Licenciatura Engenharia Informática e Multimédia

Inteligência Artificial para Sistemas Autónomos

Relatório Final

Data: 6 Julho 2022

Docente:
Eng. Paulo Vieira

Realizado por:
Luís Fonseca - 45125

Índice de Conteúdos

1 Introdução

Este relatório é resultante de um conjunto de projetos que foram realizados ao longo do semestre, da unidade curricular Inteligência Artificial para Sistemas Autónomos, realizado pelo aluno identificado na capa e com o auxílio do docente da unidade curricular.

A realização destes projetos teve como principal objetivo estudar e compreender os conceitos chave da Inteligência Artificial, dando aos alunos as bases e ferramentas necessárias para poderem resolver projetos nesta área(uma área com enorme potencial e futuro).

Sendo a inteligência artificial uma das áreas mais complexas da área de software, os conteúdos aqui tratados e descritos são de grande complexidade, ou seja, recomenda-se que se tenha alguns conhecimentos sobre esta área para uma clara e total compreensão de todo o projeto.

2 Enquandramento Teórico

2.1 Inteligência Artificial

Antes de ser possível aprofundar esta área tão vasta é importante tentar perceber a definição da mesma.

Inteligência Artificial pode ser de um modo muito básico definida a partir das duas palavras que a compõem, resultando de tal análise, a ideia de que o assunto trata da criação de sistemas que apresentam inteligência pelo homem, através de ferramentas que permitam tal simulação, a computação.

Apesar desta definição não ser incorreta, a mesma é extremamente pobre quando comparada à imensão de aplicações desta área, bem como a vasta complexidade de uma das maiores áreas de conhecimentos dos nossos tempos.

Com o surgimento do conceito de inteligência artificial, foi também importante definir o conceito de inteligência. Este conceito por ser descrito através de um ciclo, onde primeiramente é necessário captar uma **perceção** que é uma função cerebral que atribui significado a determinado estímulo. Seguidamente é necessário obter consciência do problema de modo a ser possível realizar um **planeamento**. Este processo designa-se por **deliberação**. Este processo de deliberação leva o agente a tomar **ação**.

A inteligência artificial é dividida em diversos paradigmas, sendo os principais os seguintes:

- **Simbólico** - Estruturas simbólicos, representações gráficas ou sonoras da realidade. É resultante da ação de processos computacionais sobre estruturas simbólicas, sendo que um sistema de símbolos físicos tem os meios necessários e suficientes para a atividade inteligente em geral.
- **Conexionista** - Interação entre unidade de processamento. É uma propriedade emergente das interações de um número elevado de unidades elementares de processanento.
- **Comportamental** - Dinâmica comportamental individual e conjunta de múltiplos sistemas a diferentes escalas de organização.

2.2 Linguagem UML

A linguagem UML constitui uma forma de modelação concreta. A Unified Modeling Language (UML) fornece mecanismos padrão da indústria para visualizar, especificar, construir e documentar sistemas de software.

Em UML, objetos e classes são modelados por diagramas de objetos e classes. **Um diagrama de classes** mostra a estrutura estática das classes no sistema. As classes podem estar relacionadas entre si de várias maneiras, podendo apresentar relações de dependência, associação, agregação, composição e generalização. Estes relacionamentos são apresentados num **diagrama de classes**, juntamente com a estrutura interna das classes, relativamente a atributos e métodos. O diagrama é considerado estático no sentido de que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema.

A interação entre os objetos durante a execução do software é representada por **diagramas de sequência**, que demonstram uma sequência de mensagens enviadas entre os objetos e a interação entre objetos, e por diagramas de comunicação, que além de mostrar a troca de mensagens (chamada de interação), também apresenta os objetos e os seus relacionamentos (por vezes designado de contexto).

Inseridos na seção dos **diagramas de dinâmica** existem os diagramas de transição de estado, que possui todos os estados possíveis que os objetos da classe podem ter durante uma instância do ciclo de vida e quais eventos que fazem com que o estado mude, e os diagramas de atividade, que, por norma, são usados para descrever as atividades realizadas num fluxo de trabalho de processo geral .

