INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ÁREA DEPARTAMENTAMENTAL DE ENGENHARIA DE ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

Licenciatura em Engenharia de Redes de Comunicação e Multimédia

Inteligência Artificial para Sistemas Autónomos

RELATÓRIO FINAL

2016-2017

ALUNOS:

39085, André Rodrigues

41943, André Santos

40644, Bruno Tavares

ÍNDICE

ÍNDICE FIGURAS

ÍNDICE TABELAS

INTRODUÇÃO

A inteligência artificial é um conceito muito amplo que engloba diversos métodos, algoritmos e tecnologias cujo objectivo geral é reproduzir por meio de máquinas, nomeadamente computadores, actividades humanas qualificadas como inteligentes.

Essas actividades podem ser o tratamento de informações simbólicas não numéricas utilizando heurísticas, a resolução geral de problemas, o reconhecimento de formas, o reconhecimento automático da linguagem, entre outras.

De facto, a inteligência artificial é em muito semelhante à inteligência humana, pois um dos seus domínios é o desenvolvimento de sistemas com base em representação de conhecimento e modelação de raciocínio.

Tal como a inteligência humana, também a inteligência artificial utiliza o raciocínio lógico, a memória, o conhecimento e a linguagem, mas falta-lhe uma componente humana, que marca a diferença fundamental entre as máquinas e o ser humano - a emoção.

Assim podemos definir um agente inteligente como um componente de hardware e/ou software capaz de executar determinadas tarefas, com base num raciocínio e de acordo com o modelo do seu mundo e conhecimento específico do problema a ser solucionado. Deve ser capaz de tomar decisões autonomamente ao analisar o problema, ponderar soluções viáveis de acordo com os recursos disponíveis (tempo, memória, combustível…) e planear acções que levem à solução do problema.

Esta idealização de agente possuidor de uma inteligência semelhante à de um humano ainda se encontra um pouco longe da realidade. Embora um computador actual seja capaz de processar milhões de cálculos em segundos (ao contrário dos seres humanos), tarefas tão simples como reconhecer um rosto ou interpretar uma piada que até uma criança seria capaz de entender tornam-se algo complexas para uma máquina, enquanto os seres humanos o fazem trivialmente.

1 Arquitectura de Agentes Inteligentes

1.1 Requisitos

1.1.1 Agente

Um agente é uma entidade capaz de efectuar acções autonomamente para cumprir o objectivo/propósito com que foi modelado. O agente existe num ambiente com o qual pode interagir através actuadores e também percepcionar através dos sensores.

Para que seja considerado inteligente um agente deve ser:

- **Autónomo**: adaptar-se e agir sem a intervenção de humanos ou outros agentes
- **Reactivo**: responder reactivamente às alterações do ambiente
- **Pró-actividade**: tomar a iniciativa para agir tendo em vista a concretização dos seus objectivos
- **Sociabilidade**: interagir com outros agentes no mesmo ambiente

Um agente, através da sua percepção do ambiente, pode manter um modelo do mundo interno que permite a implementação de agentes mais complexos capazes de aprender com a experiência, efectuar simulações para definir um plano de acção ou até gerar uma política comportamental (processos de decisão de Markov).

1.1.2 Percepção

A percepção do agente consiste na informação sobre o ambiente que o agente interpreta através dos seus sensores. A percepção depois de processada produz estímulos que podem desencadear ou não acções no agente.

1.1.3 Acção

Uma acção é um comando/instrução explícita que produz uma acção ao nível dos actuadores do agente. Uma acção pode ser por exemplo um comando para fazer o agente deslocar para a frente ou para trás, rodar, saltar, etc.

1.1.4 Comportamento

Todo o agente foi modelado para cumprir uma função/objectivo em específico. Os comportamentos de um agente são basicamente tarefas (sub-objectivos) que este tem de realizar para cumprir o seu objectivo.

1.2 Conceitos

- **Inteligência**: é o conjunto que forma todas as características intelectuais de um indivíduo, ou seja, a faculdade de conhecer, compreender, raciocinar, pensar e interpretar. Em cada circunstância, escolher a melhor acção para atingir os objectivos.
- **Cognição**: Processo através do qual obtemos conhecimento e o processamos. É caracterizada como uma propriedade global dos organismos, expressa através da capacidade de realizar a acção adequada dadas as condições do ambiente
- **Racionalidade**: Comportamento gera resposta. É a capacidade de agir no sentido de conseguir o melhor resultado possível perante os objectivos que se pretende atingir
    - **Limitada**: A capacidade de gerar um comportamento suficientemente bom dada a informação e os recursos computacionais disponíveis.
    - **Ilimitada**: A capacidade de gerar o melhor comportamento possível dada a informação disponível.
- **Tipos e arquitecturas de agente**:
    - **Reactivas (comportamentais)** – ênfase no acoplamento com o ambiente.
    - **Deliberativas (cognitivas)** – ênfase nas representações internas do mundo.
    - **Híbridas** – integração de abordagens reactivas e deliberativas.

