

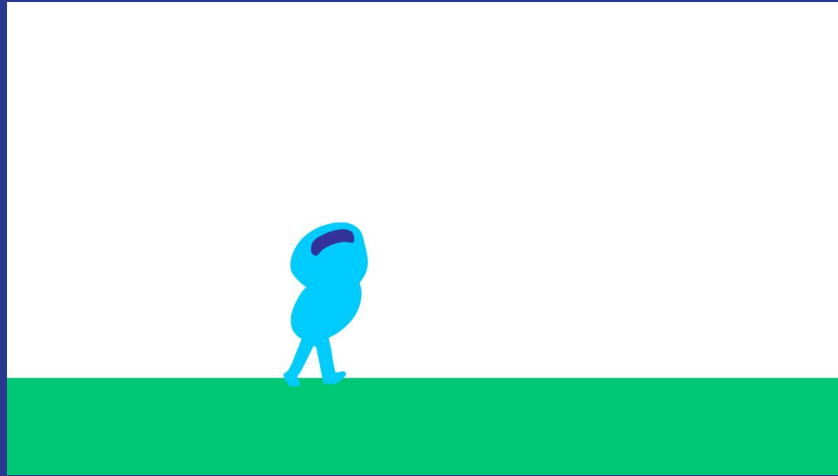
CEFET/RJ

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

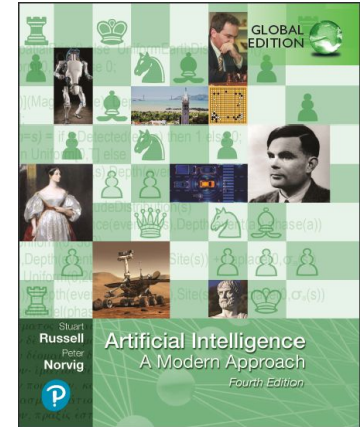
**GCC1734 - INTELIGÊNCIA ARTIFICIAL**

Eduardo Bezerra (CEFET/RJ)  
ebezerra@cefet-rj.br

# APRENDIZADO POR REFORÇO I



Material dessa apresentação é baseado no livro texto (AIMA 4ª edição, capítulo 23)



# Visão geral

- Introdução
- Aplicações
- Processos de Decisão de Markov
- Abordagens (algoritmos de aprendizado)
  - **Q-Learning**

# Introdução

# Agente simples

## Formulação de um Problema de Busca

- Um problema de busca é formulado pela definição de cinco componentes:
  1. Conjunto  $S$  (**espaço de estados**), com um **estado inicial**,
  2. Função  $ACTIONS(s)$ : produz as **ações** possíveis em cada estado,
  3. Função  $RESULT(s, a)$ : **modelo de transição** ou **função sucessora**, produz o estado resultante de selecionar a ação  $a$  no estado  $s$ .
  4. Função **teste de objetivo**, que permite ao agente determinar se seu objetivo foi alcançado.
  5. Função **custo de caminho** (função aditiva e cumulativa), que permite ao agente comparar planos alternativos.

# Agente adversarial

## Formulação (jogo)

- Formulação possível (uma dela):
  - ▣ Estados:  $S$
  - ▣ Jogadores:  $P = \{1..N\}$  (usualmente intercalam ações)
  - ▣ Funções  $ACTIONS(s): S \rightarrow A$ 
    - Ações possíveis podem também depender do jogador.
  - ▣ Função de transição  $RESULT(s, a) : S \times A \rightarrow S$
  - ▣ Teste de estado terminal:  $S \rightarrow \{True, False\}$
  - ▣ Função utilidade:  $U(s, p): S \times P \rightarrow \mathbb{R}$

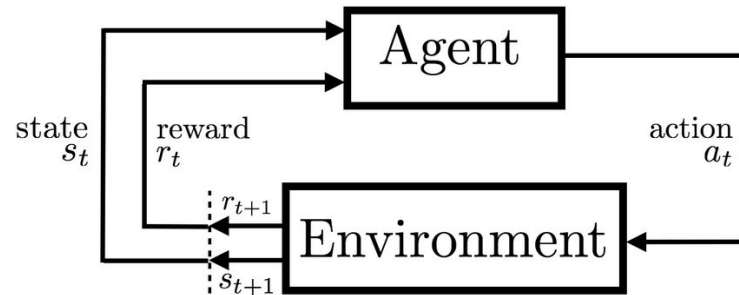
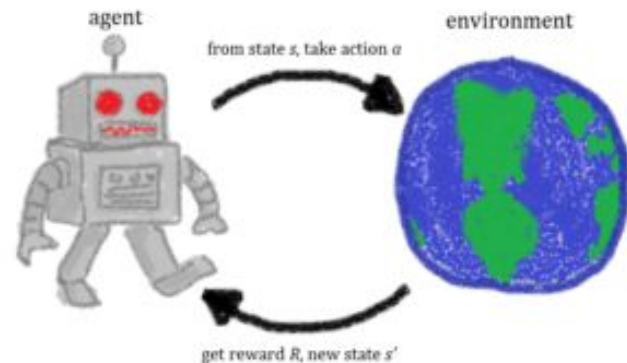
# Aprendizado

- Duas características inerentes a seres inteligentes são:
  - Capacidade de determinar estratégias para resolver problemas.
  - Capacidade de aprender.
- Os agentes que estudamos até aqui são capazes de determinar planos ou políticas para resolver problemas.
- Entretanto, esses agentes não têm capacidade de **aprendizado**.



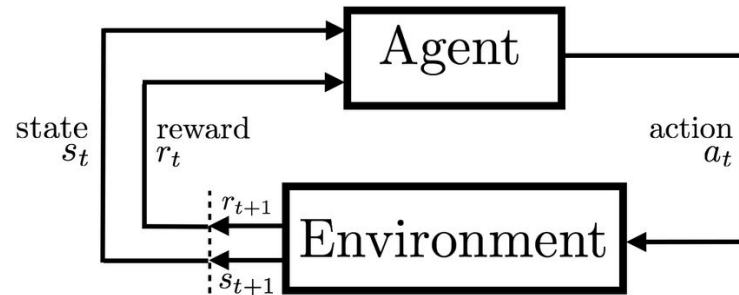
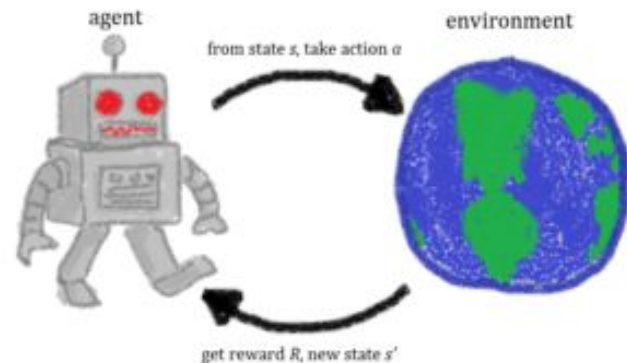
# Aprendizado por Reforço

- Ideias básicas:
  - De início
    - o agente sabe que ações pode realizar,
    - mas não sabe que ações são boas e quais são ruins (em cada estado).
  - Ao interagir com o ambiente, o agente recebe *feedback* na forma de **recompensas** pelas ações que seleciona.
    - O agente seleciona uma ação no estado corrente; como resultado, ele recebe uma recompensa e transita para outro estado.



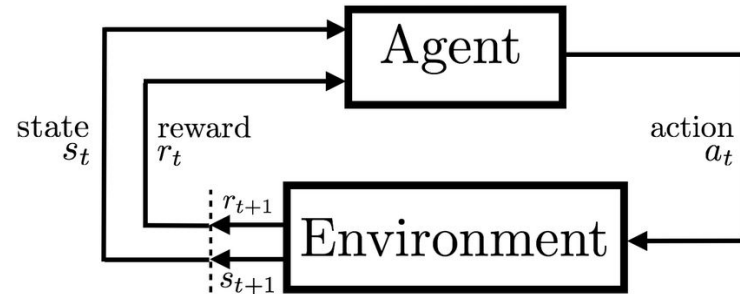
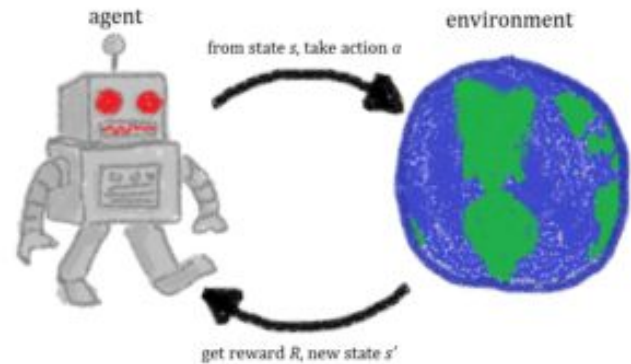
# Aprendizado por Reforço

- Ideias básicas (cont.):
  - A utilidade de um estado (ou estado-ação) é uma função das recompensas recebidas pelo agente.
  - Agente deve (aprender a) a selecionar ações de tal forma a **maximizar a soma esperada de recompensas**.