Por fim, é importante destacar os **diagramas de módulos (Packages)** que têm como objetivo apresentar a organização de um módulo, proporcionando uma gestão e organização de sistemas complexos, e dividir um sistema em subsistemas e módulos.

2.3 Engenharia de Software

Consta no desenvolvimento,operação e manutenção do Software de maneira sistemática e quantificável. Um dos aspetos chave da engenharia de software é a complexidade, razão pela qual também se pode designar por engenharia da complexidade, uma vez que um sistema de software é um sistema bastante complexo. A engenharia de software propõe-se a desenvolver o software como uma atividade de engenharia. Ou seja, tal como se projeta e constrói um edifício, também é possível projetar e construir um software com foco na eficiência e na qualidade. Os aspetos chave para alcançar eficiência é ser sistemático e quantificável, de forma a criar uma ordem e uma organização que garanta o resultado pretendido. Sendo assim, de forma sucinta, o desafio e o propósito da engenharia de software é a capacidade em desenvolver software eficiente, sistemático e quantificável perante a complexidade e a mudança.

Na engenharia de software destacam-se 2 tipos de complexidade, a desorganizada e a organizada.

A complexidade organizada:

- Resulta do número e heterogeneidade das partes de um sistema
- As partes podem atuar entre si, mas a sua interação é irregular
- As características globais do sistema podem ser infereidas com base em métodos estatísticos

A complexidade desorganizada:

- Resulta dos padrões de inter-relacionamento entre as partes
- As interações entre partes obedevem a padrões correlacionáveis no espaço e no tempo
- Uma complexidade que produz propósito(finalidade) e função do sistema

Sendo que a engenharia de software tem o objetivo de controlo e reduzir a complexidade, a engenharia de software tem por base, três métricas:

- **Métrica:** corresponde ao que nos permite quantificar aquilo que está a ser realizado em termos de qualidade da arquitetura
- **Princípios:** os conhecimentos base para o correto desenvolvimento de software
- **Padrões:** permite aproveitar conhecimento anterior para o sistema em questão (designados por padrões de arquitetura de software)

2.3.1 Métricas de Arquitetura

Coesão: Nível de coerência funcional de um subsistema/módulo (até que ponto esse módulo realiza uma única função)

- Um nível de coesão baixo leva a que, em caso de necessidade de alteração de um subsistema, o número de módulos afetados seja elevado
- Se o nível de coesão for elevado, o número de módulos afetados será minimizado
- Um módulo com um nível de coesão é mais complexo, logo mais difícil de conceber e de testar

Acoplamento: grau de interdependência entre subsistemas.Quando falamos de acoplamento estamos a referir-nos ao modo de ligação das partes, em que se verifica que quanto maior forem as dependências maior é a complexidade. Como tal, pretendemos reduzir esta dependência. Podemos ter diferentes tipos de acoplamento:

- **Acoplamento de natureza estrutural:** as partes estão estruturalmente interdependetes (dependem de aspetos concretos uma das outras como por exemplo, variáveis ou atributos internos)
- **Acoplamento de natureza funcional:** as partes dependem da função que outras partes implementam/disponibilizam
- **Acoplamento de natureza denotacional:** acoplamento de natureza por significado (as partes não dependem de funções específicas, mas sim do significado dessas funções)
- **Acoplamento de natureza motivacional:** as partes determinam com quem interagir e de que forma interagir

Simplicidade: Nível de facilidade de compreensão e comunicação da arquitetura implementada.

Adaptabilidade: Nível de facilidade de alteração da arquitetura para incorporação de novos requisitos ou de alterações nos requisitos previamente definidos.

2.3.2 Princípios de Arquitetura

Modularidade: Consiste na descomposição e encapsulamento de um sistema em partes coesas para sistematizar e lidar com a explosão combinatória o qual permite um aumento da coesão e diminuição do acoplamento.

- Decomposição: permite sistematizar interações e lidar com a explosão combinatória. A fatorização permite eliminar a redundância e garantir a consistência.