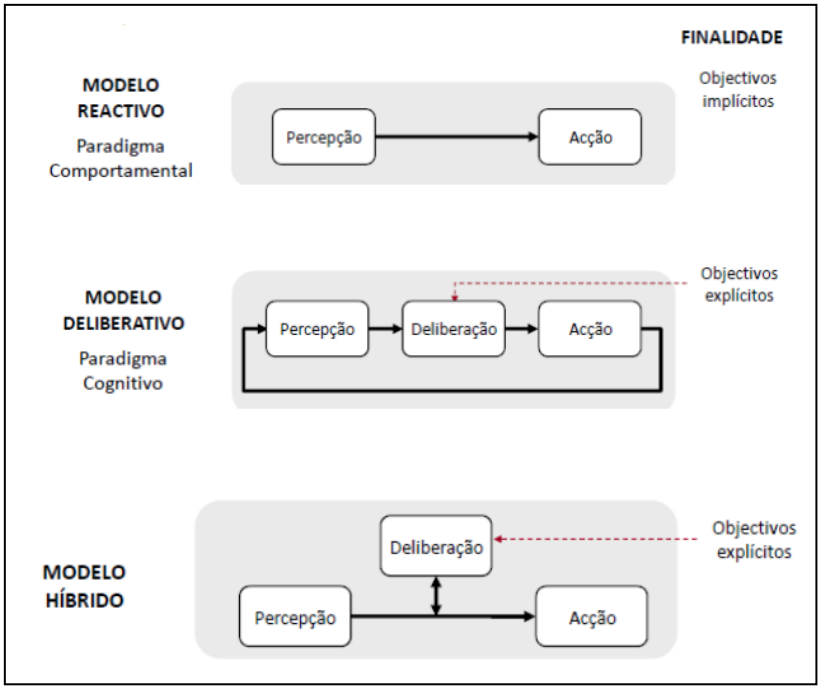


Figura 1 - Representação interna de uma arquitectura de agente

2 Arquitectura Reactiva

Os agentes reactivos são agentes simples que baseiam a sua operação em regras estímulo reposta para reagirem a estímulos do ambiente. Uma das principais características é serem rápidos a reagir (não é necessária uma planificação) e por isso são indicados para utilização em ambientes de operação em tempo real.

As acções deste tipo de agente são o resultado da activação dos comportamentos pré-definidos que correspondem ao mapeamento da percepção dos sensores do agente a um conjunto de acções ao nível dos actuadores, para cumprir uma determinada tarefa ou objectivo.

2.1 Trabalho prático 1

2.1.1 Ambiente

O Ambiente proporciona ao Personagem estímulos para o seu comportamento, na forma de eventos.

O método gerarEvento() gera um evento em função do comando inserido pelo utilizador na consola do jogo.

Os eventos que ocorrem no ambiente, representados pela classe EventoAmb são apresentados ao Personagem na forma de Estímulo.

2.1.2 Personagem

Cada objecto *Personagem* contém uma referência para um *Ambiente*, que corresponde ao "mundo" com que a personagem vai interagir. O *Personagem* é a modelação do agente.

2.1.3 Comportamento

O comportamento da *Personagem* é baseado numa arquitectura reactiva simples, o que significa que todas as decisões tomadas pelo agente dependem directamente da sua percepção do ambiente em que se encontra.

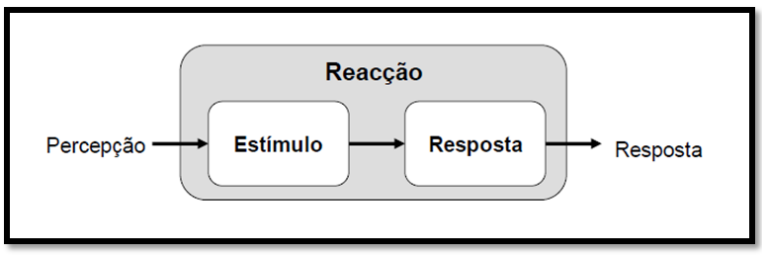


Figura 2 - Reacção

2.1.4 Desenvolvimento da Máquina de estados

Uma máquina de estados pode ter vários estados e encontra-se sempre num deles, o estado actual. Para que haja uma transição de estado é necessário que se verifique uma certa condição.

Nesta implementação cada *Estado* contém um mapa do tipo (*Estimulo*: *Estado*) que define as transições de estado possíveis a partir do estado actual, em função de um *Estimulo* (condição).

Assim o método *processar*(*Estimulo*) da classe *Estado* retorna o próximo *Estado* caso no mapa de transições exista uma chave igual ao *Estimulo* passado como parâmetro, ou *null* caso não exista um *Estado* em resposta ao *Estimulo*.

O método *processar*(*Estimulo*) da classe *MaquinaEstados* chama o método *processar(Estimulo)* do *Estado* actual e se o valor retornado for diferente de *null*, então esse é o próximo *Estado* da máquina.

2.1.5 Implementação dos comportamentos reactivos

A resposta do personagem a um *Estímulo* é vista como uma *Acção*.

Uma *Reacção* consiste num par *Estímulo – Acção*.

Uma vez que para cada *Estado* existente a resposta a um *Estimulo* do ambiente pode ser diferente, é necessário definir para cada *Estado* qual o conjunto de *Reacções* que caracterizam o comportamento do agente.

Para isso existe na classe *ComportMaqEstados* um mapa que para cada *Estado* mantém um objecto do tipo *ComportHierarq* cuja função é guardar as *Reacções* do agente para o *Estado* correspondente.

2.1.6 Definição dos estados e transições

É nesta classe que é definida sucintamente o comportamento da *Personagem*.

Aqui são instanciados os *Estados* da máquina assim como são definidas todas as transições entre esses *Estados*, através do método *trans(Estimulo,Estado)*.

Para cada *Estado* da máquina existe uma classe com o nome correspondente. Estas derivam de *ComportHierarq* e contêm as *Reacções* do agente específicas para o *Estado* a que correspondem.

Assim, utilizando o método *comport(Estado,Comportamento)* é possível fazer corresponder a cada *Estado* da máquina um conjunto de *Reacções*, ou seja, qual a *Acçao* do agente *Estímulos* específicos do *Ambiente*.

