AGENTES INTELIGENTES

2.1 DEFINIÇÃO

"Um agente pode ser visto como um artefato que tem capacidade de "perceber" o ambiente através de sensores e realizar ações neste ambiente por meio de atuadores"

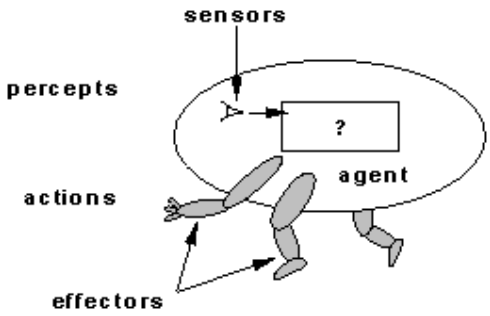


Figura 1 - Agente interagindo com o ambiente por meio de sensores e atuadores (*effectors*).

2.2 COMO OS AGENTES DEVEM AGIR?

Um agente racional é um agente que atua corretamente ⇒ aquele que faz a coisa correta ⇒ leve o agente a atingir sucesso ⇒ **Problema:** *como* e *quando* avaliar o sucesso de um agente?

Medida de Desempenho: determina qual é o sucesso do agente.

- Não existe uma única medida de desempenho para todos os agentes. Ex: uma medida de performance para dirigir um carro pode ser velocidade e/ou consumo de combustível.

- Nós como observadores definimos um padrão do que significa sucesso no ambiente em questão e usamos esse padrão como medida de desempenho para o agente.

  Ex.: aspirador de pó:

1º medida: limpar em tempo hábil;
2º medida: consumo baixo de energia;
3º medida: limpar de forma silenciosa.

Racionalidade		**Omnisciência**
⇓	**x**	⇓
Preocupa-se com o resultado esperado dado o que foi percebido		Saber o resultado de suas ações e poder agir apropriadamente.

Ex.: atravessar a rua, discutido em sala de aula.

A racionalidade de um agente depende de:

- medida de desempenho que define o grau de sucesso;
- seu histórico, tudo que já foi percebido pelo agente (sequência de percepções);
- conhecimento do agente sobre seu ambiente;
- conjunto de ações que ele pode executar.

Assim:

"Um Agente Racional deve executar ações que maximizem seu desempenho, com base na sequência de percepções já percebidas e em seu conhecimento interno"

Um agente faz um **mapeamento** (*mapping*) para realizar as ações: construir uma lista de ações a serem tomadas a partir de cada sequência de percepções ⇒ em muitos casos esta tabela pode se tornar muito grande ou até infinita.

Ex.: agente que dado um número inteiro calcula a raiz quadrada.

Mapeamento Ideal: quando a percepção for um número positivo x, a ação correta é exibir um número positivo z, tal que $z^2 \approx x$.

É claro que esse exemplo não requer o desenvolvimento completo de uma tabela ⇒ um programa simples implementa o mapeamento:

```
function SQRT(x)
        z ← 1.0
        repeat until |z²-x| < 10⁻¹⁵
                z ← z - (z² - x)/(2z)
        end
return z
```

Assim, um exemplo simples poderia gerar uma tabela muito grande.

Agente Autônomo:

- Comportamento baseado na própria experiência: se o agente não tiver experiência (ou pouca) ⇒ agir aleatoriamente.
- Agentes inteligentes artificiais com algum conhecimento inicial e com habilidade/capacidade de aprender.

2.3 A ESTRUTURA DE AGENTES INTELIGENTES

Trabalho da IA é modelar/desenvolver programa agente ⇒ função que implemente o mapeamento das percepções para ações do agente.

Agente = arquitetura + programa

Arquitetura: Recurso computacional necessário para executar o programa (áudio, vídeo, etc...). Em geral, a arquitetura recebe as percepções dos sensores e valida para o programa, executa o programa e transmite para os atuadores (*effectors*), as ações escolhidas pelo programa.

Antes de definir um programa agente, deve-se identificar o **PAGE**[1] (*percept, actions, goal e environment)*: quais serão os possíveis estímulos, ações, metas/objetivos e qual o ambiente ele deve operar.

