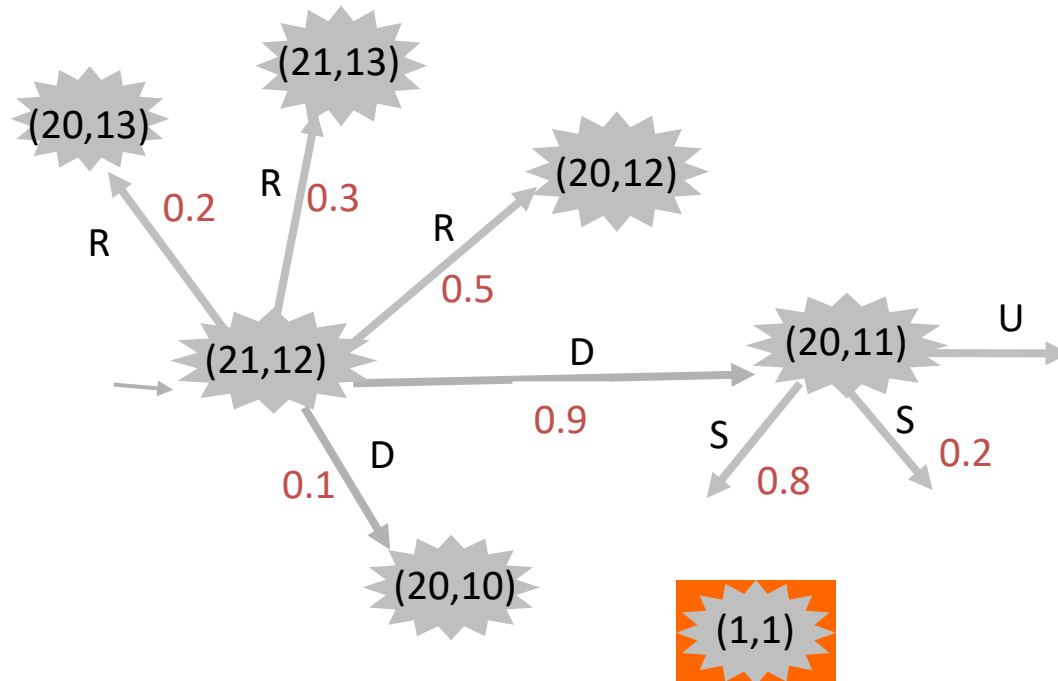
- Une **décision** est un choix d'une action dans un état
 - ◆ c'est une règle « *if state then action* »

Exemples:

$(21,12) \rightarrow R$

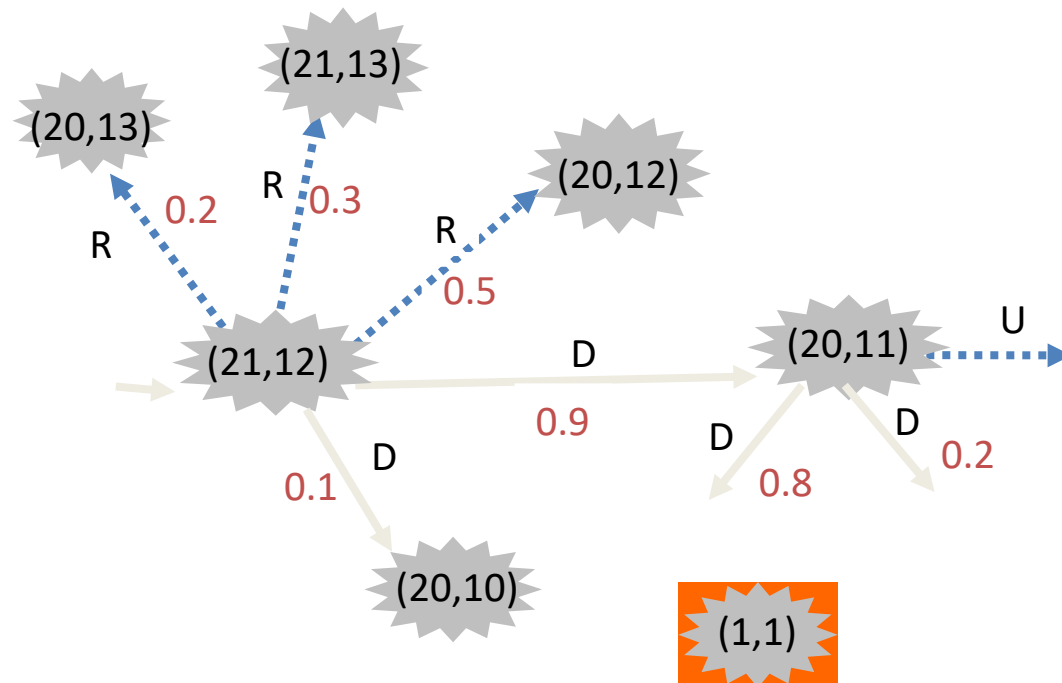
ou

$(19,12) \rightarrow L$



Politique (plan)

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



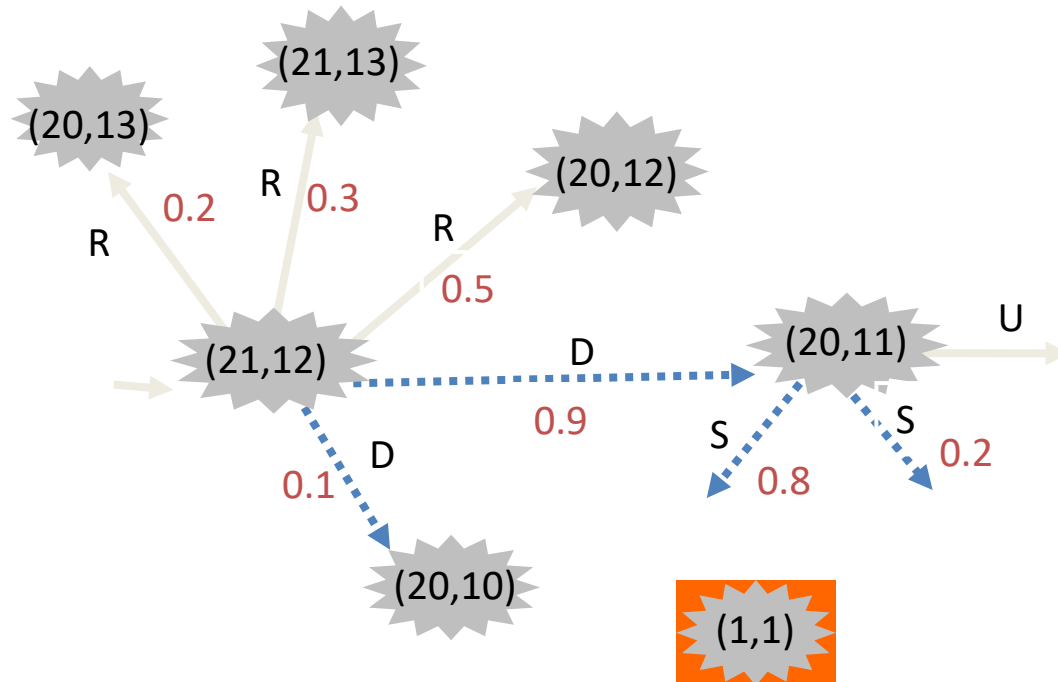
Exemples:

