Apprentissage par renforcement

Présentation ISN 2016 - 17/03/2016

Vincent THOMAS – université de lorraine
Vincent.thomas@loria.fr

http://webloria.loria.fr/~vthomas/mediation/ISN_2015/

Point de départ

- Sujet de recherche
  - Prise de décision séquentielle
  - Apprentissage par renforcement

- Principe
  - Systèmes qui savent agir
    - Robots
    - Systeme diagnostic (médical, ...)
  - Comment donner des comportements intelligents ?

Plan

- Conditionnement opérant
- Problème de prise de décision séquentiel
- Monde deterministe
  - Equation de Bellman
  - Planification en environnement connu
  - Apprentissage en environnement inconnu
- Monde stochastique
- Problèmes ouverts

"Un chocolat ?"

- Inspiration "Big Bang Theory" Ep03 S03

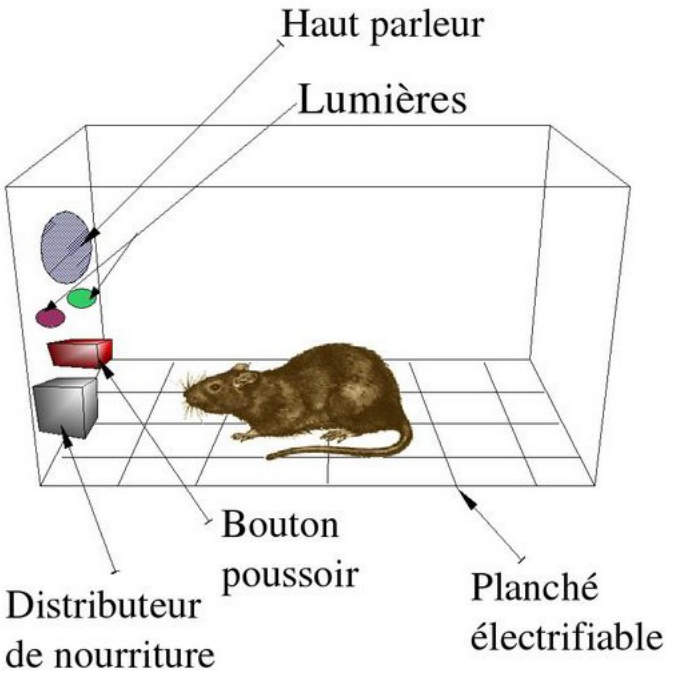
https://www.youtube.com/watch?v=JA96Fba-WHk

Conditionnement opérant

- Behaviorisme
  - Lois stimulus / réponse

- Conditionnement opérant (Skinner 1898)
  - Boite de Skinner
  - cf wikipedia

- Objectif informatique
  - Apprendre automatiquement
  - A partir du vécu



Plan

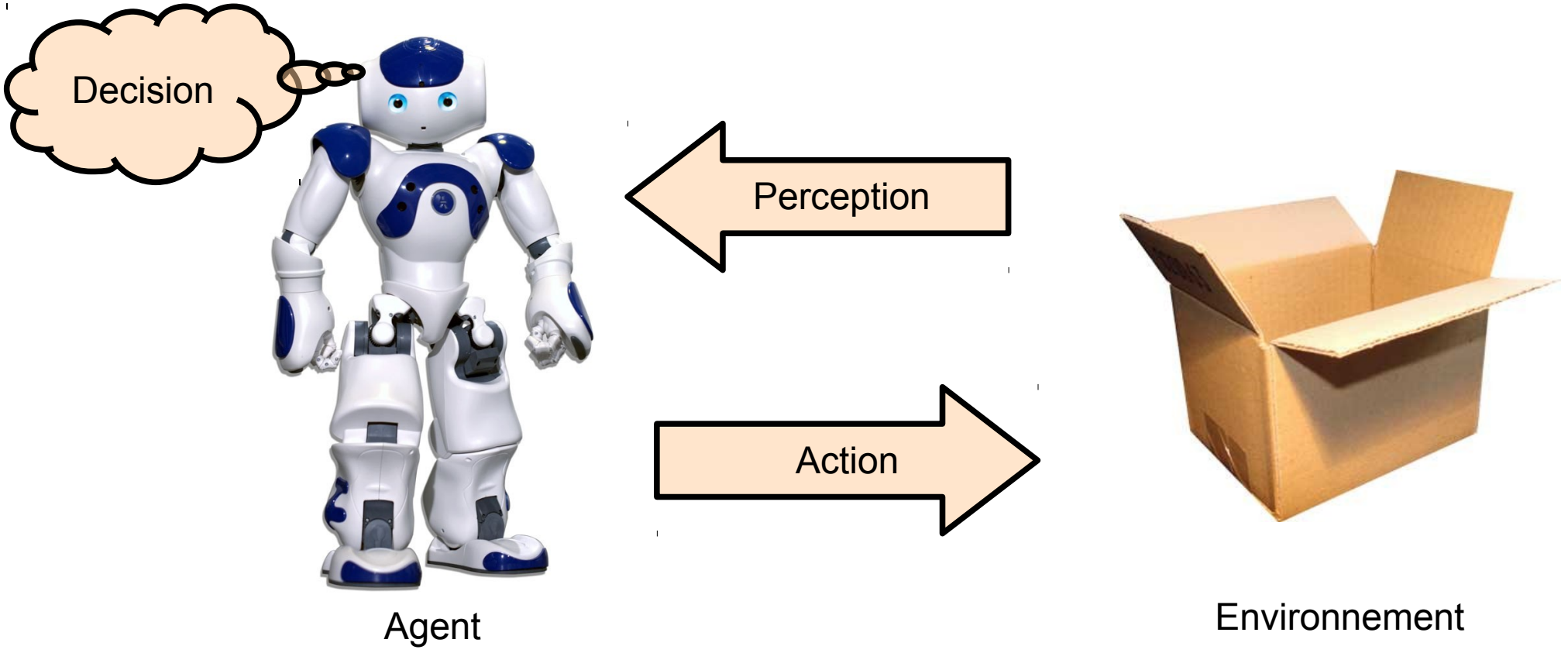
- Conditionnement opérant
- Problème de prise de décision séquentiel
- Monde deterministe
  - Equation de Bellman
  - Planification en environnement connu
  - Apprentissage en environnement inconnu
- Monde stochastique
- Problèmes ouverts

Prise de décision sequentielle

- Informatique
  - Poser un modèle générique et réutilisable
  - Construire solutions génériques

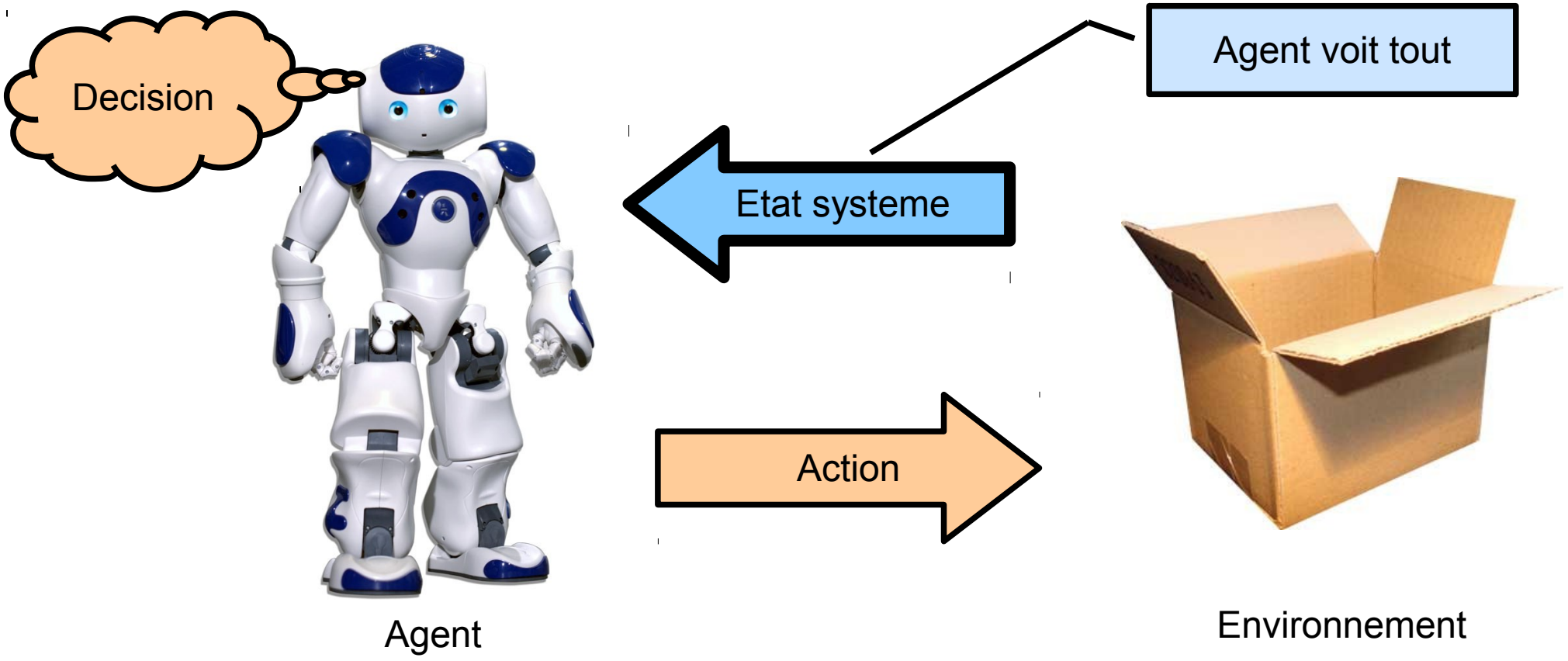
- Objectif
  - Modéliser un système qui sait agir sur le monde
  - Cas simple: l'agent voir tout

Boucle perception-action



Un **agent** intelligent est une entité réelle ou artificielle, dotée de **capteurs** et **d'effecteurs**, capable d'agir de manière **autonome** grâce à une **fonction de décision**

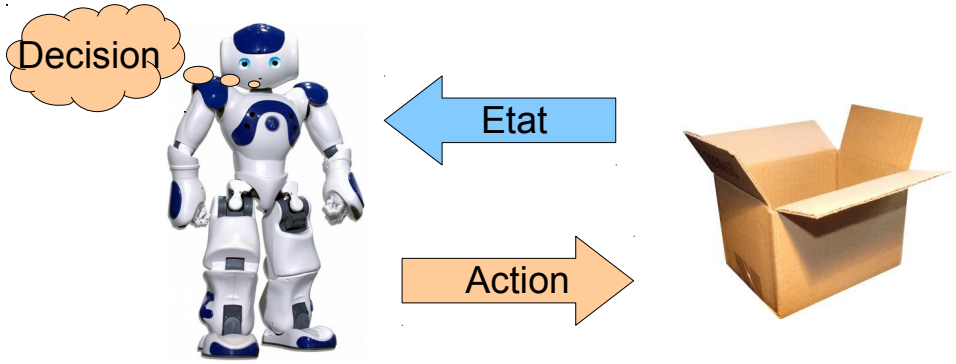
Boucle perception-action (simple)



Un **agent** intelligent est une entité réelle ou artificielle, dotée de **capteurs** et **d'effecteurs**, capable d'agir de manière **autonome** grâce à une **fonction de décision**

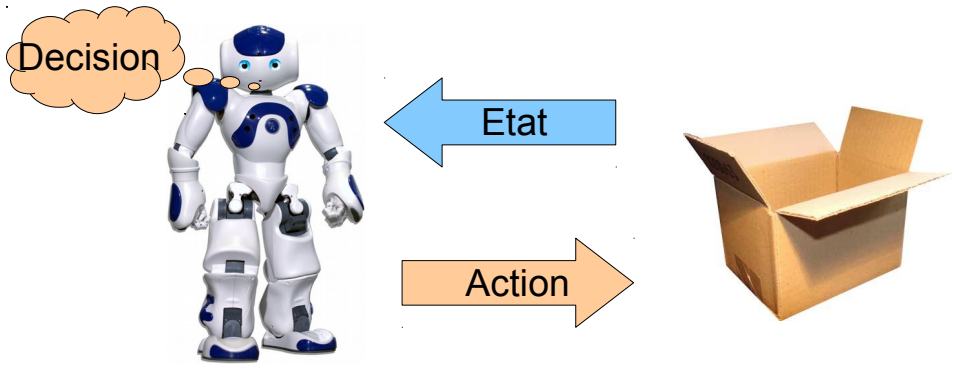
Boucle perception-action (simple)

- Comment modéliser cela ?



Boucle perception-action (simple)

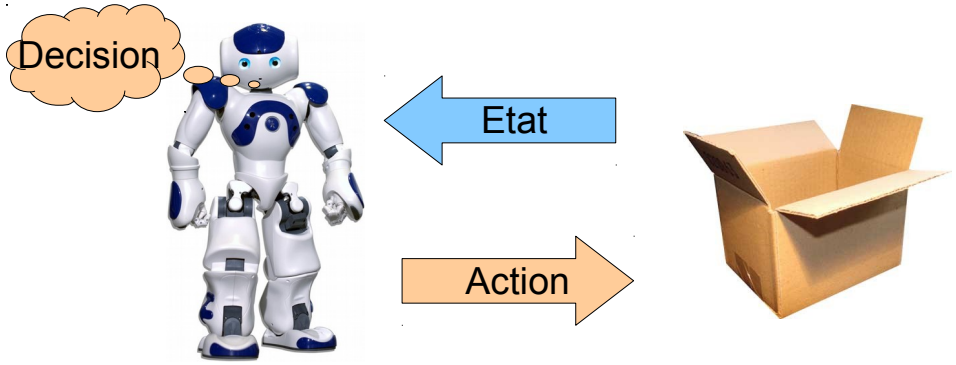
- Comment modéliser cela ?



- 2 ensembles
  - S : ensemble d'états (à définir)
    - Un etat = {caractéristiques pour prédire au mieux}
  - A : ensemble d'actions

Boucle perception-action (simple)

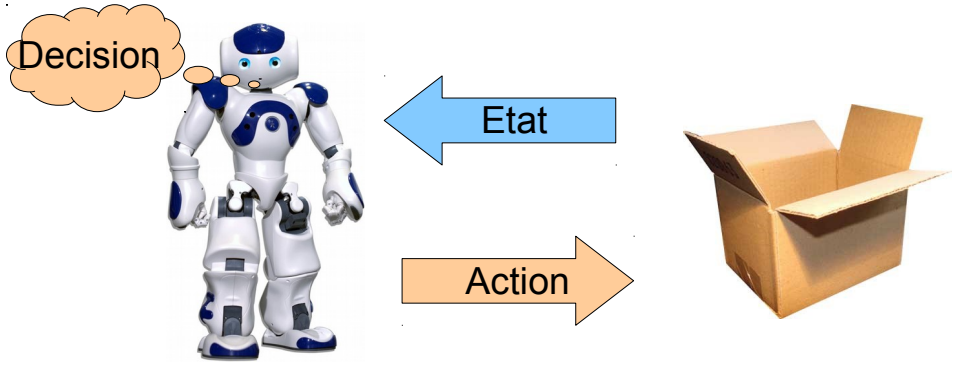
- Comment modéliser cela ?



- 2 ensembles (etats, actions)
- 2 fonctions

Boucle perception-action (simple)

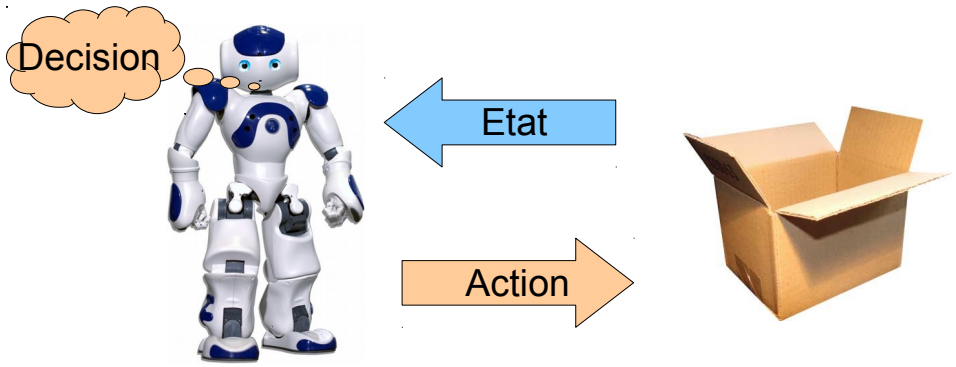
- Comment modéliser cela ?



- 2 ensembles (etats, actions)
- 2 fonctions
  - Fonction transition: S x A -> S
  - Fonction décision: S -> A

Boucle perception-action (simple)

- Comment modéliser cela ?



- 2 ensembles (etats, actions)
- 2 fonctions
  - Fonction transition: S x A -> S
  - Fonction décision: S -> A

Écrire code python

Coder système agent

```
class Probleme:
    """permet de definir un probleme"""

    def actions(self):
        """returne la liste d'actions"""
        return ([])

    def etats(self):
        """returne la liste d'etats"""
        return ([])


    def transition(self,s,a):
        """definit les consequence d une action"""
        return(s)


    def recompense(self,s,a,sarr):
        """definit la recompense obtenue"""
        return(0)
```

Exemple de Système

- Une lampe
  - S = { "Allumee", "Eteinte"}

- Deux actions
  - A = { "Appuyer", "Ne rien faire"}

Exemple de Système

- Une lampe
  - S = { "Allumee", "Eteinte"}

Écrire code python

- Deux actions
  - A = { "Appuyer", "Ne rien faire"}

- Transition T: S x A -> S

(S	X A)	→ S
« Allume »	« Appuyer »	→ « Eteinte »
« Allume »	« Ne rien faire »	→ « Allume »
« Eteinte »	« Appuyer »	→ « Allume »
« Eteinte »	« Ne rien faire »	→ « Eteinte »

Coder système agent

```
class Lampe:

    def actions(self):
        return(['appuyer','rien'])

    def etats(self):
        return(['allume','eteint'])

    def transition(self,s,a):
        if (a=='rien'):
            return(s)
        if (a=='appuyer'):
            if (s=='allume'):
                return('eteint')
            if (s=='eteint'):
                return('allume')
        return('erreur')
```

Comportement agent

- Fonction politique: S -> A
  - Un exemple avec lampe

S	A
« Allume »	« Ne rien faire»
« Eteinte »	« Appuyer »

- Comportement
  - On est dans s
  - Choisit a selon strategie(s)
  - Evoluer dans s' selon T(s,a)

- Nombre de politiques différentes ?

