

Inteligência Artificial

INF01048 - 2019/I

Prof. Bruno Castro da Silva

Aula 05

Busca local e em espaços contínuos

Problemas de otimização

Hill Climbing (Método da Subida de Encosta)

Simulated Annealing (Método da Têmpera Simulada)

Última aula

- **Problemas de Satisfação de Restrições**
 - *resolução de problemas com restrições*
 - problema formulado em termos de **variáveis, domínios, restrições**

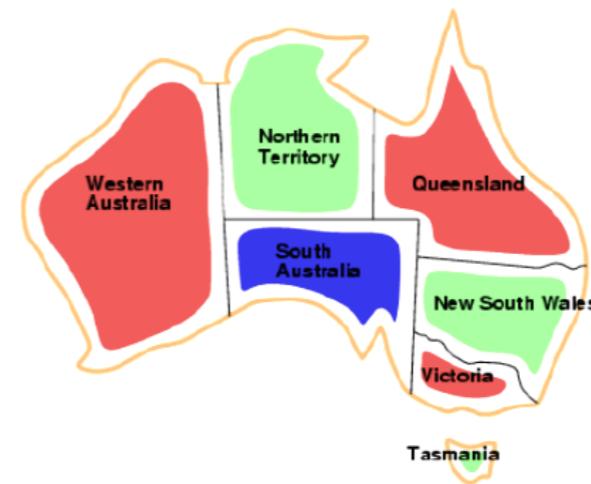
Última aula

- **Problemas de Satisfação de Restrições**
 - *resolução de problemas com restrições*
 - problema formulado em termos de variáveis, domínios, restrições
 - estado: valores atribuídos a um subconjunto de variáveis;
 - arcos: atribuição de um valor a uma das variáveis ainda não atribuídas;
 - teste se objetivo foi alcançado: teste se todas restrições são respeitadas;

Última aula

- **Problemas de Satisfação de Restrições**
 - *resolução de problemas com restrições*
 - problema formulado em termos de variáveis, domínios, restrições
 - estado: valores atribuídos a um subconjunto de variáveis;
 - arcos: atribuição de um valor a uma das variáveis ainda não atribuídas;
 - teste se objetivo foi alcançado: teste se todas restrições são respeitadas;
- Busca com Retrocesso
 - Busca em Profundidade; uma variável atribuída a cada passo

5	3		7			
6			1	9	5	
	9	8				6
8			6			3
4		8	3			1
7			2			6
	6			2	8	
		4	1	9		5
			8		7	9



Última aula

- Ordem em que variáveis e valores são consideradas é importante
 - Seleção de variável: *Valores Restantes Mínimos & Heurística do Maior Grau*
 - Seleção de ordem de valores: *Heurística do Valor Menos Restritivo*

Última aula

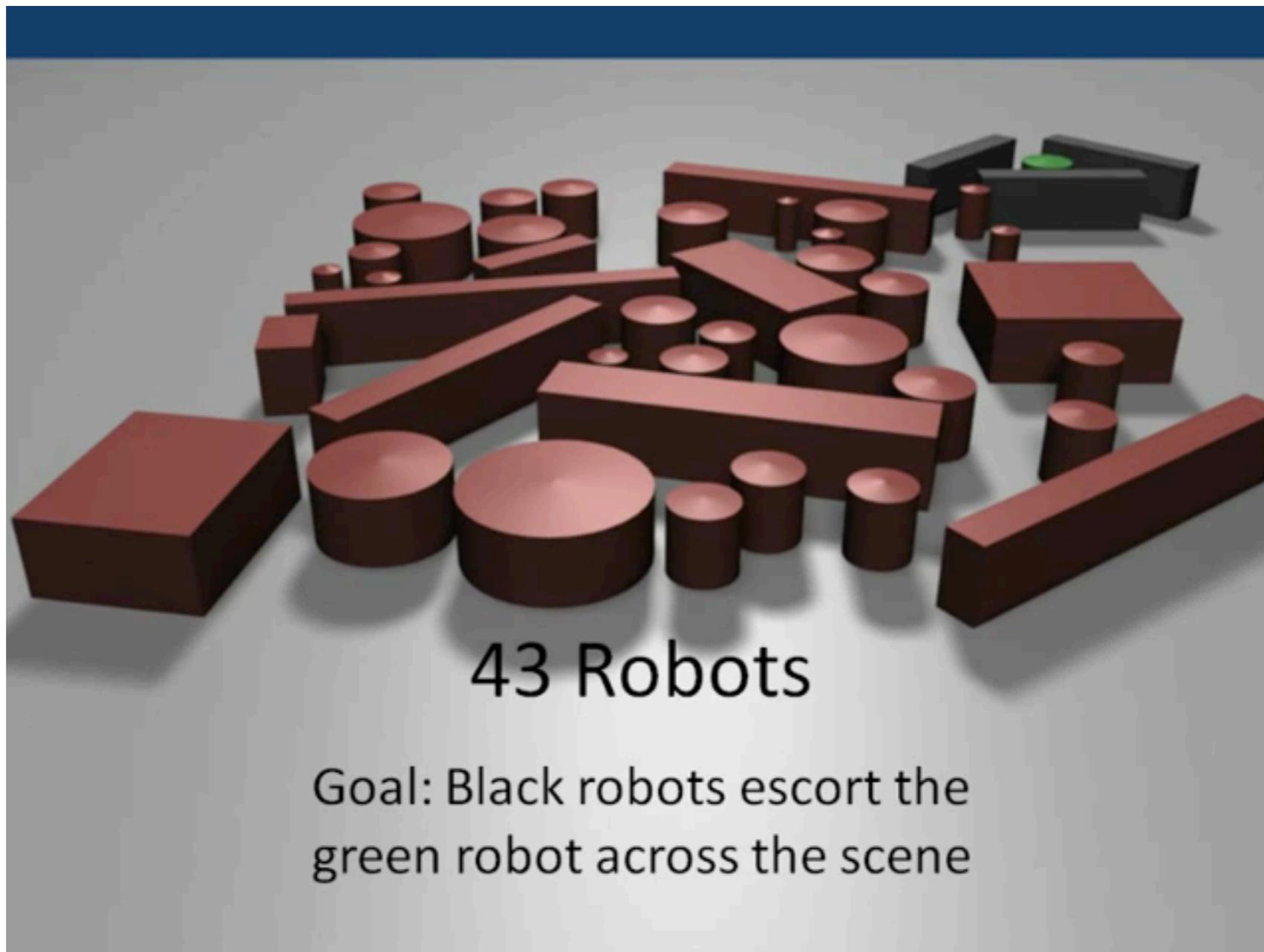
- Ordem em que variáveis e valores são consideradas é importante
 - Seleção de variável: *Valores Restantes Mínimos & Heurística do Maior Grau*
 - Seleção de ordem de valores: *Heurística do Valor Menos Restritivo*
- **Verificação Prévia (Forward Checking)**
 - incrementalmente constrói listas de valores permitidos para cada variável não atribuída
 - evita atribuições que levam a conflitos futuros
- **Propagação de Restrições (e.g. consistência de arcos)**
 - detecta inconsistências não capturadas por Verificação Prévia

Última aula

- Ordem em que variáveis e valores são consideradas é importante
 - Seleção de variável: *Valores Restantes Mínimos & Heurística do Maior Grau*
 - Seleção de ordem de valores: *Heurística do Valor Menos Restritivo*
- Verificação Prévia (*Forward Checking*)
 - incrementalmente constrói listas de valores permitidos para cada variável não atribuída
 - evita atribuições que levam a conflitos futuros
- Propagação de Restrições (e.g. *consistência de arcos*)
 - detecta inconsistências não capturadas por Verificação Prévia
- **Ao contrário de heurísticas usadas, p.ex., em A*, as acima se aplicam a todos PSR's!**

Última aula - PSR (exemplo)

Coordenação Dinâmica de Times de Robôs

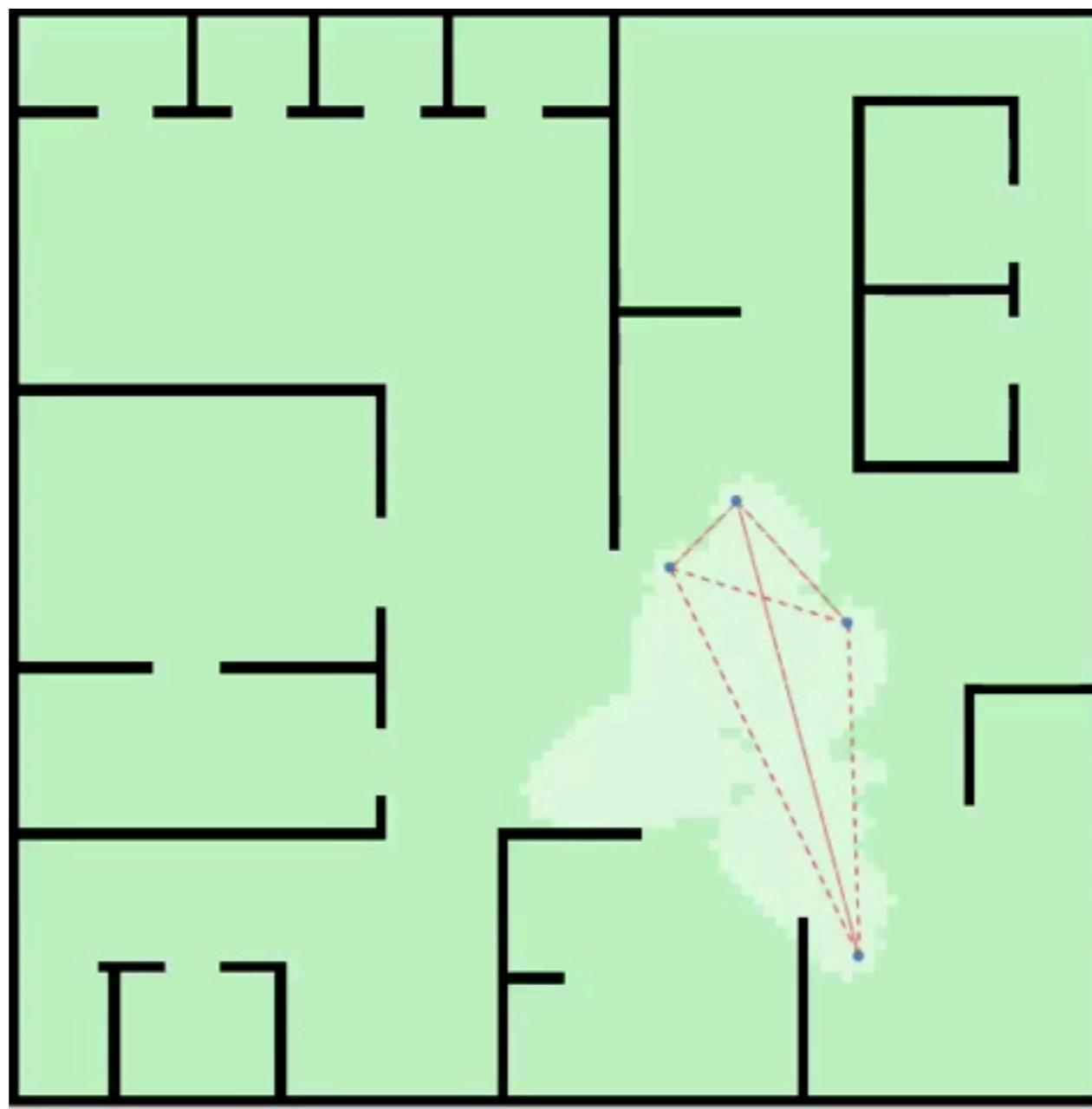


Multi-robot Coordination Using Generalized Social Potential Fields (ICRA2009)

<https://www.youtube.com/watch?v=Jsw9lc7UASQ>

Última aula - PSR (exemplo)

Coordenação Dinâmica de Times de Robôs



Distributed constraint reasoning applied to multi-robot exploration (ICTAI2009)

<https://www.youtube.com/watch?v=w8sRUHduuP0>

Busca com Retrocesso

- **Busca_Com_Retrocesso(PSR)**
 - retorna Retrocesso-Recursivo(PSR, {})
- **Retrocesso_Recursivo(PSR, atribuição)**
 - Se atribuição é completa, Retorna atribuição
 - $var \leftarrow \text{SELEÇÃO_VARIAVEL_NÃO_ATRIBUÍDA}(\text{PSR.vars})$
 - Para cada $valor$ em $\text{ORDENA_VALORES_DO_DOMÍNIO}(var)$
 - Se $valor$ é consistente com PSR.restricções
 - $\text{atribuição} \leftarrow \text{atribuição} \cup \{var = valor\}$
 - $\text{resultado} \leftarrow \text{Retrocesso_Recursivo}(\text{PSR}, \text{atribuição})$
 - Se $\text{resultado} \neq \text{Falha}$
 - Retorna resultado
 - Senão
 - remove $\{var = valor\}$ de atribuição // irá tentar outro valor na próxima iteração do loop
 - Retorna Falha

Busca com Retrocesso

- **Busca_Com_Retrocesso(PSR)**

- retorna Retrocesso-Recursivo(PSR, {})

- **Retrocesso_Recursivo(PSR, atribuição)**

- Se atribuição é completa, Retorna atribuição
 - $var \leftarrow \text{SELEÇÃO_VARIÁVEL_NÃO_ATRIBUÍDA}(\text{PSR.vars})$
 - Para cada valor em $\text{ORDENA_VALORES_DO_DOMÍNIO}(var)$
 - Se valor é consistente com PSR.restricções
 - atribuição $\leftarrow \text{atribuição} \cup \{var = valor\}$
 - resultado $\leftarrow \text{Retrocesso_Recursivo}(\text{PSR}, \text{atribuição})$
 - Se resultado \neq Falha
 - Retorna resultado
 - Senão
 - remove $\{var = valor\}$ de atribuição // irá tentar outro valor na próxima iteração do loop
 - Retorna Falha

**Busca
sistêmatica
em um grafo**

Busca Sistemática em Grafos

- Busca em largura/profundidade/aprofundamento iterativo/A*/PSR
 - todos exemplos de algoritmos de busca em grafo
 - fazem a busca sistemática em um espaço de estados na busca pelo estado objetivo

Busca Sistemática em Grafos

- Busca em largura/profundidade/aprofundamento iterativo/A*/PSR
 - todos exemplos de algoritmos de busca em grafo
 - fazem a busca sistemática em um espaço de estados na busca pelo estado objetivo
- Métodos de busca sistemática em grafo mantêm em memória:
 - quais alternativas/estados já foram explorados/visitados
 - p.ex., através das listas *nodos_fechados* e *nodos_abertos*

Busca Sistemática em Grafos

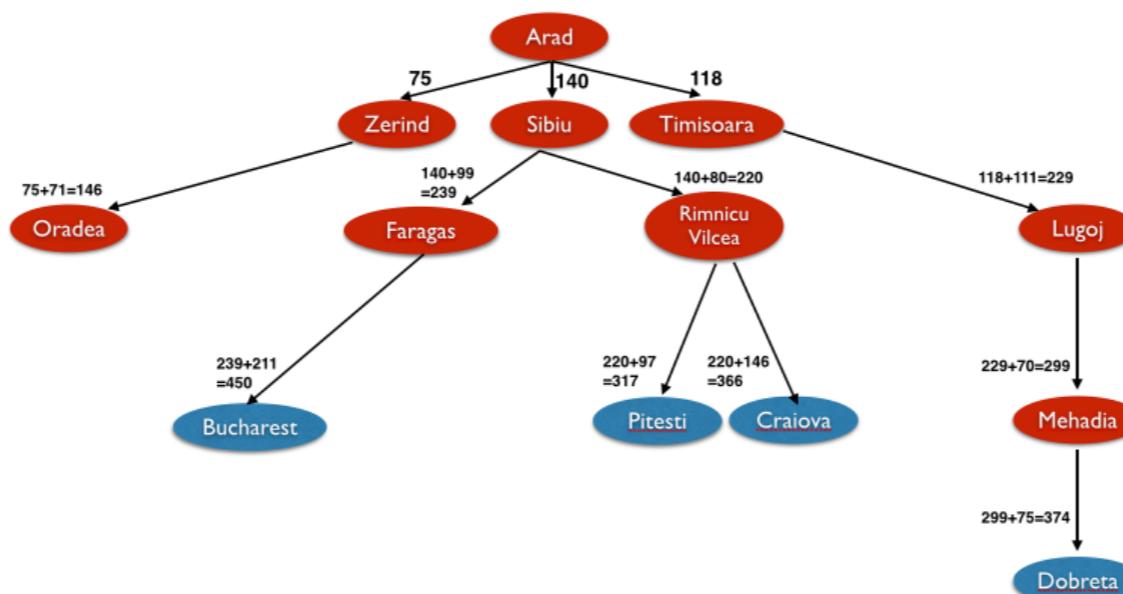
- Busca em largura/profundidade/aprofundamento iterativo/A*/PSR
 - todos exemplos de algoritmos de busca em grafo
 - fazem a busca sistemática em um espaço de estados na busca pelo estado objetivo
- Métodos de busca sistemática em grafo mantêm em memória:
 - quais alternativas/estados já foram explorados/visitados
 - p.ex., através das listas *nodos_fechados* e *nodos_abertos*
 - memória deste tipo auxilia a percorrer o espaço de busca de forma *sistemática*
 - até que uma solução seja encontrada

Busca Sistemática em Grafos

- Busca em largura/profundidade/aprofundamento iterativo/A*/PSR
 - todos exemplos de algoritmos de busca em grafo
 - fazem a busca sistemática em um espaço de estados na busca pelo estado objetivo
- Métodos de busca sistemática em grafo mantêm em memória:
 - quais alternativas/estados já foram explorados/visitados
 - p.ex., através das listas *nodos_fechados* e *nodos_abertos*
 - memória deste tipo auxilia a percorrer o espaço de busca de forma *sistemática*
 - até que uma solução seja encontrada
 - note que o caminho até a solução é importante
 - p.ex., a menor *rota* em um mapa; melhor seqüência de movimentos de um robô

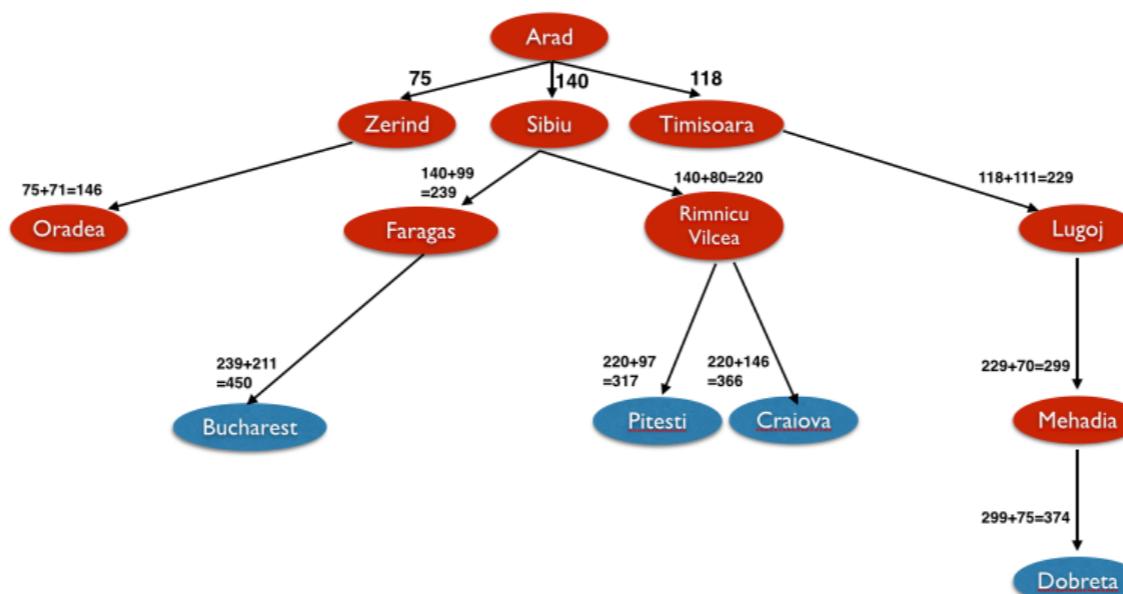
Busca Local

- Ao invés de manter em memória vários caminhos a serem explorados



Busca Local

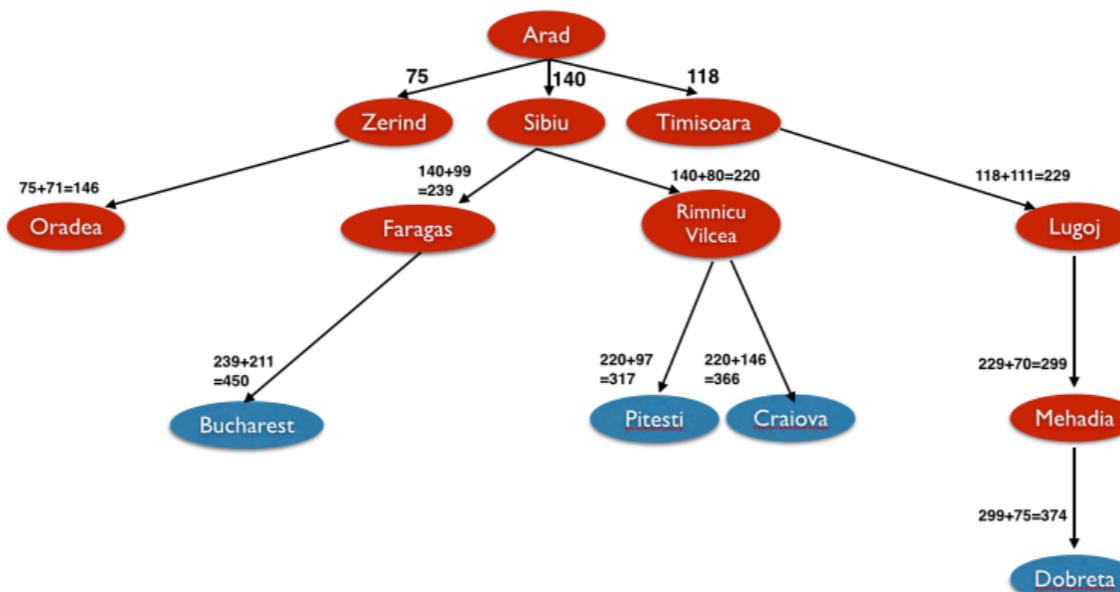
- Ao invés de manter em memória vários caminhos a serem explorados



mantém apenas o estado/solução potencial atual (e talvez seus vizinhos imediatos)

Busca Local

- Ao invés de manter em memória vários caminhos a serem explorados



mantém apenas o estado/solução potencial atual (e talvez seus vizinhos imediatos)

Motivação



problemas em que o caminho até uma solução não importa

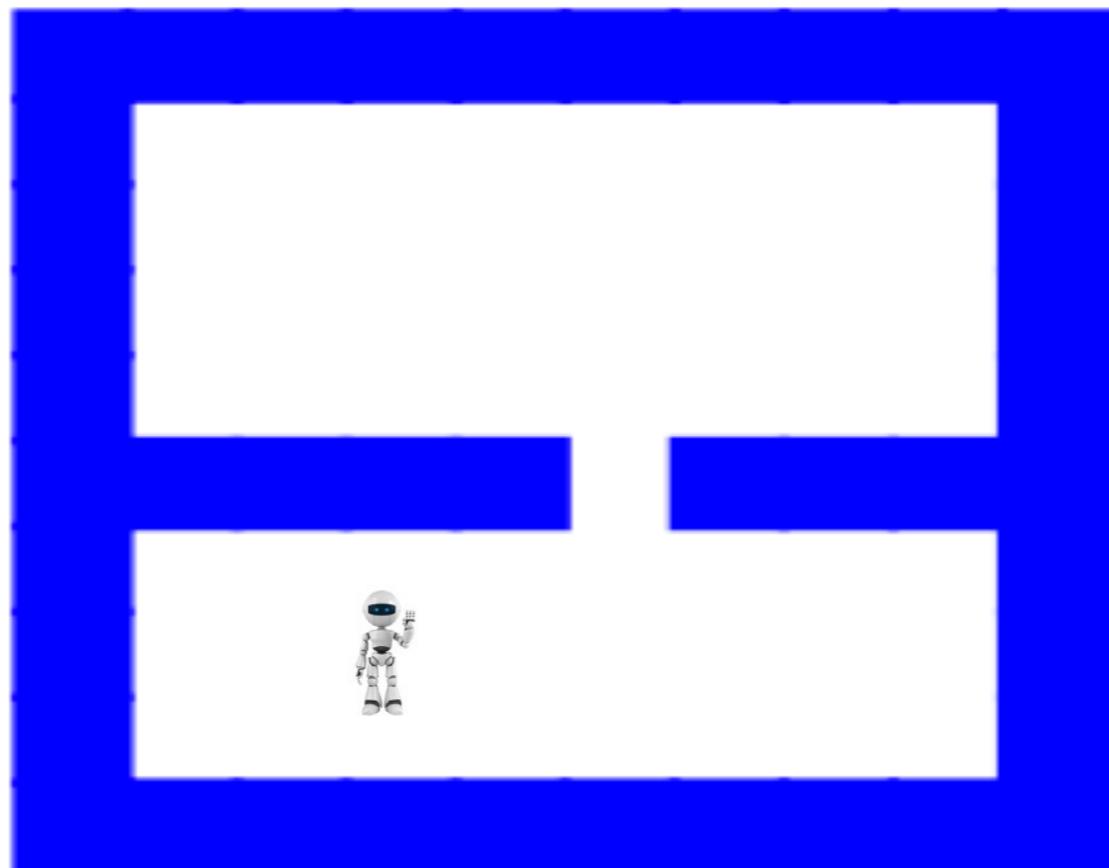
problemas de otimização

cada estado/solução potencial possui um valor; queremos encontrar a com maior valor

Espaço de Estados

Problema das Duas Salas

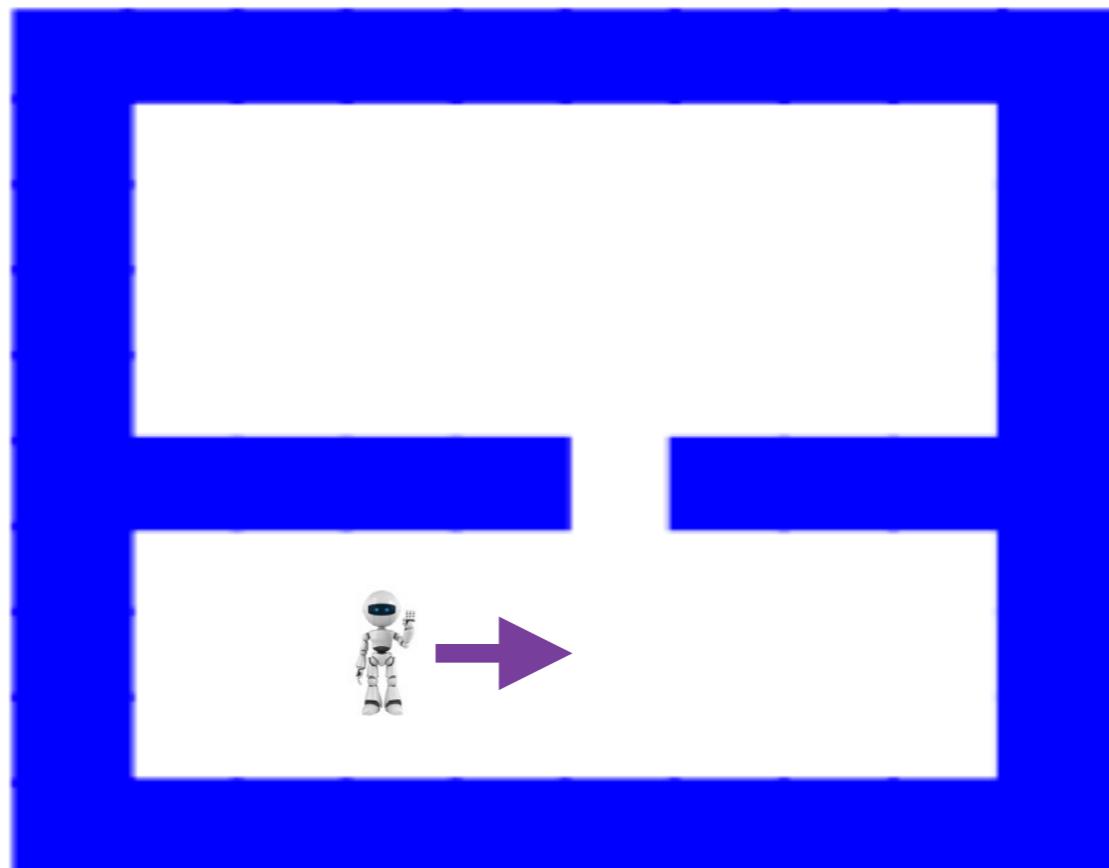
- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais



Espaço de Estados

Problema das Duas Salas

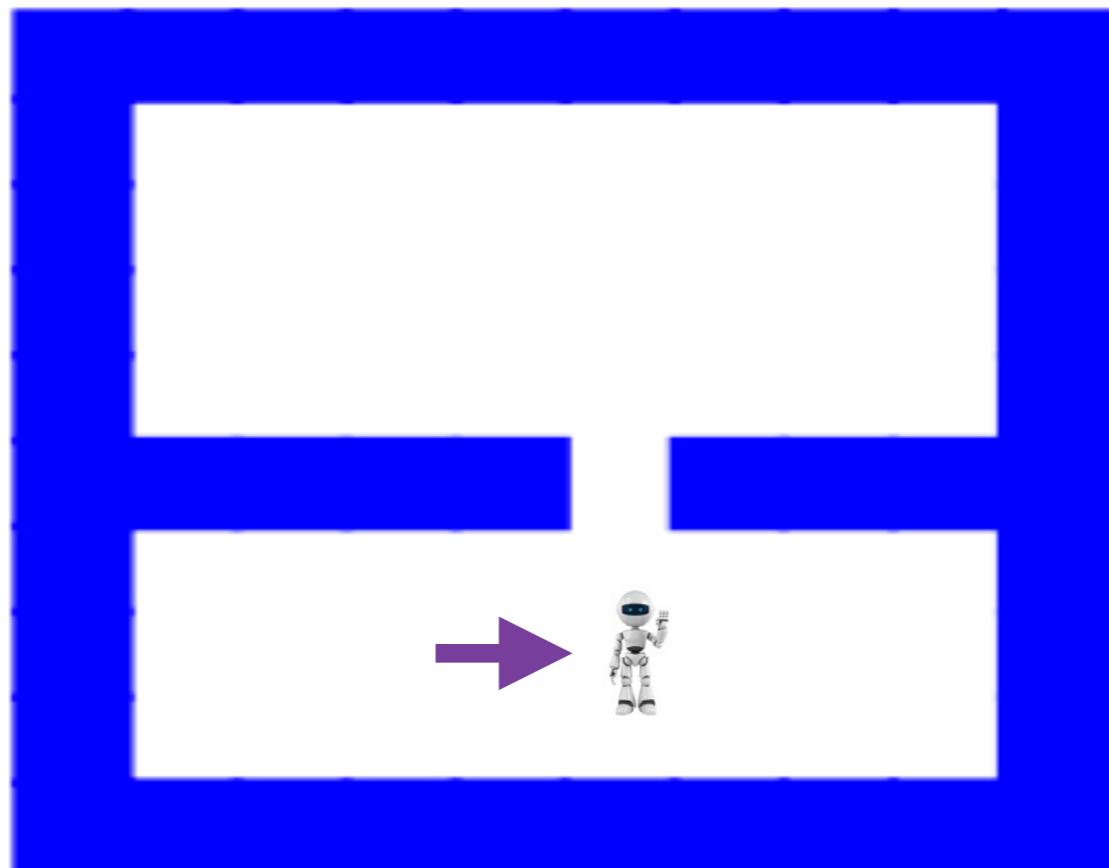
- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais



Espaço de Estados

Problema das Duas Salas

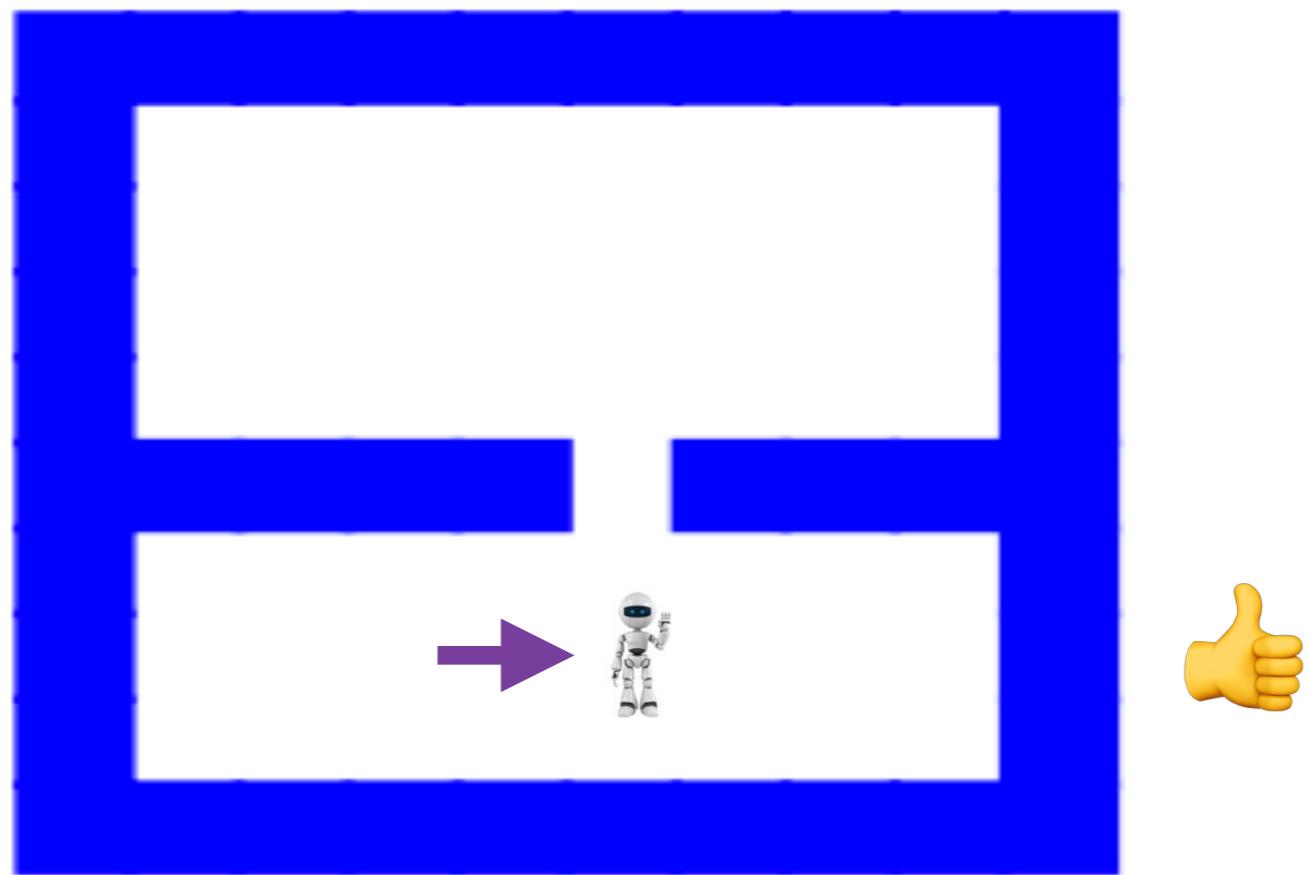
- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais



Espaço de Estados

Problema das Duas Salas

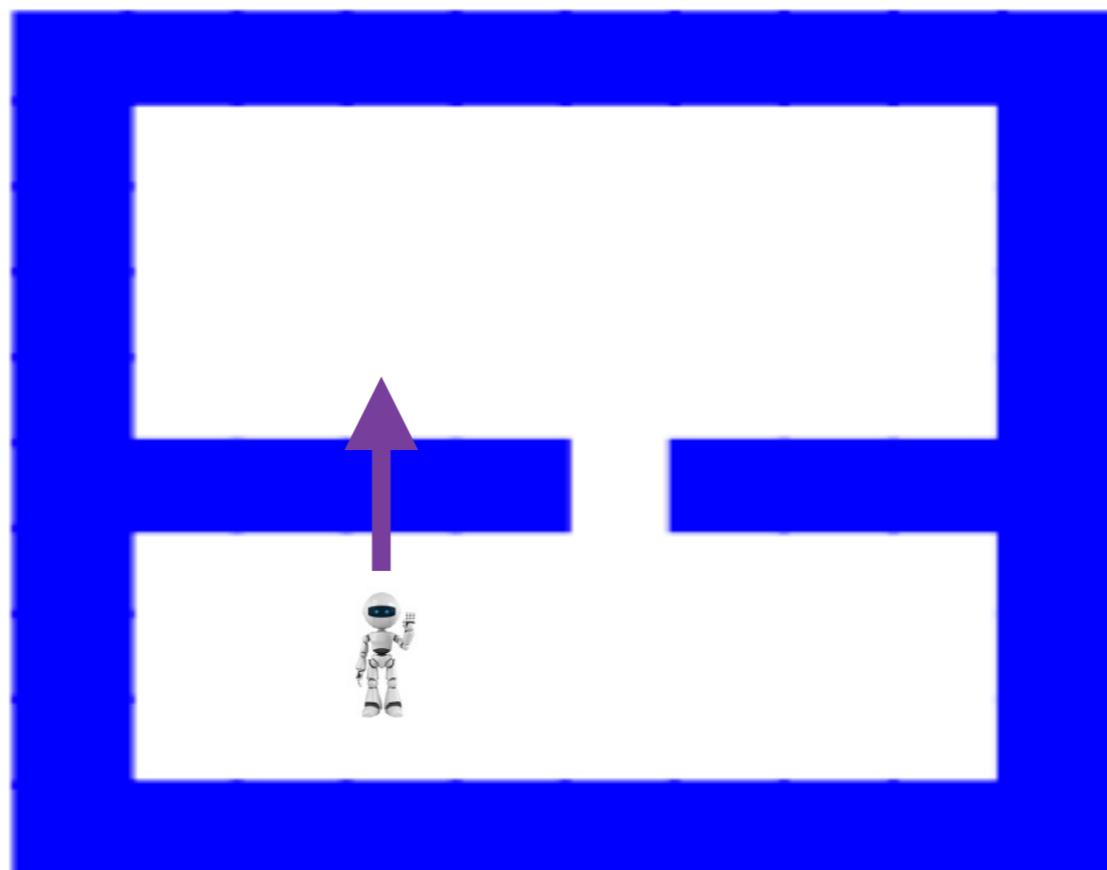
- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais



Espaço de Estados

Problema das Duas Salas

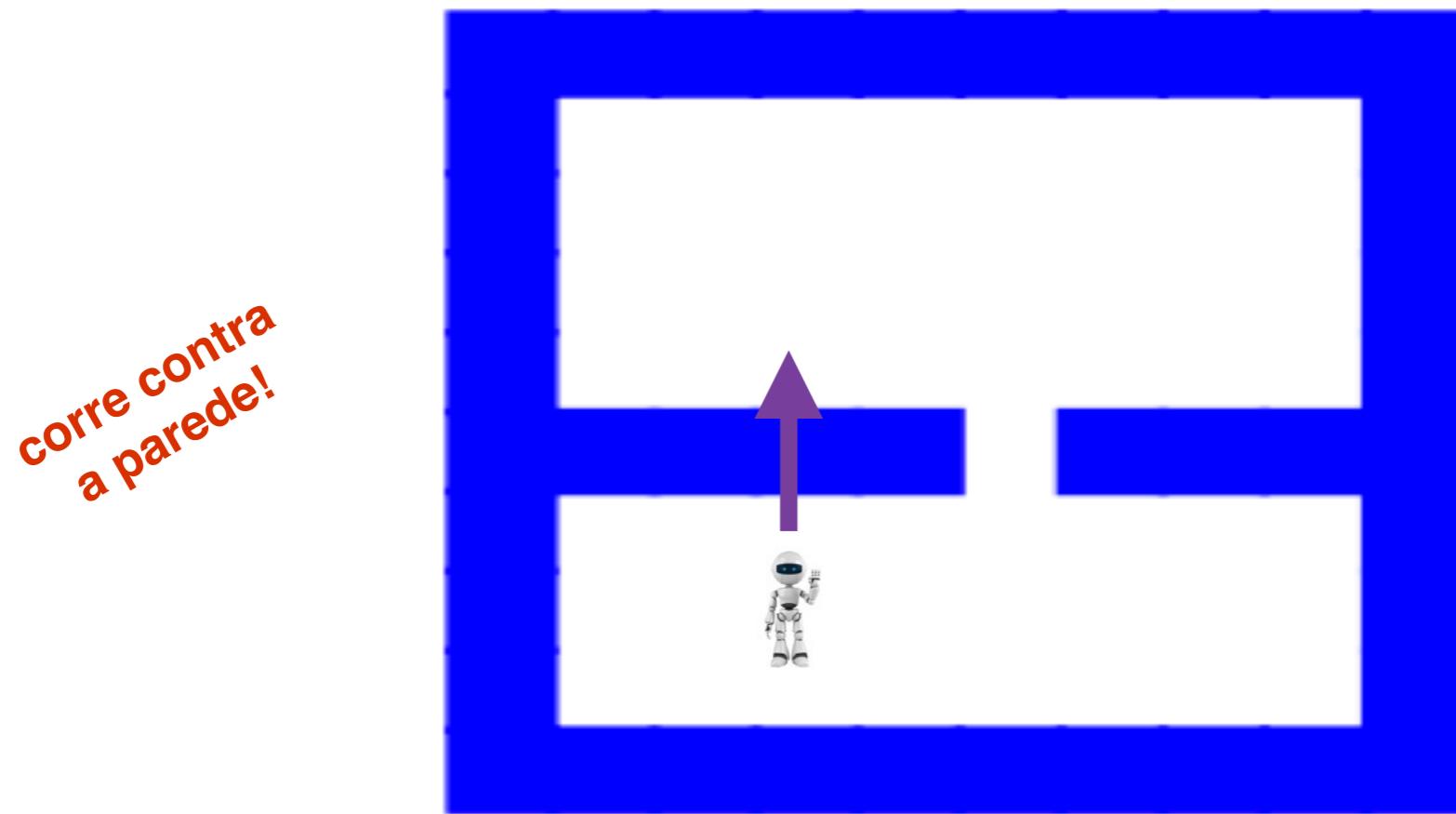
- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais



Espaço de Estados

Problema das Duas Salas

- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais



Espaço de Estados

Problema das Duas Salas

- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais

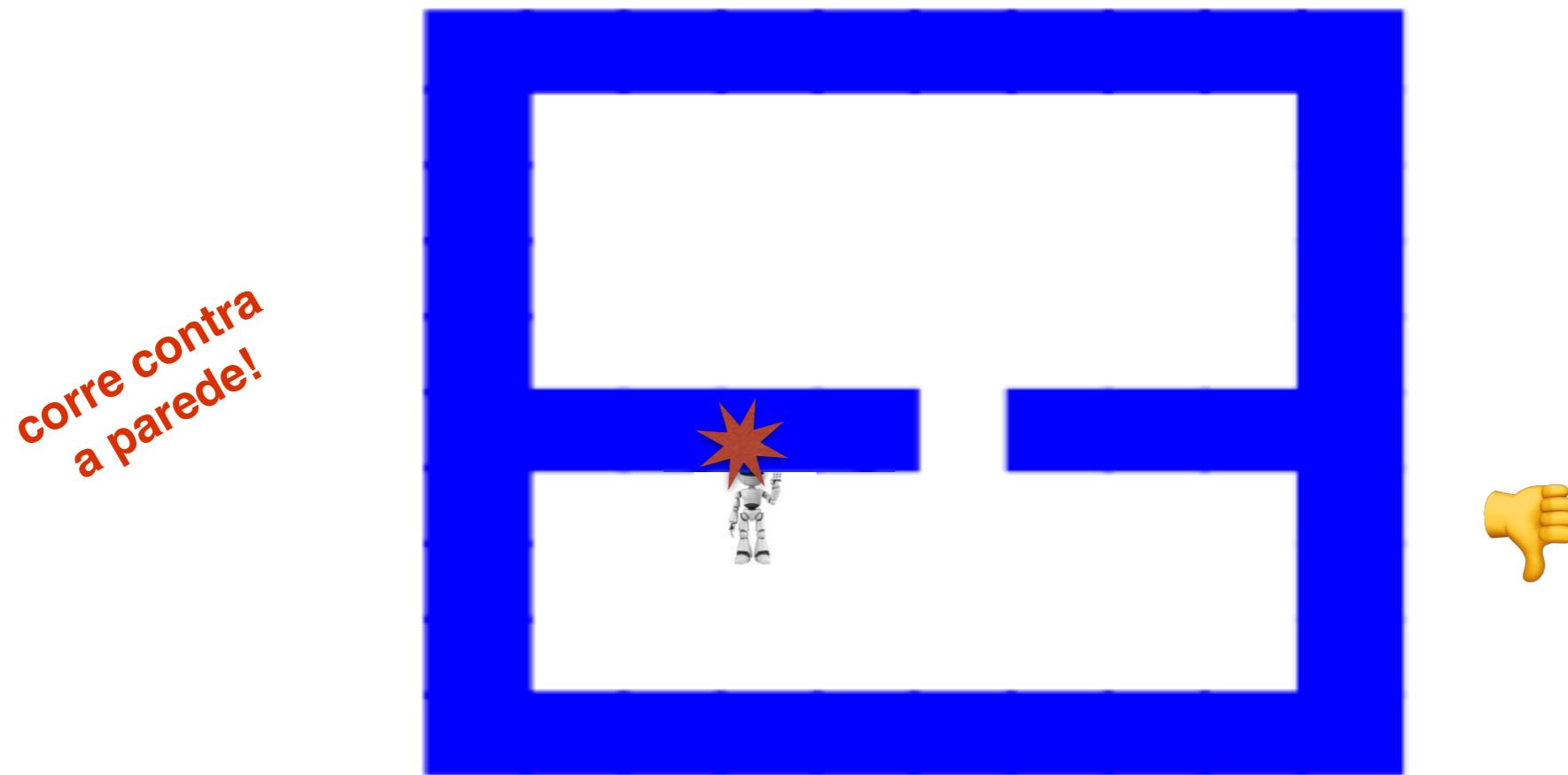
corre contra
a parede!



Espaço de Estados

Problema das Duas Salas

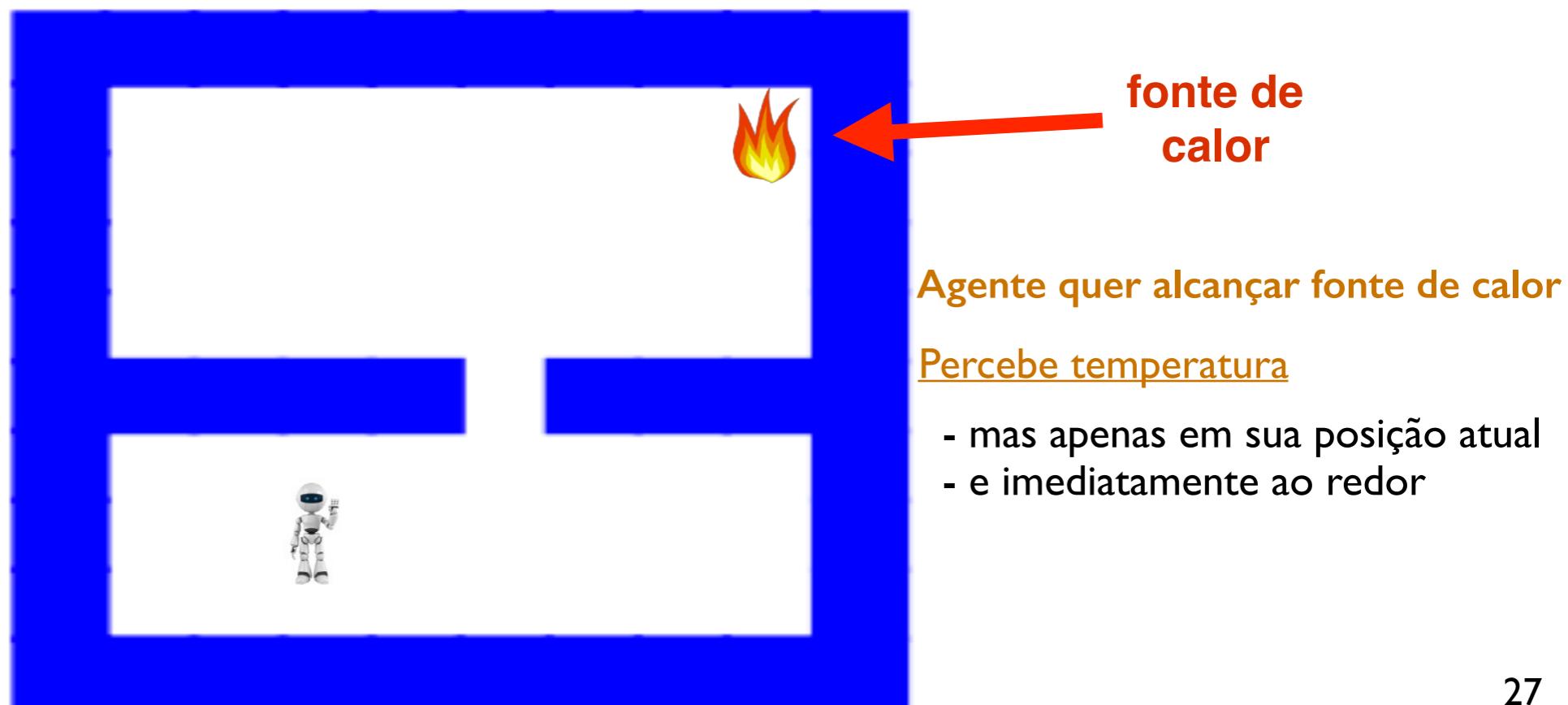
- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais



Espaço de Estados

Problema das Duas Salas

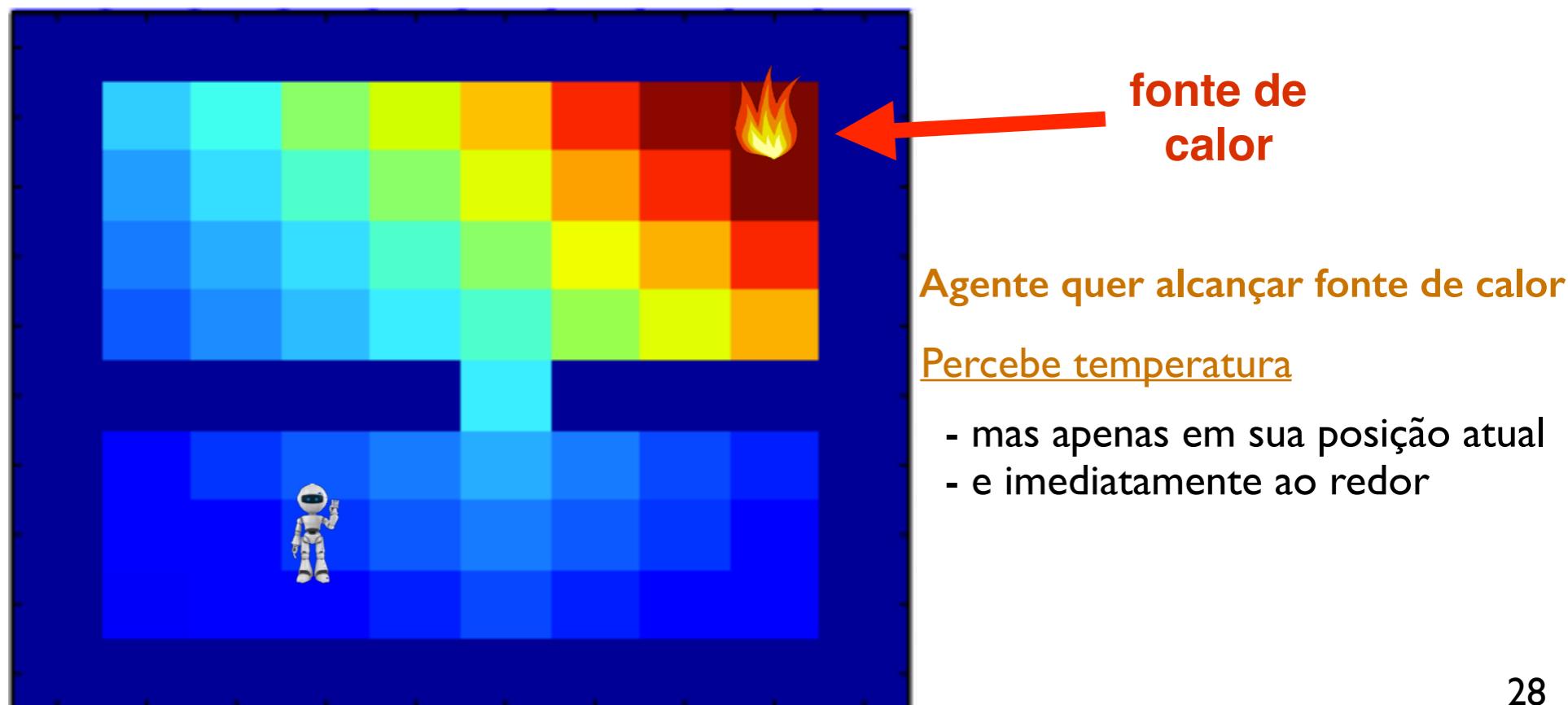
- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais

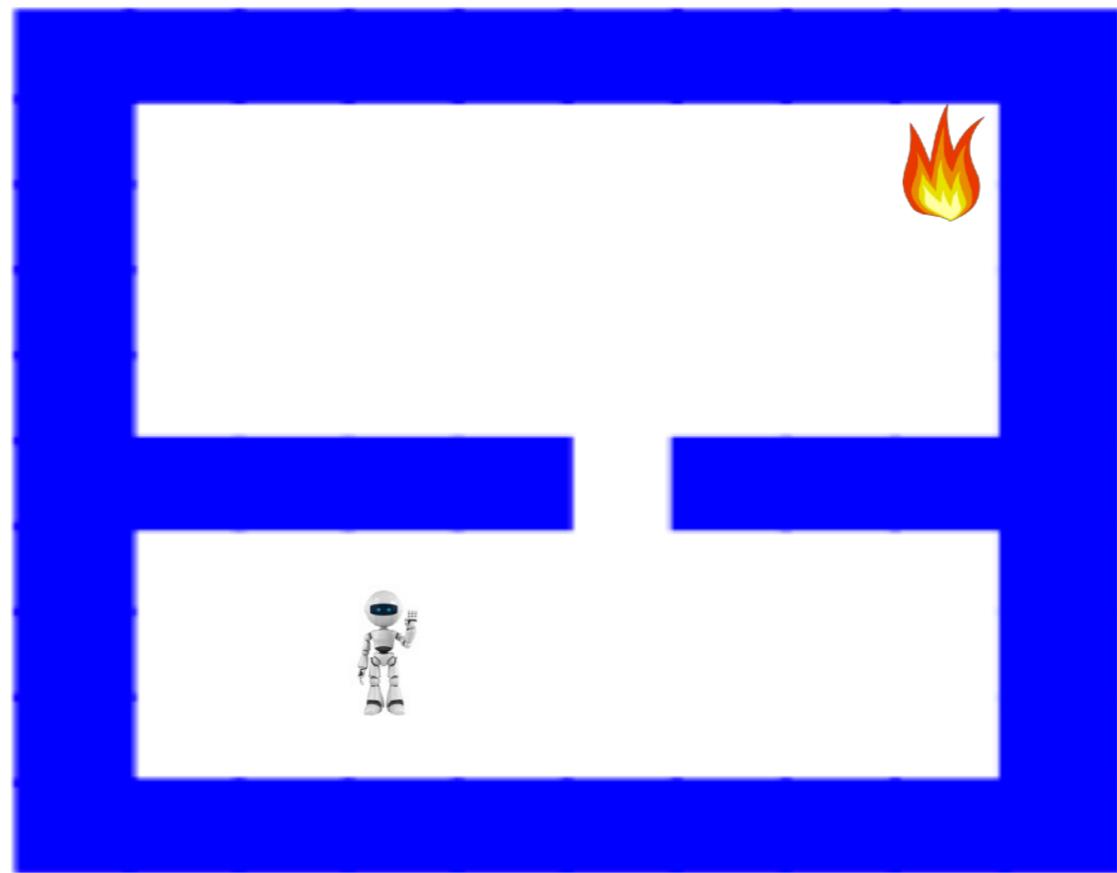


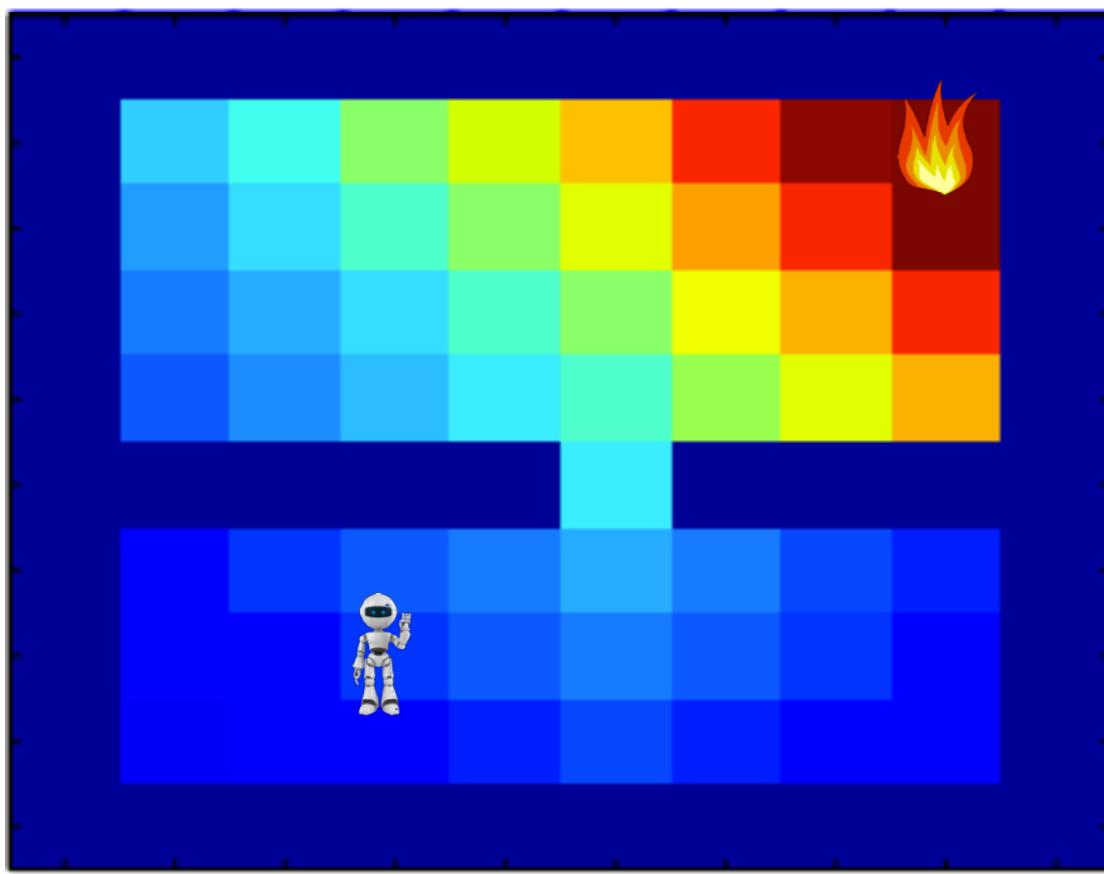
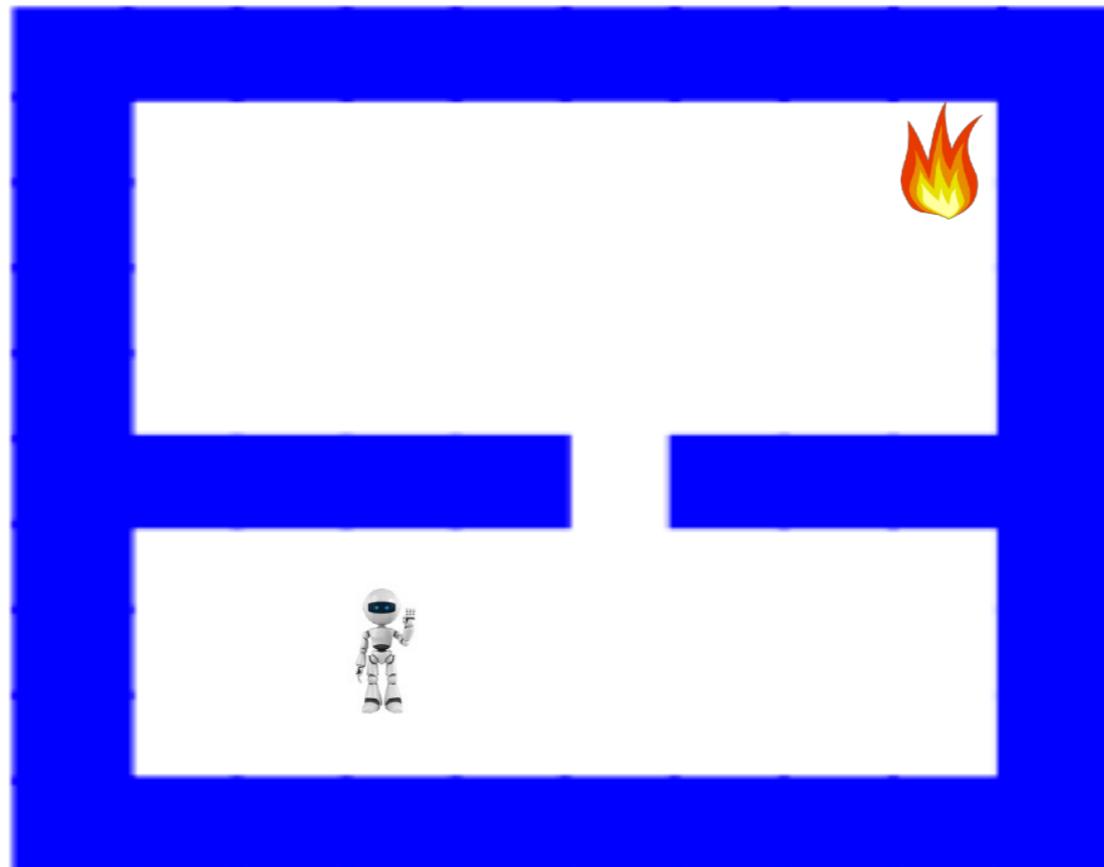
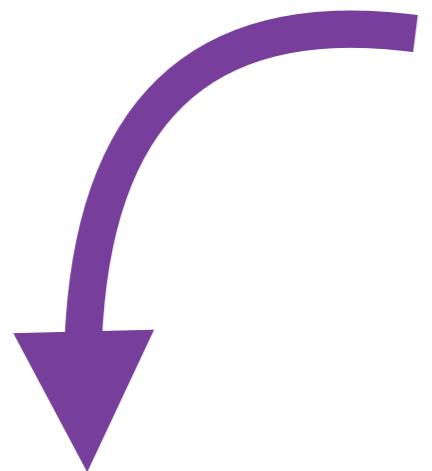
Espaço de Estados