Plan π_1

$\{$ $(21,12) \rightarrow R,$
 $(20,13) \rightarrow U,$
 $(21,11) \rightarrow D,$
 $(19,12) \rightarrow L,$
 \dots
 $\}$

Politique

- Un **politique** est un choix d'une action **pour chaque état**



Exemples:

Plan π_1

$\{ (21,12) \rightarrow R,$
 $(20,13) \rightarrow U,$
 $(21,11) \rightarrow D,$
 $(19,12) \rightarrow L,$
 \dots
 $\}$

Plan π_2

$\{ (21,12) \rightarrow R,$
 $(20,11) \rightarrow D,$
 $(19,12) \rightarrow L,$
 $\dots \}$

Exécution d'une politique

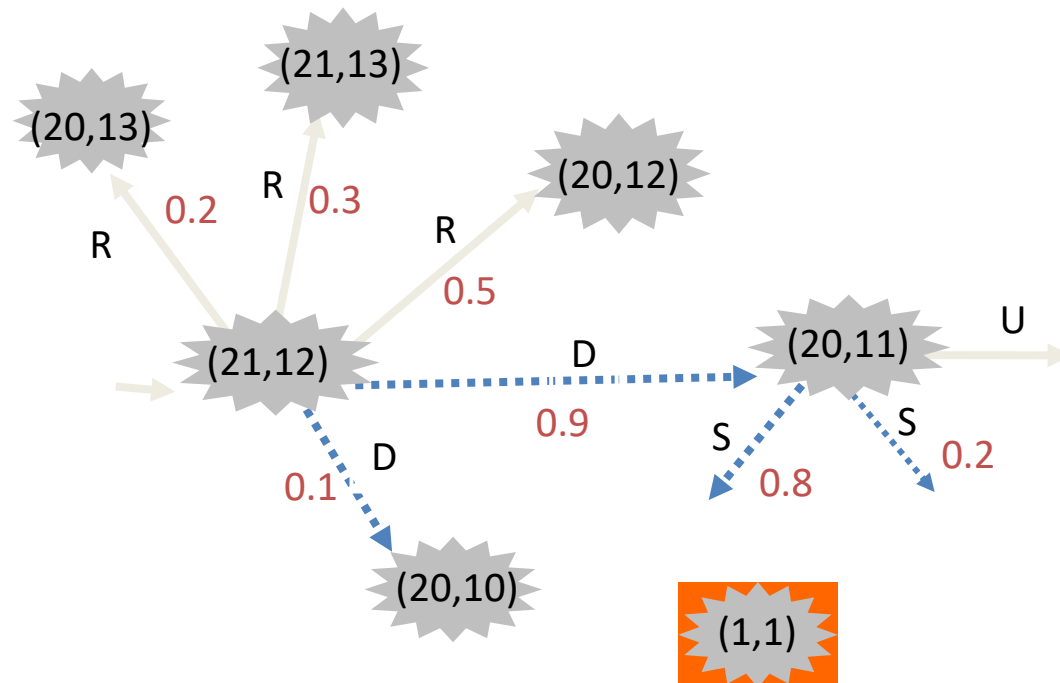
- Une politique est un choix d'action pour chaque état
- Notons $\pi(s)$ l'action choisie pour l'état s
- Voici un algorithme d'exécution ou d'application de la politique π

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

- L'étape 1 peut impliquer du filtrage pour reconnaître l'état courant s
- 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, dont le modèle de transition est donné par $P(s'|s, \pi(s))$
- La chaîne se déroule en un arbre potentiellement infini



Exemples:

Plan π_1

$\{ (21,12) \rightarrow R,$
 $(20,13) \rightarrow U,$
 $(21,11) \rightarrow D,$
 $(19,12) \rightarrow L,$
 \dots
 $\}$

Plan π_2

$\{ (21,12) \rightarrow R,$
 $(20,11) \rightarrow D,$
 $(19,12) \rightarrow L,$
 $\dots \}$

Horizon fini vs Horizons infini

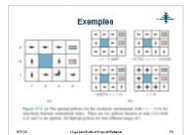
- Horizon fini:

- ◆ L'exécution termine après un nombre fini d'étapes.
- ◆ On peut utiliser $\gamma=1$, la somme des récompenses demeurera finie.
- ◆ Pour un horizon fini, le temps a de l'importance. La **politique est non stationnaire**.

- Horizon infini:

- ◆ L'exécution ne termine pas (des boucles)
- ◆ Il faut ut $U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1 - \gamma)$ mpenses soient finie

- ◆ La **politique est stationnaire**. Le temps n'a pas d'importance.



Interprétation/application d'un plan

- La qualité d'un plan est déterminée par l'ensemble des séquences d'états qui peuvent potentiellement en résulter.
- C-à-d., les séquences déroulables de la chaîne de Markov correspondante.
- La qualité peut être formalisée selon:
 - ◆ Une approche logique classique: chaque séquence doit satisfaire une condition de succès (conduire au but ou satisfaire une formule de logique temporelle)
 - ◆ Une approche de théorie de la décision: fonction d'utilité ou de récompense.
 - ◆ Une combinaison des deux.
- Chaque approche donne lieu à un algorithme de planification différent:
 - ◆ Recherche dans un graphe et/ou pour l'approche logique classique (And-OR A* - Voir le manuel du cours).
 - ◆ Programmation dynamique pour l'approche de théorie de la décision (ce qu'on voit dans cette leçon).
 - ◆ Il existe d'autres approches

Fonction de récompense/utilité et qualité des plans

- Une fonction de récompense/utilité, $R(s)$, assigne un nombre réel à chaque état s .
 - ◆ $R(s)$ désigne le degré de désirabilité de l'état s .
- Le but et le coût des actions sont indirectement modélisés par la fonction de récompense/utilité.
- Ainsi, la qualité d'un plan est déterminée par l'espérance des récompenses qu'on peut potentiellement obtenir en suivant/exécutant le plan
 - ◆ Un plan optimal est celui qui maximise les récompenses.
 - ◆ Plus un plan est proche de l'optimal optimal, plus il est de qualité.
- Ainsi un plan fait un compromis entre:
 - ◆ La maximisation de la probabilité d'atteindre le but (réduction de la probabilité d'échec).
 - ◆ La maximisation des récompenses (optimisation du coût des actions).