Comportement agent

- Fonction politique: S -> A
  - Un exemple avec lampe

S	A
« Allume »	« Ne rien faire»
« Eteinte »	« Appuyer »

- Comportement
  - On est dans s
  - Choisit a selon strategie(s)
  - Evoluer dans s' selon T(s,a)

- Nombre de politiques différentes ? $|A|^{|S|}$

Code python execution agent

```
pb=Lampe()

pi={}
pi['allume']='rien'
pi['eteint']='appuyer'

print("*** test execution ***")
systemExec = SystemeExecute(pb)
systemExec.executerPi(pi,'eteint',10)
```

```
class SystemeExecute:

    def __init__(self,pb):
        self.pb=pb

    def executerPi(self,pi,depart,nb):
        s=depart
        for i in range(nb):
            action=pi[s]
            sFin=pb.transition(s,action)
            print(s," -> ",action," : ",sFin)
            s=sFin
```

Probleme agent

- Pour le moment
  - Un systeme dynamique controlable (via actions)
  - Des politiques

- Pour définir problème agent
  - Besoin de définir un critère de performance
  - Choisir la meilleure politique
  - Récompense -> Score (positif/négatif)
  - Agent rationnel -> Maximiser la somme

MDP

- Markov Décision Process (cadre deterministe)
  - S : ensemble états
  - A : ensemble actions
  - T : transition : S x A -> S
  - R : récompenses : S x A x S -> Réel
- Solution
  - pi : S -> A qui maximise somme des récompenses
- Evaluation solution (par execution)

Exemple lampe

- Fonction récompense
  - +10 quand j'allume la lampe

- S x A x S -> R (mais juste sur S et A)
  - "Allume" x "Appuyer" -> 0
  - "Eteindre" x "Appuyer" -> +10
  - "Allume" x "Ne rien faire" -> 0
  - "Eteindre" x "Ne rien faire" -> 0

Code python récompense


```
class Lampe:

    def actions(self):
        return(['appuyer','rien'])

    def etats(self):
        return(['allume','eteint'])


    def transition(self,s,a):
        if (a=='rien'):
            return(s)
        if (a=='appuyer'):
            if (s=='allume'):
                return('eteint')
            if (s=='eteint'):
                return('allume')
        return('erreur')


    def recompense(self,s,a,sarr):
        if (s=='eteint') and (a=='appuyer'):
            return(10)
        return(0)
```

Code python execution politique

- Récupére la somme des récompenses en plus

Code python execution politique


```
class SystemeExecute:

    def __init__(self,pb):
        self.pb=pb

    def executerPi(self,pi,depart,nb):
        s=depart
        for i in range(nb):
            action=pi[s]
            sFin=pb.transition(s,action)
            print(s," -> ",action," : ",sFin)
            s=sFin

    def executerPiRec(self,pi,depart,nb):
        s=depart
        somme=0
        for i in range(nb):
            action=pi[s]
            sFin=pb.transition(s,action)
            r=pb.recompense(s,action,sFin)
            somme=somme+r
            print(s," -> ",action," : ",sFin,"<",r,">")
            s=sFin
        return somme
```

Code python execution politique


```
class SystemeExecute:

    def __init__(self,pb):
        self.pb=pb

    def executerPi(self,pi,depart,nb):
        s=depart
        for i in range(nb):
            action=pi[s]
            sFin=pb.transition(s,action)
            print(s," -> ",action," : ",sFin)
            s=sFin

    def executerPiRec(self,pi,depart,nb):
        s=depart
        somme=0
        for i in range(nb):
            action=pi[s]
            sFin=pb.transition(s,action)
            r=pb.recompense(s,action,sFin)
            somme=somme+r
            print(s," -> ",action," : ",sFin,"<",r,">")
            s=sFin
        return somme
```

Resultat execution

- Lancer l'execution
- Regarder résultat

Resultat execution

- Lancer l'execution

```
*** politique ***
{'eteint': 'appuyer', 'allume': 'rien'}

*** test execution recompense ***
eteint  ->  appuyer  :  allume < 10 >
allume  ->  rien  :  allume < 0 >
allume  ->  rien  :  allume < 0 >
allume  ->  rien  :  allume < 0 >
allume  ->  rien  :  allume < 0 >
allume  ->  rien  :  allume < 0 >
allume  ->  rien  :  allume < 0 >
allume  ->  rien  :  allume < 0 >
allume  ->  rien  :  allume < 0 >
allume  ->  rien  :  allume < 0 >
somme:  10
```

- Selon vous, est-ce la meilleure politique ?

Resultat execution

- Autre politique

```
*** politique ***
{'allume': 'appuyer', 'eteint': 'appuyer'}

*** test execution recompense ***
eteint  ->  appuyer  :  allume < 10 >
allume  ->  appuyer  :  eteint < 0 >
eteint  ->  appuyer  :  allume < 10 >
allume  ->  appuyer  :  eteint < 0 >
eteint  ->  appuyer  :  allume < 10 >
allume  ->  appuyer  :  eteint < 0 >
eteint  ->  appuyer  :  allume < 10 >
allume  ->  appuyer  :  eteint < 0 >
eteint  ->  appuyer  :  allume < 10 >
allume  ->  appuyer  :  eteint < 0 >
somme:  50
```

Coder probleme agent

- Ecrire une classe python
  - qui représente un probleme
  - sans écrire le contenu des fonctions

- Markov Décision Process (cadre deterministe)
  - S : ensemble états
  - A : ensemble actions
  - T : transition : S x A -> S
  - R : récompenses : S x A -> R

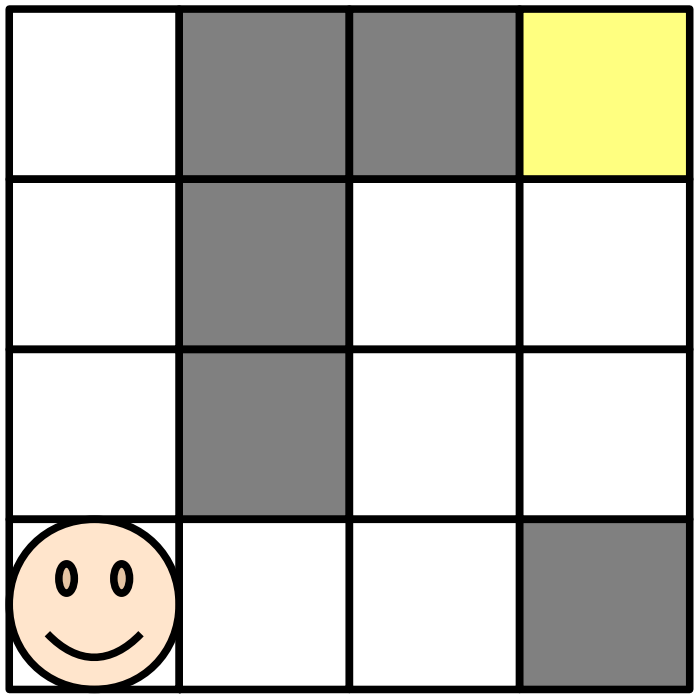
Coder probleme agent


```
class Probleme:
    """permet de definir un probleme"""

    def actions(self):
        """returne la liste d'actions"""
        return ([])

    def etats(self):
        """returne la liste d'etats"""
        return ([])


    def transition(self,s,a):
        """definit les consequence d une action"""
        return(s)


    def recompense(self,s,a,sarr):
        """definit la recompense obtenue"""
        return(0)
```

Exemple 2 - Labyrinthe

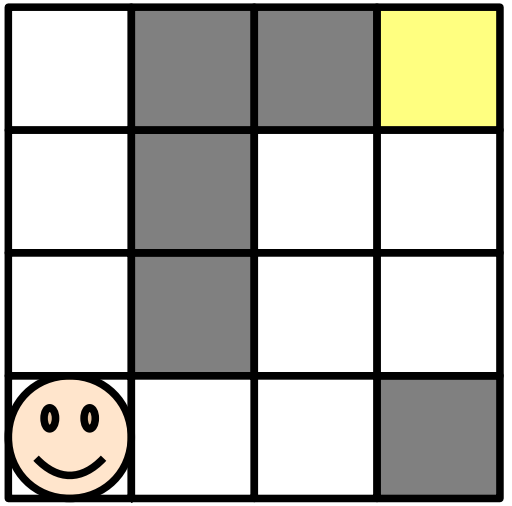
Probleme labyrinthe

- Labyrinthe à traverser avec des trous



Probleme labyrinthe

- Labyrinthe à traverser avec des trous



- Etat : position robot
- Action : N, S, E, O
- Transition :
  - position x action -> position
- Recompense
  - Position == trou => -5
  - Position == arrivee ==> +100
  - Deplacer ==> -1

Probleme labyrinthe

- Etats et actions

```
    def actions(self):
        return(['N','S','E','O'])

    def etats(self):
        etats=[]
        for i in range(0,11):
            for j in range(0,11):
                etats+=[(i,j)]

        #etat final
        etats+=[(-1,-1)]

        return(etats)
```

Probleme labyrinthe

- Transition

```
def transition(self,s,a):
        """fait evoluer d une unite le systeme"""
        x=s[0];
        y=s[1];

        #si on part de 10,10 on retourne en 0
        if (x==10) and (y==10):
            return(-1,-1)
        if (x==-1) and (y==-1):
            return(-1,-1)

        if (a=='N'):
            y=y-1
            if (y<0):
               y=0
        if (a=='S'):
            y=y+1
            if (y>10):
                y=10
        if (a=='E'):
            x=x+1
            if (x>10):
                x=10
        if (a=='O'):
            x=x-1
            if (x<0):
                x=0
        return((x,y))
```

Probleme labyrinthe

- Recompense

```
def recompense(self,s,a,sarr):

        """ok si on est en 10,10"""
        if ((sarr[0]==10)and(sarr[1]==10)):
            return(100)

        if (sarr in self.trou):
            return(-5)

        if (sarr[0]==-1)and (sarr[1]==-1):
            return(0)

        return(-1)
```

Probleme labyrinthe

- Recompense

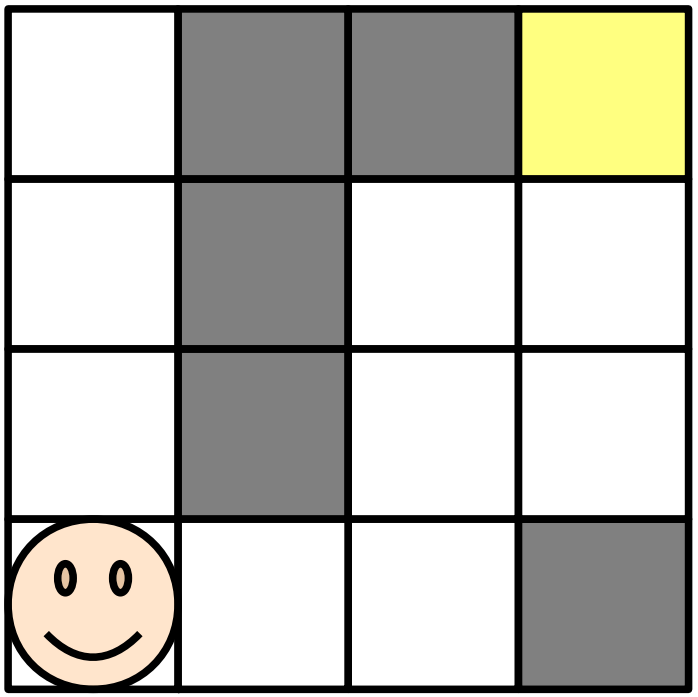
```
def recompense(self,s,a,sarr):

        """ok si on est en 10,10"""
        if ((sarr[0]==10)and(sarr[1]==10)):
            return(100)

        if (
```

Cf code sur site

Probleme du labyrinthe

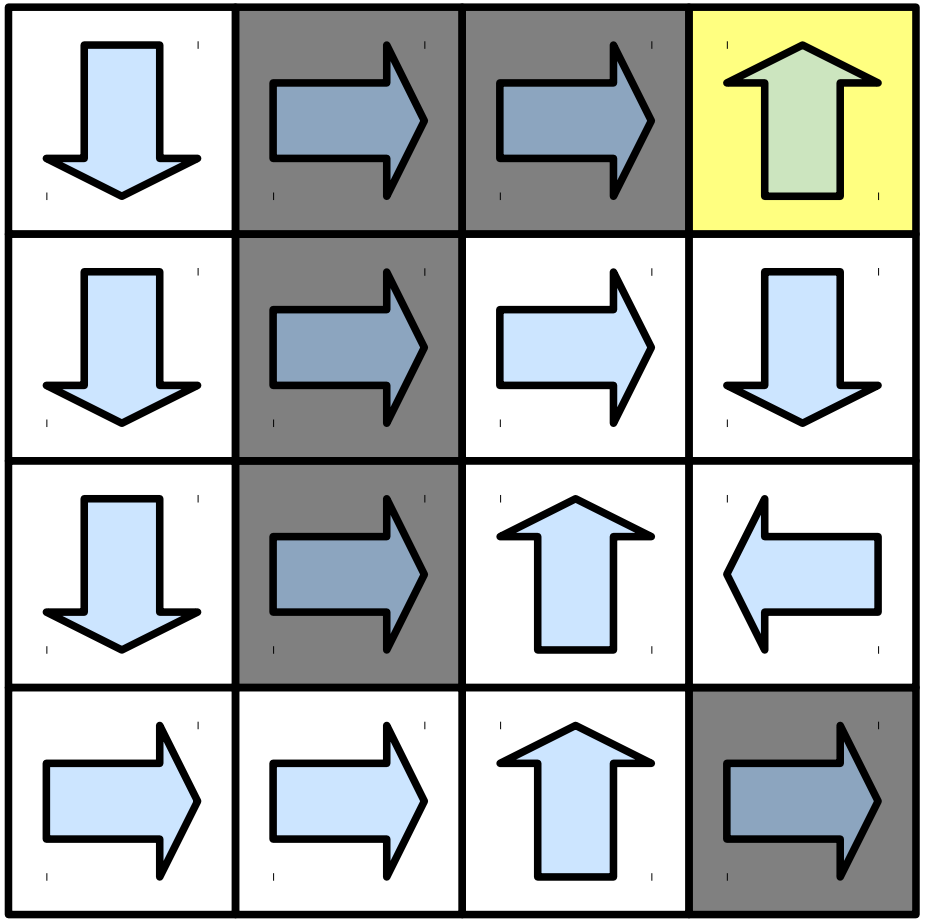
- Exemple de solution ?

- Politique : S -> A
  - Générateur de politiques



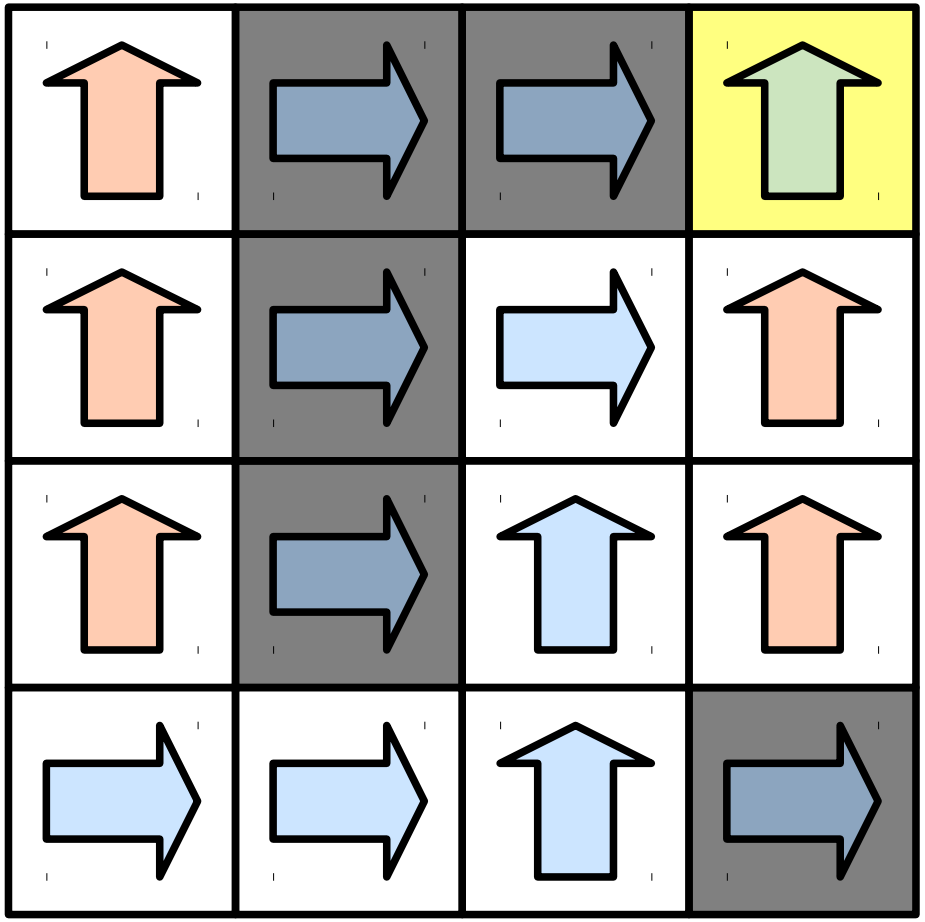
Probleme du labyrinthe

- Exemple 1 de solution ?