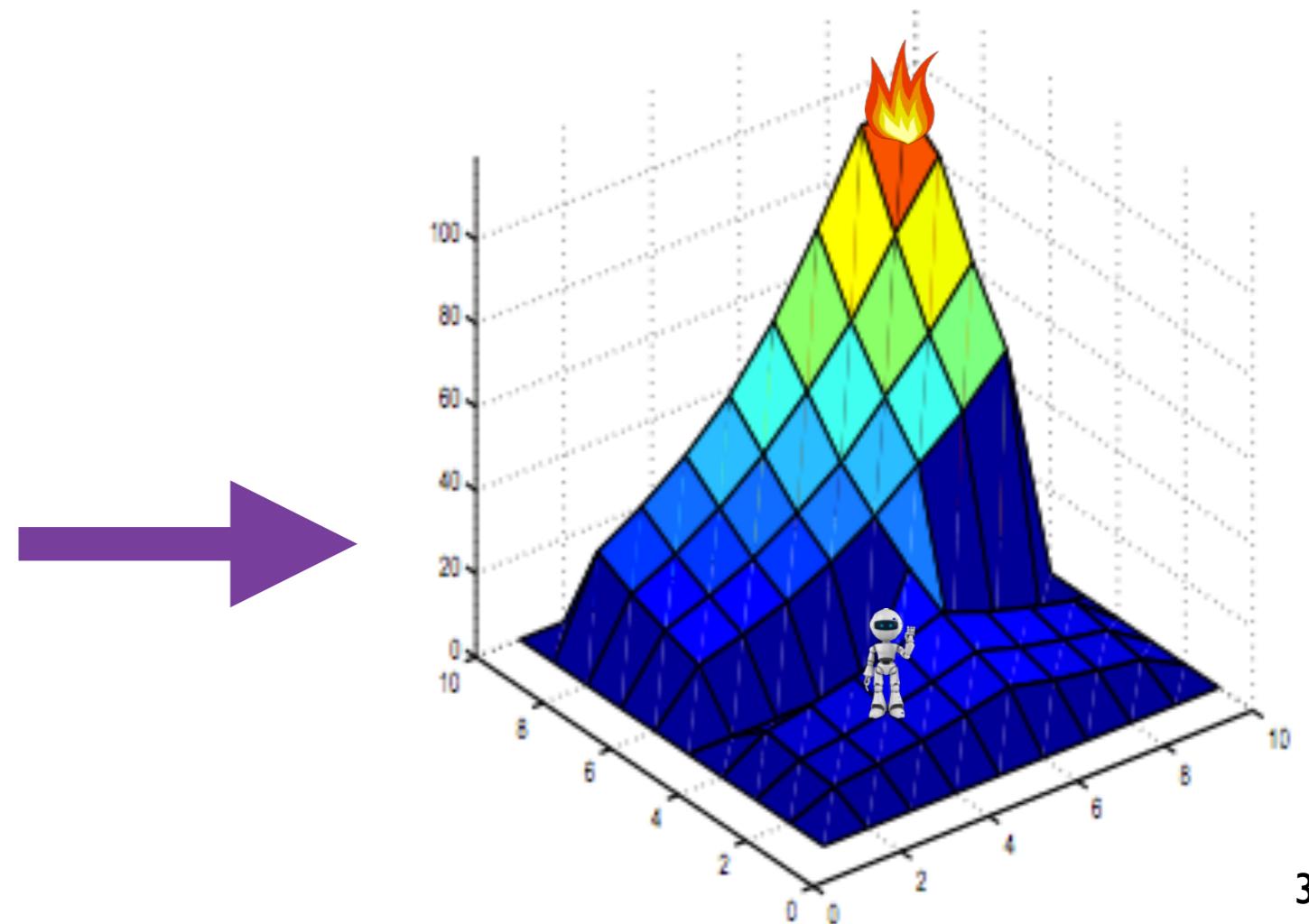
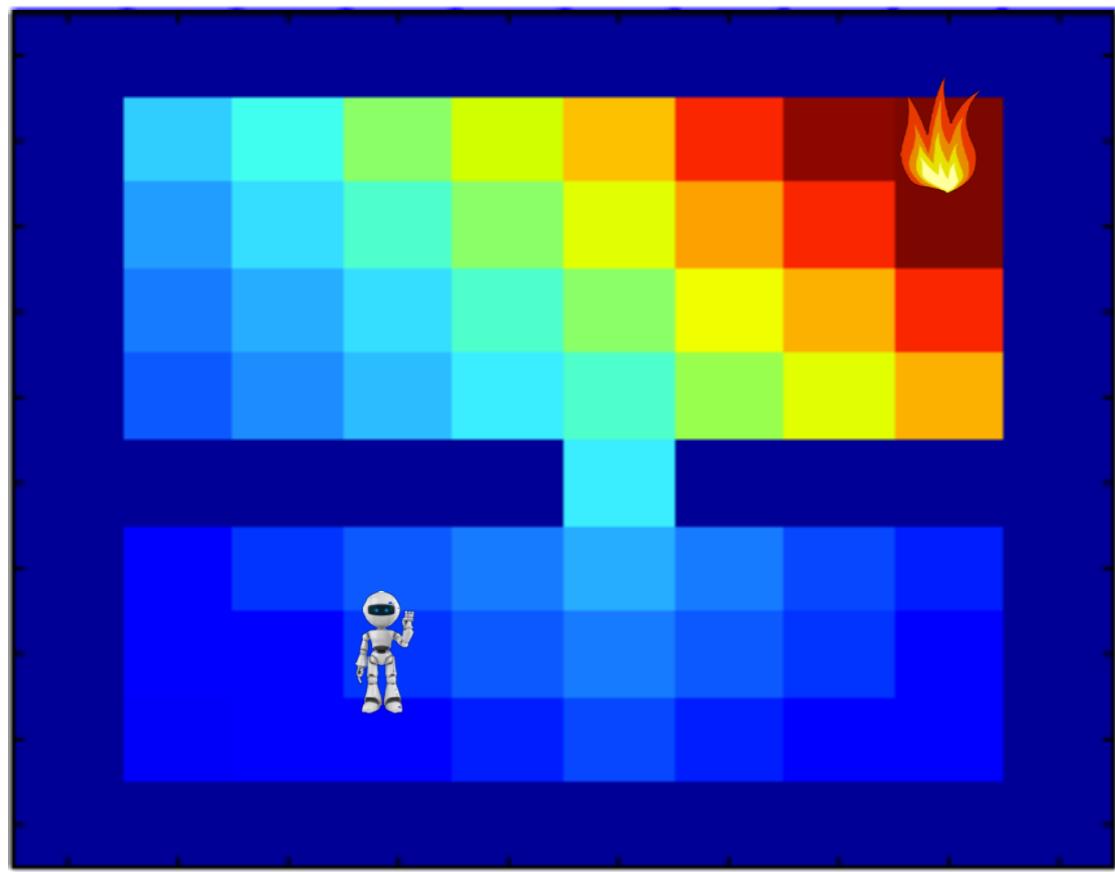
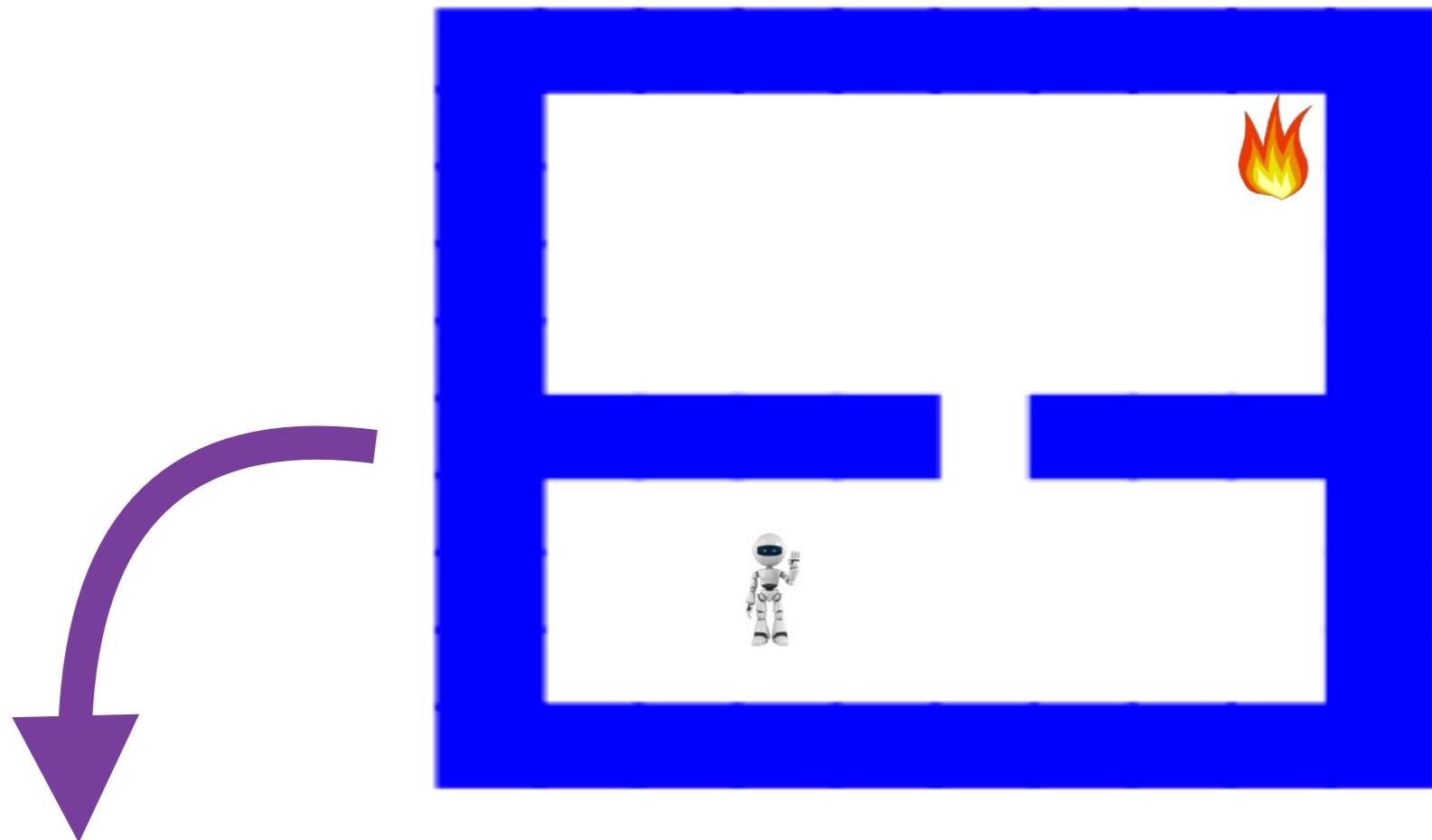
Problema das Duas Salas

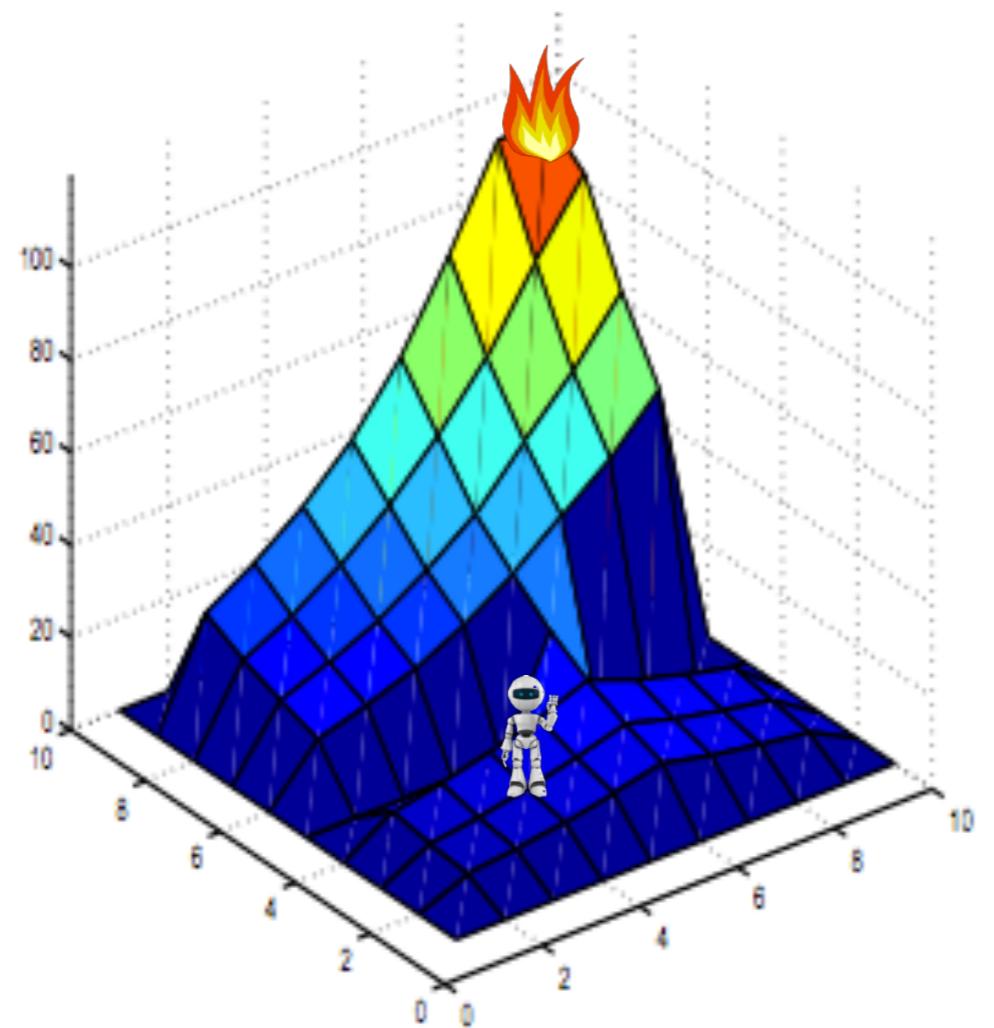
- Robô é colocado em um ambiente formado por duas salas
- Mas não conhece um mapa do ambiente!
 - Não tem acesso ao grafo especificando quais estados se conectam com quais

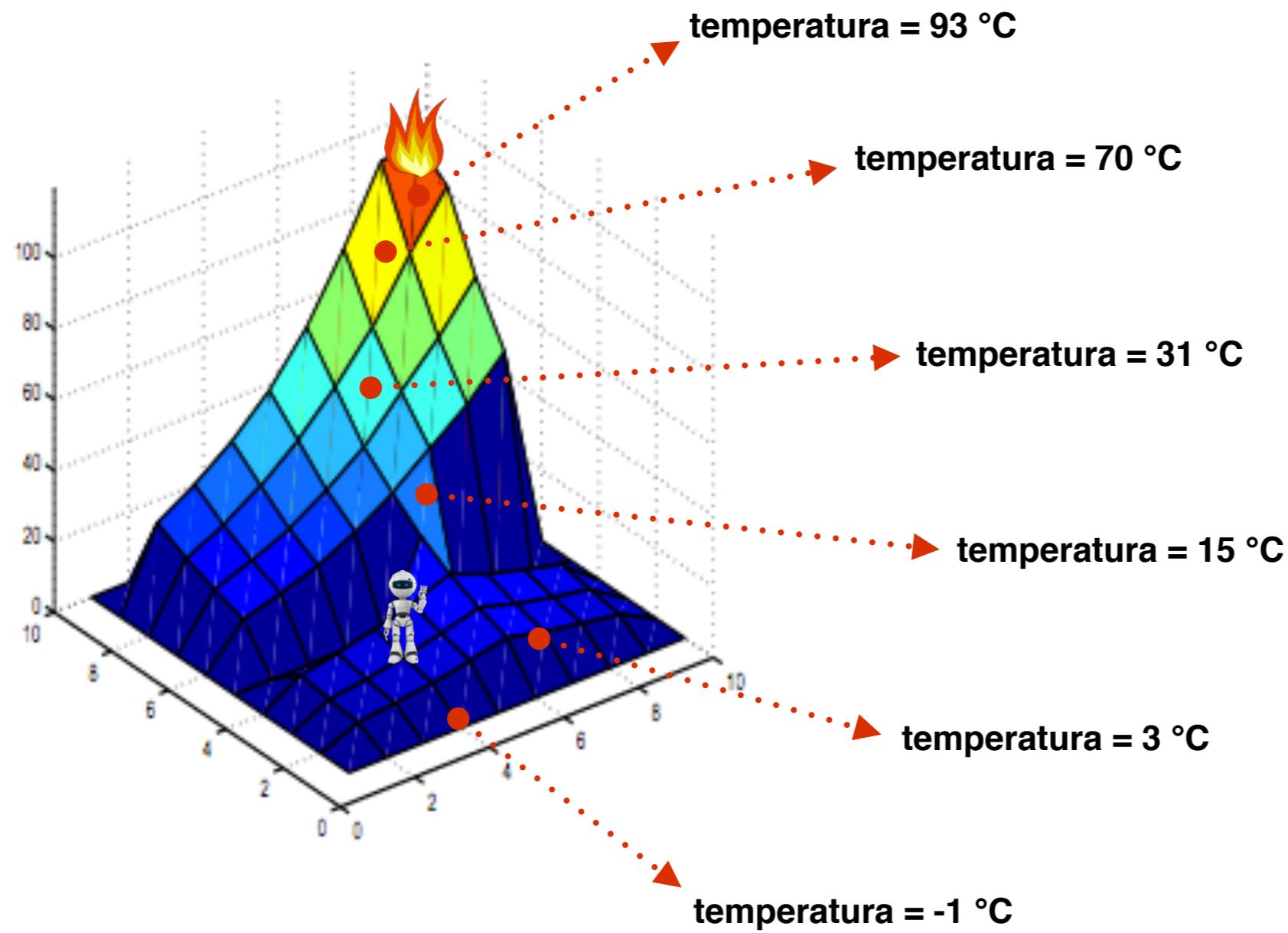


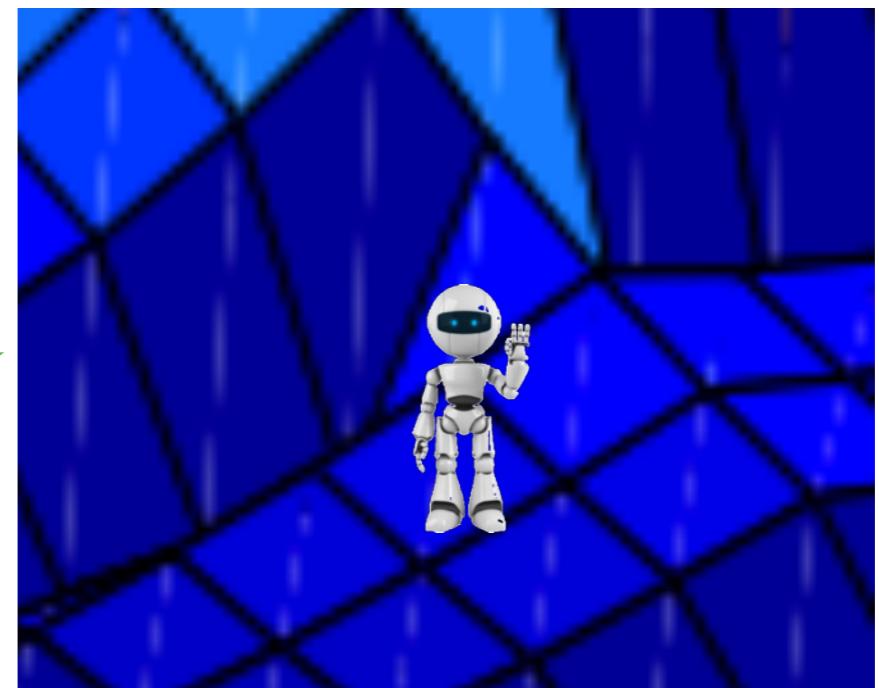
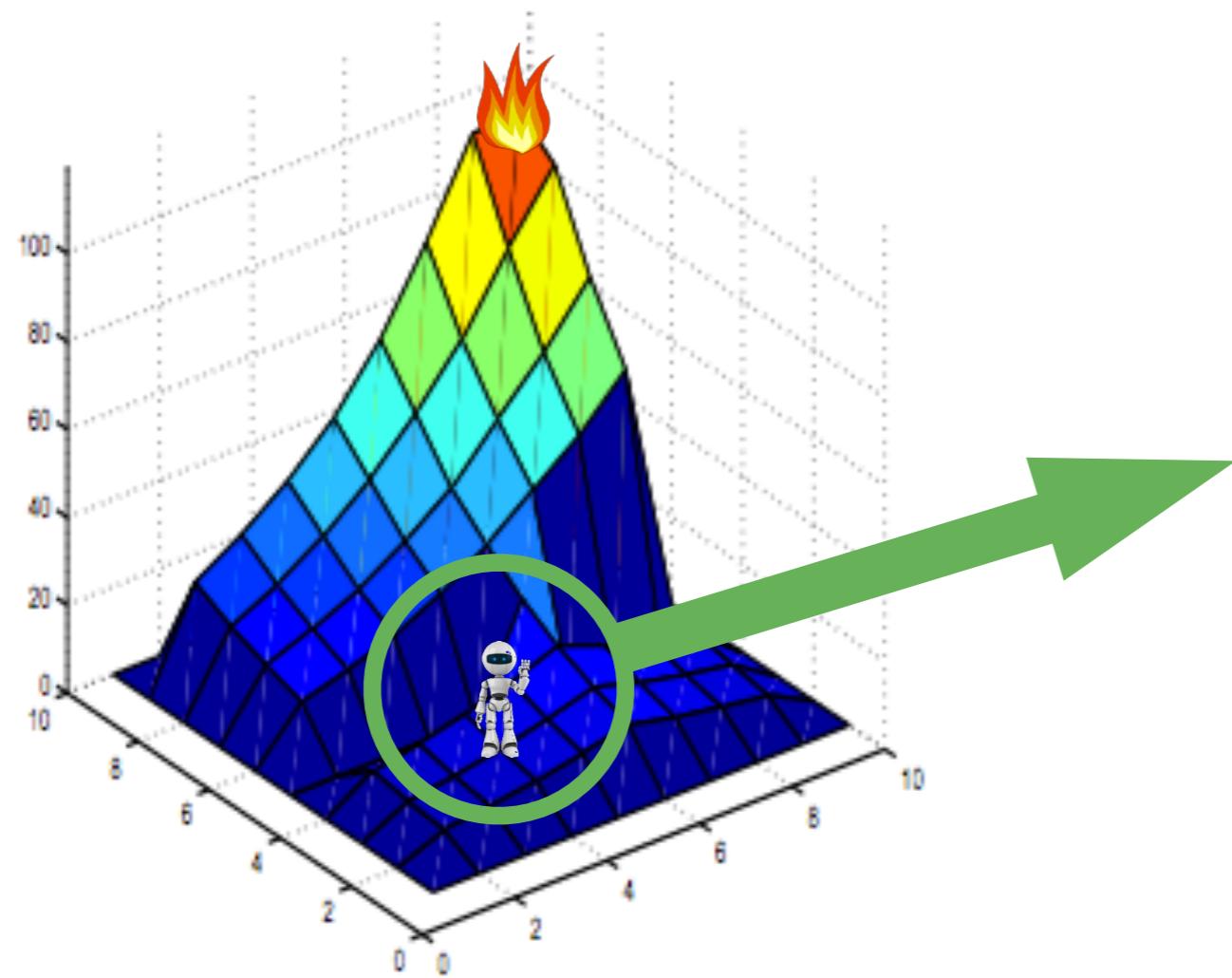


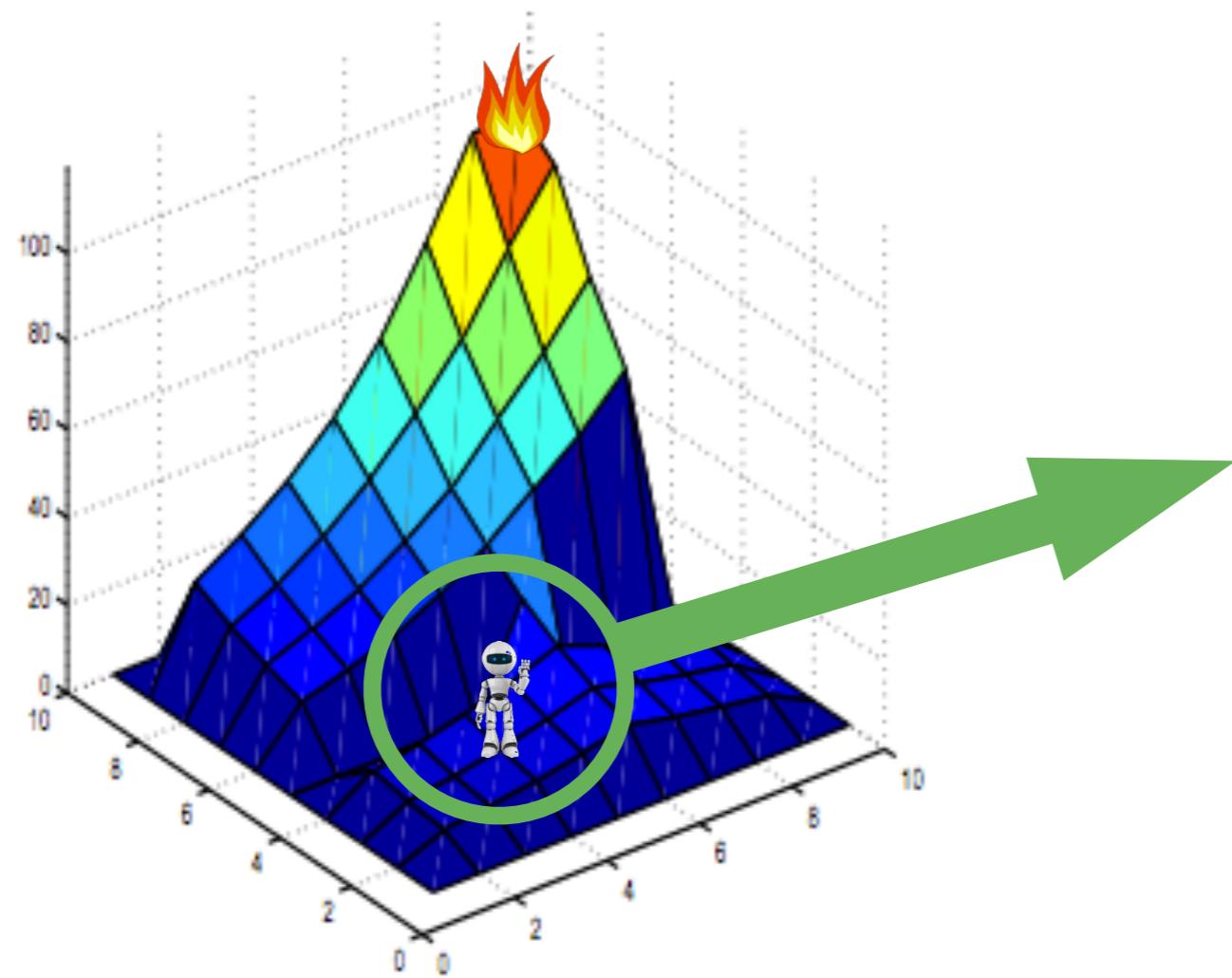


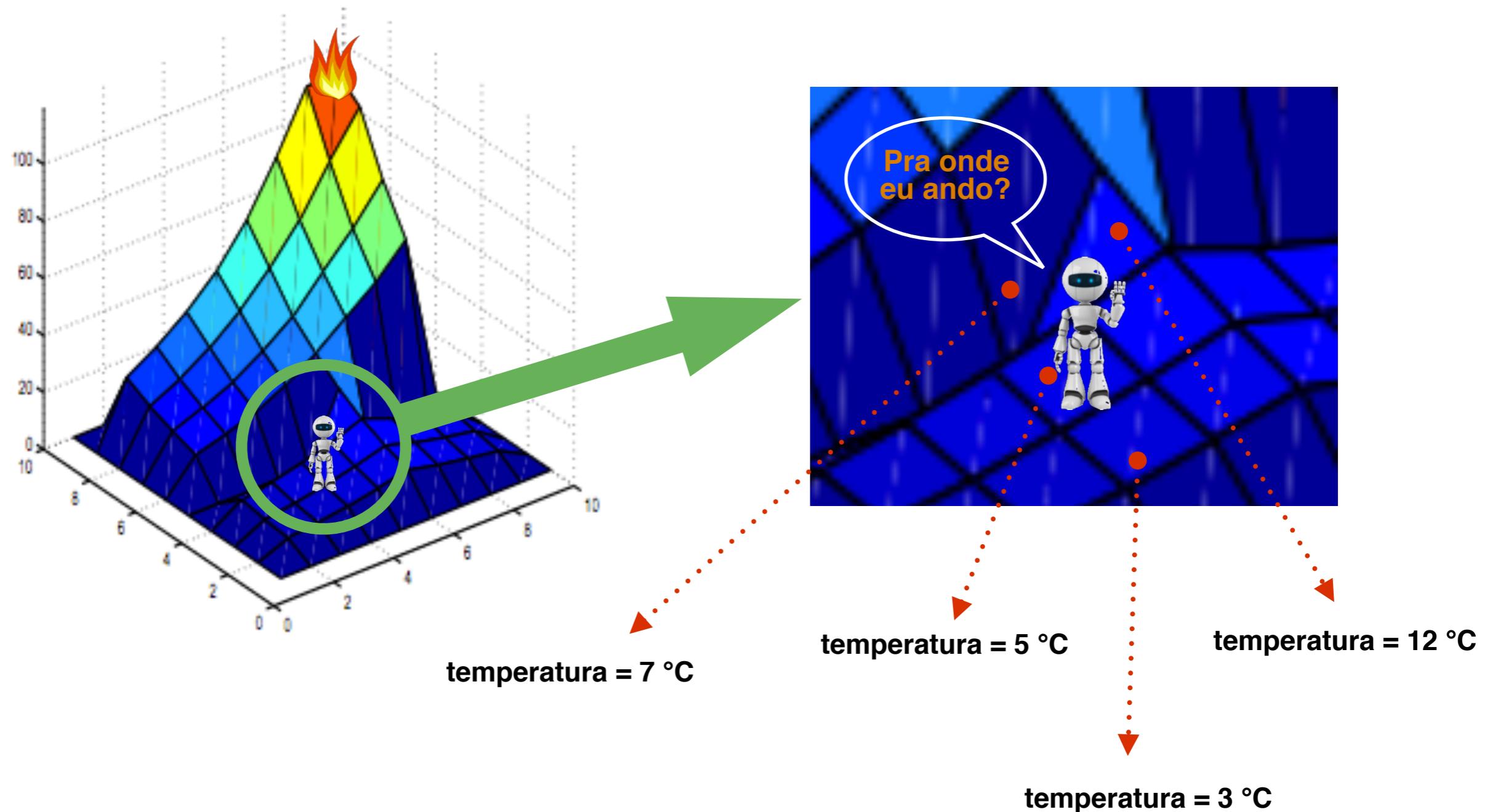


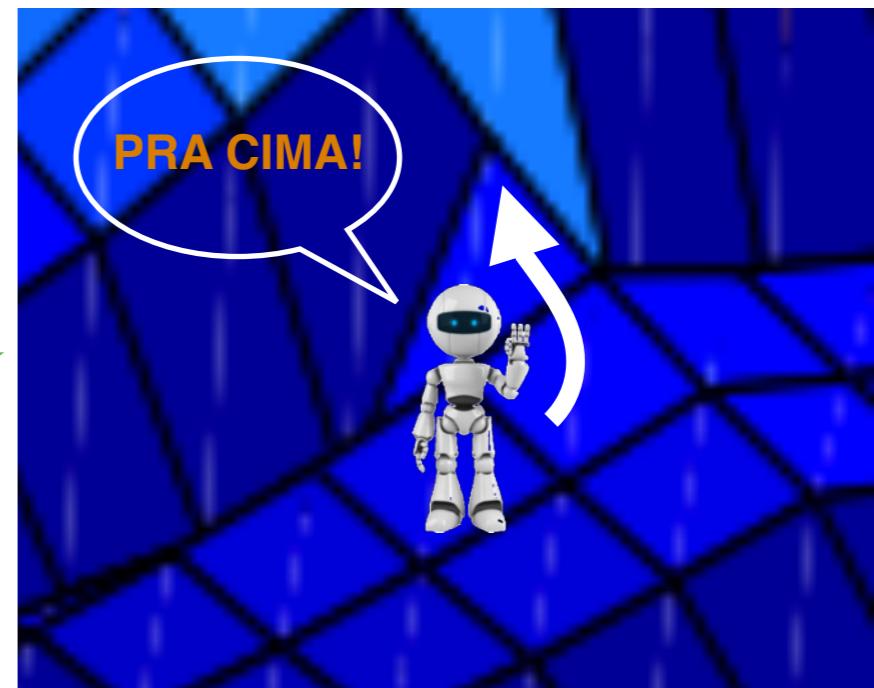
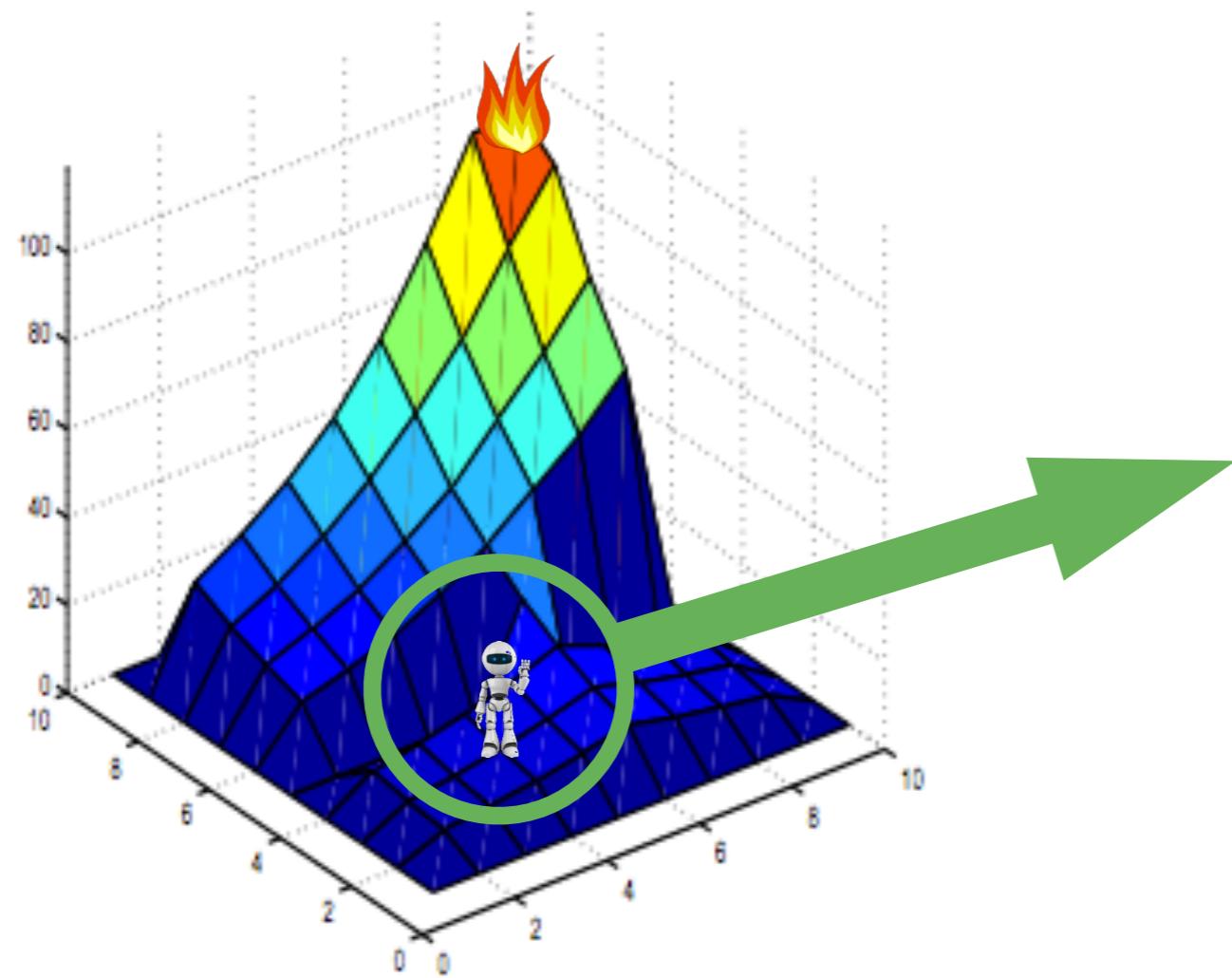


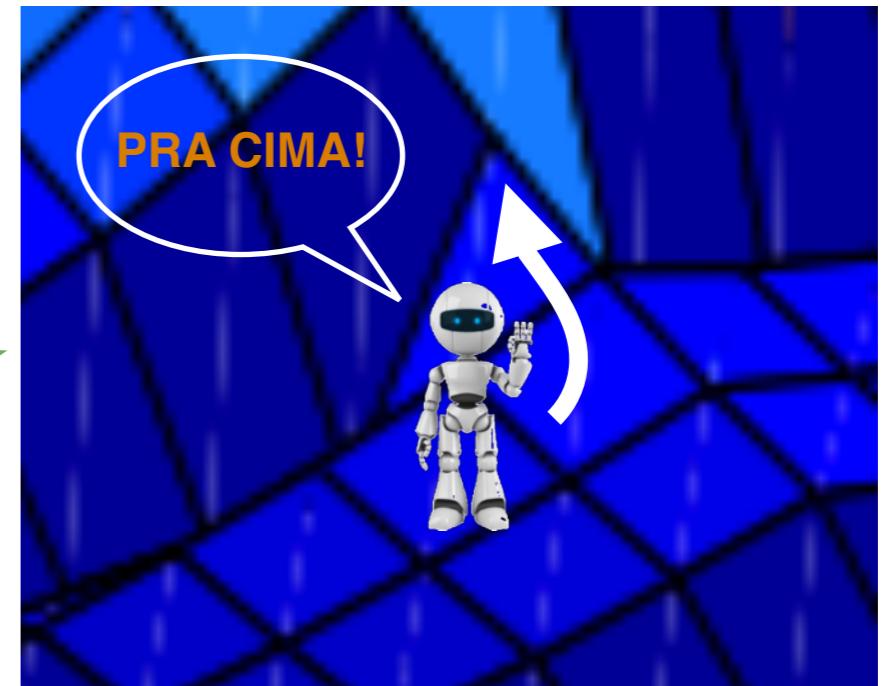
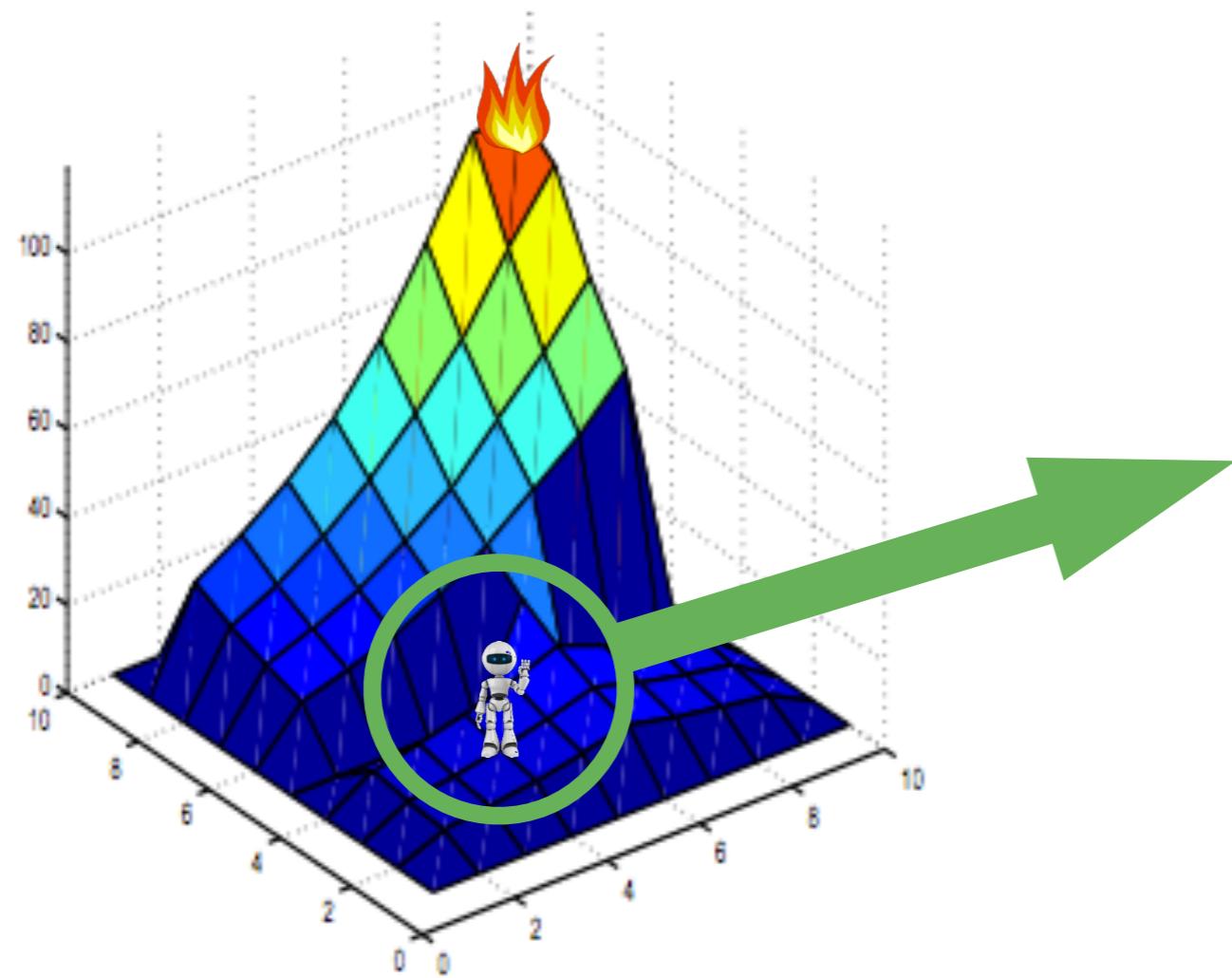






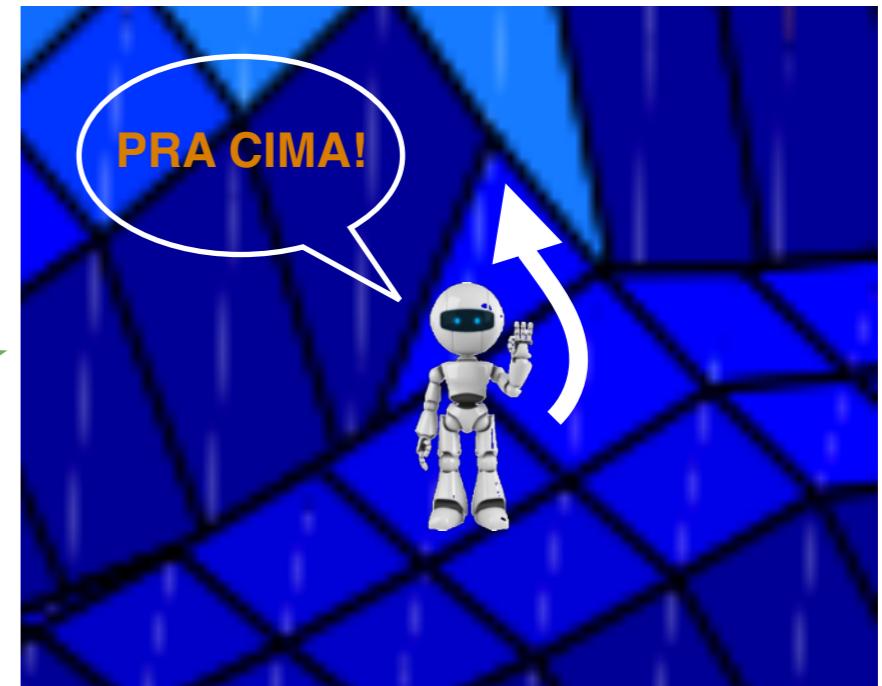
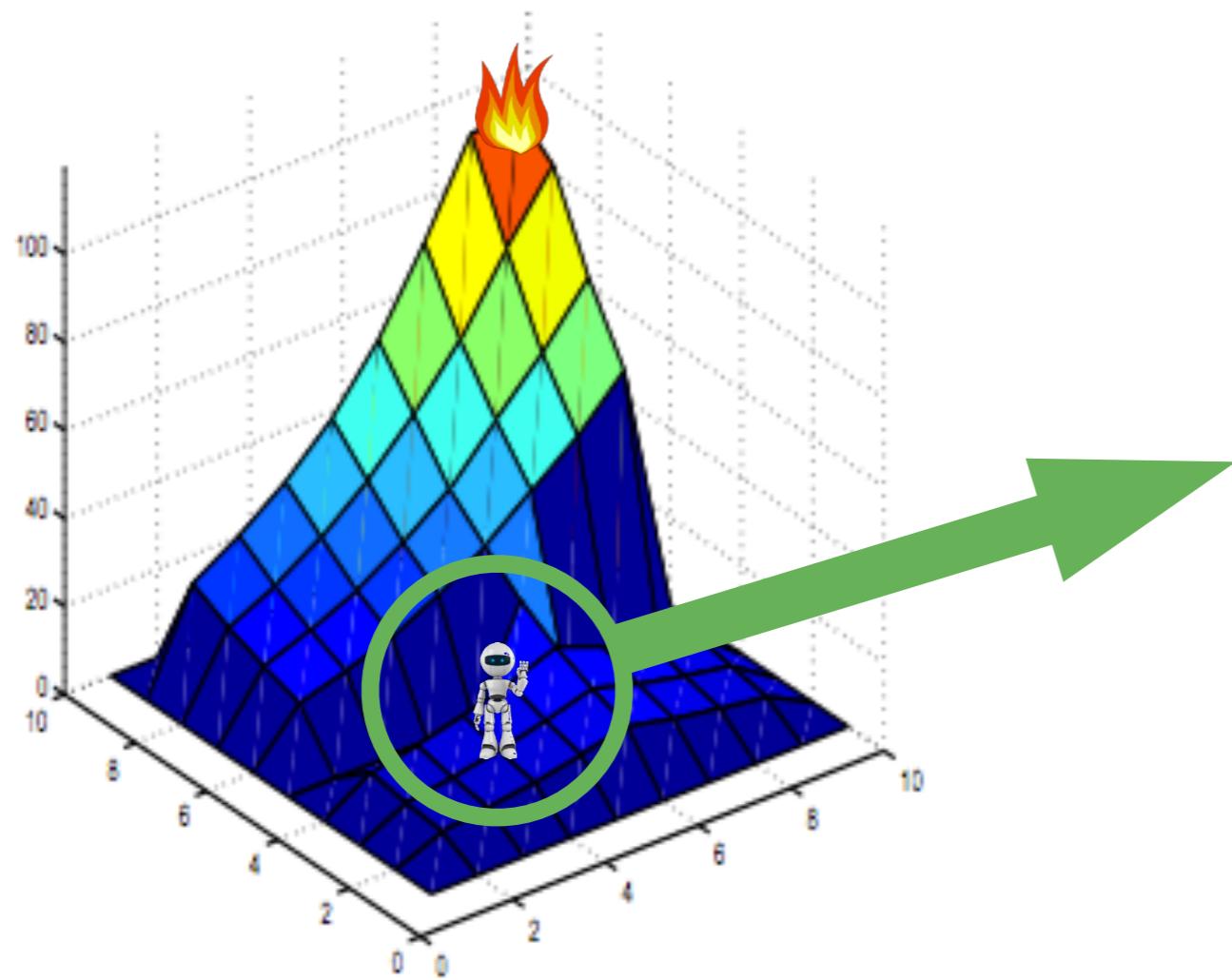






Cada estado s tem um valor associado a ele → $\text{VALOR}(s)$

Objetivo é alcançar/identificar o estado s^* : $\underset{s}{\text{argmax}} \text{VALOR}(s)$

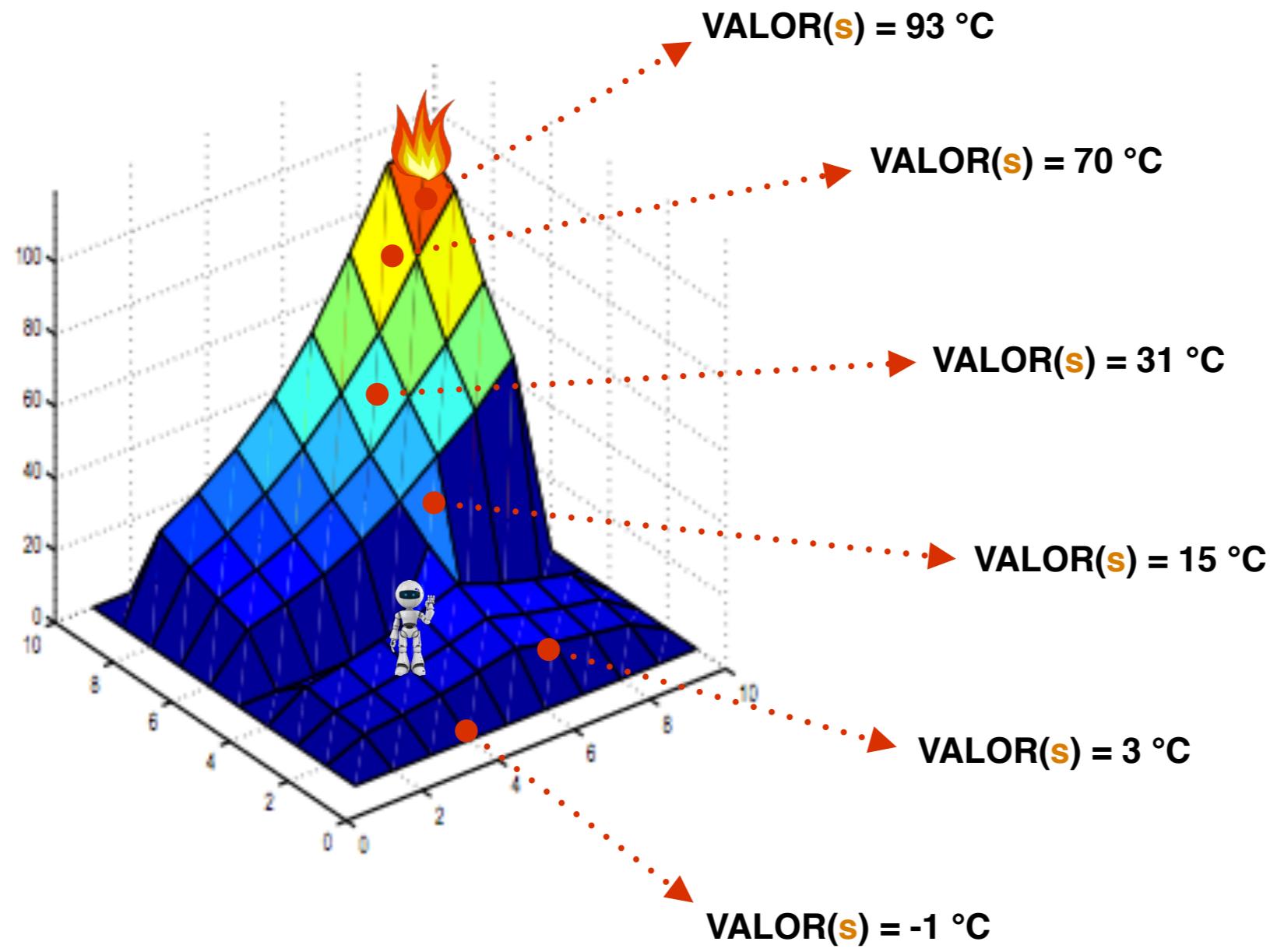


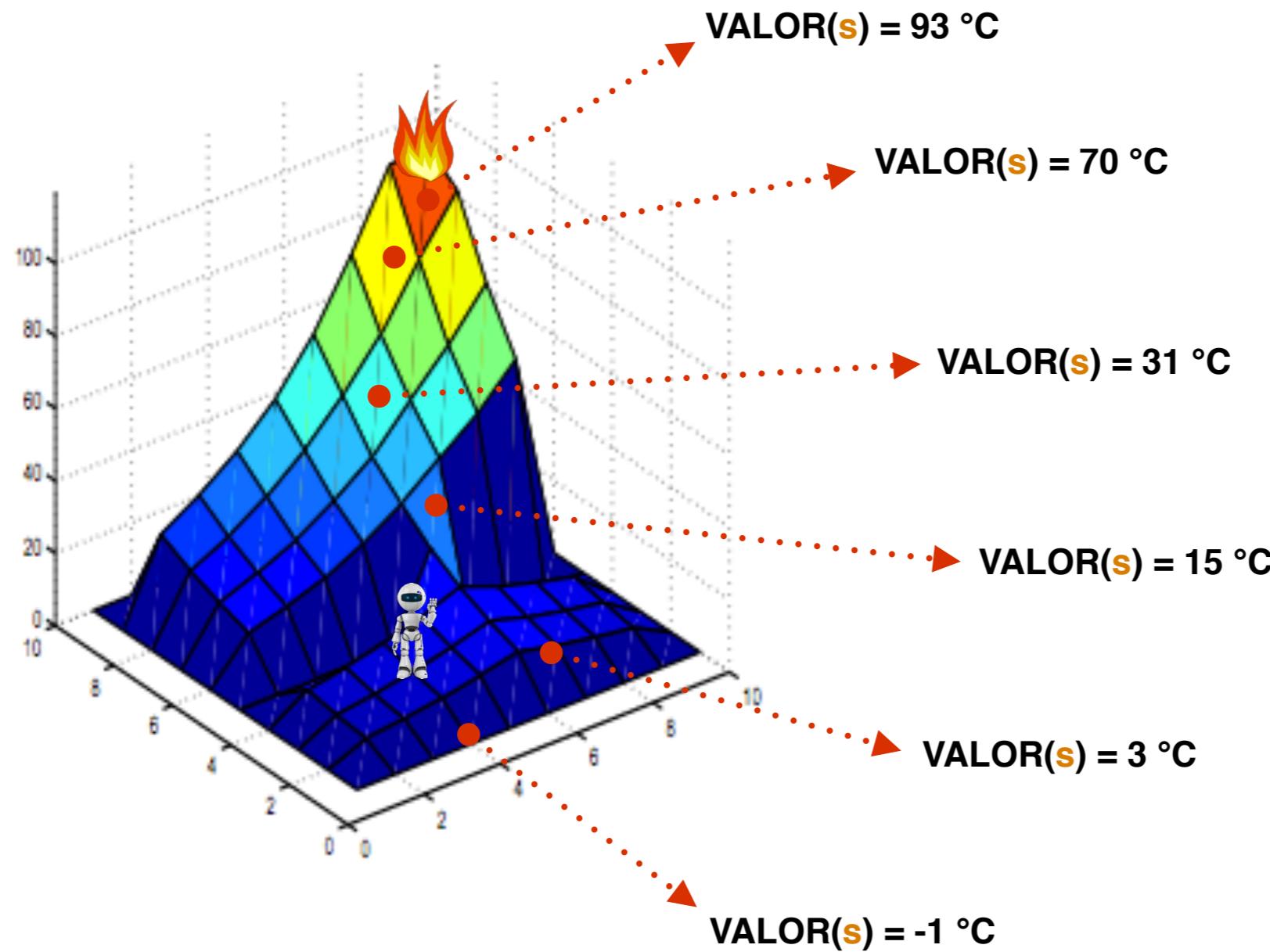
Cada estado s tem um valor associado a ele → $\text{VALOR}(s)$

Objetivo é alcançar/identificar o estado s^* : $\underset{s}{\text{argmax}} \text{VALOR}(s)$

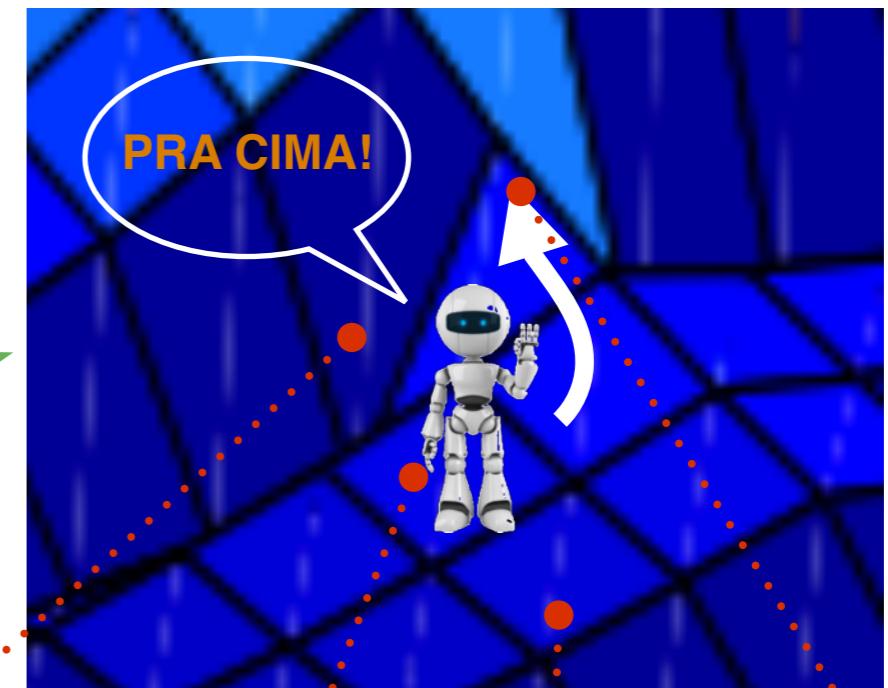
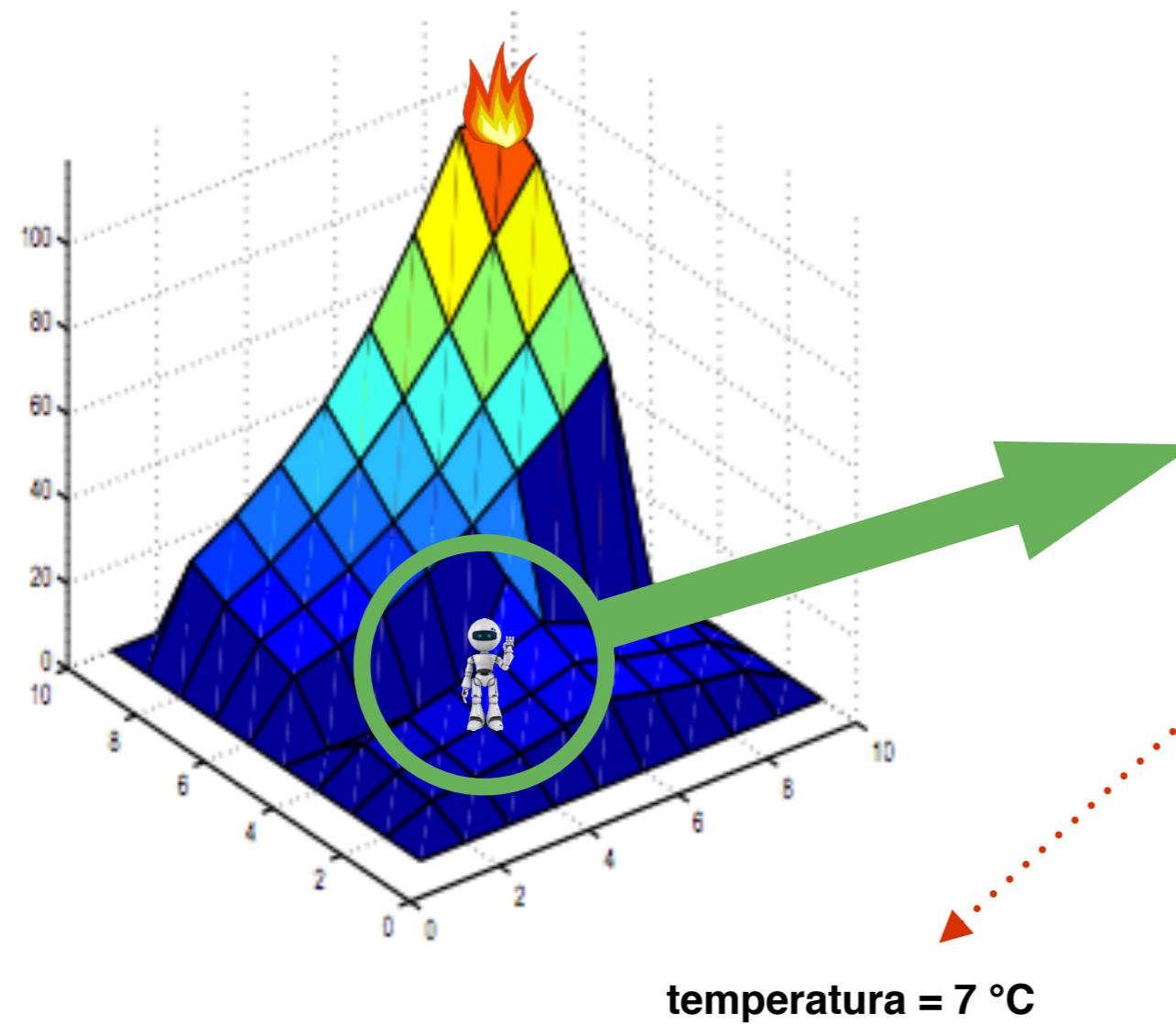