Utilité d'une politique

- L'**utilité** d'une politique (plan) π exécutée à partir à l'état s est donnée par

$$\begin{aligned} U^\pi(s) &= E \left[\sum_{t=0}^{\infty} \gamma^t R(st, \pi(s), st_{+1}) \right] \\ &= \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')] \end{aligned}$$

c.-à-d., la somme des récompenses futures espérées pondérées par les probabilités de transition

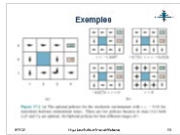
- ◆ γ : facteur d'escompte ($0 \leq \gamma \leq 1$)
- ◆ $R(s, a, s')$: **récompense** pour la transition (s, a, s')
- ◆ S : espace d'états
- ◆ $\pi(s)$: action du plan à l'état s
- ◆ $P(s'|s, \pi(s))$: probabilité de la transition du MDP

Rôle du facteur d'escompte

- Le facteur d'escompte permet de pondérer les récompenses selon l'importance d' « agir bien » dans un horizon proche ou un horizon lointain.
 - ◆ Plus γ est petit, plus l'horizon est proche (on se concentre sur les récompenses dans un horizon proche).
 - ◆ Autrement dit, de façon général, le facteur d'escompte est vu comme un taux d'inflation.
- Pour les problèmes avec un horizon infini, le facteur d'escompte assure la convergence de la somme infinie des récompenses.
 - ◆ Comme on vient de voir, il faut utiliser $0 \leq \gamma < 1$ pour que la somme des récompenses soient finie

Politique optimale

- Un politique π **domine** un politique π' si les deux conditions suivantes sont réunies:
 - ◆ $U^\pi(s) \geq U^{\pi'}(s)$ pour tout état s
 - ◆ $U^\pi(s) > U^{\pi'}(s)$ pour au moins un s
- Un politique est **optimale** si elle n'est pas dominée par une autre
 - ◆ il peut y avoir plusieurs politiques optimales, mais elles ont tous la même valeur
 - ◆ on peut avoir deux politiques **incomparables** (aucun ne domine l'autre)
 - » la dominance induit une fonction d'ordre partiel sur les politiques
- Deux algorithmes différents pour le calcul des politiques optimales:
 - ◆ **itération par valeurs** (*value iteration*)
 - ◆ **itération par politiques** (*policy iteration*)



Équations de Bellman

- Les **équations de Bellman** nous donnent l'utilité d'un état (c.à-d., l'utilité des politiques optimales exécutées à partir d'un état)

$$U(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')]$$

- Si nous pouvons calculer U , nous pourrions calculer un plan optimal aisément: il suffit de choisir dans chaque état s l'action qui maximise $U(s)$

$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')]$$

Fonction action-utilité

- La fonction action-utilité (***Q-function***) est donnée par

$$\begin{aligned} Q(s,a) &= \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')] \\ &= \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma \max_{a' \in A(s')} Q(s',a')] \end{aligned}$$

- On a donc

$$U(s) = \max_{a \in A(s)} Q(s,a)$$

- Si nous pouvons calculer $Q(s,a)$, pour calculer un plan optimal il suffit de choisir dans chaque état s l'action qui maximise $Q(s,a)$

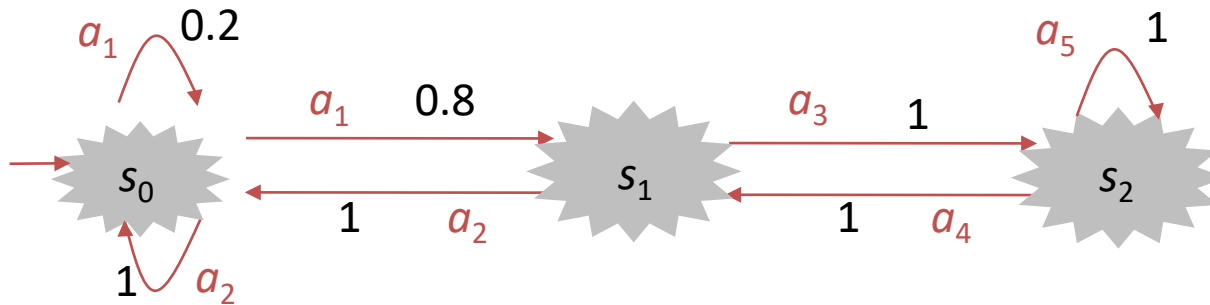
$$\pi^*(s) = \operatorname{argmax}_{a \in A(s)} Q(s,a)$$

Algorithme *Value Iteration*

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

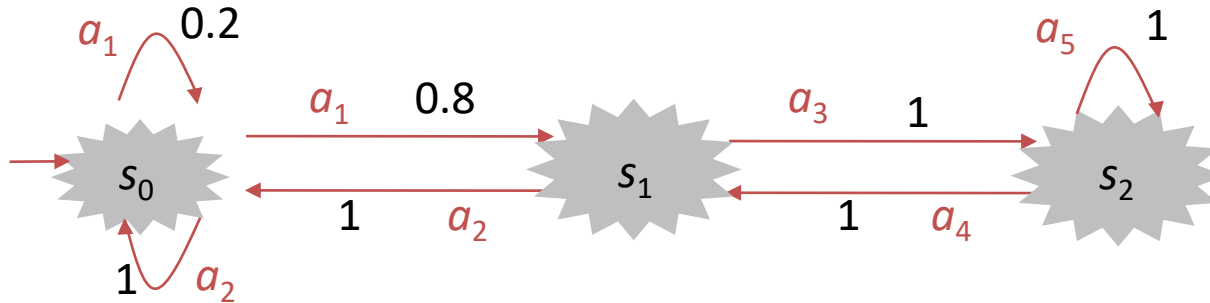
- En mots, on choisit l'action qui **maximise l'espérance** des sommes de récompenses futures
- Complexité:
 - $(O(|S|^4 |A|^2))$ [[Kaelbling, 1996](#)]
 - Polynomial pourvu que le nombre d'itérations pour une politique ϵ -optimale est polynomial [[Littman, Dean, Kaelbling, UAI-95](#)] (chaque itération est $O(|S| |A|^2)$)

Exemple de MDP



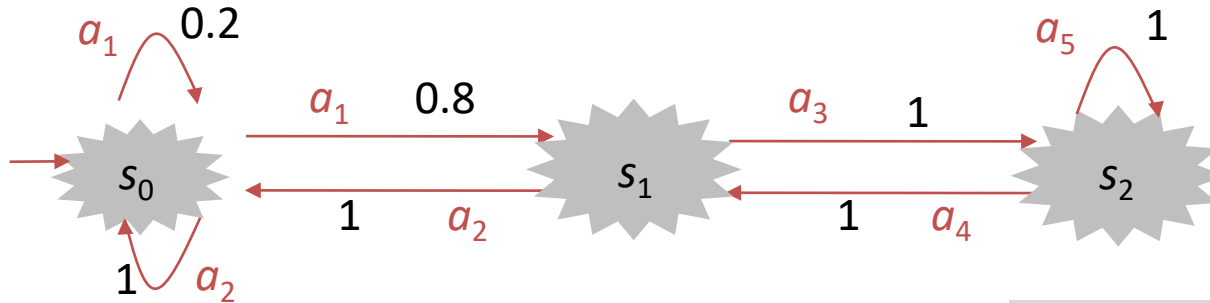
- MDP à 3 états: $S = \{s_0, s_1, s_2\}$
- But: s_2