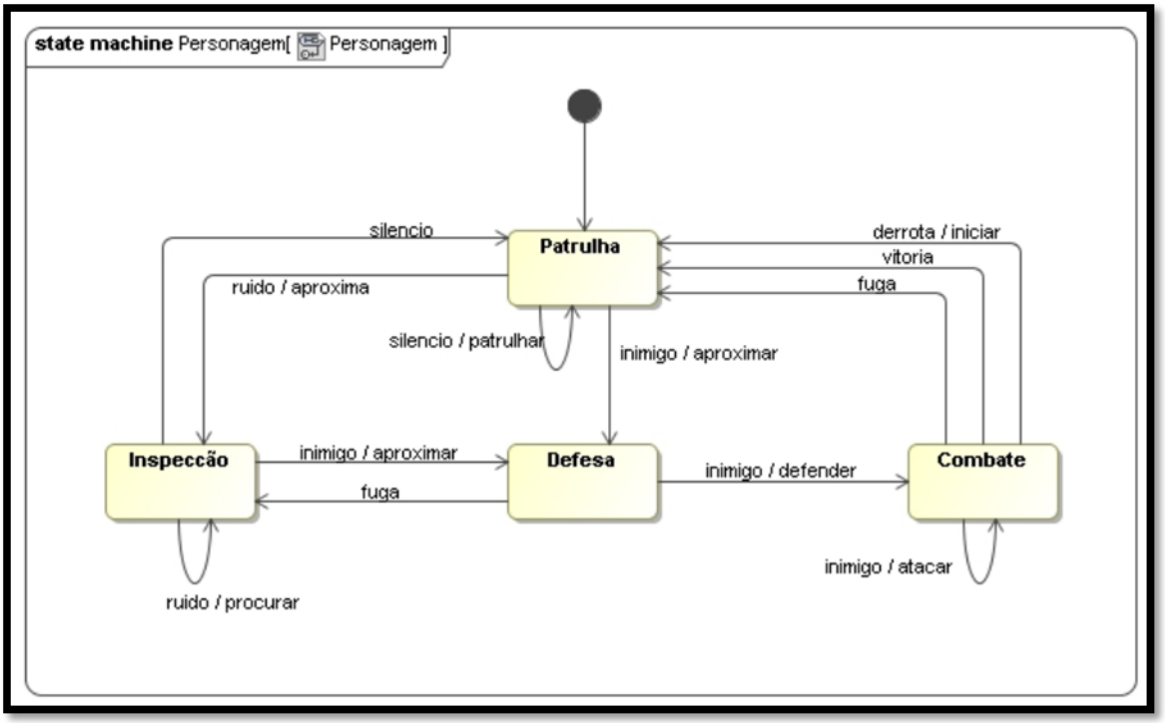


Figura 3 - Esquema de comportamentos da personagem

Inimigo - Existe um inimigo perto da personagem. Se a personagem estiver no estado:

- Patrulha - Irá aproximar-se dele, entrando em estado Defesa.
- Defesa - Irá combater o inimigo.
- Combate - Continua a combater.
- Inspecção - Irá aproximar-se dele, entrando em estado Defesa.

Ruído - Se a personagem estiver em estado:

- Patrulha - Irá aproximar-se do ruído, entrando em estado Inspecção.
- Inspecção - Irá procurar o inimigo.

Silêncio - Se a personagem estiver em estado:

- Patrulha - Mantém-se a patrulhar.
- Inspecção - Entra em estado Patrulha.

Fuga - Se a personagem estiver em estado:

- Defesa - Entra em estado Inspecção.
- Combate - Começa a patrulhar.

Vitória - A personagem venceu o inimigo.

Derrota - A personagem foi derrotada pelo inimigo.

Terminar - Termina o jogo.

2.2 Trabalho prático 3

2.2.1 Lógica do Agente

A lógica por trás do funcionamento do agente é a seguinte:

1. **Percepcionar**: obter informação do ambiente através dos sensores do agente
2. **Processar(Percepção)**: Este processamento ocorre num Controlo que é passado no construtor do Agente, e que permite reutilizar o Agente para testar diversos controlos.
3. **Actuar(Acção)**: Caso a activação do comportamento com a percepção resulte numa acção, esta é transmitida aos actuadores do agente.

2.2.2 Selecção da acção

A activação das reacções pode originar várias respostas, pelo que é necessário seleccionar uma:

- **Simples:** um estímulo gera uma resposta
- **Em paralelo:** as respostas não interferem umas com as outras e são executadas em paralelo, dentro dos limites da arquitectura.
- **Hierarquia:** as respostas estão organizadas numa hierarquia fixa de supressão.
- **Prioridade:** as respostas são seleccionadas de acordo com uma prioridade associada que pode variar ao longo da execução.
- **Fusão**: as respostas são combinadas numa única resposta (soma vectorial por exemplo)

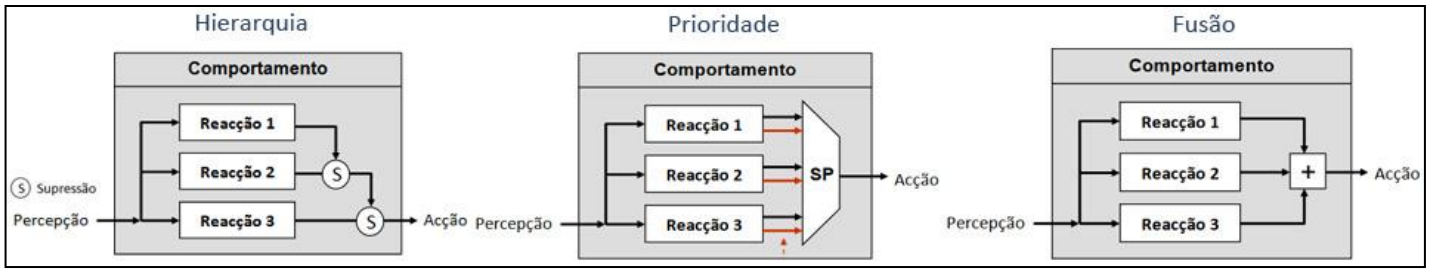


Figura 4 - Selecção da acção em Hierarquia, Prioridade e Fusão

2.2.3 Arquitectura de subsunção

Neste tipo de arquitectura os comportamentos inteligentes e mais complexos são decompostos em módulos comportamentais mais simples.