# Aprendizado por Reforço

- O aprendizado corresponde ao processo pelo qual o agente identifica qual ação é adequada em cada estado.
- Na prática, corresponde a muitas (milhares, milhões) realizações do ciclo ilustrado nas figuras.
- No fim desse processo, a agente terá determinado (i.e., aprendido) uma **política**.

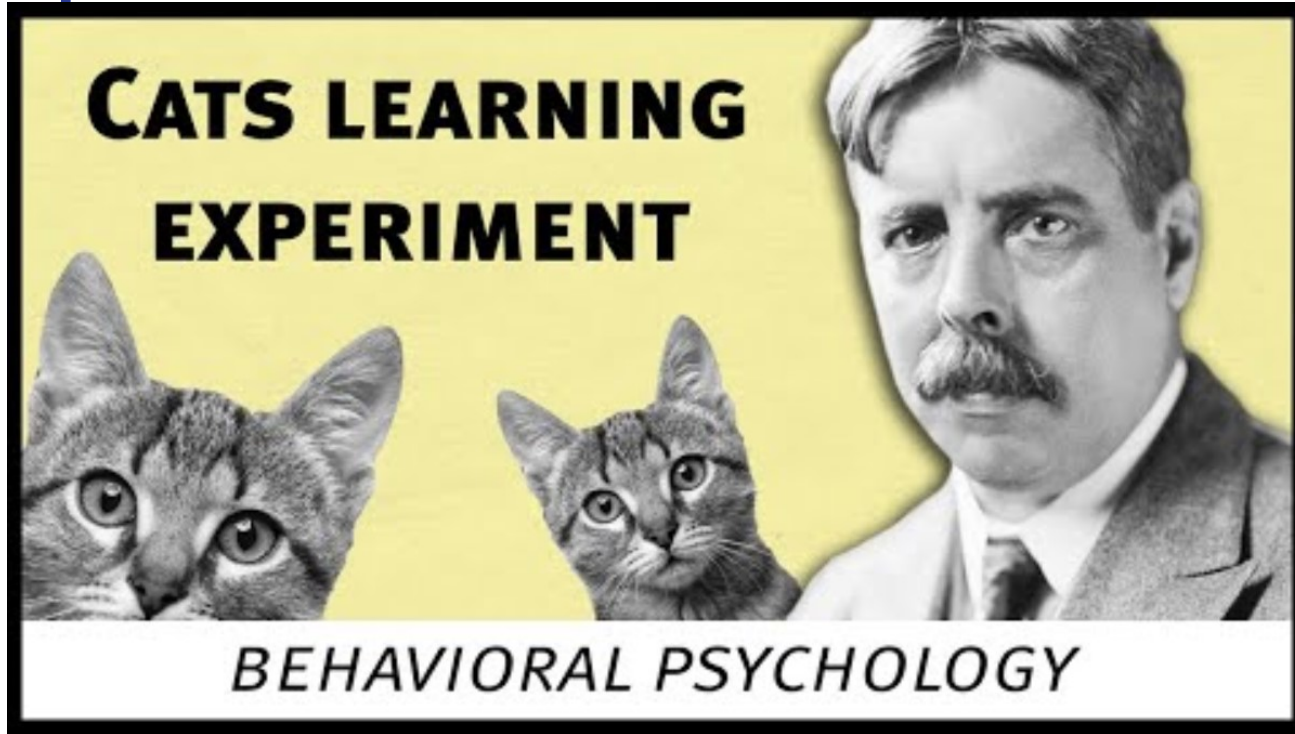


# Hipótese de recompensa

- O aprendizado por reforço é baseado na **hipótese de recompensa** (*reward hypothesis*):
  - “Qualquer objetivo [a ser alcançado pelo agente] pode ser formalizado como o resultado da maximização de uma recompensa cumulativa.”

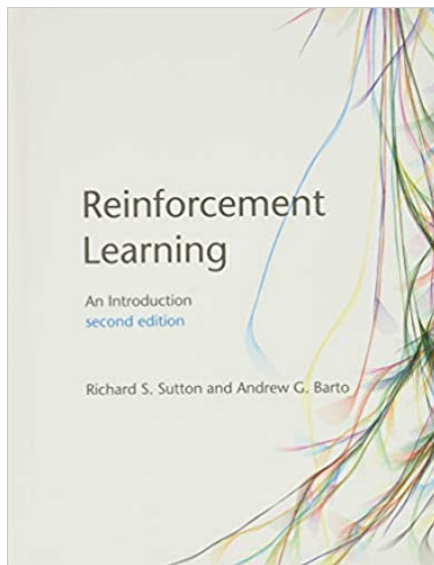
*That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward).*

# Inspiração: psicologia comportamental



# Livro texto sobre RL

- Reinforcement Learning: An Introduction, Sutton & Barto 2018
  - <http://incompleteideas.net/book/the-book-2nd.html>



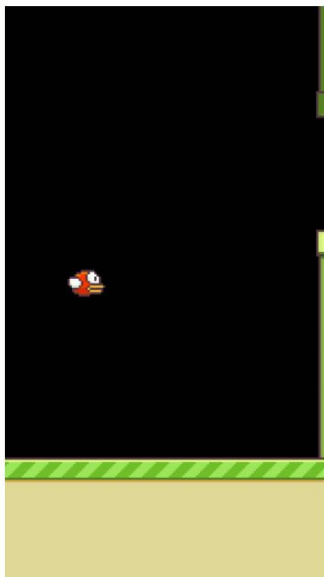
# Aplicações

# Exemplos de aplicações

- Em geral, se o objetivo é que o agente aprenda a realizar uma tarefa via interações com um ambiente, aprendizado por reforço pode ser usado.



# Flapping Bird



## Ações

Para cada estado, duas ações possíveis: "Go up", "Do nothing"

## Recompensas

A recompensa é baseada no parâmetro "Vida".

**+1** se o Flappy Bird ainda está vivo

**-1000** se o Flappy Bird morre

# Atari

## Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

[Affiliations](#) | [Contributions](#) | [Corresponding authors](#)

*Nature* **518**, 529–533 (26 February 2015) | doi:10.1038/nature14236

Received 10 July 2014 | Accepted 16 January 2015 | Published online 25 February 2015



Citation



Reprints



Rights & permissions



Article metrics

The theory of reinforcement learning provides a normative account<sup>1</sup>, deeply rooted in psychological<sup>2</sup> and neuroscientific<sup>3</sup> perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use

## Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

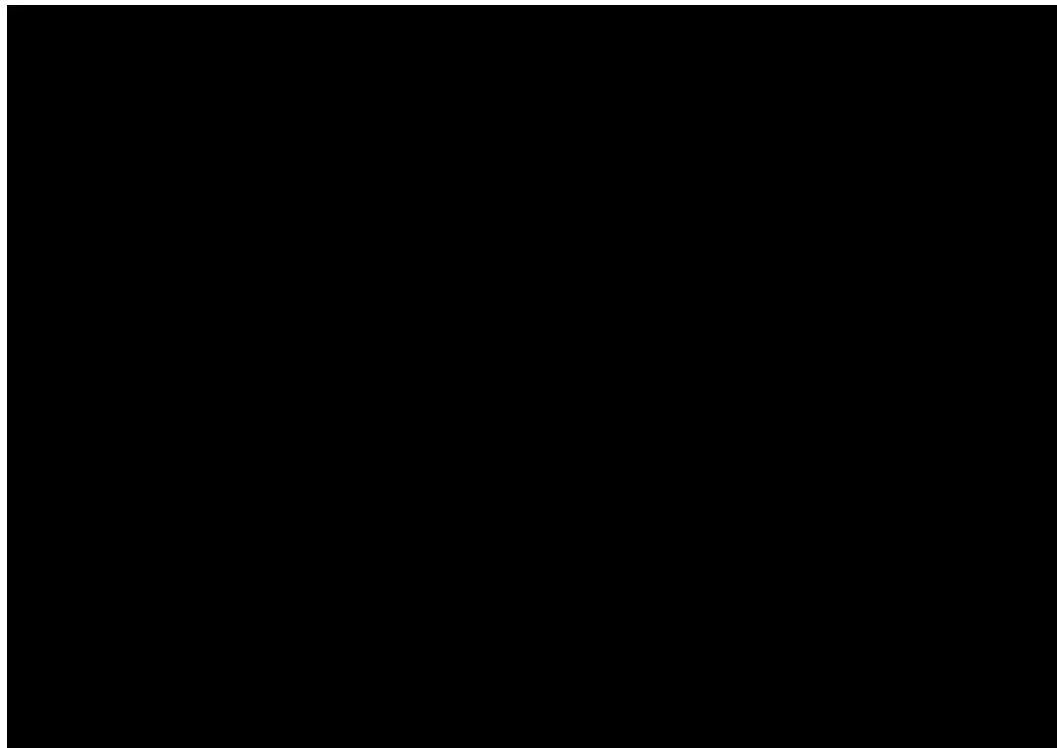
DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

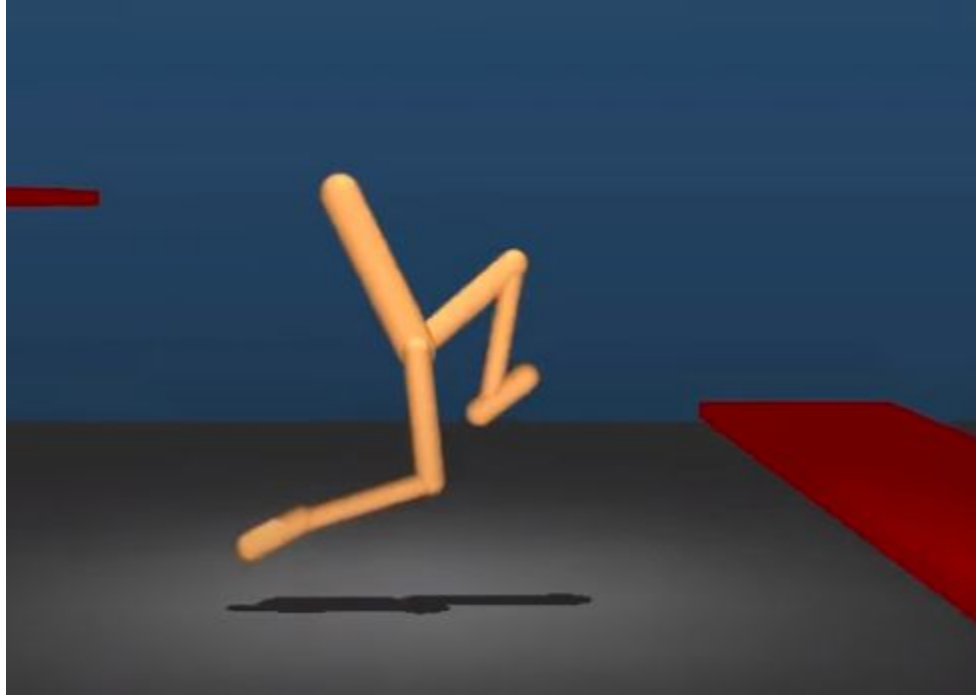
### Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

# Atari breakout



# Locomoção



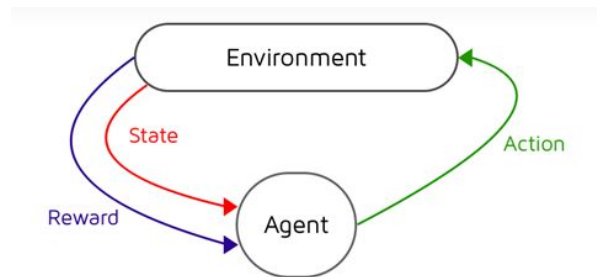
# Fazer um robô andar



# Processo de Decisão de Markov

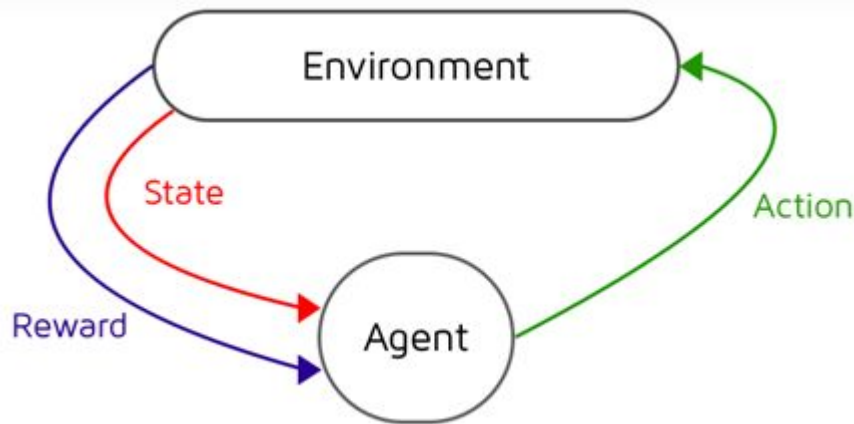
# Processo de Decisão de Markov (PDM)

- Um PDM é um formalismo usado para formular um problema de aprendizado por reforço.
- Em um PDM:
  - um agente interage com o ambiente realizando ações;
  - a cada ação executada pelo agente,
    - ele recebe uma recompensa e
    - experimenta uma transição de estado.



# Processo de Decisão de Markov (PDM)

- Ao interagir com o ambiente, o agente busca maximizar a soma de recompensas por ele recebidas.
  - Hipótese de recompensa.





# Processo de Decisão de Markov (PDM)

- Um PDM é especificado por diversos componentes:
  - ▣ Um conjunto de estados  $S$ 
    - Um estado inicial (ou distribuição inicial)
    - Zero ou mais estados terminais
  - ▣ Um conjunto de ações  $A$
  - ▣ Uma função recompensa  $R(s, a, s')$
  - ▣ Um modelo de transição  $T(s, a, s')$ 
    - Probabilidade condicional de se alcançar  $s'$  a partir de  $s$ , se a ação  $a$  for executada em  $s$ , i.e.,  $\Pr(s' | s, a)$ .

# Processo de Decisão de Markov (PDM)

- Usamos a expressão resolver um PDM para denotar o procedimento de determinar uma política  $\pi(s)$  a partir dos componentes desse PDM.
- Há diversos algoritmos para resolver um PDM, parcial ou completamente especificado.

# Processo de Decisão de Markov (PDM)

- Complicador no contexto do aprendizado por reforço: o agente recebe um PDM parcialmente especificado.
- Concretamente: o agente não recebe as funções  $T$  e  $R$ .
  - i.e., de início o agente não sabe que transições são mais (ou menos) prováveis, e nem que ações geram que resultado (positivo ou negativo).
  - O agente deve efetivamente experimental ações e estados para **aprender** a política ótima.

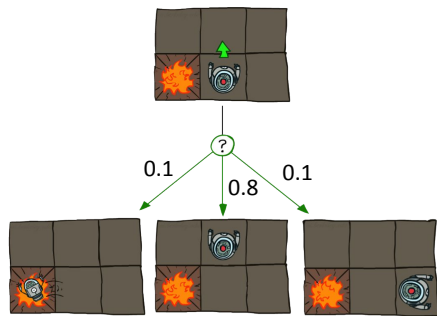
# Função recompensa $R(s, a, s')$

- A função recompensa codifica a recompensa recebida pelo agente quando ele toma uma ação  $a$  em um estado  $s$  e, como resultado, transita para o estado  $s'$ .
  - ▣ Recompensas podem ser valores positivos ou negativos!