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:
 - ◆ $R(s, a, s') = 1$ si $s' = s_2$
 $= 0$ sinon
- Le facteur d'escompte est $\gamma = 0.5$

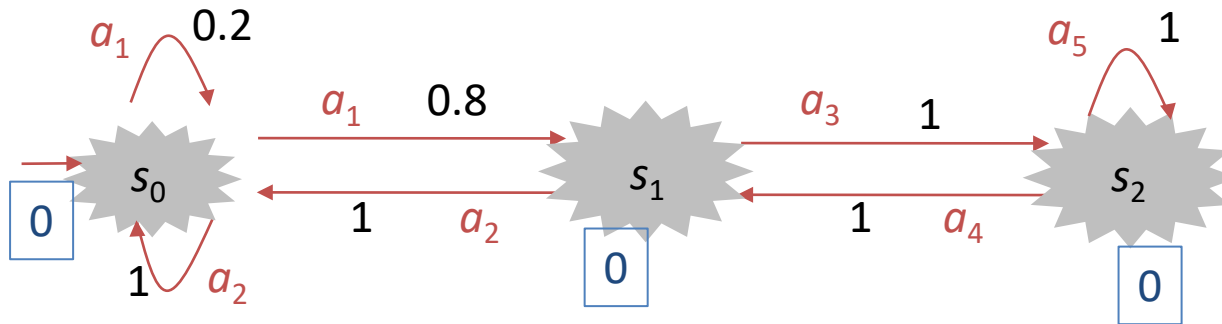
Exemple de MDP



$R(s,a,s') = 0$ si $s' = s_0$ ou $s' = s_1$
 $R(s,a,s') = 1$ pour s_2

- $U(s)$: utilité actuelle l'état s
- $U'(s)$: nouvelle utilité de l'état s
 - ◆ $U'(s) = \max_a Q(s,a)$
- Notons $u_i = U(s_i)$

Value iteration: initialisation



- Valeurs initiales fixées à 0:

$$u_0 = 0$$

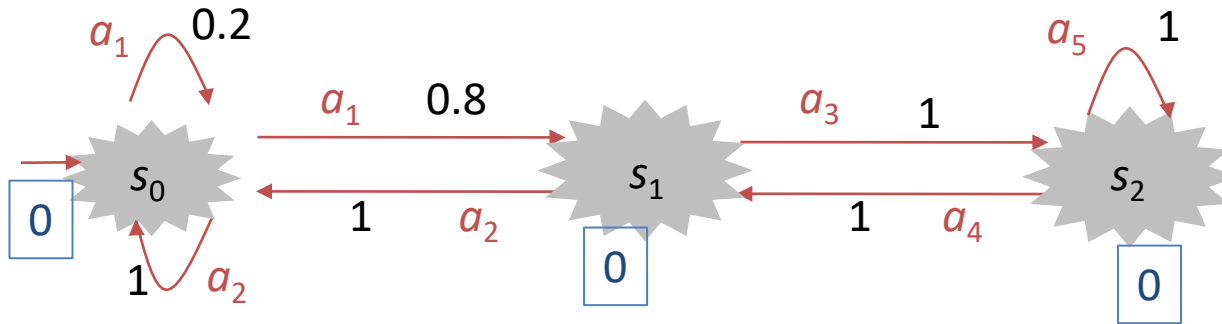
$$u_1 = 0$$

$$u_2 = 0$$

$$R(s, a, s') = 0 \text{ si } s' = s_0 \text{ ou } s' = s_1$$
$$R(s, a, s') = 1 \text{ pour } s_2$$

- Sur la figure, les valeurs U des états sont les étiquettes en bleu

Value iteration: itération #1



$R(s,a,s') = 0$ si $s' = s_0$ ou $s' = s_1$
 $R(s,a,s') = 1$ pour s_2

- Mise à jour droite-gauche des valeurs

$$u'_0 \leftarrow \max\{ 0.2*[0 + 0.5u_0] + 0.8*[0 + 0.5u_1], u_0 \} = \max\{ 0, 0 \} = 0$$

$$u'_1 \leftarrow \max\{ 1*[0 + 0.5u_0], 1*[1 + 0.5u_2] \} = \max\{ 0, 1 \} = 1$$

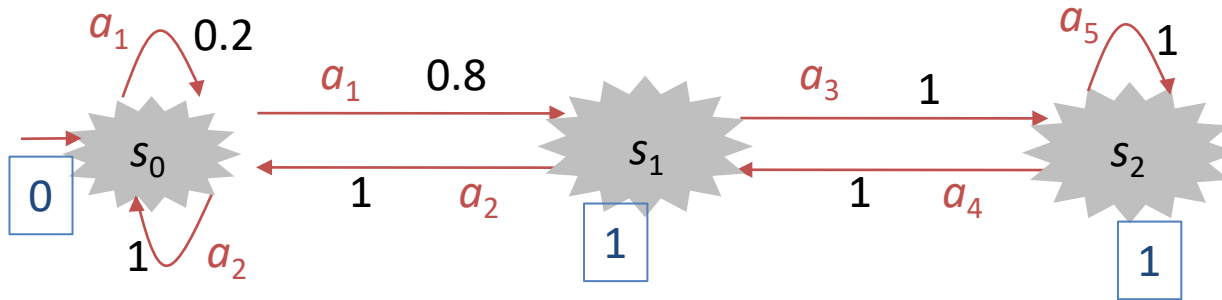
$$u'_2 \leftarrow \max\{ 1*[0 + 0.5u_1], 1*[1 + 0.5u_2] \} = \max\{ 0, 1 \} = 1$$



- Les nouvelles valeurs sont $u_0 = 0$, $u_1 = 1$, $u_2 = 1$

$$Q(s,a) = \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')]$$

Value iteration: itération #2



$R(s,a,s') = 0$ si $s' = s_0$ ou $s' = s_1$
 $R(s,a,s') = 1$ pour s_2

- Mise à jour droite-gauche des valeurs

$$u'_0 \leftarrow \max\{ 0.2*[0 + 0.5u_0] + 0.8*[0 + 0.5u_1], u_0 \} = \max\{ 0.4, 0 \} = 0.4$$