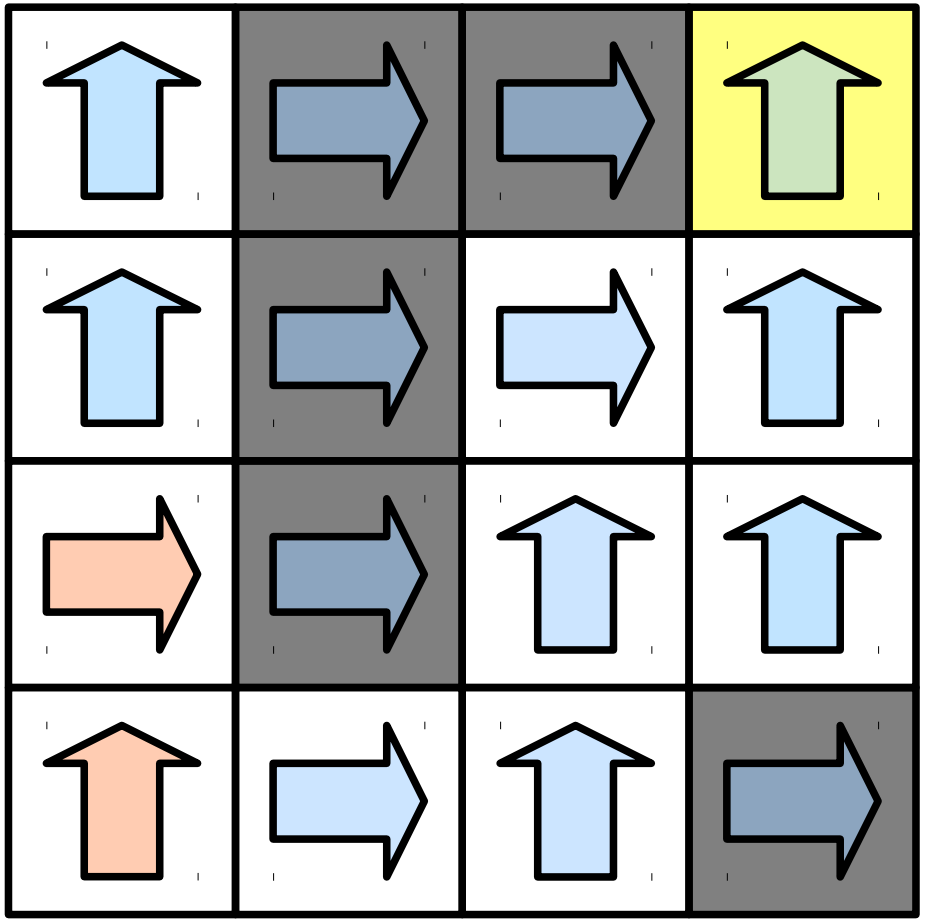
Probleme du labyrinthe

- Autre Exemple de solution ?

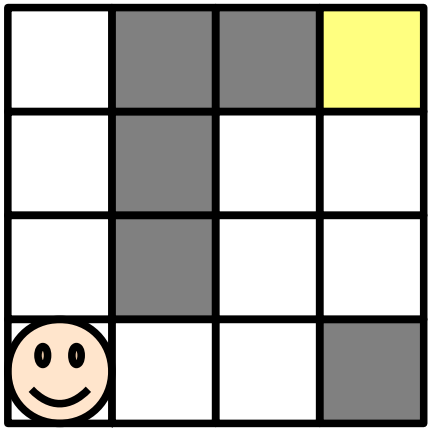


Probleme du labyrinthe

- Autre Exemple de solution ?

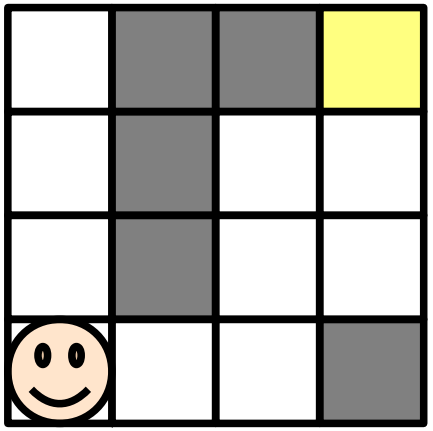


Probleme du labyrinthe



- Enumerer les solutions
- Combien de solutions différentes ?

Probleme du labyrinthe



- Enumerer les solutions
- Combien de solutions différentes ?

- Nombre de solutions $|A|^{|S|}$
  - Ici => 4^16 = 4.294.967.296
  - Énumération impossible

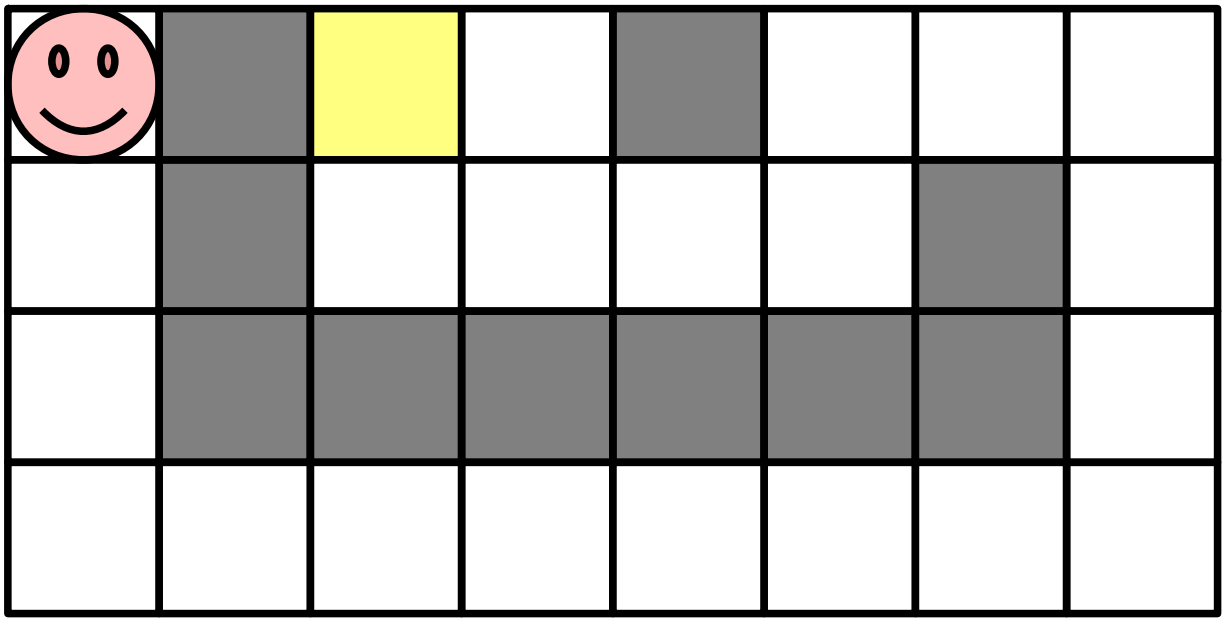
Probleme du labyrinthe

- Evaluer une politique
  - Exactement meme code qu'avant

- Meilleure politique dépend
  - Compromis entre récompense trou (-5)
  - Distance parcourue (-1 case)
  - Récompense objectif (100)

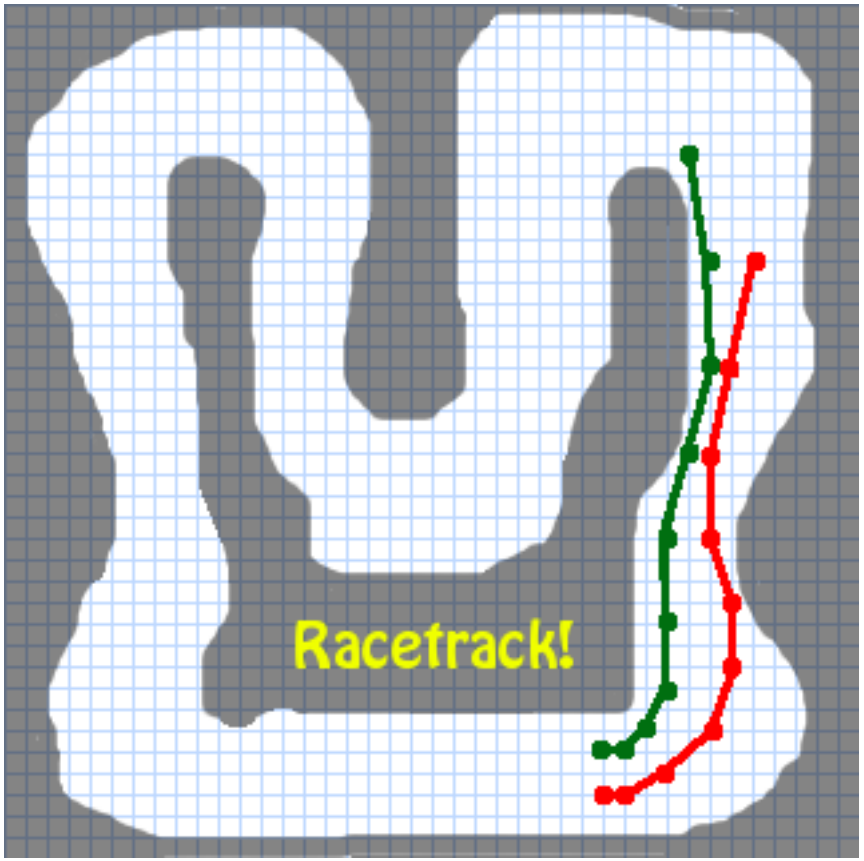
Probleme du labyrinthe

- Meilleure politique dépend
  - Compromis entre récompense trou (-5)
  - Distance parcourue (-1 case)
  - Récompense objectif (100)



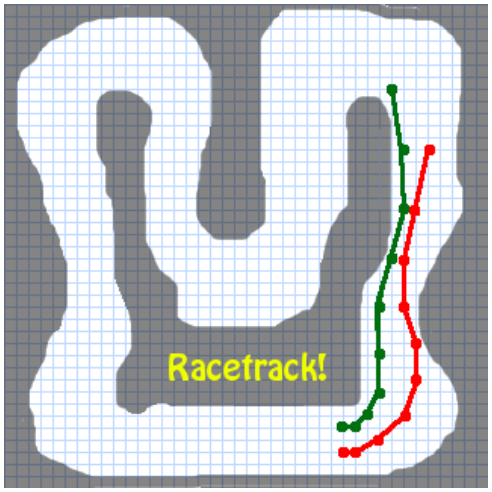
Modélisation de problemes

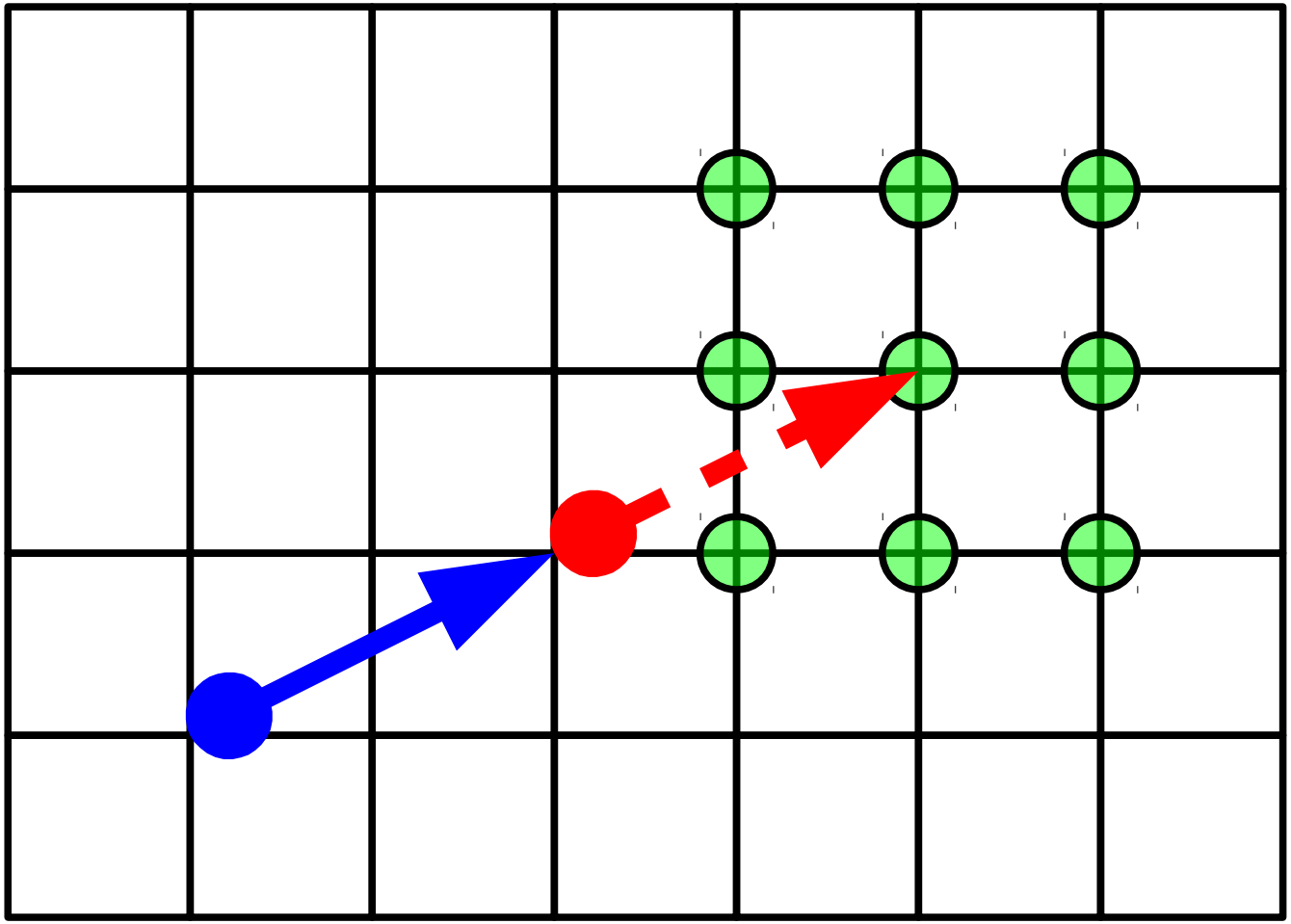
- Probleme RaceTrack (wikipedia)



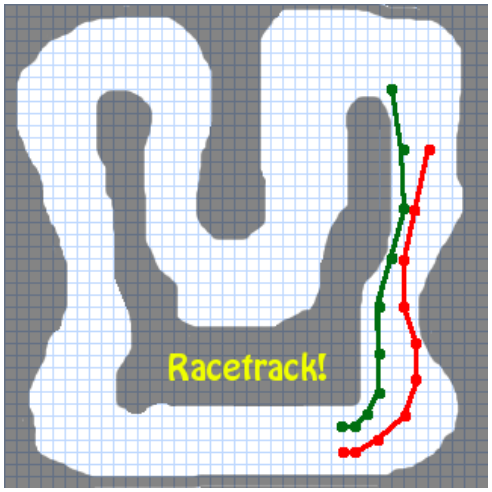
Modélisation de problemes

- Probleme RaceTrack

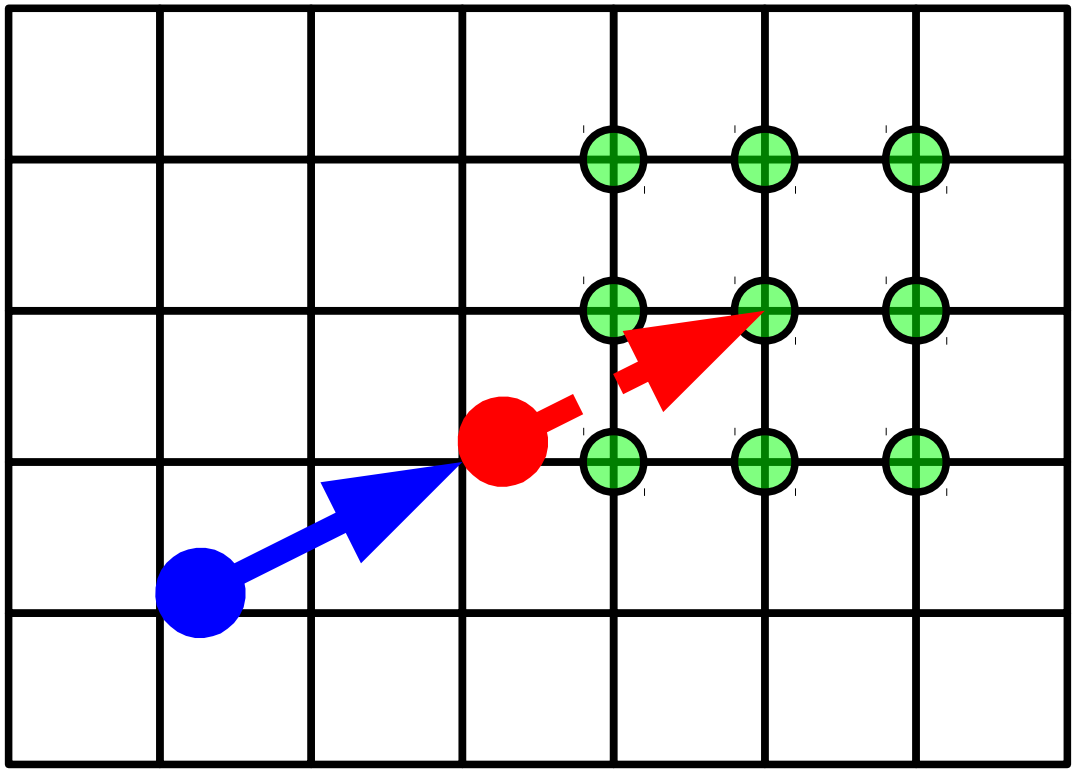




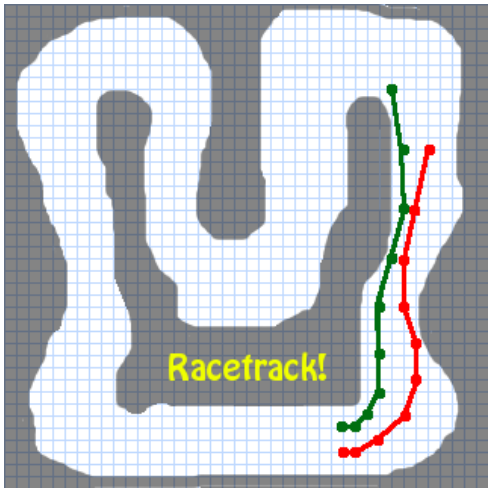
Modélisation de problemes



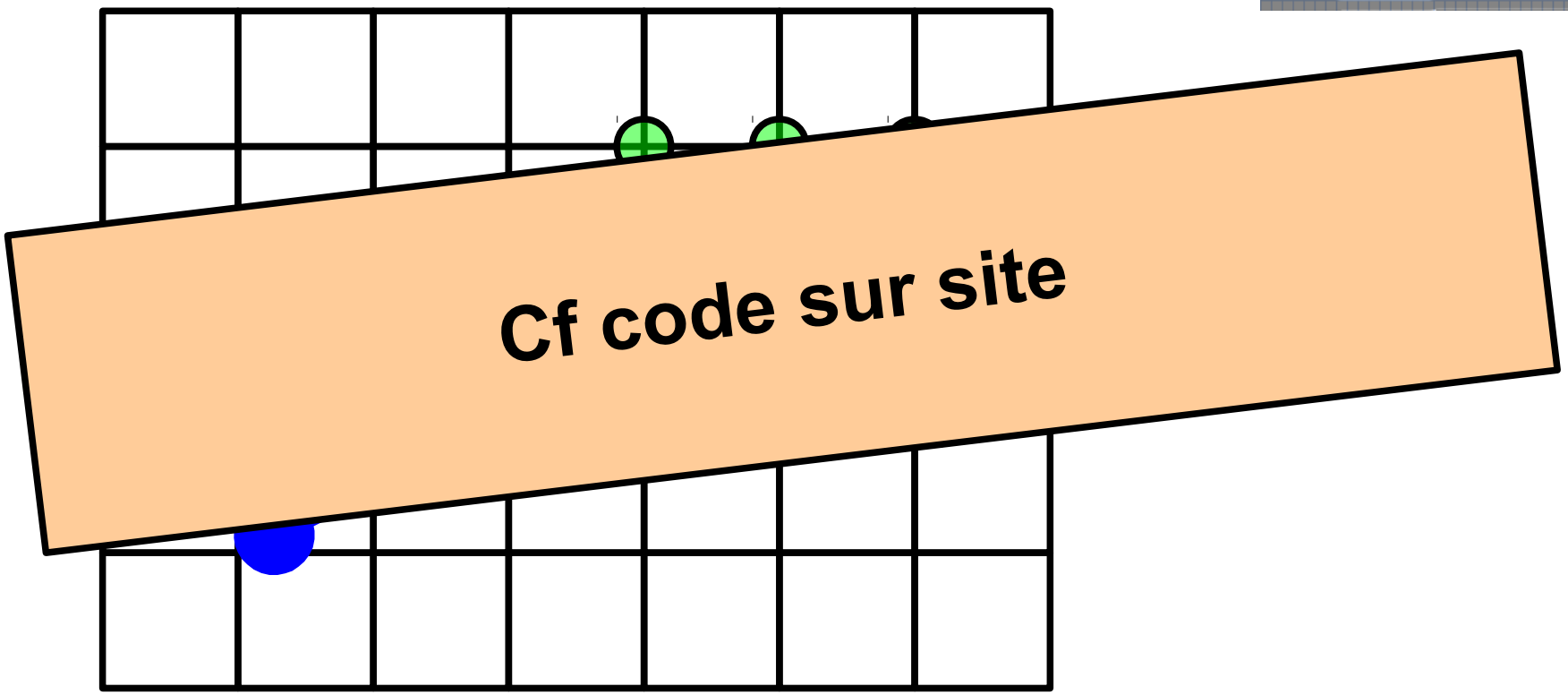
- Probleme RaceTrack
  - Etat ?
  - Action ?