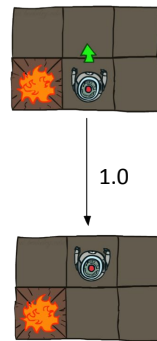
$$R(s, a, s')$$

# Modelo de transição $T(s, a, s')$

- Forma de representar a incerteza no resultado de uma ação.



Busca não-determinística



Busca determinística

# Modelo de transição $T(s, a, s')$

- “Dado o estado atual, o estado seguinte é independente dos estados passados.”
- Para PDMs, “Markov” significa que os resultados de uma ação depende apenas do estado atual:

$$\begin{aligned} P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots, S_0 = s_0) \\ = \\ P(S_{t+1} = s' | S_t = s_t, A_t = a_t) \end{aligned}$$

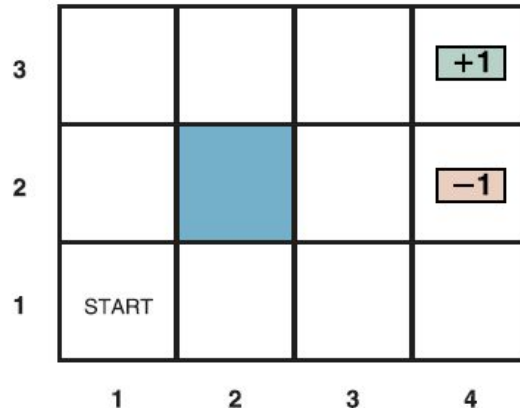
- Similar a uma busca em que a função sucessora apenas depende do estado e ação atuais (e não de todo o histórico).



Andrey Markov (1856-1922)

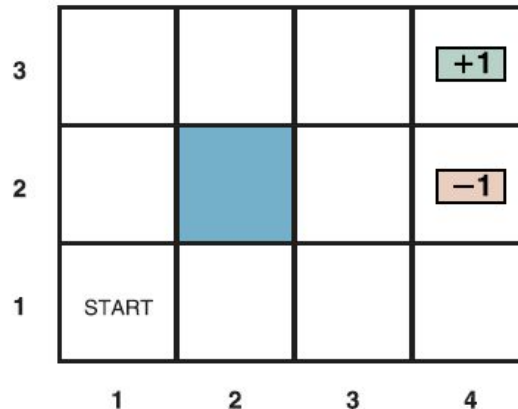
# Exemplo: Mundo Grade

- Para exemplificar os componentes de um PDM, vamos usar o Mundo Grade.



# Mundo Grade – estados

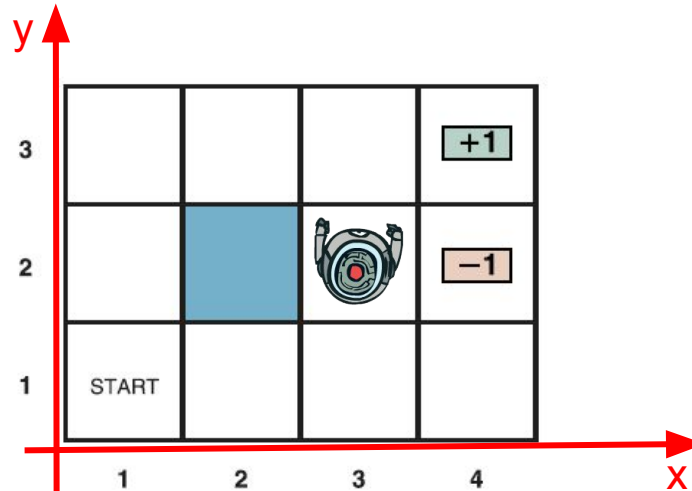
- Nesse ambiente, há dois estados terminais e nove estados não-terminais.





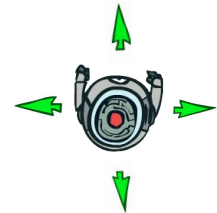
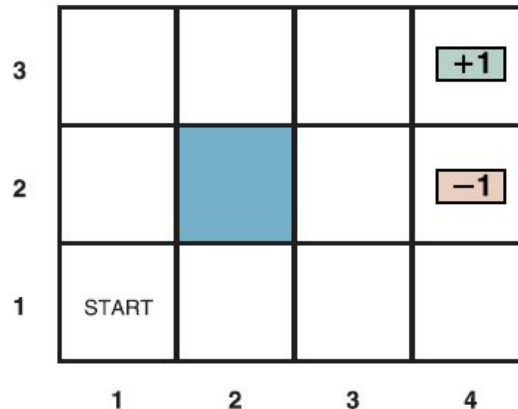
# Mundo Grade – estados

- Nesse ambiente, há dois estados terminais e nove estados não-terminais.
  - Cada estado é representado por um par ordenado  $(x, y)$  (e.g., na figura a seguir o agente se encontra no estado  $(3, 2)$ ).



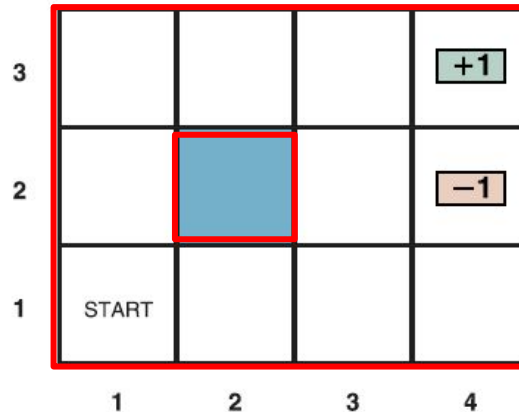
# Mundo Grade – ações

- ❑ Ações disponíveis em cada estado não-terminal: Up, Down, left, Right.
- ❑ Única ação disponível nos estados terminais: Exit.
- ❑ Ações fazem com que o agente transite (mude) de estado.



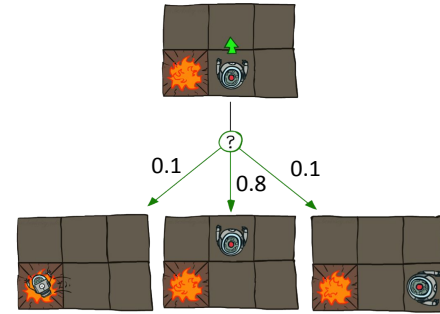
# Mundo Grade – ações (cont.)

- Há **obstáculos** que limitam o movimento (ações) do agente.
  - Se há um obstáculo na direção em que o agente quer se mover, ele permanece onde está no próximo passo de tempo.



# Mundo Grade – modelo de transição

- O modelo de transição para o mundo grade é estocástico.
  - i.e., uma mesma ação tomada em um mesmo estado pode produzir diferentes resultados em momentos diferentes.
- Contudo, note que um problema de AR em geral não precisa envolver um modelo de transição estocástico.

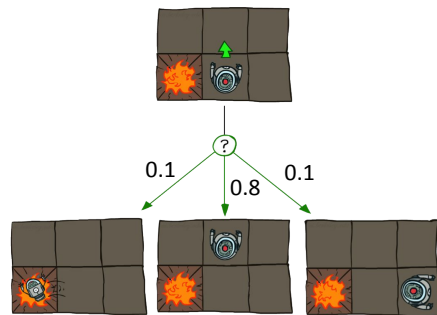


# Mundo Grade – modelo de transição

- No Mundo Grade, o modelo de transição é definido de tal modo que o resultado de uma ação selecionada tem 20% de ruído.

# Mundo Grade – modelo de transição

- ❑ No Mundo Grade, o modelo de transição é definido de tal modo que o resultado de uma ação selecionada tem 20% de ruído.
- ❑ Exemplos (veja figura):
  - Em 80% das vezes, a ação “Up” leva o agente para o estado (2,2);
  - Em 10% das vezes, “Up” leva para (1,1);
  - Nos restantes 10% das vezes, leva para (3,1).