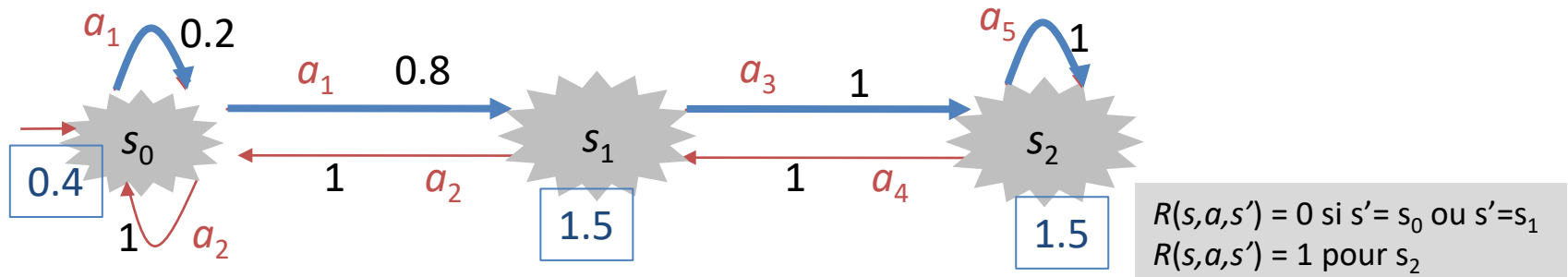
$$u'_1 \leftarrow \max\{ 1*[0 + 0.5u_0], 1*[1 + 0.5u_2] \} = \max\{ 0, 0 \} = 1.5$$

$$u'_2 \leftarrow \max\{ 1*[0 + 0.5u_1], 1*[1 + 0.5u_2] \} = \max\{ 0.5, 1.5 \} = 1.5$$

- Les nouvelles valeurs sont $u_0 = 0.4$, $u_1 = 1.5$, $u_2 = 1.5$

$$Q(s,a) = \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')]$$

Value iteration: itération #2



$$\begin{aligned} \pi(s_0) &= \underset{a}{\operatorname{argmax}} \{Q(s_0, a_1, s_1), Q(s_0, a_2, s_0)\} = \underset{a}{\operatorname{argmax}} \{0.2 * 0.5u_0 + 0.8 * 0.5u_1, 0.5u_0\} \\ &= \underset{a}{\operatorname{argmax}} \{0.2 * 0.5 * 0.4 + 0.8 * 0.5 * 1.5, 0.5 * 0.4\} = \underset{a}{\operatorname{argmax}} \{0.64, 0.2\} = a_1 \end{aligned}$$

$$\pi(s_1) = \underset{a}{\operatorname{argmax}} \{Q(s_1, a_2, s_0), Q(s_1, a_3, s_2)\} = \underset{a}{\operatorname{argmax}} \{0.5u_0, 1 + 0.5u_2\} = \underset{a}{\operatorname{argmax}} \{0.2, 1.75\} = a_3$$

$$\pi(s_2) = \underset{a}{\operatorname{argmax}} \{Q(s_2, a_4, s_1), Q(s_2, a_5, s_2)\} = \underset{a}{\operatorname{argmax}} \{0.5u_1, 1 + 0.5u_2\} = \underset{a}{\operatorname{argmax}} \{0.75, 1.75\} = a_5$$

$$Q(s,a) = \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma U(s')]$$

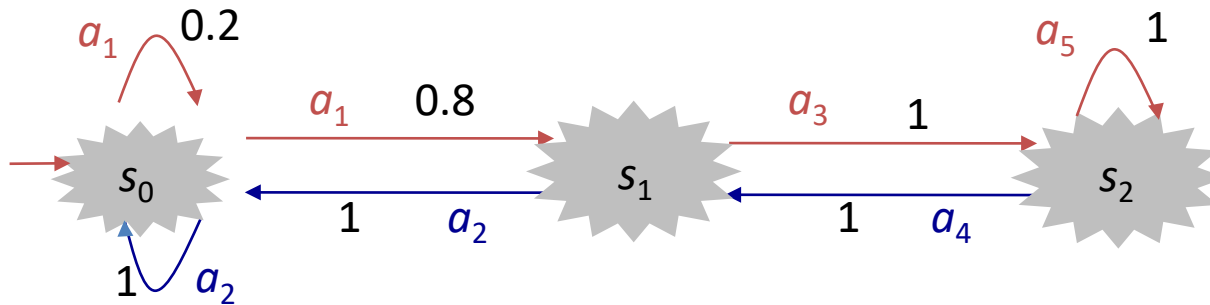
Algorithme *Policy Iteration*

1. Choisir un plan arbitraire π'
2. Répéter jusqu'à ce que π devienne inchangée:
 - I. $\pi := \pi'$
 - II. pour tout s dans S , calculer $U^\pi(s)$ en résolvant le système de $|S|$ équations et $|S|$ inconnues
$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$
 - III. pour tout s dans S
$$a^* = \operatorname{argmax}_{a \in A(s)} Q(s, a)$$

si $Q(s, a^*) > Q(s, \pi(s))$ alors $\pi'(s) := a^*$ sinon $\pi'(s) := \pi(s)$
3. Retourne π

- Converge en temps polynomial pourvu que le nombre d'itérations pour une politique ϵ -optimale est polynomial [\[Littman, Dean, Kaelbling, UAI-95\]](#):
 - ◆ Chaque itération (calcul de la valeur d'un plan) est $O(|S|^3)$
 - ◆ Le nombre d'itérations est $O(|S| |A|^2)$

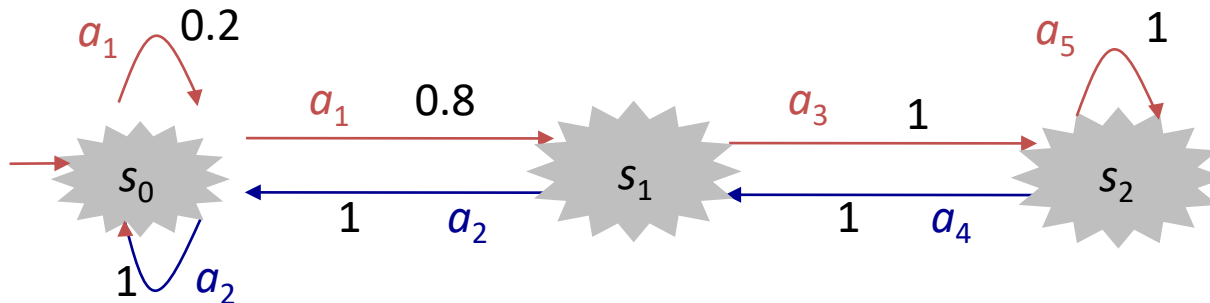
Policy iteration: initialisation



- 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: $u_0 = 1 * [0 + 0.5 * u_0]$;
 $u_1 = 1 * [0 + 0.5 * u_0]$;
 $u_2 = 1 * [0 + 0.5 * u_1]$