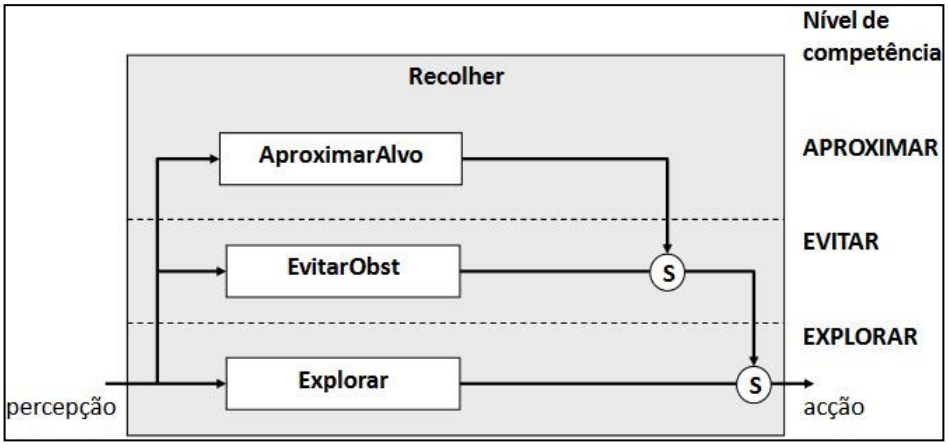
A organização é feita por níveis/camadas de competência, com cada nível a corresponder a um comportamento. As camadas superiores podem suprimir o comportamento gerado pelos níveis abaixo. Enquanto que as camadas inferiores não têm conhecimento acerca dos superiores, estas últimas podem usar as saídas das camadas inferiores como entradas.

Ou seja, as camadas superiores controlam as inferiores, resultando numa hierarquia de comportamentos.

- **Inibição**: desactivação de comunicação entre módulos
- **Supressão**: desactivação do comportamento
- **Reinício**: Reposição do estado inicial do comportamento

2.2.4 Controlo Reactivo

Comportamento Composto - Recolher(Hierarquia):



O controlo reactivo em primeiro lugar vai activar o único comportamento que lhe foi passado como parâmetro, *Recolher*.

O *comportamento Recolher* é um tipo de comportamento composto (*ComportamentoComp*), ou seja, é composto por outros sub-comportamentos modelados para realizar o comportamento pretendido.

A activação de um comportamento composto produz várias respostas, geradas pela activação dos sub-comportamentos que o compõem.

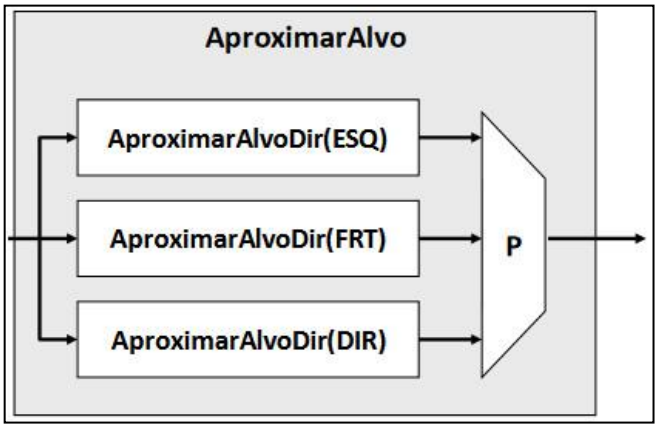
Como o *comportamento Recolher* é do tipo *Hierarquia*, é necessário manter a ordem de activação dos comportamentos, e por isso a estrutura do contentor dos comportamentos

é, de acordo com as imagens acima, a seguinte: [*AproximarAlvo, Evitar, Contornar, Explorar*].

O método *seleccionar_resposta* vai retornar a primeira resposta gerada uma vez que os comportamentos do topo da hierarquia são activados primeiro.

O *comportamento Explorar* não é do tipo *Reacção* como os outros porque não necessita de um estímulo para gerar uma resposta, é sempre gerada uma *Resposta* quando este comportamento é activo, pelo que podemos dizer que é o comportamento por defeito.

Comportamento Composto - AproximarAlvo(Prioridade):



Já o comportamento *AproximarAlvo* também é composto (*ComportamentoComp*), no entanto do tipo *Prioridade*. O estímulo para esta *Reacção* é a distância ao alvo e a prioridade é inversamente proporcional á distância ao alvo. A *Resposta* gerada com maior prioridade será a primeira a ser accionada.

2.2.5 Conceitos

- ***Comportamento*:** interface que disponibiliza o método *activar(Percepcao)*
- ***ComportamentoComp***: possui várias reacções que serão activadas em função da *Percepção*. A selecção da resposta é feita pelo *seleccionar_resposta* que é abstracto para esta classe, mas está implementado noutras como por exemplo *Hierarquia* e *Prioridade*.
- ***Reacção***: se através da percepção do ambiente, obtida pelos sensores do agente, uma reacção der origem a um estímulo (detectar_estimulo) então gera uma resposta (*gerar_resposta*).
- ***Resposta***: consiste numa instrução explícita a efectuar ao nível dos actuadores do agente (*Acção*) e num valor de prioridade, que por defeito é igual a 0

3 Procura em Espaço de Estados

A procura em espaço de estados é uma das técnicas mais utilizadas na resolução de problemas em inteligência artificial. A ideia consiste em modelar um agente capaz de executar um conjunto de acções que modificam o estado actual do seu "mundo".

Assim dados um estado inicial (representação do mundo do agente), um conjunto de acções que o agente é capaz de executar (operadores) e um estado objectivo (estado que se pretende atingir), a solução do problema consiste numa sequência de acções que, quando executadas pelo agente, transformam o estado inicial em estado objectivo.

3.1 Requisitos

3.1.1 Espaço de estados

O conjunto de todos os estados acessíveis a partir de um estado inicial chama-se espaço de estados.

O espaço de estados é definido por um conjunto de *S* estados e *A* acções que permitem a transição de um estado para outro. Também pode ser interpretado como um grafo em que os nós são estados e os arcos são acções.

3.1.2 Representação de estados

Os estados são representados por estruturas em que cada componente corresponde a um atributo do estado representado. No problema particular do puzzle os estados são as configurações possíveis do puzzle e um estado é representado por um *array* bidimensional com a informação do puzzle.