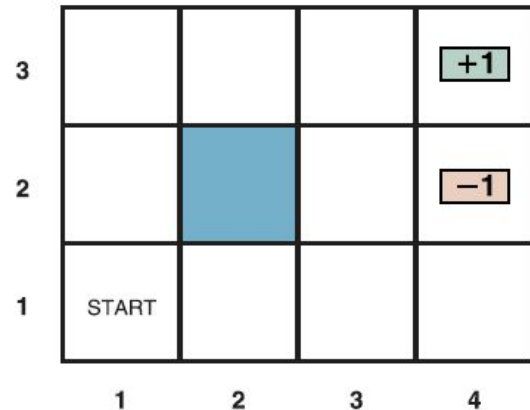
$$T((2,1), Up, (2,2)) = 0.8$$

$$T((2,1), Up, (1,1)) = 0.1$$

$$T((2,1), Up, (3,1)) = 0.1$$

# Mundo Grade – função recompensa

- No Mundo Grade, a função de recompensa é normalmente definida de tal modo que valores são
  - pequenos em cada estado não-terminal;
  - grandes nos estados terminais.

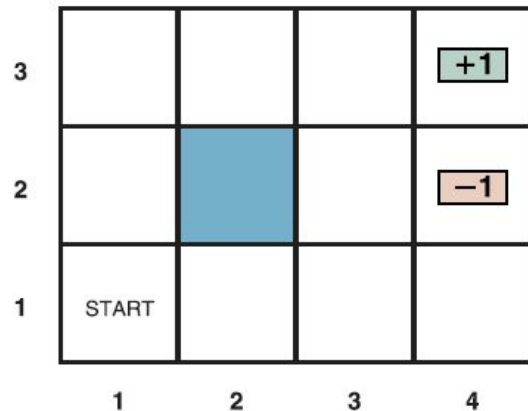


# Mundo Grade – exemplo

Considerare:

- ▣  $R([4,3], \text{Exit}, \blacksquare) = +1$
- ▣  $R([4,2], \text{Exit}, \blacksquare) = -1$
- ▣  $R(s, a, s') = -0.04$  nos demais casos.

0



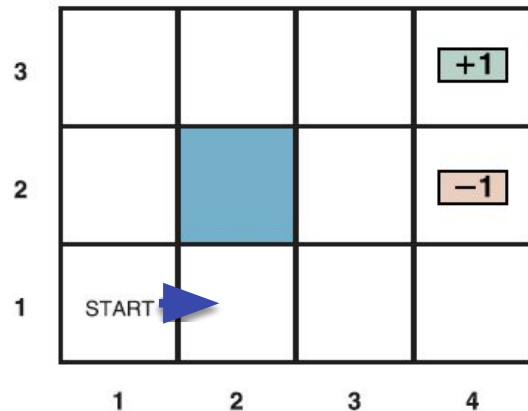


# Mundo Grade – exemplo

Considerare:

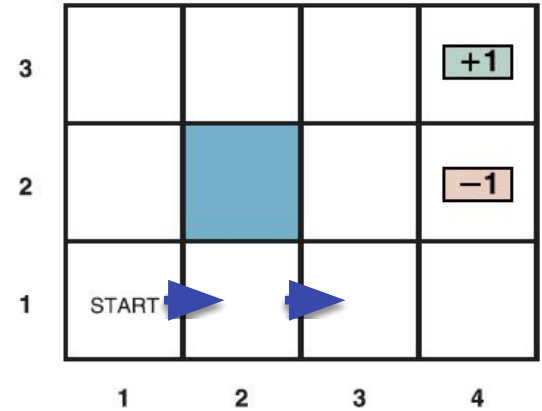
- ▣  $R([4,3], \text{Exit}, \blacksquare) = +1$
- ▣  $R([4,2], \text{Exit}, \blacksquare) = -1$
- ▣  $R(s, a, s') = -0.04$  nos demais casos.

$0 - 0.04$



# Mundo Grade – exemplo

$0 - 0.04 - 0.04$

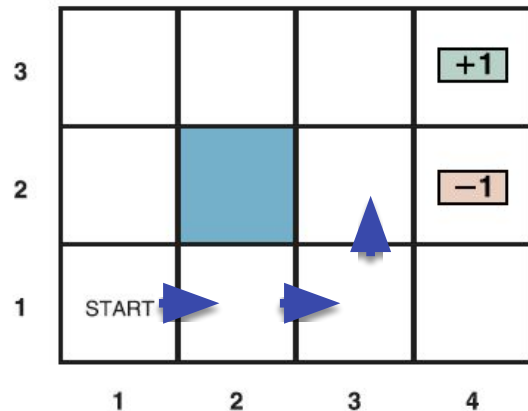


# Mundo Grade – exemplo

Considerare:

- ▣  $R([4,3], \text{Exit}, \blacksquare) = +1$
- ▣  $R([4,2], \text{Exit}, \blacksquare) = -1$
- ▣  $R(s, a, s') = -0.04$  nos demais casos.

$0 - 0.04 - 0.04 - 0.04$

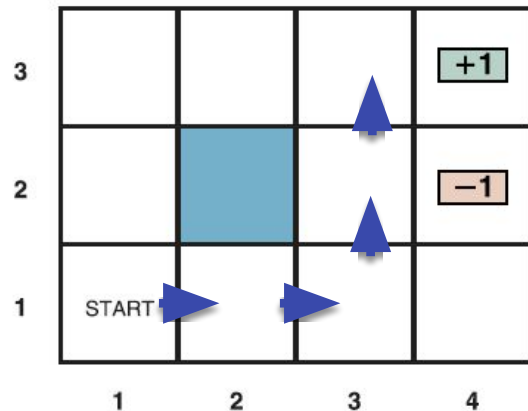


# Mundo Grade – exemplo

Considerare:

- ▣  $R([4,3], \text{Exit}, \blacksquare) = +1$
- ▣  $R([4,2], \text{Exit}, \blacksquare) = -1$
- ▣  $R(s, a, s') = -0.04$  nos demais casos.

$0 - 0.04 - 0.04 - 0.04 - 0.04$

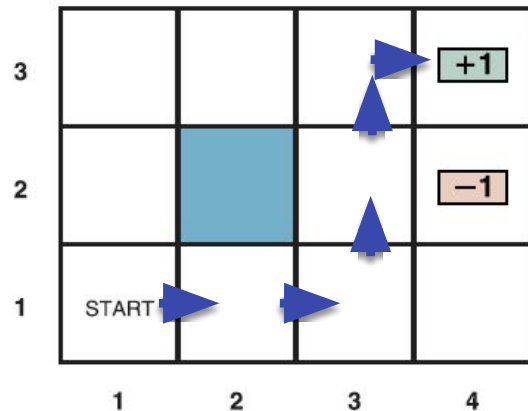


# Mundo Grade – exemplo

Considerare:

- ▣  $R([4,3], \text{Exit}, \blacksquare) = +1$
- ▣  $R([4,2], \text{Exit}, \blacksquare) = -1$
- ▣  $R(s, a, s') = -0.04$  nos demais casos.

$0 - 0.04 - 0.04 - 0.04 - 0.04 - 0.04$



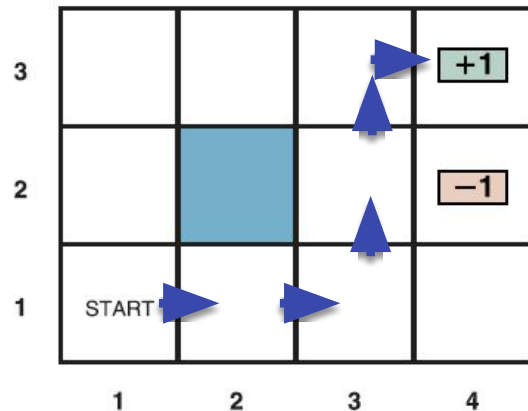
# Mundo Grade – exemplo

Considera:

- ▣  $R([4,3], \text{Exit}, \blacksquare) = +1$
- ▣  $R([4,2], \text{Exit}, \blacksquare) = -1$
- ▣  $R(s, a, s') = -0.04$  nos demais casos.