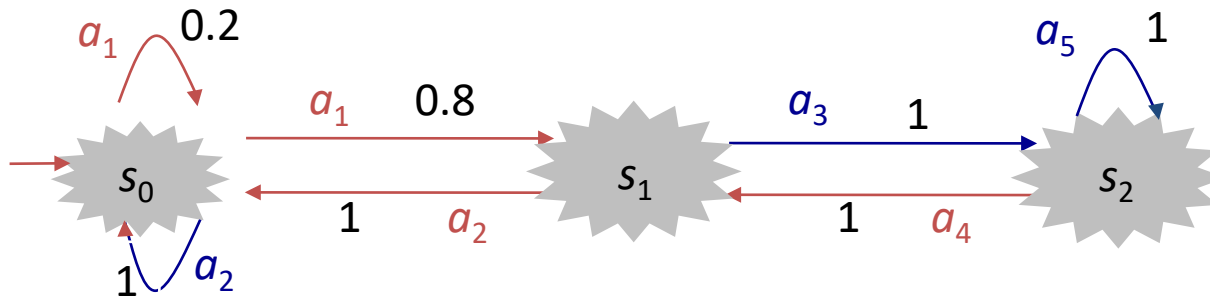
Solution: $u_0 = 0, u_1 = 0, u_2 = 0$

III. $s_0 \rightarrow a_1: 0.2 * [0 + 0.5 * u_0] + 0.8 * [0 + 0.5 * u_1] = 0$;
 $s_1 \rightarrow a_3: 1 * [1 + 0.5 * u_2] = 1 > 0$;
 $s_2 \rightarrow a_5: 1 * [1 + 0.5 * u_2] = 1 > 0$;
 $\pi' = \{ s_0 \rightarrow a_2, s_1 \rightarrow a_3, s_2 \rightarrow a_5 \}$

ne change pas
change
change

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Policy iteration: itération #2



I. $\pi = \pi'$

II. Équations: $u_0 = 1 * [0 + 0.5 * u_0]$;
 $u_1 = 1 * [1 + 0.5 * u_2]$;
 $u_2 = 1 * [1 + 0.5 * u_2]$

Solution: $u_0 = 0, u_1 = 2, u_2 = 2$

III. $s_0 \rightarrow a_1: 0.2 * [0 + 0.5 * u_0] + 0.8 * [0 + 0.5 * u_1] = 0.8 > 0$

$s_1 \rightarrow a_2: 1 * [0 + 0.5 * u_0] = 0 < 2$;

$s_2 \rightarrow a_4: 1 * [0 + 0.5 * u_1] = 1 < 2$;

change

ne change pas

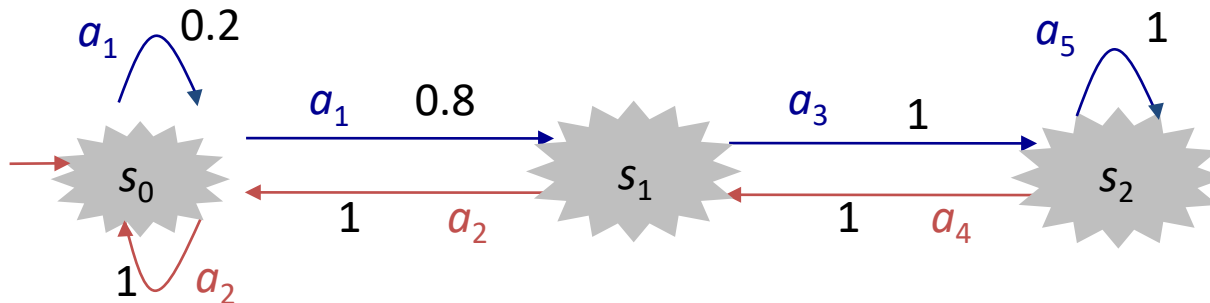
ne change pas

$\pi' = \{ s_0 \rightarrow a_1, s_1 \rightarrow a_3, s_2 \rightarrow a_5 \}$, c-à-d. Π

● Solution finale: π

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Policy iteration: itération #3



I. $\pi = \pi'$

II. Équations: $u_0 = 0.2 * [0 + 0.5 * u_0] + 0.8 * [0 + 0.5 * u_1]$
 $u_1 = 1 * [1 + 0.5 * u_2];$
 $u_2 = 1 * [1 + 0.5 * u_2]$

Solution: $u_0 = 4/45, u_1 = 2, u_2 = 2$

III. $s_0 \rightarrow a_2: 1 * [0 + 0.5 * u_0] = 2/45 < 4/45$
 $s_1 \rightarrow a_2: 1 * [0 + 0.5 * u_0] = 2/45 < 2;$
 $s_2 \rightarrow a_4: 1 * [0 + 0.5 * u_1] = 1 < 2;$
 $\pi' = \{ s_0 \rightarrow a_2, s_1 \rightarrow a_3, s_2 \rightarrow a_5 \}$

ne change pas
 ne change pas
 ne change pas

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Modified Policy Iteration

- À l'étape II de l'algorithme, on lieu de calculer U^π de façon exacte par la résolution du système d'équations, on l'approxime avec l'itération par valeur simplifiée.
- L'itération par valeur simplifiée est un algorithme général pour à approximer la valeur d'une politique donnée (dans ce cas-ci la politique π à l'étape II) par nombre d'étapes fixe de mises à jour de *value-iteration*

Répéter N fois

Pour chaque état s

$$U^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma U^\pi(s')]$$

- ◆ À l'infini, on obtient U^π
- ◆ C.-à-d., plus N est grand, plus on se rapproche de U^π
- ◆ Le N est choisi de façon empirique.

Asynchronous Policy Iteration

- À chaque iteration de policy itération
 - ◆ Choisir un sous ensemble d'états (au lieu de tous les états du MDP)
 - ◆ Appliquer à ce sous-ensemble,
 - » Soit une évaluation approximative de la politique (par *simplified value iteration*)
 - » Soit une évaluation exacte par résolution du système d'équations