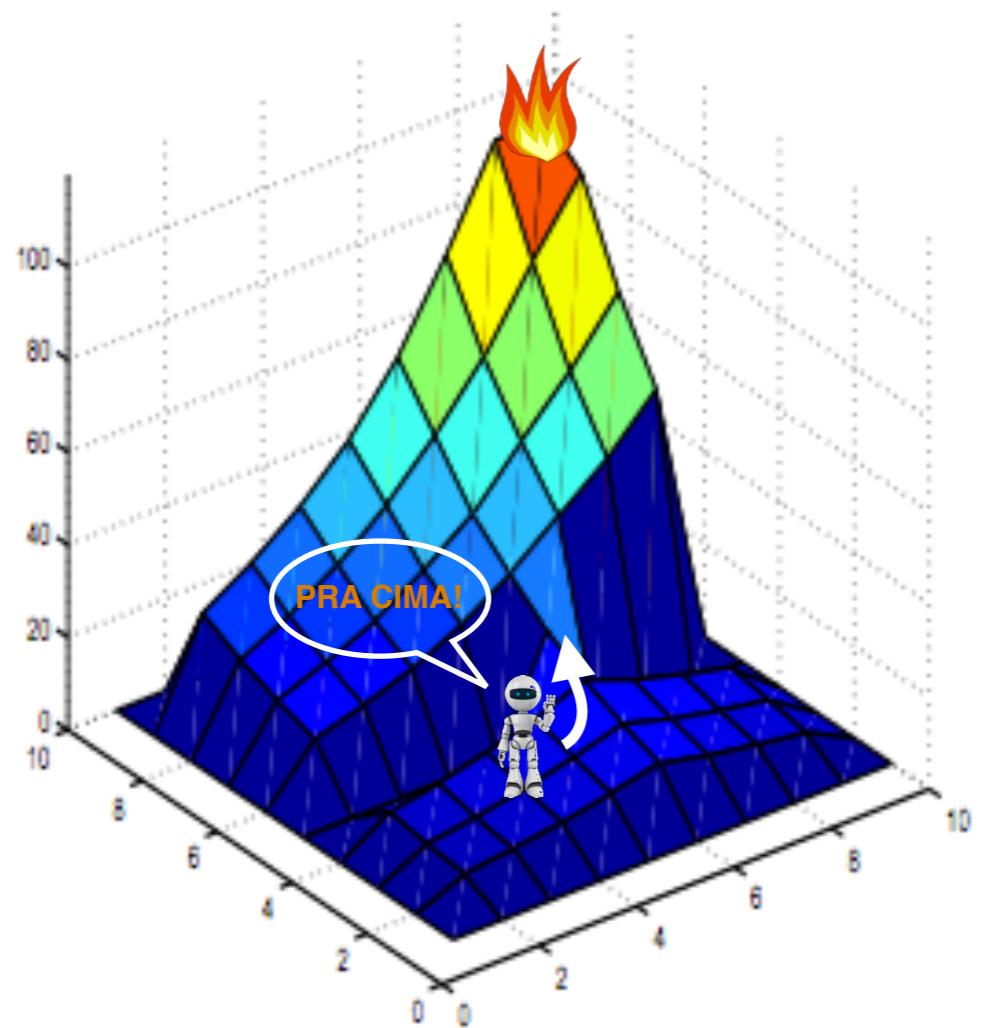
PROBLEMA DE OTIMIZAÇÃO

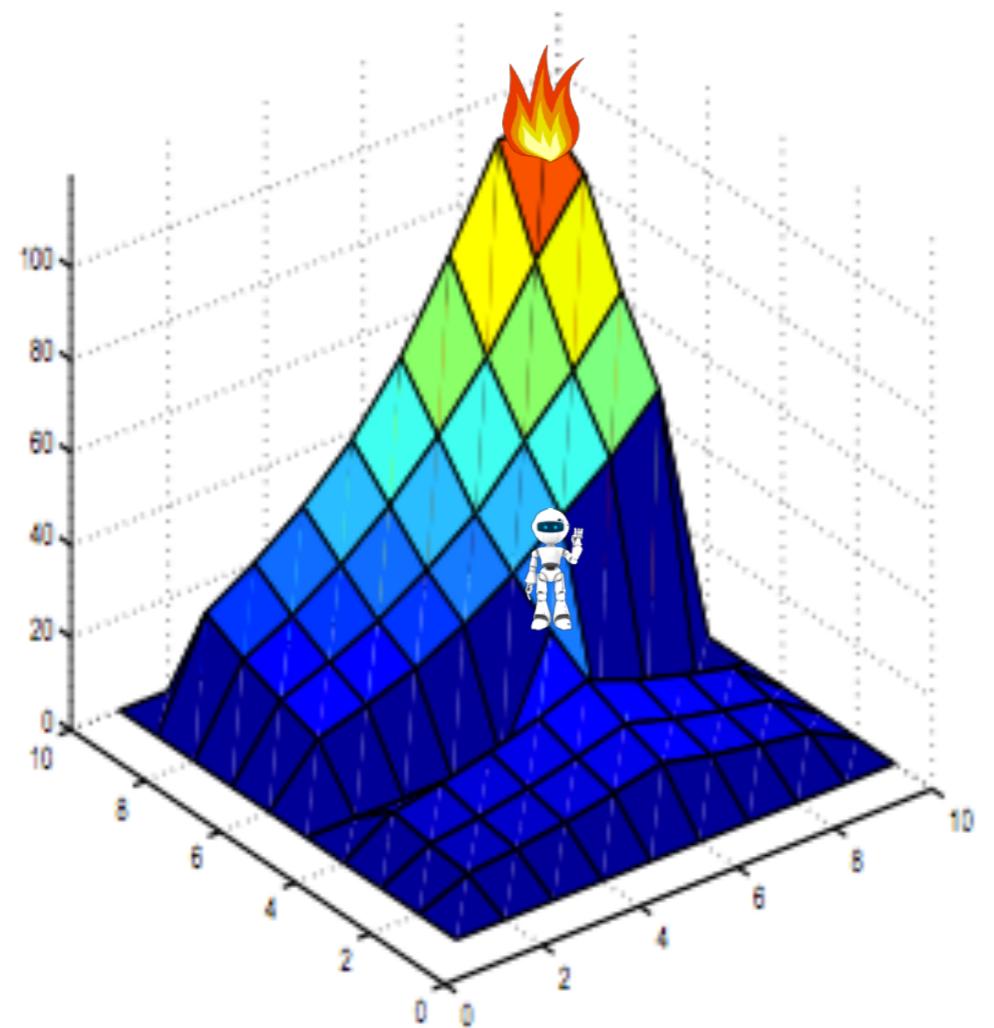


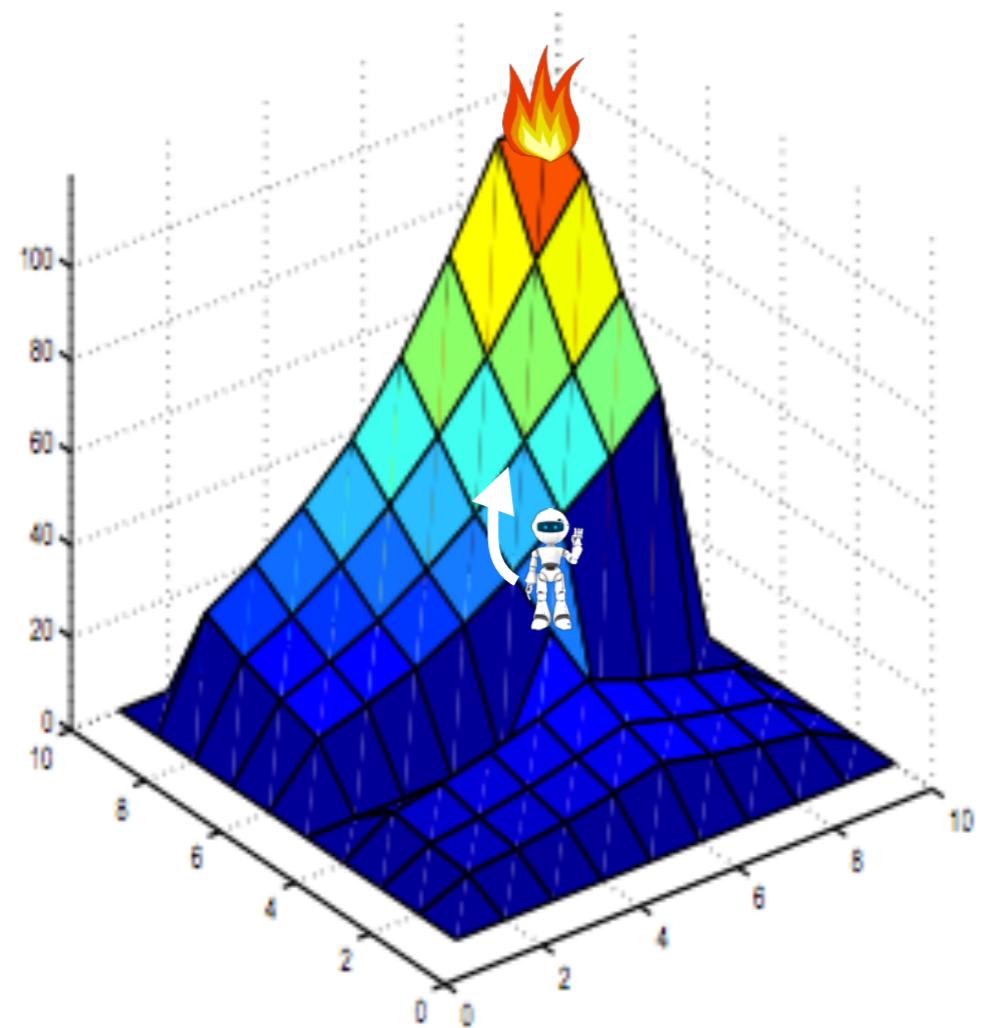


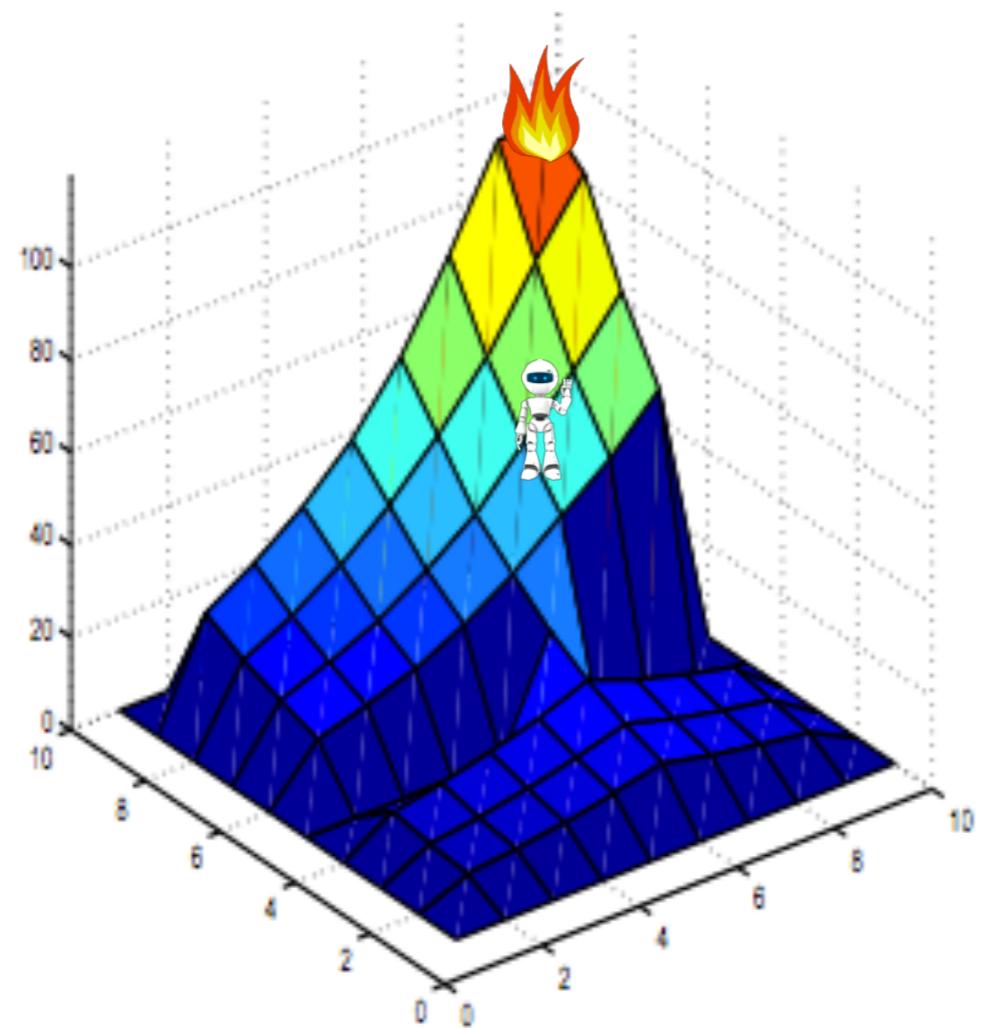
Objetivo é alcançar/identificar o estado s^* : $\underset{s}{\operatorname{argmax}} \text{VALOR}(s)$

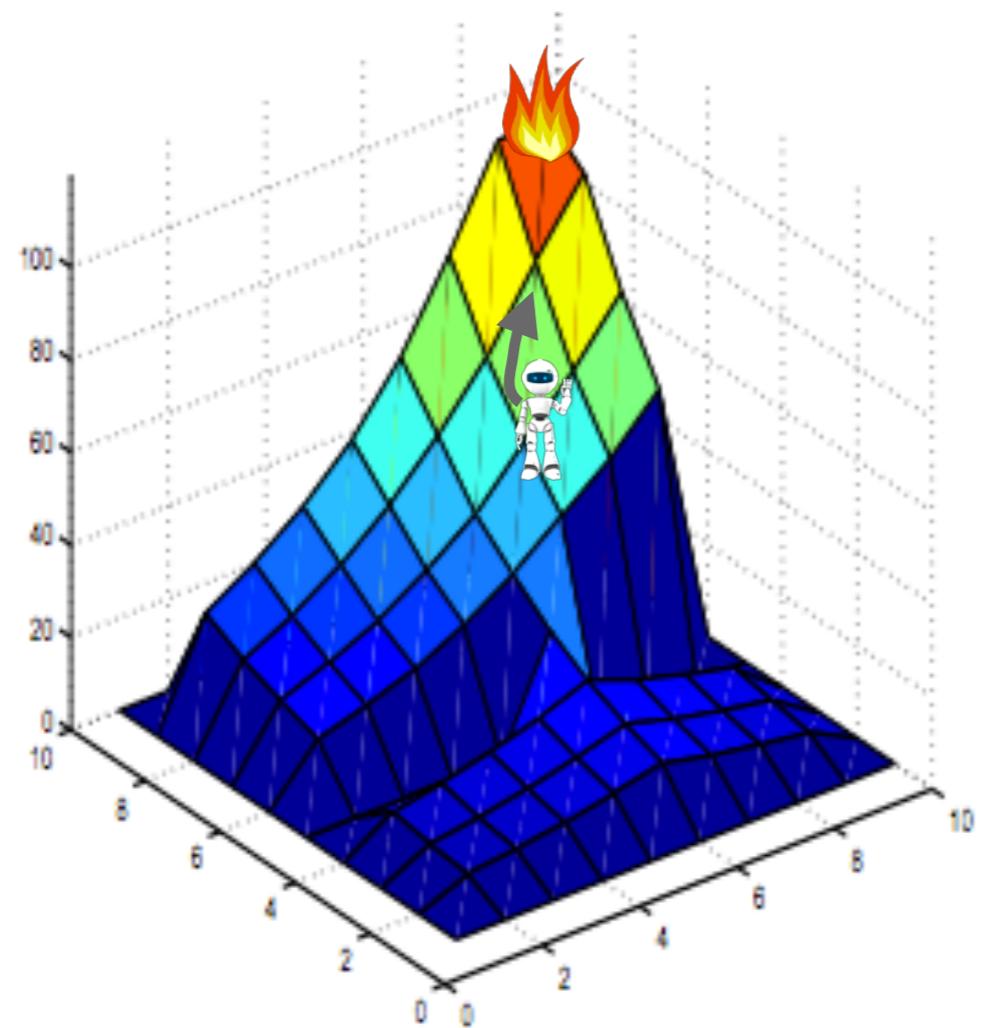


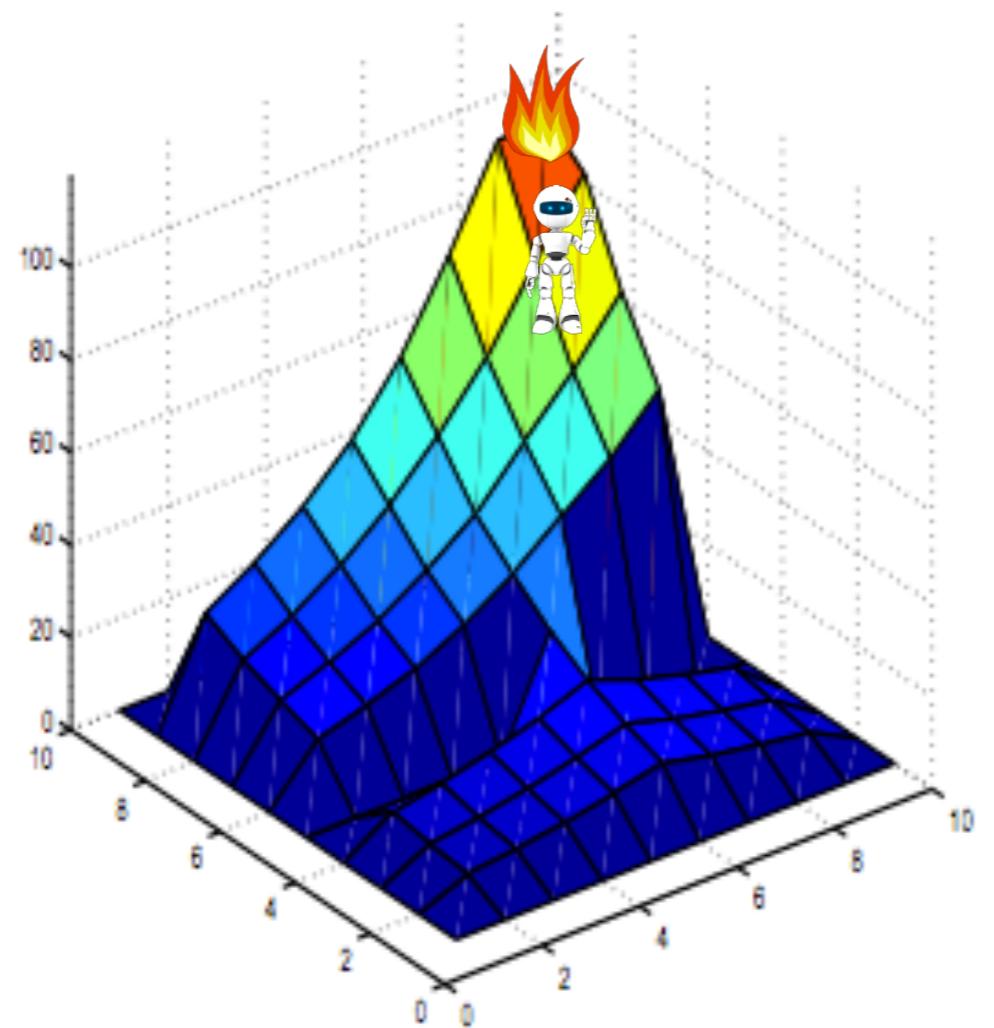


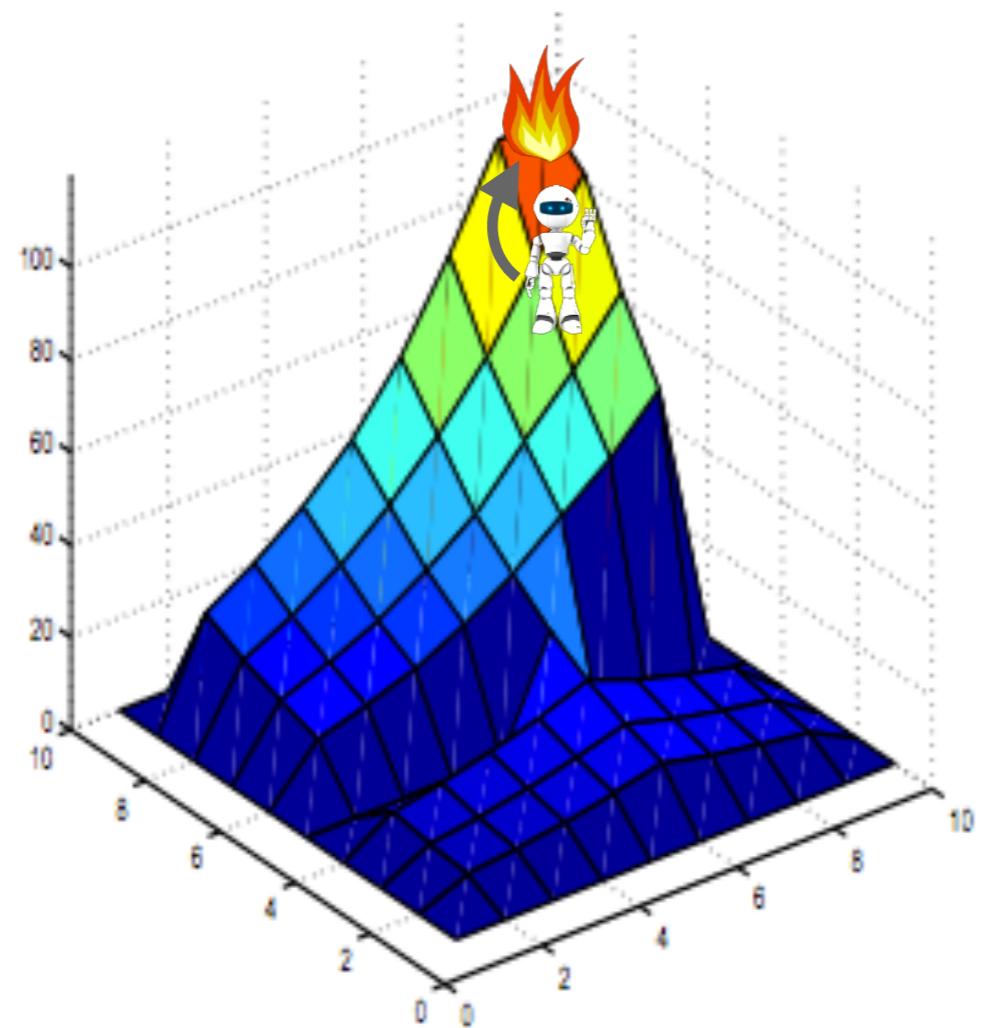


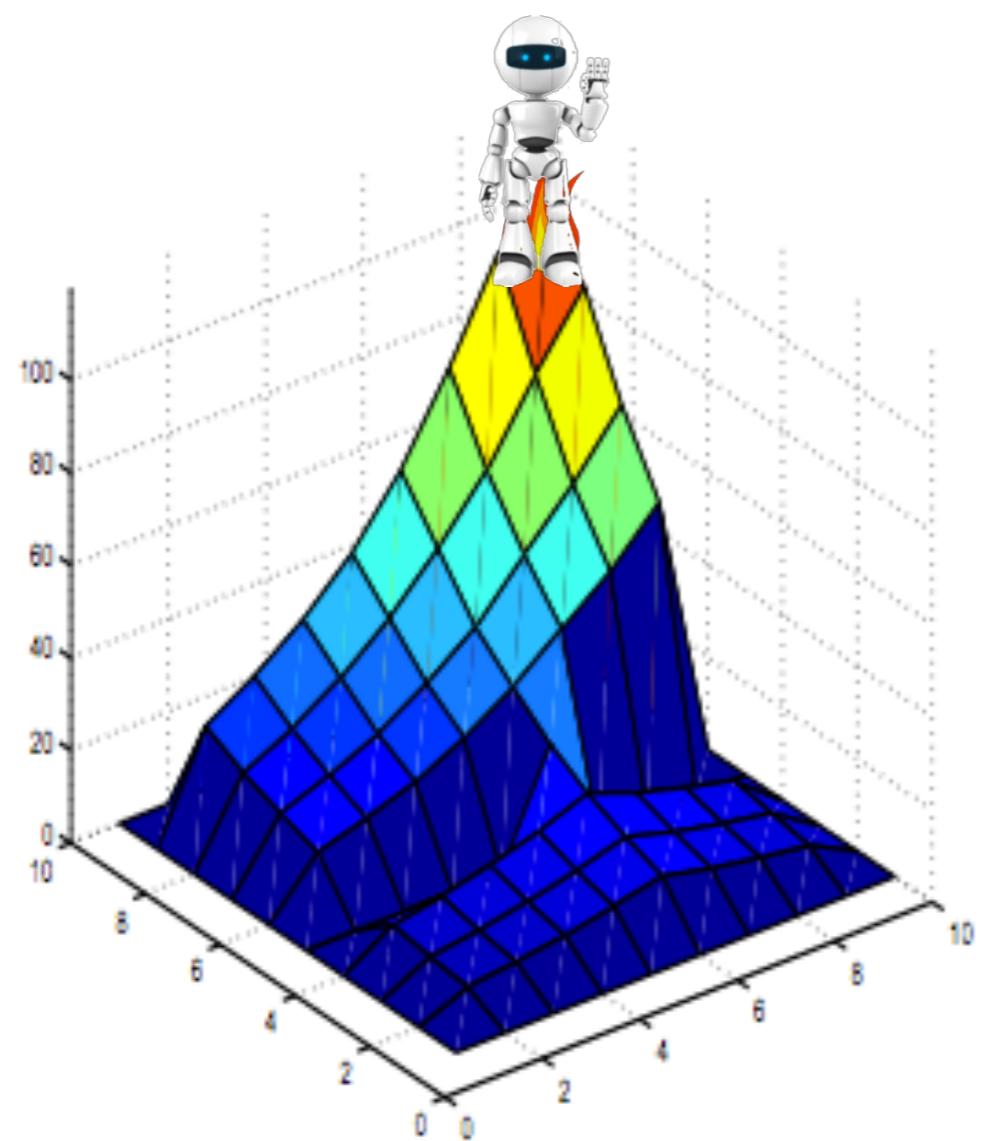


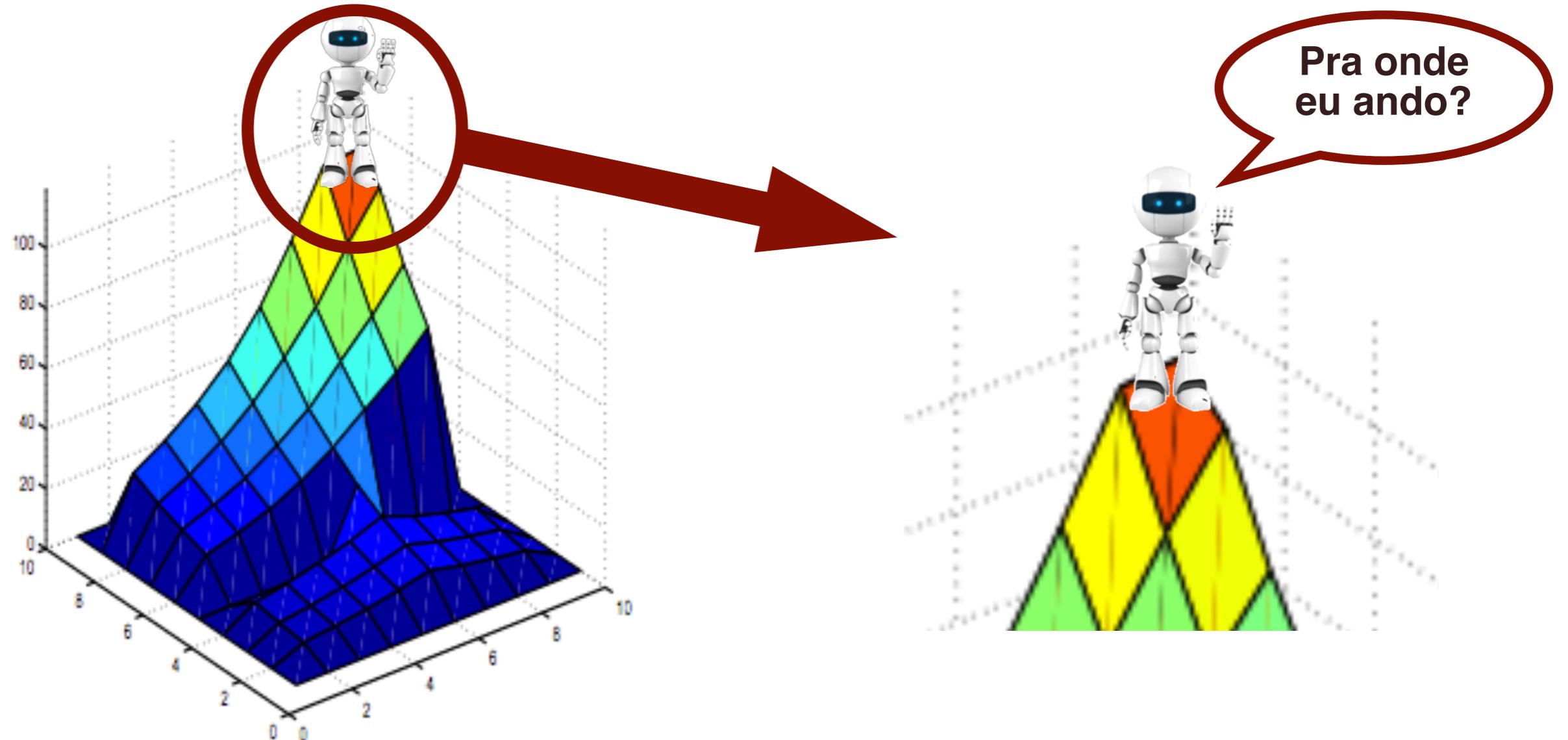


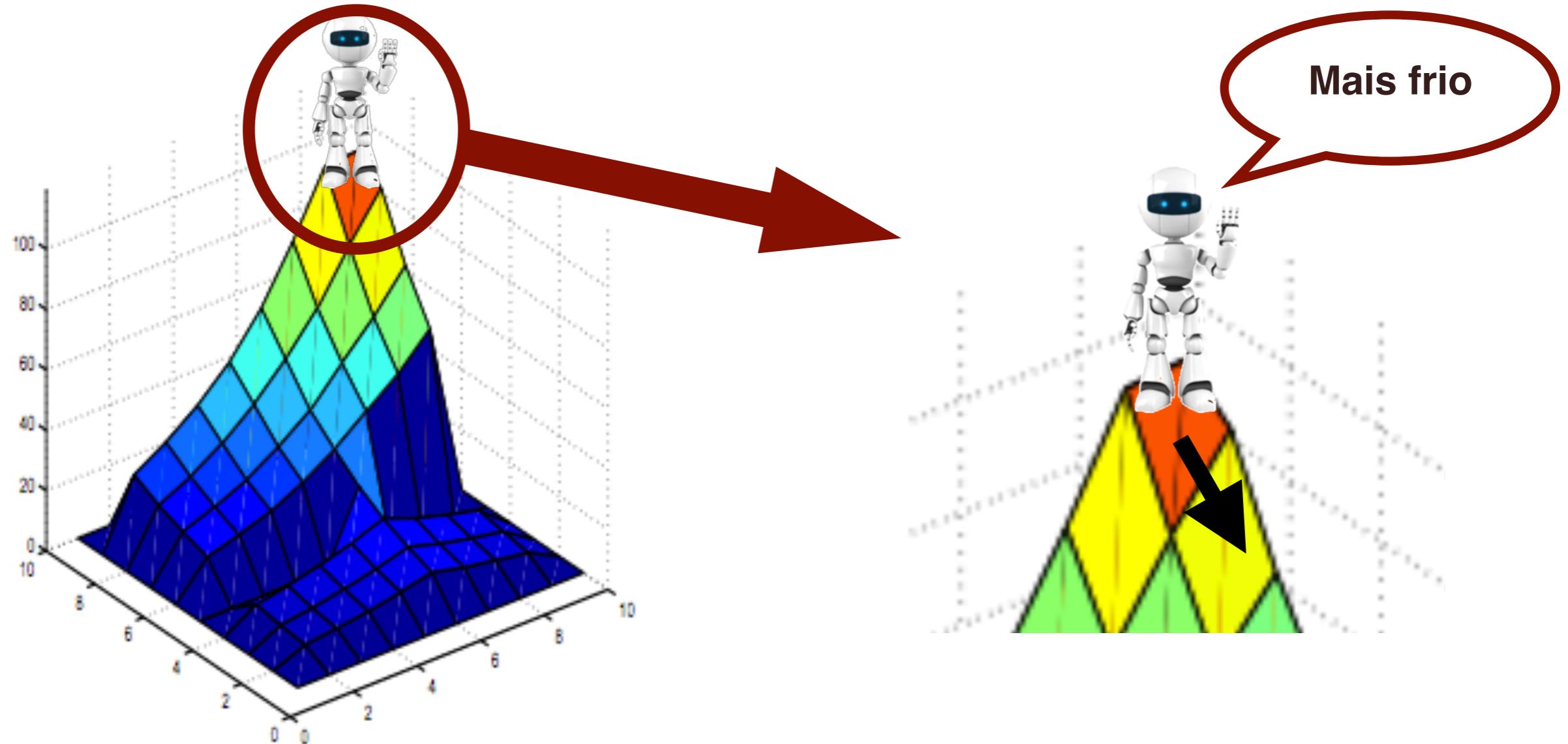


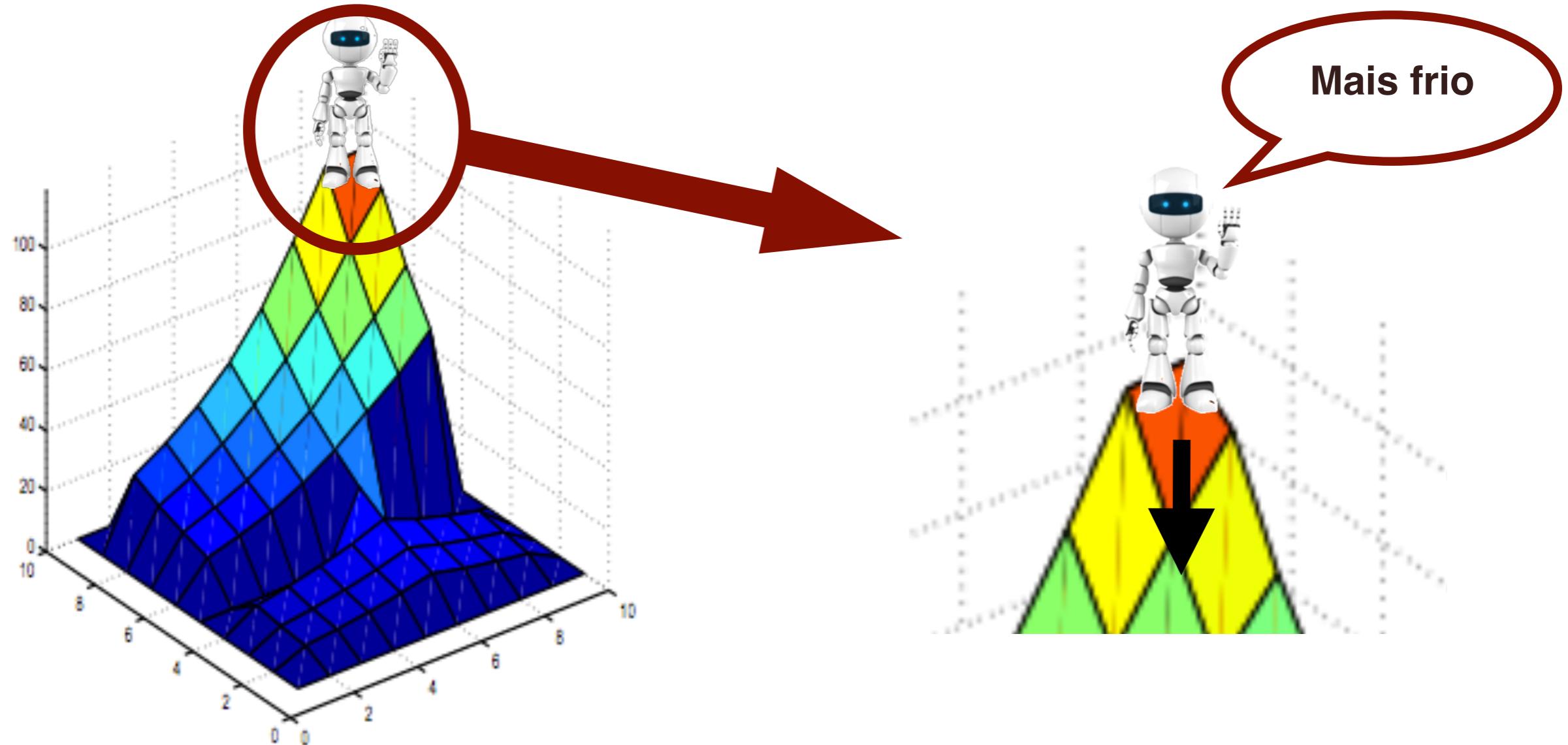


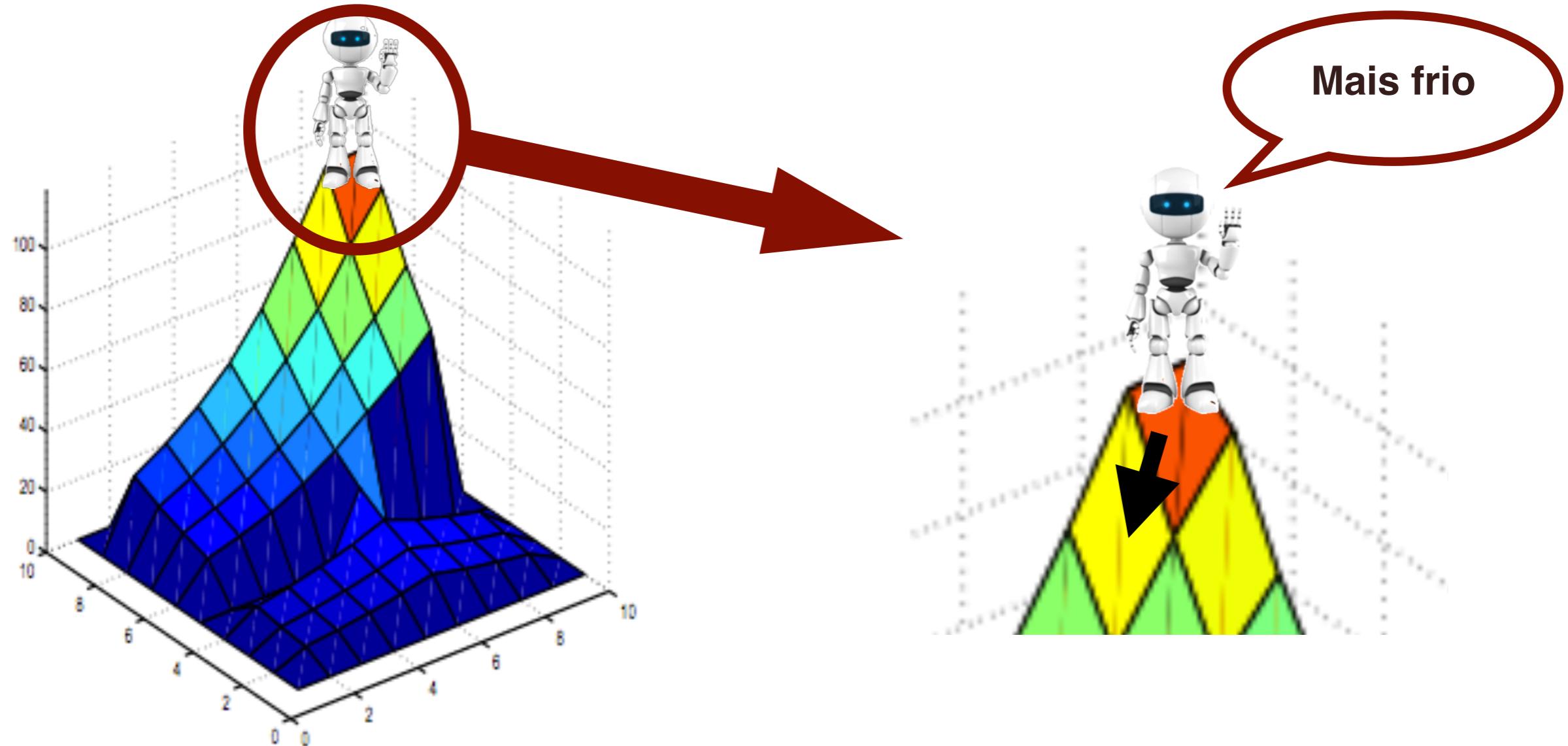


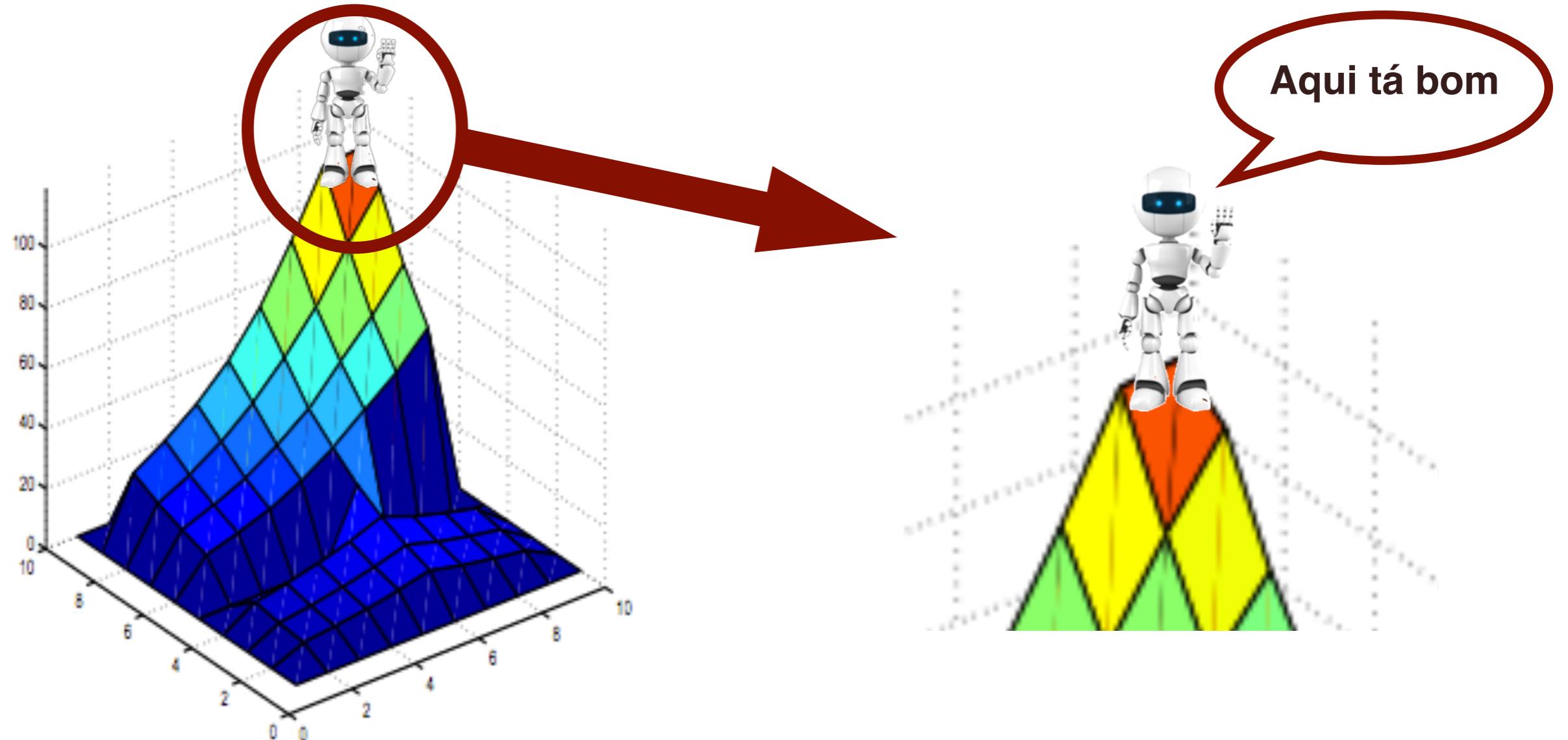


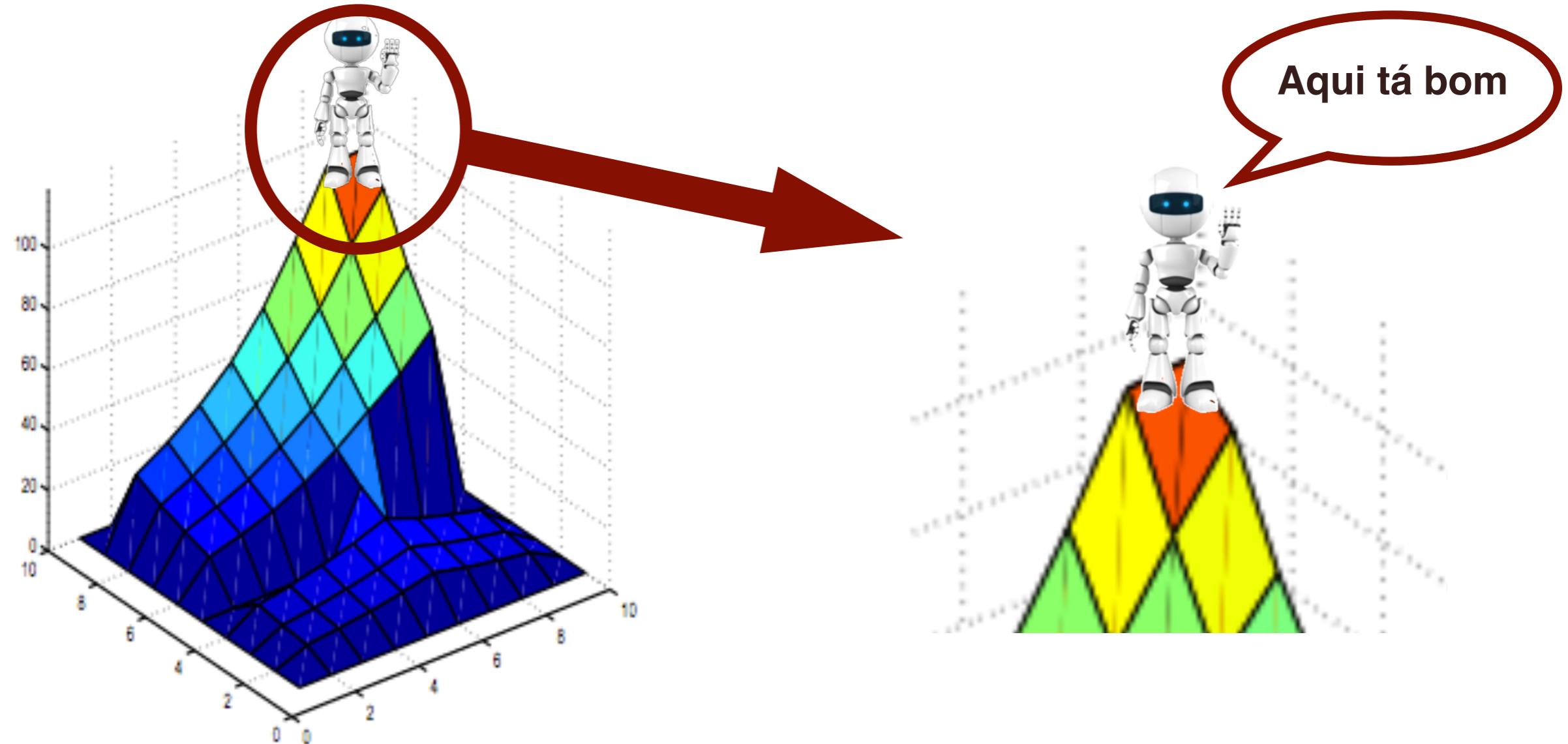






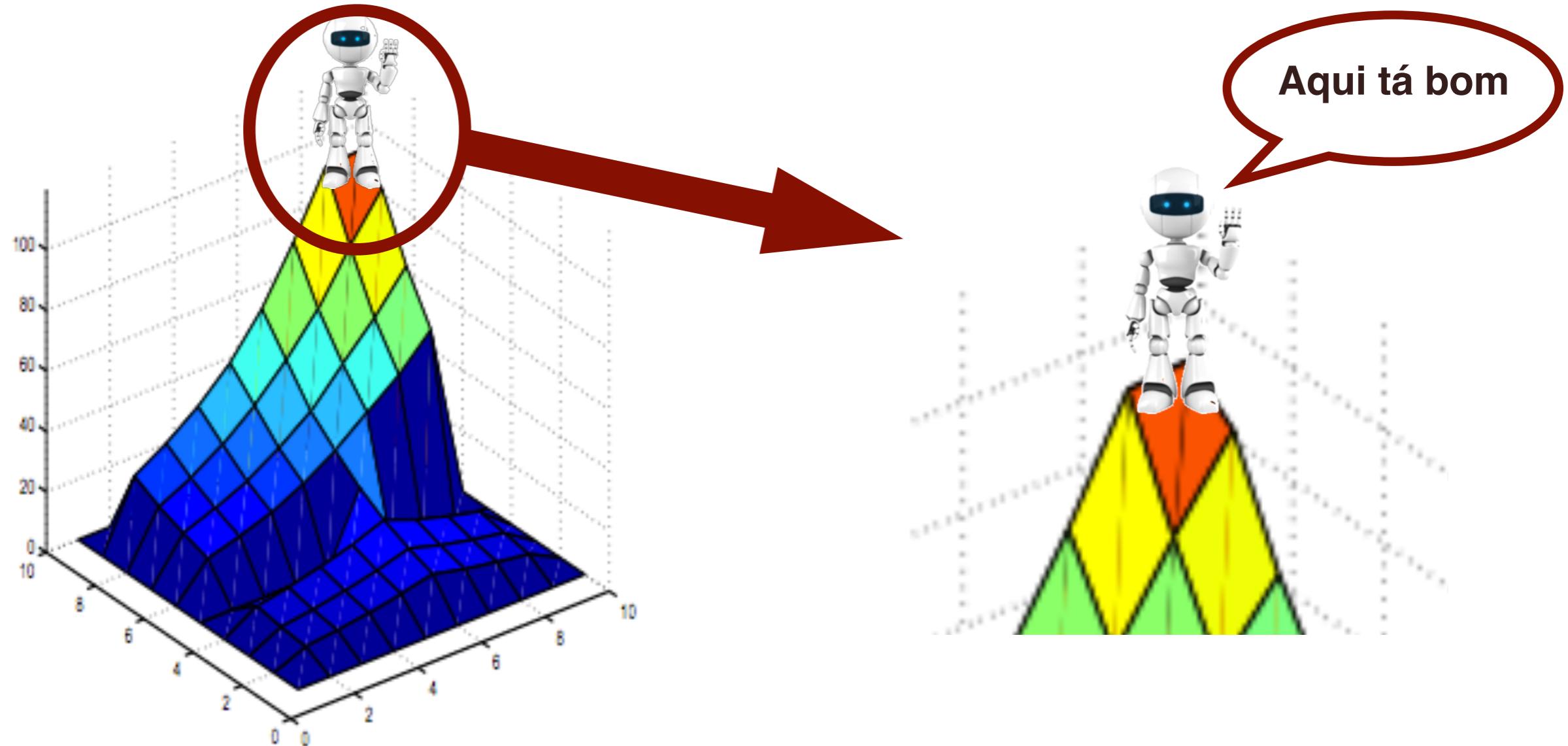






Ótimo global

Método encontrou estado s com maior VALOR(s)
analizando apenas informação local ao agente

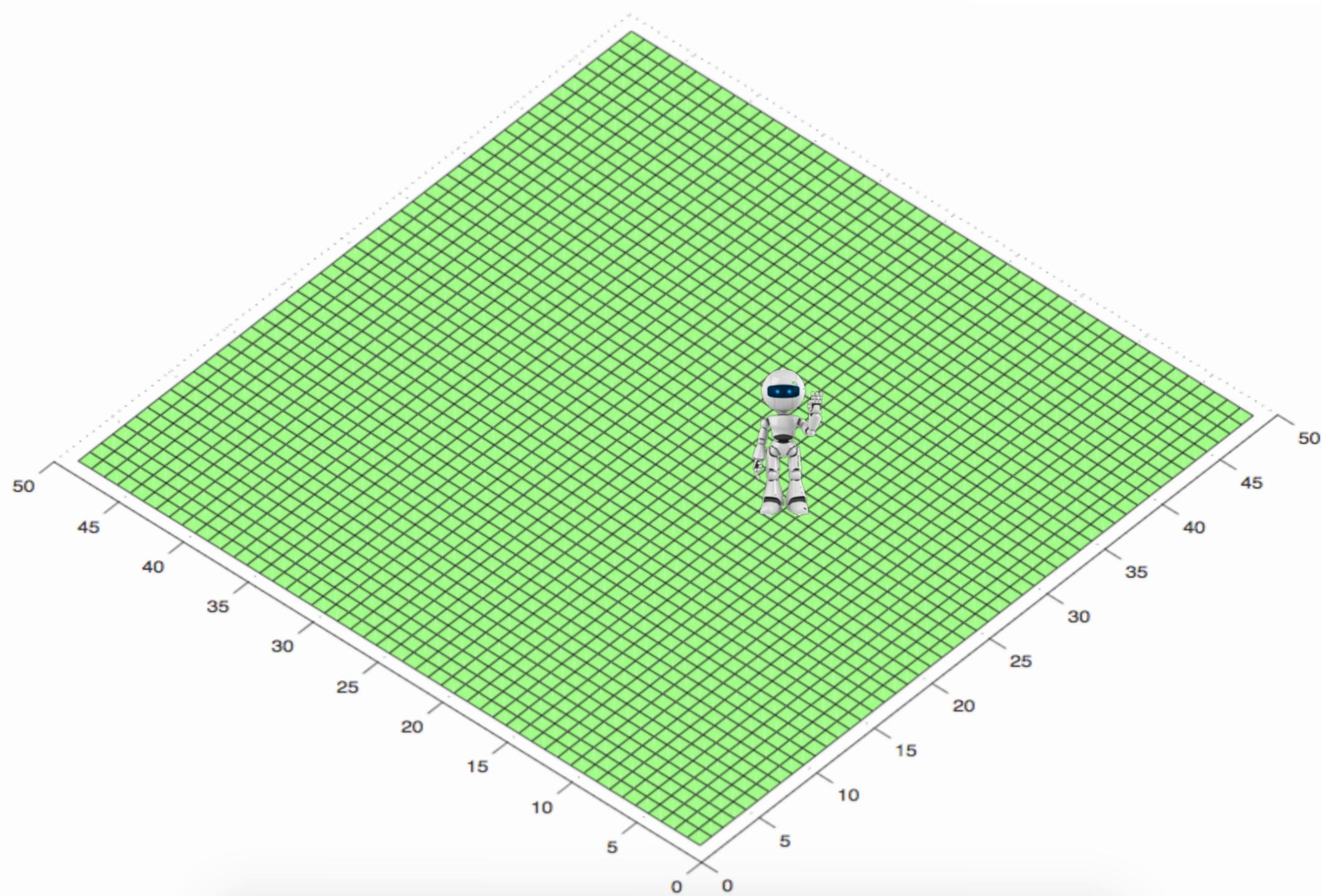


Ótimo global

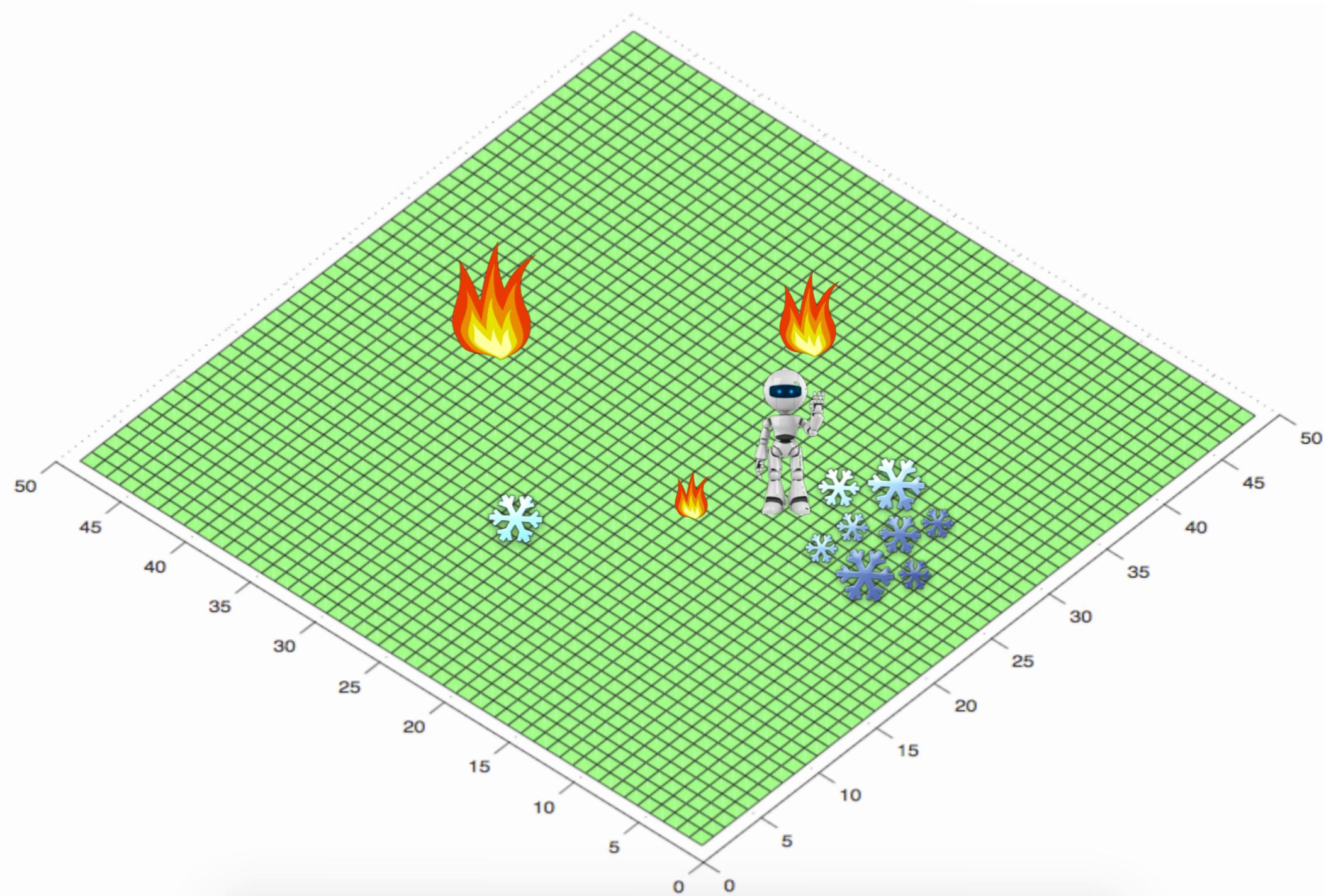


Sempre é alcançado por um método de busca local?

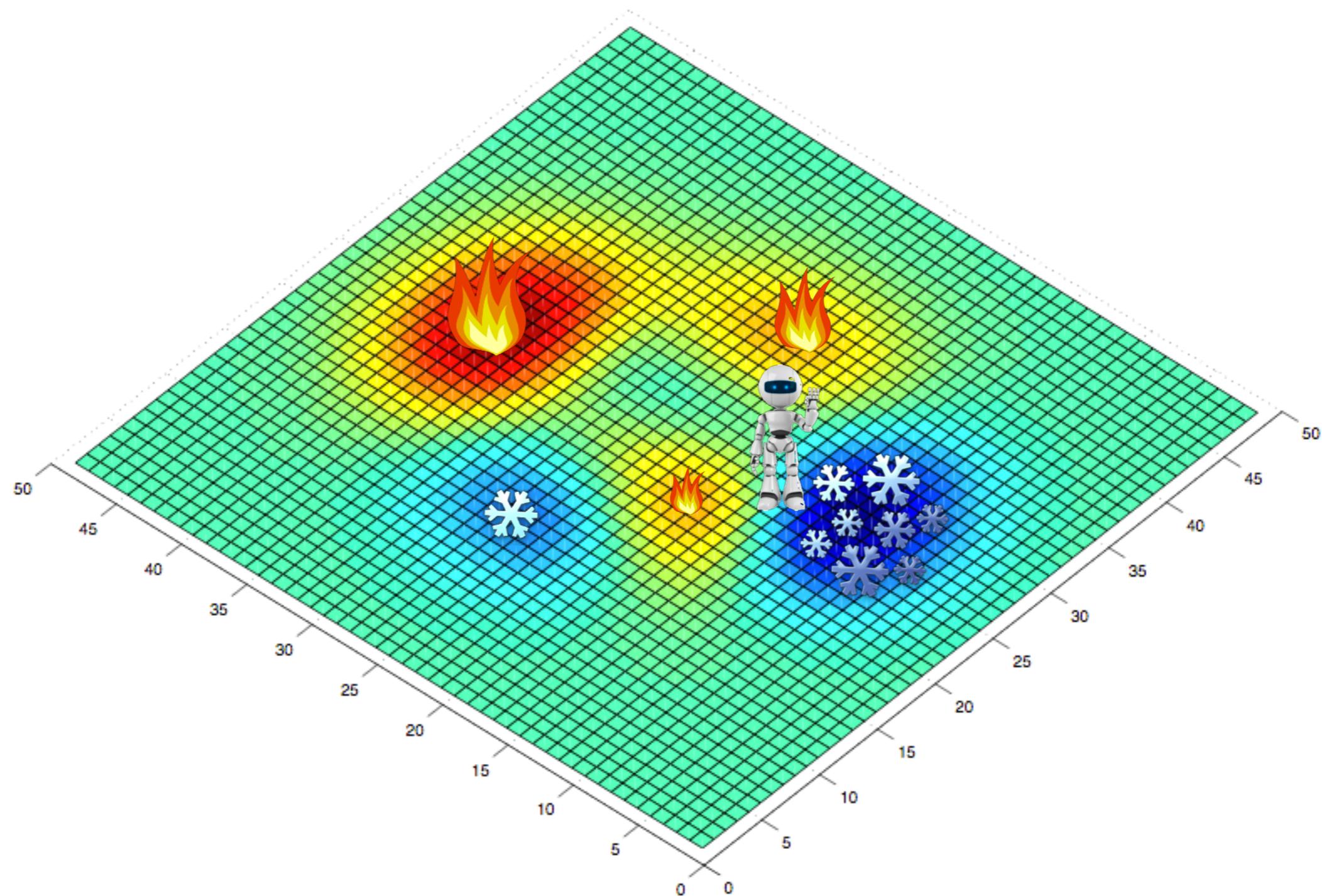
Complicando as Coisas...



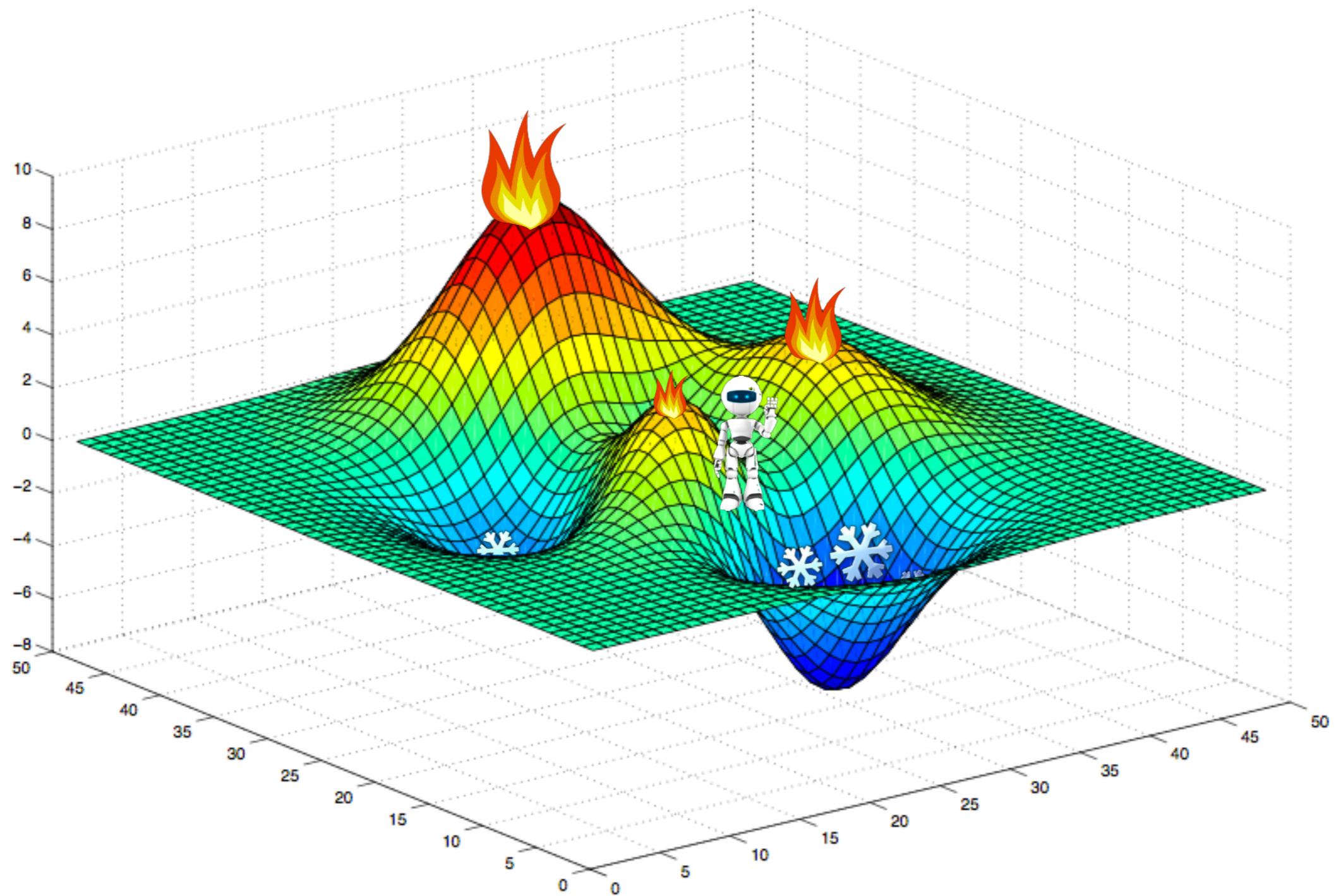
Complicando as Coisas...



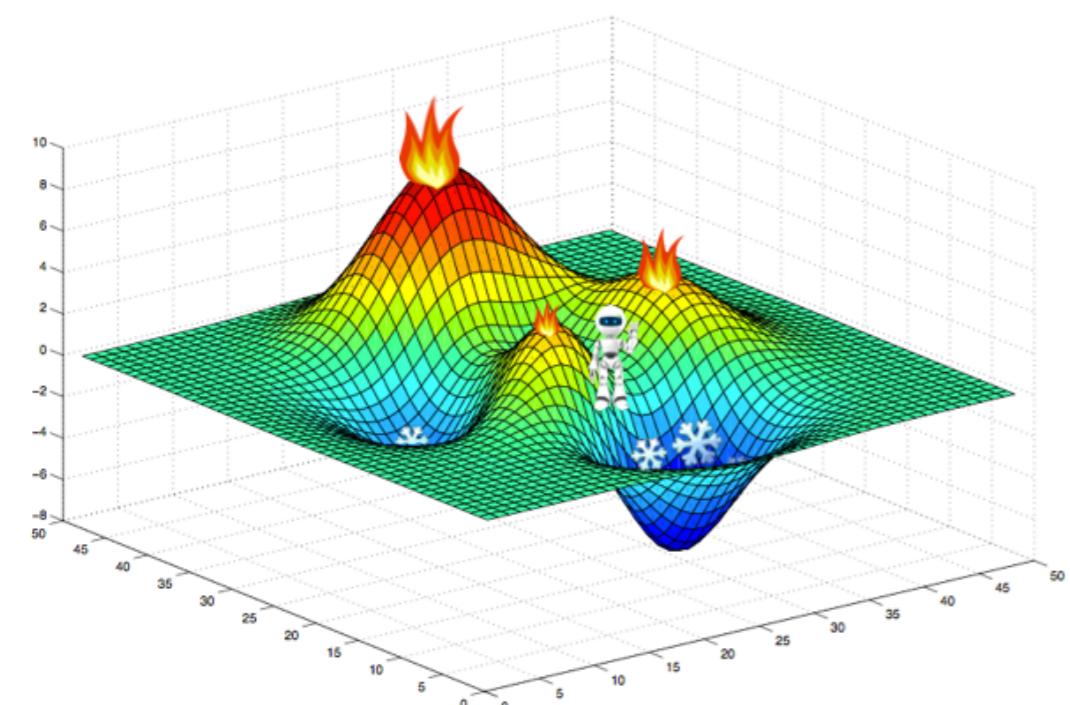
Complicando as Coisas...