Modélisation de problemes



- Probleme RaceTrack
  - Etat ?
  - Action ?



Bilan

- UNE manière de représenter des pb
  - Prise de décision séquentiel
  - Suite de décisions
- Permet de représenter de trés nombreux pb
  - Cadre MDP (deterministe)

Plan

- Conditionnement opérant
- Problème de prise de décision séquentiel
- Monde deterministe
  - Equation de Bellman
  - Planification en environnement connu
  - Apprentissage en environnement inconnu
- Monde stochastique
- Problèmes ouverts

Equation de bellman

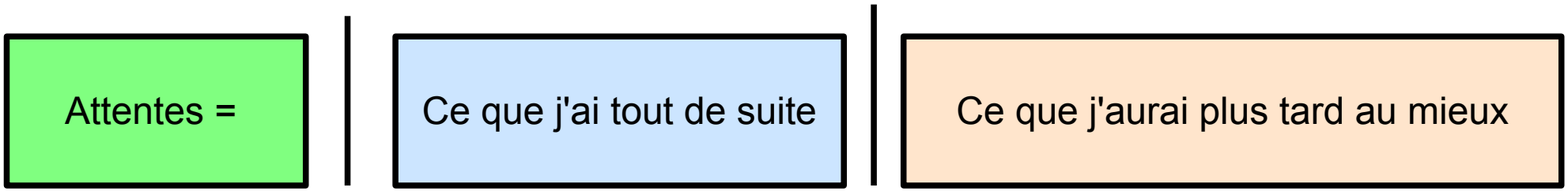
- Principe d'optimalité
  - Propriété que vérifie la politique
  - Fonction de valeur optimale Q*(s,a)
    - Ce que je peux attendre en partant de s en faisant a
    - Puis en suivant la meilleure politique

- Equation optimalité de bellman

Equation de bellman

- Principe d'optimalité
  - Propriété que vérifie la politique
  - Fonction de valeur optimale Q*(s,a)
    - Ce que je peux attendre en partant de s en faisant a
    - Puis en suivant la meilleure politique

- Equation optimalité de bellman



Equation de bellman

- Principe d'optimalité
  - Propriété que vérifie la politique
  - Fonction de valeur optimale Q*(s,a)
    - Ce que je peux attendre en partant de s en faisant a
    - Puis en suivant la meilleure politique

- Equation optimalité de bellman

$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

Attentes =	Ce que j'ai tout de suite	Ce que j'aurai plus tard au mieux

Facteur actuation

- Probleme en temps infini
  - Somme diverge

- Facteur actuation gamma < 1
  - Ratio aujourd'hui / demain

$$Q^*(s,a) = R(s,a,T(s,a)) + \gamma . max_{a'} Q^*(T(s,a),a')$$

Plan

- Conditionnement opérant
- Problème de prise de décision séquentiel
- Monde deterministe
  - Equation de Bellman
  - Planification en environnement connu
  - Apprentissage en environnement inconnu
- Monde stochastique
- Problèmes ouverts

Planification (env connu)

- Si le monde est connu
  - S, A, T, R
- Possible d'appliquer équation Bellman
  - Converge forcément (gamma < 1)
  - Suite recurente
- Algorithme value iteration

Value iteration

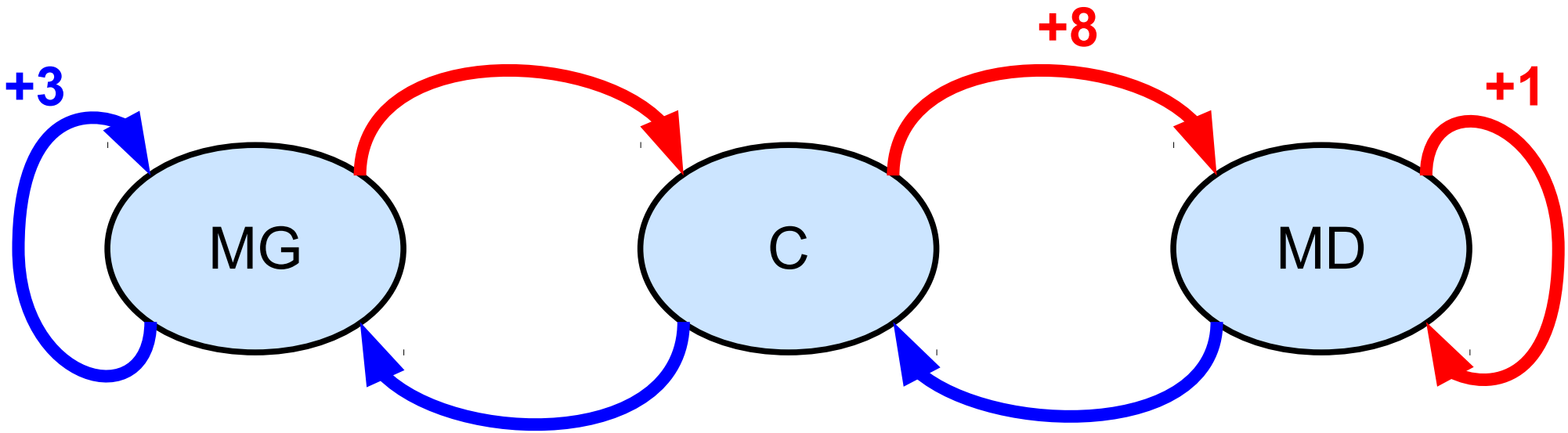
- Repeter pour t
  - Chaque état et chaque action
    - $Q^*_{t+1}(s,a) = R(s,a,T(s,a)) + \gamma . max_{a'} Q^*_t(T(s,a),a')$

- Principe
  - Initialiser avec Q*=0
  - Remonter le temps
    - Q*(t+1) dépend de la valeur de l'état d'arrivée

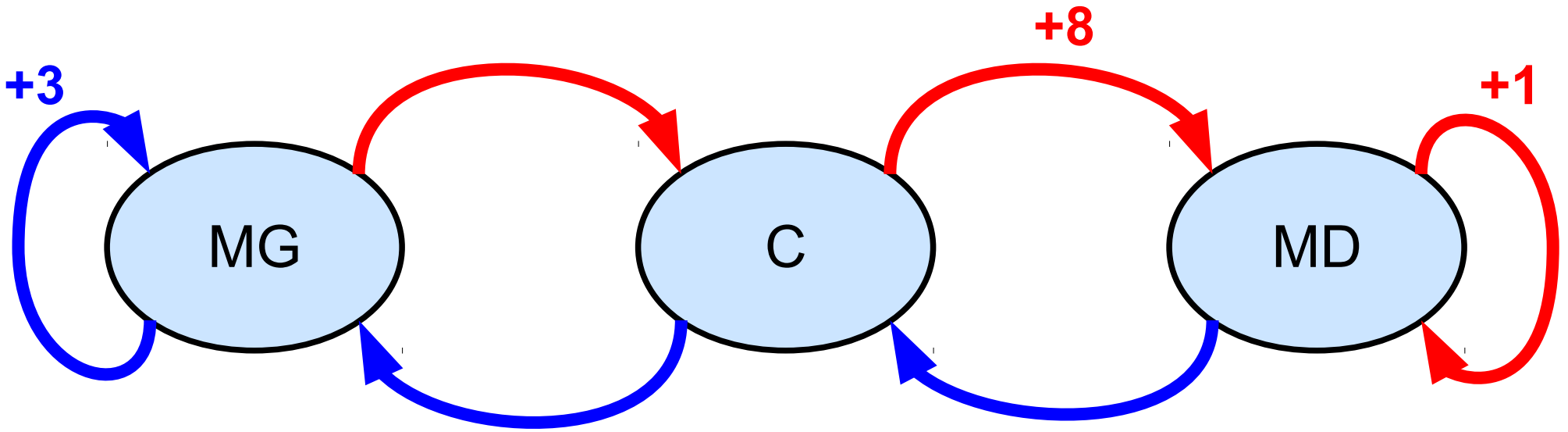
- Python : Q = dictionnaire (etat,action) -> valeur

Exemple – chercheur or

- 3 etats:
  - Chemin (C), Mine gauche/droite (MG/MD)
- 2 actions:
  - Gauche (G), droite (D)

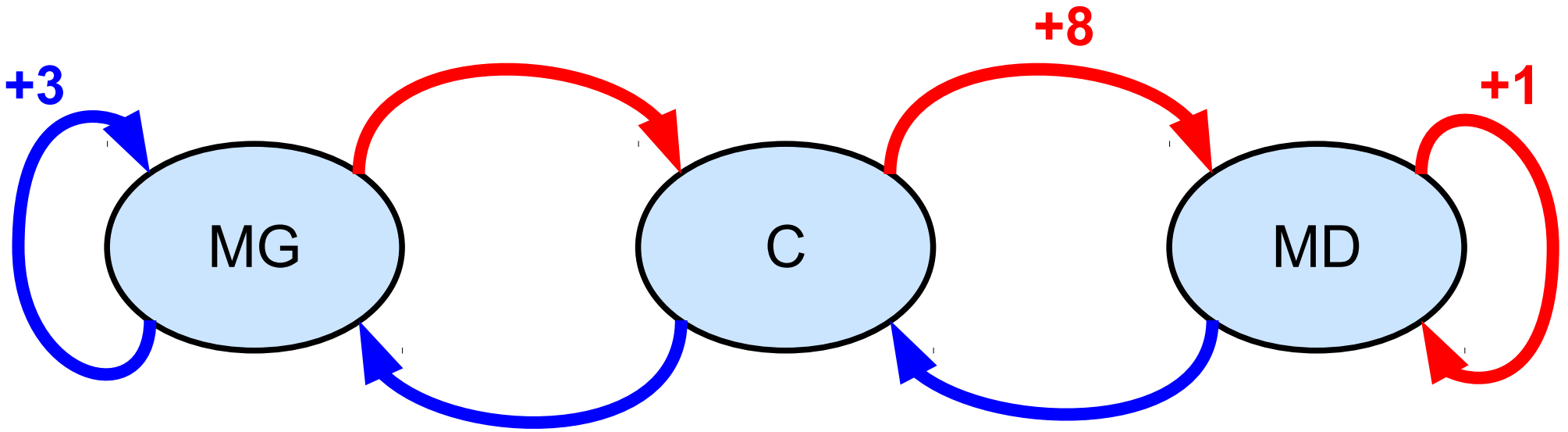


Exemple – chercheur or



- Meilleure politique ?

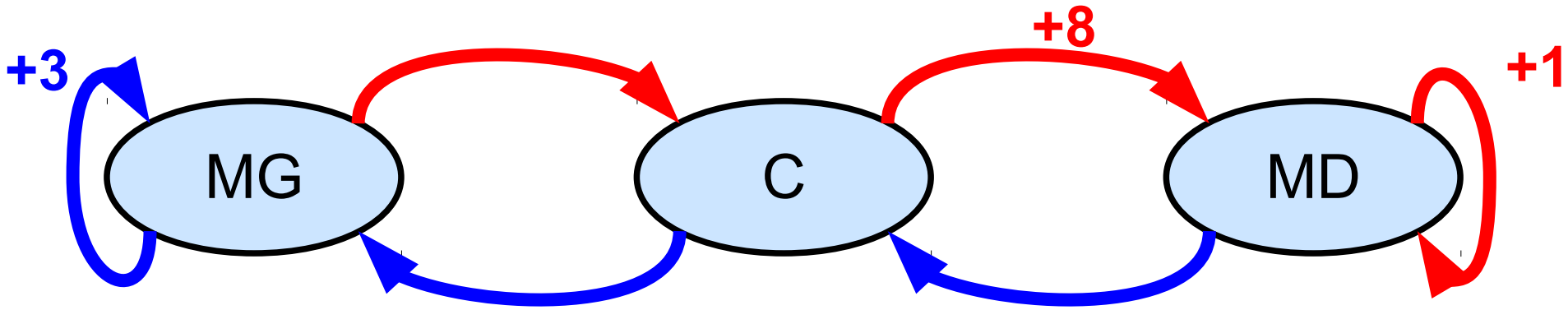
Exemple – chercheur or



- Récriture Equation Bellman

$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

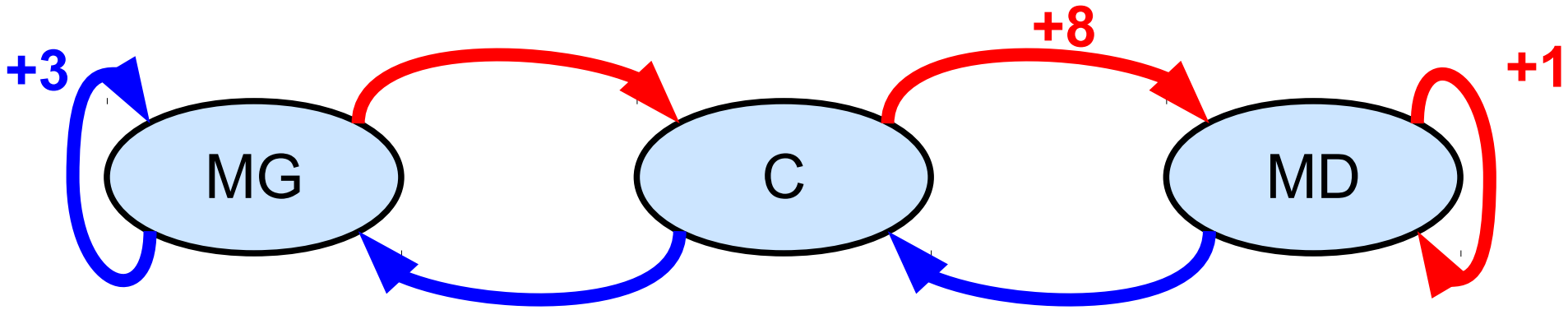
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

- 6 cas

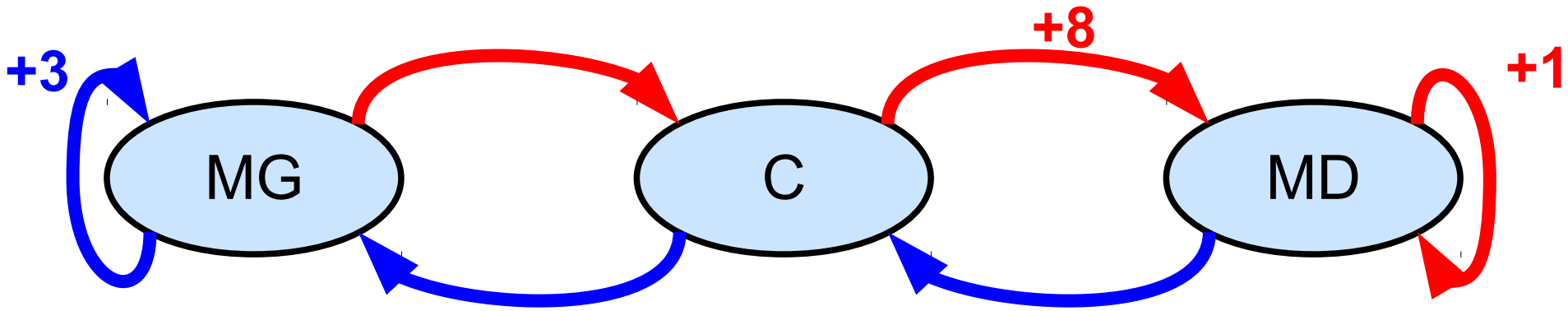
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

- 6 cas
  - MG + Gauche -> MG, +3
  - MG + Droite -> C, +0
  - C + Droite -> MD, +8
  - C + Gauche -> MG, +0
  - MD + Gauche -> C, +0
  - MD + Droite -> MD, +1