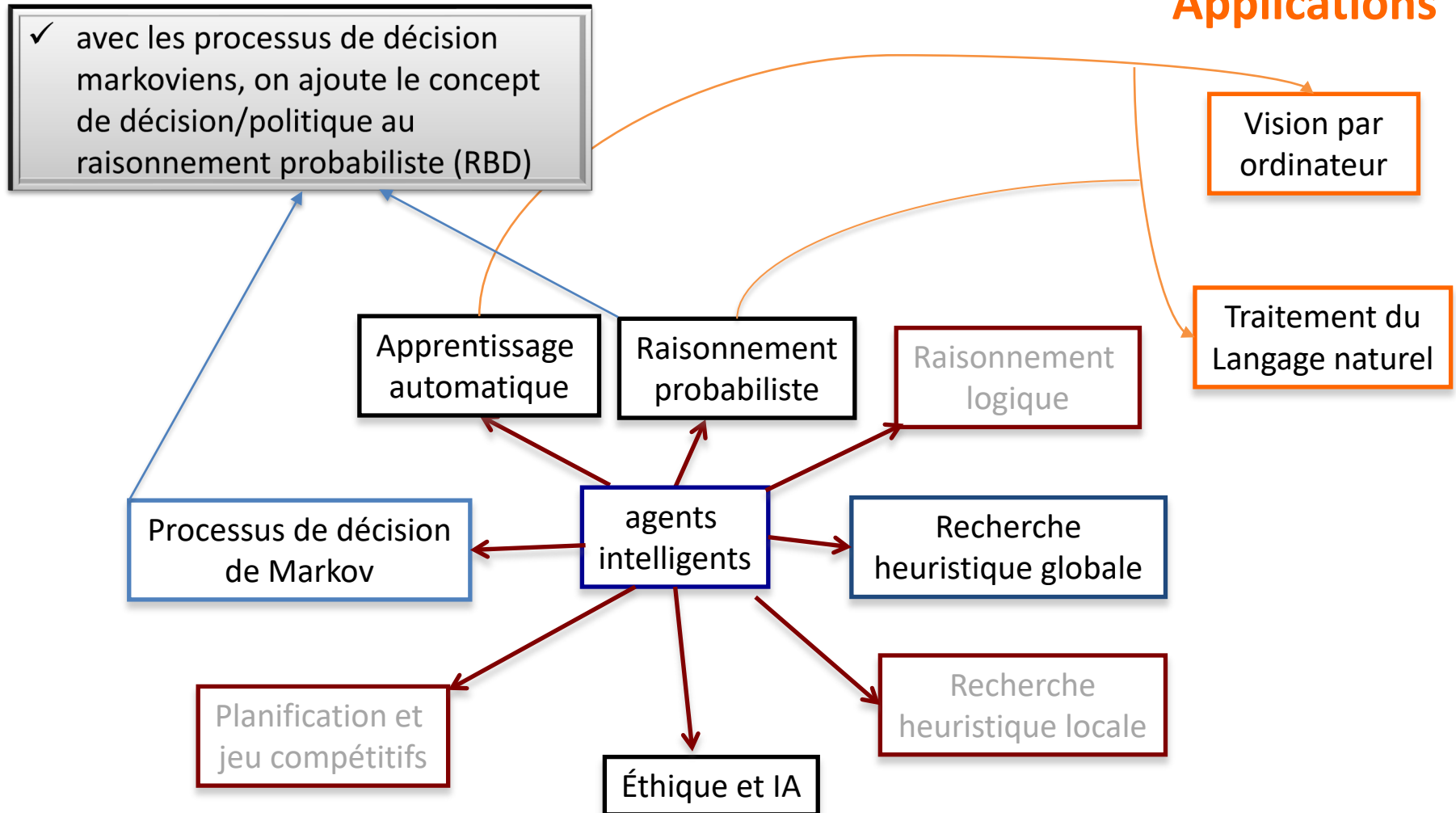
Au de là de MDP ...

- Les processus de décision markoviens sont attrayants parce qu'ils offrent un cadre de planification qui combine raisonnement probabiliste, théorie de la décision et optimisation avec élégance
- Les algorithmes *value-iteration* et *policy-iteration* ne sont pas efficaces (espace d'états trop grand). Il existe des approches approximatives par échantillonnage et des approches hiérarchiques.
- Ces algorithmes supposent que nous avons un modèle. Ce sont des algorithmes de planification.
- Les algorithmes d'apprentissage par renforcement sont fondés sur ces concepts pour apprendre la politique à partir d'interactions avec l'environnement.

Sujets couverts par le cours

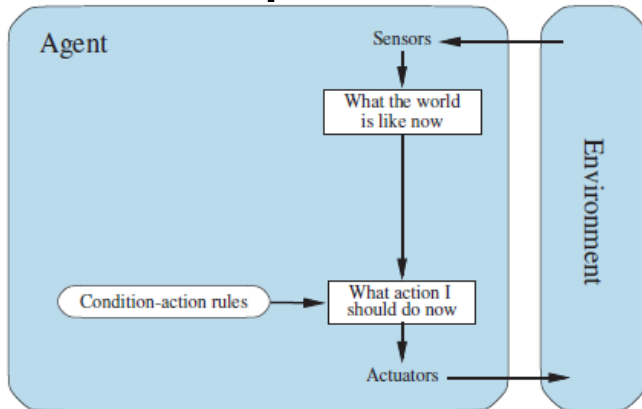
Concepts et algorithmes

Applications

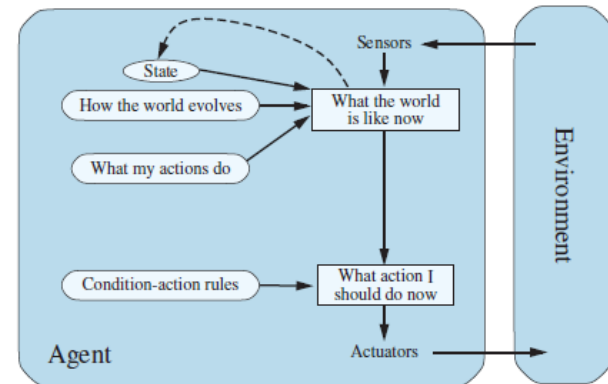


MDP pour quel type d'agents?