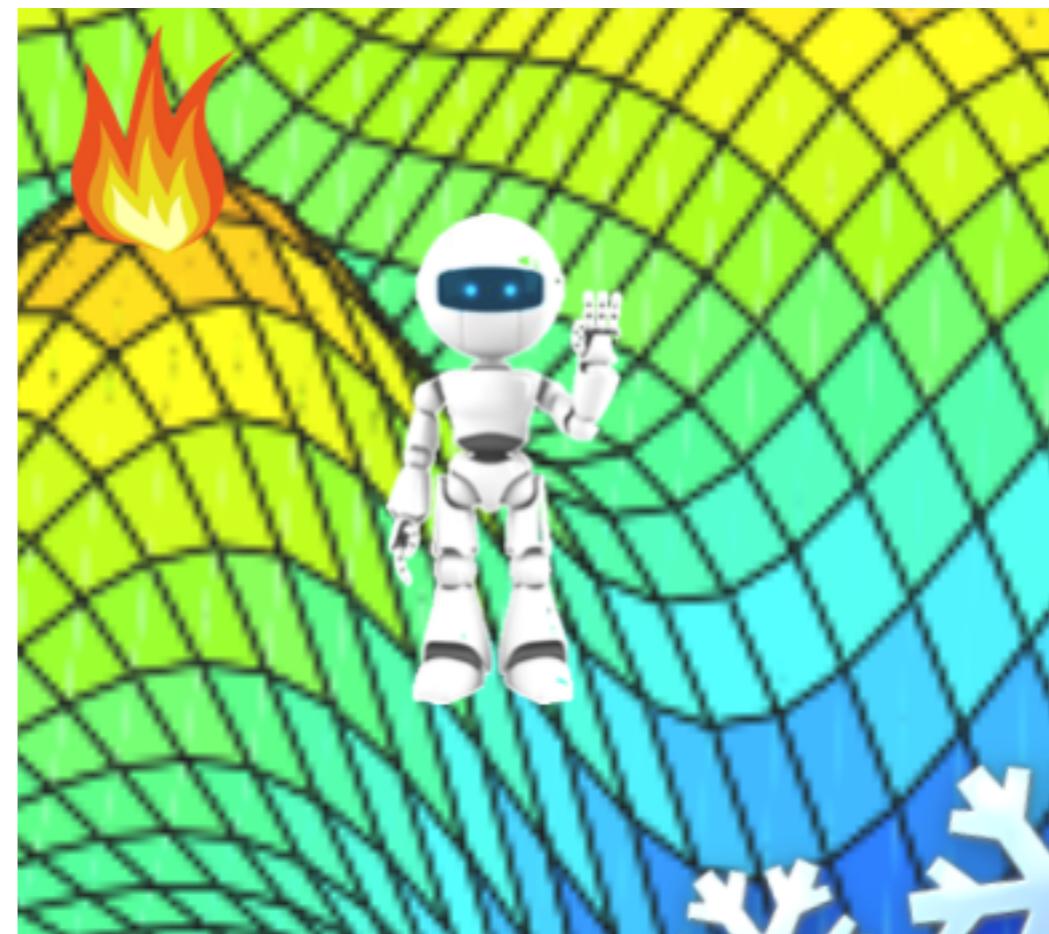
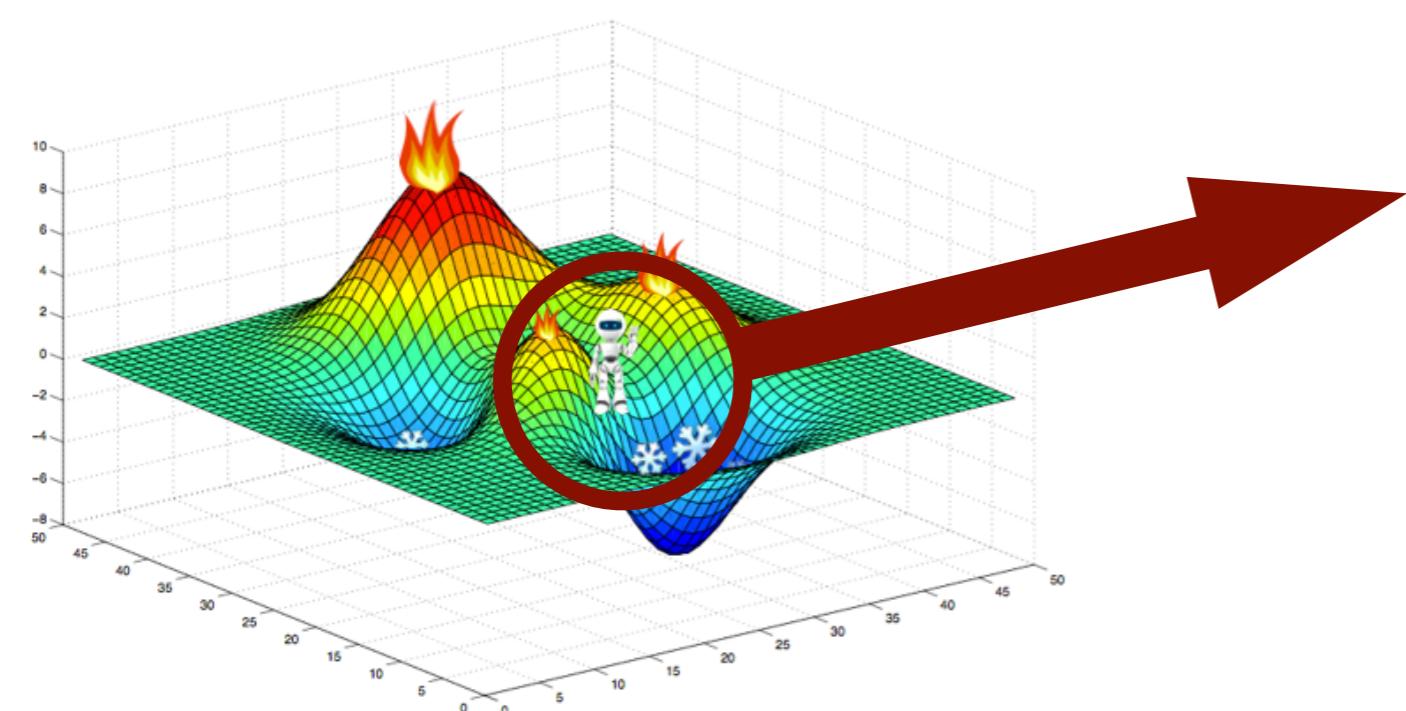
Complicando as Coisas...



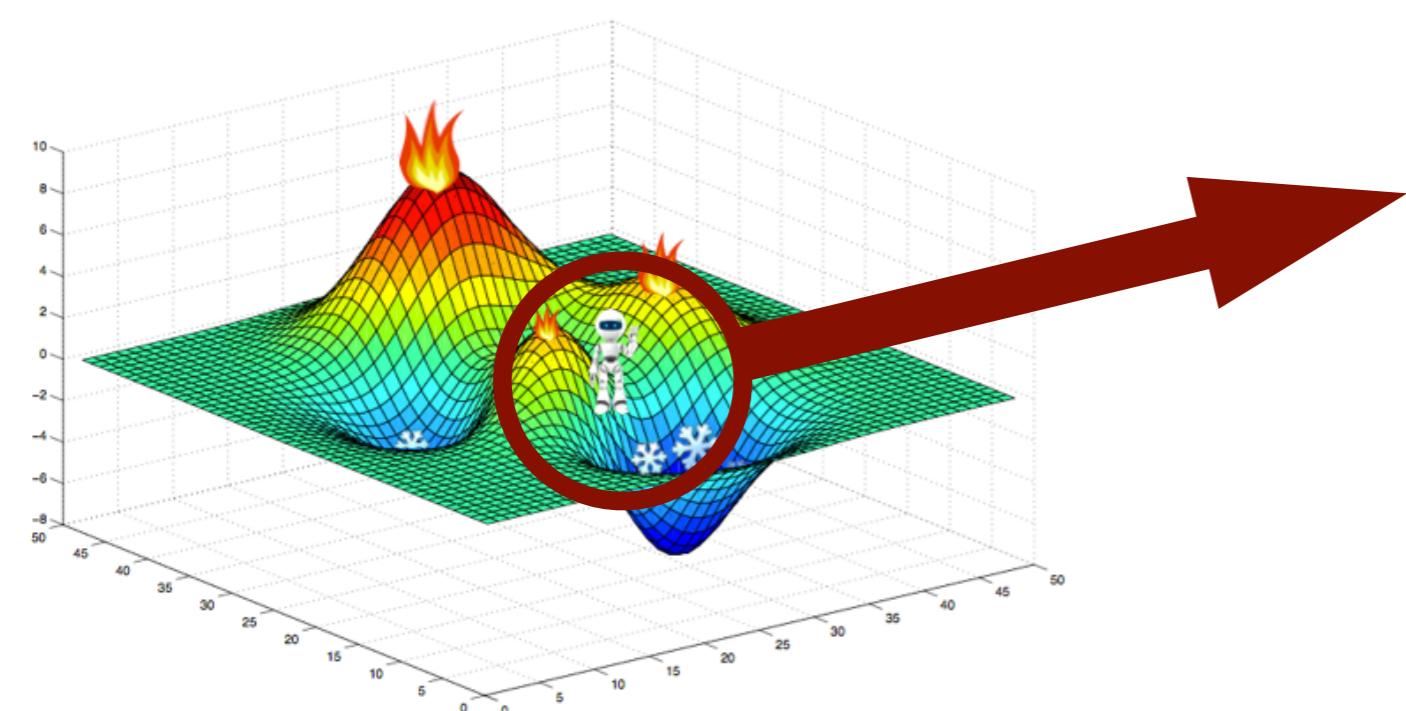
Complicando as Coisas...



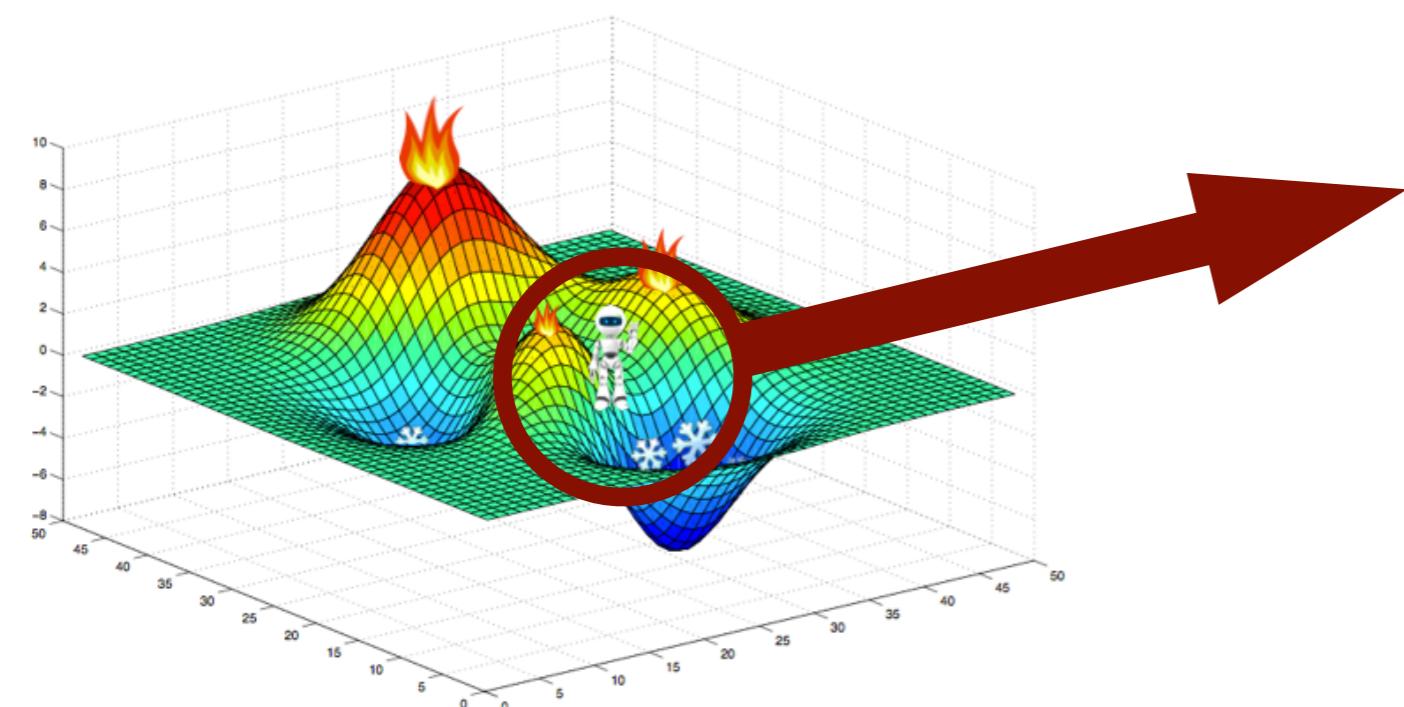
Complicando as Coisas...



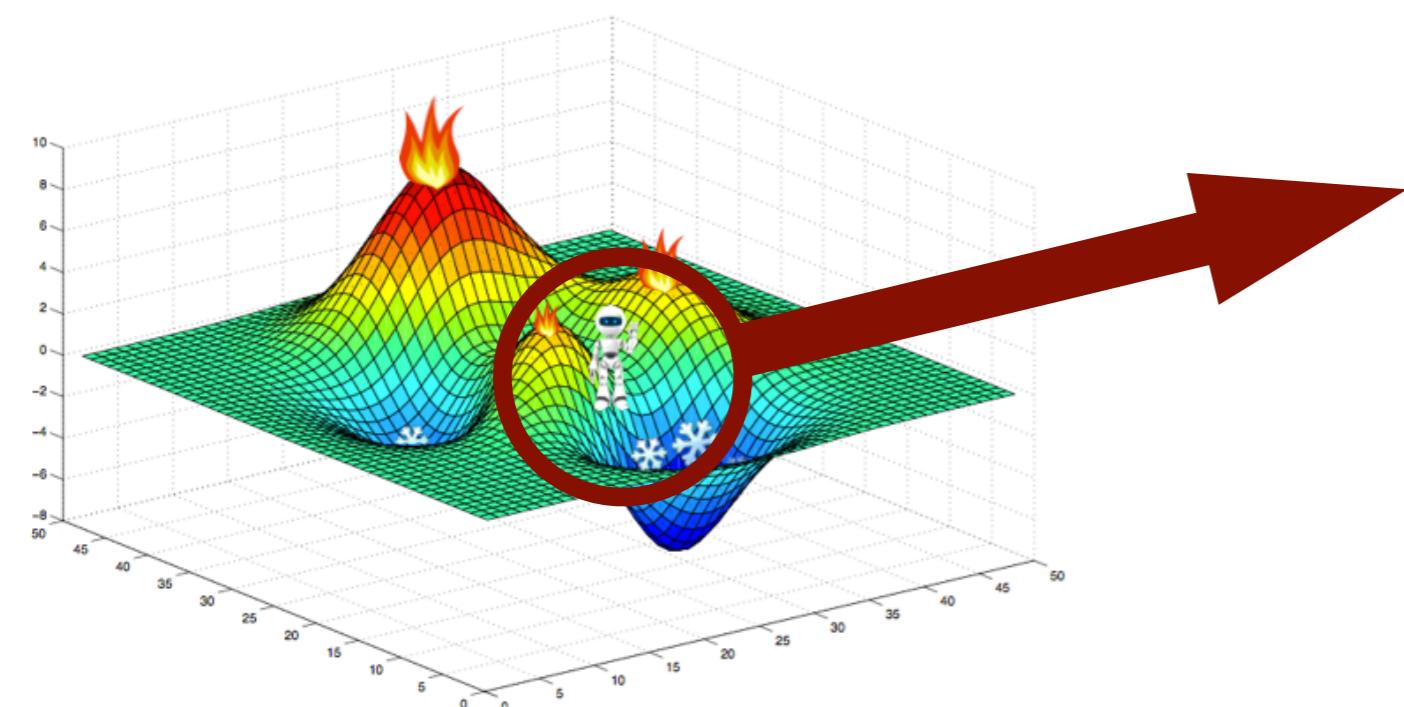
Complicando as Coisas...



Complicando as Coisas...

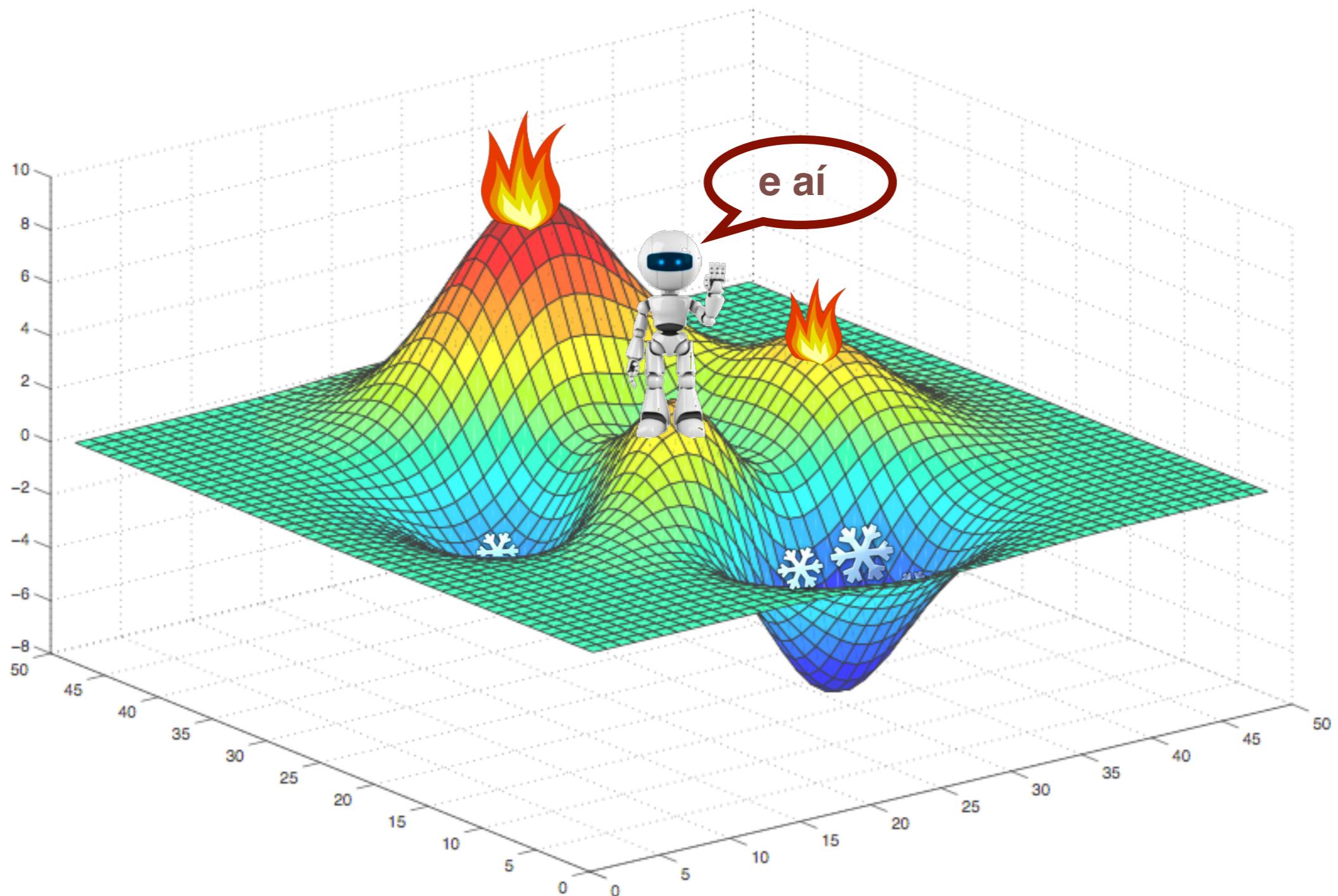


Complicando as Coisas...

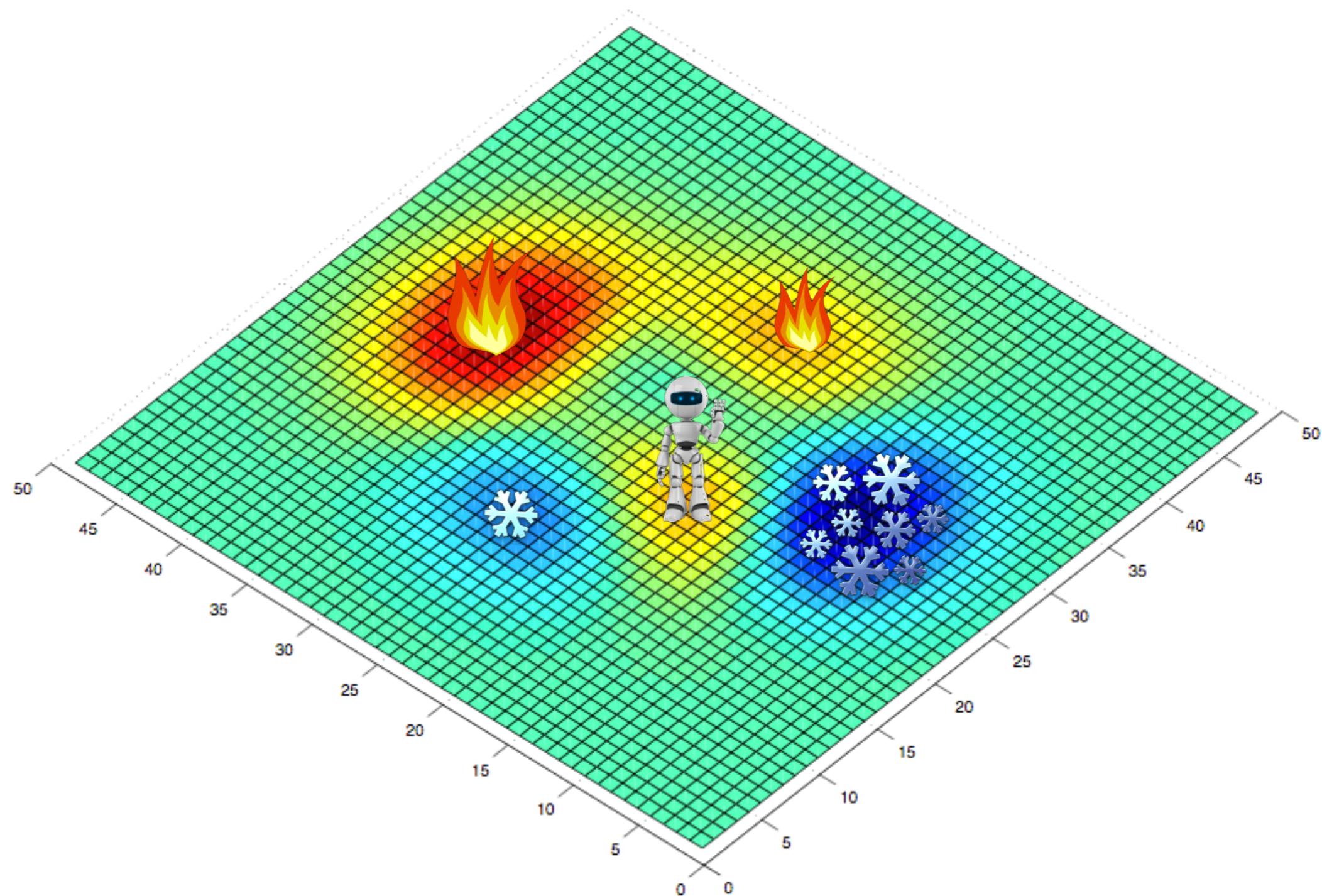


O que acontece quando o agente chega lá?

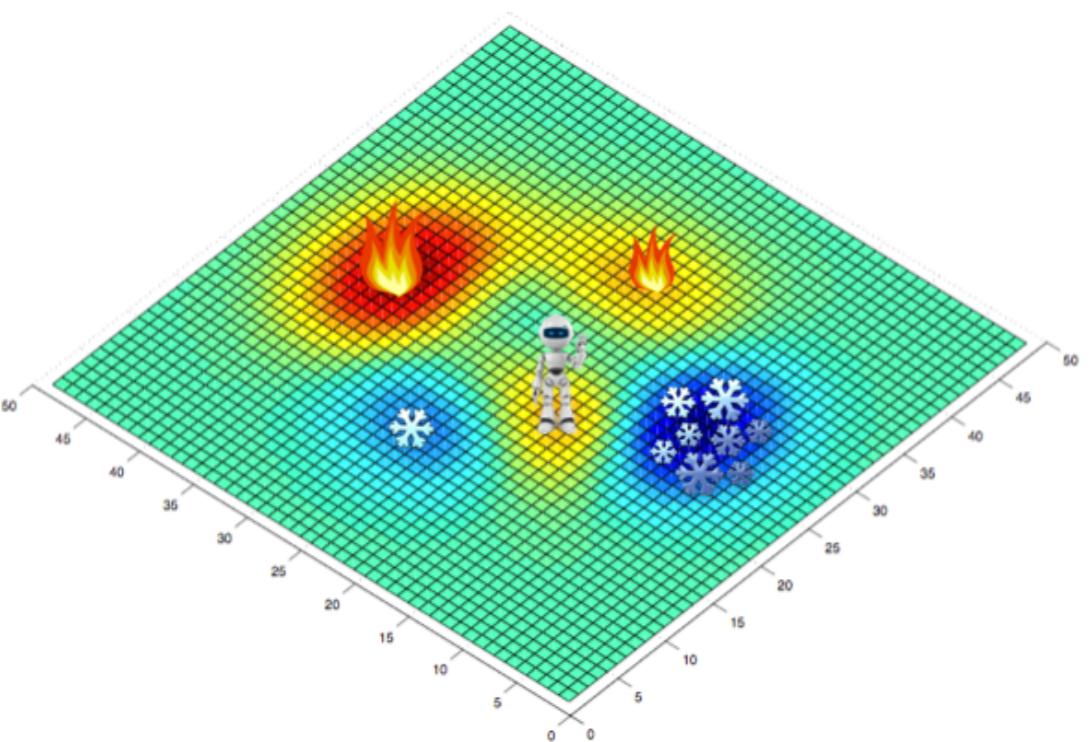
Complicando as Coisas...



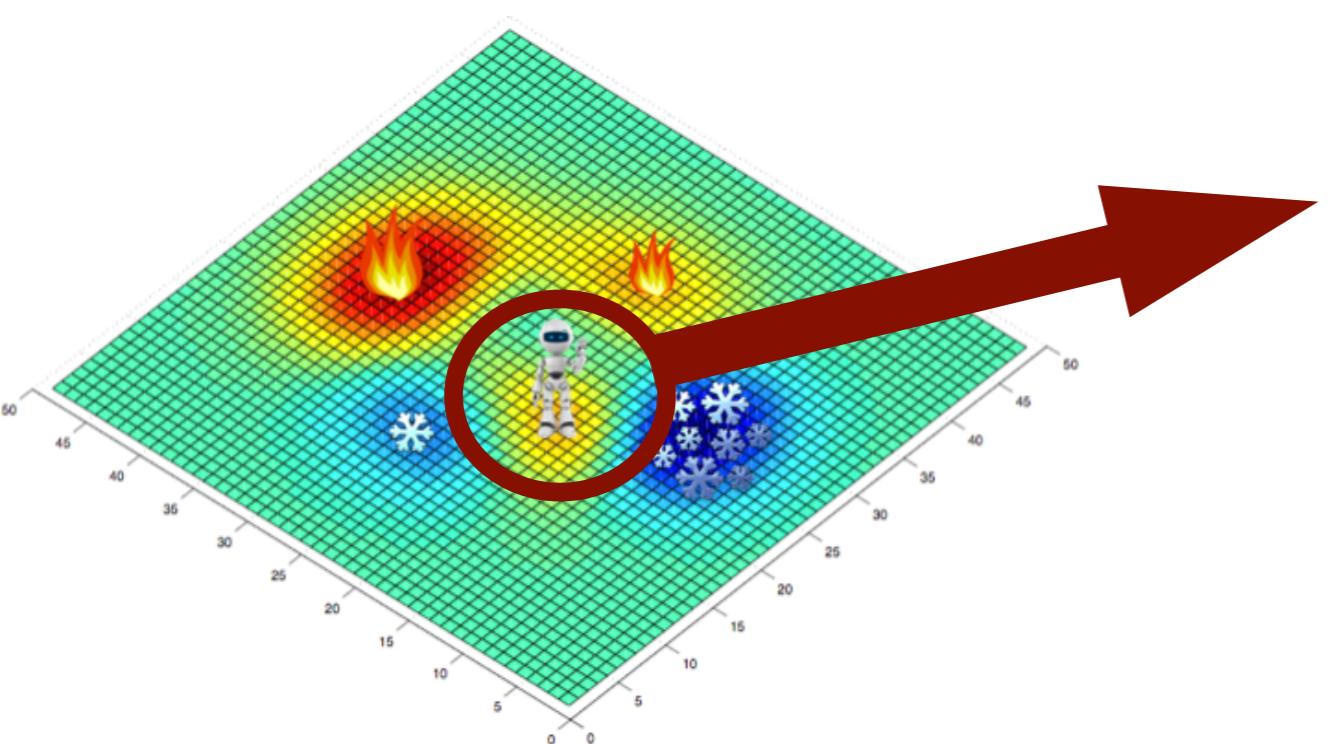
Complicando as Coisas...



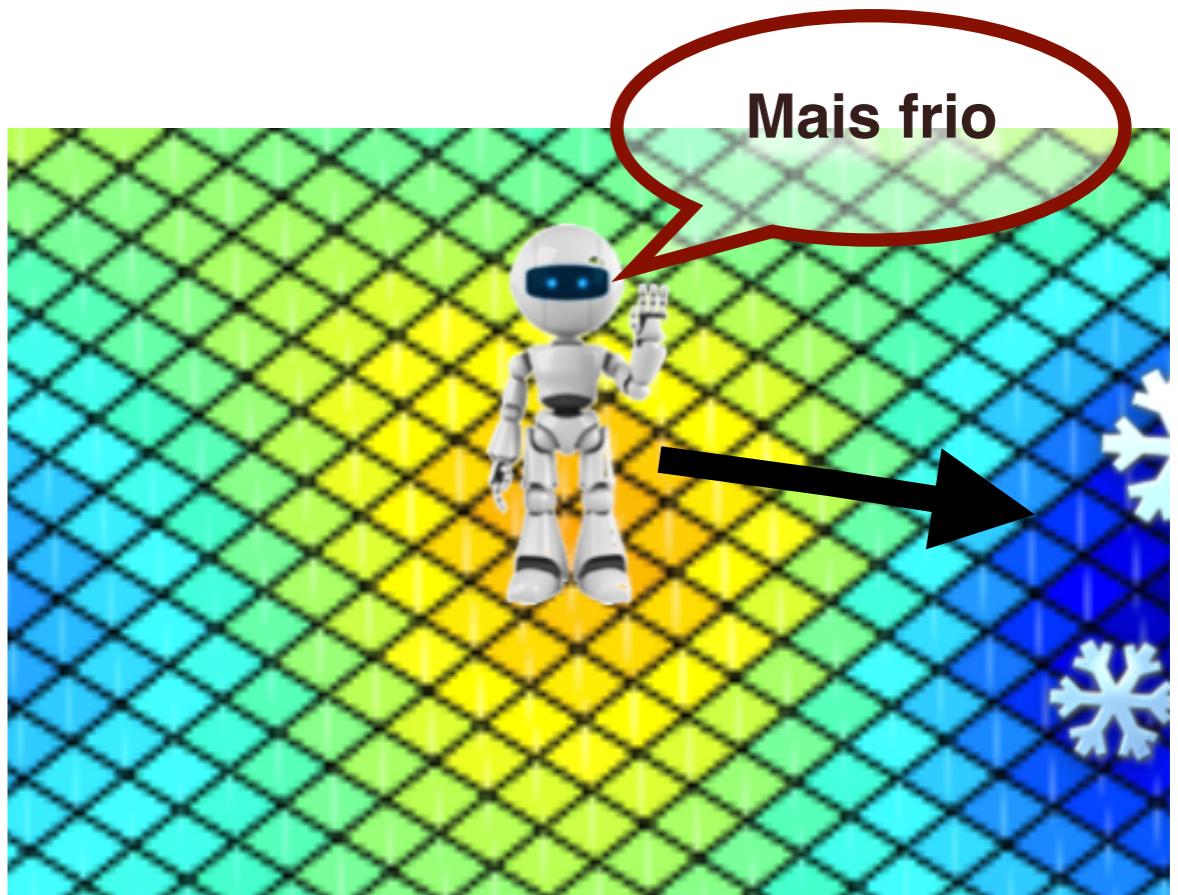
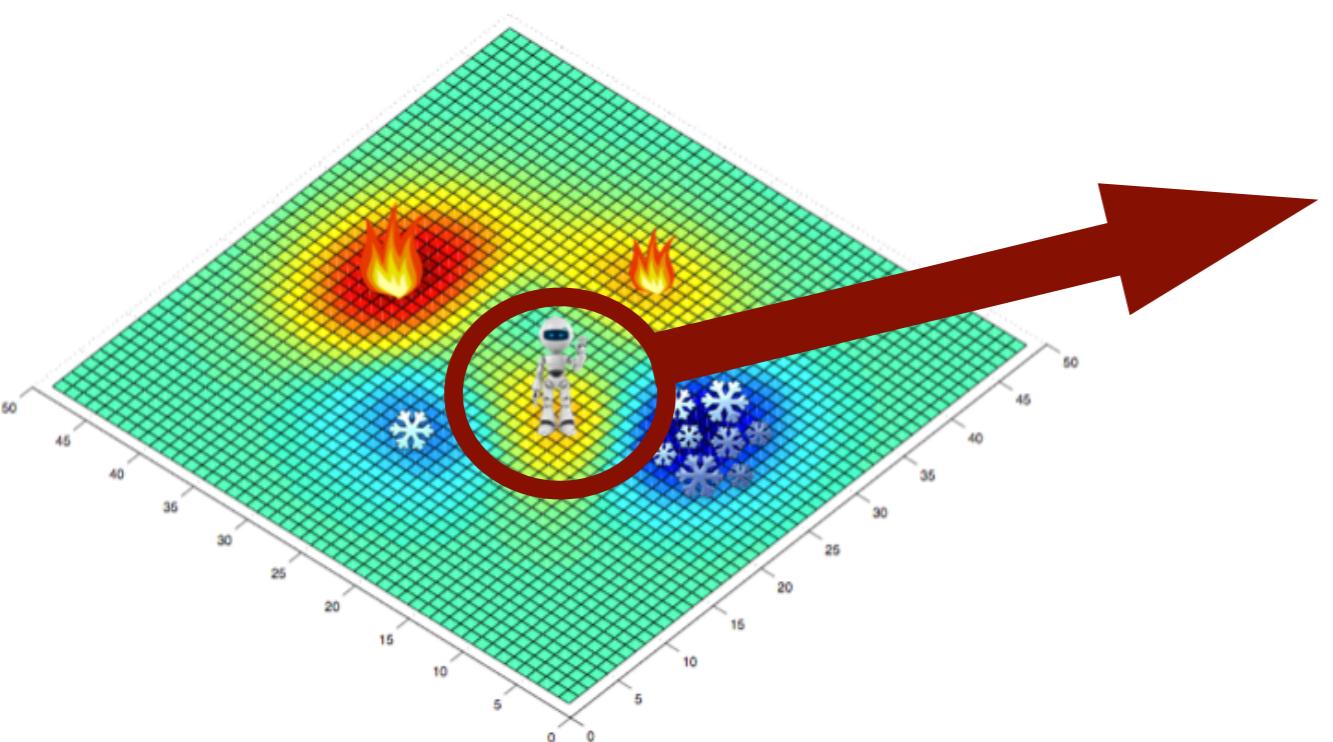
Complicando as Coisas...



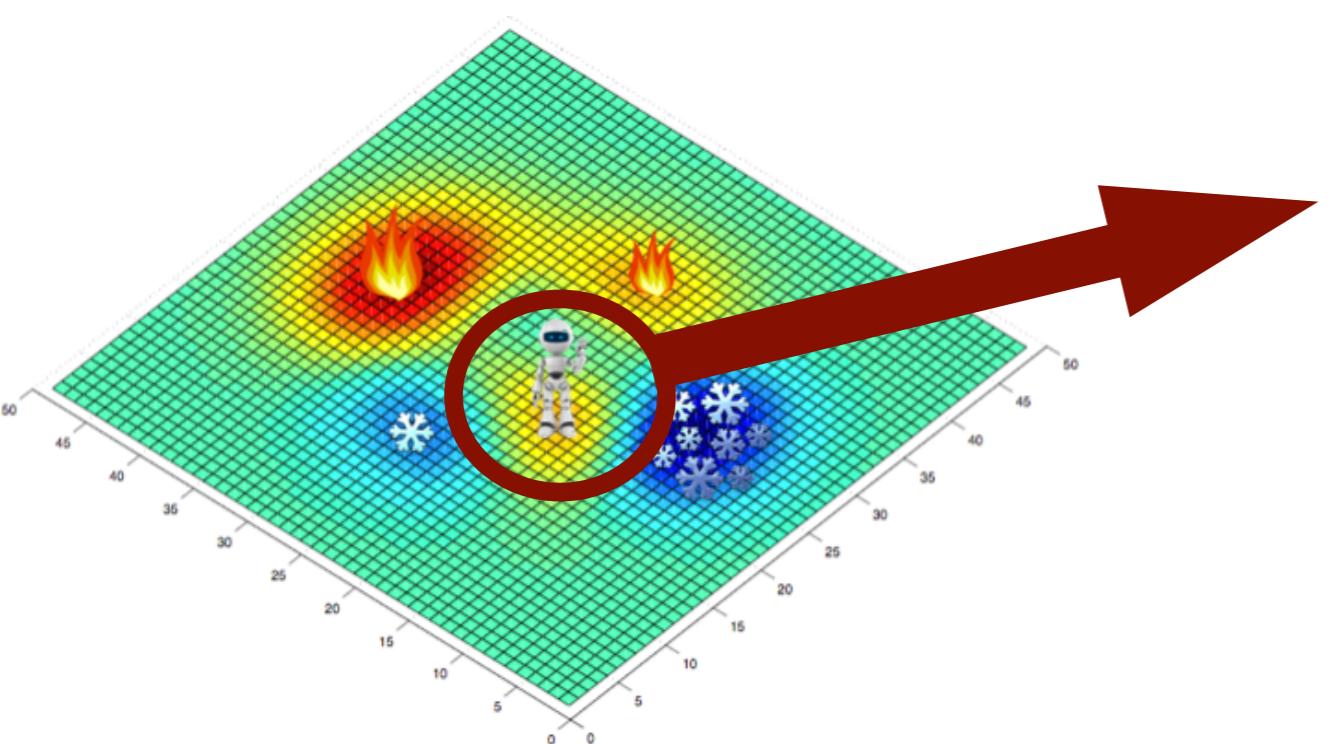
Complicando as Coisas...



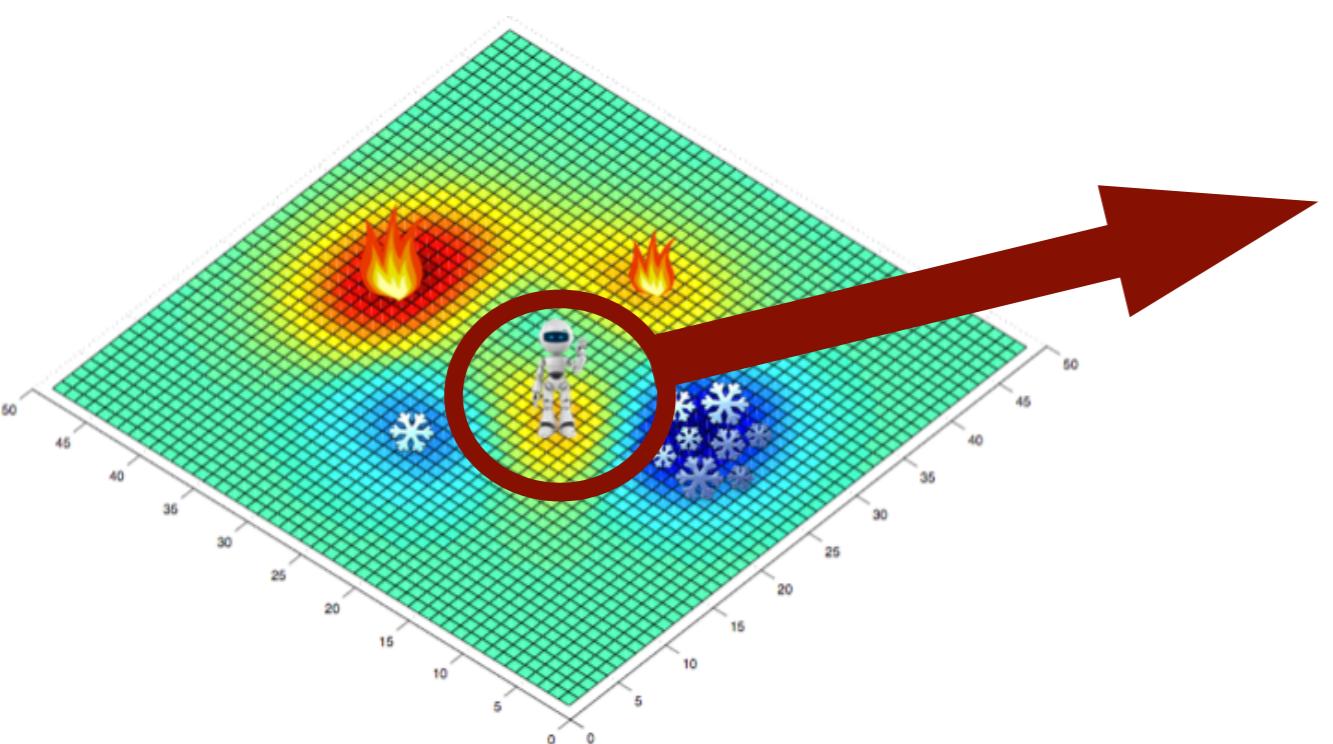
Complicando as Coisas...



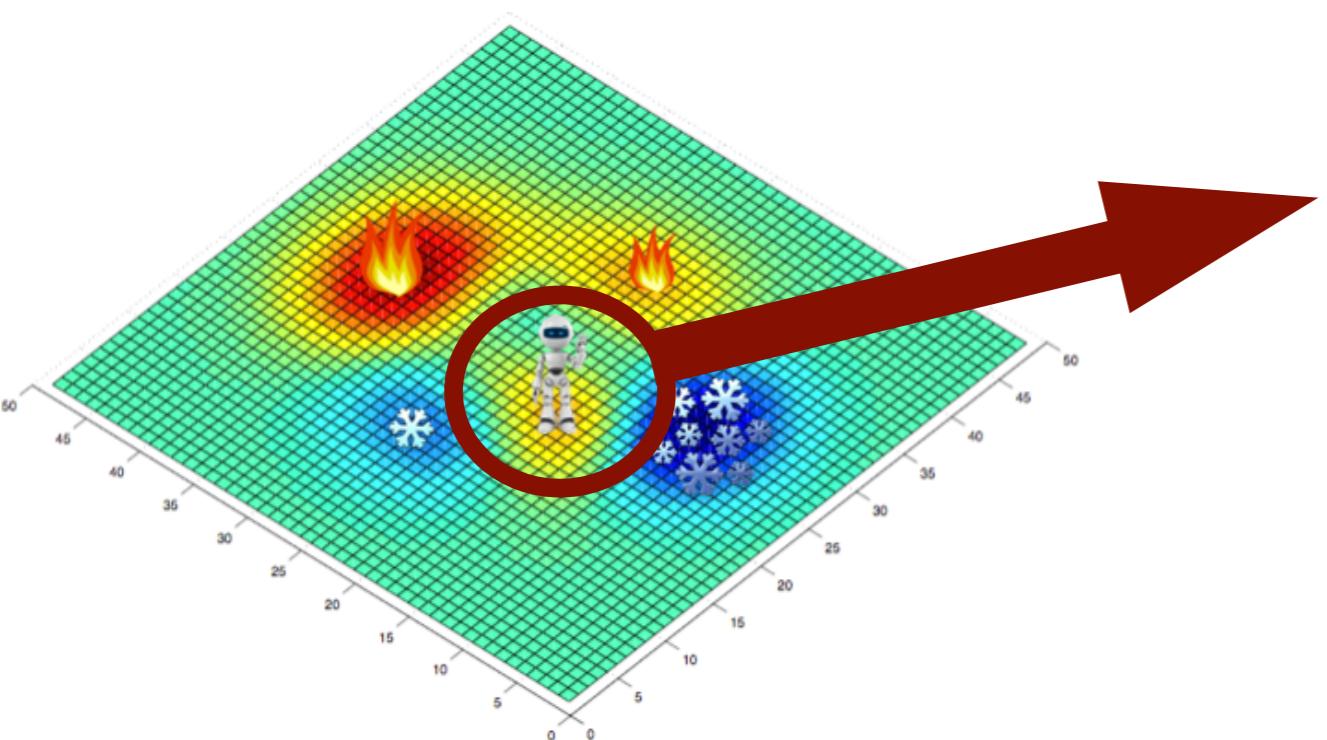
Complicando as Coisas...



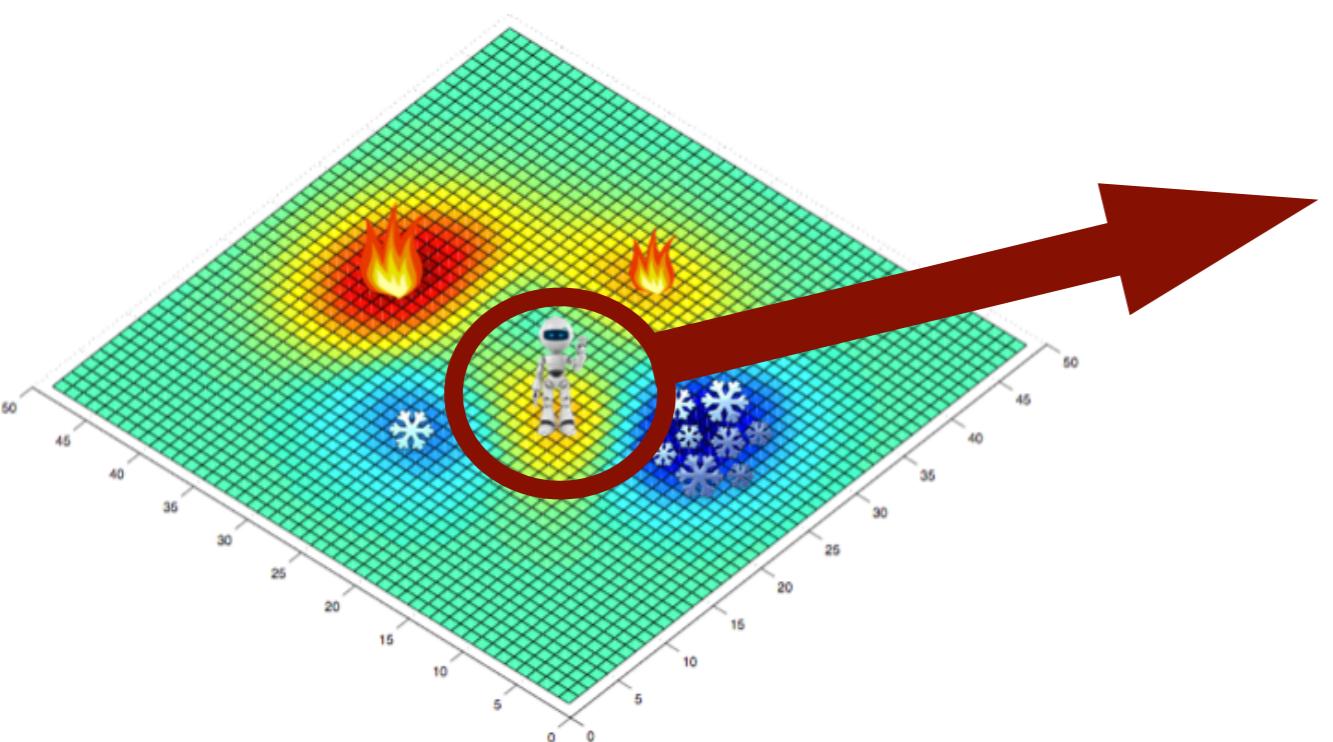
Complicando as Coisas...



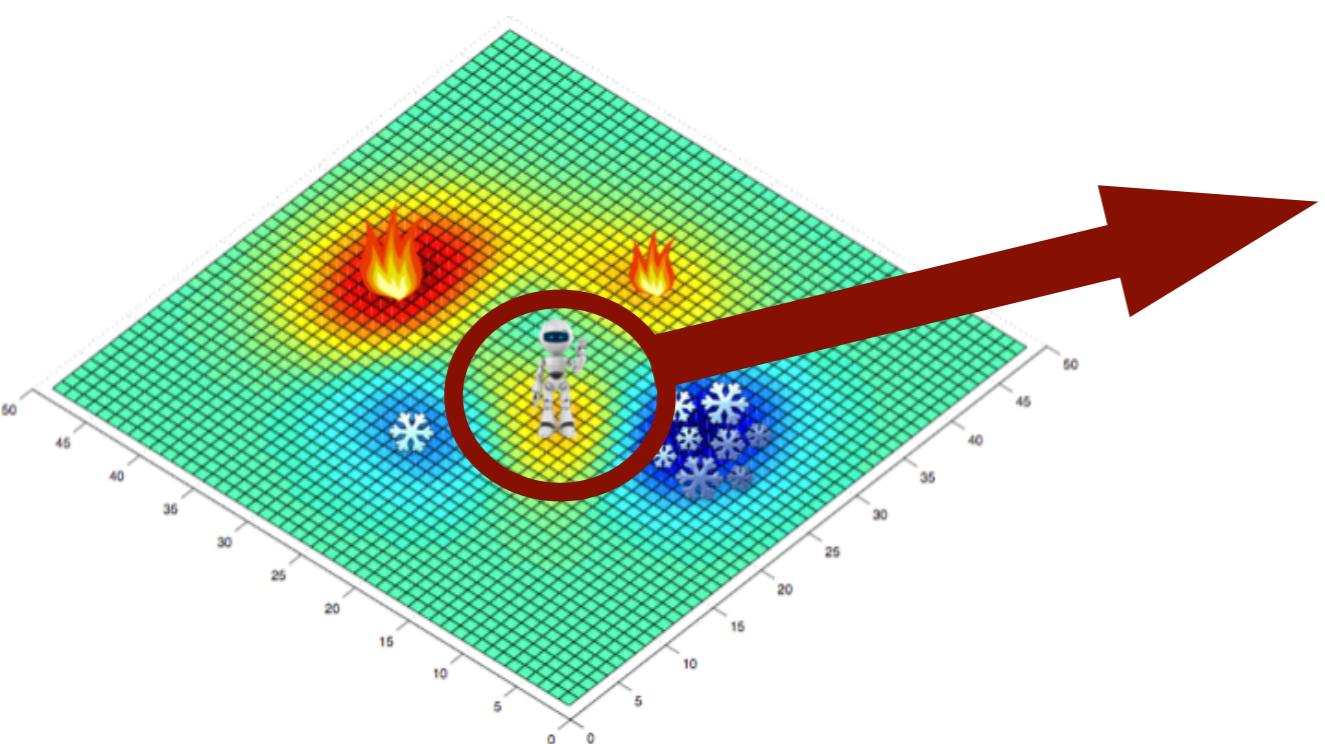
Complicando as Coisas...



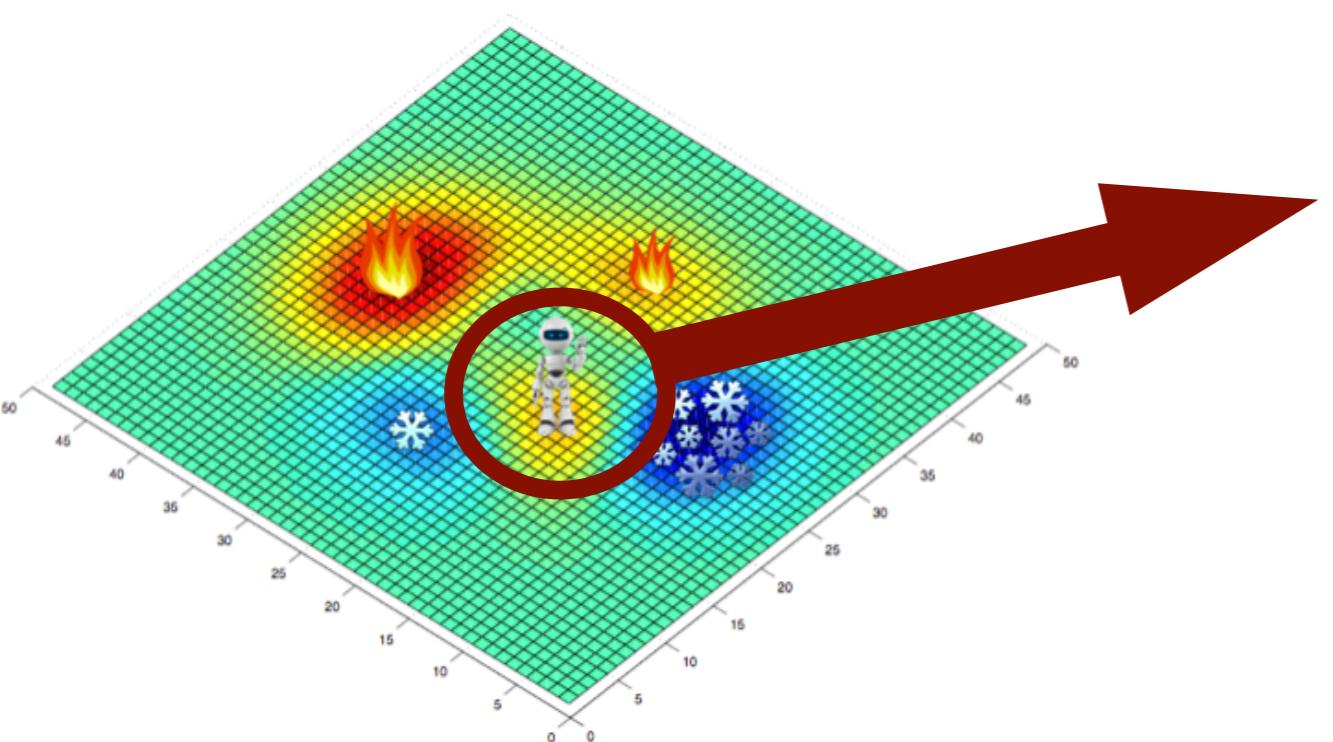
Complicando as Coisas...



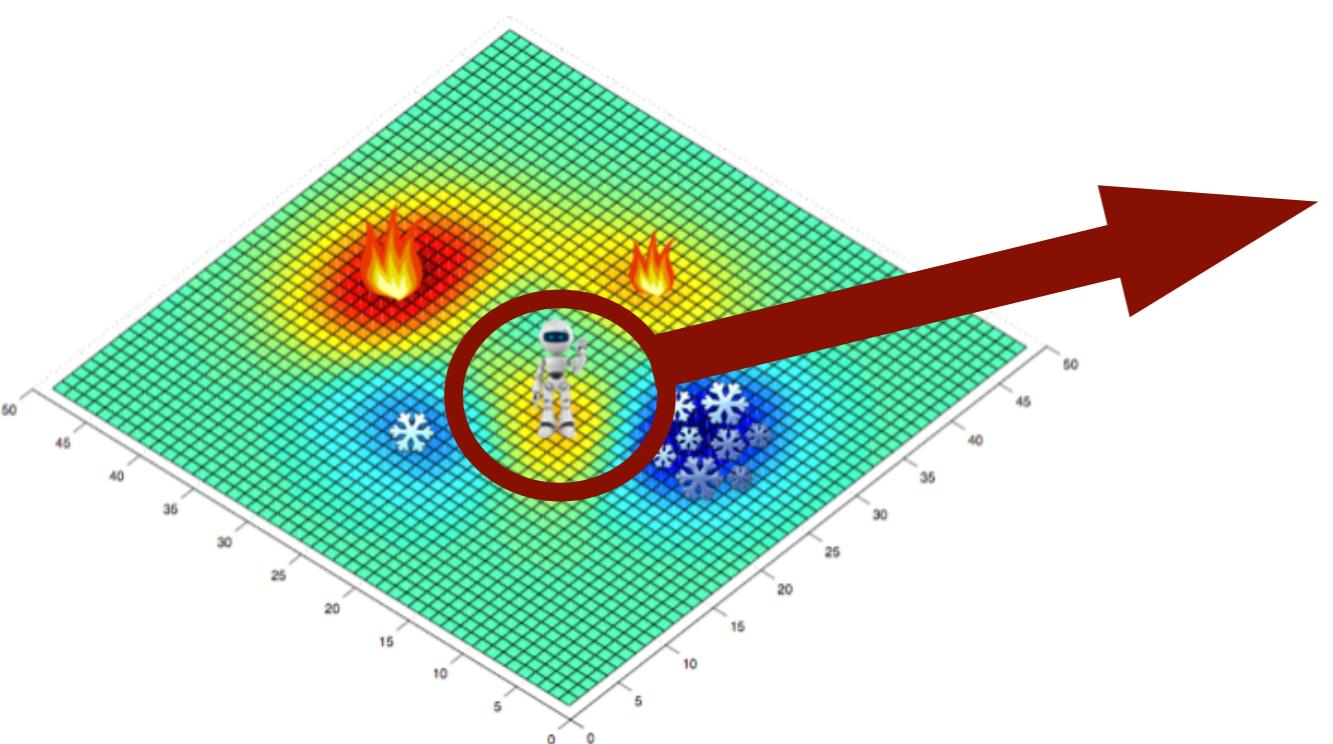
Complicando as Coisas...



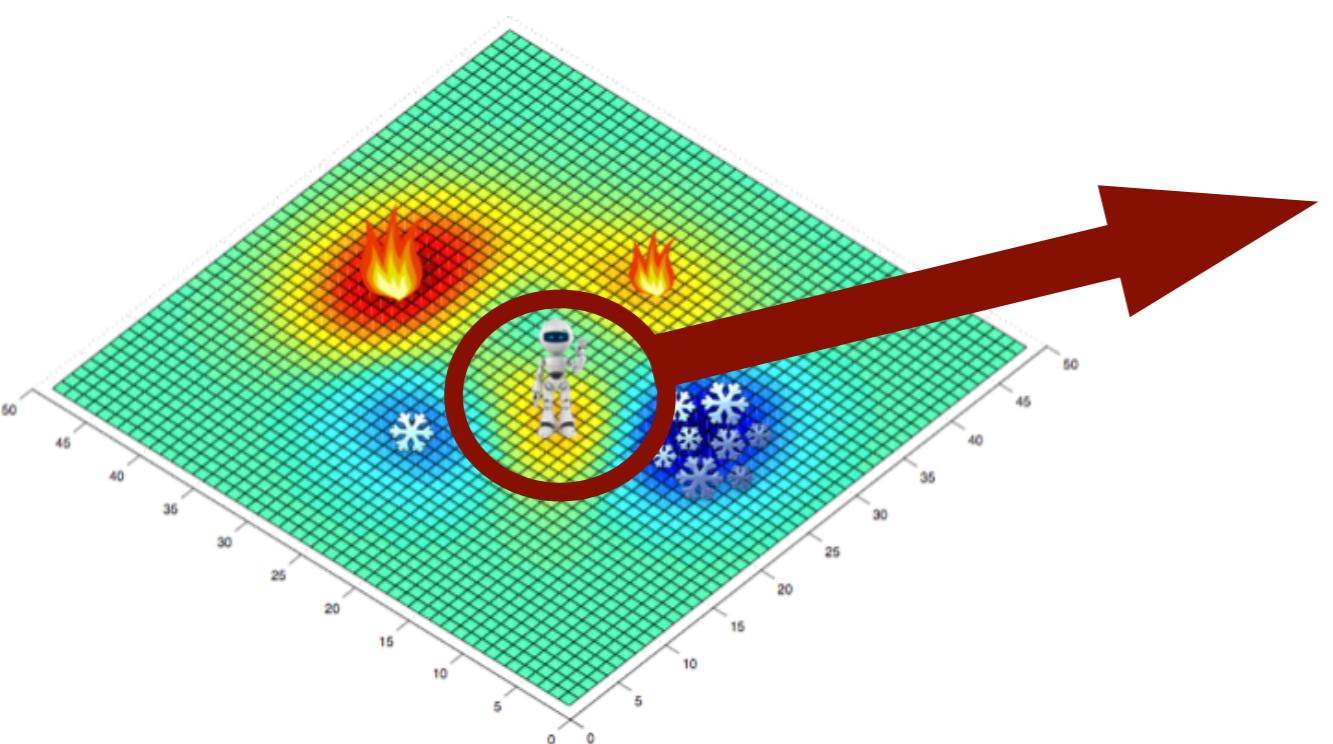
Complicando as Coisas...



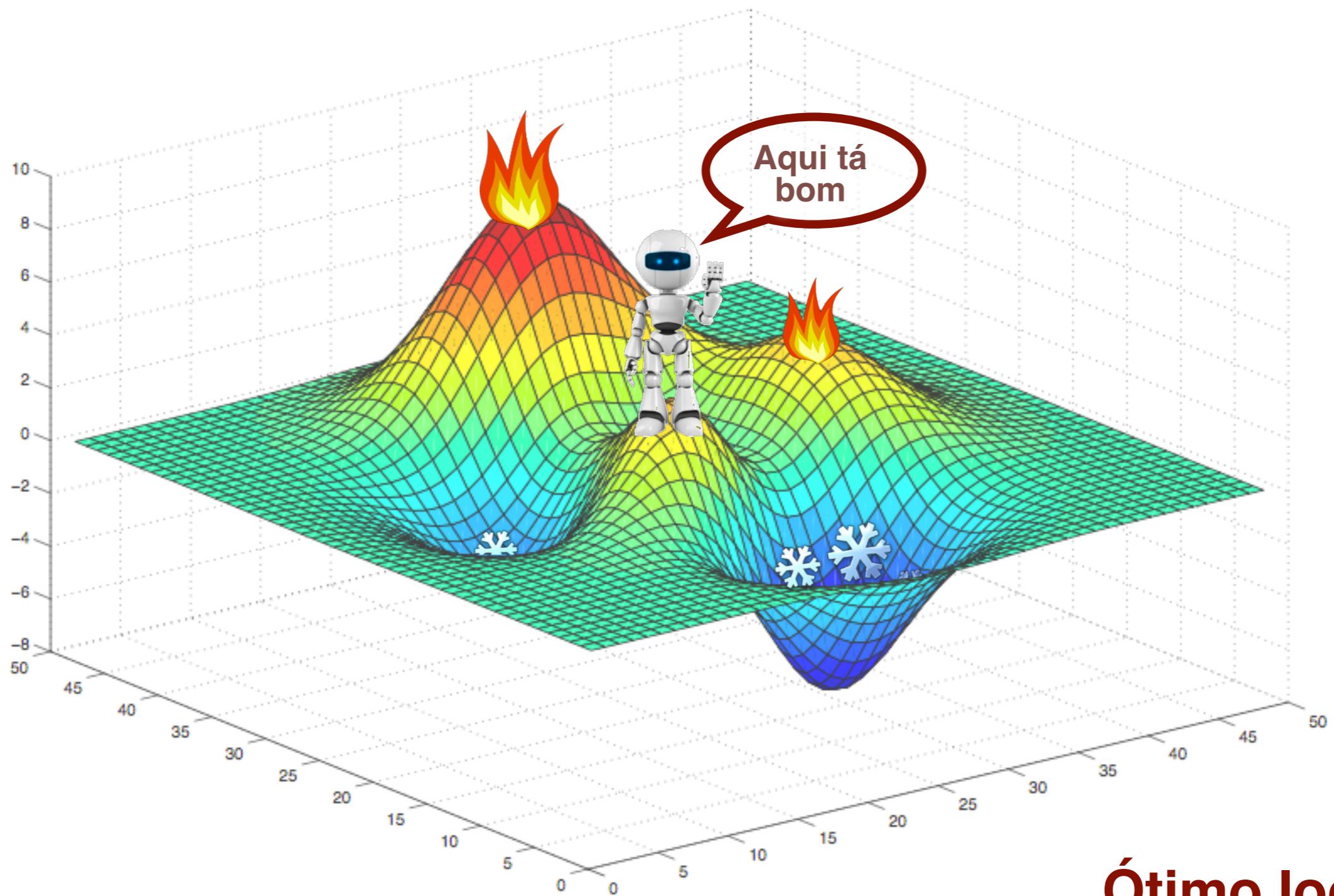
Complicando as Coisas...



Complicando as Coisas...



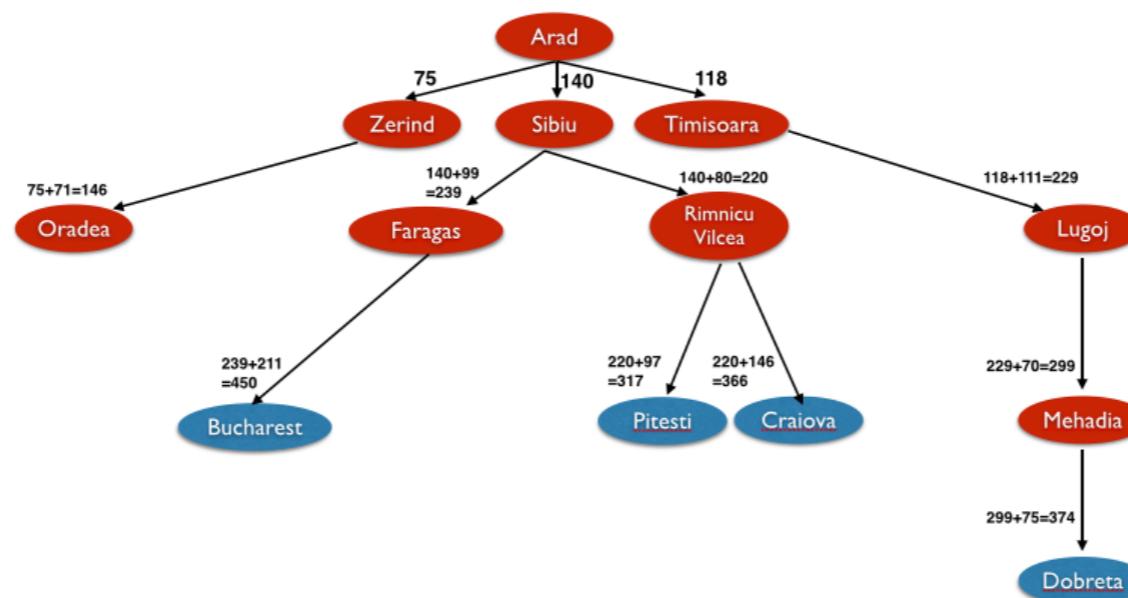
Complicando as Coisas...



Ótimo local !