3.1.3 Representação de acções

As acções são representadas por operadores, que quando aplicados sobre um determinado estado podem, ou não, provocar uma transição de estado. O conjunto de operadores do problema do puzzle são os movimentos da peça não numerada nas quatro direcções possíveis: (CIMA, BAIXO, ESQUERDA, DIREITA).

3.2 Procura

3.2.1 O problema da procura

Um problema de procura em espaço de estados pode ser definido através de três componentes:

- Espaço de estados (composto pelos conjuntos *S* (estados) e *A* (operadores))
- Estado inicial (denotado por um estado particular S0 pertence a S)
- Estado objectivo (estado final pretendido)

A solução para um problema de procura em espaço de estados consiste numa sequência de acções que leva o estado inicial ao estado objectivo, no espaço de estados do problema.

3.2.2 Estratégias de procura

Os vários tipos de procura são avaliados de acordo com os seguintes critérios:

- **Completo**: o algoritmo encontra sempre a solução, se ela existir
- **Complexidade** temporal: número de nós gerados
- **Complexidade** espacial: número máximo de nós na memória
- **Óptimo**: encontra a solução óptima

Complexidade temporal e espacial:

- **b**: factor de ramificação da árvore (número máximo de sucessores de qualquer nó)
- **d**: profundidade do nó objectivo menos profundo
- **m**: comprimento máximo de qualquer caminho no espaço de estados (pode ser infinito)

3.2.3 Estratégias de procura não-informada

As estratégias de procura não informada usam apenas a informação disponível na definição do problema, apenas geram sucessores e verificam se o estado objectivo foi atingido.

Consiste em políticas que sistematizam o comportamento de um algoritmo de procura, sem levar em conta a qualidade da solução encontrada. Distinguem-se entre si pela ordem em que os nós são expandidos.

3.3 Procura em Largura

Neste tipo de procura o estado inicial (nível 0) é expandido primeiro, e os seus sucessores posicionados no nível 1 da árvore de busca. De seguida cada um dos estados do nível 1 é expandido, os seus sucessores posicionados no nível 2 e assim por diante, de maneira que todos os estados num nível *n* são expandidos antes daqueles no nível *n+1*

O algoritmo de busca em largura é obtido simplesmente sistematizando a ordem de inserção e remoção de estados na fronteira de exploração, que se comporta como uma fila, uma *FIFO* (*first in first out*).

Completa	Óptima	Tempo	Espaço
Sim (se b é finito)	**Sim** (se todas as acções tiverem o mesmo custo)	O((b^d)+ 1)	O((b^d)+1)(todos os nós são mantidos em memória)

Tabela 1 - Procura em Largura

3.4 Procura em Profundidade

Na busca em profundidade o primeiro nó a ser expandido é expandido sucessivamente até ao nível mais profundo da árvore de busca, até atingir um nó que já foi expandido anteriormente ou que não pode ser mais expandido. Nesse caso é expandido o próximo estado ainda não expandido, posicionado mais á esquerda no nível mais profundo da árvore de procura.

Isto torna a procura em profundidade altamente sensível à aplicação dos operadores, por exemplo: um determinado operador pode "levar" a árvore de procura por um ramo sem solução enquanto outro pode levar imediatamente á solução.

Como na procura em profundidade, o algoritmo é obtido pela sistematização da ordem de inserção e remoção de estados na fronteira de exploração. Em vez de uma fila é usada uma pilha, uma lista LIFO (*last in first out*).

Completa	Óptima	Tempo	Espaço
Não (falha em espaços com profundidade infinita ou com loops) **Sim** (se modificada para evitar espaços repetidos, é completa em espaços finitos)	**Não**	O(b^m)(mau quando *m* é muito maior que *d,* mas se existirem muitas soluções pode ser mais eficiente que a largura)	O(bm)(espaço linear)

Tabela 2 - Procura em Profundidade

3.5 Procura em Profundidade Iterativa

Combina o melhor da busca em profundidade e da busca em largura, respectivamente a pouca necessidade de memória e a capacidade de examinar todo o espaço de estados encontrando a solução óptima.

Na primeira iteração, a árvore é gerada utilizando a busca em profundidade limitada com limite = 1. Se a solução não for encontrada, inicia-se a segunda iteração: a árvore anterior é descartada e uma nova construída através da busca em profundidade limitada com limite = 2, e assim sucessivamente até que a solução seja encontrada ou toda a árvore for gerada.

Completa	Óptima	Tempo	Espaço
Sim	**Sim** (se todas as acções tiverem o mesmo custo)	O(b^l)	O(bl)

Tabela 3 - Procura em Profundidade Iterativa

3.6 Estratégias de Procura Heurística

Função de custo f(n): as estratégias de procura heurística utilizam esta função para a avaliação do custo da solução, através de n. Esta função é utilizada para ordenar os estados na fronteira de exploração.

- f(n) >0

Custo do caminho g(n): se as acções tiverem um custo, numa árvore de procura, todo o estado *s* está associado a um caminho que o leva do estado inicial s0 até *s*. Então podemos definir o custo de um estado numa árvore de procura como o caminho do custo que leva até ele (custo acumulado).

Função de heurística h(n): é uma função que estima o custo mínimo de um caminho desde o estado *n* até ao estado objectivo. É necessário conhecimento acerca do domínio do problema para gerar a procura.

Esta função é independente do percurso até *n*, depende apenas do estado associado a *n* e do estado objectivo.

Seja **h*(n)** uma função que calcula o custo mínimo exacto de um caminho que leva a um terminado estado objectivo (na prática, com esta função não seria necessário um algoritmo de procura):

- h(n) = 0, se *n* for estado objectivo
- 0<=h(n)<= h*(n)

Estas propriedades garantem que a estimativa dada pela função heurística é admissível, ou seja, nunca sobrestima o custo real de uma solução. Por outras palavras, a estimativa é sempre inferior ou igual ao custo efectivo mínimo.