$$0 - 0.04 - 0.04 - 0.04 - 0.04 - 0.04 + 1$$

Recompensa total acumulada neste episódio: 0.8



# Mundo Grade – exemplo

- Neste episódio o agente experimentou uma sequência de

**transições:**

$([1,1], \text{Right}, [2,1], -0.04)$

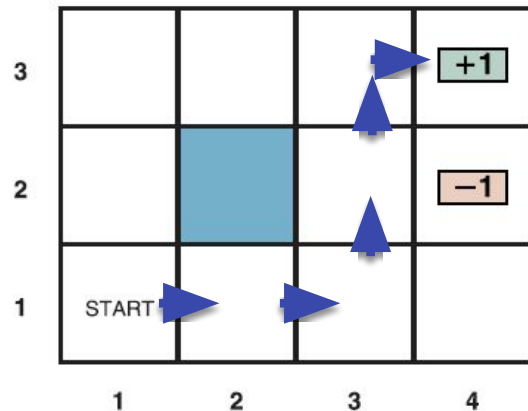
$([2,1], \text{Right}, [3,1], -0.04)$

$([3,1], \text{Up}, [3,2], -0.04)$

$([3,2], \text{Up}, [3,3], -0.04)$

$([3,3], \text{Right}, [4,3], -0.04)$

$([4,3], \text{Exit}, \blacksquare, +1)$



# Outros conceitos relevantes

- ☐ Política
- ☐ Política ótima
- ☐ Estado
- ☐ Q-estado
- ☐ Função utilidade  $V(s)$
- ☐ Função utilidade  $Q(s, a)$



# Política

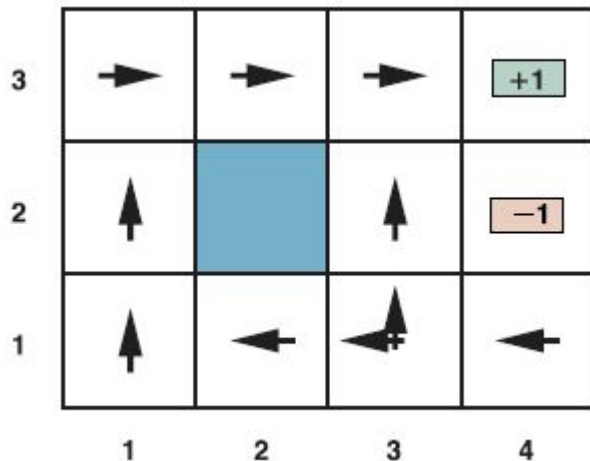
- Nos problemas de busca determinística envolvendo apenas um agente, vimos que o agente busca por um **plano**, ou sequência de ações do estado inicial até o estado objetivo no grafo de espaço de estados.

# Política

- Nos problemas de busca determinística envolvendo apenas um agente, vimos que o agente busca por um **plano**, ou sequência de ações do estado inicial até o estado objetivo no grafo de espaço de estados.
- Já em um PDM, o agente busca uma **política**, que corresponde a uma função  $\pi: S \rightarrow A$ .
  - Uma política recomenda uma ação para cada estado possível em que o agente pode se encontrar.
  - Uma política explícita define um agente reflexivo.

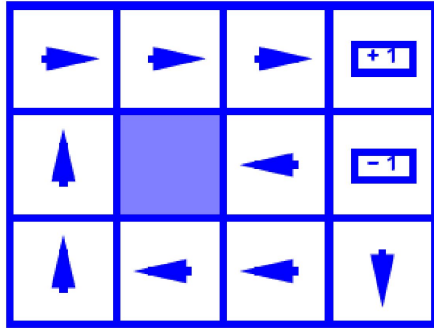
# Política ótima

- Uma **política ótima** é aquela que, se seguida pelo agente, maximiza a utilidade esperada.

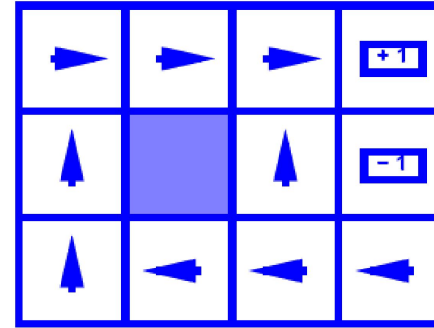


Política ótima quando  $R(s, a, s') = -0.04$  para todos os estados não-terminais  $s$

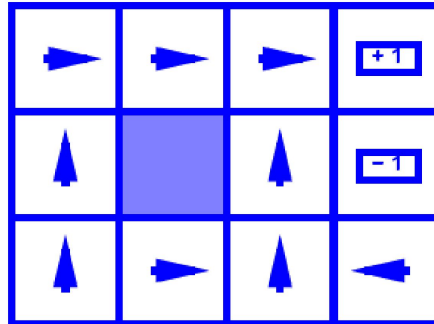
# Política ótima – outros exemplos



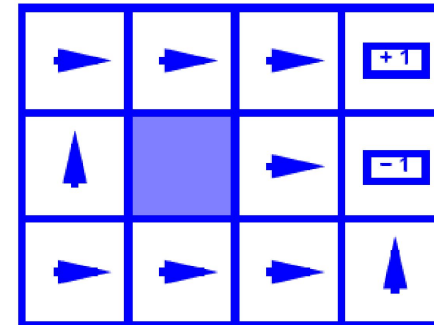
$$R(s, a, s') = -0.01$$



$$R(s, a, s') = -0.03$$



$$R(s, a, s') = -0.4$$



$$R(s, a, s') = -2.0$$

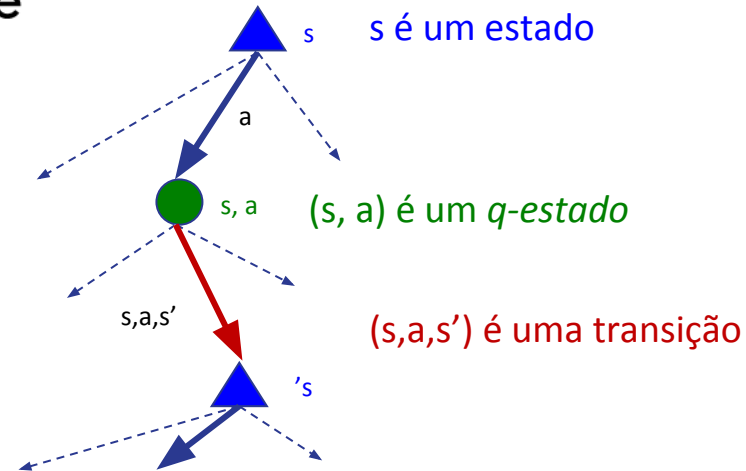
$\forall s, a, s' (R = cte)$   
 $s$  não-terminal

# Política ótima

- Um política ótima, denotada por  $\pi^*(s)$ , recomenda a ação ótima a partir de um dado estado  $s$ .
- Essa recomendação é feita com base em um **valor** associado ao estado  $s$ ...

# Estados e q-estados

- Além dos estados propriamente ditos, em um PDM também há o conceito de **q-estado**.
- Um q-estado é definido por um par estado-ação:  $(s, a)$ .
- Equivale ao conceito de nó de acaso na busca expectimax.



# Estados e q-estados

- ❑ Para cada estado (ou q-estado) de um PDM, podemos computar um valor correspondente à sua utilidade.
  - ▣ A utilidade de um estado  $s$  é um valor do quanto se espera que o agente receba em recompensas a partir de  $s$ .
  - ▣ A utilidade de um q-estado  $(s, a)$  é um valor do quanto se espera que o agente receba em recompensas a partir de  $s$  e tomando  $a$  ação a neste estado.
- ❑ **Esses valores podem ser usados para determinar a política ótima para o agente.**

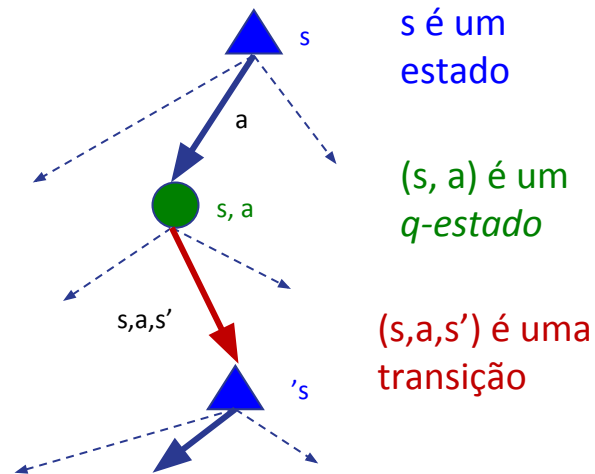
# Estados e q-estados

## Utilidade de um estado $s$ :

- ▣  $V^*(s)$  = utilidade esperada começando em  $s$  e agindo *otimamente* (i.e., *racionalmente*).