Busca Local - Ótimo Local

- **Métodos de Busca Local** não mantêm em memória os vários caminhos a serem explorados



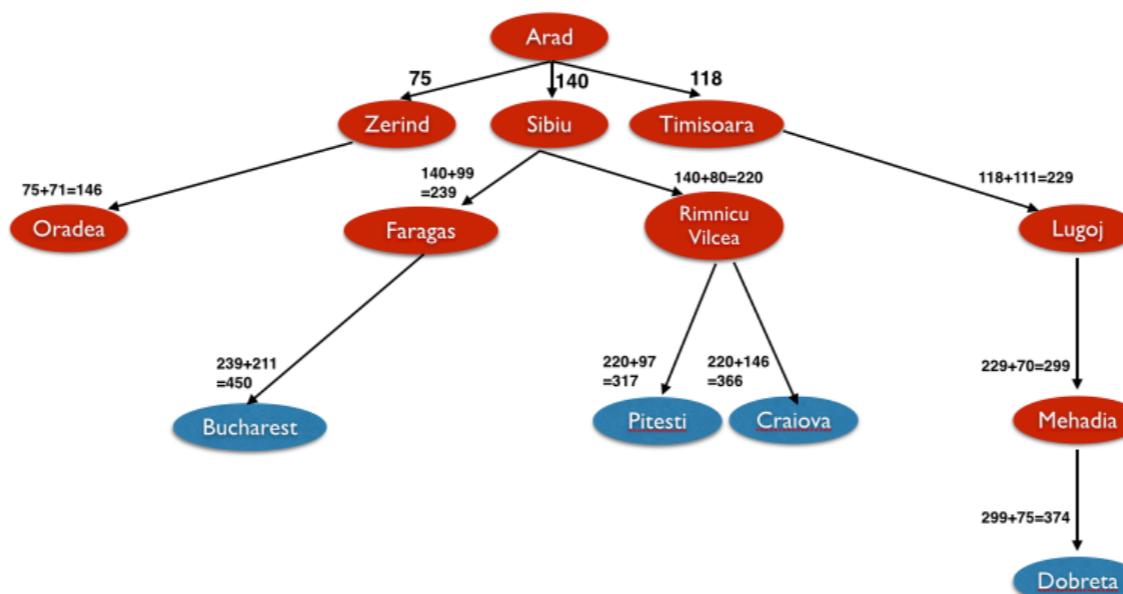
mantém apenas o estado atual (e talvez seus vizinhos imediatos)



risco de convergir/encontrar apenas um ótimo local

Busca Local - Ótimo Local

- **Métodos de Busca Local** não mantêm em memória os vários caminhos a serem explorados

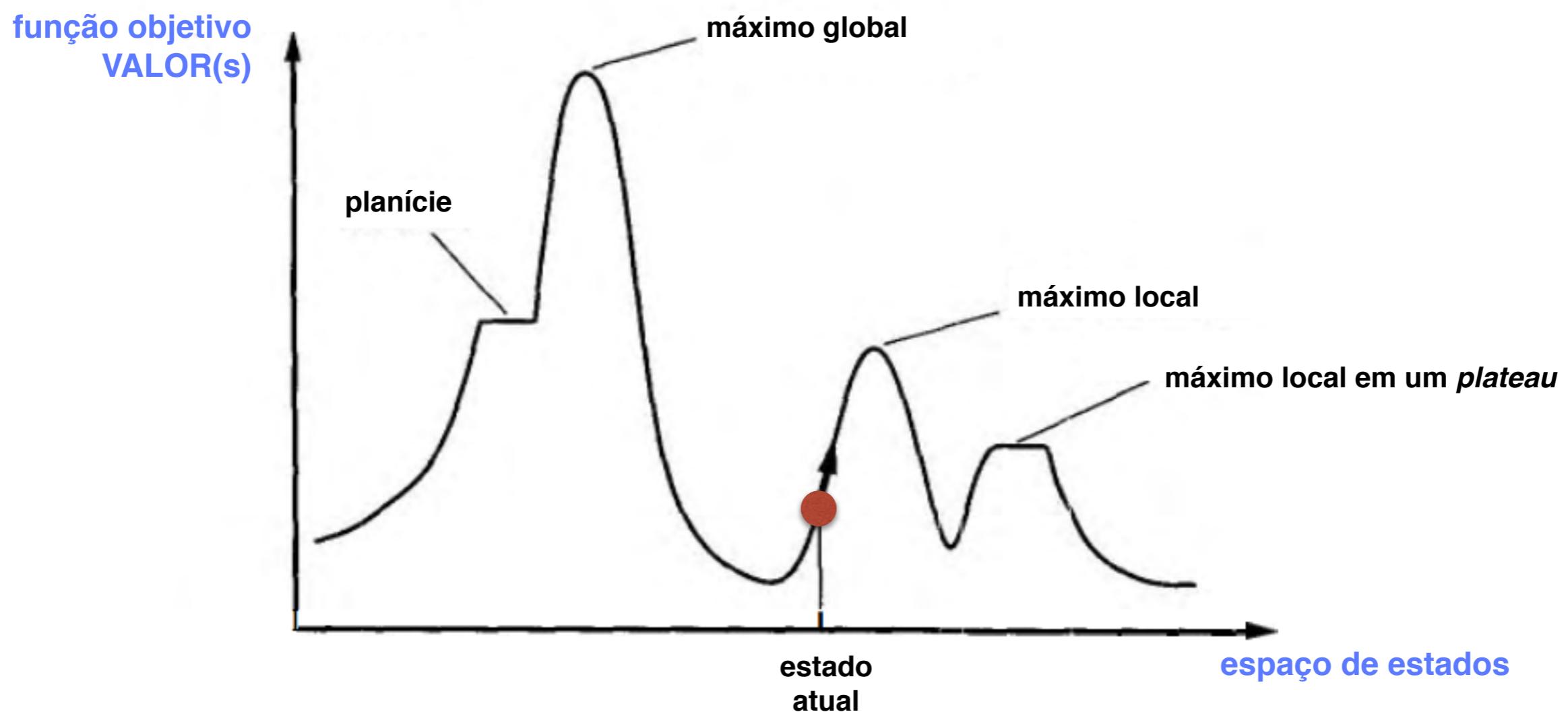


mantém apenas o estado atual (e talvez seus vizinhos imediatos)

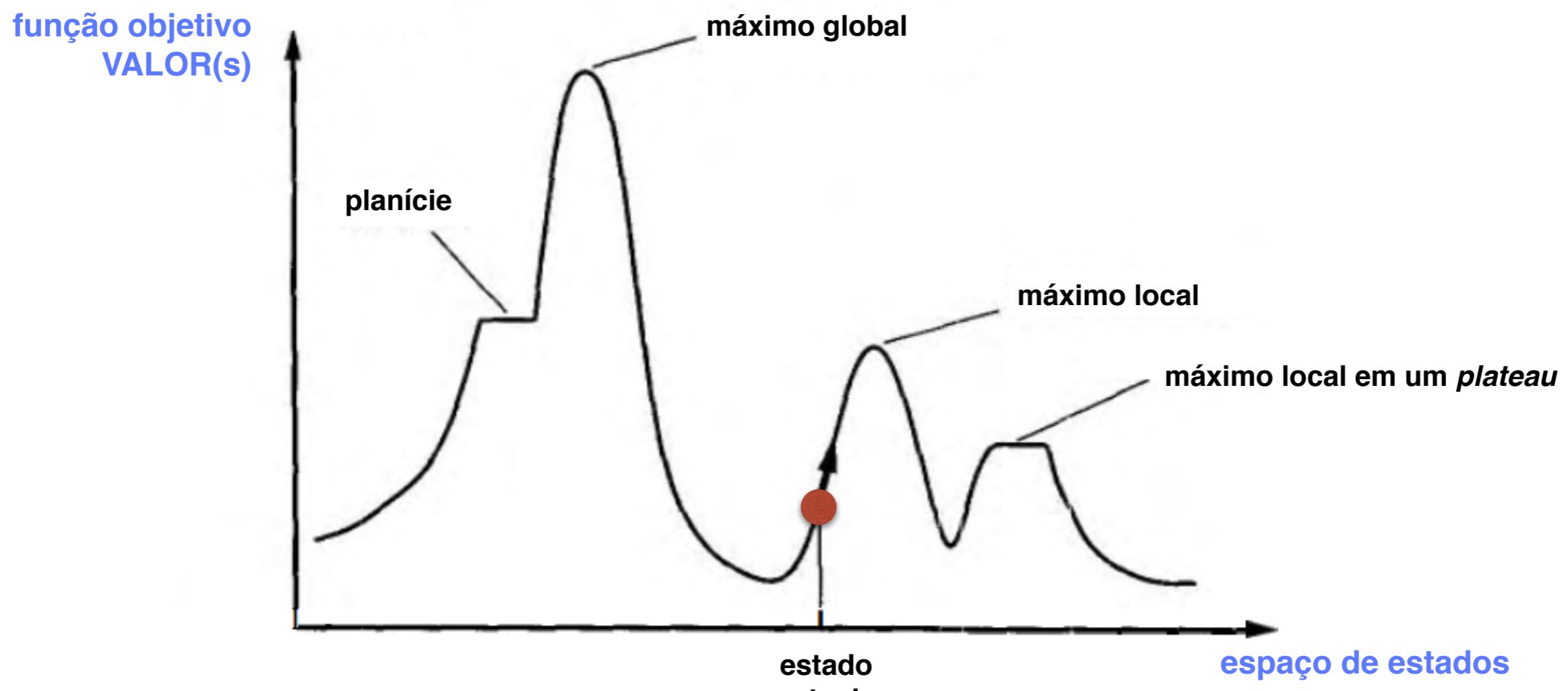
↗ risco de convergir/encontrar apenas um ótimo local

algum estado **s** que parece (*localmente*) a melhor escolha
mas que não é o melhor estado possível $s^* : \underset{s}{\operatorname{argmax}} \text{VALOR}(s)$

Busca Local

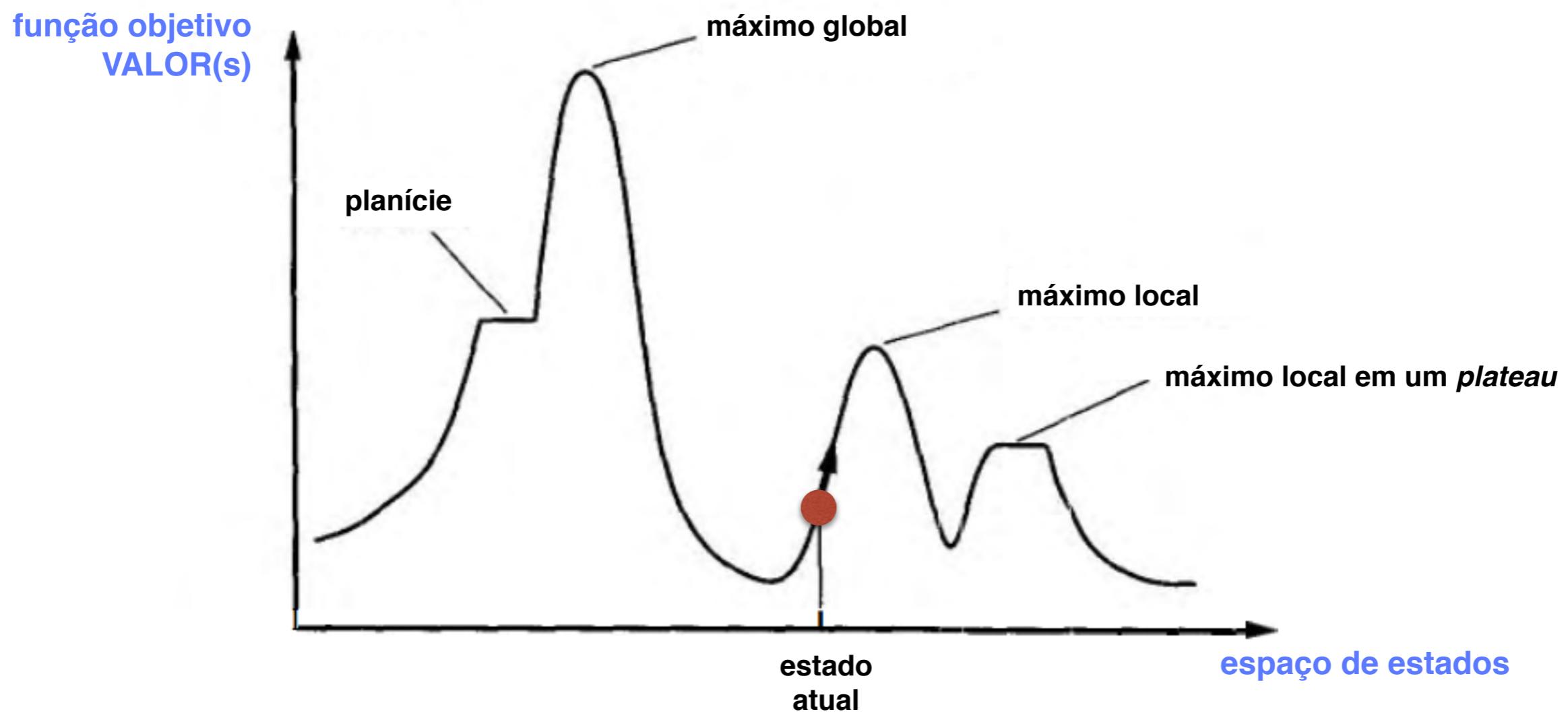


Busca Local



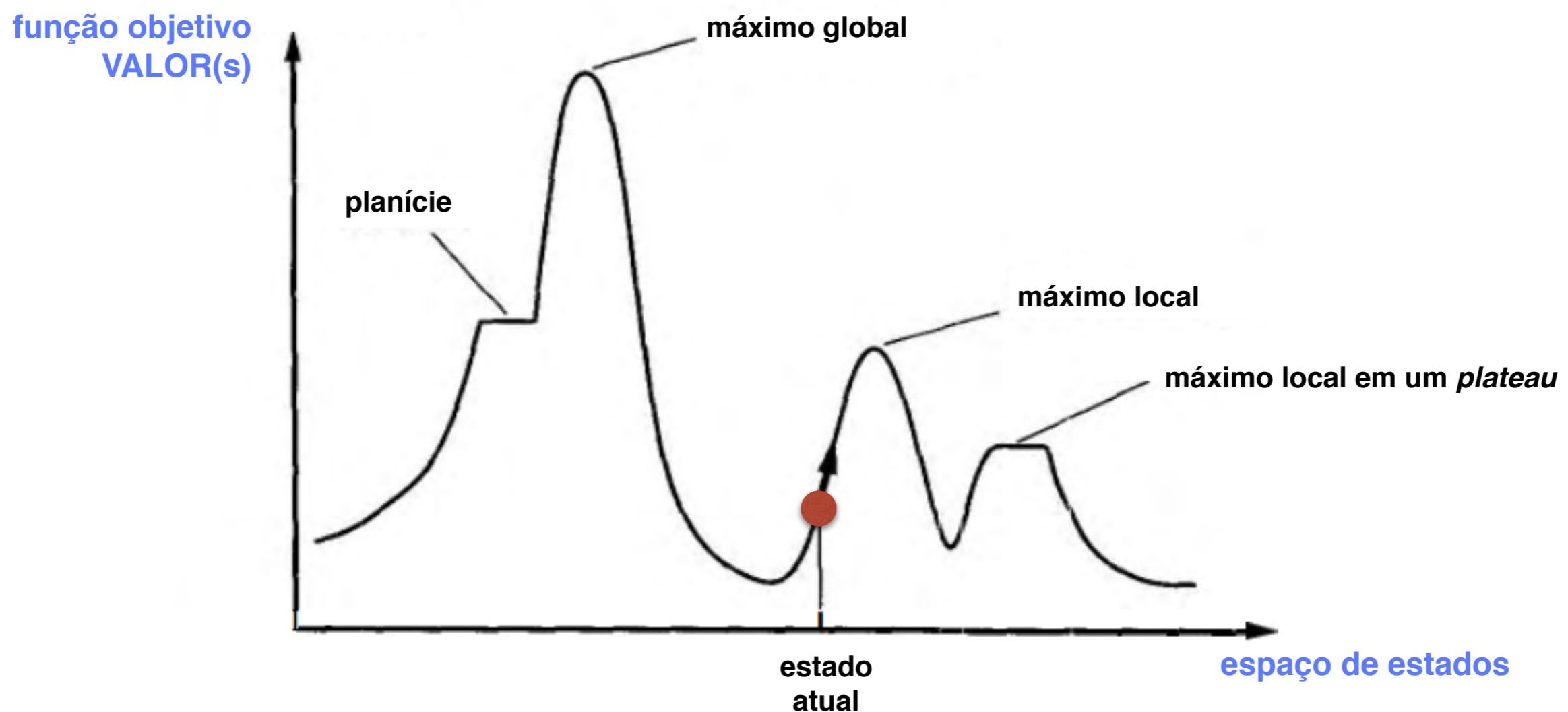
Como sabe que é um máximo local?

Busca Local



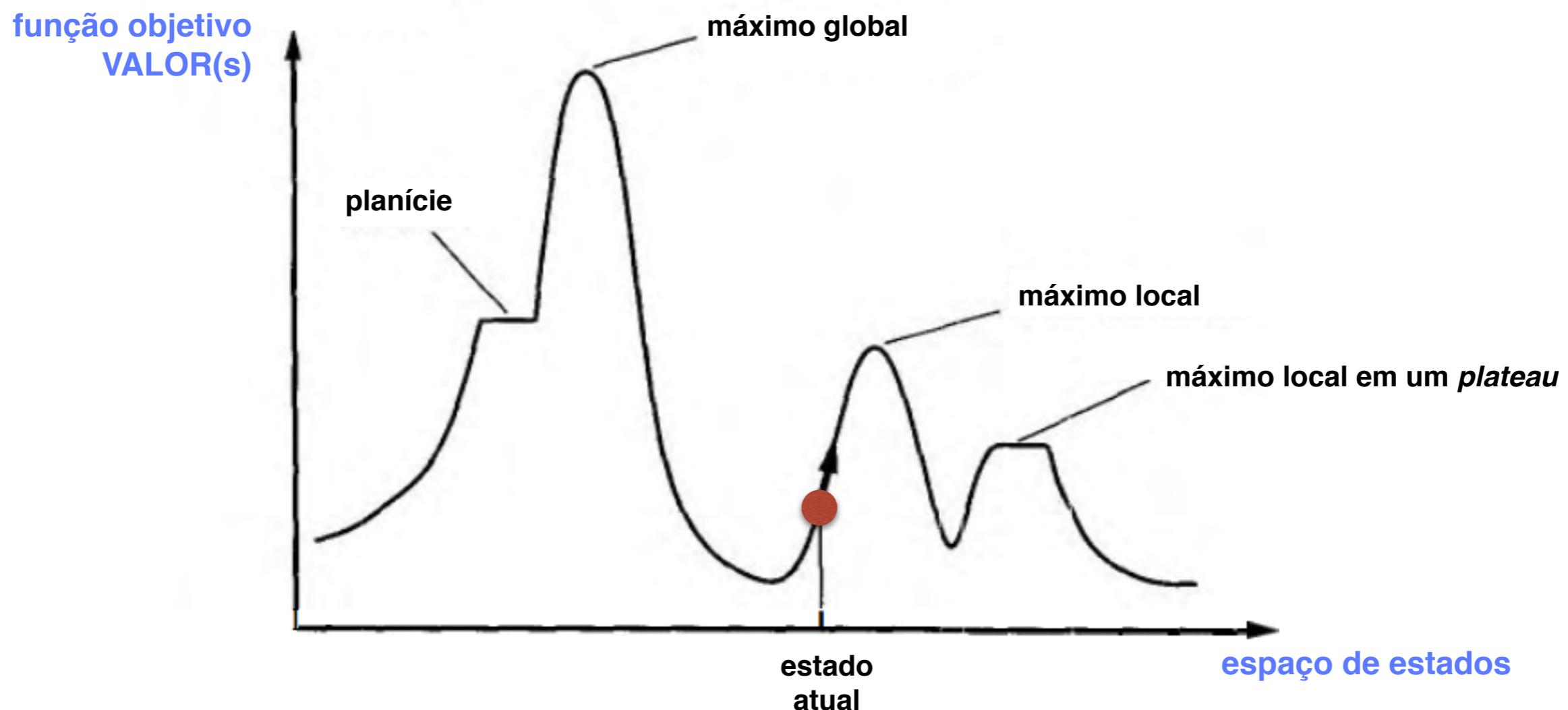
Como sabe que é um máximo local? **Nenhum vizinho é melhor**

Busca Local



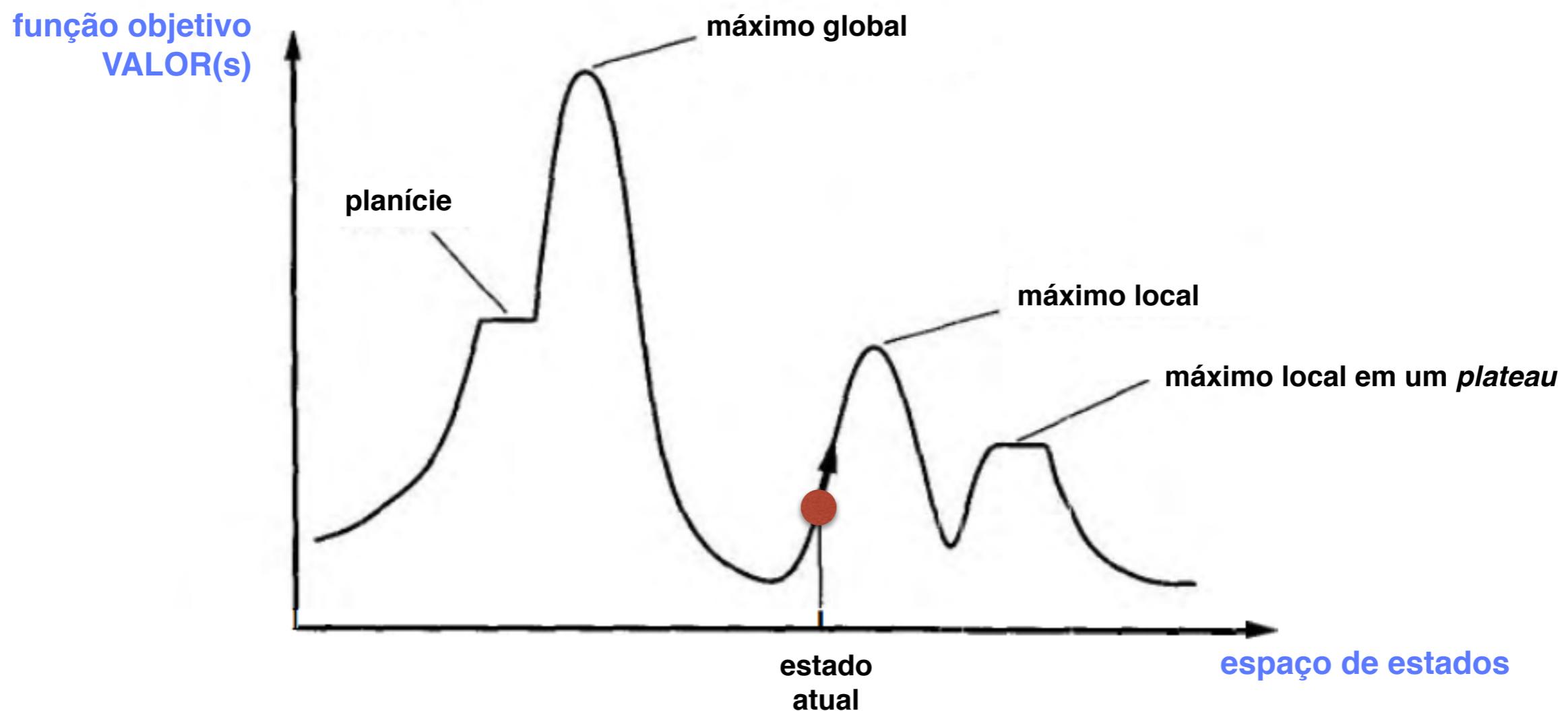
Como sabe que é um máximo local? **Nenhum vizinho é melhor**
Logo, o máximo global também é um máximo local!
(exceto que nenhum outro estado é melhor)

Busca Local



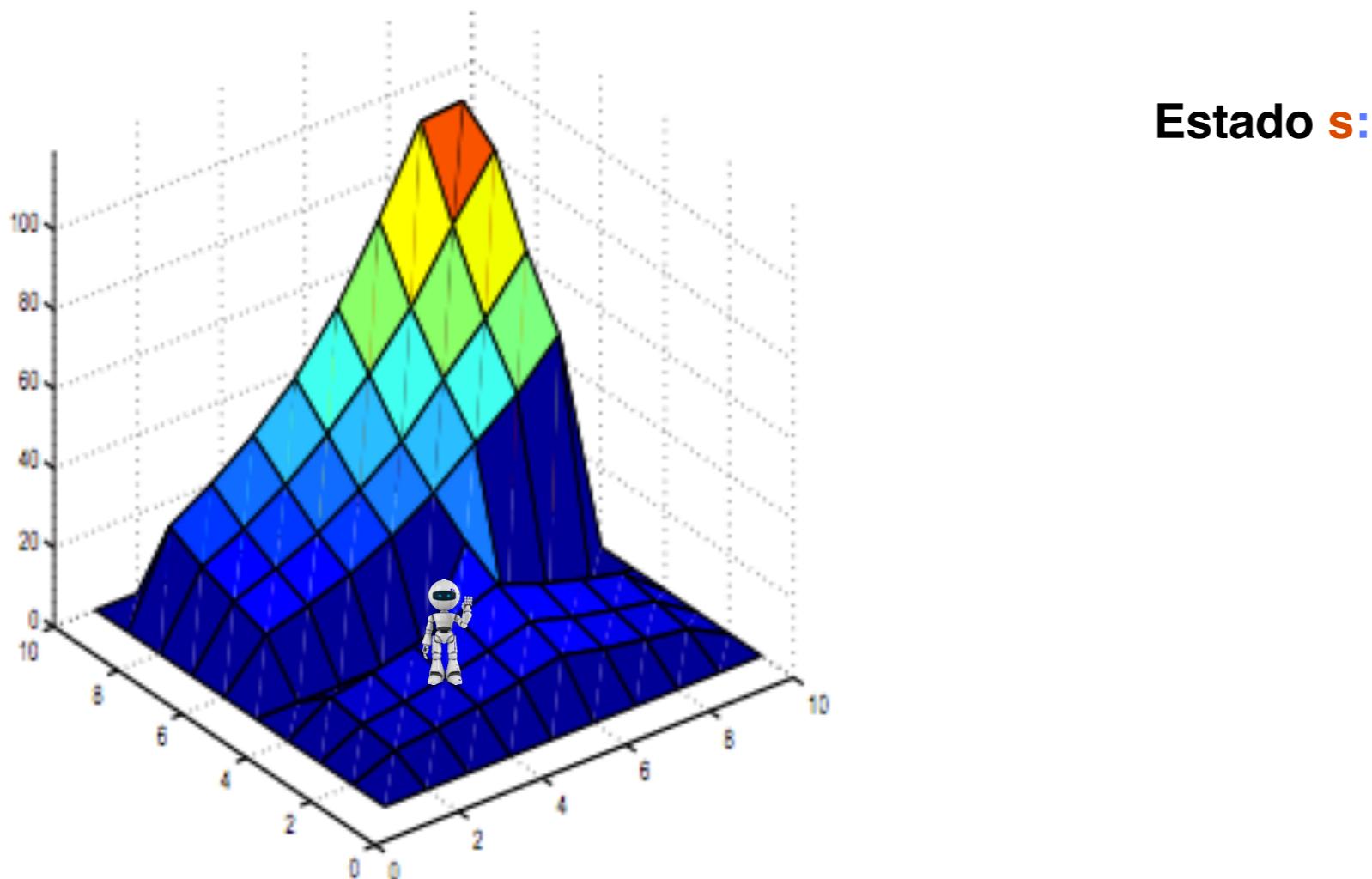
Como sabe que é um máximo local? **Nenhum vizinho é melhor**
Logo, o máximo global também é um máximo local!
(exceto que nenhum outro estado é melhor)
Como sabe que é um máximo global?

Busca Local



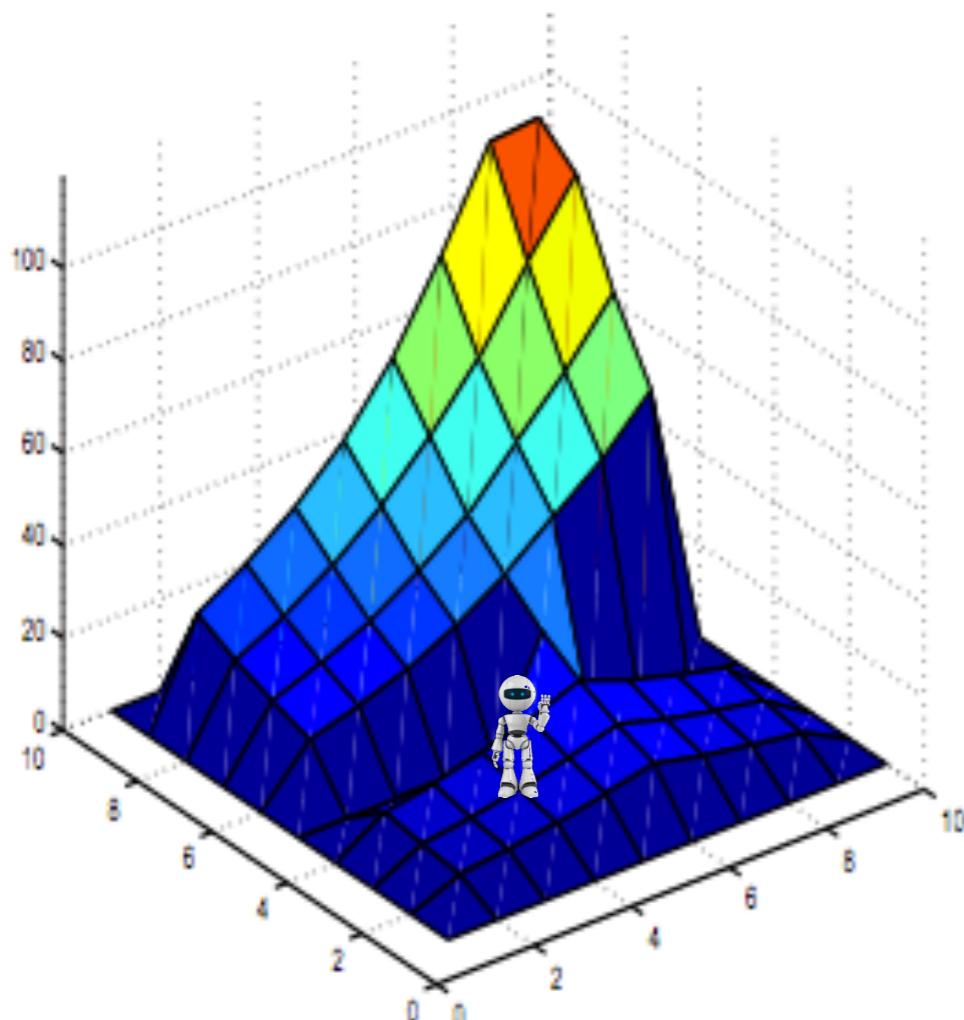
Como sabe que é um máximo local? **Nenhum vizinho é melhor**
Logo, o máximo global também é um máximo local!
(exceto que nenhum outro estado é melhor)
Como sabe que é um máximo global? **Não sabe...**

Busca Local



Estado **s**:

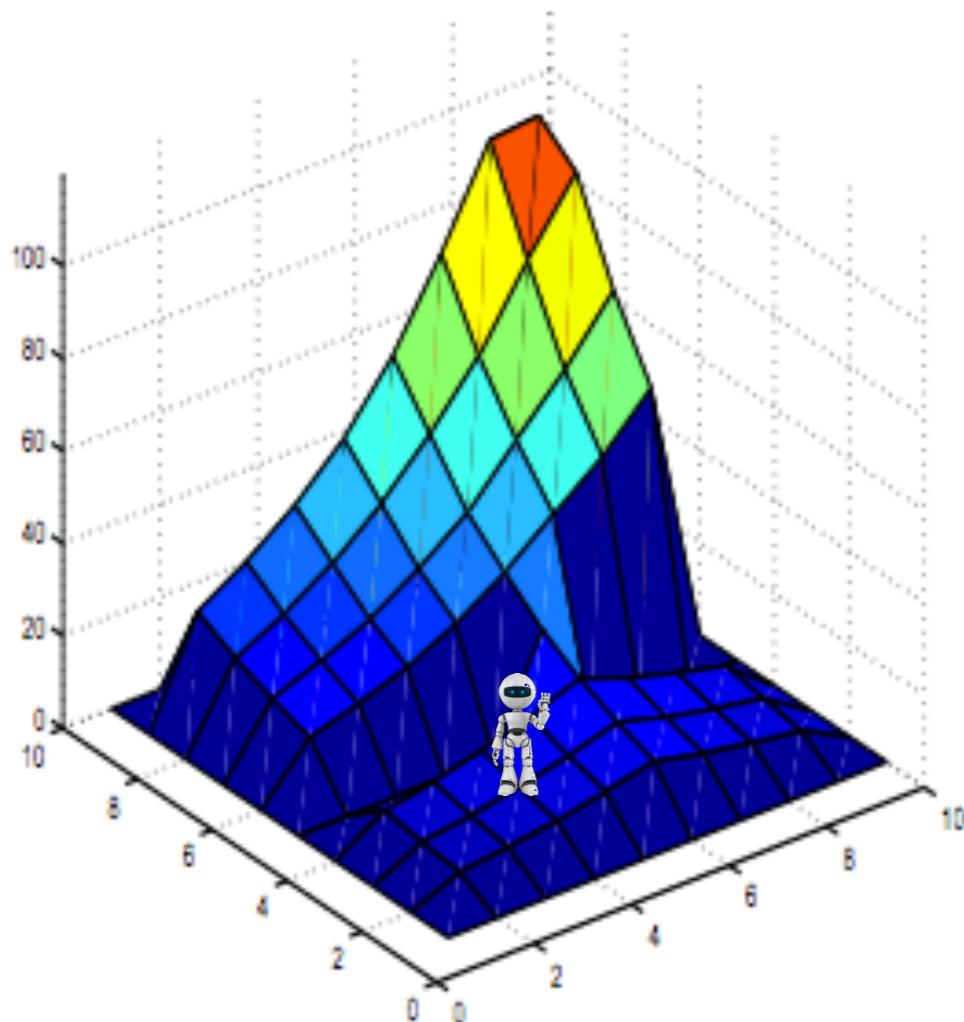
Busca Local



Estado **s**:

vetor com coordenadas [x,y] do agente

Busca Local

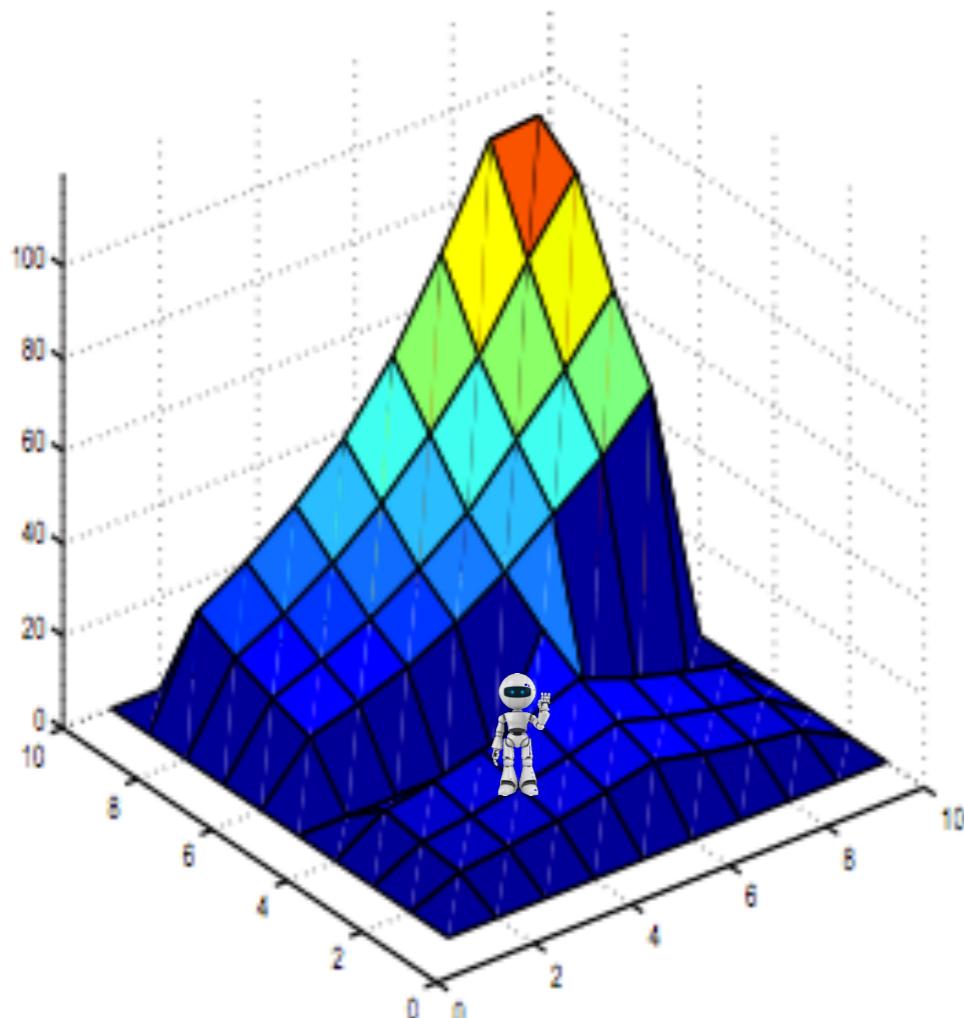


Estado **s**:

vetor com coordenadas [x,y] do agente

Vizinhos **s**: digamos que **s**=[10,14]

Busca Local



Estado s :

vetor com coordenadas [x,y] do agente

Vizinhos s : digamos que $s=[10,14]$

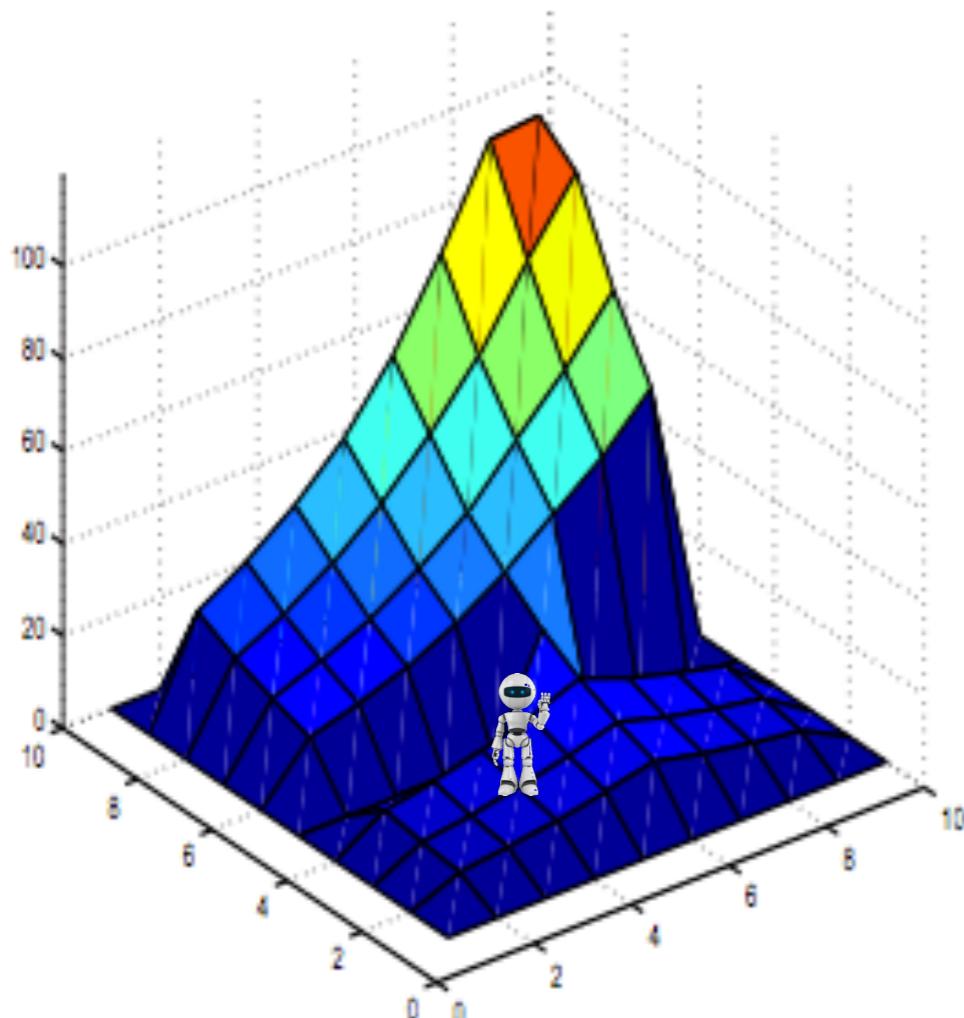
vizinhos = [11, 14]

[09, 14]

[11, 13]

[11, 15]

Busca Local



Estado s :

vetor com coordenadas $[x,y]$ do agente

Vizinhos s : digamos que $s=[10,14]$

vizinhos = [11, 14]

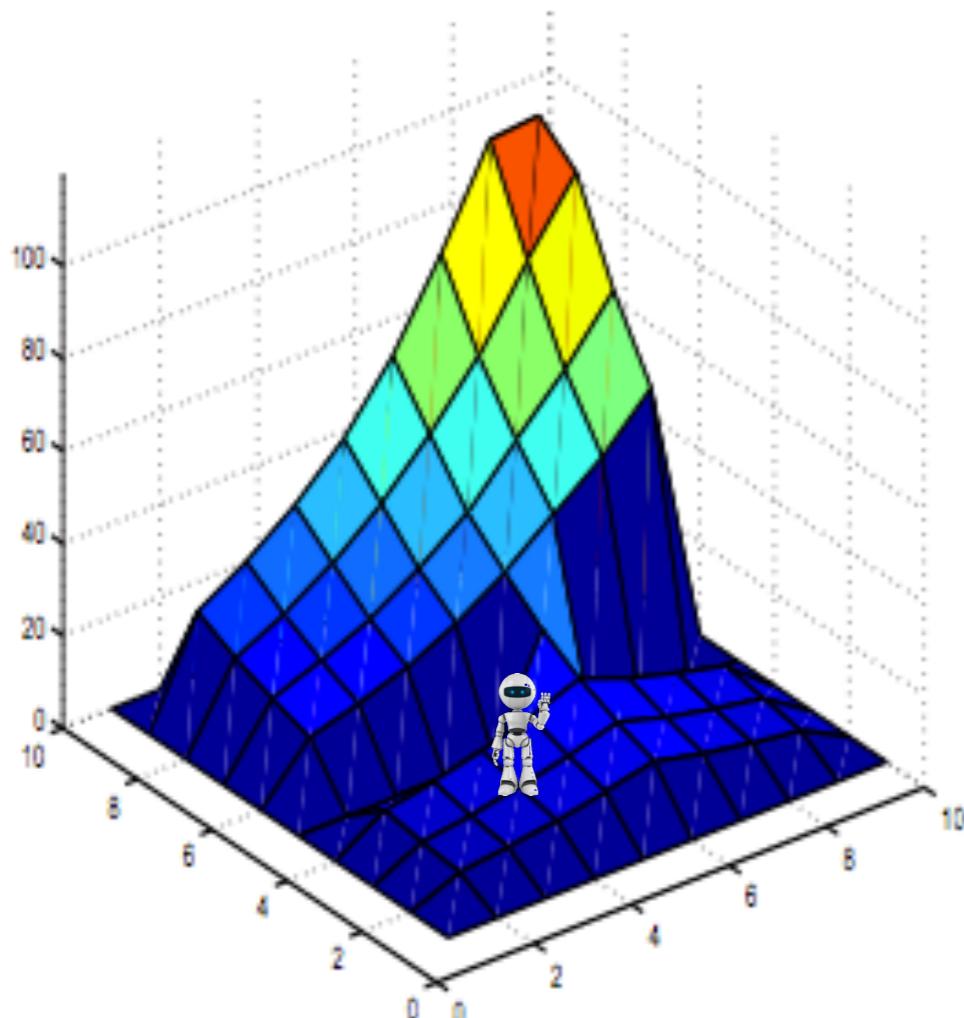
[09, 14]

[11, 13]

[11, 15]

Valor(s)

Busca Local



Estado s :

vetor com coordenadas $[x,y]$ do agente

Vizinhos s : digamos que $s=[10,14]$

vizinhos = [11, 14]

[09, 14]

[11, 13]

[11, 15]

Valor(s)

temperatura na posição $s=[x,y]$ atual

Função de Valor

Otimização de chutes em competição de futebol de robôs

- Solução candidata s : como representar um possível chute/solução através de um conjunto de números?
- Vizinhos de s : como alterar aqueles números para gerar possíveis chutes/soluções vizinhas?
- Valor(s): como quantificar o quanto bom é um possível chute s ?



<https://www.youtube.com/watch?v=nZguKrgLTBQ>

Função de Valor

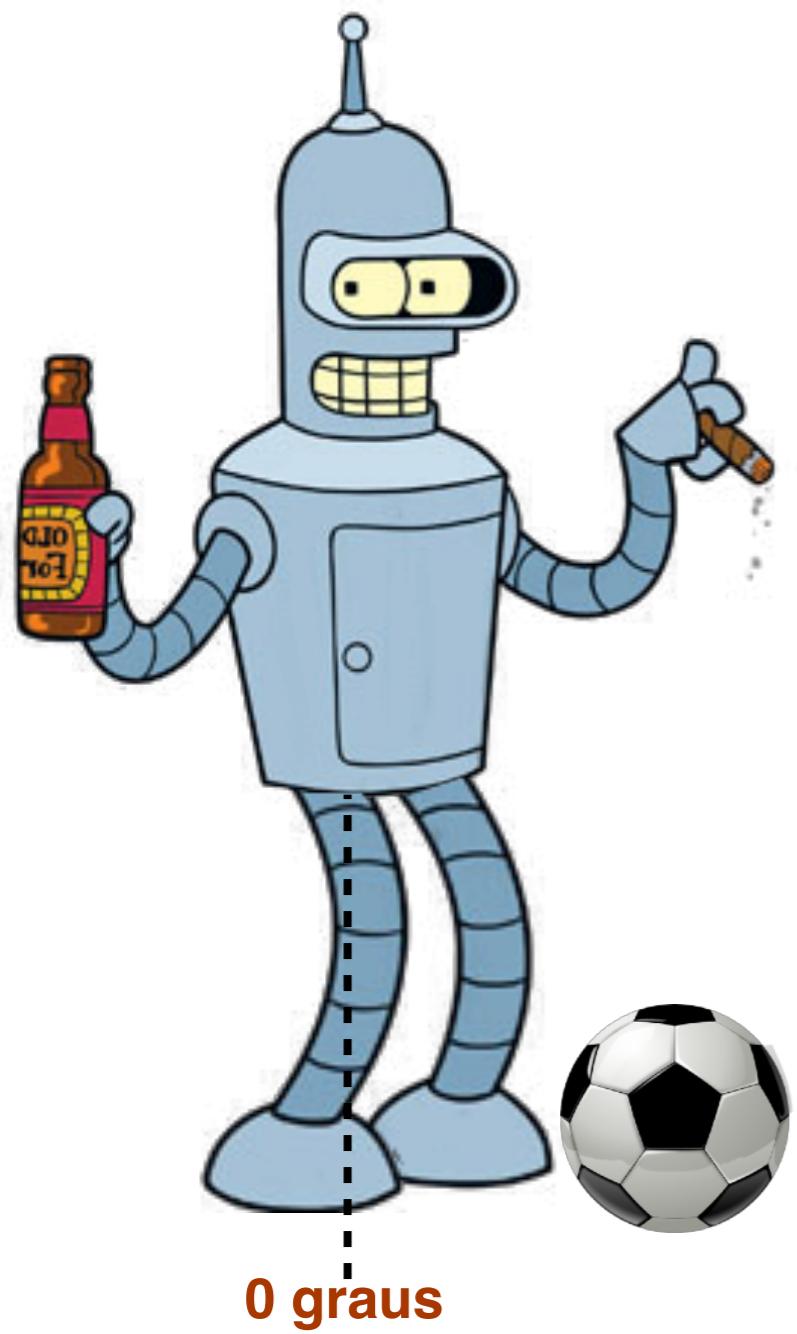
Otimização de chutes em competição de futebol de robôs

- Solução candidata s : seqüência de posições (ângulos das juntas) da perna do robô durante o chute
- Vizinhos de s : pequenas modificações na seqüência de posições da perna (i.e., uma variação do chute)
- Valor(s): distância que a bola percorre ao se executar o chute s
- Busca local pelo chute s que maximiza Valor(s) — objetivo é aprender a executar chutes poderosos



<https://www.youtube.com/watch?v=nZguKrgLTBQ>

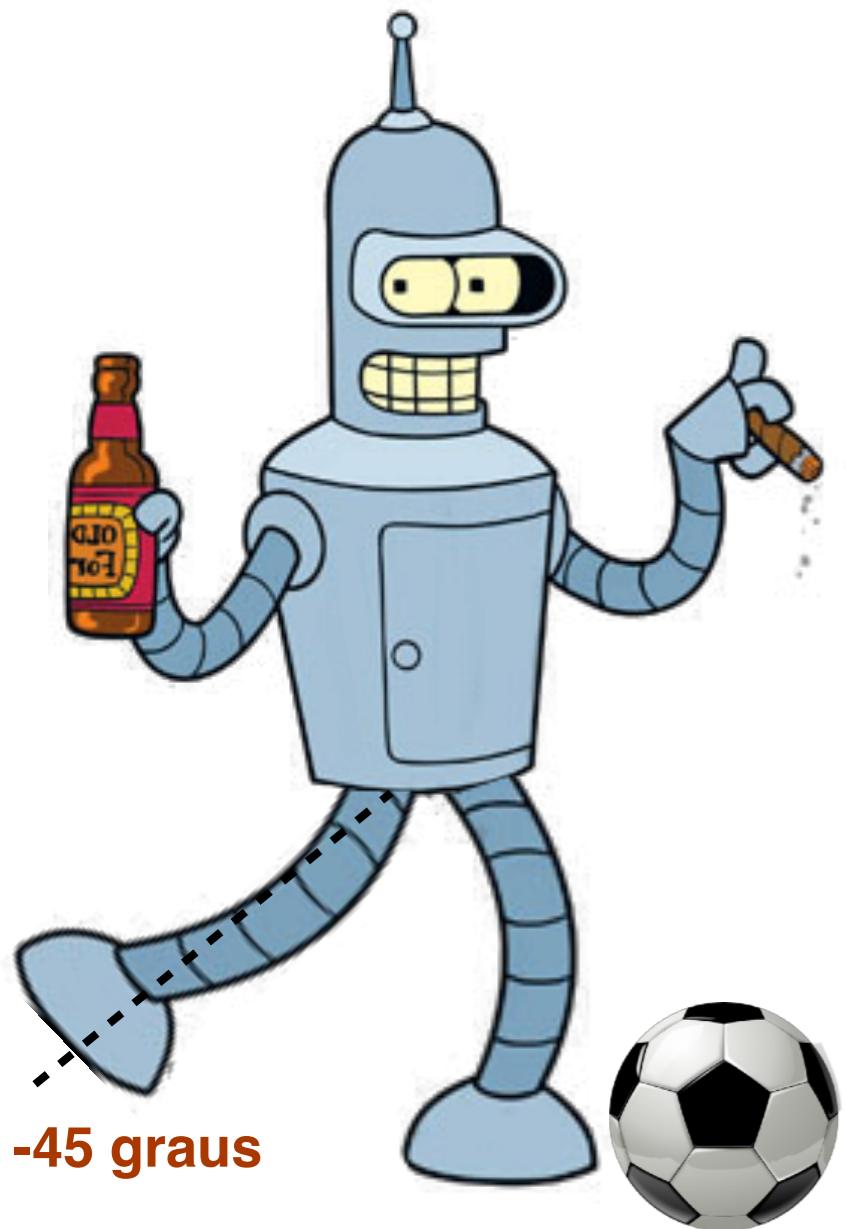
Tentativa 1: $s_1=[0, -45, +30]$



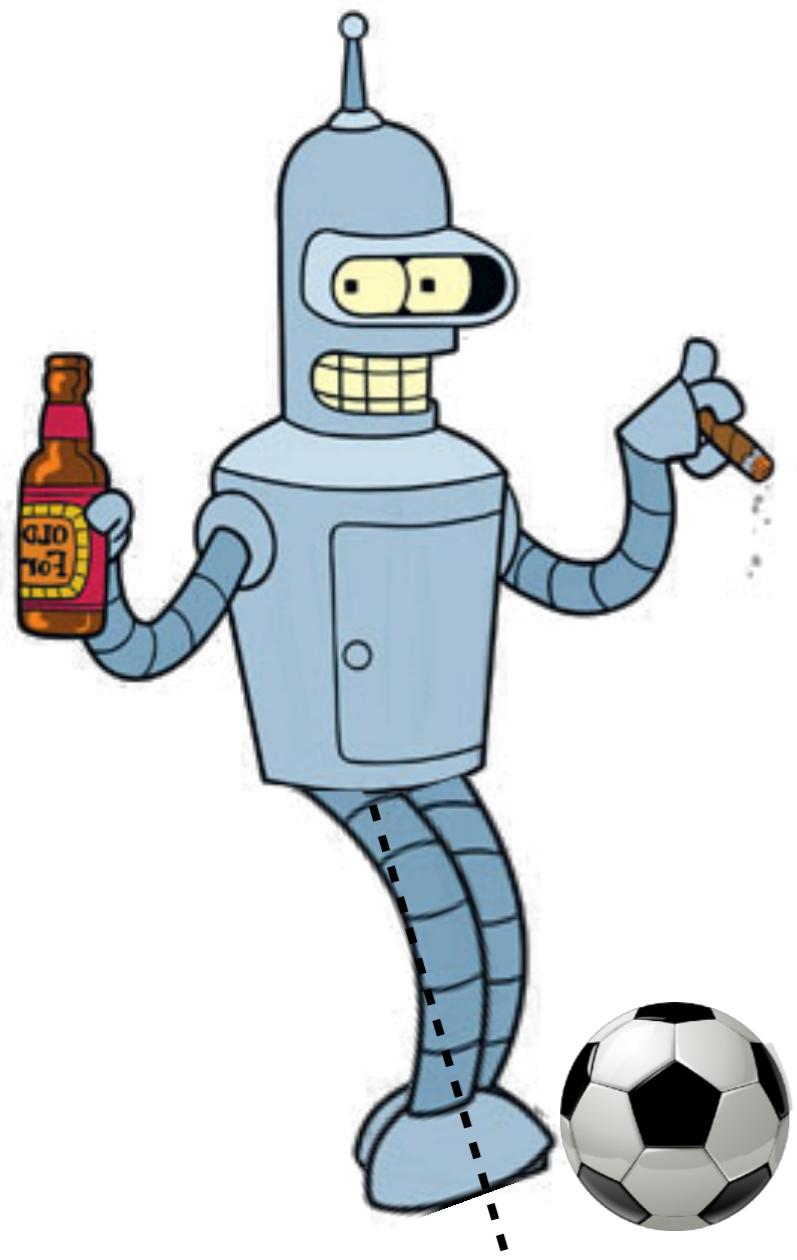
0 graus



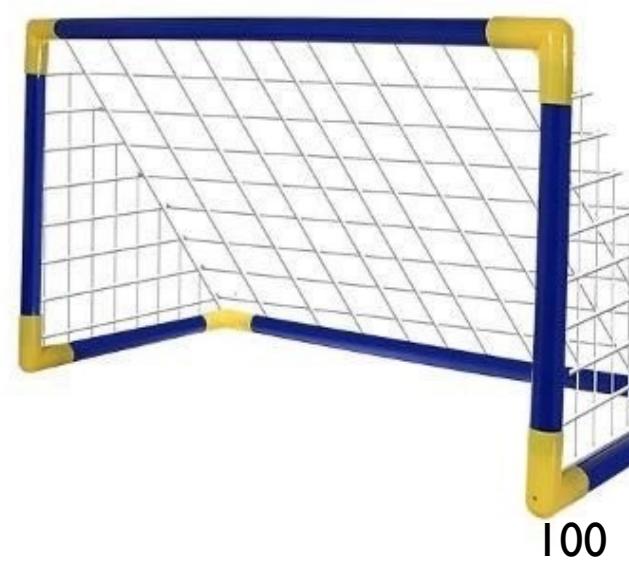
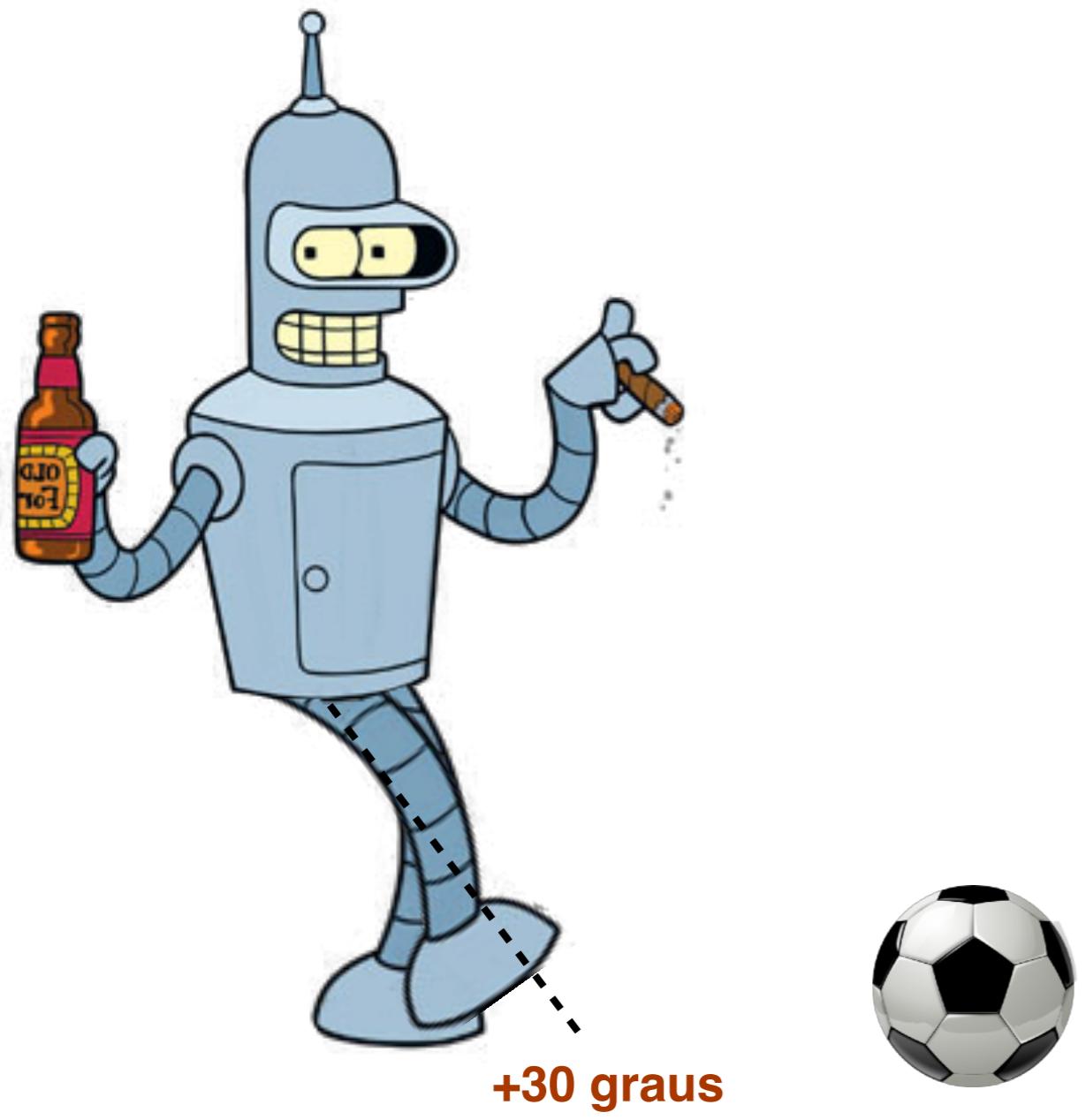
Tentativa 1: $s_1=[0, -45, +30]$



Tentativa 1: $s_1=[0, -45, +30]$

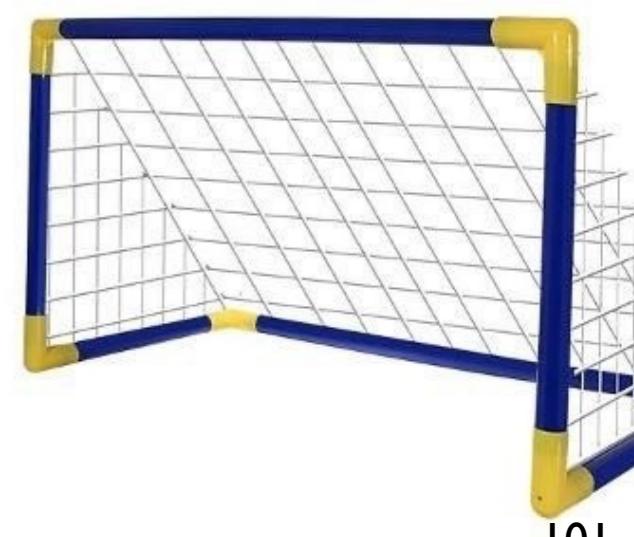
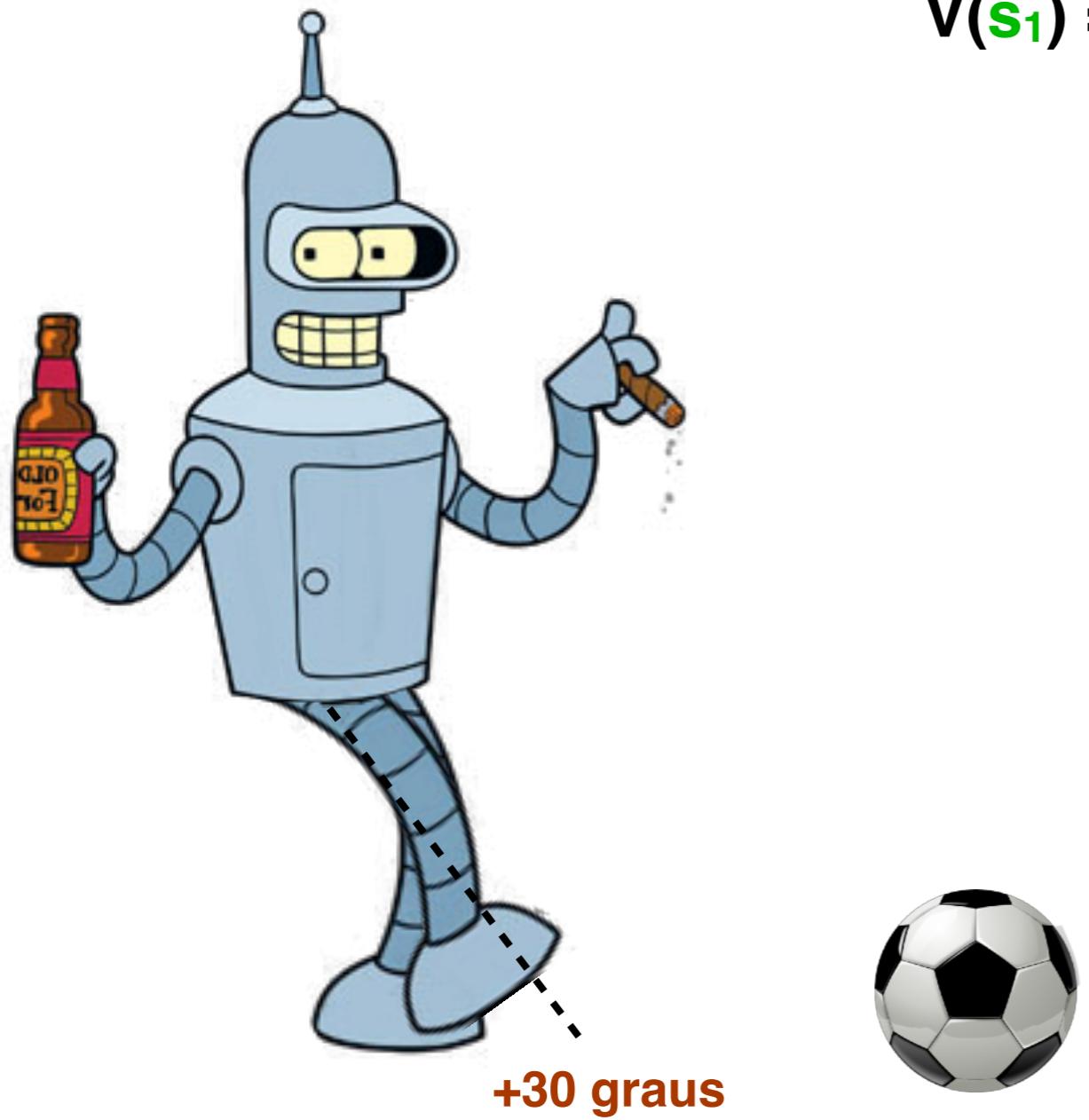