3.7 Procura de Custo Uniforme

Neste tipo de procura sempre que um estado é expandido, é associado um custo a cada um dos seus sucessores. O algoritmo progride ao expandir sempre o estado de menor custo, até que um estado objectivo seja encontrado.

O algoritmo de procura é obtido ao ordenar a fronteira de exploração por custo, implementado através do uso uma fila de prioridades ascendente, ordenada em função de:

- f(n) = g(n)

Completa	Óptima	Tempo	Espaço
Sim	Sim	O(b^(C*/e))	O(b^(C*/e))

Tabela 4 - Procura Custo Uniforme

3.8 Procura Sôfrega

Esta tipo de procura é semelhante á procura de custo uniforme com a única diferença na função de avaliação de custo:

- f(n)=h(n)

A fronteira de exploração é ordenada com base no custo da função heurística (estimativa de custo).

3.9 Procura A*

O algoritmo de procura A*combina as duas procuras anteriores combinando a função de custo acumulado e a função de heurística:

- f(n) = g(n) + h(n)

O método de procura A* é completo e óptimo se a heurística for consistente.

3.10 Heurística consistente

Uma heurística admissível pode não ser consistente. Uma heurística consistente é obrigatoriamente admissível.

Se a função **h(n)** for consistente, os valores de **f(n)** nunca diminuem ao longo de um caminho.

Para uma heurística consistente:

- Sempre que é expandido um estado, o percurso desse estado é óptimo

- São expandidos todos os estados com **f(n) < C***(custo da solução óptima)
- Podem ser expandidos estados com **f(n) = C*** antes do estado objectivo

Para garantir a heurística consistente, é mantido um mapa Estado-Custo. Sempre que um estado é explorado, se o custo do caminho que levou até ele for menor do que o encontrado no mapa, este é substituído no mapa e adicionado à fronteira de exploração.

Assim garantimos que para qualquer estado, é sempre mantido o caminho de menor custo, minimizando assim o custo total da solução.

3.11 Análise de Desempenho

PUZZLE A	Procura em Largura	Procura em Profundidade	Procura Prof. Iterativa	Procura Custo	Procura AA	Procura Sôfrega
Custo	14	34 366	14	14	14	18
Número de nós expandidos	14 172	146 364	29 300	12 640	332	340
Número de nós na fronteira de exploração	10 630	109 774	21 976	9 481	250	256

Tabela 5 - Análise Desempenho Puzzle A

PUZZLE B	Procura em Largura	Procura em Profundidade	Procura Prof. Iterativa	Procura Custo	Procura AA	Procura Sôfrega
Custo	26	60 178	26	26	26	62
Número de nós expandidos	654 748	296 468	5 005 556	653 576	9 760	1 100
Número de nós na fronteira de exploração	491 062	222 352	3 754 168	490 183	7 321	826

Tabela 6 - Análise Desempenho Puzzle B

4 Arquitectura de Agentes Deliberativos

Os agentes deliberativos são mais complexos do que os agentes reactivos uma vez que não se limitam a reagir a estímulos do ambiente, estes agentes são capazes de tomar decisões com base em raciocínio lógico.

Este tipo de raciocínio é baseado em manipulação simbólica e só é possível devido ao facto de estes de agentes conterem uma representação simbólica interna do ambiente em que "existem" (modelo do mundo). A lógica do agente deliberativo é a seguinte:

1. **Assimilar(percepção)**: actualizar/inicializar o modelo interno do mundo com base na percepção do agente.
2. **Reconsiderar**: avaliar se é ou não necessário gerar um novo plano de acção para o agente (se não existirem objectivos, se não existir plano pendente ou se o modelo do mundo tiver sido alterado)
    - **Deliberar**: definir os estados objectivos para o planeamento
    - **Planear**: gerar um plano para guiar as acções do agente. Podem ser usados diversos mecanismos de raciocínio, podendo ou não encontrar solução óptima. Este planeamento requer grande processamento, muita memória e tempo para processar, pelo que não é adequado em ambientes muito dinâmicos
3. **Executar**: O plano é um conjunto de acções que levam o agente ao estado objectivo. O método executar vai buscar a próxima acção do plano. Caso não exista, o plano termina.

4.1 Conceitos

- ***ModeloMundo*:** é a representação interna do mundo do agente. Consiste numa estrutura de dados elementos <Coordenadas, Valor> e no estado actual do agente. As coordenadas correspondem a uma posição no ambiente enquanto que o estado corresponde ao valor dessa posição e pode ser do tipo "alvo","obst" ou "vazio".
- ***PlaneadorPEE*:** é responsável por construir o plano. Utiliza um mecanismo de procura em espaço de estados para gerar um plano a partir do modelo do mundo, estado inicial, e os objectivos pretendidos.
- **Plano:** é um conjunto de operadores/passos de solução, que levam ao objectivo planeado. O primeiro passo de solução não possui operador, só o estado inicial do agente.
- ***ProblemaPlan*:** é a modelação do problema para poder usar o mecanismo de procura em espaço de estados. Necessita de definir uma função de objectivo e de heurística. É instanciado com um estado inicial, estado final e os operadores do problema.
- ***OperadorMover*:** é a modelação do operador para poder usar o mecanismo de procura em espaço de estados. Cada operador possuí um ângulo de direcção para o qual o agente se pode deslocar, uma Acção para movimento do agente nesse

angulo, e também o modelo do mundo. O operador só é aplicável se a nova posição gerada pela aplicação do operador, na direcção interna que este possuí, corresponder a uma posição com "alvo" ou "vazia".

- **Estado:** neste problema em particular corresponde a cada uma das posições (coordenadas) que podem ser ocupadas no ambiente.

4.2 Processos de decisão de Markov

Um processo de decisão de Markov é uma forma de modelar processos em que as transições entre estados são probabilísticas.

Em qualquer momento é possível observar em que estado o processo está e é possível interferir no processo periodicamente executando acções.