Exemplos:

Tipo do agente	Percepções	Ação	Objetivos	Ambiente
Tutor de inglês interativo	Palavras escritas	Exercícios, sugestões, correções	Maximizar a pontuação do estudante	Conjunto de estudantes

ESQUELETO DE UM PROGRAMA AGENTE:

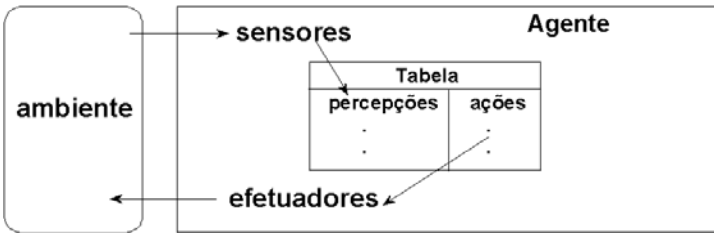
```
function SKELETON-AGENT (percept) returns action
static: memory, the agent's memory of the world

        memory ← UPDATE-MEMORY (memory, percept)
        action ← CHOOSE-BEST-ACTION(memory)
        memory  ← UPDATE-MEMORY(memory,action)
return action
```

Observe que:

- Um agente recebe um estímulo por vez, enquanto que as sequências são guardadas na memória;
- A medida de desempenho não é parte do programa, deve ser aplicada externamente.

AGENTE DIRIGIDO POR TABELAS:

Agente que define uma ação apropriada baseado numa tabela indexada pela sequência de percepções, armazenada na memória.



[1] Na segunda edição do livro, o autor sugere utilizar a notação PEAS (Performance, Environment, Actuators e Sensors).

```
function TABLE-DRIVEN-AGENT (percept) returns action
static: percepts, a sequence, initially empty
        table, a table, indexed by percept sequences,
                          initially fully specified

       append percept to the end of percepts
       action <- LOOKUP(percepts, table)
return action
```

Problemas dessa proposta:

Quanto à tabela:

- Tamanho da tabela: problemas simples podem gerar tabelas muito grandes. Ex: um agente para jogar xadrez teria 35^{100} entradas.

- Tempo de desenvolvimento: nem sempre é possível definir previamente todas as sequências de ações para construir uma tabela. E mesmo que seja possível construí-la, isto poderia levar muito tempo.

Quanto ao Agente:

- Ausência de Autonomia: o agente não tem autonomia pois as ações são derivadas do seu conhecimento interno (e não da sua experiência). Uma troca inesperada no ambiente poderia resultar em falha.

- Tempo de Aprendizagem: se for dado ao agente um mecanismo de aprendizagem para aumentar o grau de autonomia, ele poderá ficar aprendendo "para sempre" sem que todas as entradas da tabela sejam preenchidas.

$\Rightarrow$ Trabalhar com busca em tabela está fora de questão pois problemas simples como por exemplo uma entrada visual de uma câmera, geraria uma tabela muito grande.

Tipos de Programas Agentes:

- Agente baseado em simples reflexo;
- Agente reativo com estado interno;
- Agente baseado em metas/objetivos;
- Agente baseado em utilidade.

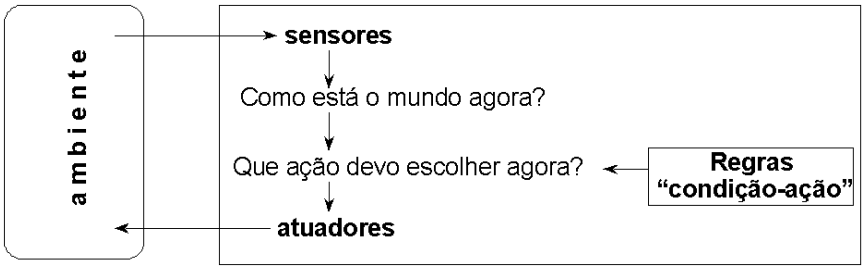
AGENTE BASEADO EM SIMPLES REFLEXO:

Pode-se resumir partes da tabela a partir de ocorrências comuns de associações de entrada e saídas. Considere um agente motorista de táxi: algum processamento é feito na entrada visual para estabelecer a condição "o carro da frente está freiando", isso então gera a ação "inicia a frenagem":

Regras de Condição-Ação:

If *car-in-front-is-breaking*

then *initiate-braking*




```
function SIMPLE-REFLEX-AGENT(percept)  return action
static: rules, a set of condition-action rules

      state ← INTERPRET-INPUT(percept)   //a description of current state
      rule  ← RULE-MATCH (state, rules)  //the first rule
      action ← RULE-ACTION[rule]
return action
```

Observe que:

- INTERPRET-INPUT: Gera uma descrição abstrata do estado a partir do que foi percebido;
- RULE-MATCH: Retorna a primeira regra que "casou" com a descrição do estado.