Tentativa 1: $s_1=[0, -45, +30]$



Tentativa 1: $s_1=[0, -45, +30]$

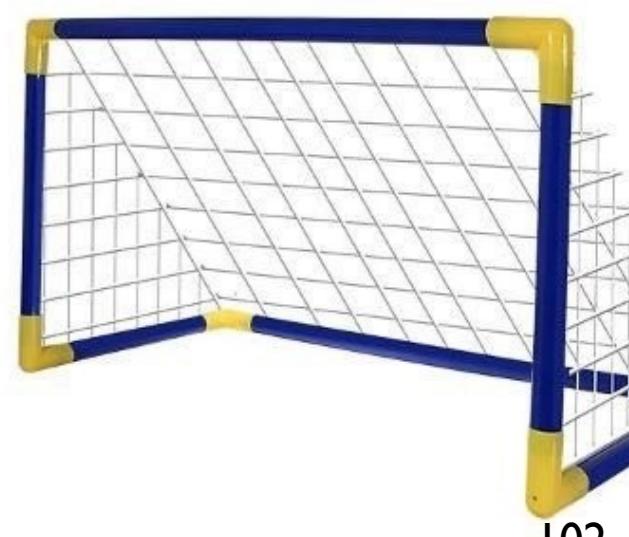
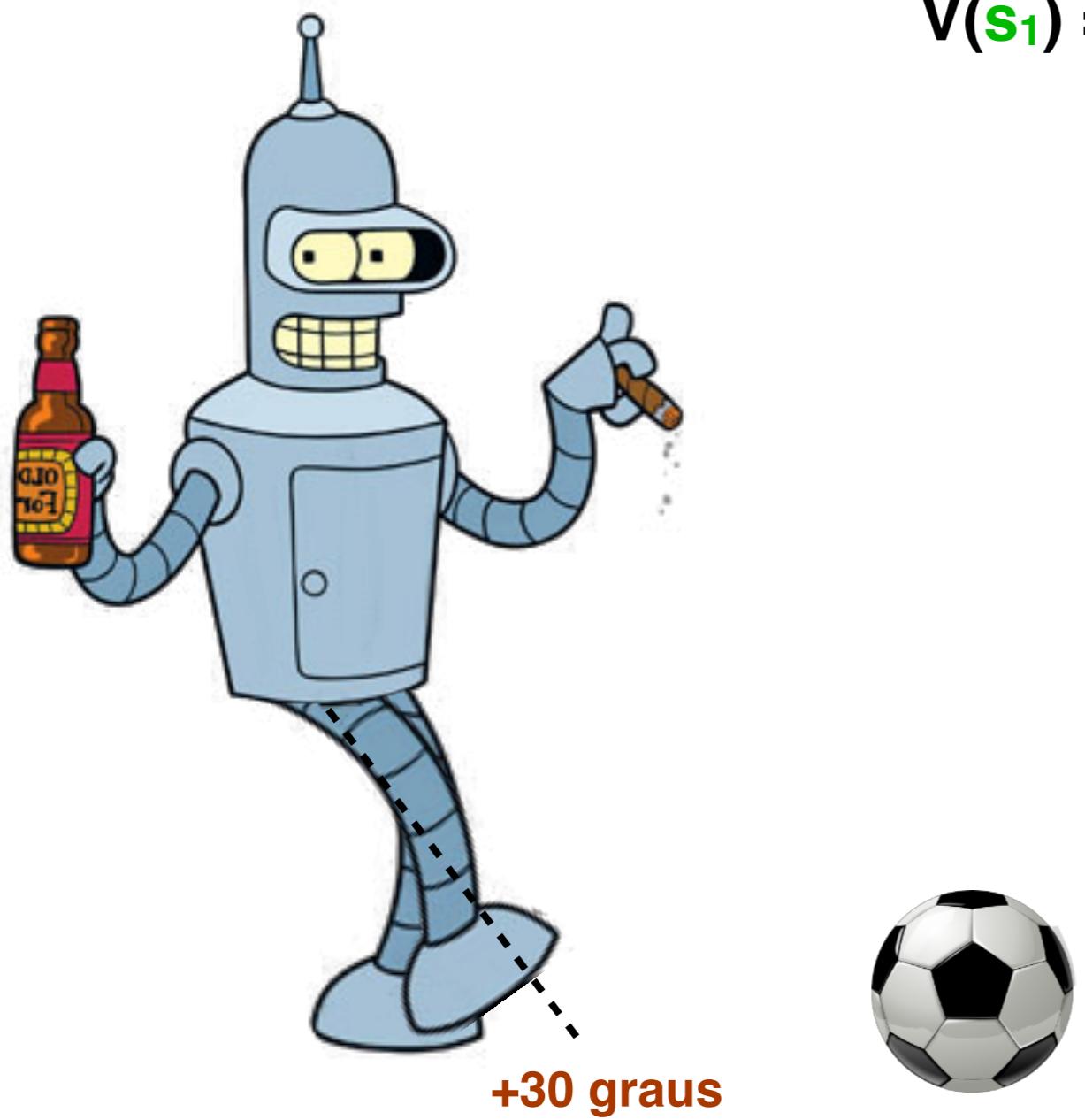
$$V(s_1) = 5 \text{ metros}$$



$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

Tentativa 1: $s_1=[0, -45, +30]$

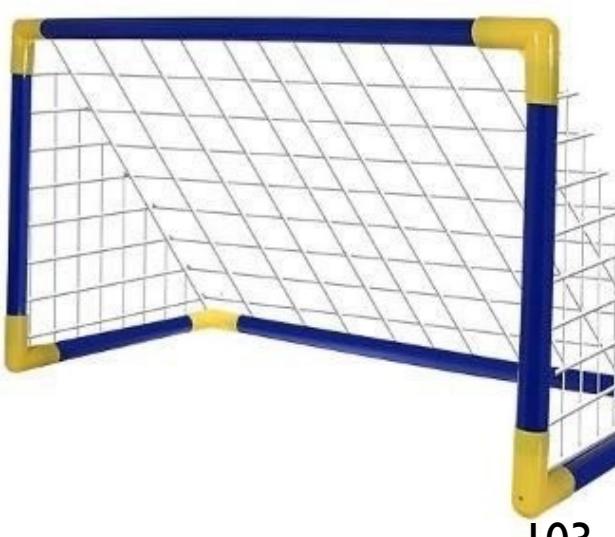
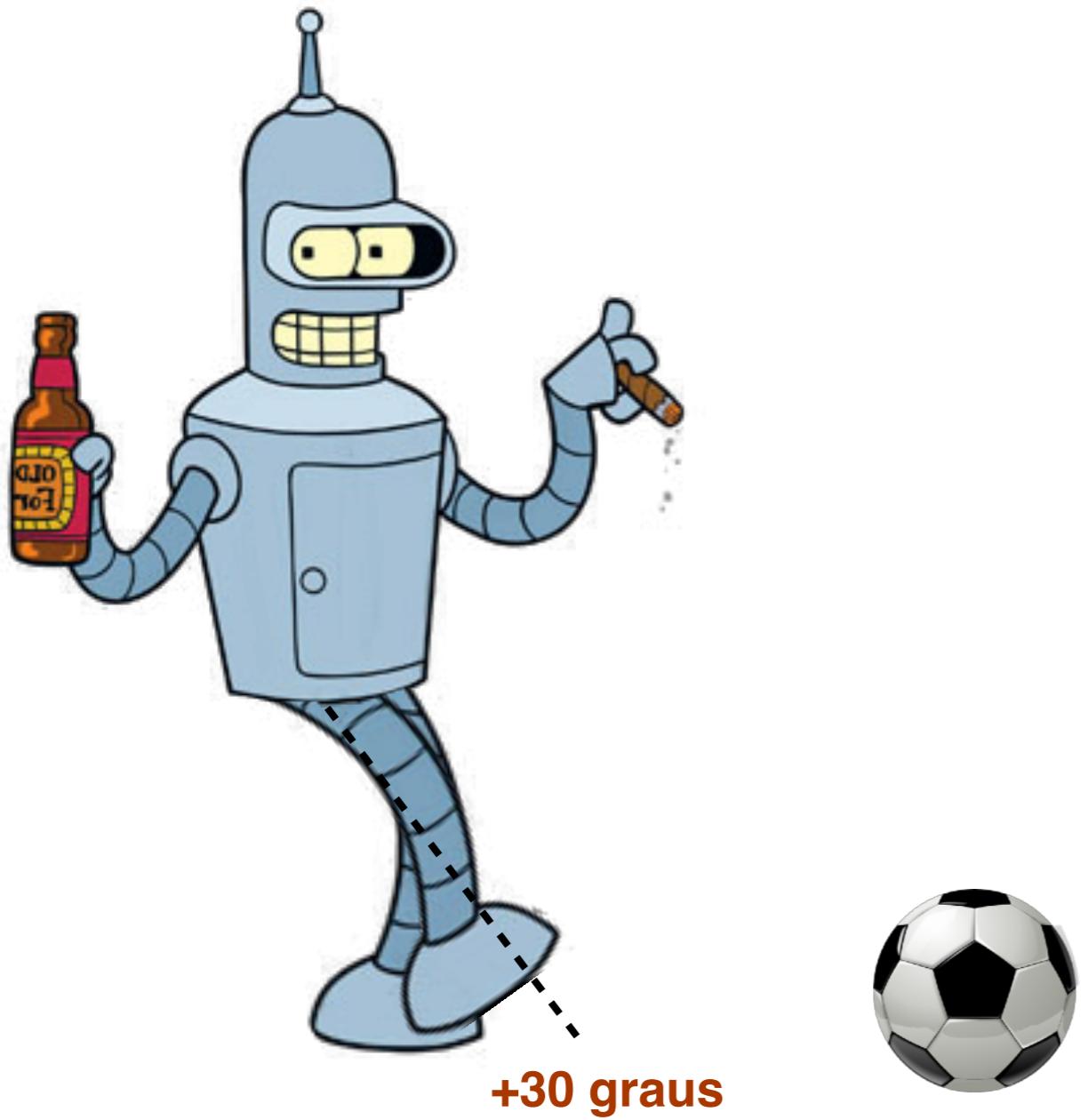
$V(s_1) = 5$ metros



$s_1 = [0, -45, +30]$
 $V(s_1) = 5 \text{ metros}$

$s_1 = [0, -45, +30]$
 $V(s_1) = 5 \text{ metros}$

Estados
vizinhos?

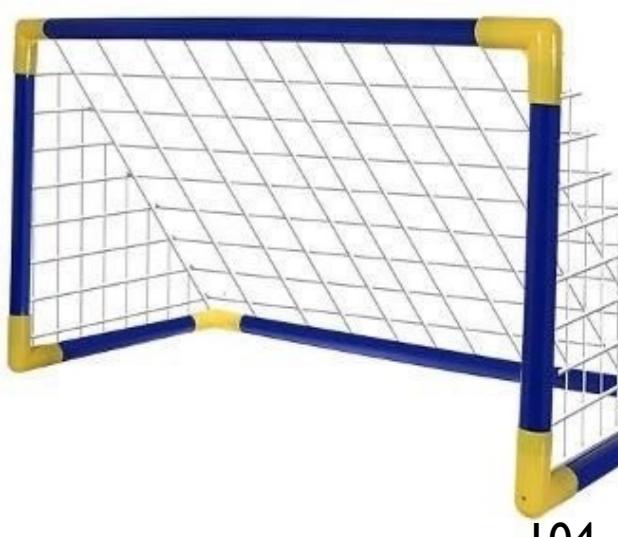


$s_1=[0, -45, +30]$
 $V(s_1) = 5 \text{ metros}$

$s_1=[0, -45, +30]$
 $V(s_1) = 5 \text{ metros}$

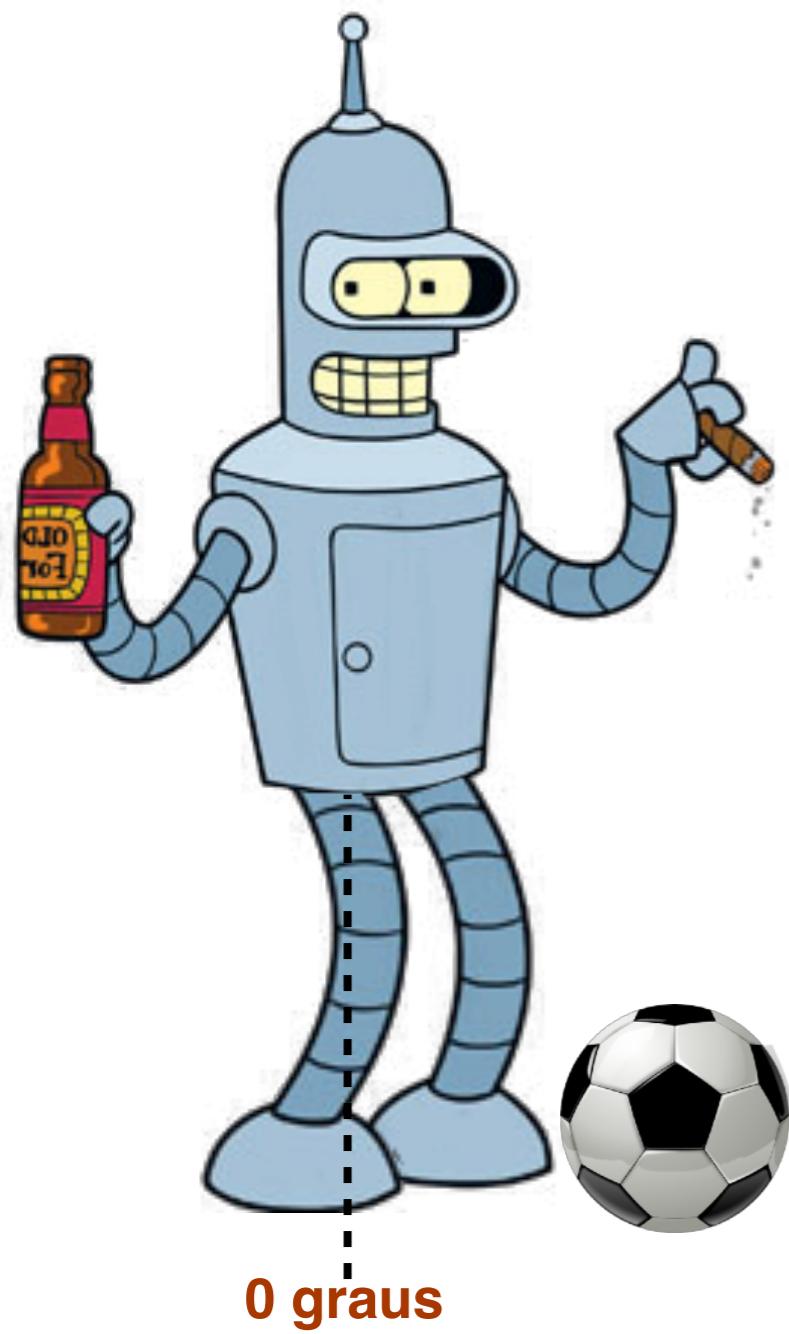
Estados vizinhos

$s_2=[0, -20, +20]$
 $s_3=[0, -45, +60]$



$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

Tentativa 2: $s_2=[0, -20, +20]$



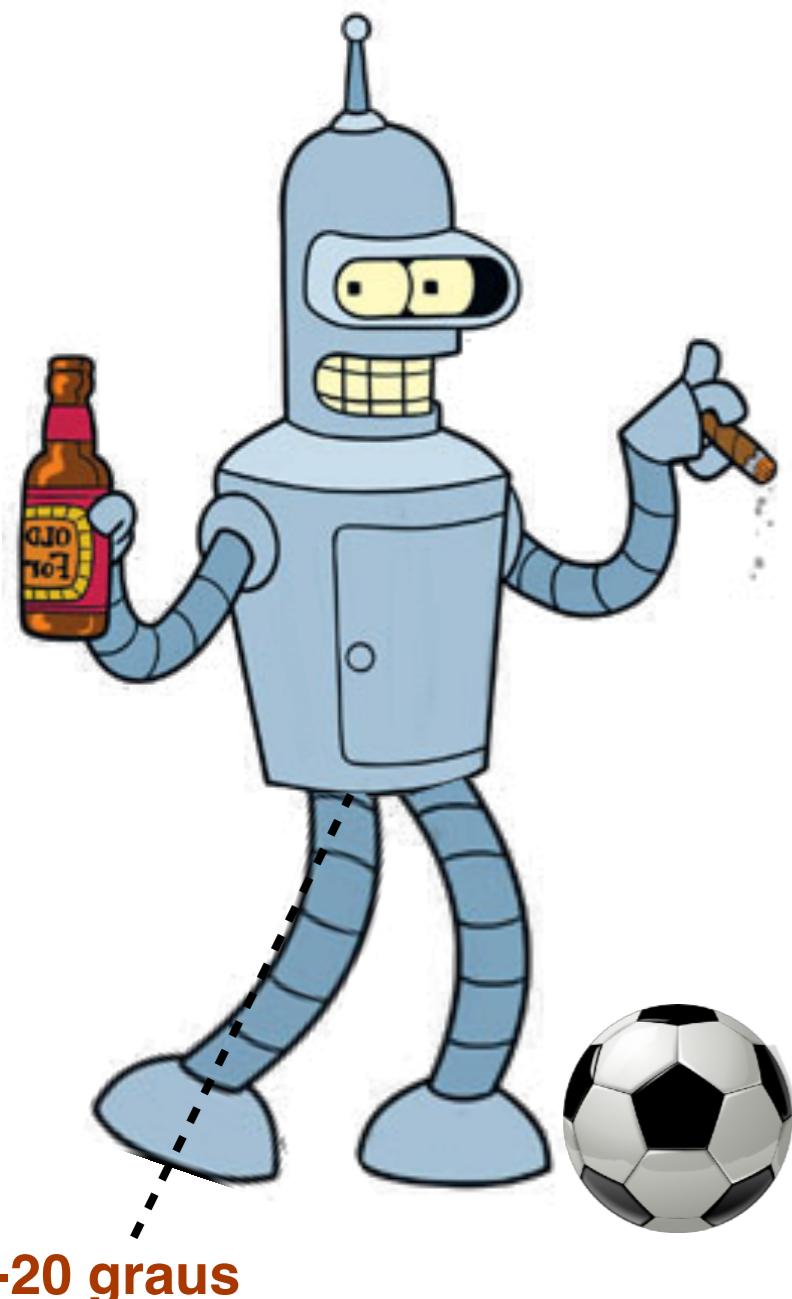
0 graus



105

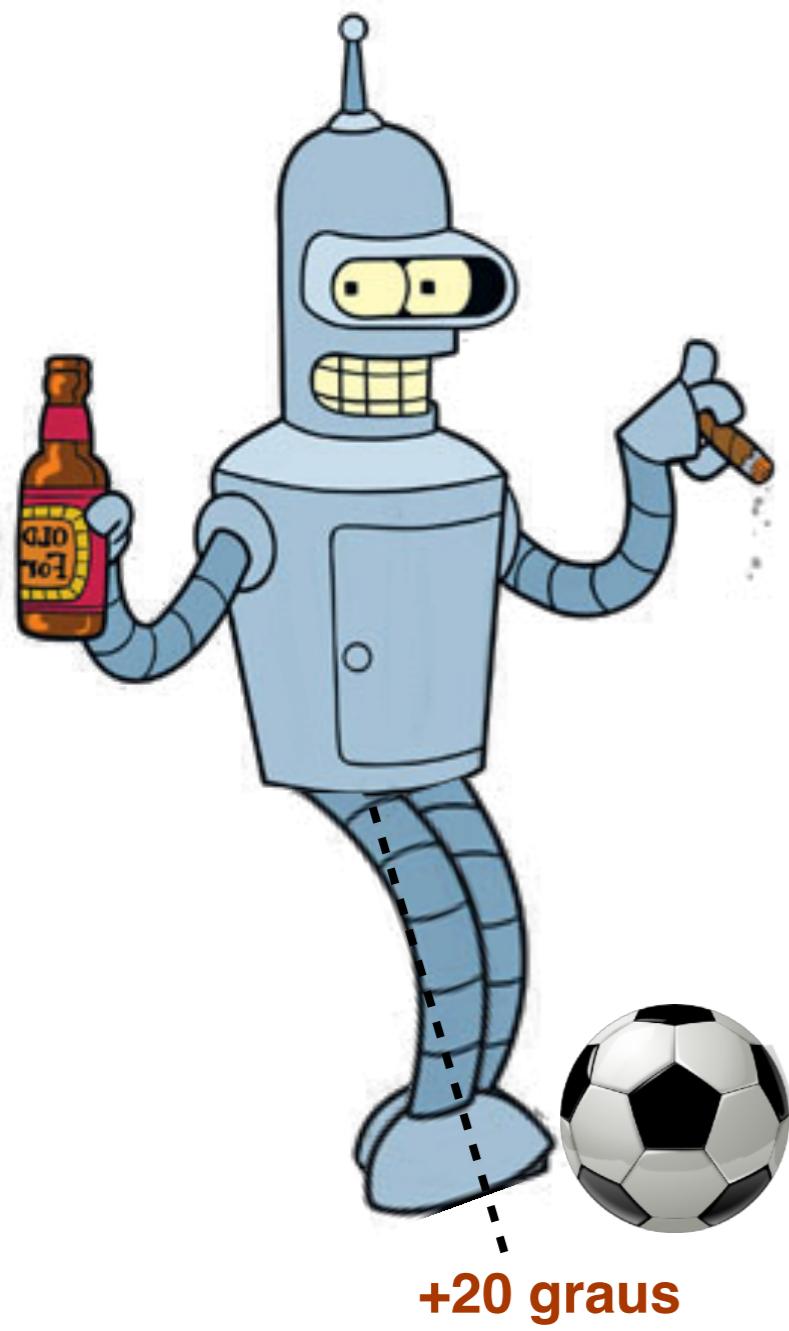
$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

Tentativa 2: $s_2=[0, -20, +20]$



$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

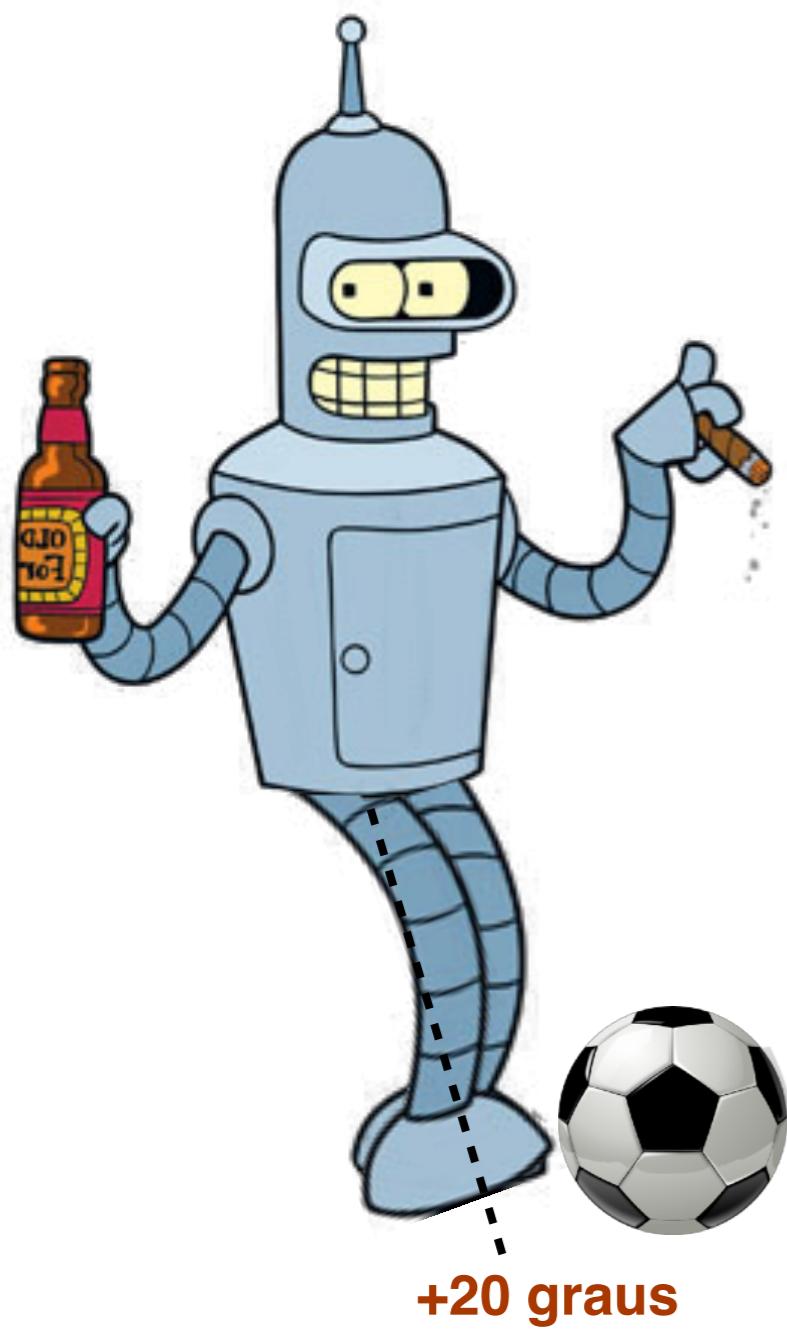
Tentativa 2: $s_2=[0, -20, +20]$



$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

Tentativa 2: $s_2=[0, -20, +20]$

$V(s_2) = 0$ metros



$s_1=[0, -45, +30]$

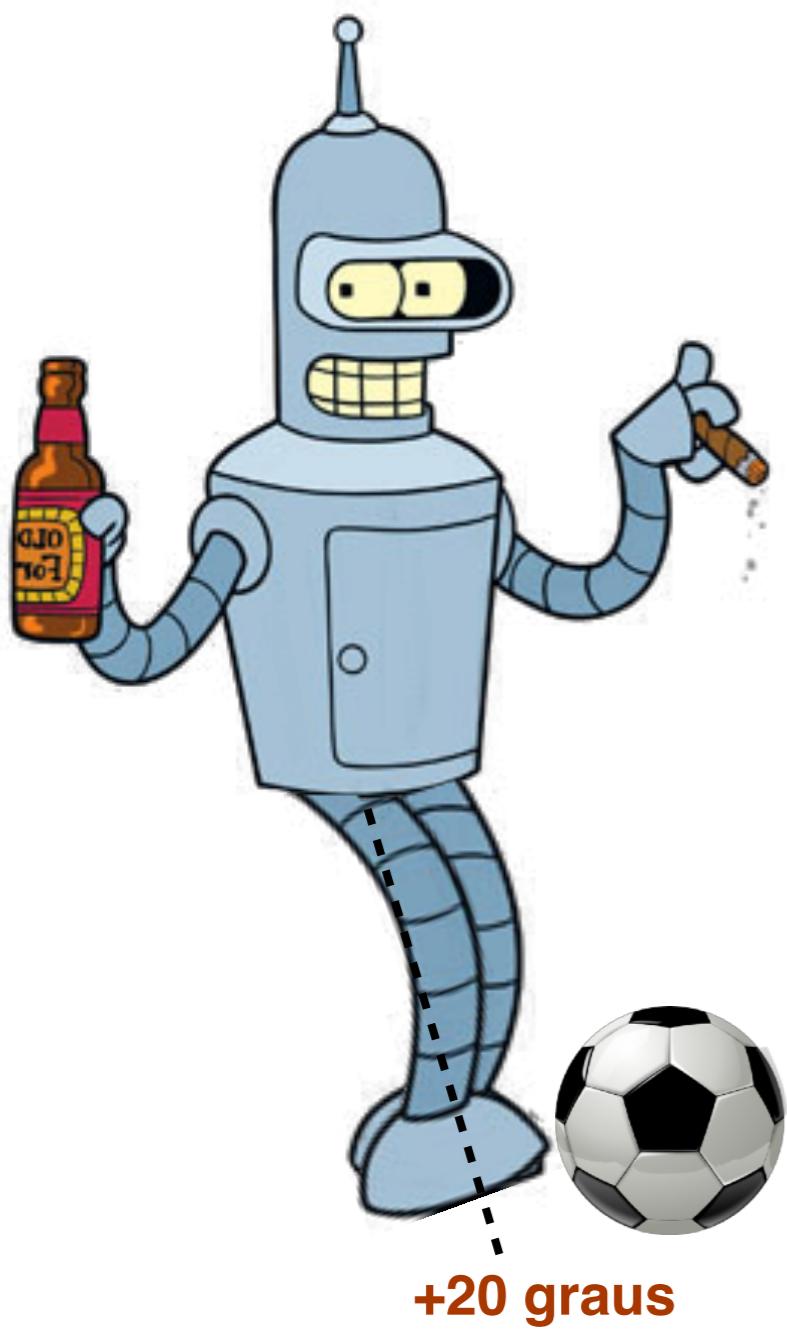
$V(s_1) = 5$ metros

$s_2=[0, -20, +20]$

$V(s_2) = 0$ metros

Tentativa 2: $s_2=[0, -20, +20]$

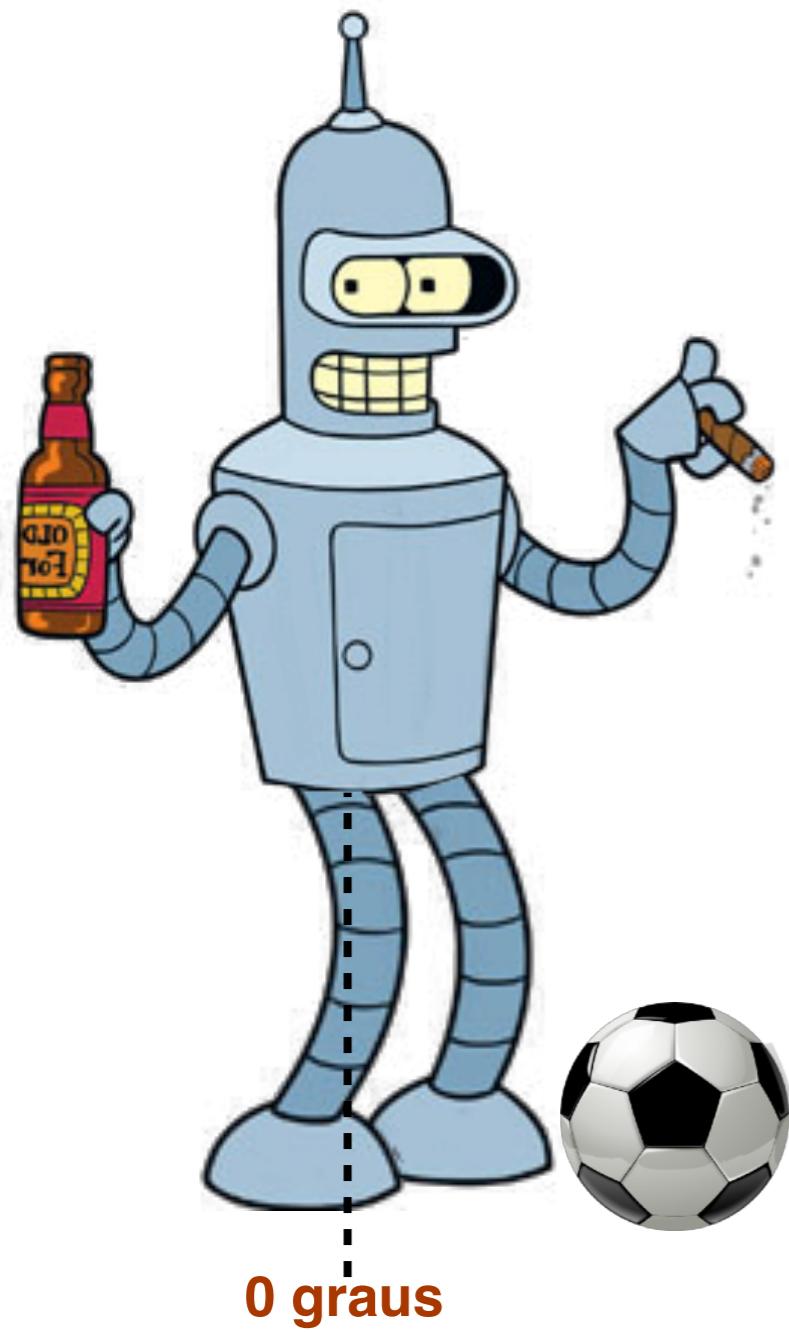
$V(s_2) = 0$ metros



$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

$s_2=[0, -20, +20]$
 $V(s_2) = 0$ metros

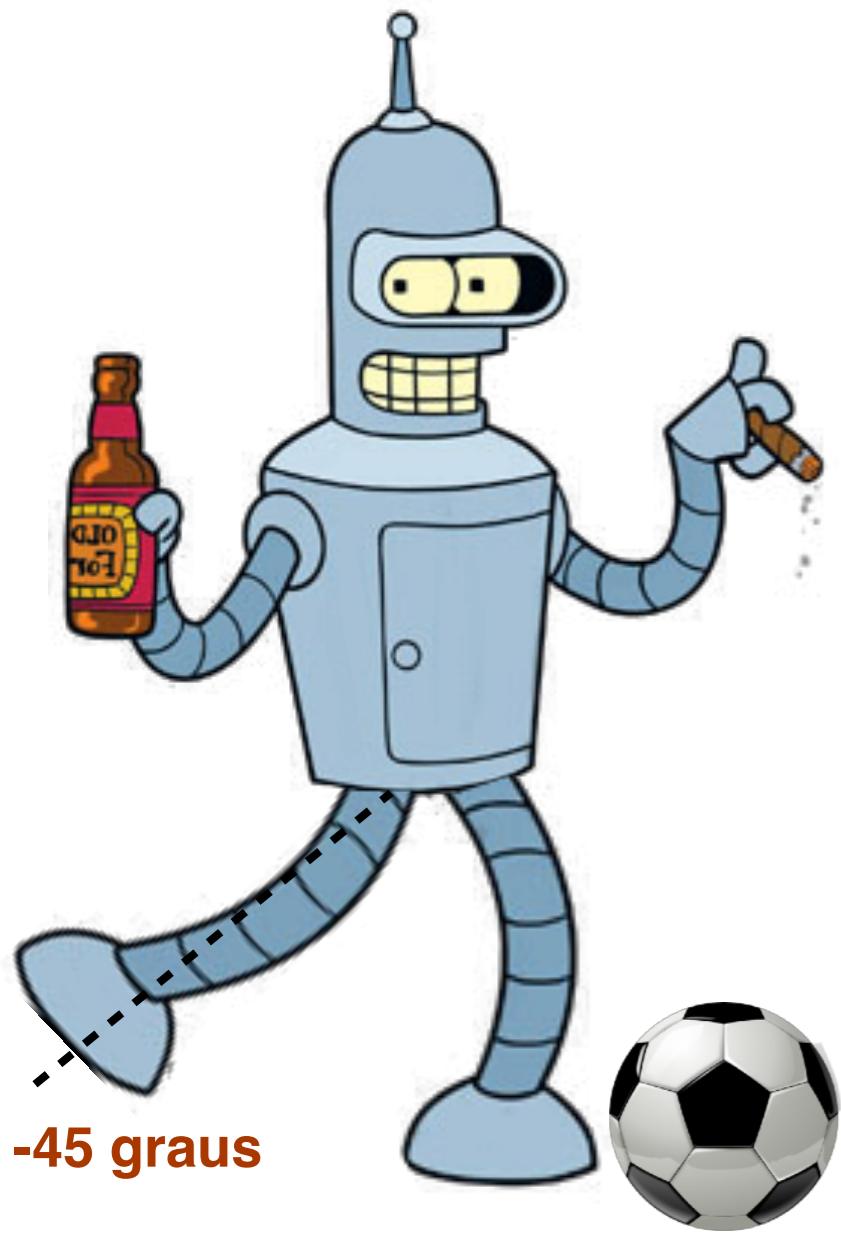
Tentativa 3: $s_3=[0, -45, +60]$



$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

$s_2=[0, -20, +20]$
 $V(s_2) = 0$ metros

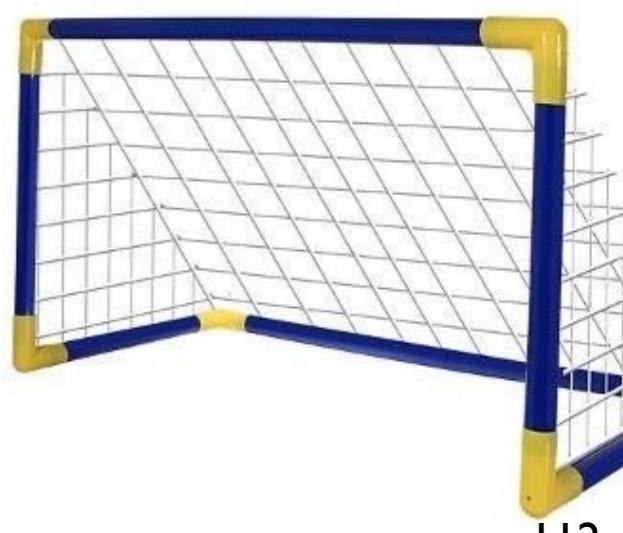
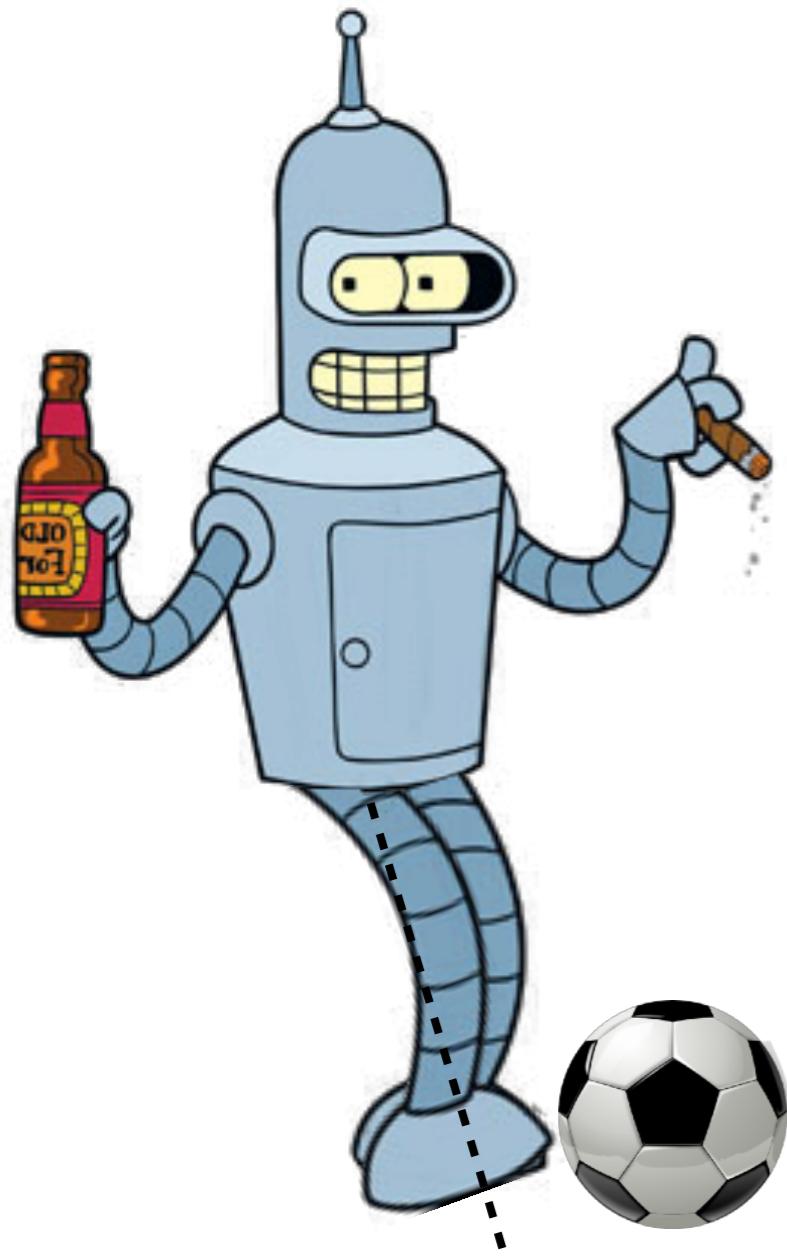
Tentativa 3: $s_3=[0, -45, +60]$



$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

$s_2=[0, -20, +20]$
 $V(s_2) = 0$ metros

Tentativa 3: $s_3=[0, -45, +60]$



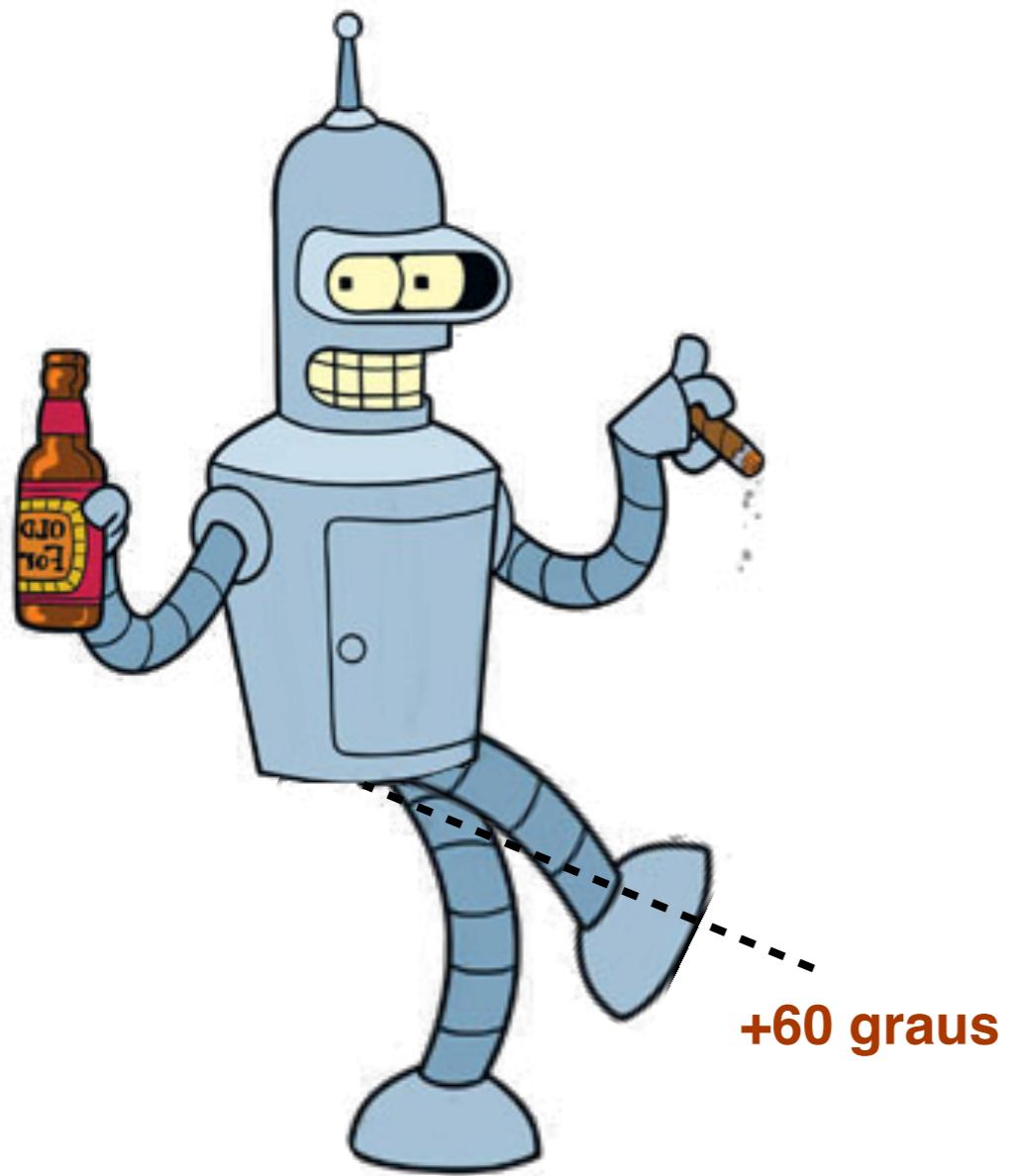
$s_1=[0, -45, +30]$

$V(s_1) = 5$ metros

$s_2=[0, -20, +20]$

$V(s_2) = 0$ metros

Tentativa 3: $s_3=[0, -45, +60]$

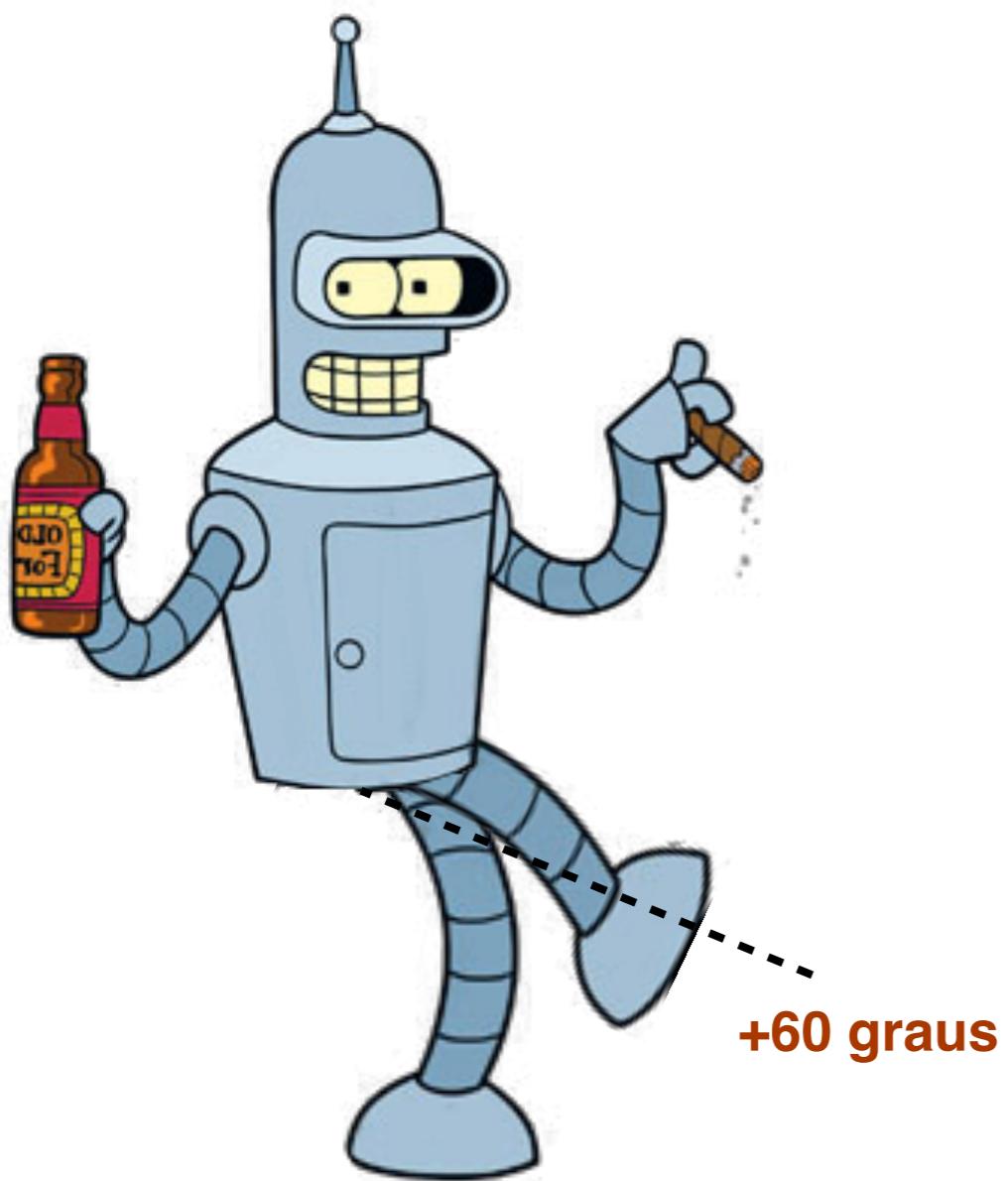


$s_1=[0, -45, +30]$
 $V(s_1) = 5$ metros

$s_2=[0, -20, +20]$
 $V(s_2) = 0$ metros

Tentativa 3: $s_3=[0, -45, +60]$

$V(s_3) = 12$ metros



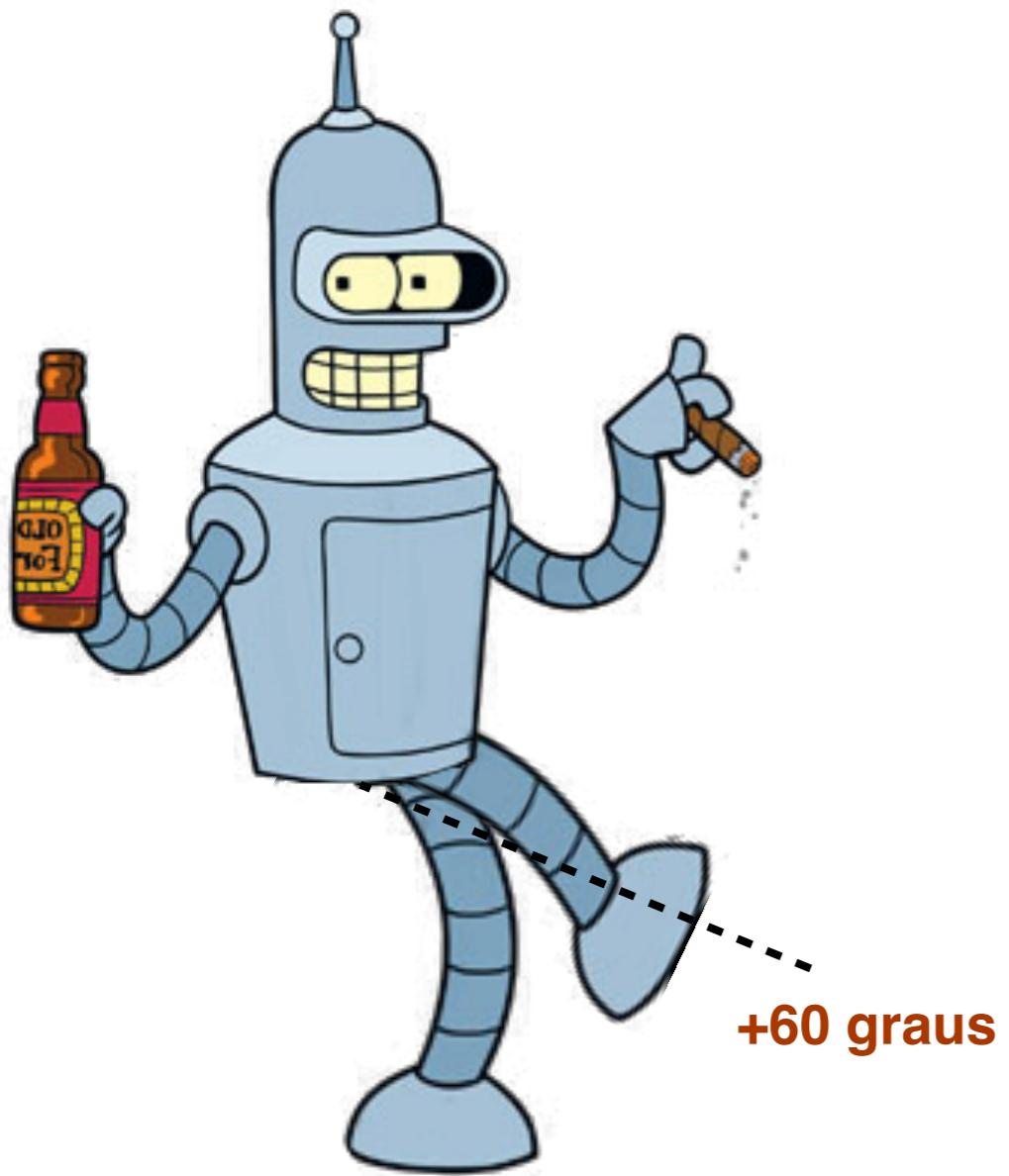
$s_1=[0, -45, +30]$
 $V(s_1) = 5 \text{ metros}$

$s_2=[0, -20, +20]$
 $V(s_2) = 0 \text{ metros}$

Tentativa 3: $s_3=[0, -45, +60]$

$s_3=[0, -45, +60]$
 $V(s_3) = 12 \text{ metros}$

$V(s_3) = 12 \text{ metros}$

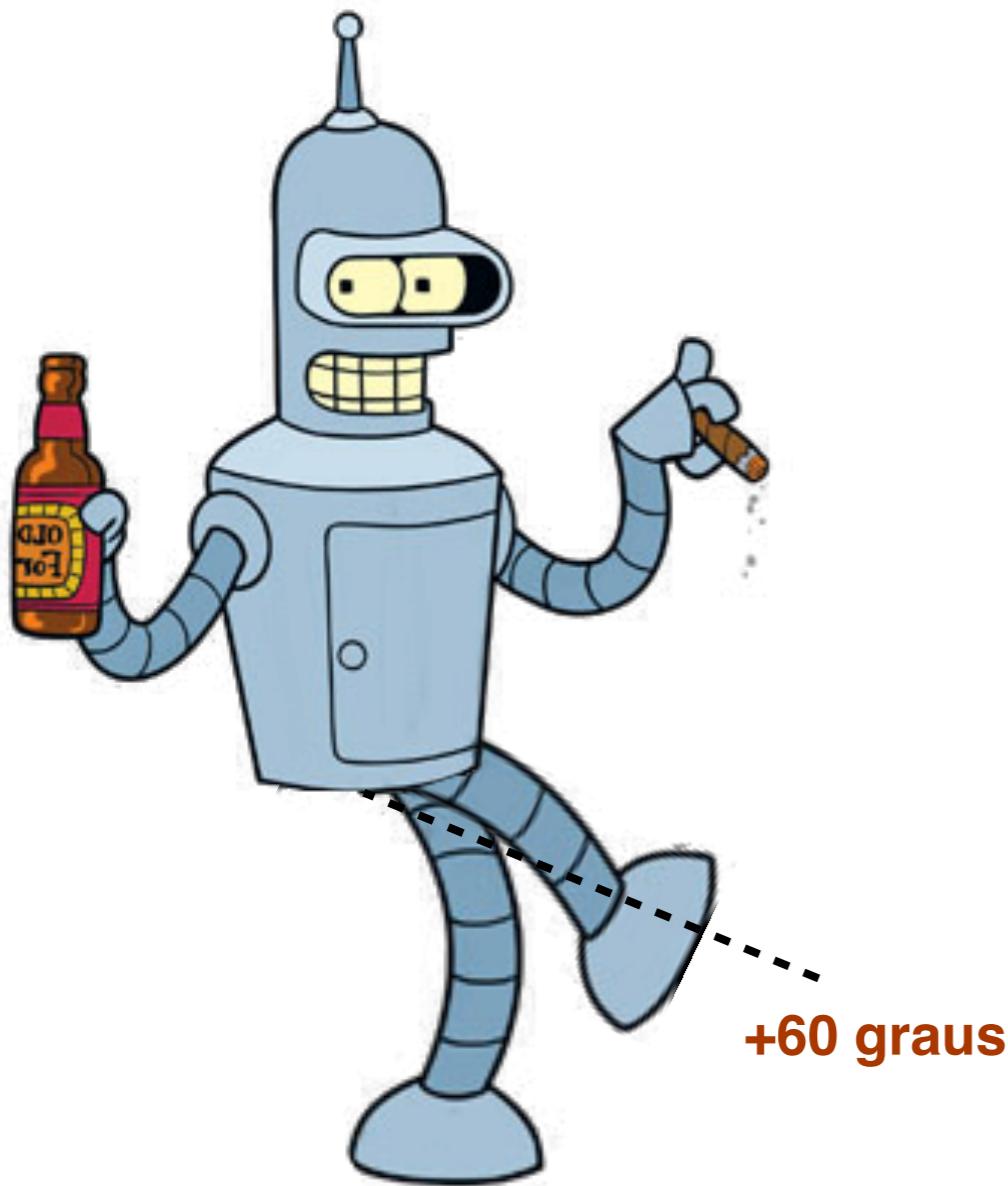


$s_1=[0, -45, +30]$ $V(s_1) = 5$ metros

$s_2=[0, -20, +20]$ $V(s_2) = 0$ metros

$s_3=[0, -45, +60]$ $V(s_3) = 12$ metros

Estados vizinhos

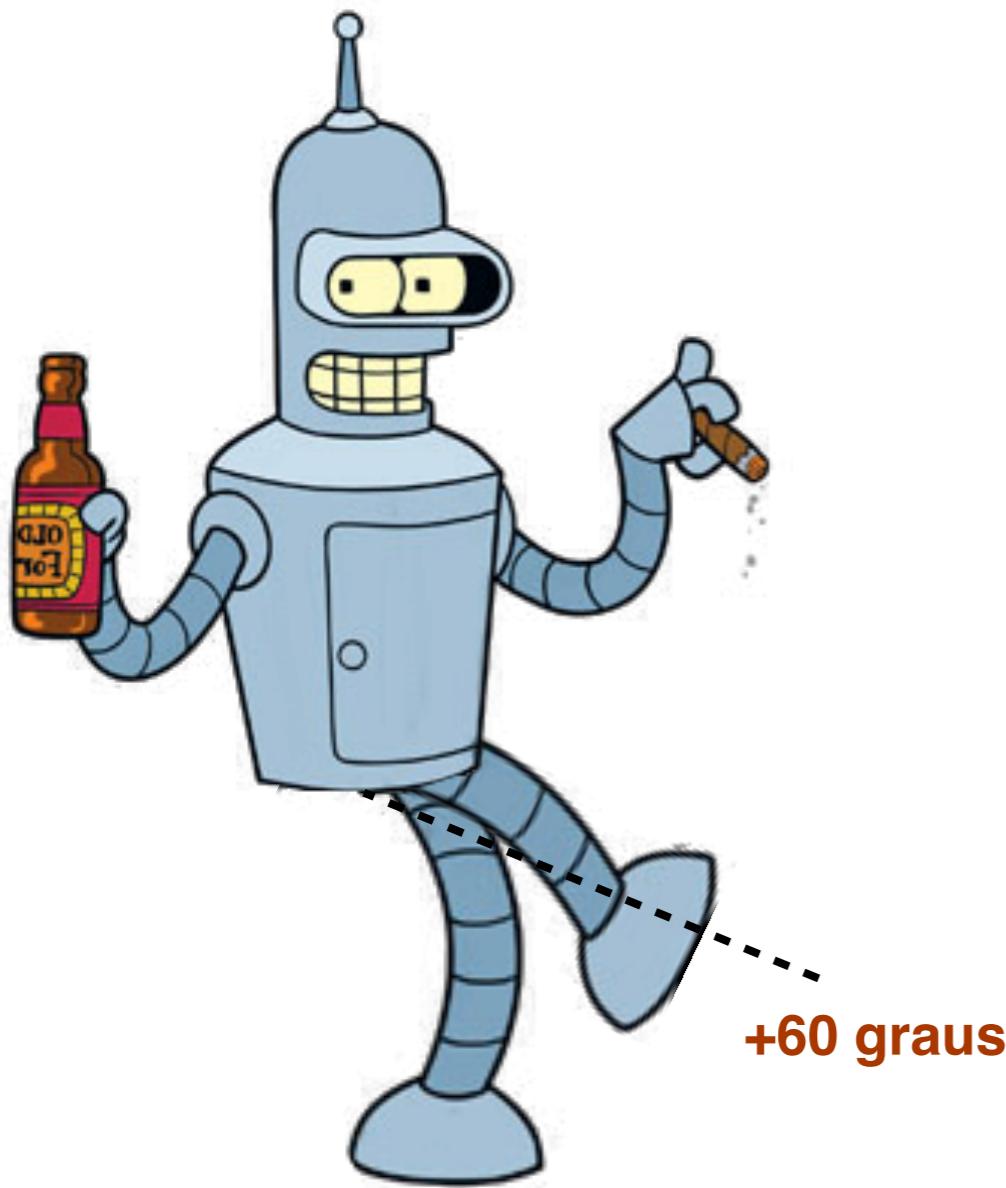


$s_1=[0, -45, +30]$ $V(s_1) = 5$ metros

$s_2=[0, -20, +20]$ $V(s_2) = 0$ metros

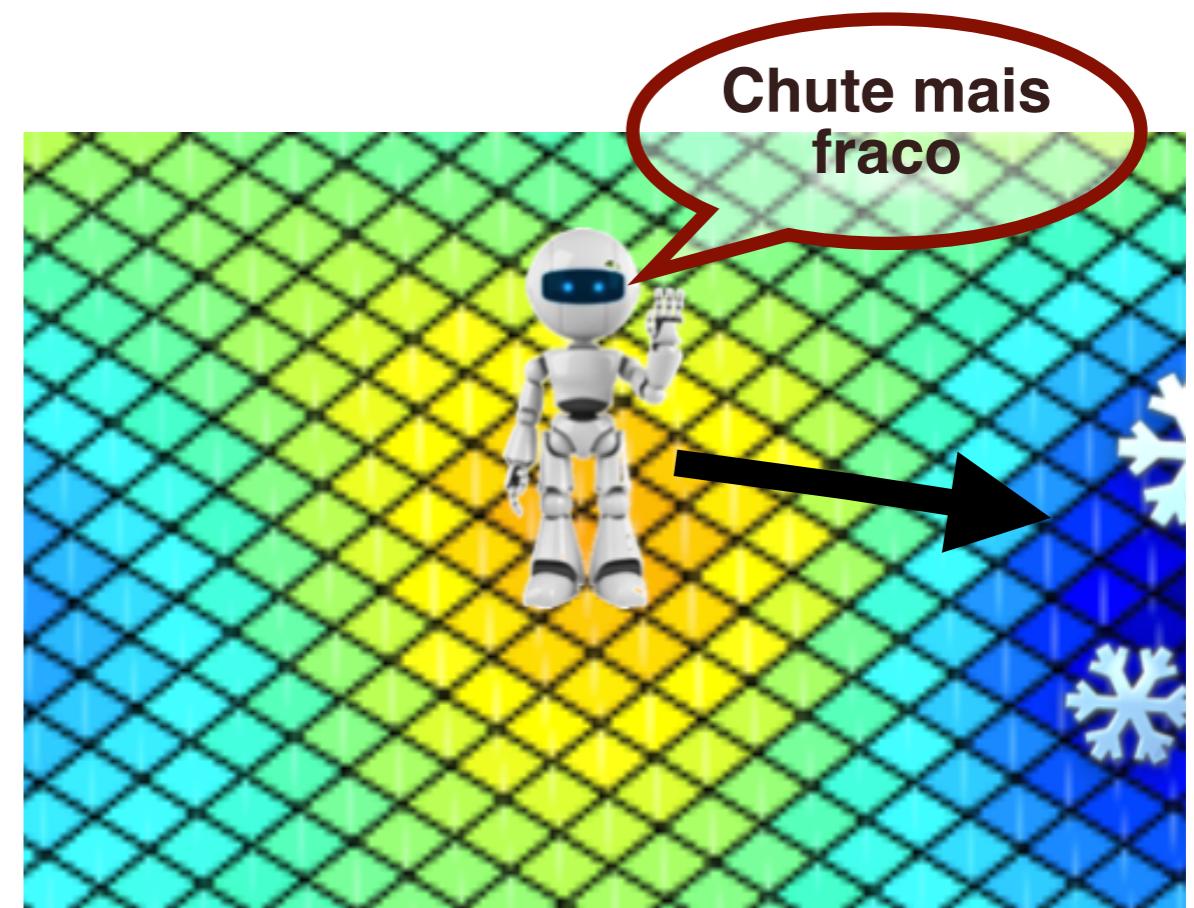
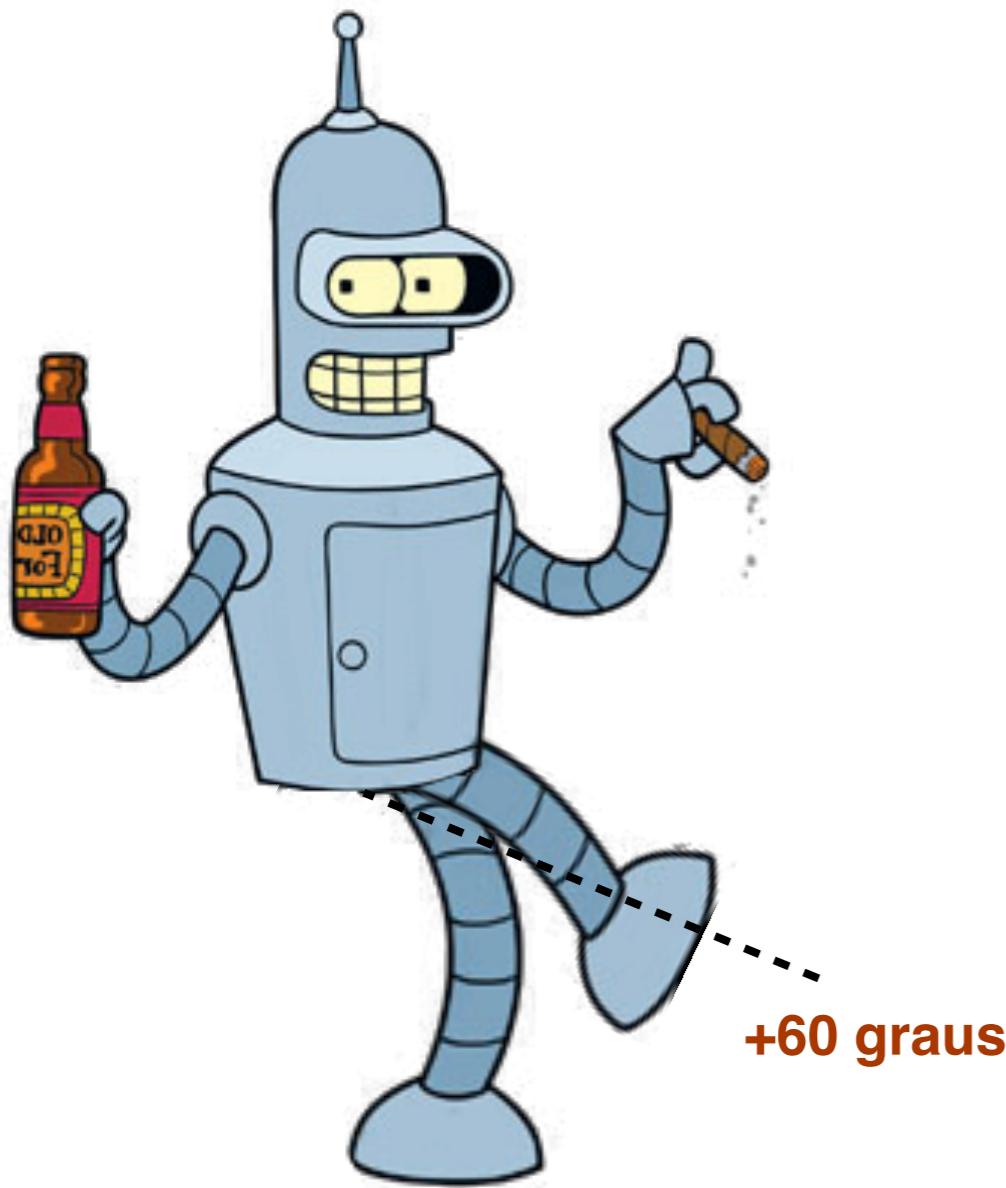
$s_3=[0, -45, +60]$ $V(s_3) = 12$ metros