Cada acção tem uma recompensa (ou custo), que depende do estado em que o processo se encontra. Alternativamente podem definir recompensas por estado apenas, sem que estas dependam da acção executada.

São chamados processos de Markov porque os processos modelados obedecem à propriedade de Markov: o efeito de uma acção em um estado depende apenas da acção e do estado actual do sistema e não do percurso que levou o processo até esse estado. Ou seja, a previsão dos estados seguintes só depende do estado presente.

São chamados de processos de decisão porque modelam a possibilidade de um agente (ou "tomador de decisões") interferir periodicamente no sistema executando acções.

- **S:** espaço de estados do mundo
- **A(s):** conjunto de acções possíveis no estado s que pertence a S
- **T(s, a, s') –** probabilidade de transição de s para s' através de a
- **R(s, a, s') –** retorno esperado na transição de s para s' através de a
- **y –** taxa de desconto para recompensas diferidas no tempo [0,1]

4.2.1 Utilidade

Como o problema de decisão é sequencial, a função de utilidade depende de uma sequência de estados (histórico de evolução h). Em cada estado s o agente recebe uma recompensa R(s, a, s') que pode ser positiva ou negativa.

A utilidade de um histórico de ambiente pode ser definida com a soma das recompensas adquiridas.

- **Recompensas aditivas** - Uh([s0, s1, …]) = R(s0) + R(s1) + …
- **Recompensas descontadas no tempo** - Uh([s0, s1, …]) = R(s0) + yR(s1) +y^2R(s1) …

4.2.2 Política

É uma forma de representar o comportamento do agente. Define qual a acção a realizar para qualquer estado (estratégia de acção).

Se o agente tiver uma política completa, independentemente do estado resultante das suas acções, saberá qual o estado seguinte.

- **Política determinista:** S -> A(s)
- **Política não determinista:** S x A(s) -> [0,1]

Então podemos dizer que a qualidade de uma política é medida pela utilidade esperada de todos os históricos de evolução possíveis de serem gerados por essa política.

Política óptima π*(s) - maximiza a utilidade esperada:

- A política é uma lista que contém, para cada estado s, a respectiva recomendação de acção

A função de utilidade mede a recompensa a longo prazo de uma política

Utilidade de estado para uma política π

$$U^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^{\pi}(s')]$$

Política óptima π^*

$$\pi^*(s) = \arg\max_{a} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U(s')]$$

Utilidade de estado para a política óptima π^*

$$U^{\pi^*}(s) = \max_{a} \sum T(s,a,s')[R(s,a,s') + \gamma U^{\pi}(s')]$$

Figura 5 - Fórmulas de Utilidade-Política, Política Óptima e Utilidade-Política Óptima

5 Aprendizagem por Reforço

5.1 Aprendizagem Automática

Aprendizagem: Melhoria de desempenho, para uma dada tarefa, com a experiência.

- Melhorar o desempenho para uma dada **tarefa *T***
- Com base numa medida de **desempenho *D***
- Com base na **experiência *E***

Exemplos:

Aprender a jogar xadrez

•***T***: jogar xadrez

•***D***: Percentagem de jogos ganhos

•***E***: Jogos realizados

Aprender a reconhecer escrita manual

•***T***: reconhecer e classificar caracteres escritos manualmente

•***D***: Percentagem de caracteres reconhecidos correctamente

•***E***: Conjunto de exemplos de caracteres e respectiva classificação

Aprender a conduzir um veículo

•***T***: conduzir com base na informação proveniente de câmaras de vídeo

•***D***: Distância média percorrida sem erros

•***E***: Sequências de imagens e de comandos de condução obtidos através da observação de um condutor humano

Aprendizagem ≠ Memorização

5.2 Explorar / Aproveitar

Afinal, quando é que se aprendeu o suficiente para começar a aplicar o que se aprendeu?

E necessária a exploração, escolhendo uma acção que permita explorar o mundo para melhorar a aprendizagem.

Após essa exploração, há que saber aproveitar de acordo com a acção que leva à melhor recompensa de acordo com a aprendizagem.

5.2.1 Estratégias de Selecção de Acção

Estratégia *greedy*:

$$a_t = a^*_t = \text{argmax}_a \; Q_t(a)$$

Estratégia ε-*greedy:*

$$a_t = \begin{cases} a^*_t \text{ com probabilidade } 1 - \varepsilon \\ \text{acção aleatória com probabilidade } \varepsilon \end{cases}$$

5.3 SARSA

O estado pode evoluir ao longo do tempo da seguinte forma:

S – Estados observados

A – Acções realizadas

R – Reforços obtidos

S' – Estado seguinte

A' – Acção seguinte

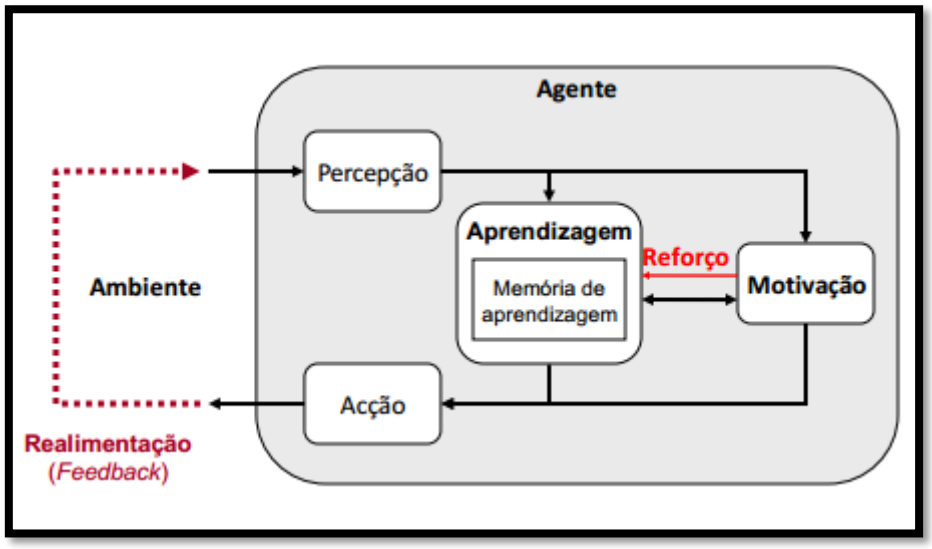
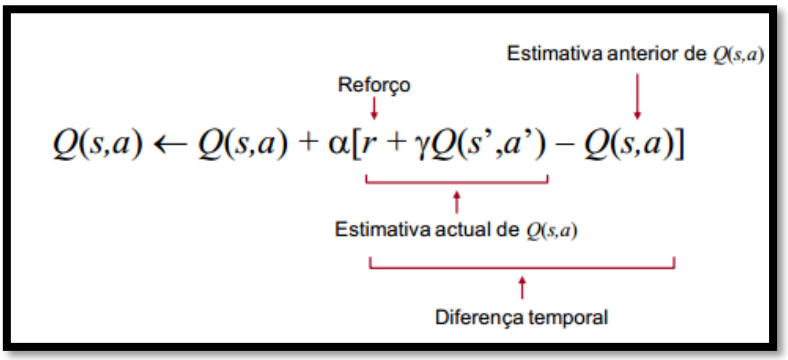