- Encapsulamento: isolamento dos detalhes internos das partes de um sistema em relação ao exterior. O cesso exclusivo pode ser obtido através de interfaces que são funcionais para interação com o exterior.

Abstração: ferramenta base para lidar com a complexidade, pois permite identificar características comuns a diferentes partes.Consiste em realçar o essencial omitindo os detalhes irrelevantes através da simplificação e focagem.

Fatorização: Permite a redução/eliminação da redundância através da utilização de mecanismos de fatorização, tais como herança e delegação.Permite reduzir a complexidade.Por exemplo, se existir um código repetido, ao alterar num local têm de ser alterados todos os locais, sendo bastante suscetível a ocorrência de erro.

Resumidamente, o processo de desenvolvimento de software está relacionado com a criação de ordem de forma progressiva, sendo um processo interativo. Ou seja, não inclui um trabalho por tentativa e erro, pois irá causar um aumento exponencial da complexidade. Assim, o desenvolvimento de software deve ser um processo organizado, que tem por base métricas, princípios e padrões de arquitetura de software.

2.4 Agentes Inteligentes

Um agente pode ser visto como qualquer coisa que perceciona o seu ambiente através de sensores e age sobre esse mesmo ambiente através de atuadores. Esta organização interna do agente pode ser observada na figura abaixo.

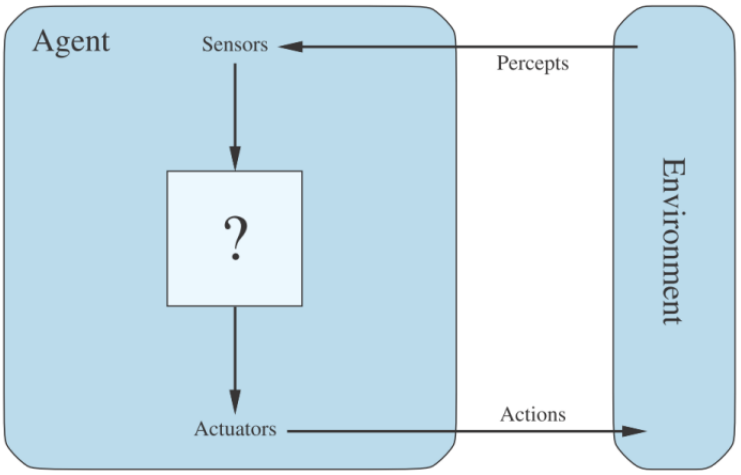

Figure 1: Organização interna de um agente

Um agente inteligente possui como principal propriedade a **autonomia**,além disso, poderá possuir outras propriedades, nomeadamente

- **Reatividade:**capacidade do agente reagir atempadamente a um estímulo do ambiente
- **Proatividade:** capacidade do agente reagir atempadamente a um estímulo do ambiente
- **Sociabilidade:** permite a agentes que pertençam a um ambiente com múltiplos agentes de operarem em conjunto e coordenem a sua atividade

Para o agente conseguir ter a sua própria autonomia, ele pode conter diferentes arquiteturas. Essas arquiteturas são as seguintes:

- **Modelo reativo** (paradigma comportamental): o processamento interno é o resultado de uma ligação muito forte entre a perceção e a ação. O foco é a modelação das associações entre os estímulos e as respostas (reatividade). Neste modelo, a especificação dos objetivos é implícita, uma vez que os mesmos

vão ser representados através da regra estímulo-resposta que caracterizam o sistema. A esquematização deste modelo pode ser observada na Figura abaixo.

Figure 2: Modelo reativo

- **Modelo deliberativo** (paradigma cognitivo): apresenta uma ênfase nas representações internas. Com as representações internas, o agente consegue realizar diferentes inferências que se designam por deliberação. A deliberação constitui um intermediário entre a perceção e a ação. Além disso, como podemos observar na abaixo, existe um ciclo fechado que implica a alimentação dos modelos internos pelos efeitos das ações no ambiente. Este ciclo pode ser designado por ciclo percecionar-deliberar-agir. Neste modelo, o agente possui outra propriedade, nomeadamente a proatividade e a capacidade de representar internamente informações e conhecimento sobre o ambiente envolvente, o que permite a definição explicita de objetivos.