Exemple – chercheur or



$$Q^*(s,a)=R(s,a,T(s,a))+max_{a'}Q^*(T(s,a),a')$$

$$Q^*(MG,G)=R(MG,G,MG)+0{,}9.max(Q^*(MG,G),Q^*(MG,D))$$

$$Q^*(MG,D)=R(MG,D,C)+0{,}9.max(Q^*(C,G),Q^*(C,D))$$

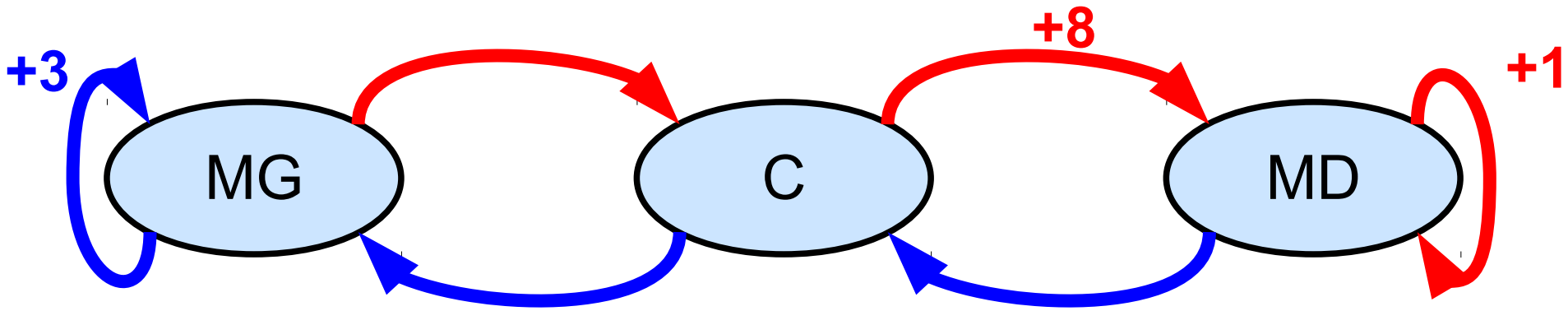
$$Q^*(C,G)=R(C,G,MG)+0{,}9.max(Q^*(MG,G),Q^*(MG,D))$$

$$Q^*(C,D)=R(C,D,MD)+0{,}9.max(Q^*(MD,G),Q^*(MD,D))$$

$$Q^*(MD,G)=R(MD,G,C)+0{,}9.max(Q^*(C,G),Q^*(C,D))$$

$$Q^*(MD,D)=R(MD,D,MD)+0{,}9.max(Q^*(MD,G),Q^*(MD,D))$$

Exemple – chercheur or



$$Q^*(s,a)=R(s,a,T(s,a))+max_{a'}Q^*(T(s,a),a')$$

$$Q^*(MG,G)=3+0{,}9.max(Q^*(MG,G),Q^*(MG,D))$$

$$Q^*(MG,D)=0+0{,}9.max(Q^*(C,G),Q^*(C,D))$$

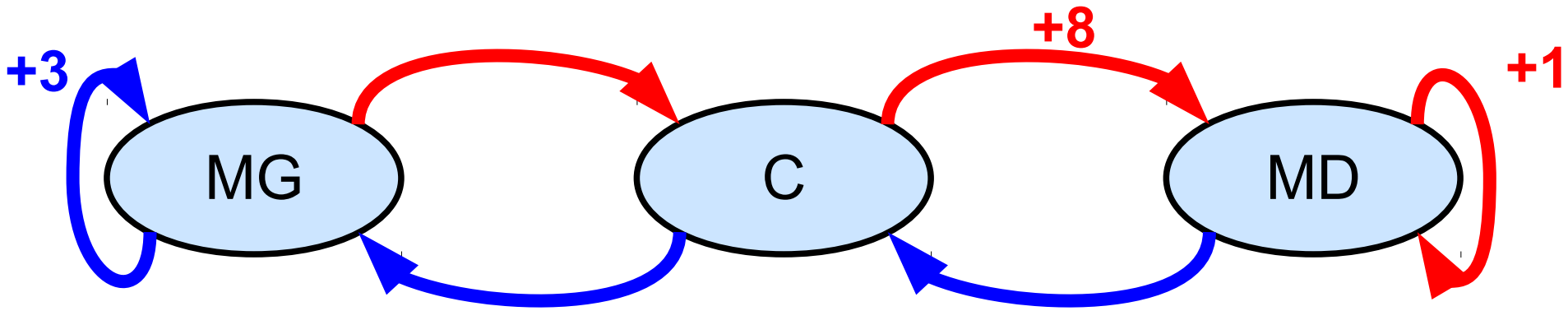
$$Q^*(C,G)=0+0{,}9.max(Q^*(MG,G),Q^*(MG,D))$$

$$Q^*(C,D)=8+0{,}9.max(Q^*(MD,G),Q^*(MD,D))$$

$$Q^*(MD,G)=0+0{,}9.max(Q^*(C,G),Q^*(C,D))$$

$$Q^*(MD,D)=1+0{,}9.max(Q^*(MD,G),Q^*(MD,D))$$

Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

$$Q^*(MG,G) = 3 + 0{,}9.max(Q^*(MG,G), Q^*(MG,D))$$

$$Q^*(MG,D) = 0{,}9.max(Q^*(C,G), Q^*(C,D))$$

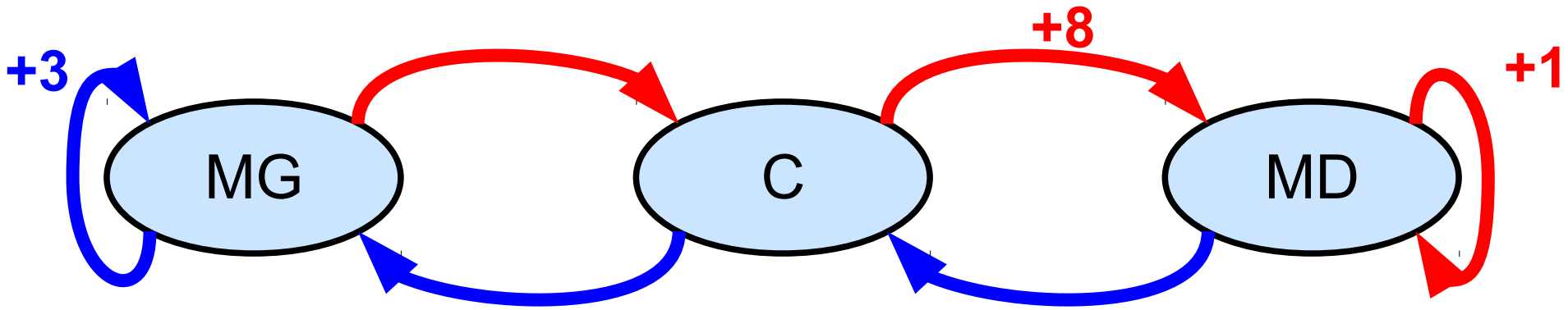
$$Q^*(C,G) = 0{,}9.max(Q^*(MG,G), Q^*(MG,D))$$

$$Q^*(C,D) = 8 + 0{,}9.max(Q^*(MD,G), Q^*(MD,D))$$

$$Q^*(MD,G) = 0{,}9.max(Q^*(C,G), Q^*(C,D))$$

$$Q^*(MD,D) = 1 + 0{,}9.max(Q^*(MD,G), Q^*(MD,D))$$

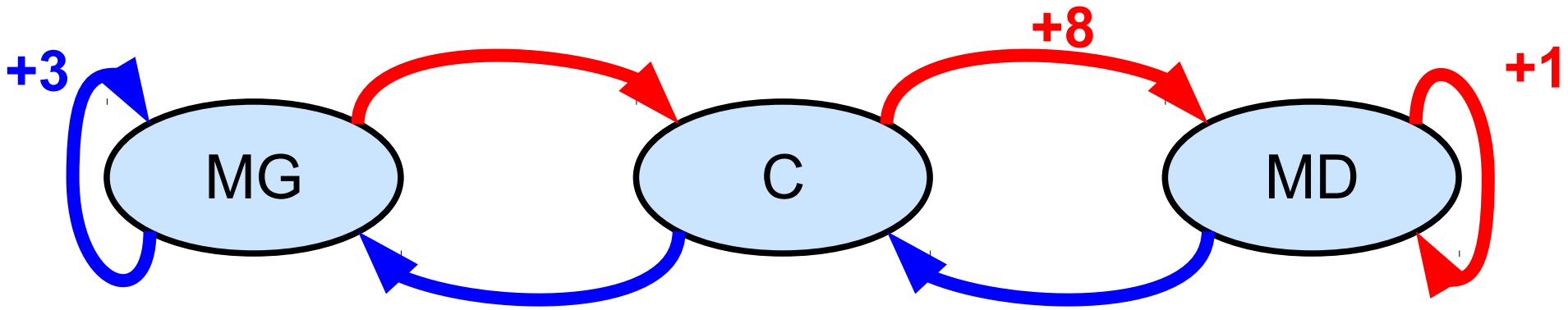
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0
MG, G	
MG, D	
C, G	
C, D	
MD, G	
MD, D	

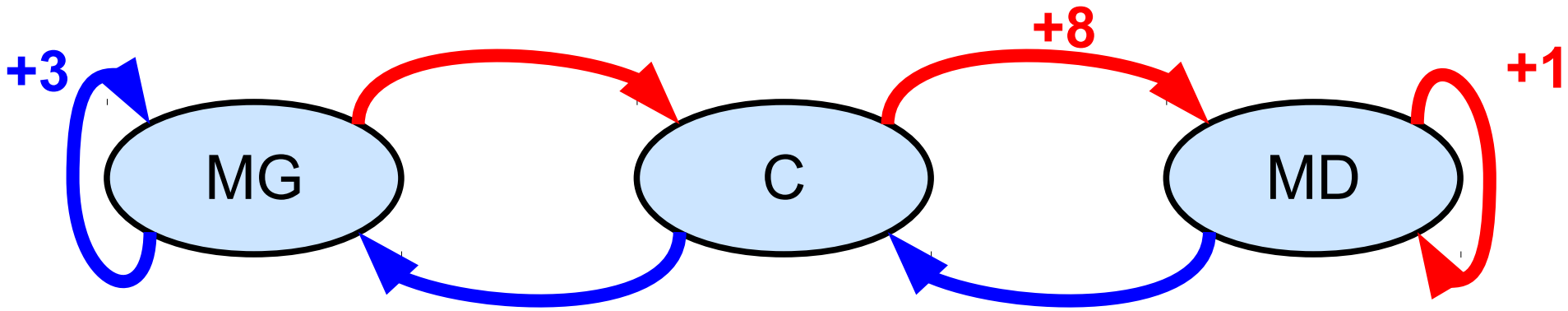
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0
MG, G	0
MG, D	0
C, G	0
C, D	0
MD, G	0
MD, D	0

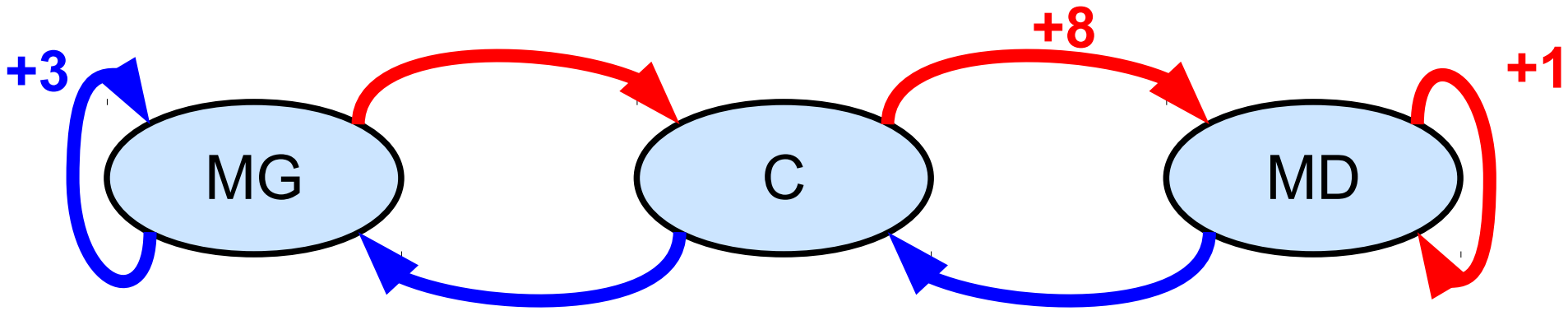
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1
MG, G	0	
MG, D	0	
C, G	0	
C, D	0	
MD, G	0	
MD, D	0	

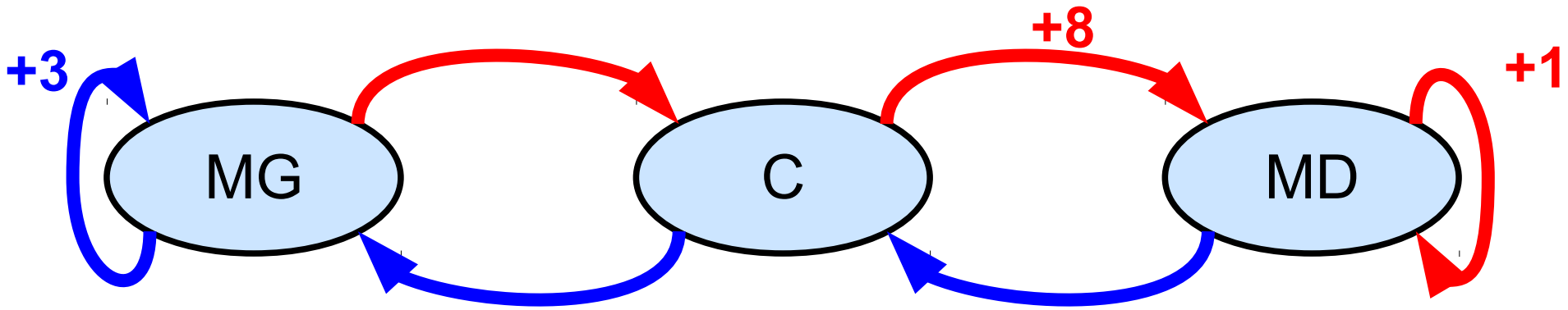
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1
MG, G	0	= 3 +0,9 max Q(MG,G), Q(MG,D)
MG, D	0	
C, G	0	
C, D	0	
MD, G	0	
MD, D	0	

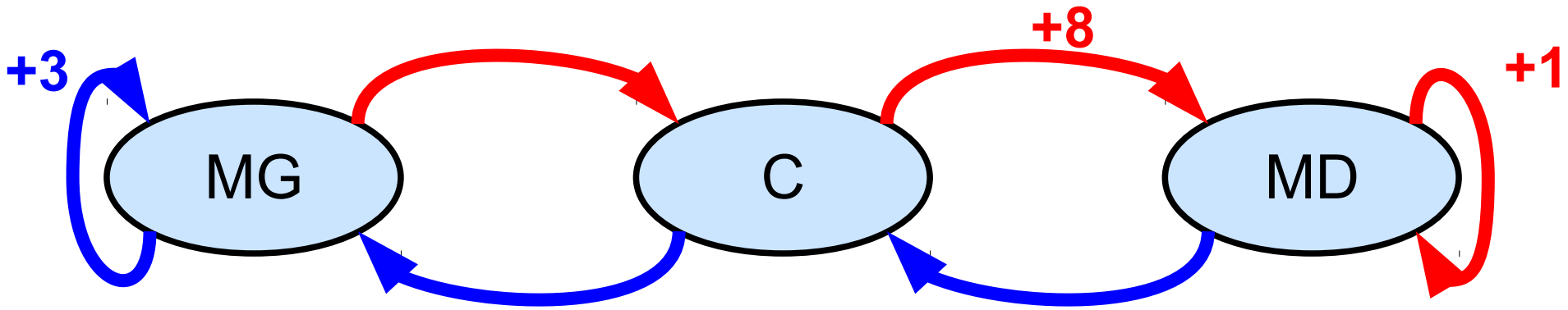
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1
MG, G	0	= 3
MG, D	0	
C, G	0	
C, D	0	
MD, G	0	
MD, D	0	

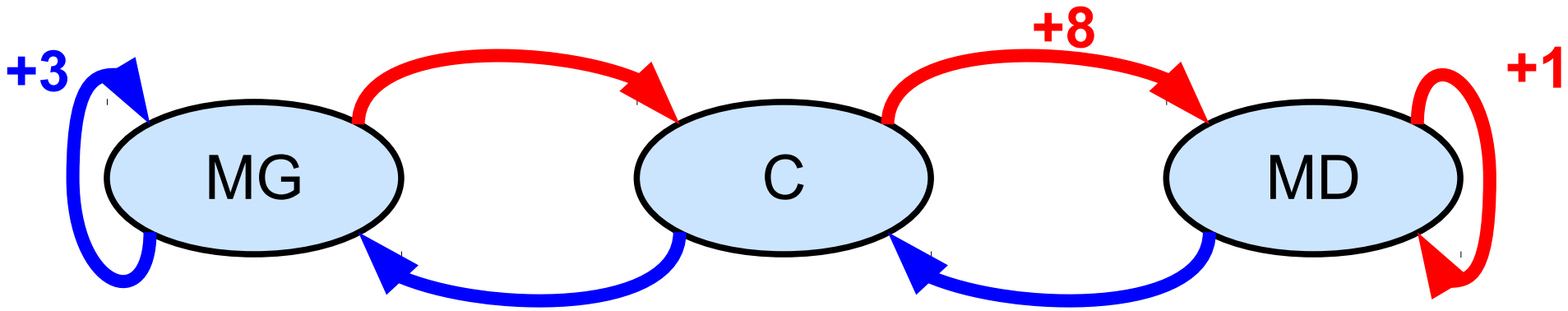
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1
MG, G	0	= 3
MG, D	0	= 0 + 0,9 max (Q (C,G), Q(C,D)
C, G	0	
C, D	0	
MD, G	0	
MD, D	0	

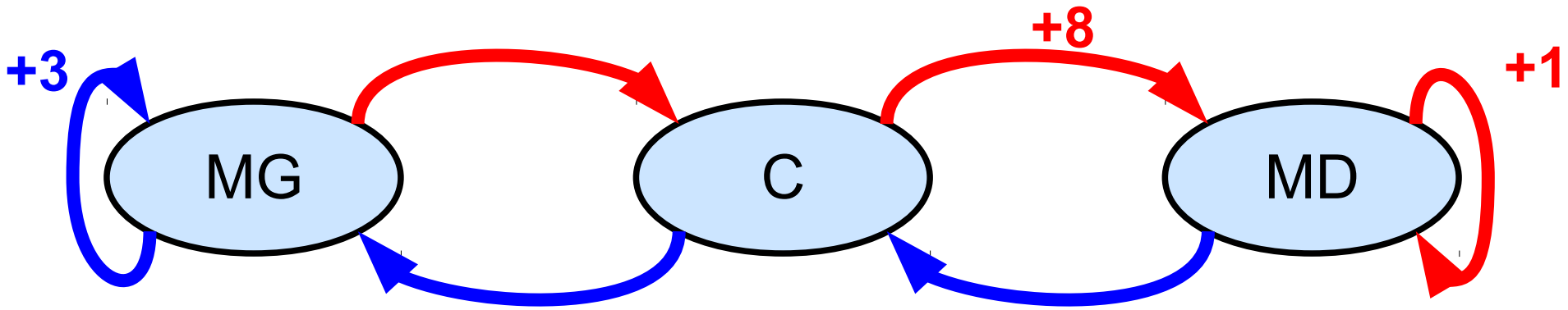
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1
MG, G	0	= 3
MG, D	0	= 0
C, G	0	= 0 + max (Q(MD, G) Q(MD,D))
C, D	0	= 8 + max (Q(MD,G), Q(MD,G))
MD, G	0	= 0 + max (Q(C,G), Q(C,D))
MD, D	0	= 1 + max (Q(MD,G),Q(MD,D))