## Utilidade de um q-estado $(s,a)$ :

- ▣  $Q^*(s, a)$  = utilidade esperada começando por tomar a ação  $a$  partindo do estado  $s$  e, desse estado em diante, agindo otimamente.





# Exemplo: $V^*(s)$ no Mundo Grade



Valores □ utilidades de cada estado.

Setas □ ações recomendadas pela política ótima.

Noise = 0.2

$\gamma = 0.9$

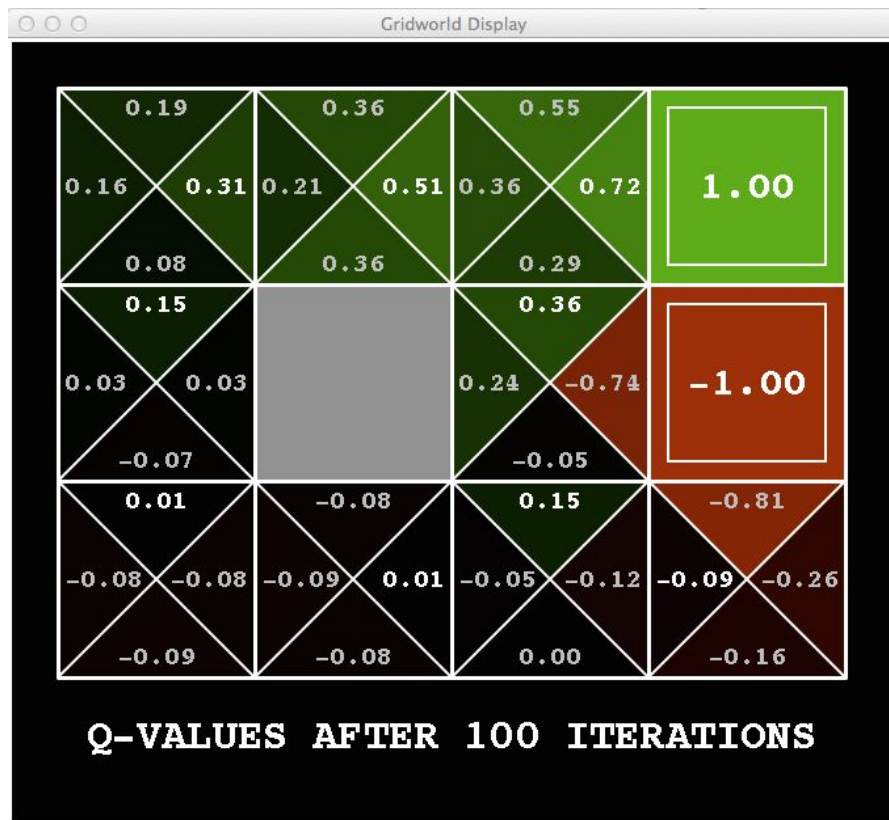
Living reward = 0

# Exemplo: $V^*(s)$ no Mundo Grade



Noise = 0.2  
 $\gamma = 0.9$   
Living reward = -0.1

# Exemplo: $Q^*(s, a)$ no Mundo Grade



Valores □ utilidades de cada q-estado.

Noise = 0.2  
 $\gamma = 0.9$   
Living reward = -0.1

# Abordagens (algoritmos de aprendizado)

# Abordagens

- Há duas famílias de algoritmos de aprendizado por reforço:
  - Baseada em modelo (*model-based*)
  - Livre de modelo (*model-free*)

# Abordagens

- Há duas famílias de algoritmos de aprendizado por reforço:
  - ▣ Baseada em modelo (*model-based*): estima as funções de transição  $T(s,a,s')$  e recompensa  $R(s,a,s')$  e usa essas estimativas para resolver o MDP.
  - ▣ Livre de modelo (*model-free*)

# Abordagens

- Há duas famílias de algoritmos de aprendizado por reforço:
  - Baseada em modelo (*model-based*): estima as funções de transição  $T(s, a, s')$  e recompensa  $R(s, a, s')$  e usa essas estimativas para resolver o MDP.
  - Livre de modelo (*model-free*): estima  $V(s)$  ou  $Q(s, a)$  diretamente, sem antes construir um modelo de recompensas e transições do MDP.

# Abordagens

- Há duas famílias de algoritmos de aprendizado por reforço:
  - Baseada em modelo (*model-based*): estima as funções de transição  $T(s,a,s')$  e recompensa  $R(s,a,s')$  e usa essas estimativas para resolver o MDP.
  - Livre de modelo (*model-free*): estima  $V(s)$  ou  $Q(s,a)$  diretamente, sem antes construir um modelo de recompensas e transições do MDP.
- Todos esses algoritmos usam amostras de transições obtidas durante o treinamento.



# Abordagens

- Há duas famílias de algoritmos de aprendizado por reforço:
  - Baseada em modelo (*model-based*)
  - Livre de modelo (*model-free*)
    - Avaliação Direta
    - Aprendizado por Diferença Temporal
    - **Q-learning**

# Q-learning

# Q-Learning

- Um algoritmo iterativo para determinar os q-valores  $Q(s, a)$ .
- No Q-Learning, toda vez que o agente experimenta uma transição de um estado  $s$  para outro  $s'$  por meio da realização de uma ação  $a$ , a estimativa para  $Q(s, a)$  é atualizada.
- Essa atualização ocorre por meio da interpolação (i.e., média ponderada) de dois valores:
  - ▣ (1) a estimativa atual de  $Q(s, a)$
  - ▣ (2) a recompensa imediata obtida com a transição

# Q-Learning

- Um agente equipado com o algoritmo Q-Learning aprende os q-valores de cada q-estado.
  - Isso equivale a aprender que ação tomar em cada estado, i.e., aprender o quão bom (ou ruim) é tomar uma determinada ação  $a$  em determinado estado  $s$ .
- Ideia básica: agente deve aprender a partir de cada experiência!
  - Atualizar  $Q(s, a)$  cada vez que o agente experimenta uma transição  $(s, a, s', r)$
  - Os estados  $s'$  mais prováveis de serem alcançados a partir de  $s$  irão contribuir com atualizações em  $Q(s, a)$  de forma mais frequente.

# Q-Learning

- Uma forma de representar a q-função sendo aprendida durante o treinamento é usar uma **tabela**.
  - Cada entrada armazena a utilidade de um q-estado.
  - Entradas são iniciadas com zero.


Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	327	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

# Q-Learning

-  O treinamento corresponde a atualizar gradativamente as entradas dessa tabela para cada transição  $(s, a, r, s')$  experimentada.

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	327	0	0	0	0	0	0
	.	.	.	.	.	.	.
States	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
States	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.

# Q-Learning

- ❏ O treinamento corresponde a atualizar gradativamente as entradas dessa tabela para cada transição  $(s, a, r, s')$  experimentada.
- ❏ Teorema: após uma quantidade grande de episódios, o Q-learning converge para os q-valores ótimos.



# Q-Learning

- ❑ Inicia com q-valores iguais a zero:  $Q(s, a) \leftarrow 0, \forall (s, a)$
- ❑ A cada transição  $(s, a, r, s')$  que o agente experimenta, ele atualiza a entrada  $(s, a)$  da q-função (q-tabela).
- ❑ Regra de atualização:

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$



# Regra de atualização

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

# Taxa de aprendizado (*learning rate*)

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$



# Taxa de aprendizado (*learning rate*)

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

- Constante para controlar a importância relativa entre dois aspectos:
  - O que o agente já aprendeu sobre um determinado estado-ação.
  - O que aconteceu na transição mais recente envolvendo este estado-ação.

# Taxa de aprendizado (*learning rate*)

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

- ❑ O que acontece se  $\alpha = 0$ ?
- ❑ O que acontece se  $\alpha = 1$ ?

Em ambos os casos, não há aprendizado!

# Fator de desconto

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$



# Fator de desconto

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

- Constante que determina a importância das recompensas futuras ( $0 \leq \gamma \leq 1$ ).
  - Um fator de 0 faz com que o agente priorize apenas as recompensas atuais ( $r$  na regra de atualização).
  - Um fator que se aproxima de 1 faz com que ele se esforce por uma alta recompensa de longo prazo.