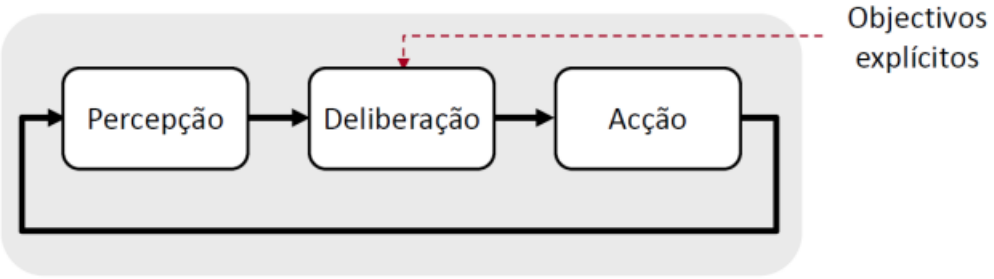

Figure 3: Modelo deliberativo

2.5 Arquitetura de Agentes Reativos

Os agentes reativos constituem o tipo mais simples de agente. Estes agentes selecionam ações com base na perceção atual, ignorando o histórico de perceção. Os agentes reativos são admiravelmente simples, possuindo inteligência limitada. Estes agentes apenas tomam uma decisão correta se a mesma pode ter por base apenas a perceção atual, isto é, se o ambiente for totalmente observável. Mesmo considerando um pouco de não observabilidade pode causar sérios problemas.

Os agentes reativos adotam uma arquitetura reativa, cujo esquema pode ser observado na Figura 4. O processamento interno destes agentes é constituído por regras estímulo-resposta, em que as ações são diretamente ativadas em função das perceções. Esta arquitetura implementa as 3 principais atividades internas destes agentes, nomeadamente percecionar, processar (reagir) e atuar.

No entanto, num sistema real complexo podem existir um grande número de reações e estas não devem ficar aglomeradas no processamento interno do agente. Como tal, é necessário modularizar o conjunto de reações que designam por módulos comportamentais ou simplesmente comportamentos. Assim, uma perceção pode potencialmente ativar múltiplas reações que por sua vez ativam distintas ações. Quando possuímos uma situação de múltiplas ações, a ação a ser realizada é determinada com base no **mecanismo de seleção de ação.**

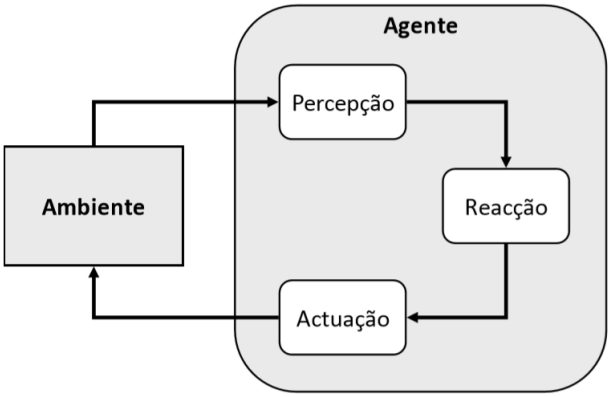

Figure 4: Arquitetura Reativa de um agente

2.5.1 Mecanismo de Reação

Este tipo de mecanismo segue a regra do estímulo-resposta, na qual as ações são diretamente ativadas em função das percepções

Neste tipo de mecanismo:

- Não são utilizadas representações internas do mundo
- Existem respostas rápidas a alterações no ambiente
- Respostas fixas e predefinidas aos estímulos do ambiente

No entanto, na arquitetura do tipo reativa simples **não são mantidas representações internas** do estado do mundo, as ações são ativadas **diretamente** em **função das percepções.**