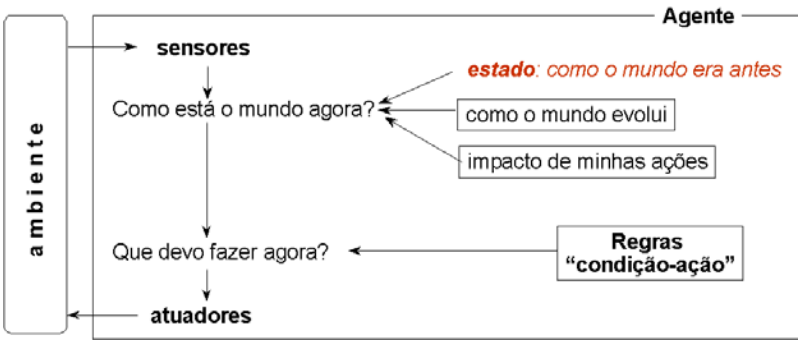
Vantagens e Desvantagens:

- Simples mas limitado: só funciona se a decisão puder ser tomada com base no estado atual ⇒ ambiente completamente acessível.
- Regras *condição-ação* têm representação inteligível, modular e simples;

- Autonomia: não armazena uma sequência de estímulos, sendo assim trabalha somente se a decisão correta puder ser efetuada com base na percepção corrente ⇒ leque pequeno de aplicações.

AGENTE REATIVO COM ESTADO INTERNO:

Considere o problema do carro freiando. Como identificar que a luz de freio do carro da frente acendeu? Para isso, é preciso que o agente mantenha um estado interno para escolher a ação, neste caso o agente deve manter um quadro anterior da câmera de vídeo para detectar quando duas luzes vermelhas acenderem ou apagarem simultaneamente.



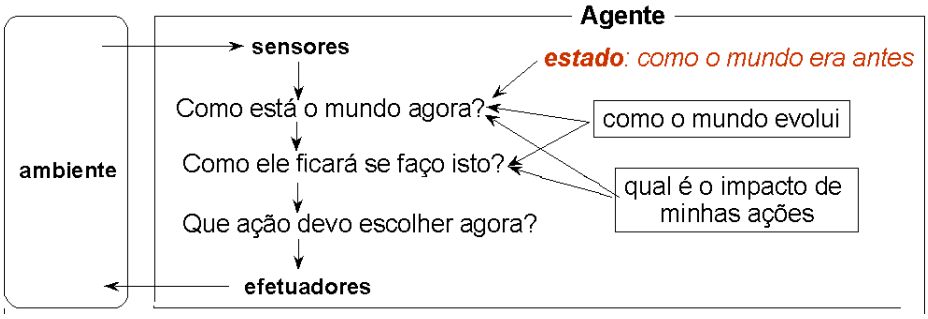

```
function REFLEX-AGENT-WITH-STATE(percept) return action
static: state, a description of the current world state
        rules, a set of condition-action rules
        state ← UPDATE-STATE(state, percept)
        rule  ← RULE-MATCH (state, rules)
        action ← RULE-ACTION[rule]
        state ← UPDATE-STATE(state, action)
return action
```

A atualização do estado deve levar em consideração informações de como o mundo evolui, e como o mundo é alterado pelas ações realizadas pelo agente.

AGENTE BASEADO EM OBJETIVO:

Considere o problema do agente motorista de táxi ao chegar num entroncamento. Qual deve ser a sua ação? Convergir para a direita ou para esquerda? A resposta possivelmente dependerá do seu objetivo, ou seja de onde ele deseja chegar. Para isso o agente deve manter informações sobre os objetivos a serem atendidos.

Vantagem: Adaptável ⇒ não é necessário reescrever as regras.



Busca e Planejamento: subcampos da Inteligência Artificial preocupados em encontrar sequência de ações para atingir o objetivo do agente.

AGENTE BASEADO EM UTILIDADE:

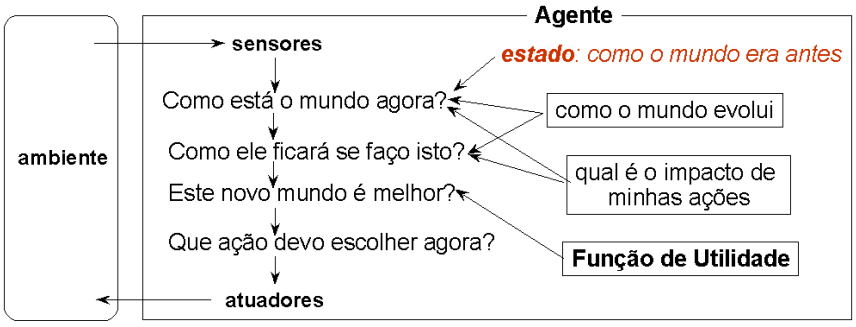
Somente objetivos não são suficientes para gerar um comportamento de qualidade. Considere o problema do agente motorista de táxi: existem várias maneiras de levá-lo ao seu destino, com fatores diferenciados, como custo, tempo, dentre outros ⇒ **Utilidade.**

⇓

Grau de satisfação

⇓

Avaliação de um estado envolvendo rapidez, segurança, confiabilidade, preço, etc...