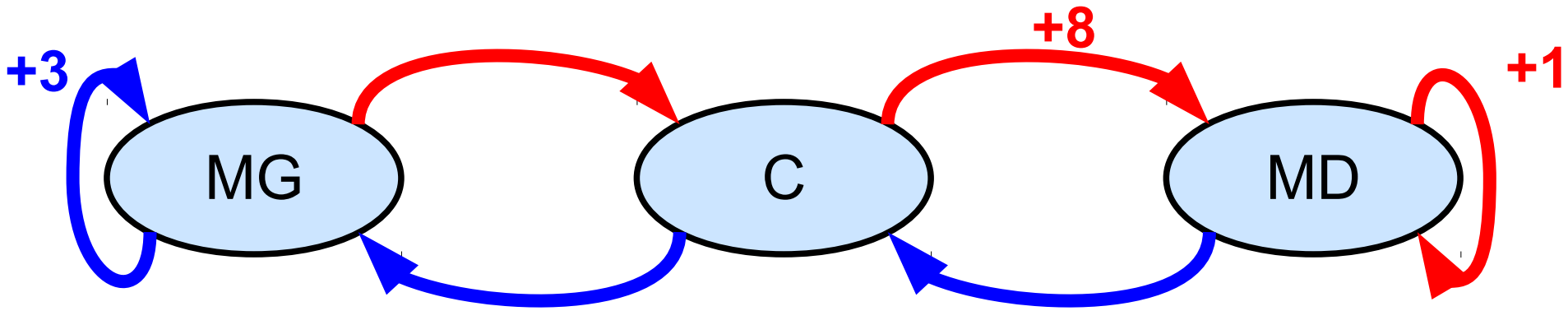
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1
MG, G	0	= 3
MG, D	0	= 0
C, G	0	= 0
C, D	0	= 8
MD, G	0	= 0
MD, D	0	= 1

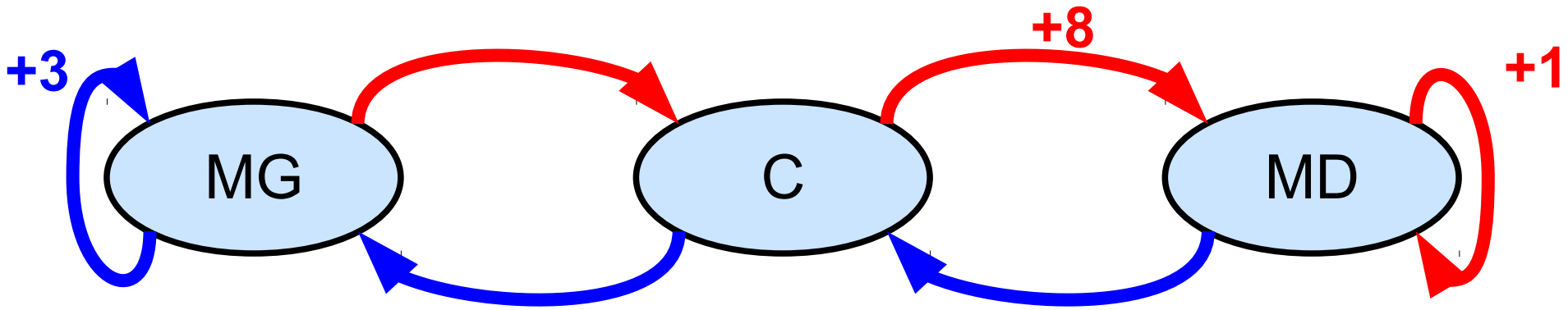
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1
MG, G	0	3
MG, D	0	0
C, G	0	0
C, D	0	8
MD, G	0	0
MD, D	0	1

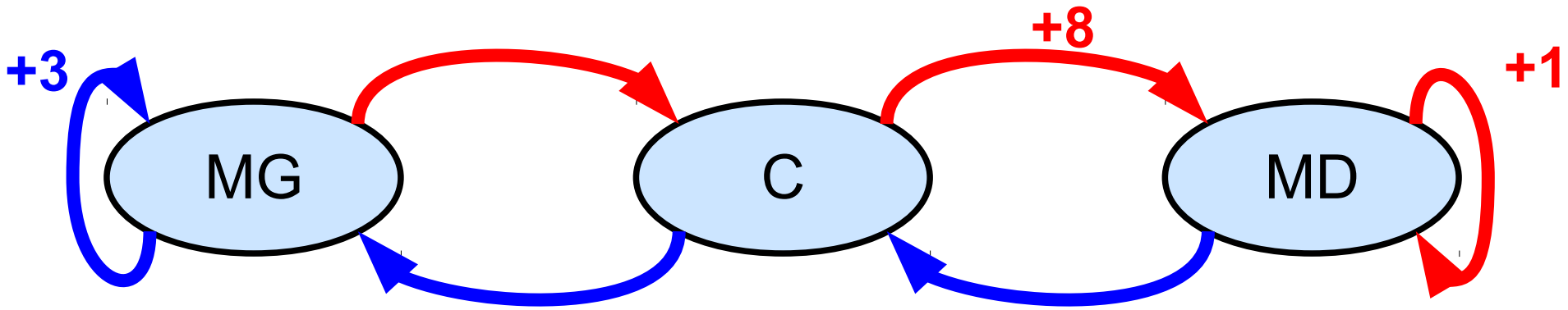
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1		t=2
MG, G	0	3	MG	= 3 + 0,9 . max Q(MG)
MG, D	0	0		= 0 + 0,9 . max (C)
C, G	0	0	C	= 0 + 0,9 . max (MG)
C, D	0	8		= 8 + 0,9 . max (MD)
MD, G	0	0	MD	= 0 + 0,9 . max (C)
MD, D	0	1		= 1 + 0,9 . max (MD)

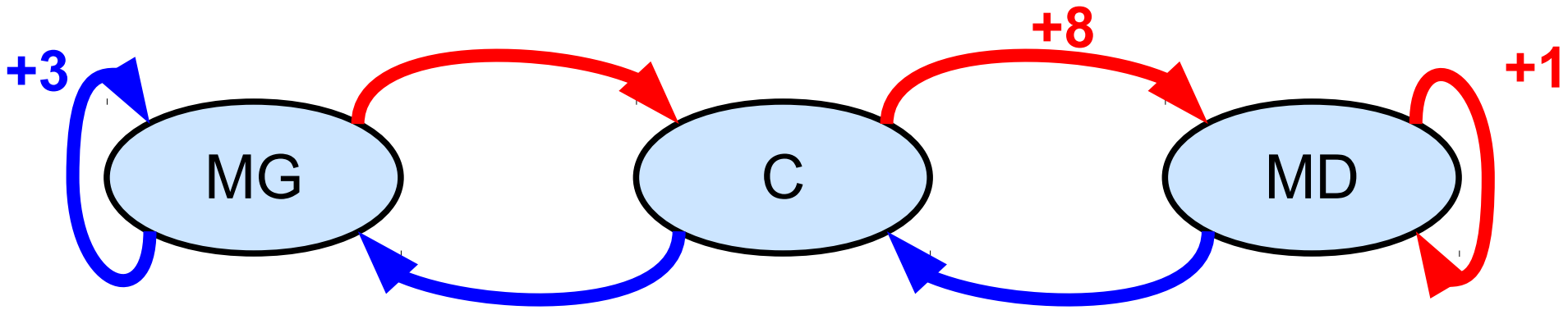
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1		t=2
MG, G	0	3	MG	= 3 + 0,9 * 3
MG, D	0	0		= 0 + 0,9 *8
C, G	0	0	C	= 0 + 0,9 * 3
C, D	0	8		= 8 + 0,9 * 1
MD, G	0	0	MD	= 0 + 0,9 * 8
MD, D	0	1		= 1 + 0,9 * 1

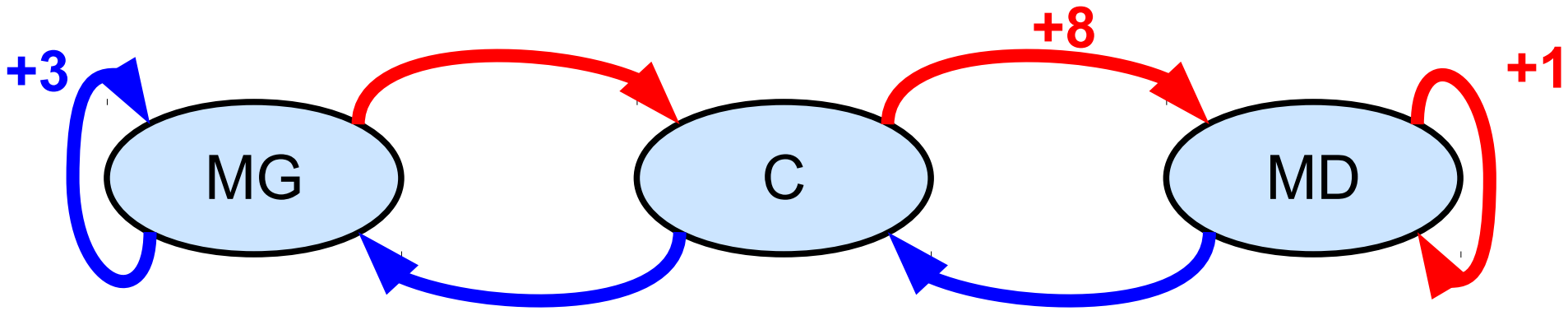
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1		t=2
MG, G	0	3	MG	= 5,7
MG, D	0	0		= 7,2
C, G	0	0	C	= 2,7
C, D	0	8		= 8,9
MD, G	0	0	MD	= 7,2
MD, D	0	1		= 1,9

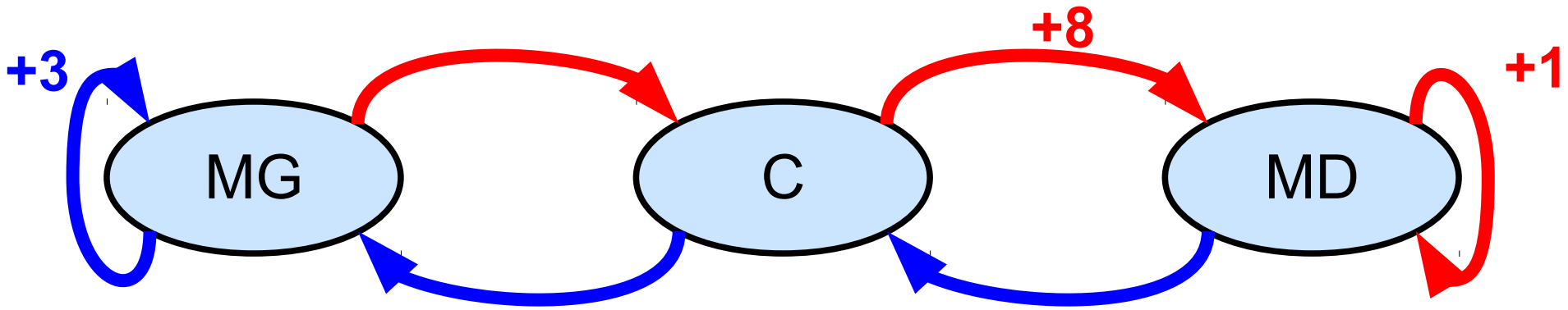
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1	t=2	
MG, G	0	3	5,7	MG
MG, D	0	0	7,2	
C, G	0	0	2,7	C
C, D	0	8	8,9	
MD, G	0	0	7,2	MD
MD, D	0	1	1,9	

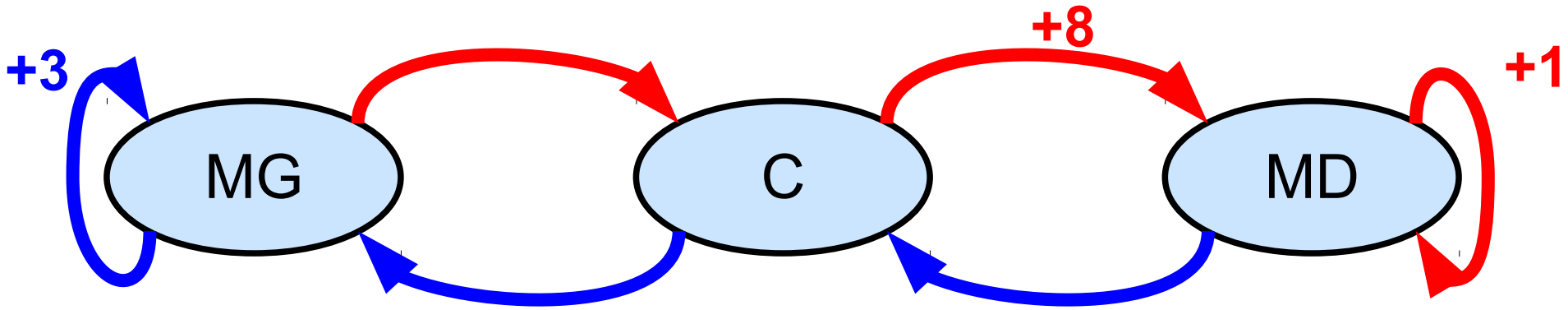
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1	t=2		t=3
MG, G	0	3	5,7	MG	3 + 0,9 * 7,2
MG, D	0	0	7,2		0 + 0,9 * 8,9
C, G	0	0	2,7	C	0 + 0,9 * 7,2
C, D	0	8	8,9		8 + 0,9 * 7,2
MD, G	0	0	7,2	MD	0 + 0,9 * 8,9
MD, D	0	1	1,9		1 + 0,9 * 7,2

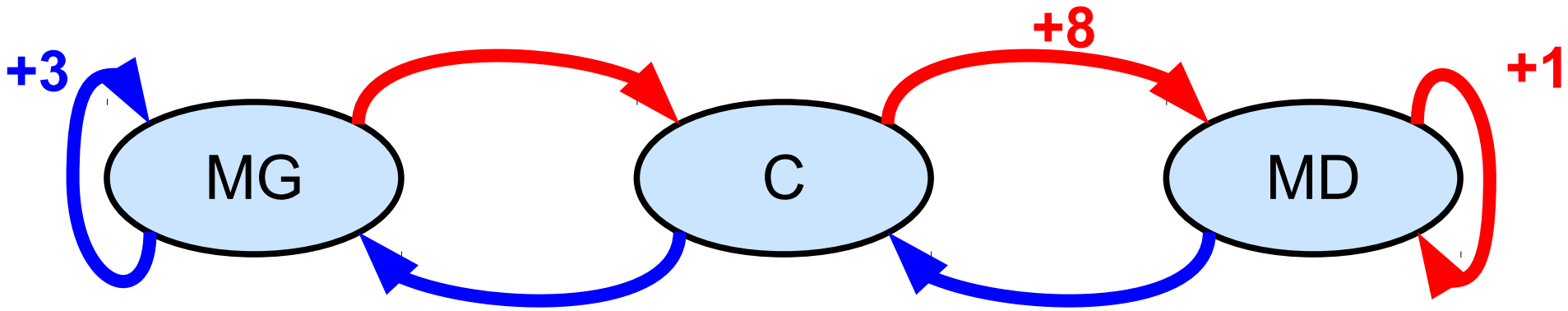
Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1	t=2		t=3
MG, G	0	3	5,7	MG	9,48
MG, D	0	0	7,2		8,01
C, G	0	0	2,7	C	6,48
C, D	0	8	8,9		14,48
MD, G	0	0	7,2	MD	8,01
MD, D	0	1	1,9		7,48

Exemple – chercheur or



$$Q^*(s,a) = R(s,a,T(s,a)) + max_{a'} Q^*(T(s,a),a')$$

	t=0	t=1	t=2	t=3		t=10		t=100
MG, G	0	3	5,7	9,48		23.18..		37.10...
MG, D	0	0	7,2	8,01		24.68..		37.89...
C, G	0	0	2,7	6,48		20.18..		34.10...
C, D	0	8	8,9	14,48		27.81..		42.10...
MD, G	0	0	7,2	8,01		24.68..		37.89...
MD, D	0	1	1,9	7,48		20.81..		35.10...