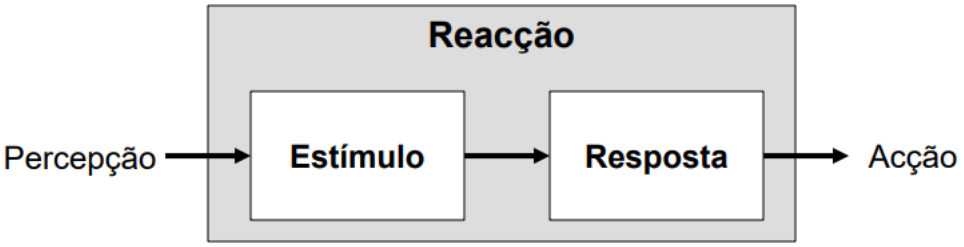

Figure 5: Mecanismo de reação segundo a regra estímulo-resposta

2.5.2 Mecanismo de Seleção de ação

Neste tipo de mecanismo pode acontecer que uma mesma percepção possa conter mais do que uma reação, gerando múltiplas ações, levantando a pergunta "Como podemos selecionar as ações a realizar?" Na figura 6, podemos ver que a mesma perceção pode ativar várias reações.

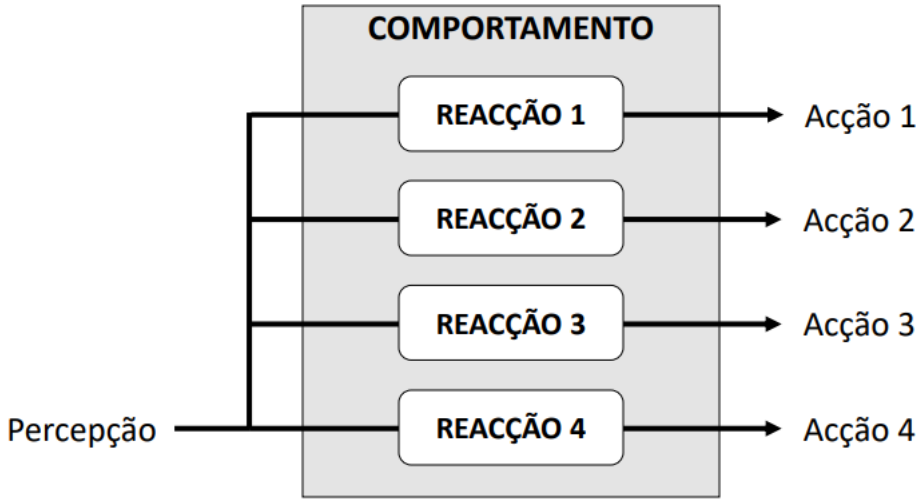

Figure 6: Mecanismo de reação ativando múltiplas reaçõe

.

Para ajudar na seleção da ação, podemos dividir em 3 principais ações:

- **Ações paralelas:** as várias ações podem ser executadas em paralelo, ou seja, as ações não entram em conflito umas com as outras
- **Ações prioritárias:** atribuição de prioridade às diferentes reações, ação relativa à reação com maior prioridade é executada
- **Ações combinadas:** mecanismo mais elaborado, em que ações distintas, desencadeadas por distintas perceções, são combinadas numa única ação

Para saber qual a ação a realizar existem um conjunto de diferentes mecanismos. Os mecanismos são os seguintes:

- **Hierarquia:**Os comportamentos estão fixos numa hierarquia fixa de supressão onde as reações superiores suprimem as que estão abaixo delas.

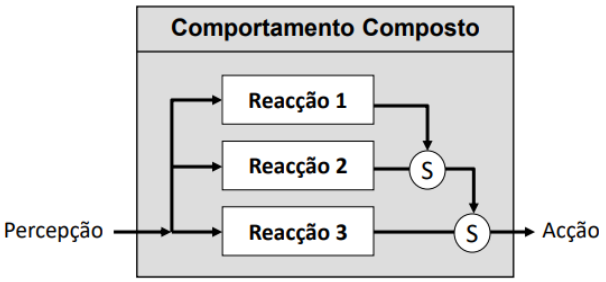

Figure 7: Modelo Hierarquia

- **Prioridade:**As respostas são selecionadas de acordo com uma prioridade que tem associada.