Usa-se uma **função utilidade** para tomada de decisão em dois tipos de casos:

- Decidir entre metas conflitantes. Ex.: velocidade e segurança.
- Várias metas mas todas sem 100% de probabilidade de sucesso (probabilidade x importância das metas).

2.4 AMBIENTES:

PROPRIEDADES:

Acessível x Inacessível:

Um ambiente é acessível quando os sensores do agente conseguem detectar todos os aspectos relevantes para a escolha da ação (relevante depende da medida de desempenho), ou seja, o agente pode obter informação completa e precisa sobre o estado do ambiente que ele habita. Ambientes acessíveis são convenientes porque não é necessário manter o estado interno sobre o mundo. Exemplo: jogo de 8-peças.

Um ambiente pode ser parcialmente acessível, ou observável devido a imprecisão dos sensores ou por que parte do ambiente é desconhecido para os sensores. Exemplo: aspirador de pó com sensor que indica somente se a posição corrente contém sujeira mas dá informação sobre onde existem sujeiras.

Determinístico x Não-Determinístico:

Um ambiente é determinístico quando o próximo estado do ambiente pode ser completamente determinado pelo estado atual e a ação selecionada pelo agente, ou seja, uma ação tem um efeito garantido, não existindo incerteza quanto ao estado resultante desta ação. Caso contrário, ele é estocástico ou simplesmente não determinístico. Exemplo: motorista de táxi: estocástico, aspirador de pó: determinístico/estocástico (incluindo aparecimento aleatório de sujeiras e mecanismos de sucção também aleatórios)

Episódico x Sequencial:

Um ambiente é episódico quando a experiência do agente é dividida em episódios. Cada episódio consiste em o agente perceber e agir. Cada episódio não depende das ações que ocorreram em episódios anteriores. Ambientes episódicos são mais simples. Exemplo: xadrez e táxi: seqüenciais e aspirador de pó: episódico

Estático x Dinâmico:

Um ambiente é estático quando não muda enquanto o agente está escolhendo a ação a realizar. Semi-estático ou semi-dinâmico: o ambiente não muda enquanto o agente delibera, mas

o desempenho do agente muda. O ambiente estático é mais simples por que o agente não precisa manter controle do mundo enquanto escolhe a ação. Exemplo: motorista de táxi: dinâmico, palavra cruzada: estático e xadrez com relógio: semi-dinâmico.

Discreto x Contínuo:

Um ambiente é discreto quando existe um número fixo de percepções e ações possíveis. Exemplo: motorista de táxi: contínuo e xadrez: discreto

Single Agent x MultiAgent:

Exemplo: palavra cruzada: single agent e xadrez: two agents. Num ambiente multiagent, os agentes podem ser classificados em:

Agentes competitivos: Exemplo: no xadrez o oponente maximiza sua medida de desempenho $\Rightarrow$ minimizar a medida de desempenho do outro agente;

Agentes Cooperativos: Exemplo: motorista de táxi: para evitar colisões a medida de desempenho de todos os agentes é maximizada.

Qual a classificação do pior ambiente?

SIMULADOR DE AMBIENTE:

Os principais benefícios da simulação de ambientes está no fato de ser mais simples e mais conveniente simular o ambiente. Além disso, permite testes prévios, evita riscos, etc...

O programa simulador de ambiente deve:

- Receber os agentes como entrada;
- Fornecer repetidamente a cada um deles as percepções corretas e recebe as ações;
- Atualizar os dados do ambiente em função dessas ações e de outros processos;
- Ser definido por um estado inicial e uma função de atualização ;
- Refletir a realidade .

Programa básico de simulação de ambiente: para cada agente dado, e seus estímulos, retorna as ações e atualiza o ambiente:

```
procedure RUN-ENVIRONMENT(state, UPDATE-FN, agents, termination)
repeat
       // state, the initial state of the environment
       for each agent in agents do
          PERCEPT[agent] ← GET-PERCEPT (agents, state)
       end

       for each agent in agents do
          ACTION[agent] ←PROGRAM[agent](PERCEPT[agent])
       end

       // UPDATE-FN, function to modify the environment
       state ← UPDATE-FN(actions, agents,state)
until termination(state)
```

Programa simulador de ambiente que considera medidas de desempenho de cada agente:

```
function RUN-EVAL-ENVIRONMENT(state, UPDATE-FN, agents, termination, PERFOR-FN)
                                  return scores
local variables: scores, a vector the same size as agents, all 0
       repeat
          for each agent in agents do
              PERCEPT[agent] ← GET-PERCEPT (agents, state)
          end

          for each agent in agents do
             ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
          end

          state ← UPDATE-FN(actions, agents,state)
          scores ← PERFORMANCE-FN(scores, agents,state)
       until termination(state)
return scores
```