# Regra de atualização – interpretação

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

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

# Regra de atualização – interpretação

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

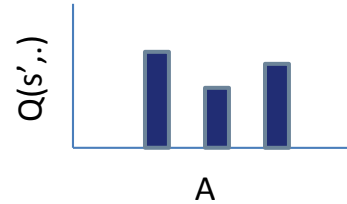
$$v = r + \gamma \max_{a'} Q(s', a')$$

$$Q_{new}(s, a) \leftarrow Q_{old}(s, a) + \alpha[v - Q_{old}(s, a)]$$

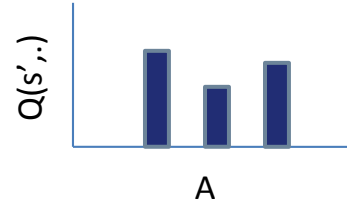


# Passo de atualização

Suponha que o agente está no estado  $s$ .  
Nesse estado, ele deve executar uma ação.

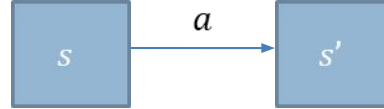
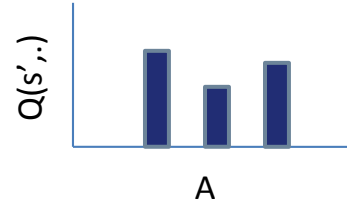


# Passo de atualização



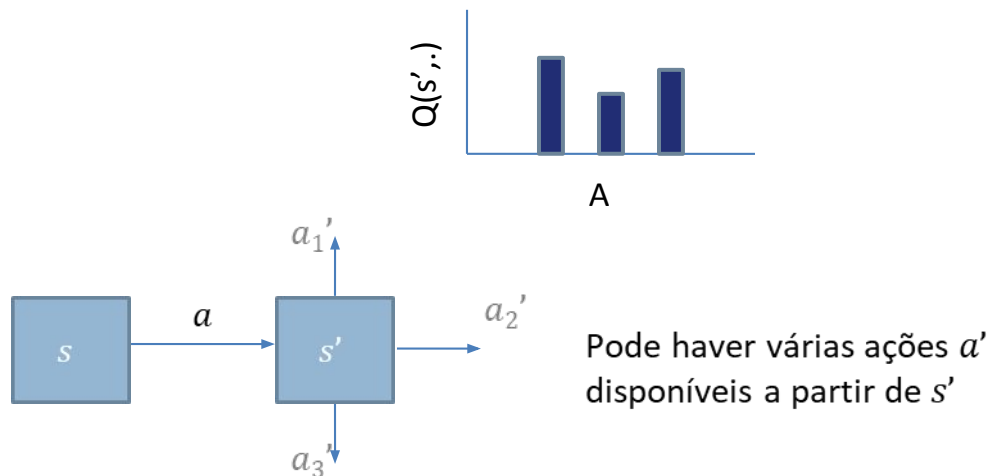
1. Em  $s$ , agente seleciona e executa uma ação  $a$

# Passo de atualização



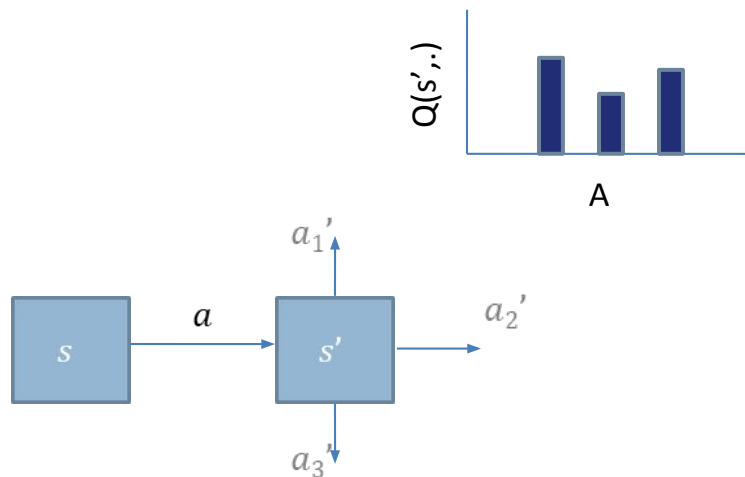
1. Em  $s$ , agente seleciona e executa uma ação  $a$
2. Agente observa recompensa  $r$  e novo estado  $s'$

# Passo de atualização



1. Em  $s$ , agente seleciona e executa uma ação  $a$
2. Agente observa recompensa  $r$  e novo estado  $s'$
3. Agente obtém (da q-tabela) os valores  $Q(s', a')$  para toda ação  $a'$  disponível a partir de  $s'$

# Passo de atualização



1. Em  $s$ , agente seleciona e executa uma ação  $a$
2. Agente observa recompensa  $r$  e novo estado  $s'$
3. Agente obtém (da q-tabela) os valores  $Q(s', a')$  para toda ação  $a'$  disponível a partir de  $s'$

$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

# Passo de atualização – exemplo 1

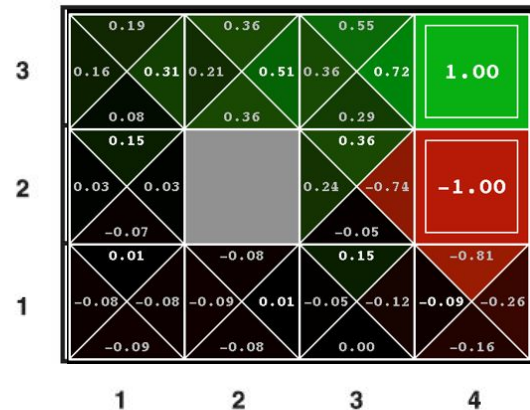
Considera:

▣  $\alpha = 0.1$

▣  $\gamma = 0.9$

▣ Experiência:  $([3,2], \text{Up}, -0.04, [3,3])$

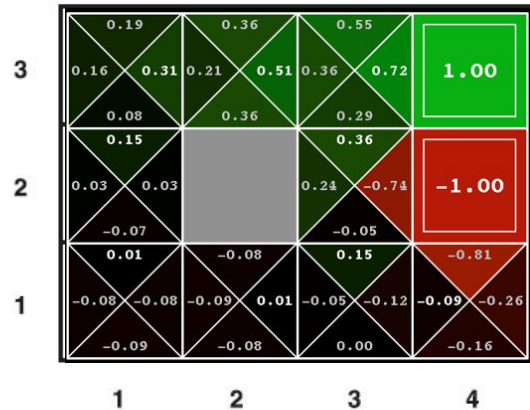
▣ Compute  $Q_{\text{new}}([3,2], \text{Up})$ .



$$Q_{\text{new}}(s, a) \leftarrow (1 - \alpha)Q_{\text{old}}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

## Passo de atualização – exemplo 2

- Consider:
  - $\alpha = 0.1$
  - $\gamma = 0.9$
  - Experiência:  $([3,2], \text{Up}, -0.04, [4,2])$
- Compute  $Q_{new}([3,2], \text{Up})$ .



$$Q_{new}(s, a) \leftarrow (1 - \alpha)Q_{old}(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$$

# Q-Learning (pseudocódigo)

## Inputs

$S$  is a set of states  
 $A$  is a set of actions  
 $\gamma$  the discount  
 $\alpha$  is the step size

## Local

real array  $Q[S,A]$   
previous state  $s$   
previous action  $a$   
initialize  $Q[S,A]$  arbitrarily  
observe current state  $s$   
**repeat**  
    select and carry out an action  $a$   
    observe reward  $r$  and state  $s'$   
     $Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$   
     $s \leftarrow s'$  **until** termination



# Q-Learning - convergência

- **Resultado importante (teorema):** q-valores que o agente aprende com o Q-learning convergem para política ótima.
  - ▣ Mesmo se o agente agir sub-otimamente no início do aprendizado!
  - ▣ Basicamente, no limite, não importa de que forma o agente seleciona ações (!)
- **Requisitos:**
  - ▣ O agente tem que **explorar** bastante o ambiente.
  - ▣ O agente deve eventualmente diminuir a taxa de aprendizado  $\alpha$ .
    - ... mas não decrescer muito rapidamente.