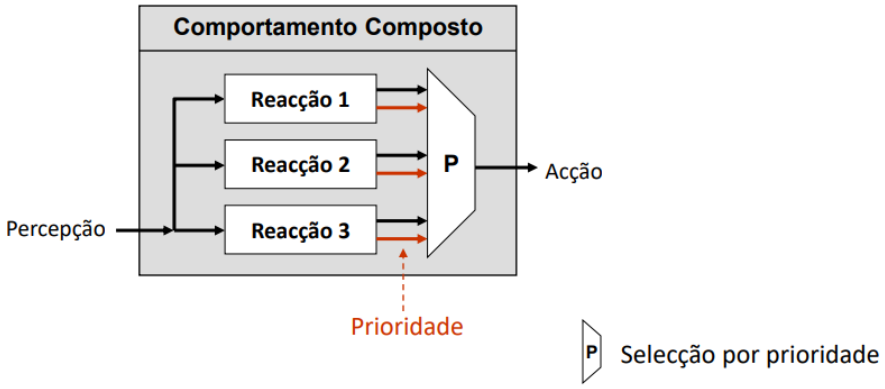

Figure 8: Modelo Prioridade

- **Fusão:**As respostas são combinadas numa só por composição.oridade que tem associada.As prioridades variam ao longo da execução do agente.

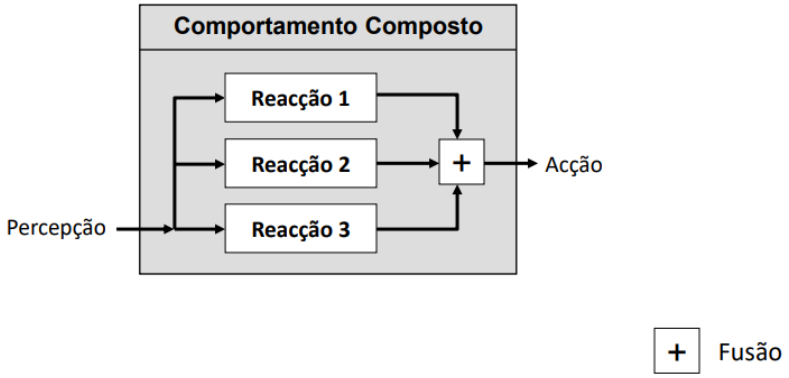

Figure 9: Modelo Fusão

2.6 Arquitetura Reativa sem Memória

Este tipo de arquitetura apresenta problemas na sua implementação. Os problemas são os seguintes:

- **Exploração:** necessita de conhecer o passado
- **Óptimos locais:** como exemplo: os veículos de Braitenberg, ficam presos nos cantos, incapazes de dar a volta
- **Comportamentos cíclicos:** Por exemplo, os Veículos de Braitenberg ficam a movimentar-se ciclicamente perante determinadas configurações de alvos e obstáculos.

A única solução para implementar este tipo de problemas, é a manutenção de estado do agente, recorrendo à arquitetura reativa com memória.

2.7 Arquitetura Reativa com Memória

Para evitar estes problemas é preciso implementar uma arquitetura reativa com memória. Com este tipo de arquitetura fica então possível:

- Produzir todo o tipo de comportamento
- Capacidade lidar com falhas
- Possibilidade de representar dinâmicas temporais, isto é, gerar uma resposta não só em função de perceções atuais, mas também em perceções prévias.