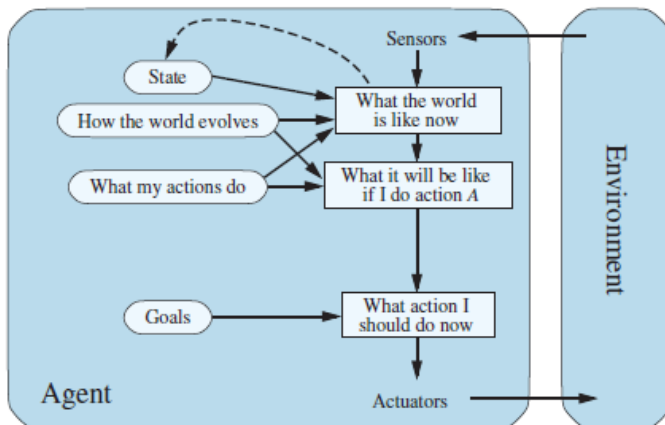
Simple reflex



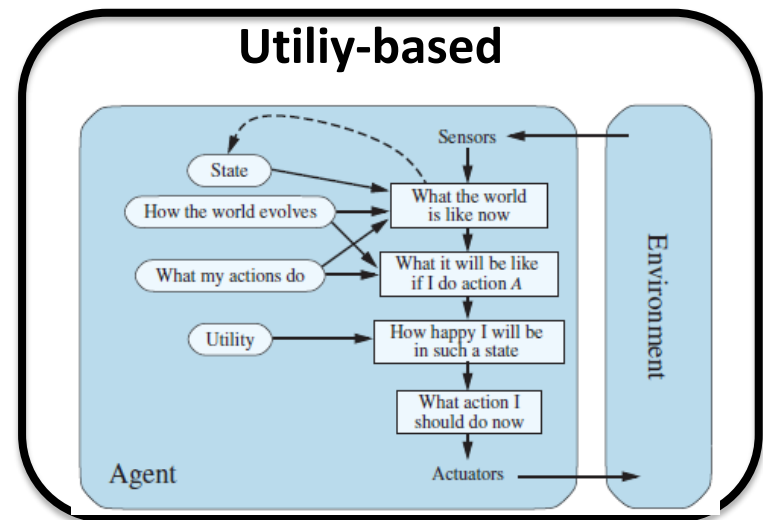
Model-based reflex



Goal-based



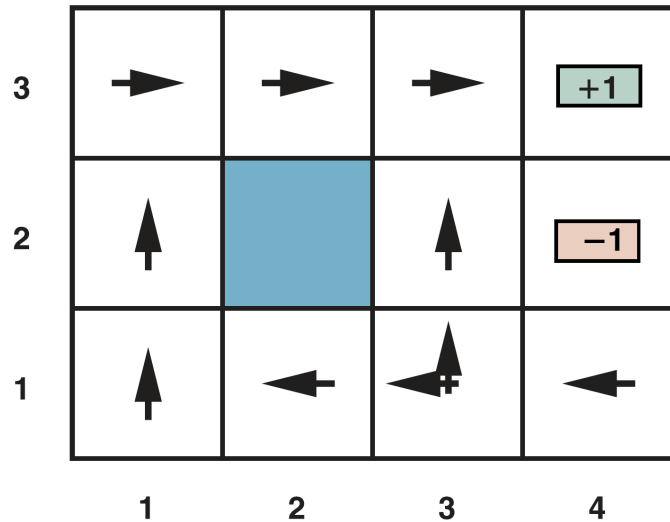
Utility-based



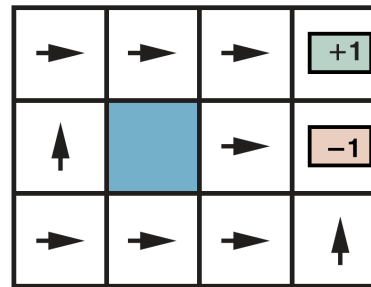
Vous devriez être capable de...

- Donner la définition d'un processus de décision markovien
 - ◆ espace d'états
 - ◆ actions
 - ◆ modèle de transition
 - ◆ fonction de récompense
 - ◆ décisions
 - ◆ plan/politique
- Expliquer et simuler *value iteration*
- Expliquer et simuler *policy iteration*
- Expliquer *asynchronous policy iteration*

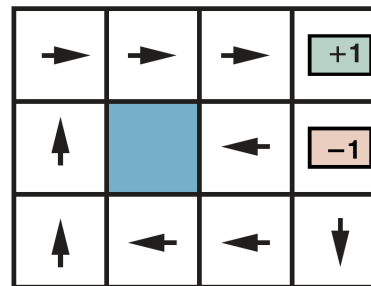
Examples



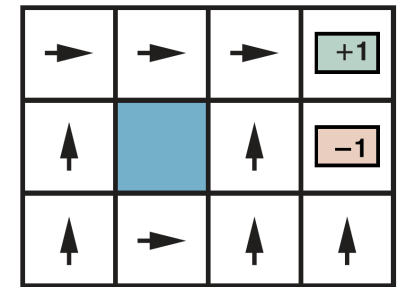
(a)



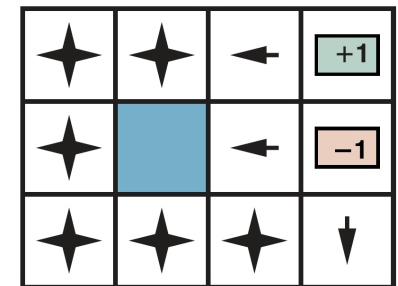
$$r < -1.6497$$



$$-0.0274 < r < 0$$



$$-0.7311 < r < -0.4526$$



$$r > 0$$

(b)

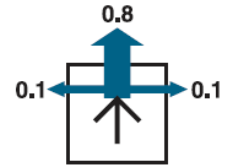


Figure 17.2 (a) The optimal policies for the stochastic environment with $r = -0.04$ for transitions between nonterminal states. There are two policies because in state (3,1) both *Left* and *Up* are optimal. (b) Optimal policies for four different ranges of r .

Rappel: systèmes d'équations linéaires

- Soit le système d'équations:

$$u_0 = 0 + 0.5 * (0.2 * u_0 + 0.8 * u_1);$$

$$u_1 = 0 + 0.5 * (1 * u_2);$$

$$u_2 = 1 + 0.5 * (1 * u_2)$$

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

$$0 = -0.9 u_0 + 0.4 u_1 \quad (1)$$

$$0 = -u_1 + 0.5 u_2 \quad (2)$$

$$-1 = -0.5 u_2 \quad (3)$$

- De l'équation (3), on conclut que $u_2 = -1 / -0.5 = 2$
- De l'équation (2), on conclut que $u_1 = 0.5 u_2 = 1$
- De l'équation (1), on conclut que $u_0 = 0.4 u_1 / 0.9 = 4/9$

Rappel: systèmes d'équations linéaires

- Soit le système d'équations:

$$u_0 = 0 + 0.5 * (0.2 * u_0 + 0.8 * u_1);$$

$$u_1 = 0 + 0.5 * (1 * u_2);$$

$$u_2 = 1 + 0.5 * (1 * u_2)$$

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

$$0 = -0.9 u_0 + 0.4 u_1 \quad (1)$$

$$0 = -u_1 + 0.5 u_2 \quad (2)$$

$$-1 = -0.5 u_2 \quad (3)$$

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

$$A = \begin{pmatrix} -0.9 & 0.4 & 0 \\ 0 & -1 & 0.5 \\ 0 & 0 & -0.5 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \quad u = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix}$$

Rappel: systèmes d'équations linéaires

- Soit le système d'équations:

$$u_0 = 0 + 0.5 * (0.2 * u_0 + 0.8 * u_1);$$

$$u_1 = 0 + 0.5 * (1 * u_2);$$

$$u_2 = 1 + 0.5 * (1 * u_2)$$

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

$$0 = -0.9 u_0 + 0.4 u_1 \quad (1)$$

$$0 = -u_1 + 0.5 u_2 \quad (2)$$

$$-1 = -0.5 u_2 \quad (3)$$

- 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)
```

```
>>> u = numpy.dot(Ainv,b)
```

```
>>> print u
```

```
[ 0.44444444  1.      2.    ]
```