Estados vizinhos



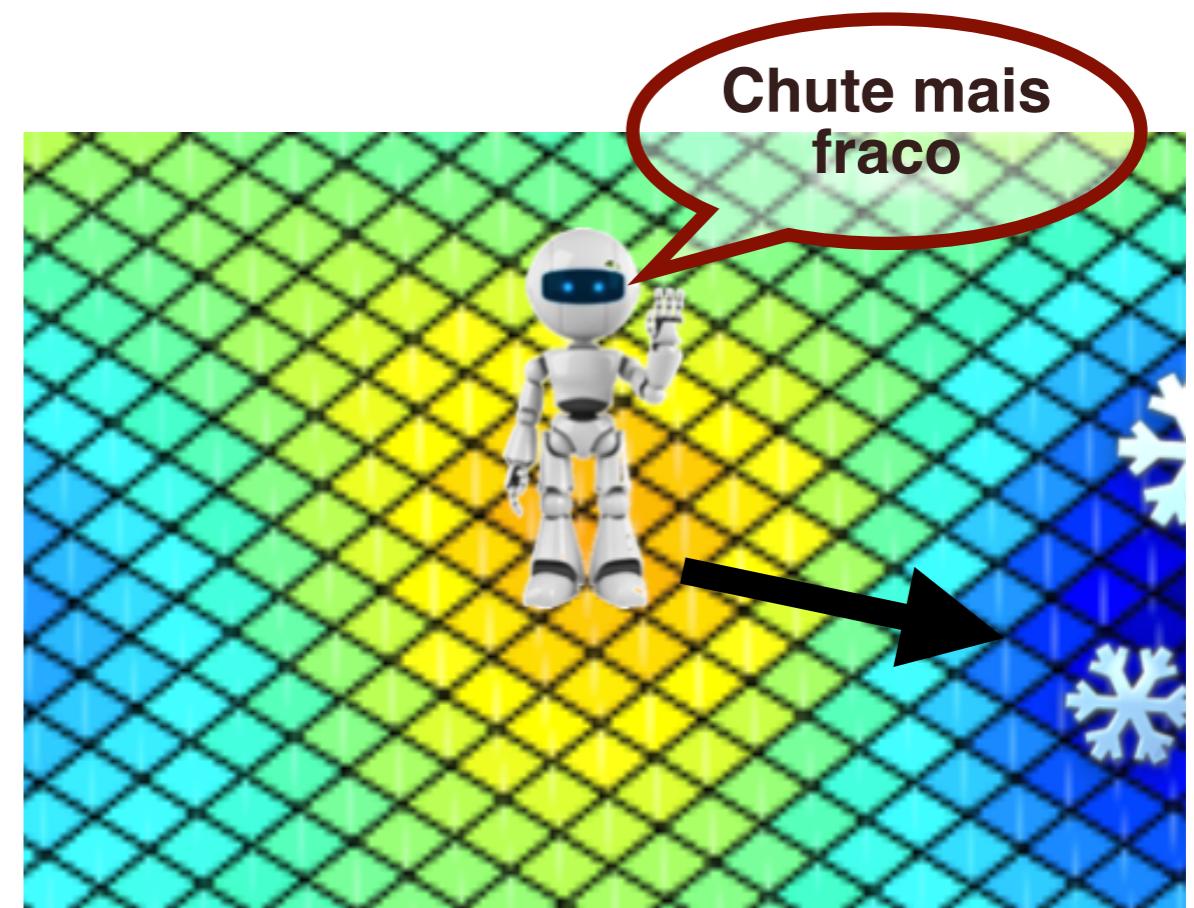
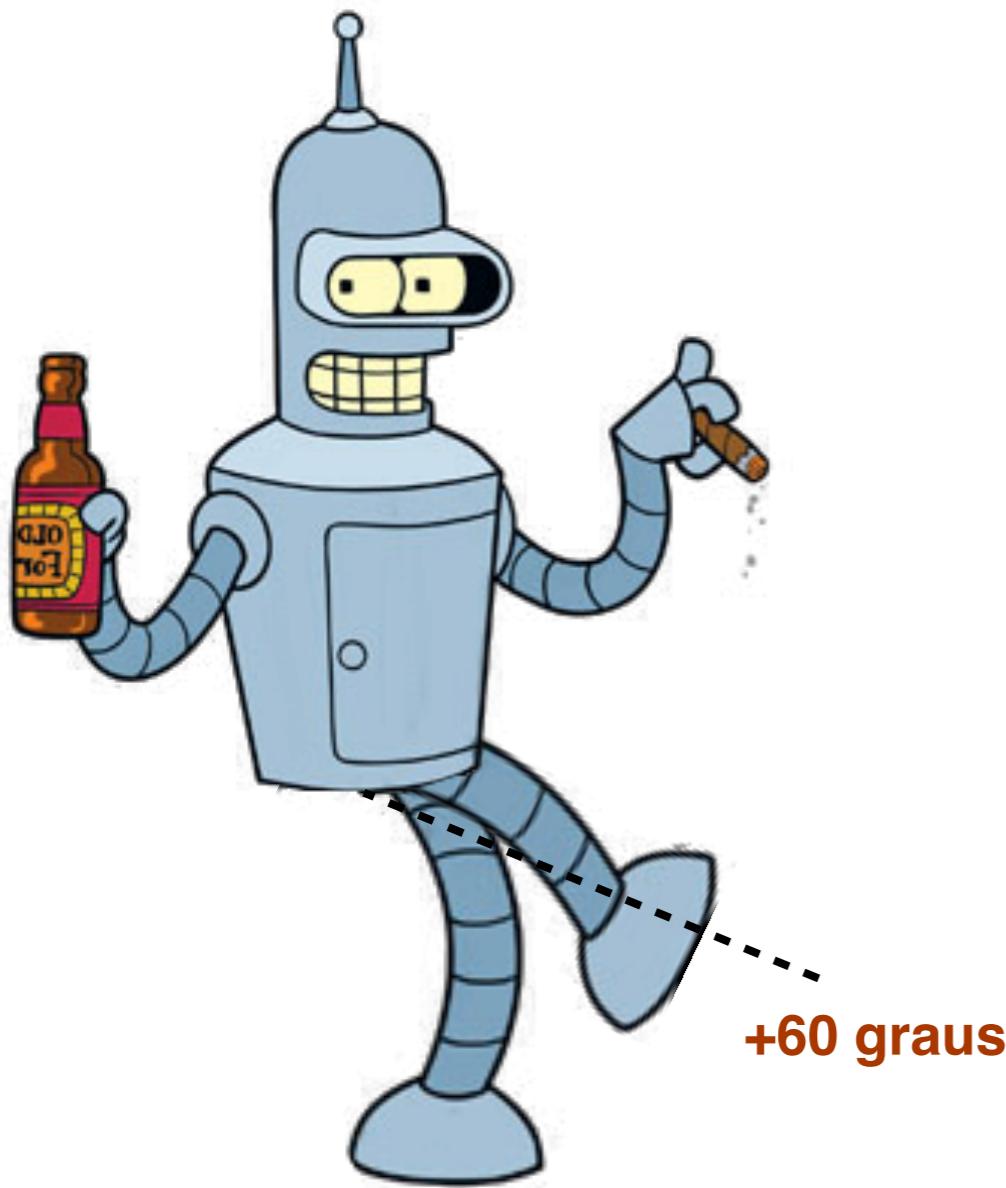
(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1=[0, -45, +30]$ $V(s_1) = 5$ metros
Estados vizinhos
 $s_2=[0, -20, +20]$ $V(s_2) = 0$ metros
 $s_3=[0, -45, +60]$ $V(s_3) = 12$ metros



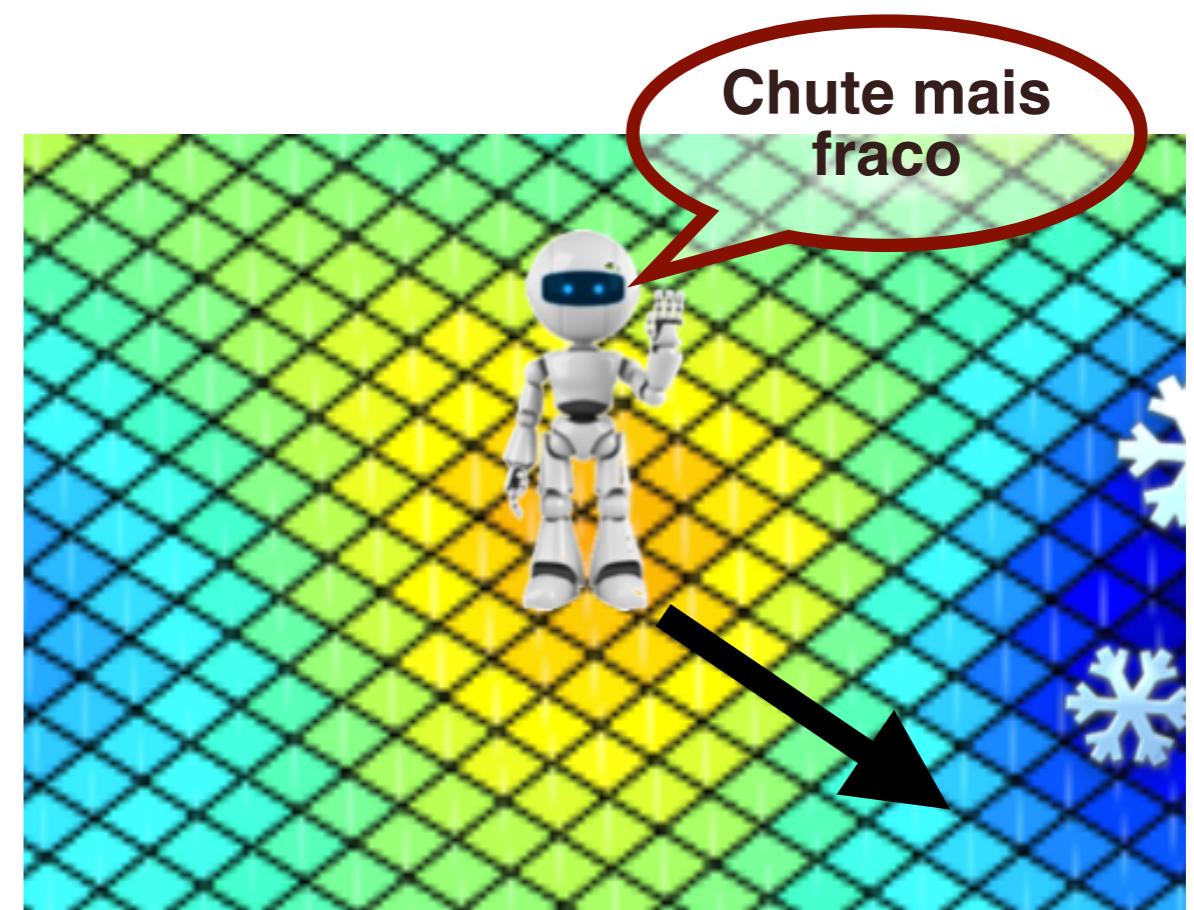
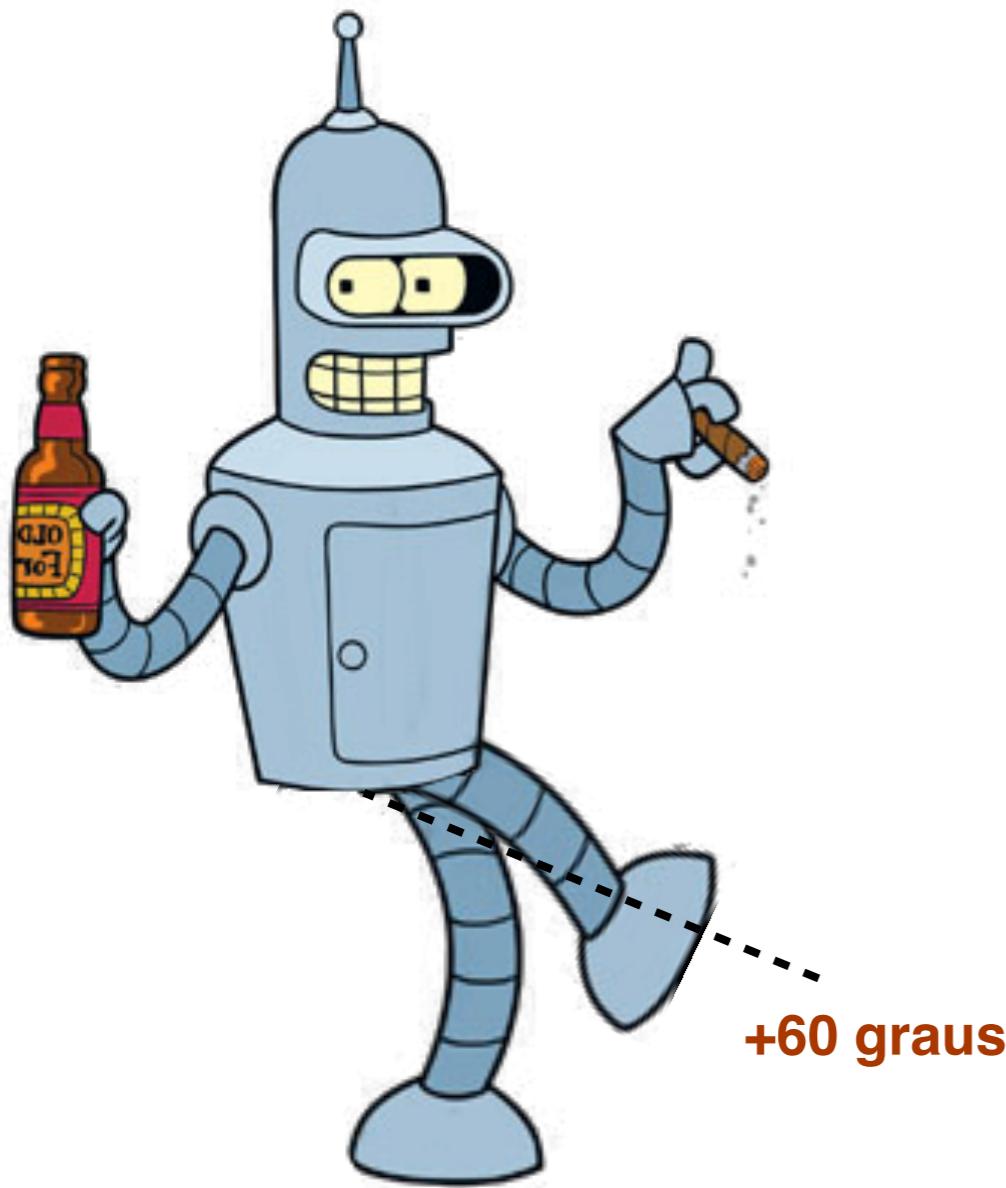
(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1=[0, -45, +30]$ $V(s_1) = 5$ metros
Estados vizinhos
 $s_2=[0, -20, +20]$ $V(s_2) = 0$ metros
 $s_3=[0, -45, +60]$ $V(s_3) = 12$ metros



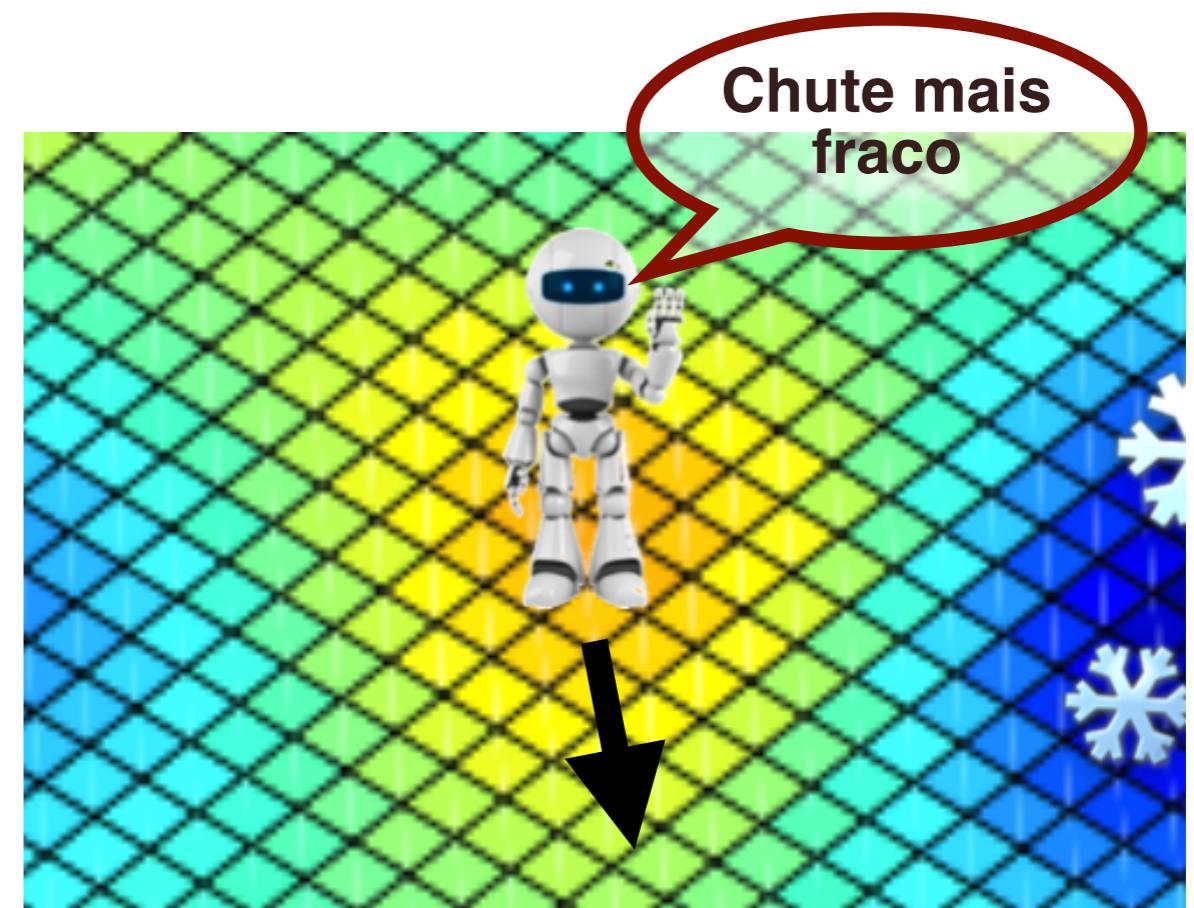
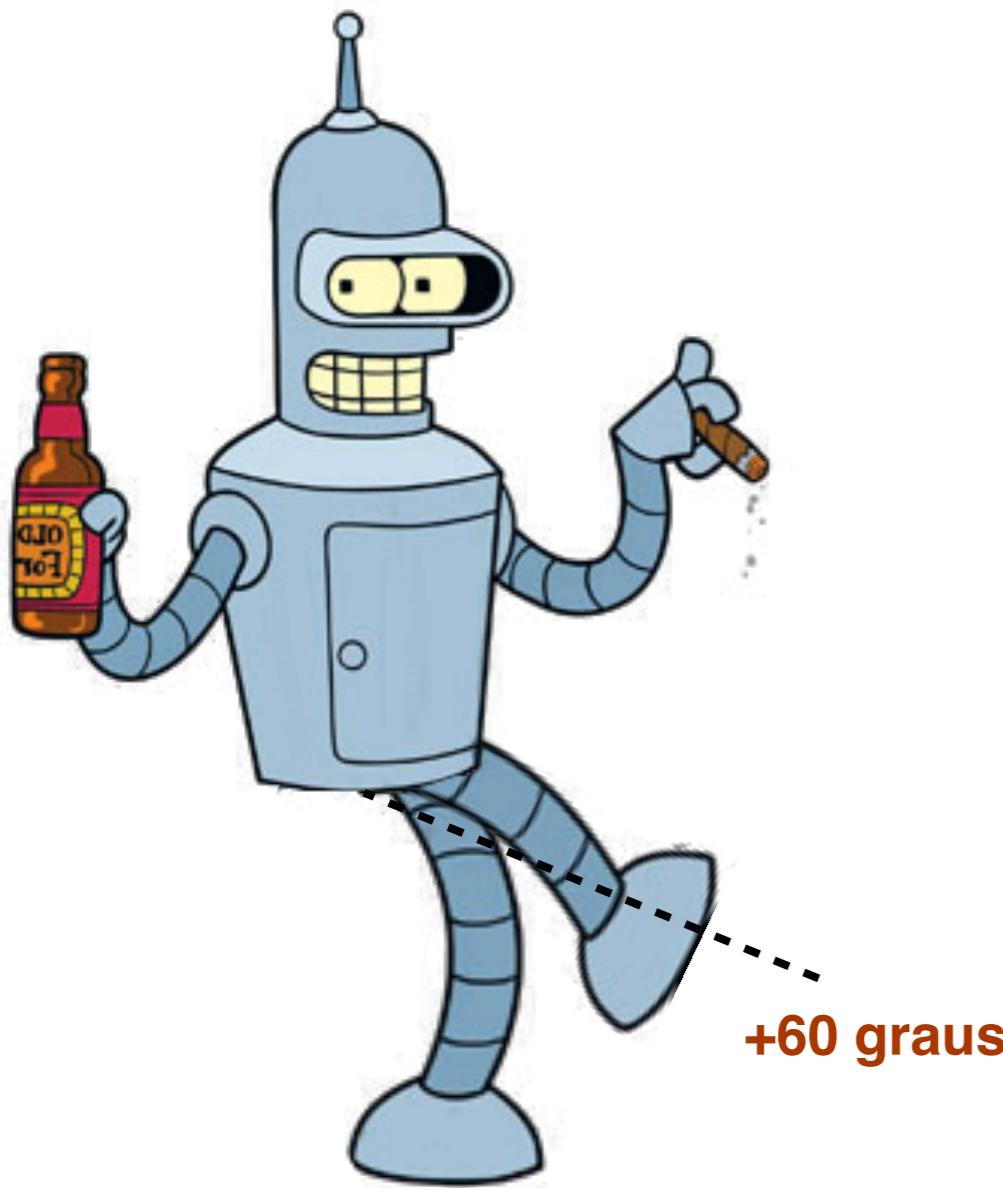
(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1 = [0, -45, +30]$ $V(s_1) = 5$ metros
Estados vizinhos
 $s_2 = [0, -20, +20]$ $V(s_2) = 0$ metros
 $s_3 = [0, -45, +60]$ $V(s_3) = 12$ metros



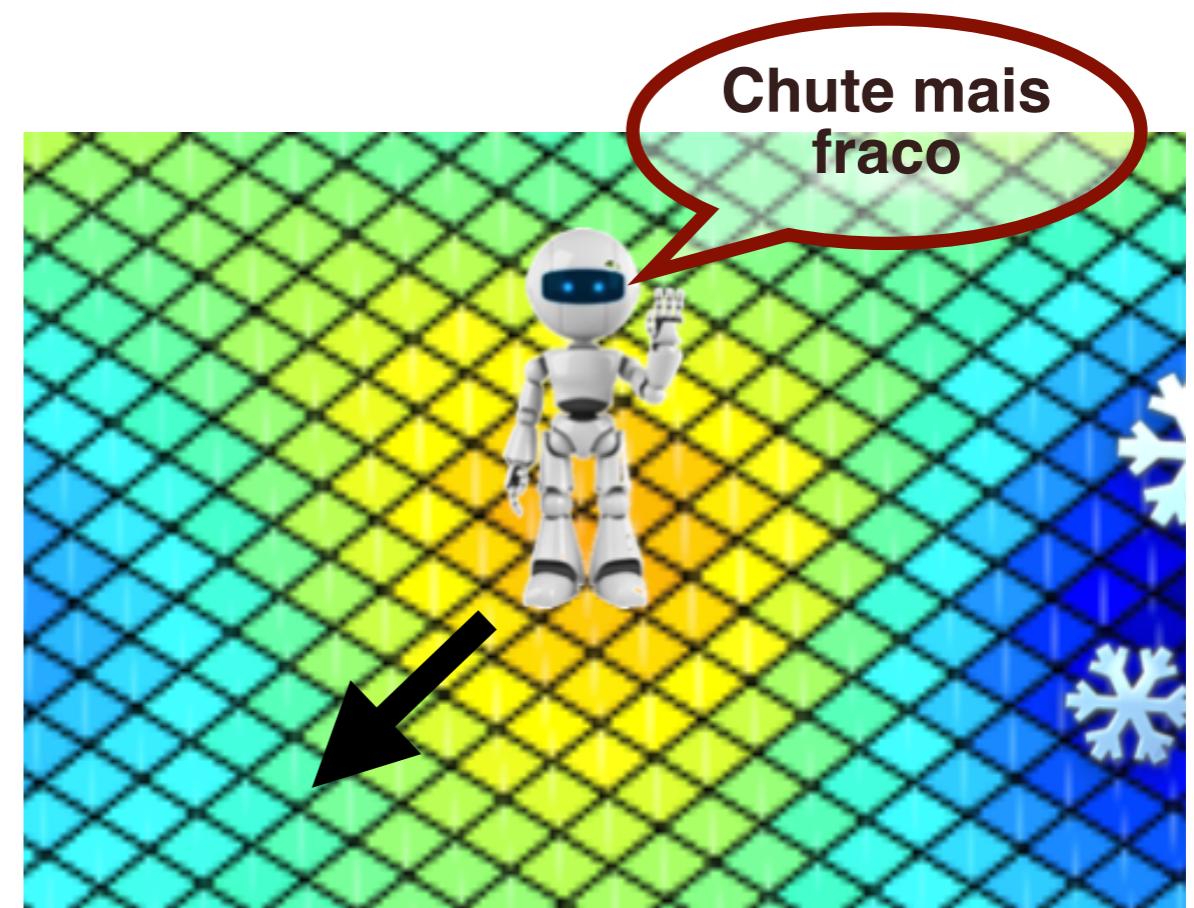
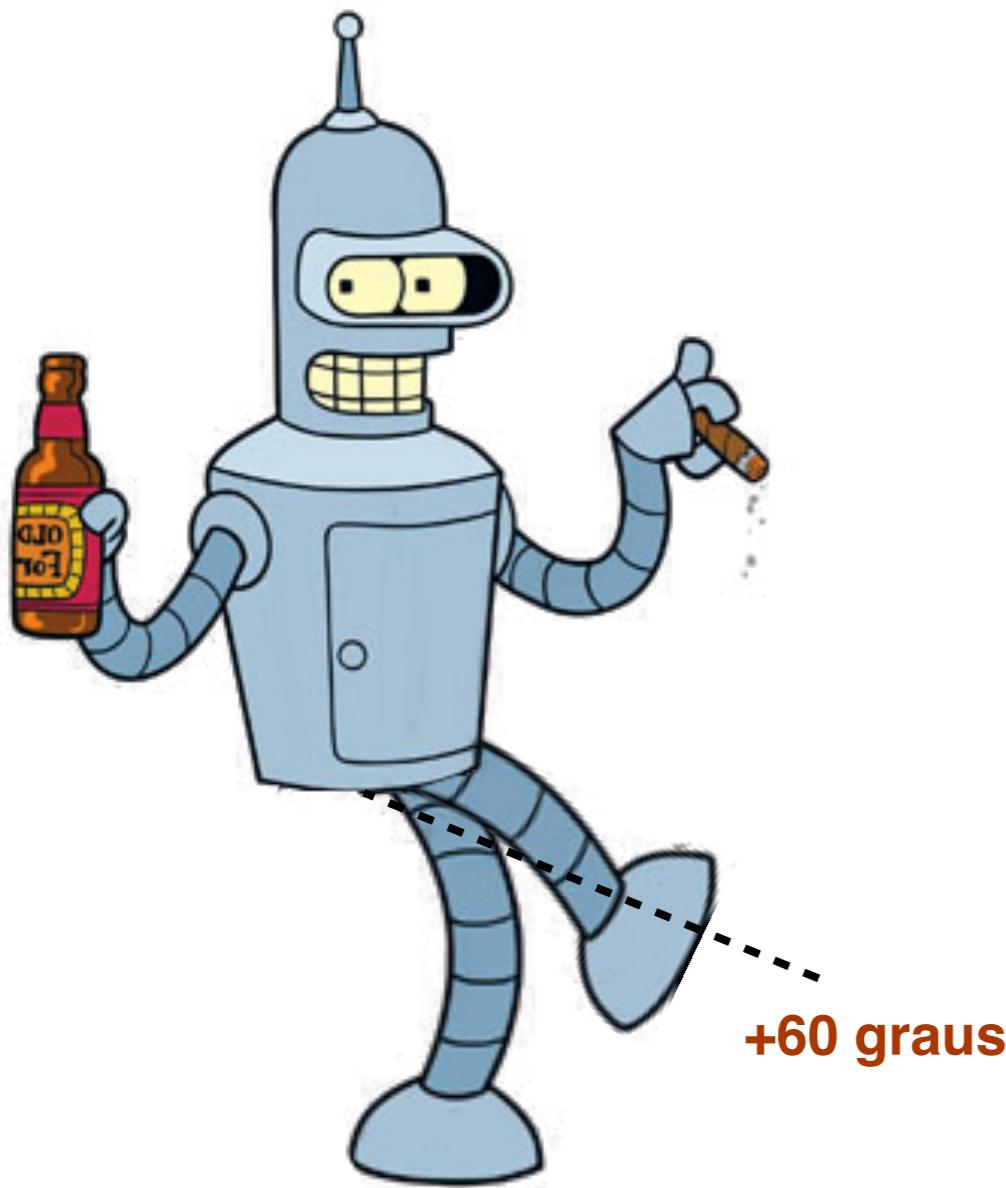
(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1 = [0, -45, +30]$ $V(s_1) = 5$ metros
Estados vizinhos
 $s_2 = [0, -20, +20]$ $V(s_2) = 0$ metros
 $s_3 = [0, -45, +60]$ $V(s_3) = 12$ metros



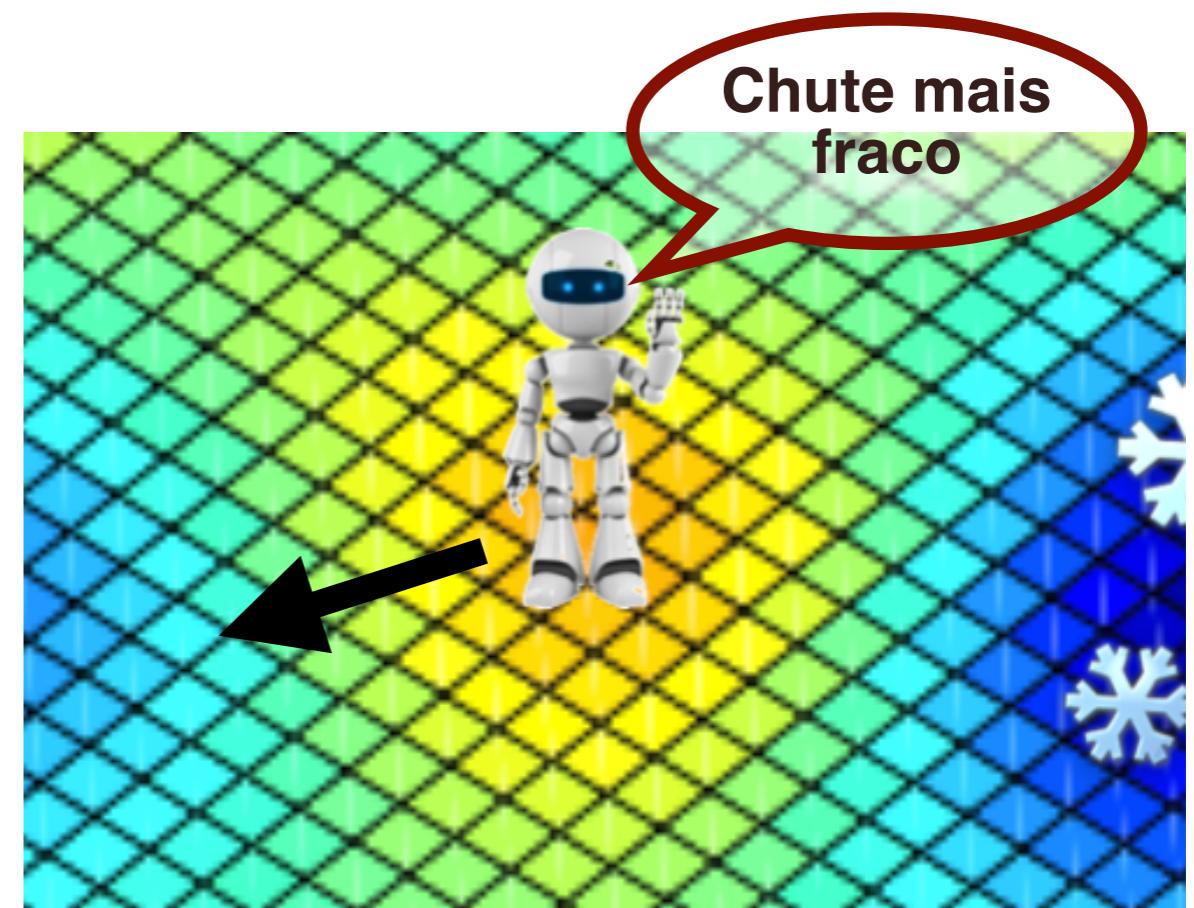
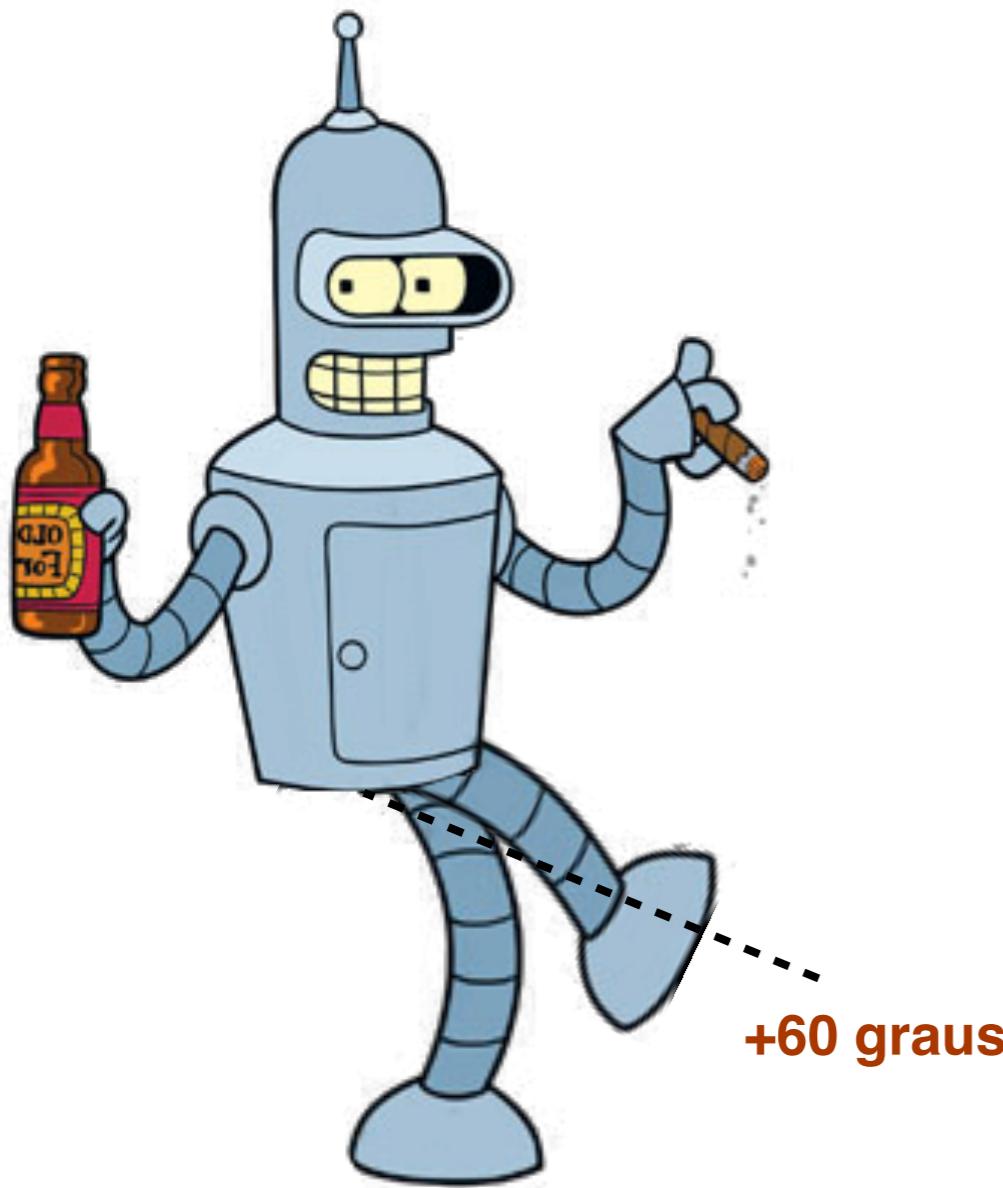
(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1 = [0, -45, +30]$ $V(s_1) = 5$ metros
Estados vizinhos
 $s_2 = [0, -20, +20]$ $V(s_2) = 0$ metros
 $s_3 = [0, -45, +60]$ $V(s_3) = 12$ metros



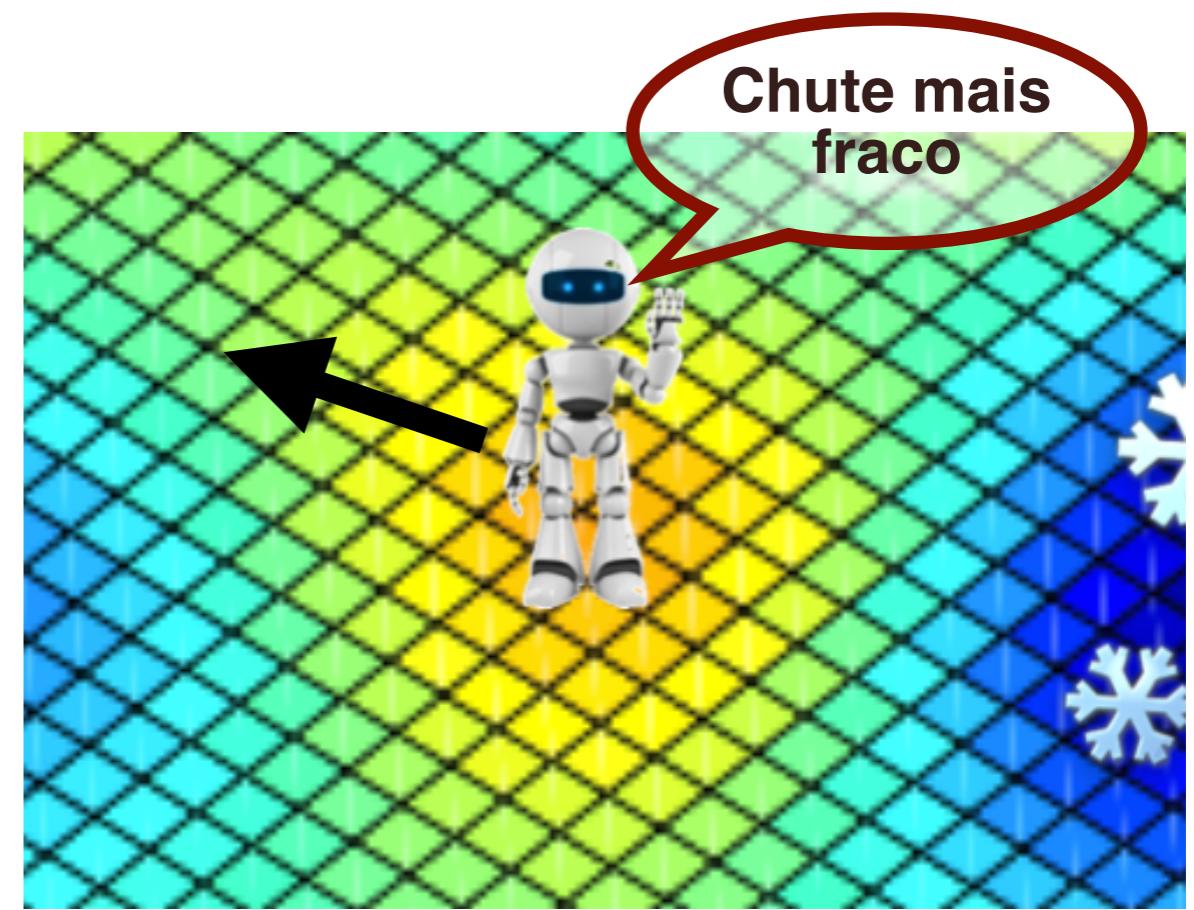
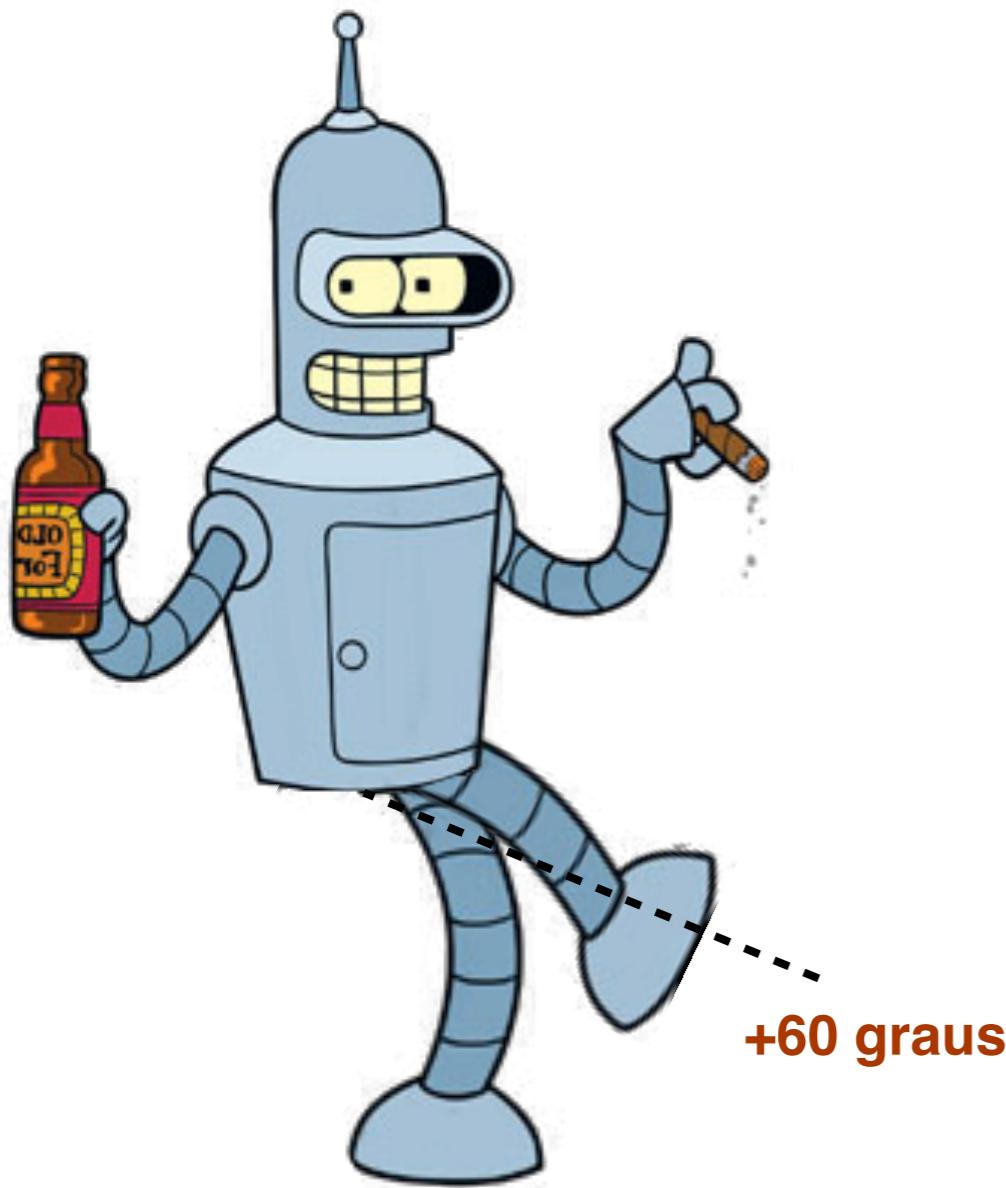
(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1=[0, -45, +30]$ $V(s_1) = 5$ metros
Estados vizinhos
 $s_2=[0, -20, +20]$ $V(s_2) = 0$ metros
 $s_3=[0, -45, +60]$ $V(s_3) = 12$ metros



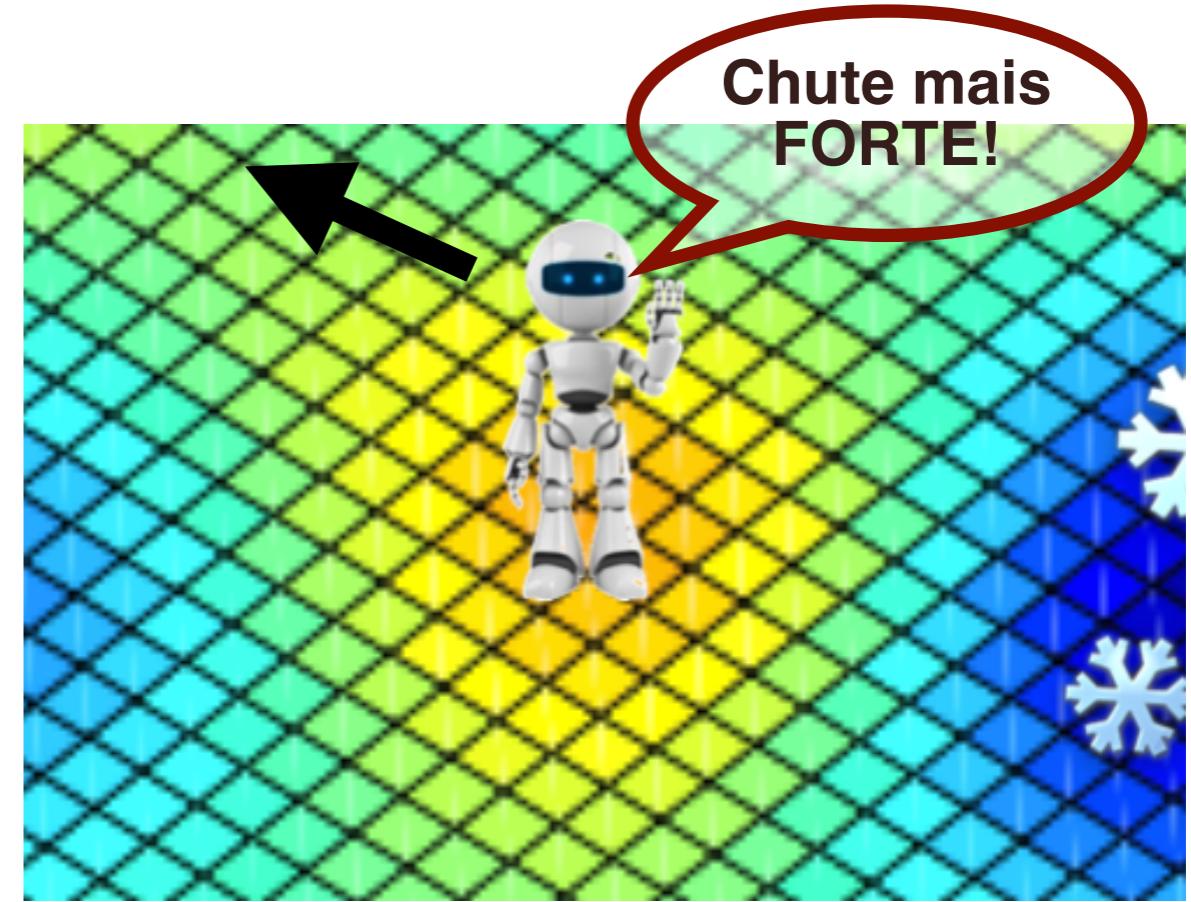
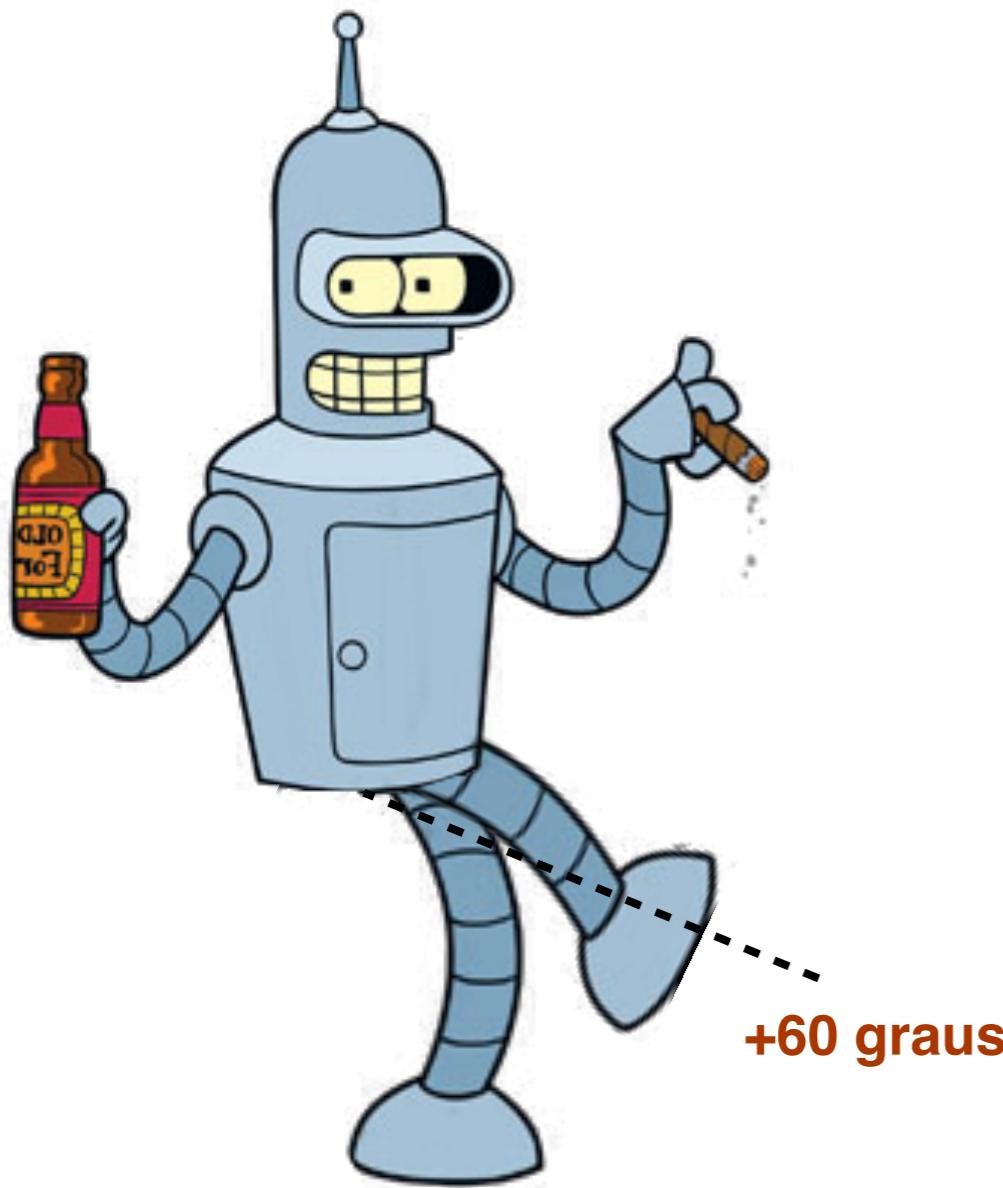
(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1 = [0, -45, +30]$ $V(s_1) = 5$ metros
Estados vizinhos
 $s_2 = [0, -20, +20]$ $V(s_2) = 0$ metros
 $s_3 = [0, -45, +60]$ $V(s_3) = 12$ metros



(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1 = [0, -45, +30]$ $V(s_1) = 5$ metros
 Estados vizinhos
 $s_2 = [0, -20, +20]$ $V(s_2) = 0$ metros
 $s_3 = [0, -45, +60]$ $V(s_3) = 12$ metros



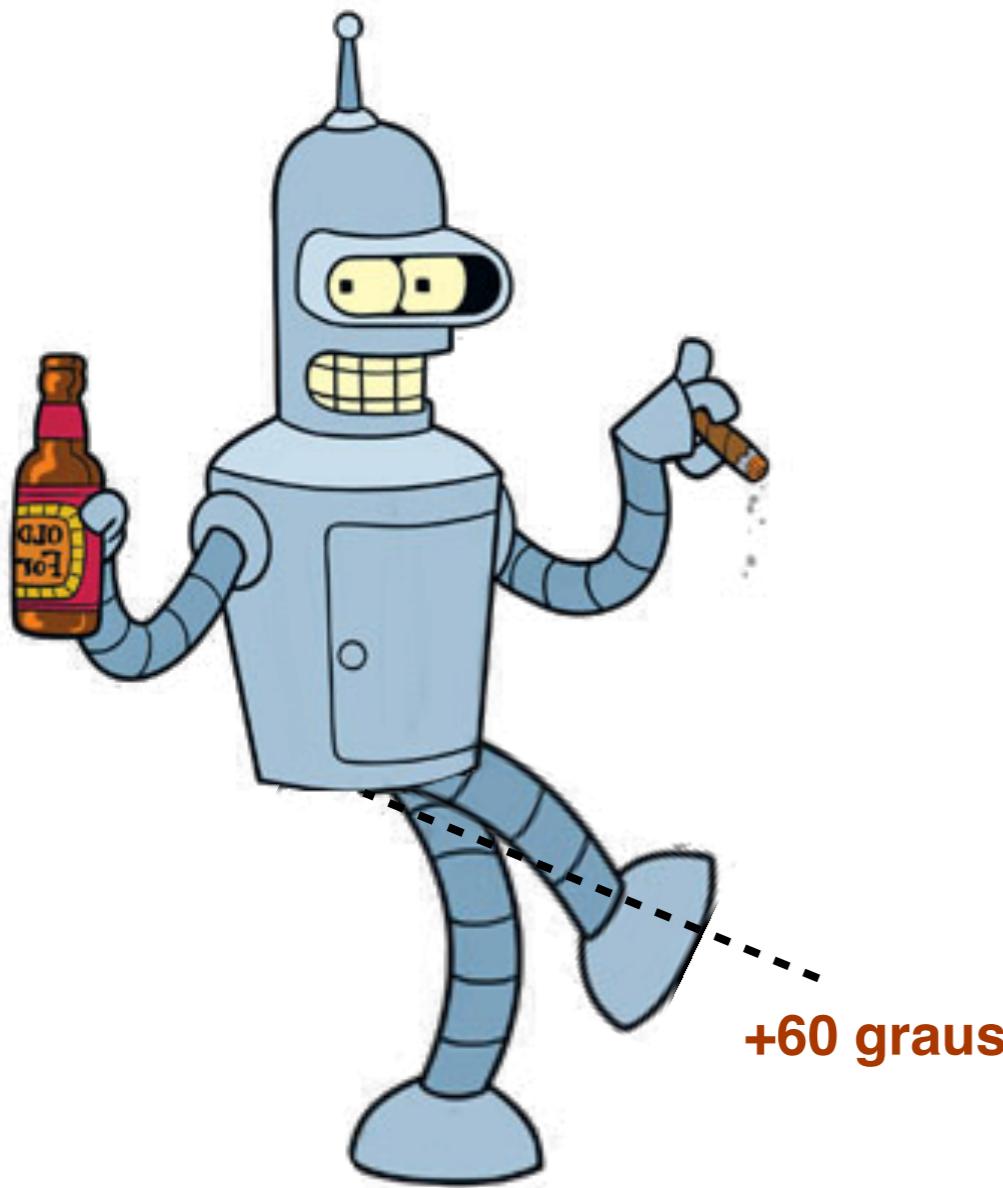
(isto é, para qual estado atualizar a solução atual s_1 ?)

$s_1=[0, -45, +30]$ $V(s_1) = 5$ metros

$s_2=[0, -20, +20]$ $V(s_2) = 0$ metros

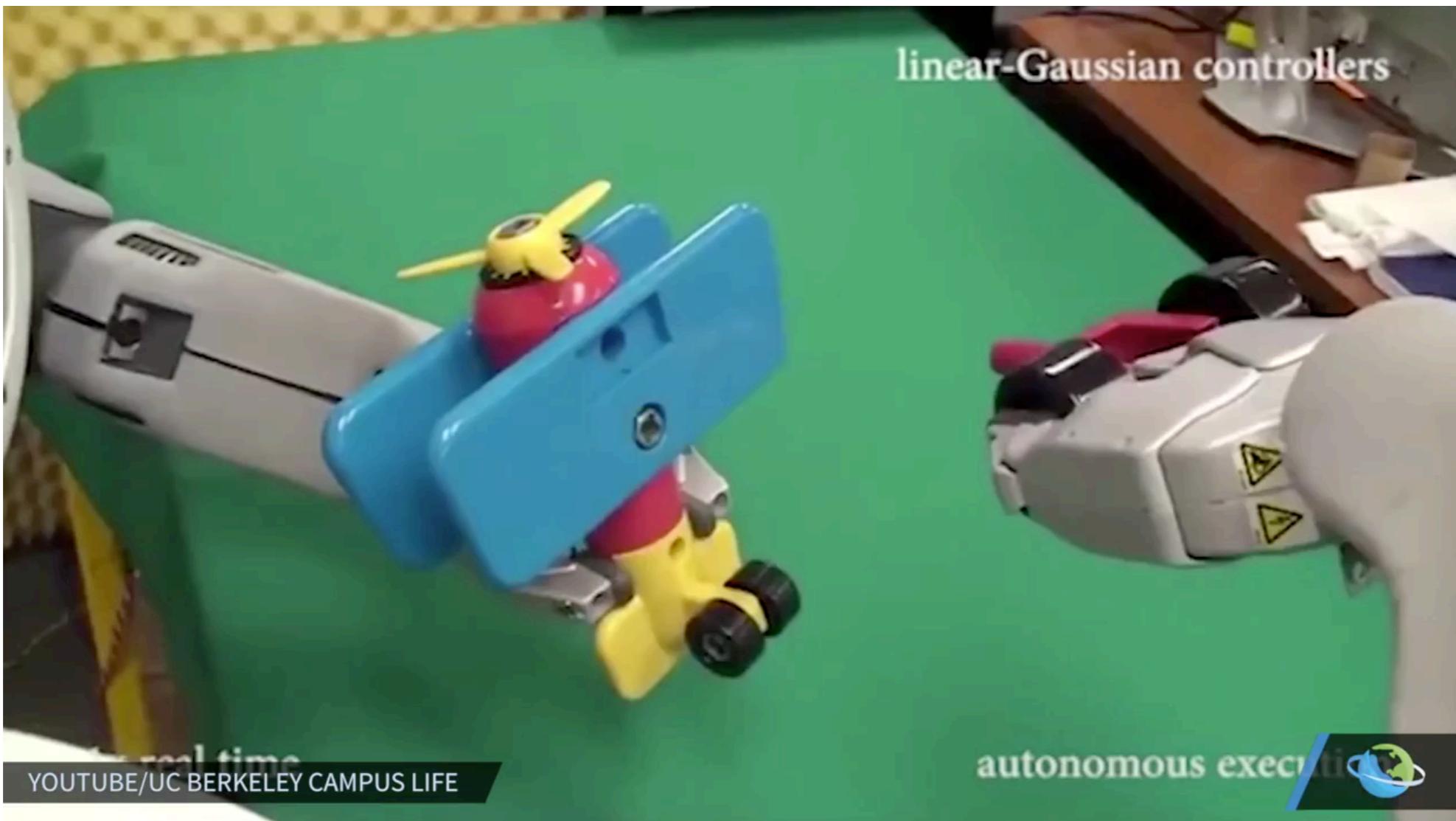
$s_3=[0, -45, +60]$ $V(s_3) = 12$ metros

Estados vizinhos



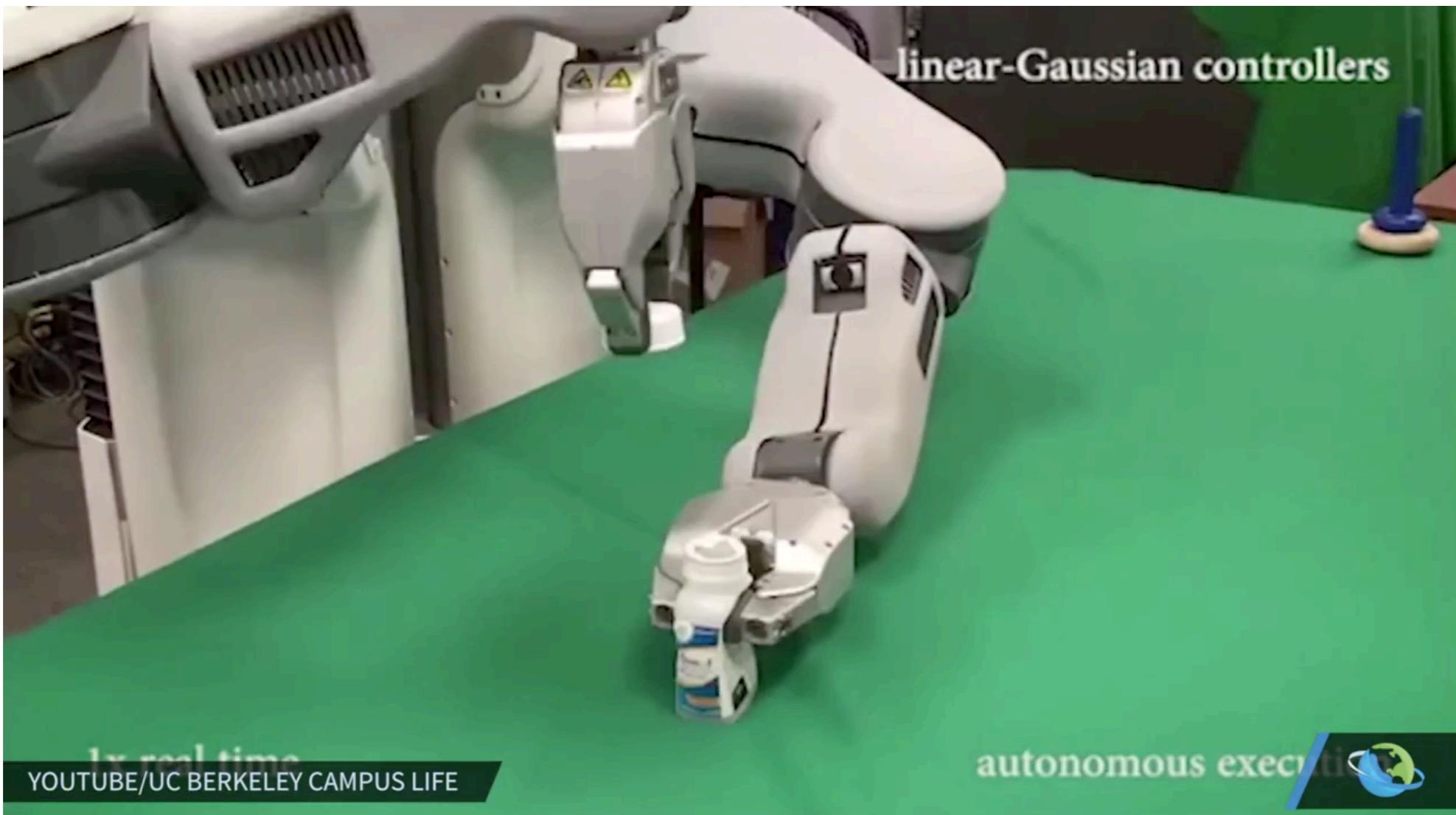
(isto é, para qual estado atualizar a solução atual s_1 ?)