Code python

- Valider value iteration

- Avec le probleme donné
  - Faire t itérations
    - Pour chaque état, chaque action
    - $Q^*_{t+1}(s,a) = R(s,a,T(s,a)) + \gamma . max_{a'} Q^*_t(T(s,a),a')$
  - Retourner dictionnaire

ValueIteration - commentaires


```
def valueIteration(self,nb,gamma):
      # initialiser Qvaleurs
      # faire nb iterations
            # executer une Maj de Q
      # retourner Q
```

```
def executerUneIteration(self,Q,gamma):
      # initialiser Q2
      # pour chaque etat
            # pour chaque action
                  # calculer arrivee et recompense
                  # cherche max arrivee
                  # Mise à jour Q2
                  Q2[(etat,action)]=r+gamma*max
      # retourne resultat Q2
```

```
def calculerMax(self,Q,sArriv):
      # pour chaque action
            # si c'est mieux que max, stocker max
      # retourner max
```

ValueIteration


```
def valueIteration(self,nb,gamma):
        Q=self.initialiserQ()
        #nb iteration iteration
        for i in range(0,nb):
            Q=self.executerUneIteration(Q,gamma)
        return(Q)
```

```
def initialiserQ(self):
        Q={}
        for s in self.pb.etats():
            for a in self.pb.actions():
                Q[(s,a)]=0
        return Q
```

ValueIteration


```
def executerUneIteration(self,Q,gamma):
        Q2={}
        #pour chaque etat,action
        for etat in self.pb.etats():
            for action in self.pb.actions():
                #calculer arrivee
                sArriv=self.pb.transition(etat,action)
                r=self.pb.recompense(etat,action,sArriv)
                # cherche max arrivee
                max=self.calculerMax(Q,sArriv)
                #mise à jour
                Q2[(etat,action)]=r+gamma*max
         #retourne resultat
        return(Q2)
```

```
def calculerMax(self,Q,sArriv):
        max=-100000
        for actionMax in self.pb.actions():
            if (Q[(sArriv,actionMax)]>max):
                max=Q[(sArriv,actionMax)]
        return(max)
```

```
def valueIteration(self,nb,gamma):
        Q={}
        for s in self.pb.etats():
            for a in self.pb.actions():
                Q[(s,a)]=0

        #une iteration
        for i in range(0,nb):
            Q2={}
            #pour chaque etat,action
            for etat in self.pb.etats():
                for action in self.pb.actions():
                    #calculer arrivee
                    sArriv=self.pb.transition(etat,action)
                    r=self.pb.recompense(etat,action,sArriv)
                    # cherche max arrivee
                    max=-100000
                    for actionMax in self.pb.actions():
                        if (Q[(sArriv,actionMax)]>max):
                            max=Q[(sArriv,actionMax)]

                    #mise à jour
                    Q2[(etat,action)]=r+gamma*max

            #on augmente iteration
            Q=Q2
        return(Q)
```

Politique optimale

- Politique optimale
  - Choisir action qui maximise Q*
  - Argmax (Q(s,a))

Propagation dans labyrinthe

- Exemple labyrinthe
  - Appliquer l'algorithme fourni

- Montrer application

Propagation dans labyrinthe

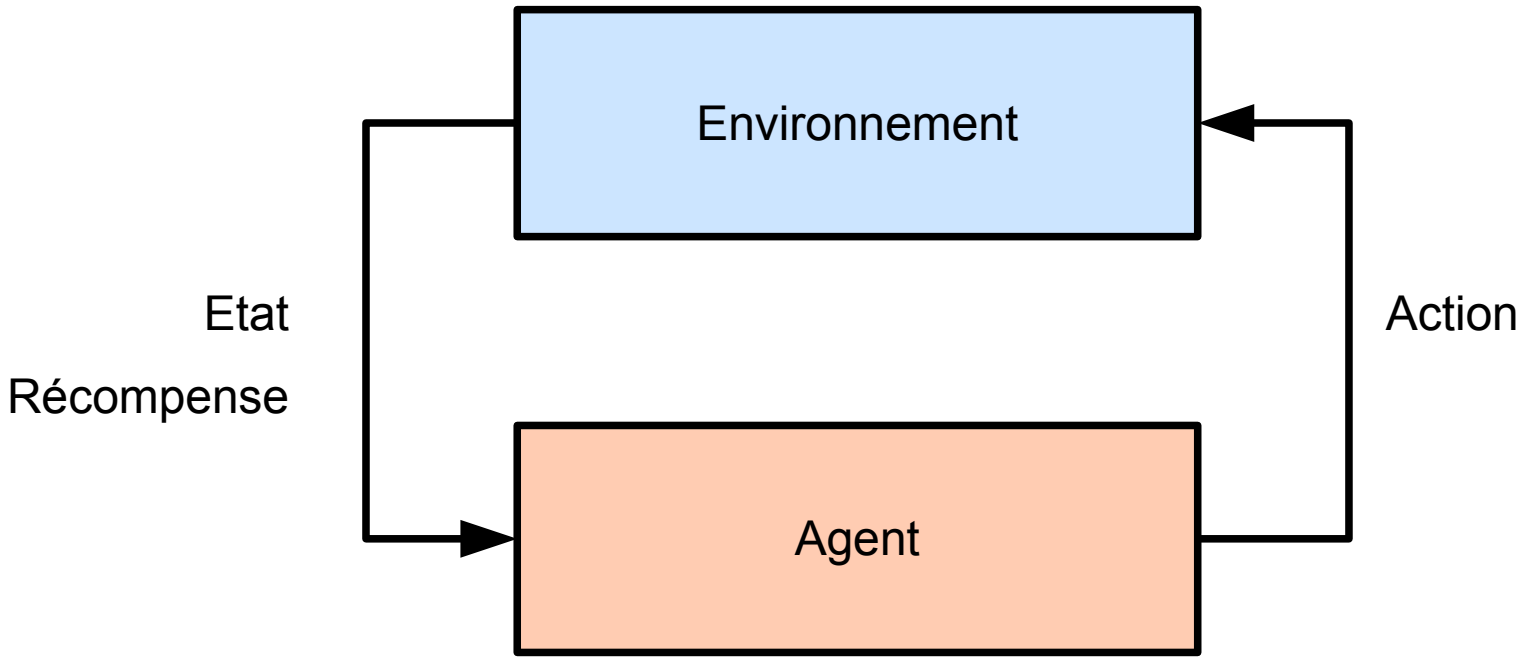
- Exemple labyrinthe
  - Appliquer l'algorithme fourni
- Montrer application
- Probleme de taille de l'espace d'état
  - Idée : ne construire Qval que états rencontrés
  - Exemple RaceTracker

Plan

- Conditionnement opérant
- Problème de prise de décision séquentiel
- Monde deterministe
  - Equation de Bellman
  - Planification en environnement connu
  - Apprentissage en environnement inconnu
- Monde stochastique
- Problèmes ouverts

Apprentissage par renforcement

- Apprentissage par renforcement
  - Probleme de décision
  - Monde inconnu (T, R), mais experiences possibles



Apprentissage par renforcement

- Apprentissage par renforcement
  - Probleme Monde inconnu
  - Connait pas T, R

- A chaque pas de temps
  - (s, a) -> s' et r
  - Mettre à jour Q(s,a) à partir informations

- Exemple
  - (C, gauche) => (MG, 0) ; (MG, D) => (C, 0) ; ...

Apprentissage par renforcement

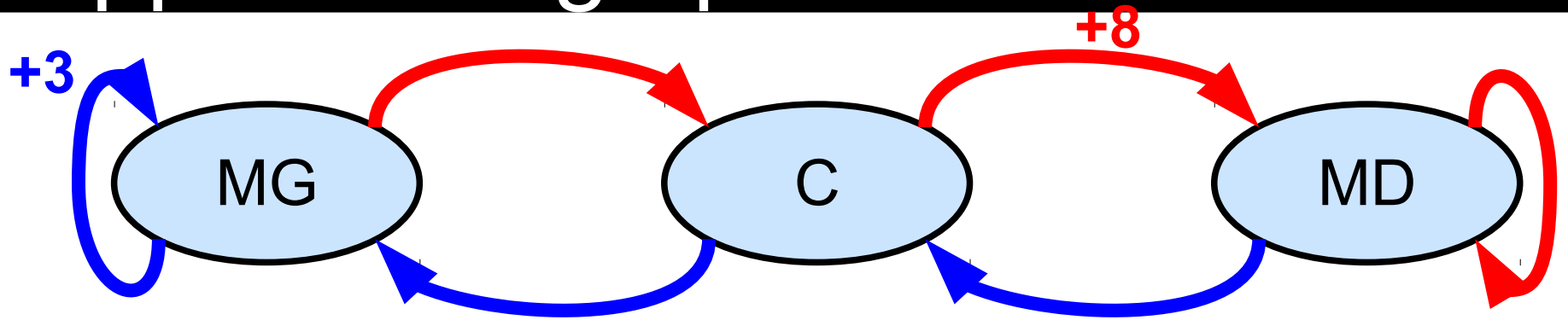
- Objectif:
  - Apprendre au fur et à mesure à partir Xp

- Principe Q-learning :
  - Apprendre directement politique (AR direct)
  - Pour tuple s, a => s', r
  - On applique équation Bellman

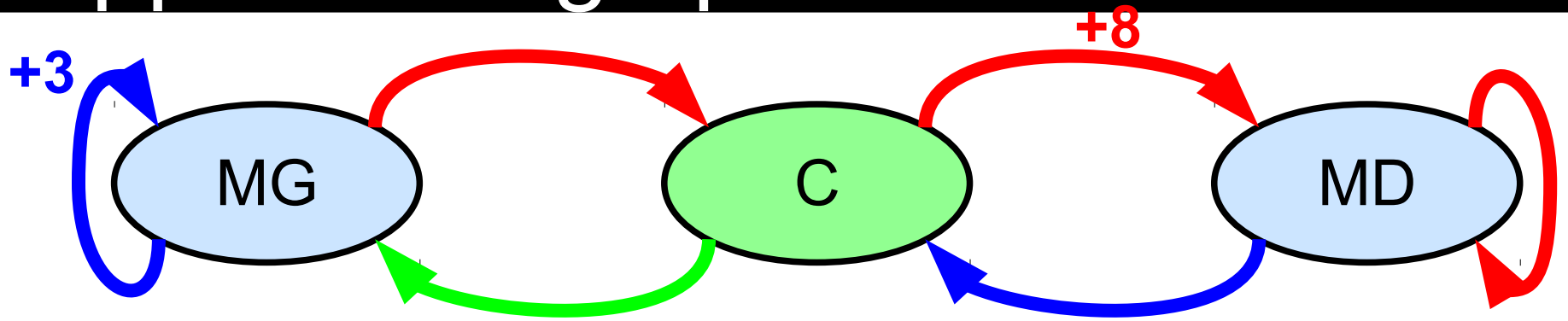
$$Q^*(s,a) = R(s,a,T(s,a)) + \gamma \, max_{a'} Q^*(T(s,a),a')$$

$$Q^*(s,a) = r + \gamma \, max_{a'} Q^*(s',a')$$

Apprentissage par renforcement



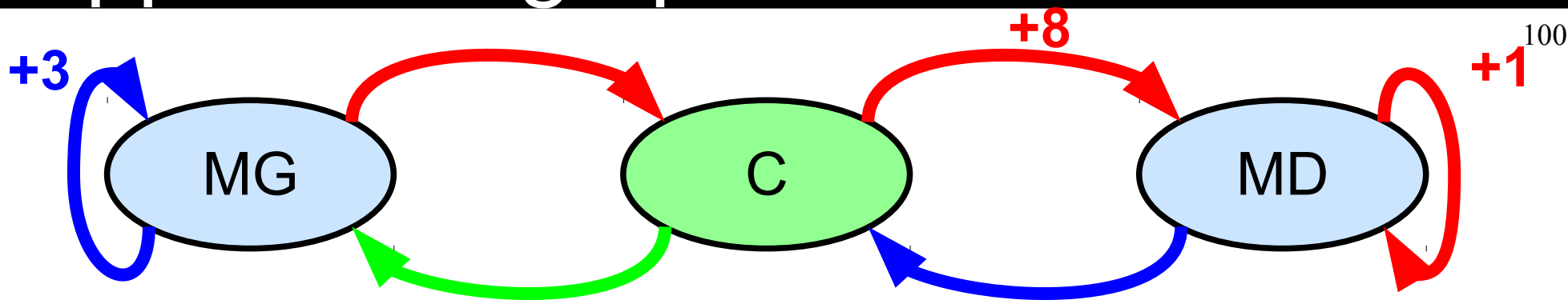
Apprentissage par renforcement



- C, Gauche ==> MG , +0

$$Q^*(C,G) = r + \gamma \, max_{a'} Q^*(MG, a')$$

Apprentissage par renforcement

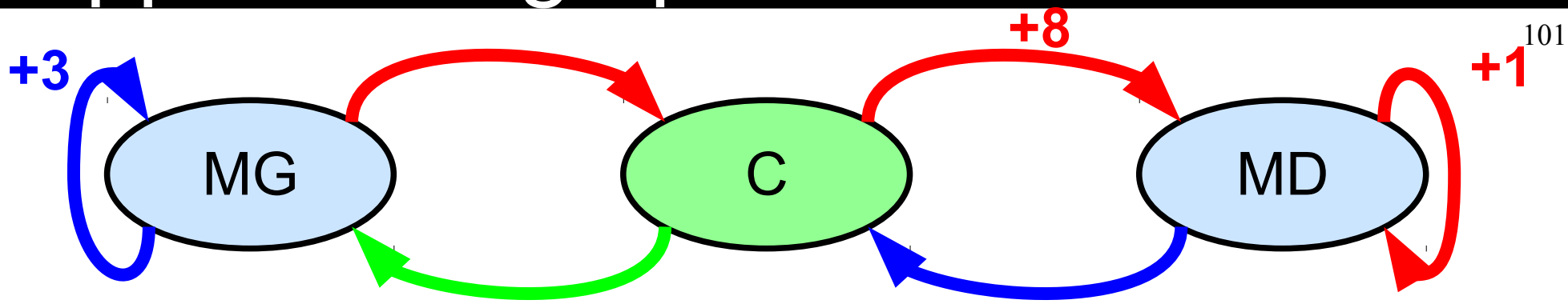


- C, Gauche ==> MG , +0

$$Q^*(C,G) = r + \gamma \, max_{a'} Q^*(MG,a')$$

	t=0	t=1
MG, G	0	0
MG, D	0	0
C, G	0	0
C, D	0	0
MD, G	0	0
MD, D	0	0

Apprentissage par renforcement

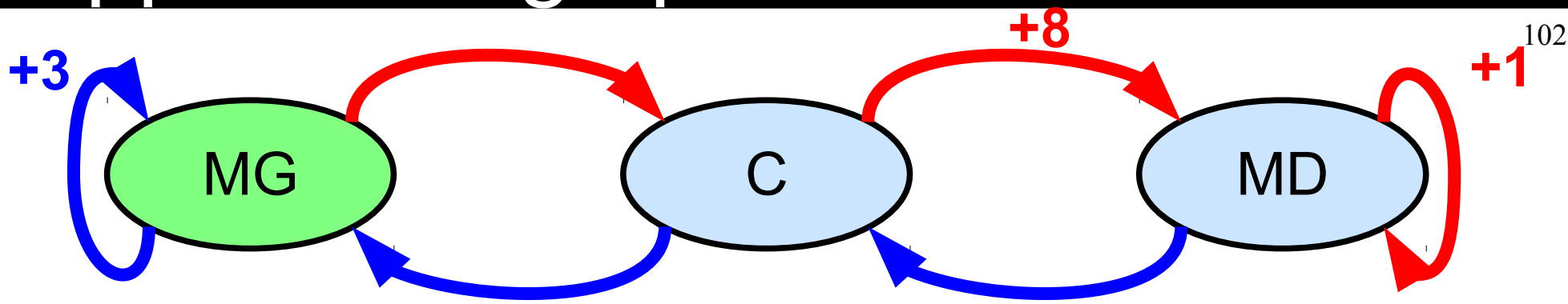


- C, Gauche ==> MG , +0

$$Q^*(C,G) = r + \gamma \, max_{a'} Q^*(MG, a')$$

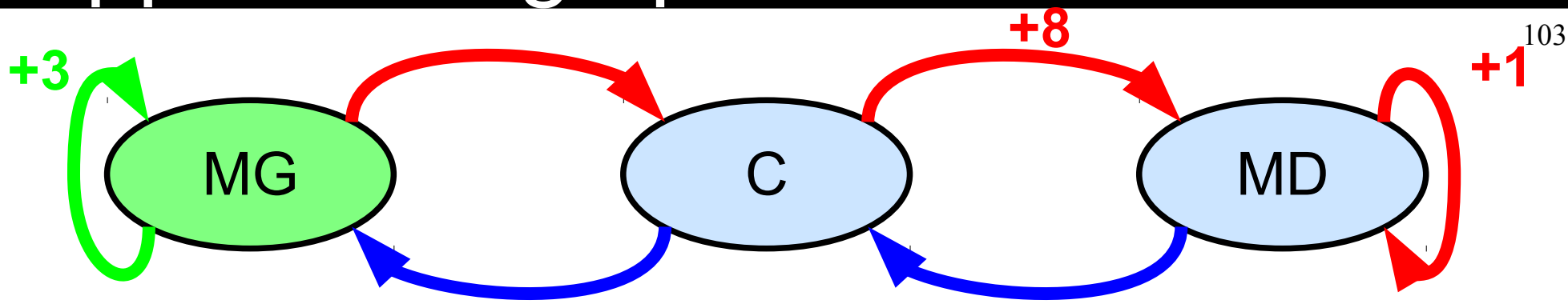
	t=0	t=1
MG, G	0	0
MG, D	0	0
C, G	0	0
C, D	0	0
MD, G	0	0
MD, D	0	0