Figura 6 - Aprendizagem por Reforço

5.3.1 Diferença Temporal

Actualização de uma estimativa de valor de estado com base na sua mudança (diferença temporal) entre instantes sucessivos.



5.3.2 Algoritmo SARSA

- Iniciar **Q(s, a)**
- Repetir (por cada episódio):
  - Iniciar **s**
  - Escolher **a** de acordo com **s** com base numa politica derivada de **Q** (por exemplo ε-*greedy*)
  - Actualizar **Q**:
    - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    - $s \leftarrow s', a \leftarrow a'$

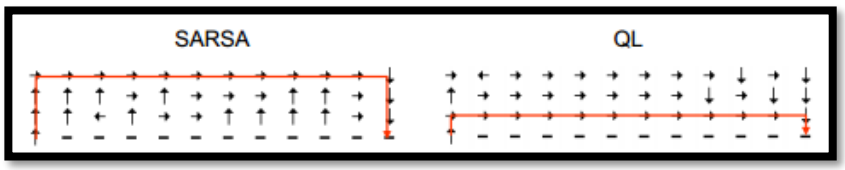
Figura 7 - Algoritmo SARSA

- Até **s** ser um estado terminal

5.4 Algoritmo Q-Learning

- Iniciar **Q**(***s***, ***a***)
- Repetir (por cada episódio)
  - Iniciar ***s***
  - Escolher ***a***de acordo com ***s*** com base numa política derivada de **Q**(por exemplo ε-*greedy*)
  - Repetir (por cada passo)
    - Executar ação ***a***, observar ***r*** e ***s***'
    - Escolher ***a'*** de acordo com ***s'*** com base numa política derivada de **Q**(por exemplo ε-*greedy*)
    - Atualizar Q:
    - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
    - $s \leftarrow s', a \leftarrow a'$
- Até ***s*** ser um estado terminal

5.5 SARSA vs. Q-Learning



Tendo em conta que no gráfico anterior os "-" são um género de abismo para o agente, é possível verificar-se que com o algoritmo SARSA o comportamento é mais cauteloso, não correndo o risco de "queda" perante o abismo. Já com Q-Learning é totalmente sôfrego, no sentido em que chega a correr o risco de, ao mínimo deslize, poder deitar tudo a perder.

5.6 Problemas e Soluções

Os grandes problemas em aprendizagem por reforço são essencialmente a complexidade dos espaços de estados, podem atingir tamanhos indesejados, e o tempo de convergência.

As soluções mais práticas são a utilização de memória episódica, de modelos do mundo, generalização do problema e até abstração.

CONCLUSÃO

Depois de completados todos os objetivos dos vários trabalhos práticos realizados podemos concluir que a Inteligência Artificial é um tema que tem tanto de interessante como de complexo, e que se ramifica em diversos mecanismos/tecnologias/métodos.

Este é um tema que cada vez mais está presente no nosso quotidiano sem darmos por isso, seja em aplicações de navegação, jogos, simulações, filmes, e que definitivamente veio para ficar.

Adquirimos conhecimentos sólidos de Engenharia de Software assim como os conceitos básicos da Inteligência Artificial e da sua implementação em sistemas autónomos, neste caso o Agente da PSA.

Apesar de ser um tema complexo e aparentemente difícil, o principal obstáculo que encontrámos relativamente aos trabalhos desenvolvidos não foi a componente teórica dos conceitos estudados, mas sim a sua implementação prática em linguagem *Python*, ao contrário do que esperávamos.

Ainda que apenas tenhamos "riscado a superfície" relativamente a este tema, sem dúvida que despertou a nossa curiosidade e nos deixou atentos para futuros desenvolvimentos nesta área, que se espera que nos próximos anos se desenvolva bastante e se torne vulgar, de modo a que os seres humanos possam aproveitar ao máximo todas as vantagens que esta área tem para oferecer.

BIBLIOGRAFIA

http://repositorio.ipl.pt/bitstream/10400.21/4412/1/Disserta%C3%A7%C3%A3o.pdf
- Integração de Reacção e Deliberação em Agentes Inteligentes
http://www2.ic.uff.br/~bianca/ia-20091/aulas/IA-Aula17.pdf
- Inteligência Artificial, Tomada de decisões complexas
http://www.psicologia.pt/artigos/textos/TL0012.PDF
- Inteligência Artificial, Humana e Emoção
http://cee.uma.pt/edu/iia/acetatos/iia-Procura%20Informada-PeB.pdf
- Inteligência Artificial, Procura Informada
https://fenix.tecnico.ulisboa.pt/downloadFile/3779573188260/cap3-procura.pdf
- Resolução de Problemas com Procura
http://www.rafaeldiasribeiro.com.br/downloads/IC1_6.pdf
- Inteligência Computacional