Exemplo: Aprendizado Motor Fino



Sensorimotor Deep Learning (UC Berkeley)
<https://www.youtube.com/watch?v=R22W2ghrlhg>

Exemplo: Aprendizado Motor Fino

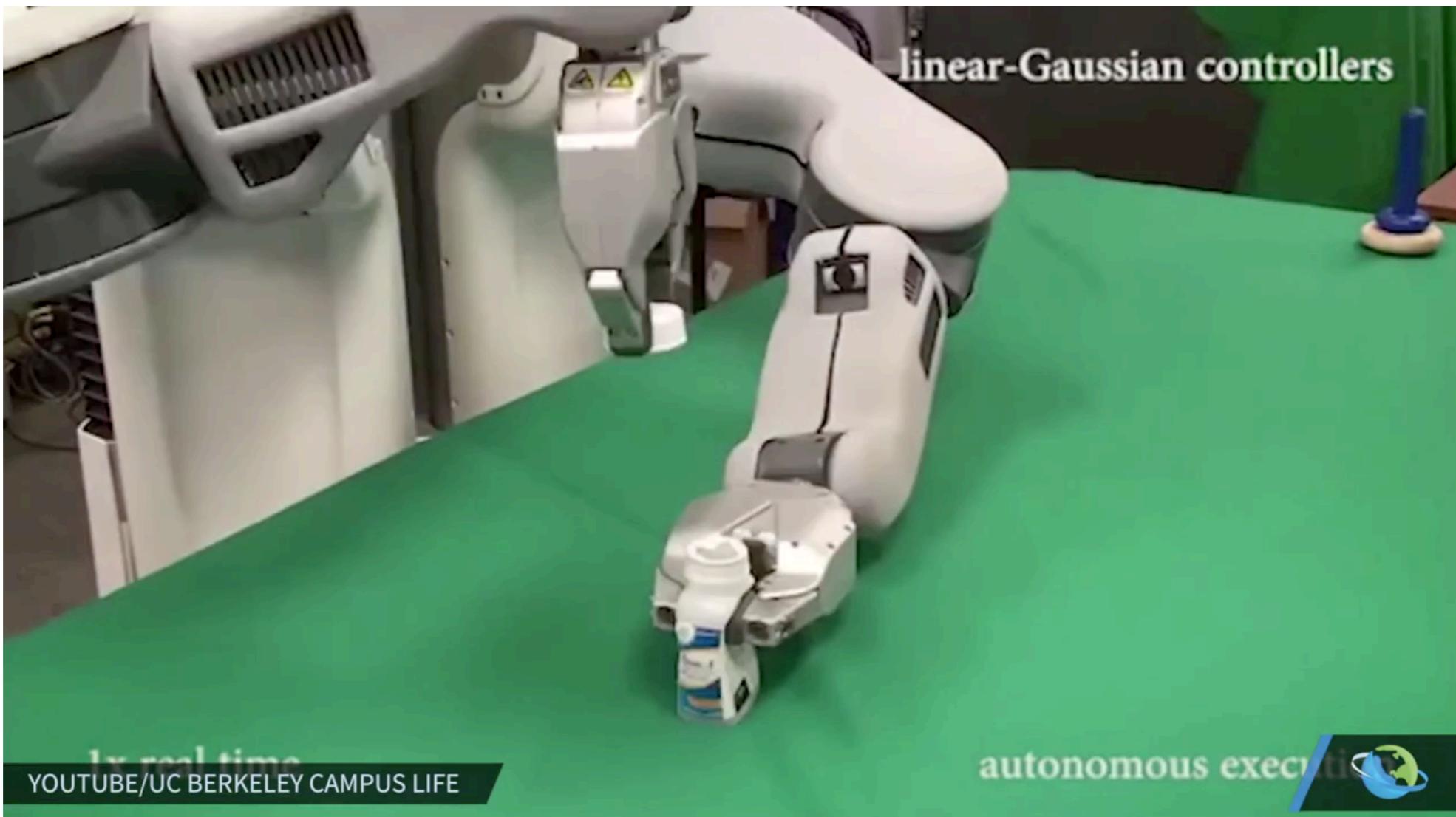


Sensorimotor Deep Learning (UC Berkeley)
<https://www.youtube.com/watch?v=R22W2ghrlhg>

Problema de fechar a garrafa de remédio:
como representar solução candidata s ?

como definir função que avalia sua performance—Valor(s)?

Exemplo: Aprendizado Motor Fino



Sensorimotor Deep Learning (UC Berkeley)
<https://www.youtube.com/watch?v=R22W2ghrlhg>

Problema de fechar a garrafa de remédio:

como representar solução candidata s ?

$s=(\text{targetXHand}, \text{targetYHand}, \text{verticalDisplacementHand}, \text{rotationHand})$

como definir função que avalia sua performance—Valor(s)?

valor número maior se consegue tocar na garrafa; e maior ainda se também a fecha tampa; etc

Hill Climbing

(Método da Subida de Encosta)

- Começa em um estado (ou solução candidata) inicial
- Olha para os estados imediatamente ao redor do atual
- Atualiza o estado para o melhor estado vizinho (i.e., vizinho com maior valor)
- Repete até que esteja em um estado no qual nenhum vizinho é melhor

Hill Climbing

(Método da Subida de Encosta)

- Começa em um estado (ou solução candidata) inicial
- Olha para os estados imediatamente ao redor do atual
- Atualiza o estado para o melhor estado vizinho (i.e., vizinho com maior valor)
- Repete até que esteja em um estado no qual nenhum vizinho é melhor



- Não mantém uma árvore de busca
- Não lembra quais estados já visitou
- Apenas repetidamente move para o melhor vizinho

por Dan Klein

Hill Climbing

(Método da Subida de Encosta)

- Começa em um estado (ou solução candidata) inicial
- Olha para os estados imediatamente ao redor do atual
- Atualiza o estado para o melhor estado vizinho (i.e., vizinho com maior valor)
- Repete até que esteja em um estado no qual nenhum vizinho é melhor



- Não mantém uma árvore de busca
- Não lembra quais estados já visitou
- Apenas repetidamente move para o melhor vizinho

**“é como escalar o Everest com neblina
e sofrendo de amnésia”**

por Dan Klein

- **HillClimbing()**

- $estado_atual \leftarrow SELEciona_ESTADO_ALEATÓRIO()$
- para $t \leftarrow 1$ até ∞
 - $V \leftarrow GERA_VIZINHOS(estado_atual)$
 - $melhor_vizinho \leftarrow estado_atual$
 - Para cada vizinho V_i em V :
 - Se $Valor(V_i) > Valor(melhor_vizinho)$
 - $melhor_vizinho \leftarrow V_i$
 - Se $Valor(melhor_vizinho) > Valor(estado_atual)$
 - $estado_atual \leftarrow melhor_vizinho$
 - Senão
 - retorna $estado_atual$

- **HillClimbing()**

- $estado_atual \leftarrow SELEciona_ESTADO_ALEATÓRIO()$
- para $t \leftarrow 1$ até ∞
 - $V \leftarrow GERA_VIZINHOS(estado_atual)$
 - $melhor_vizinho \leftarrow estado_atual$
 - Para cada vizinho V_i em V :
 - Se $Valor(V_i) > Valor(melhor_vizinho)$
 - $melhor_vizinho \leftarrow V_i$
 - Se $Valor(melhor_vizinho) > Valor(estado_atual)$
 - $estado_atual \leftarrow melhor_vizinho$
 - Senão
 - retorna $estado_atual$

Como?

- **HillClimbing()**

- $estado_atual \leftarrow SELEICIONA_ESTADO_ALEATÓRIO()$

p.ex., $estado_atual = [10.1 \quad 7.3]$

- para $t \leftarrow 1$ até ∞

- $V \leftarrow GERA_VIZINHOS(estado_atual)$

- $melhor_vizinho \leftarrow estado_atual$

- Para cada vizinho V_i em V :

- Se $Valor(V_i) > Valor(melhor_vizinho)$

- $melhor_vizinho \leftarrow V_i$

- Se $Valor(melhor_vizinho) > Valor(estado_atual)$

- $estado_atual \leftarrow melhor_vizinho$

- Senão

- retorna $estado_atual$

Como?

- **HillClimbing()**

- $estado_atual \leftarrow SELEciona_ESTADO_ALEATÓRIO()$

p.ex., $estado_atual = [10.1 \quad 7.3]$

- para $t \leftarrow 1$ até ∞

- $V \leftarrow GERA_VIZINHOS(estado_atual)$

- $melhor_vizinho \leftarrow estado_atual$

- Para cada vizinho V_i em V :

- Se $Valor(V_i) > Valor(melhor_vizinho)$

- $melhor_vizinho \leftarrow V_i$

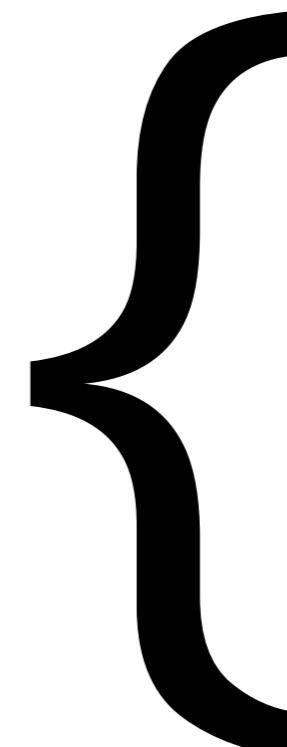
- Se $Valor(melhor_vizinho) > Valor(estado_atual)$

- $estado_atual \leftarrow melhor_vizinho$

- Senão

- retorna $estado_atual$

Como?



[10.1+0 7.3+0]

[10.1+0 7.3+ ε]

[10.1+0 7.3 - ε]

[10.1+ ε 7.3+0]

[10.1+ ε 7.3+ ε]

[10.1+ ε 7.3 - ε]

[10.1- ε 7.3+0]

[10.1- ε 7.3+ ε]

[10.1- ε 7.3 - ε]

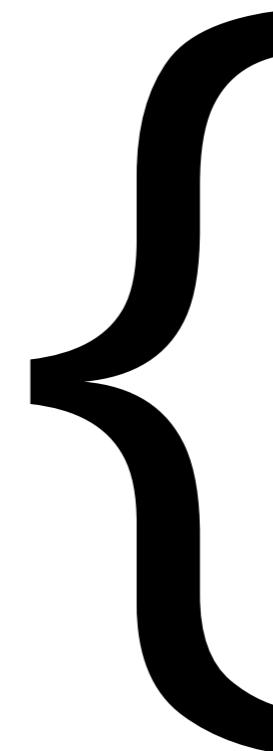
para algum valor pequeno de ε

• HillClimbing()

- $estado_atual \leftarrow SELEICIONA_ESTADO_ALEATÓRIO()$
- para $t \leftarrow 1$ até ∞
 - $V \leftarrow GERA_VIZINHOS(estado_atual)$
 - $melhor_vizinho \leftarrow estado_atual$
 - Para cada vizinho V_i em V :
 - Se $Valor(V_i) > Valor(melhor_vizinho)$
 - $melhor_vizinho \leftarrow V_i$
 - Se $Valor(melhor_vizinho) > Valor(estado_atual)$
 - $estado_atual \leftarrow melhor_vizinho$
 - Senão
 - retorna $estado_atual$

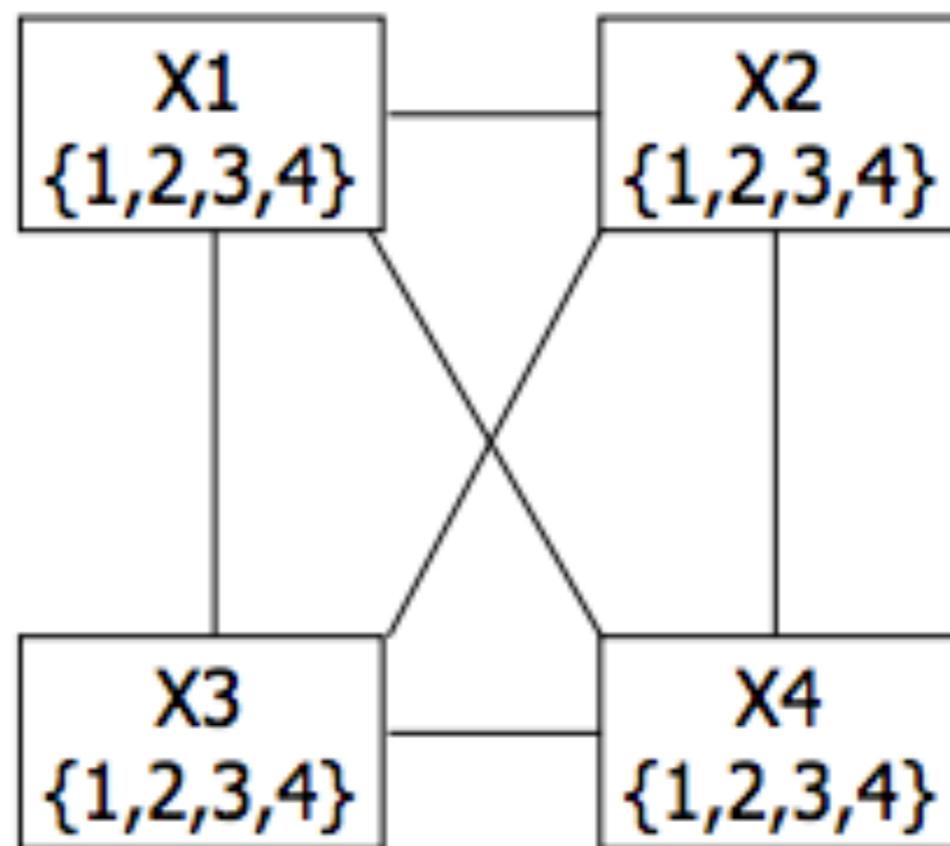
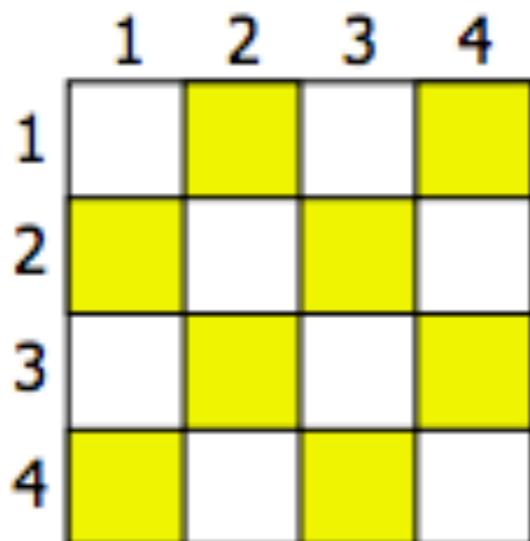
p.ex., $estado_atual = [10.1 \quad 7.3]$

Como?

- 
- $V \leftarrow \{ \}$
 - Para i de 1 até N:
 - $perturbação \leftarrow [rnd_1, rnd_2]$
 - $V_i = estado_atual + perturbação$
 - $V \leftarrow V \cup V_i$
 - Retorna V

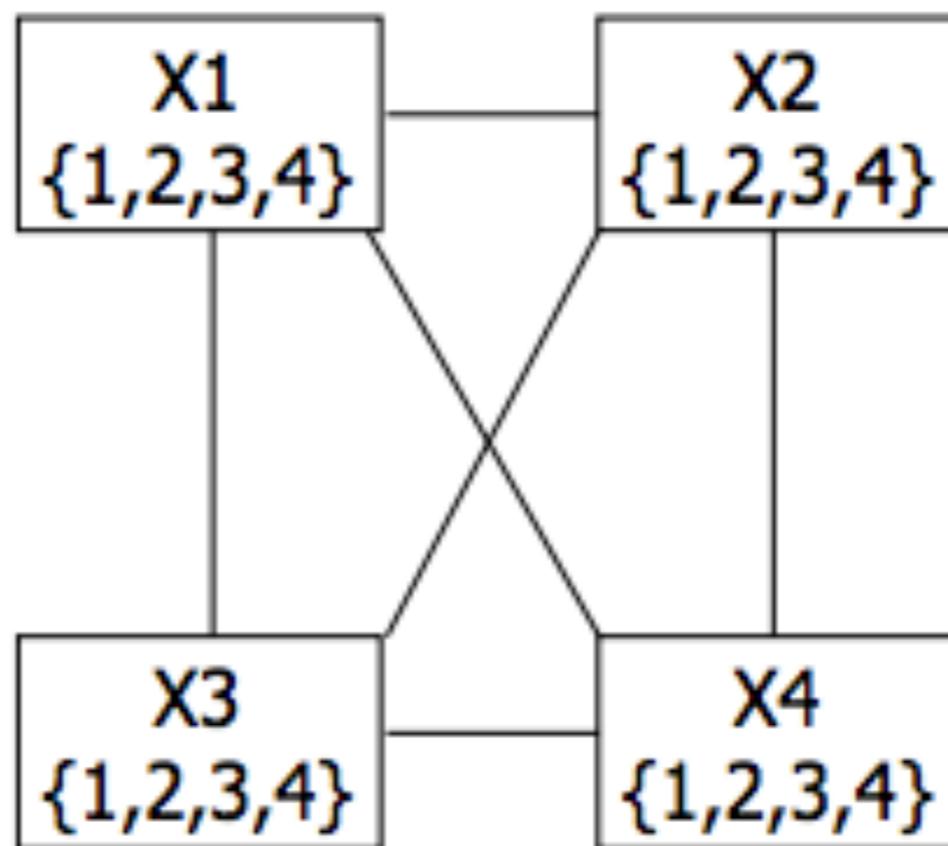
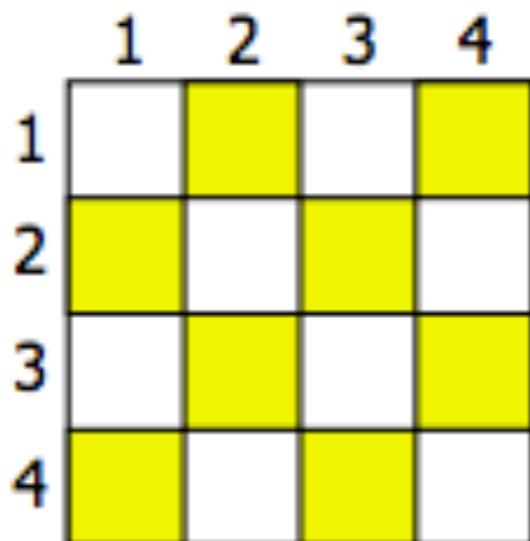
onde rnd_1 e rnd_2 são valores aleatórios,
p.ex., amostrados de uma $Normal(0, \sigma)$
com σ pequeno

O Problema das 4 Rainhas



- uma rainha X_i em cada coluna
- cada rainha X_i pode ser colocada em uma linha: $D_i = \{1, 2, 3, 4\}$
- restrições: rainhas não podem estar na mesma linha ou diagonal

O Problema das 4 Rainhas resolvido via *Hill Climbing*



- uma rainha X_i em cada coluna
- cada rainha X_i pode ser colocada em uma linha: $D_i = \{1,2,3,4\}$
- restrições: rainhas não podem estar na mesma linha ou diagonal

O Problema das 8 Rainhas resolvido via *Hill Climbing*

O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual:
- Estados vizinhos:
- Valor(s):

O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos:
- Valor(s):

O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso move uma rainha em sua coluna (quantos?)
- Valor(s):

O Problema das 8 Rainhas resolvido via *Hill Climbing*

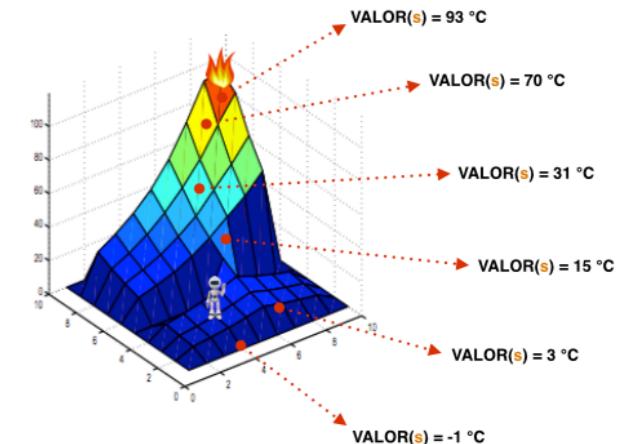
- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso move uma rainha em sua coluna (*quantos?*)
- Valor(s): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso move uma rainha em sua coluna (*quantos?*)
- Valor(s): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

NOTE:

- No Problema das Duas Salas
 - Valor(s) = temperatura em s
 - definido assim já que o objetivo do agente era maximizar a temperatura
 - i.e., achar o estado s com maior Valor(s)

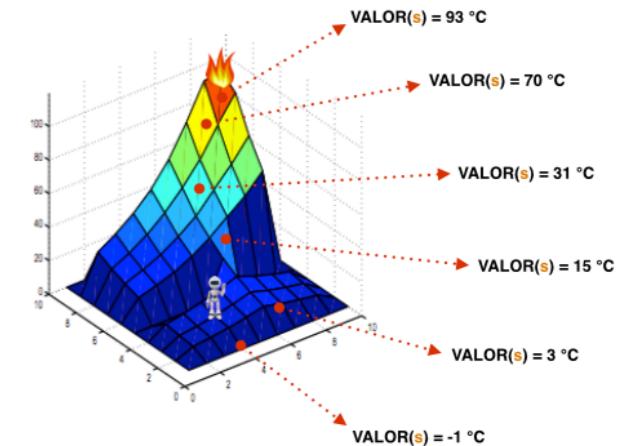


O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso move uma rainha em sua coluna (*quantos?*)
- Valor(s): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

NOTE:

- No Problema das Duas Salas
 - Valor(s) = temperatura em s
 - definido assim já que o objetivo do agente era maximizar a temperatura
 - i.e., achar o estado s com maior Valor(s)
- De forma geral, a função Valor(s) reflete o objetivo do agente
 - i.e., especifica quais estados o agente prefere
 - no caso do Problema das 8 Rainhas, estados onde poucos pares de rainhas conseguem se atacar



O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso mova uma rainha em sua coluna (*quantos?*)
- Valor(s): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

Neste estado **s**, Valor(**s**)=17

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	15	13	16	13	16
15	14	14	17	15	14	16	16
17	14	16	18	15	17	15	16
18	14	15	15	15	14	17	16
14	14	13	17	12	14	12	18

O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso mova uma rainha em sua coluna (*quantos?*)
- Valor(s): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	15	13	16	13	16
15	14	14	17	15	14	16	16
17	14	16	18	15	17	15	16
18	14	15	15	15	14	16	16
14	14	13	17	12	14	12	18

Neste estado **s**, Valor(**s**)=17

Demais valores indicam o valor de estados vizinhos, dependendo do movimento escolhido

O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso mova uma rainha em sua coluna (*quantos?*)
- Valor(s): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

Próximo passo:

?

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	15	13	16	13	16
15	14	14	15	13	16	13	16
17	14	17	15	15	14	16	16
17	17	16	18	15	17	15	17
18	14	15	15	15	14	17	16
14	14	13	17	12	14	12	18

O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso mova uma rainha em sua coluna (*quantos?*)
- Valor(s): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

Próximo passo:

mover para qualquer estado
com valor 12

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	14	13	16	13	16
17	14	17	15	15	14	16	16
18	14	17	15	15	14	17	16
14	14	13	17	12	14	12	18

O Problema das 8 Rainhas resolvido via *Hill Climbing*

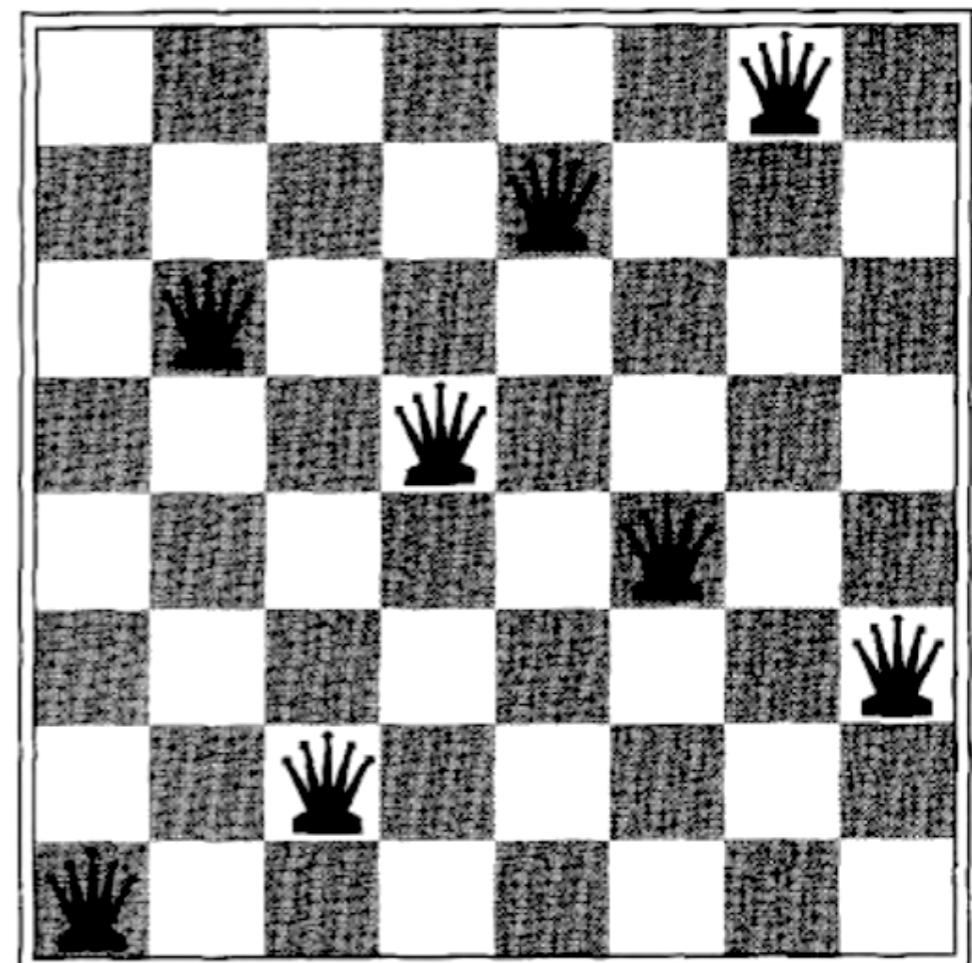
- Estado **s** atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso mova uma rainha em sua coluna (*quantos?*)
- Valor(**s**): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	14	14	13	16	13
14	14	17	15	15	14	16	16
17	14	16	18	15	14	16	16
18	14	14	15	15	14	14	16
14	14	13	17	12	14	12	18

em 5 passos
alcança este
mínimo local



Valor(**s**)=1



O Problema das 8 Rainhas resolvido via *Hill Climbing*

- Estado s atual: configuração completa do tabuleiro com as 8 rainhas
- Estados vizinhos: todas configurações de tabuleiro caso mova uma rainha em sua coluna (*quantos?*)
- Valor(s): número de pares de rainhas que conseguem se atacar (mesma linha ou diagonal)

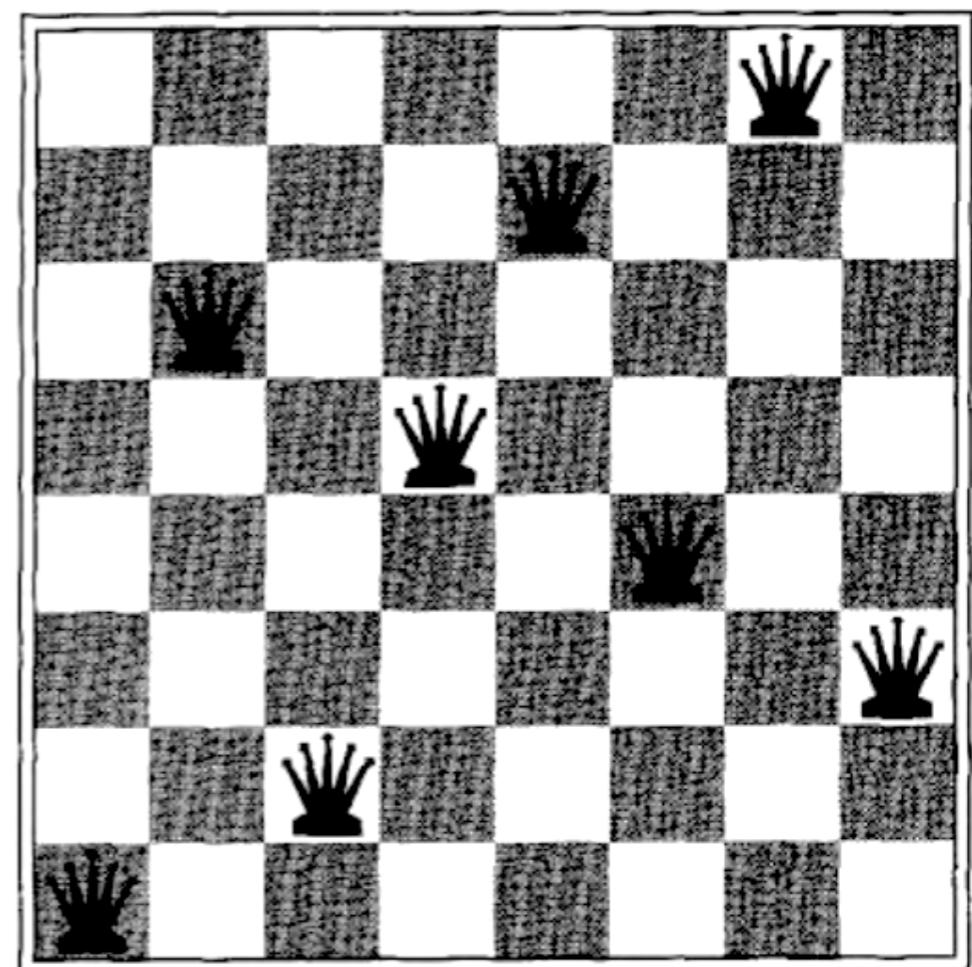
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	14	14	13	16	13
14	14	17	15	15	14	16	16
17	14	16	18	15	15	14	16
18	14	14	15	15	14	14	16
14	14	13	17	12	14	12	18

em 5 passos
alcança este
mínimo local



Valor(s)=1

Como escapar
desse mínimo?



O Problema das 8 Rainhas resolvido via *Hill Climbing*

- O que fazer se todos vizinhos forem piores ou iguais ao estado atual?
 - Se pára o algoritmo: HC resolve apenas 14% dos tabuleiros (se inicializado aleatoriamente)
 - Se move um vizinho aleatório de igual valor: resolve 94% dos tabuleiros!

O Problema das 8 Rainhas resolvido via *Hill Climbing*

- O que fazer se todos vizinhos forem piores ou iguais ao estado atual?
 - Se pára o algoritmo: HC resolve apenas 14% dos tabuleiros (se inicializado aleatoriamente)
 - Se move um vizinho aleatório de igual valor: resolve 94% dos tabuleiros!
- Variações:
 - Hill Climbing com Primeira Escolha: move para primeiro melhor vizinho encontrado
 - (caso haja muitos vizinhos para considerar)
 - Hill Climbing com re-inícios aleatórios: roda HC a partir de vários estados iniciais
 - (caso haja muitos máximos locais)
 - etc.

Busca Local (revisitada)

Motivação

- problems em que o caminho até o estado objetivo não importa
- problems de otimização
 - cada estado possui um valor; queremos encontrar o estado com maior valor

Busca Local (revisitada)

Motivação

- problemas em que o caminho até o estado objetivo não importa
- problemas de otimização
 - cada estado possui um valor; queremos encontrar o estado com maior valor
- uso é possível mesmo em problemas muito complexos
 - (nos quais busca sistemática seria muito demorada)
- usa pouca memória (normalmente, uma quantidade constante)
- encontra soluções razoáveis em espaços de estados grandes/infinitos/contínuos
 - (nos quais busca cega ou informada não seriam possíveis ou adequados)

Hill Climbing

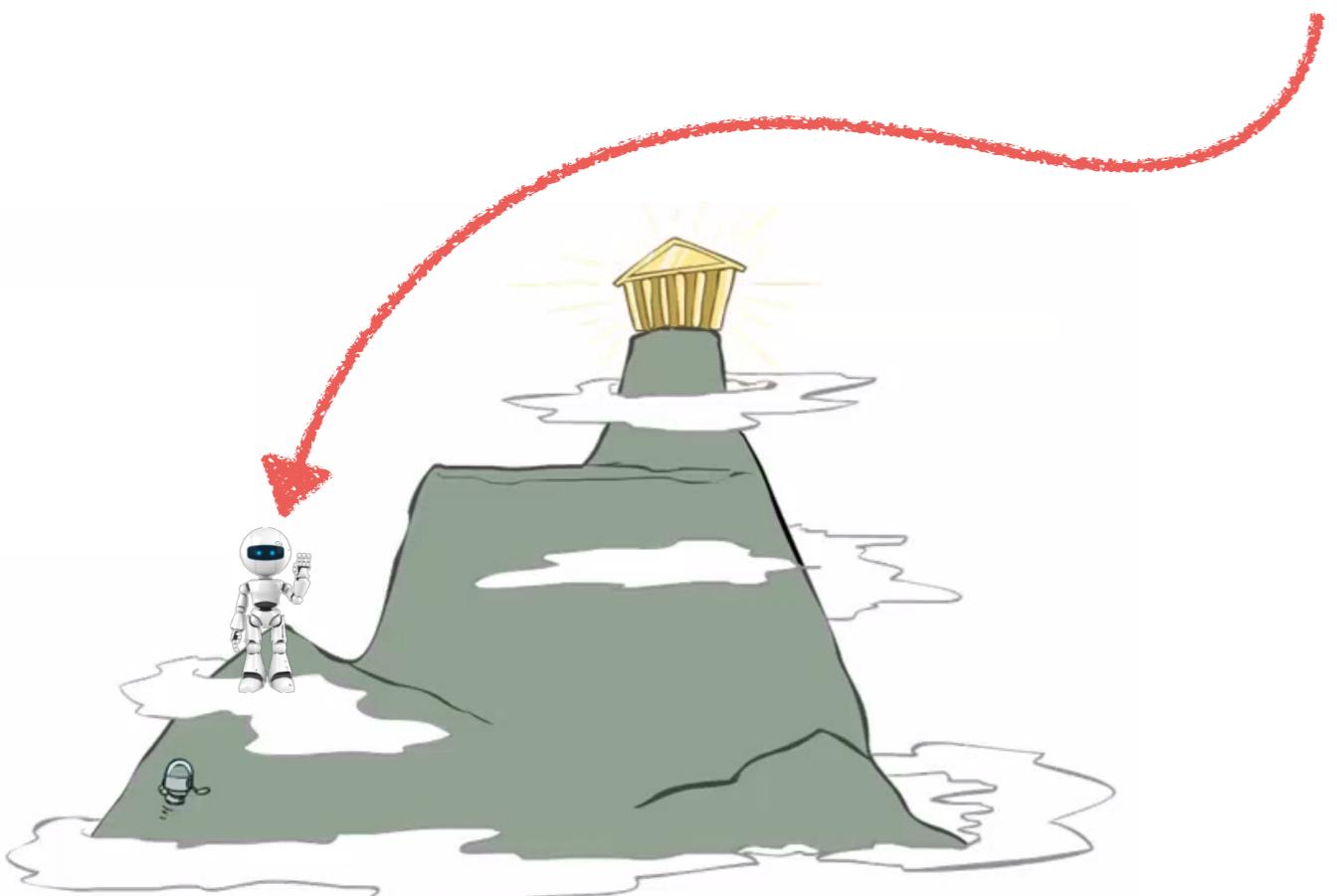
(Método da Subida de Encosta)

- Movimentos são permitidos *apenas* se forem localmente bons (*miopia*)
- Resultado: frequentemente alcança um máximo local e não consegue sair dele

Hill Climbing

(Método da Subida de Encosta)

- Movimentos são permitidos apenas se forem localmente bons (*miopia*)
- Resultado: frequentemente alcança um máximo local e não consegue sair dele

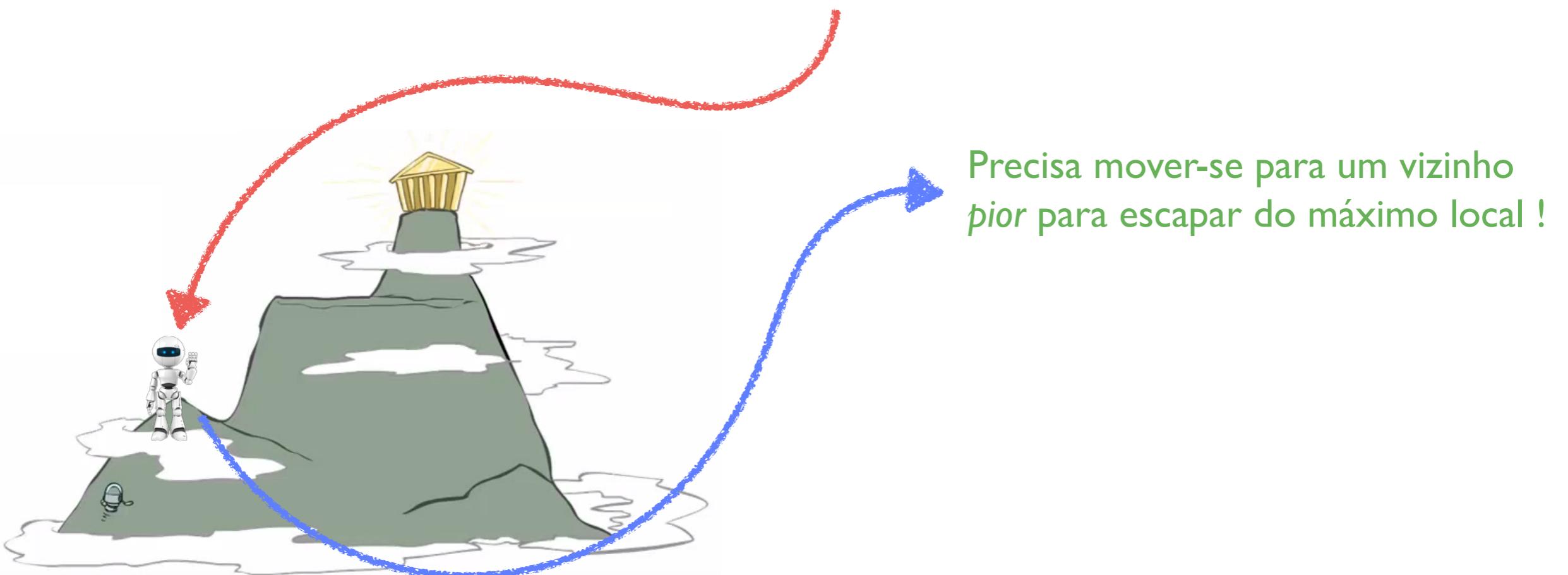


por Dan Klein

Hill Climbing

(Método da Subida de Encosta)

- Movimentos são permitidos apenas se forem localmente bons (*miopia*)
- Resultado: frequentemente alcança um máximo local e não consegue sair dele



por Dan Klein

Simulated Annealing

(Método da Têmpera Simulada)

- Idéia
 - Fugir de máximos locais permitindo alguns movimentos “ruins”
 - mas diminuindo gradualmente a frequência com que esses movimentos são permitidos

Simulated Annealing

(Método da Têmpera Simulada)

- Idéia
 - Fugir de máximos locais permitindo alguns movimentos “ruins”
 - mas diminuindo gradualmente a frequência com que esses movimentos são permitidos
- A freqüência permitida de movimentos ruins é regulada por um parâmetro de temperatura **T**
 - temperatura decresce ao longo do tempo
 - portanto, movimentações para estados piores são permitidas mais e mais raramente

Simulated Annealing

(Método da Têmpera Simulada)

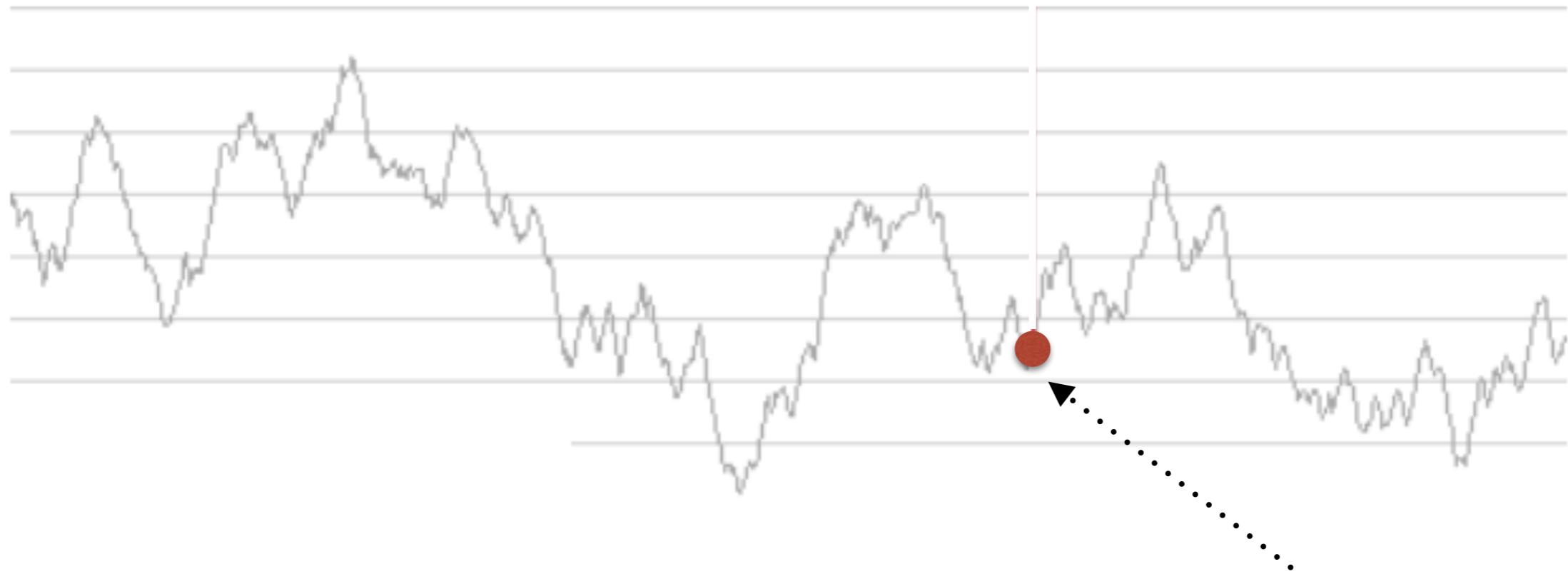
- Idéia
 - Fugir de máximos locais permitindo alguns movimentos “ruins”
 - mas diminuindo gradualmente a frequência com que esses movimentos são permitidos
 - A freqüência permitida de movimentos ruins é regulada por um parâmetro de temperatura **T**
 - temperatura decresce ao longo do tempo
 - portanto, movimentações para estados piores são permitidas mais e mais raramente
 - Temperatura **alta** → “saltos” maiores no espaço de soluções (mesmo se para vizinhos *muito* piores)
 - Temperatura **baixa** → se movimenta para estados vizinhos melhores (ou só *um pouco* piores)

Simulated Annealing

(Método da Têmpera Simulada)

- Metáfora:

- bola de ping-pong em uma mesa cheia de buracos; queremos que caia no buraco mais profundo
- mas ela tende a cair na primeira depressão! (mínimo local)
- mas, se chacoalharmos a mesa...



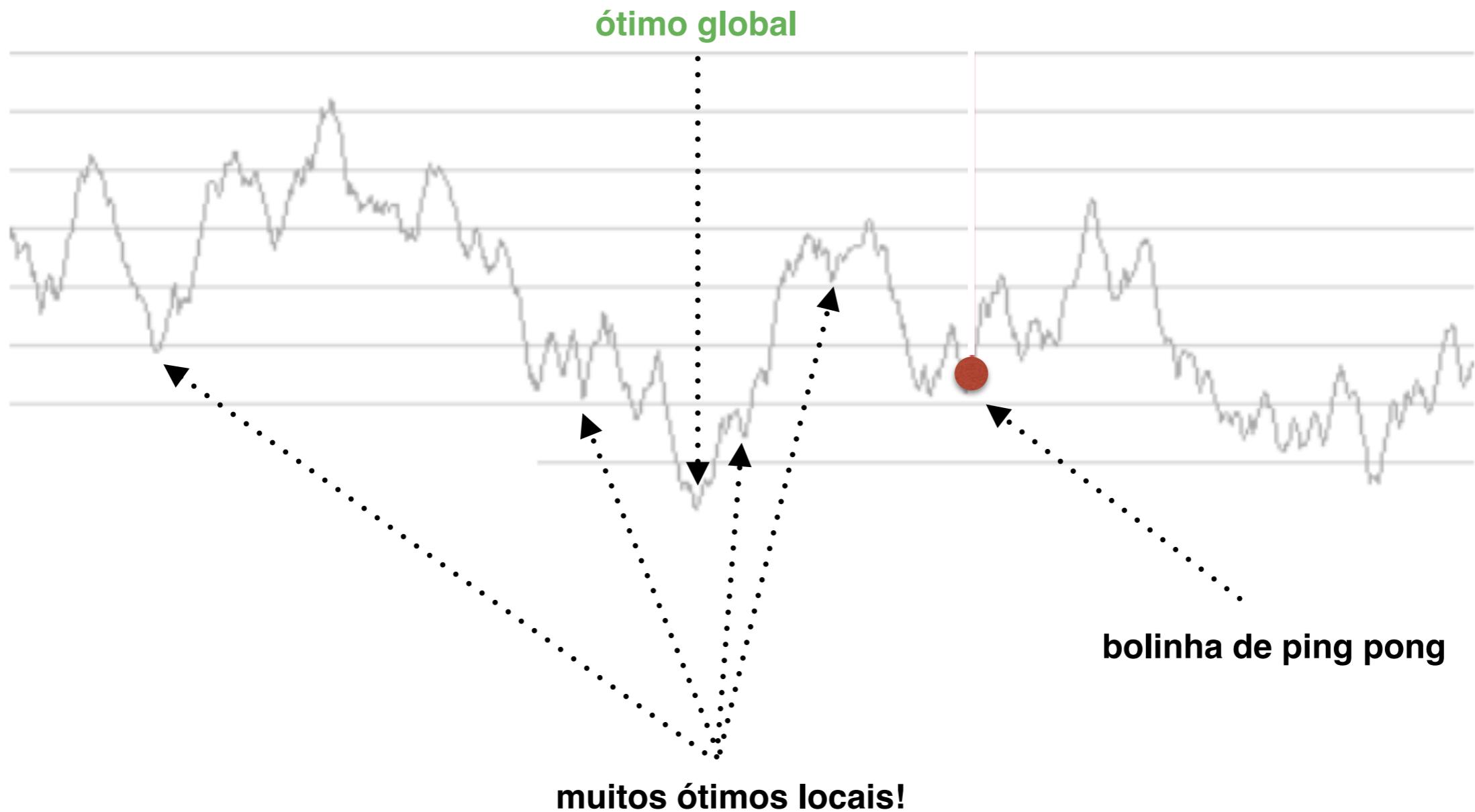
bolinha de ping pong

Simulated Annealing

(Método da Têmpera Simulada)

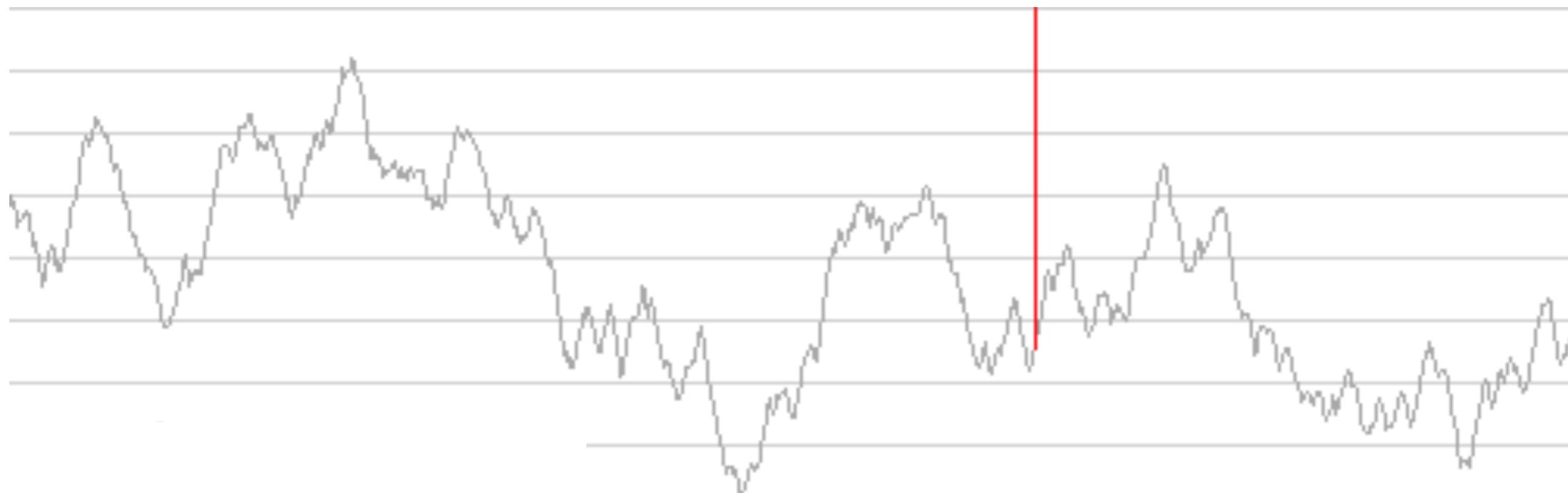
- Metáfora:

- bola de ping-pong em uma mesa cheia de buracos; queremos que caia no buraco mais profundo
- mas ela tende a cair na primeira depressão! (mínimo local)
- mas, se chacoalharmos a mesa...



Simulated Annealing

(Método da Têmpera Simulada)



Temperature: 25.0

- **Simulated_Annealing()**

- $\text{estado_atual} \leftarrow \text{SELEciona_ESTADO_ALEATÓRIO}()$
- para $t \leftarrow 1$ até ∞
 - $T \leftarrow \text{DECRESCE_TEMPERATURA}(t)$
 - Se $T = 0$, retorna estado_atual
 - Para $i \leftarrow 1$ até N
 - $\text{estado_candidato} \leftarrow \text{SELEciona_VIZINHO_ALEATÓRIO}(\text{estado_atual})$
 - $\Delta E \leftarrow \text{Valor}(\text{estado_candidato}) - \text{Valor}(\text{estado_atual})$
 - Se $\Delta E > 0$
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$
 - Senão
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$ com probabilidade $e^{\Delta E/T}$

- **Simulated_Annealing()**

- $\text{estado_atual} \leftarrow \text{SELEÇÃO ESTADO ALEATÓRIO}()$
- para $t \leftarrow 1$ até ∞
 - $T \leftarrow \text{DECRESCE TEMPERATURA}(t)$
 - Se $T = 0$, retorna estado_atual
 - Para $i \leftarrow 1$ até N
 - $\text{estado_candidato} \leftarrow \text{SELEÇÃO VIZINHO ALEATÓRIO}(\text{estado_atual})$
 - $\Delta E \leftarrow \text{Valor}(\text{estado_candidato}) - \text{Valor}(\text{estado_atual})$
 - Se $\Delta E > 0$
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$
 - Senão
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$ com probabilidade $e^{\Delta E/T}$

T

P.ex., se $\Delta E = -2$ e $T=10$

baixo P.ex., se $\Delta E = -20$ e $T=10$

- **Simulated_Annealing()**

- $\text{estado_atual} \leftarrow \text{SELEÇÃO ESTADO ALEATÓRIO}()$
- para $t \leftarrow 1$ até ∞
 - $T \leftarrow \text{DECRESCE TEMPERATURA}(t)$
 - Se $T = 0$, retorna estado_atual
 - Para $i \leftarrow 1$ até N
 - $\text{estado_candidato} \leftarrow \text{SELEÇÃO VIZINHO ALEATÓRIO}(\text{estado_atual})$
 - $\Delta E \leftarrow \text{Valor}(\text{estado_candidato}) - \text{Valor}(\text{estado_atual})$
 - Se $\Delta E > 0$
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$
 - Senão
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$ com probabilidade $e^{\Delta E/T}$

T

P.ex., se $\Delta E = -2$ e $T=10$

prob(aceitar esse vizinho pior) = $e^{-2/10} = 0.81 = 81\%$

baixo P.ex., se $\Delta E = -20$ e $T=10$

- **Simulated_Annealing()**

- $\text{estado_atual} \leftarrow \text{SELEÇÃO ESTADO ALEATÓRIO}()$
- para $t \leftarrow 1$ até ∞
 - $T \leftarrow \text{DECRESCE TEMPERATURA}(t)$
 - Se $T = 0$, retorna estado_atual
 - Para $i \leftarrow 1$ até N
 - $\text{estado_candidato} \leftarrow \text{SELEÇÃO VIZINHO ALEATÓRIO}(\text{estado_atual})$
 - $\Delta E \leftarrow \text{Valor}(\text{estado_candidato}) - \text{Valor}(\text{estado_atual})$
 - Se $\Delta E > 0$
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$
 - Senão
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$ com probabilidade $e^{\Delta E/T}$

T

P.ex., se $\Delta E = -2$ e $T=10$

prob(aceitar esse vizinho pior) = $e^{-2/10} = 0.81 = 81\%$

baixo P.ex., se $\Delta E = -20$ e $T=10$

prob(aceitar esse vizinho pior) = $e^{-20/10} = 0.13 = 13\%$

- **Simulated_Annealing()**

- $\text{estado_atual} \leftarrow \text{SELEÇÃO ESTADO ALEATÓRIO}()$
- para $t \leftarrow 1$ até ∞
 - $T \leftarrow \text{DECRESCE TEMPERATURA}(t)$
 - Se $T = 0$, retorna estado_atual
 - Para $i \leftarrow 1$ até N
 - $\text{estado_candidato} \leftarrow \text{SELEÇÃO VIZINHO ALEATÓRIO}(\text{estado_atual})$
 - $\Delta E \leftarrow \text{Valor}(\text{estado_candidato}) - \text{Valor}(\text{estado_atual})$
 - Se $\Delta E > 0$
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$
 - Senão
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$ com probabilidade $e^{\Delta E/T}$

T P.ex., se $\Delta E = -2$ e $T=10$

prob(aceitar esse vizinho pior) = $e^{-2/10} = 0.81 = 81\%$

baixo P.ex., se $\Delta E = -20$ e $T=10$

prob(aceitar esse vizinho pior) = $e^{-20/10} = 0.13 = 13\%$

T P.ex., se $\Delta E = -2$ e $T=50$

alto P.ex., se $\Delta E = -20$ e $T=50$

- **Simulated_Annealing()**

- $\text{estado_atual} \leftarrow \text{SELEÇÃO ESTADO ALEATÓRIO}()$
- para $t \leftarrow 1$ até ∞
 - $T \leftarrow \text{DECRESCE TEMPERATURA}(t)$
 - Se $T = 0$, retorna estado_atual
 - Para $i \leftarrow 1$ até N
 - $\text{estado_candidato} \leftarrow \text{SELEÇÃO VIZINHO ALEATÓRIO}(\text{estado_atual})$
 - $\Delta E \leftarrow \text{Valor}(\text{estado_candidato}) - \text{Valor}(\text{estado_atual})$
 - Se $\Delta E > 0$
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$
 - Senão
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$ com probabilidade $e^{\Delta E/T}$

T	P.ex., se $\Delta E = -2$ e $T=10$
baixo	P.ex., se $\Delta E = -20$ e $T=10$
T	P.ex., se $\Delta E = -2$ e $T=50$
alto	P.ex., se $\Delta E = -20$ e $T=50$

prob(aceitar esse vizinho pior) = $e^{-2/10} = 0.81 = 81\%$

prob(aceitar esse vizinho pior) = $e^{-20/10} = 0.13 = 13\%$

prob(aceitar esse vizinho pior) = $e^{-2/50} = 0.96 = 96\%$

- **Simulated_Annealing()**

- $\text{estado_atual} \leftarrow \text{SELEÇÃO ESTADO ALEATÓRIO}()$
- para $t \leftarrow 1$ até ∞
 - $T \leftarrow \text{DECRESCE TEMPERATURA}(t)$
 - Se $T = 0$, retorna estado_atual
 - Para $i \leftarrow 1$ até N
 - $\text{estado_candidato} \leftarrow \text{SELEÇÃO VIZINHO ALEATÓRIO}(\text{estado_atual})$
 - $\Delta E \leftarrow \text{Valor}(\text{estado_candidato}) - \text{Valor}(\text{estado_atual})$
 - Se $\Delta E > 0$
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$
 - Senão
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$ com probabilidade $e^{\Delta E/T}$

T P.ex., se $\Delta E = -2$ e $T=10$

baixo P.ex., se $\Delta E = -20$ e $T=10$

T P.ex., se $\Delta E = -2$ e $T=50$

alto P.ex., se $\Delta E = -20$ e $T=50$

prob(aceitar esse vizinho pior) = $e^{-2/10} = 0.81 = 81\%$

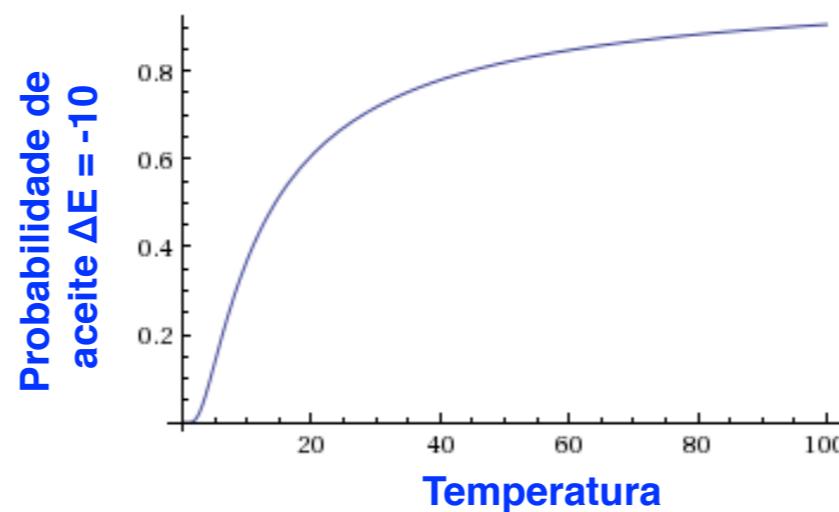
prob(aceitar esse vizinho pior) = $e^{-20/10} = 0.13 = 13\%$

prob(aceitar esse vizinho pior) = $e^{-2/50} = 0.96 = 96\%$

prob(aceitar esse vizinho pior) = $e^{-20/50} = 0.67 = 67\%$

• Simulated Annealing()

- $\text{estado_atual} \leftarrow \text{SELEÇÃO ESTADO ALEATÓRIO}()$
- para $t \leftarrow 1$ até ∞
 - $T \leftarrow \text{DECRESCE TEMPERATURA}(t)$
 - Se $T = 0$, retorna estado_atual
 - Para $i \leftarrow 1$ até N
 - $\text{estado_candidato} \leftarrow \text{SELEÇÃO VIZINHO ALEATÓRIO}(\text{estado_atual})$
 - $\Delta E \leftarrow \text{Valor}(\text{estado_candidato}) - \text{Valor}(\text{estado_atual})$
 - Se $\Delta E > 0$
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$
 - Senão
 - $\text{estado_atual} \leftarrow \text{estado_candidato}$ com probabilidade $e^{\Delta E/T}$



Simulated Annealing (Problema do Caixeiro Viajante)

304 cidades

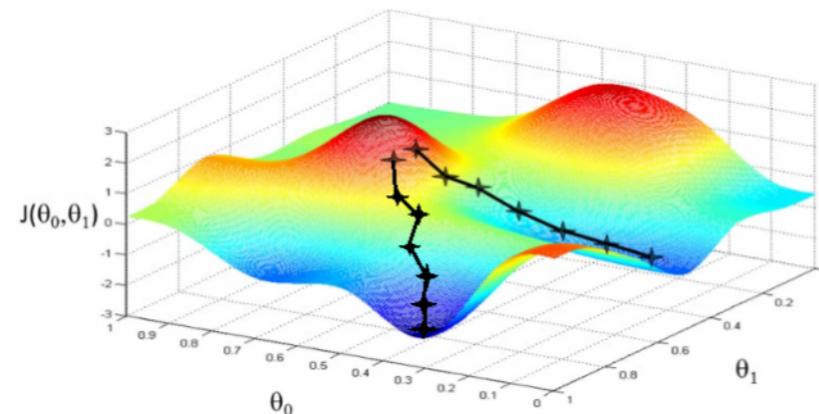
2.3×10^{624} rotas possíveis



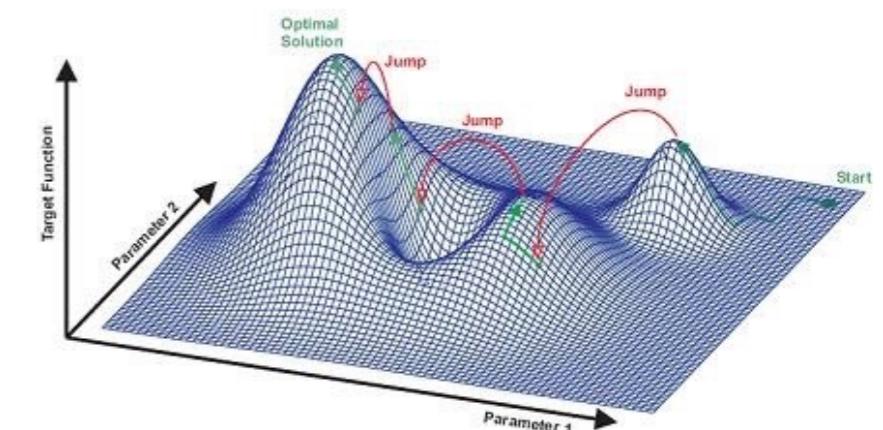
Próximas Aulas - Busca Local

5	3		7	
6		1	9	5
9	8			6
8		6		3
4		8	3	
7		2		6
6			2	8
	4	1	9	
	8		7	9

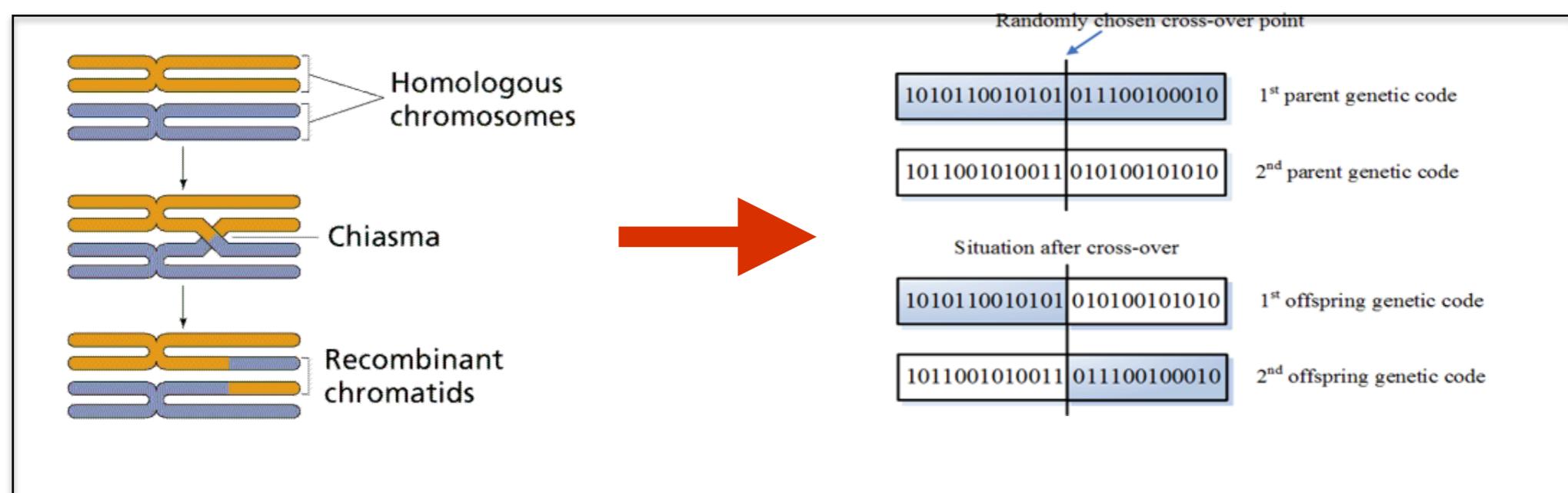
Busca Local em PSR's
de larga escala



Método do gradiente descendente
e métodos de 2^a ordem



Busca em Feixe Local



Algoritmos genéticos