Apprentissage par renforcement



	t=0	t=1
MG, G	0	0
MG, D	0	0
C, G	0	0
C, D	0	0
MD, G	0	0
MD, D	0	0

Apprentissage par renforcement

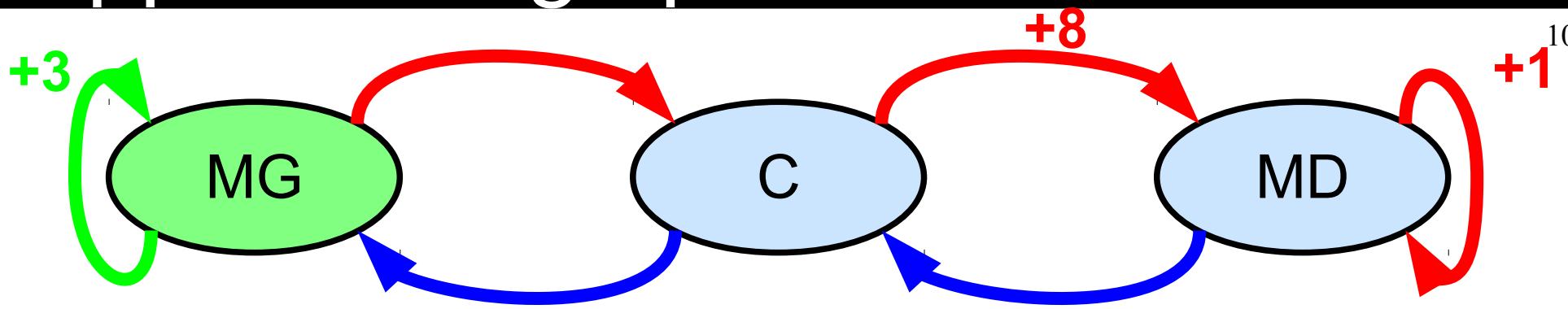


- MG, Gauche ==> MG , +3

$$Q^*(MG,G)=3+\gamma\ max_{a'}Q^*(MG,a')$$

	t=0	t=1	t=2
MG, G	0	0	XX
MG, D	0	0	0
C, G	0	0	0
C, D	0	0	0
MD, G	0	0	0
MD, D	0	0	0

Apprentissage par renforcement

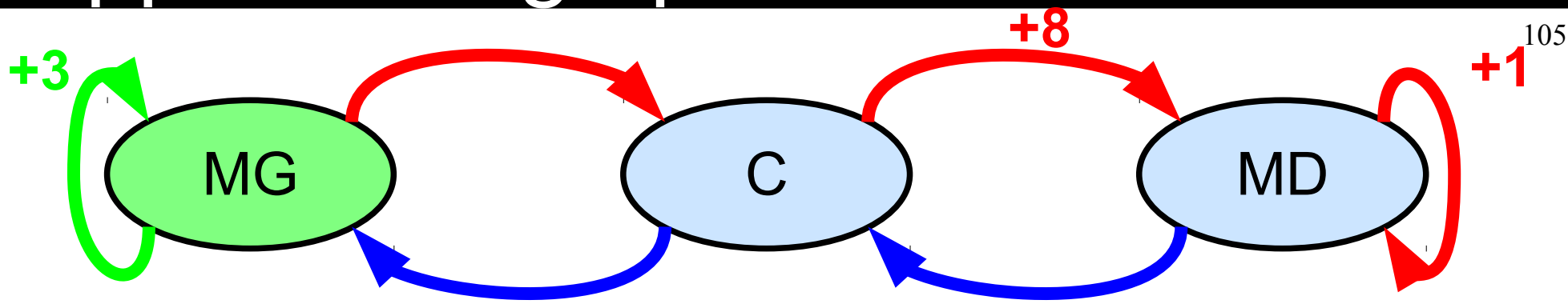


- MG, Gauche ==> MG , +3

$$Q^*(MG,G)=3+\gamma\ max_{a'}Q^*(MG,a')$$

	t=0	t=1	t=2
MG, G	0	0	3
MG, D	0	0	0
C, G	0	0	0
C, D	0	0	0
MD, G	0	0	0
MD, D	0	0	0

Apprentissage par renforcement

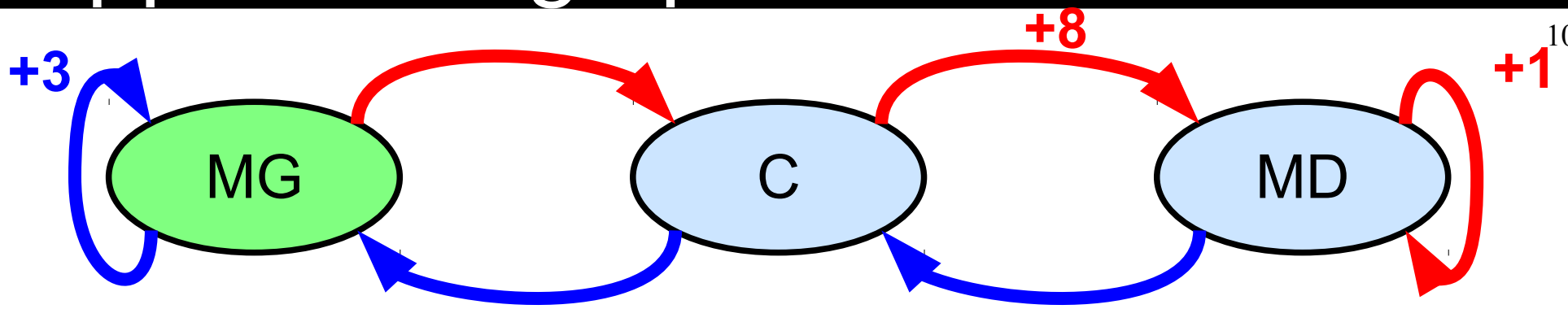


- MG, Gauche ==> MG , +3

$$Q^*(MG,G)=3+\gamma\ max_{a'}Q^*(MG,a')$$

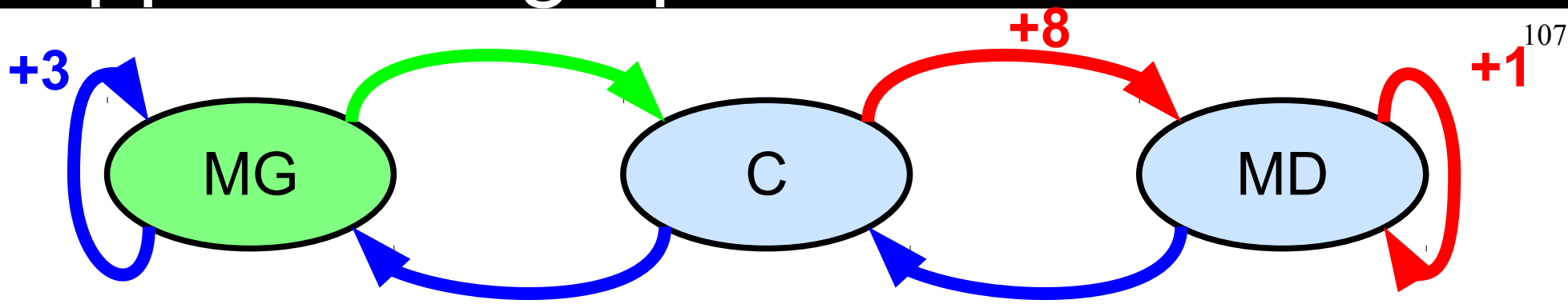
	t=0	t=1	t=2
MG, G	0	0	3
MG, D	0	0	0
C, G	0	0	0
C, D	0	0	0
MD, G	0	0	0
MD, D	0	0	0

Apprentissage par renforcement



	t=0	t=1	t=2
MG, G	0	0	3
MG, D	0	0	0
C, G	0	0	0
C, D	0	0	0
MD, G	0	0	0
MD, D	0	0	0

Apprentissage par renforcement



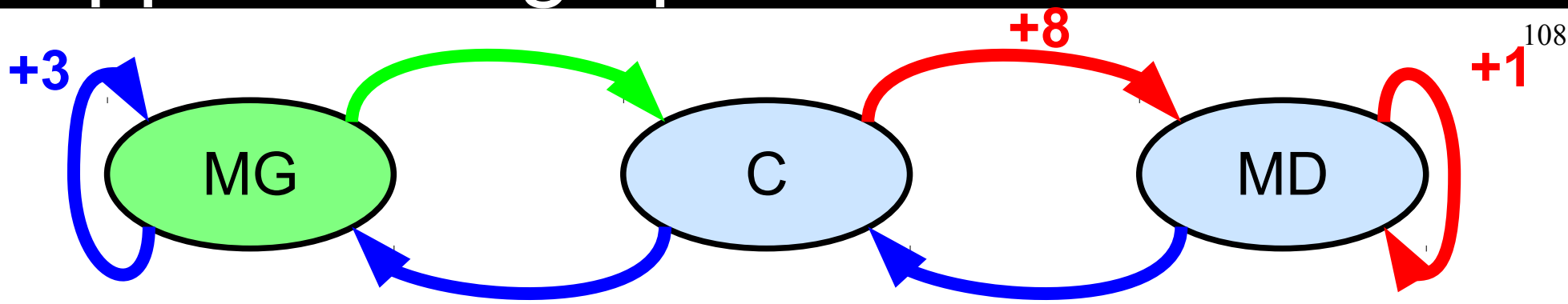
- MG, Droite ==> C , +0

$$Q^*(MG, D) = 0 + \gamma \; max_{a'} Q^*(C, a')$$

	t=0	t=1	t=2
MG, G	0	0	3
MG, D	0	0	0
C, G	0	0	0
C, D	0	0	0
MD, G	0	0	0
MD, D	0	0	0

t=3
3
XX
0
0
0
0

Apprentissage par renforcement

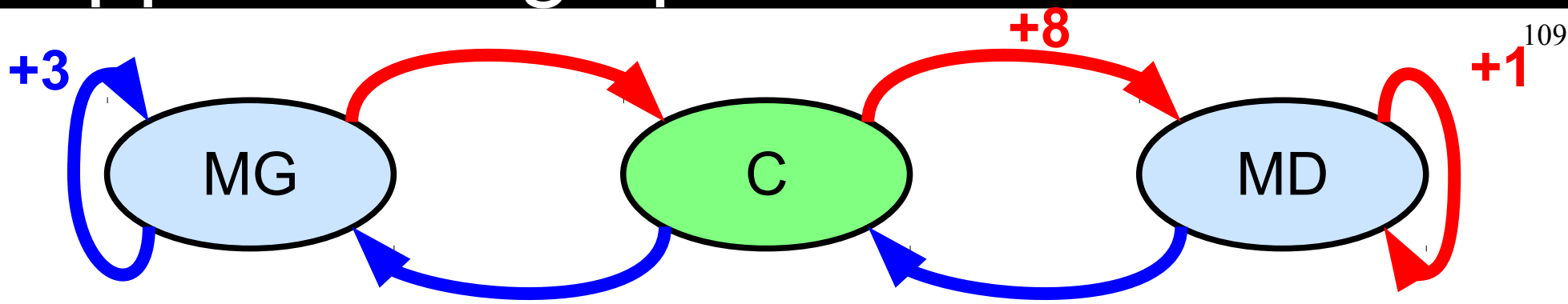


- MG, Droite ==> C , +0

$$Q^*(MG, D) = 0 + \gamma \, max_{a'} Q^*(C, a')$$

	t=0	t=1	t=2	t=3
MG, G	0	0	3	3
MG, D	0	0	0	0
C, G	0	0	0	0
C, D	0	0	0	0
MD, G	0	0	0	0
MD, D	0	0	0	0

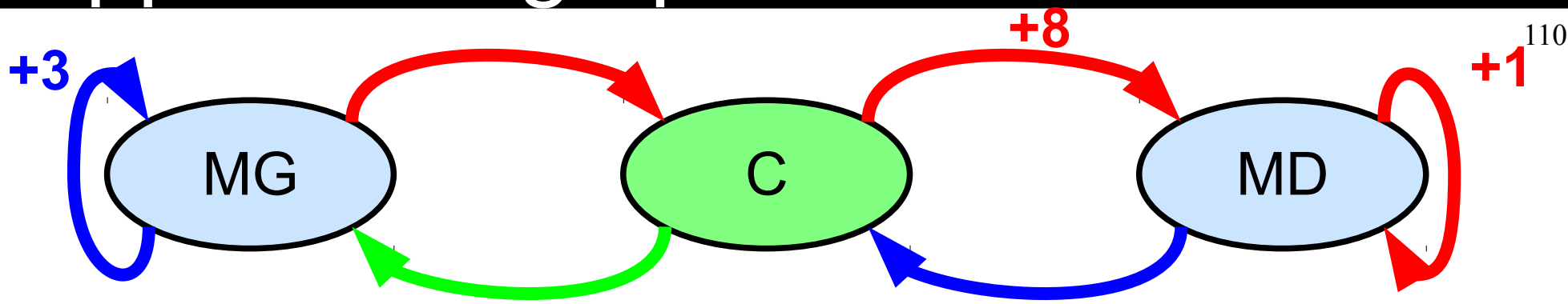
Apprentissage par renforcement



- C

	t=0	t=1	t=2	t=3
MG, G	0	0	3	3
MG, D	0	0	0	0
C, G	0	0	0	0
C, D	0	0	0	0
MD, G	0	0	0	0
MD, D	0	0	0	0

Apprentissage par renforcement

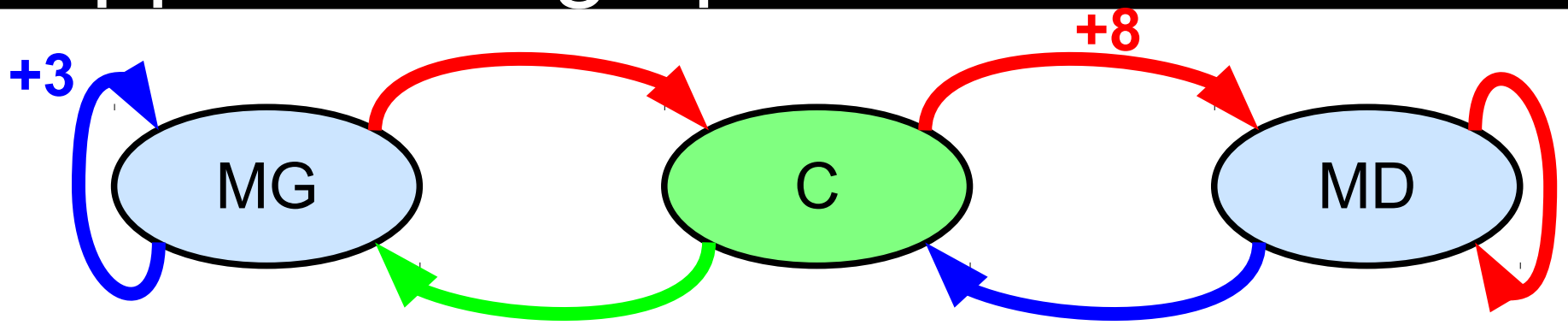


- C, Gauche => MG, 0

	t=0	t=1	t=2	t=3
MG, G	0	0	3	3
MG, D	0	0	0	0
C, G	0	0	0	0
C, D	0	0	0	0
MD, G	0	0	0	0
MD, D	0	0	0	0

t=4
3
0
XXX
0
0
0

Apprentissage par renforcement



- C, Gauche => MG, 0

$$Q^*(C,G) = 0 + \gamma \, max_{a'} Q^*(MG, a')$$

	t=0	t=1	t=2	t=3
MG, G	0	0	3	3
MG, D	0	0	0	0
C, G	0	0	0	0
C, D	0	0	0	0
MD, G	0	0	0	0
MD, D	0	0	0	0

t=4
3
0
2,7
0
0
0

Apprentissage par renforcement

- Dilemme exploration / exploitation

- Exploration
  - Je me déplace au hasard
  - Mais récompenses faibles

- Exploitation
  - Je me déplace au mieux de mes connaissances
  - MAIS ....

Apprentissage par renforcement

- Dilemme exploration / exploitation

	t=0	t=1	t=2	t=3	t=4
MG, G	0	0	3	3	3
MG, D	0	0	0	0	0
C, G	0	0	0	0	2,7
C, D	0	0	0	0	0
MD, G	0	0	0	0	0
MD, D	0	0	0	0	0

- Table me dit que mieux C ==> G
  - Mais je n'apprends pas à droite (qui est mieux)

Apprentissage par renforcement

- Ecrire code python

```
 def QLearning(self,Sdep,nPas,nEpisode,affiche):
        # repeter n episode
            # on reinitiliase au depart
            # pour chaque iteration de l'episode
                # miseAJour Qvaleur
                # part de l'etat d'arrivee
```

```
def majQLearning(self,s,Q):
        '''Permet de faire une mise a jour du Qlearning'''

        # on choisit une action au hasard
        # on execute l'action (s et rec)
        # cherche max arrivee

        # on met a jour Qvaleur
        Q[(s,a)]=rec+gamma*max

        #on retourne le nouvel etat
```

Plan

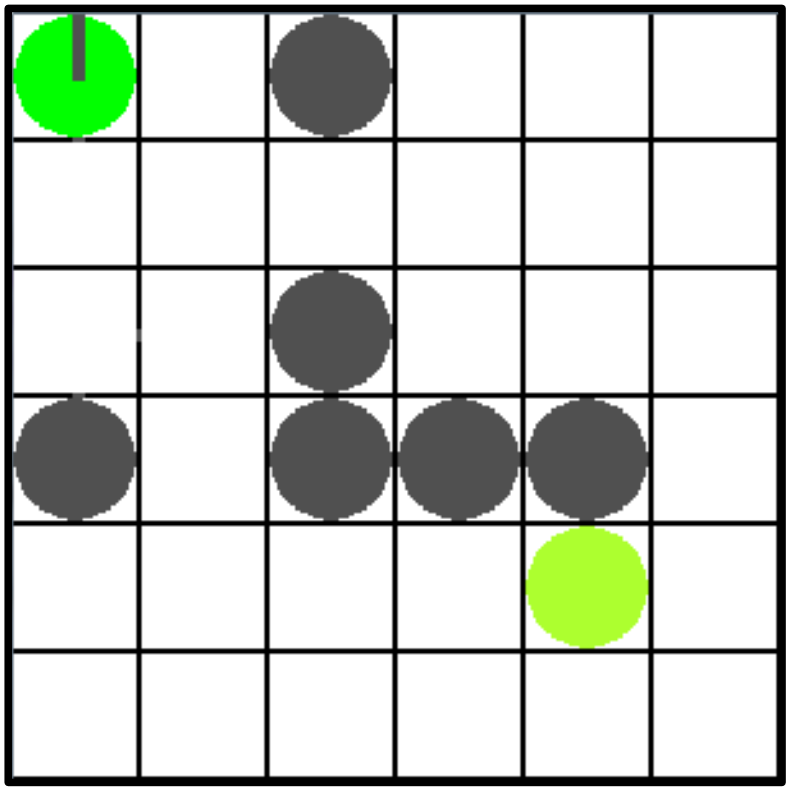
- Conditionnement opérant
- Problème de prise de décision séquentiel
- Monde deterministe
  - Equation de Bellman
  - Planification en environnement connu
  - Apprentissage en environnement inconnu
- Monde stochastique
- Problèmes ouverts

Monde stochatique

- MDP avec des probabilités
  - Transition S x A -> P(S)
- Pourquoi ?
  - Modélise phenomène continu / inconnu
  - Monde probabiliste
- Exemple

Déplacement avec glissements

- Labyrinthe identique
  - Mais quand on se déplace proba d'avancer 2 fois
- Gestion risque
  - Quelle politique ?



Equation bellman

- Equation de bellman légérement modifiée
  - Calcul d'esperance

- Monde deterministe

$$Q^*(s,a) = R(s,a,T(s,a)) + \gamma \, max_{a'} Q^*(T(s,a),a')$$

Equation bellman

- Equation de bellman légérement modifiée
  - Calcul d'esperance

- Monde deterministe

$$Q^*(s,a) = R(s,a,T(s,a)) + \gamma \, max_{a'} Q^*(T(s,a),a')$$

- Monde stochastique

$$Q^*(s,a) = \sum_{s'} P(s,a,s')\left(R(s,a,s') + \gamma \, max_{a'} Q^*(s',a')\right)$$

Value itération

- Meme principe que précédent

- Repeter pour t
  - Chaque état et chaque action

$$Q^*(s,a)=\sum_{s'} P(s,a,s')\left(R(s,a,s')+\gamma\ max_{a'}Q^*(s',a')\right)$$

Apprentissage par renforcement

- Meme principe que précédent
  - MAIS ...

Apprentissage par renforcement

- Meme principe que précédent
  - MAIS ...
  - Une expérience ne suffit plus

Apprentissage par renforcement

- Meme principe que précédent
  - MAIS ...
  - Une expérience ne suffit plus

- Principe
  - Faire des statistiques sur les résultats obtenus

$$Q^*(s,a) = \alpha\,[r + \gamma\, max_{a'} Q^*(s',a')] + (1-\alpha)\,Q^*(s,a)$$

$$\alpha = 1/t$$

Plan

- Conditionnement opérant
- Problème de prise de décision séquentiel
- Monde deterministe
  - Equation de Bellman
  - Planification en environnement connu
  - Apprentissage en environnement inconnu
- Monde stochastique
- Problèmes ouverts

Problemes difficiles

- Observabilité partielle
- Mondes continus
- Taille espace d'états