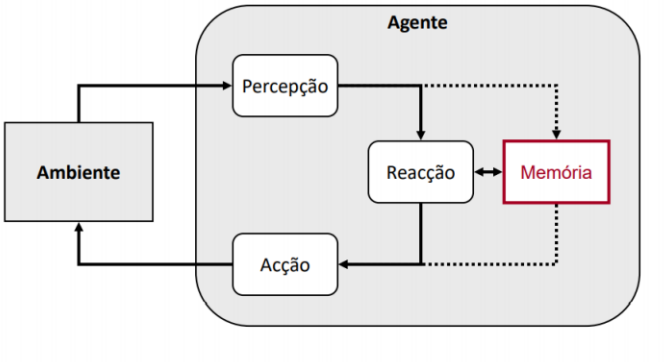

Figure 10: Arquitetura Reativa Com Memória

No entanto, este tipo de arquitetura contém as suas principais vantagens e desvantagens. As vantagens são as seguintes:

- Uma arquitetura reativa com estado pode produzir todo o tipo de comportamento
- Possibilita de representar dinâmicas temporais
    1. Evolução do estado ao longo do tempo
    2. Resposta não apenas em função das percepções atuais, mas também em função de memórias de percepções anteriores
- Possibilidade de comportamentos mais complexos baseados na evolução de estado
    1. Com continuidade no tempo
    2. Agir devido a ausência de mudança
- Capacidade de lidar com situações de falha por exploração de ações não realizadas anteriormente

As desvantagens são as seguintes:

- Necessita de memória (espaço)
    1. Aumento da complexidade espacial
- É necessário manter as representações de estado
    1. Aumento da complexidade computacional

- Mesmo com a manutenção de estado, as arquiteturas reativas não suportam representações complexas, nem exploram planos alternativos de ação

2.8 Arquitetura de subsunção

Este tipo de arquitetura permite que os comportamentos sejam organizados em camadas e que sejam responsáveis pela concretização independente de um objetivo. O resultado do comportamento pode ser a entrada de outro comportamento. Existe a possibilidade de comportamentos de camadas superiores assumirem o controlo sobre comportamentos das camadas inferiores, onde a camada inferior não tem conhecimento das camadas superiores – Hierarquia de comportamentos. Nesta arquitetura as camadas superiores controlam as camadas inferiores, onde as saídas das camadas inferiores podem ser controladas pelas camadas superiores

Nesta arquitetura as camadas superiores controlam as camadas inferiores, onde as saídas das camadas inferiores podem ser controladas pelas camadas superiores. O controle das camadas inferiores pode ser feito por:

- **Inibição-** Desativação de comunicação entre módulos;
- **Supressão-** Desativação do comportamento;
- **Reinício-** Reposição do estado inicial de um comportamento;

Na arquitetura de subsunção a implementação de módulos comportamentais pode ser feitas de varias formas:

- Com base em sequências de ativação fixa;
- Com base em regras de estímulo-resposta;
- Com base em máquinas de estado aumentadas;

2.9 Raciocínio Automático

Consiste na capacidade de um sistema computacional resolver de forma automática um problema com base numa representação de conhecimento do respetivo domínio, produzindo uma solução a partir de diversas alternativas possíveis.

Processo computacional que tendo como entrada uma representação de conhecimento de um determinado domínio, produz como resultado conclusões baseadas nesse conhecimento.

Este tipo de raciocínio pode dividir-se em:

- Exploração de opções
    1. Raciocínio prospetivo (antecipação)
    2. Simulação interno do mundo (representação interna)
- Avaliação de opções
    1. Custo
    2. Utilidade

O processo de manipulação da representação de conhecimento de forma a obter conclusões é normalmente designado **inferência**.

No entanto, para suporte do raciocínio automático, é necessário modelar o problema. Para isso irá ser feita uma representação dos seguintes conceitos:

- **Estado**
    1. Representa uma situação (configuração) na resolução de um problema
    2. Identificação única
    3. Espaço de estados: conjunto de estados e de transições de estado
- **Operador (transição de estado):**
    1. Representa uma ação
    2. Gera transformação de estado (exemplo: operador.aplicar: estado → estado)
- **Problema**
    1. Estado inicial
    2. Operadores

3. Objetivos (ou função objetivo: estado → True, False)

- **Mecanismo de Raciocínio:** Exploração de opções possíveis para encontrar uma solução através de simulação prospetiva, tendo por base uma representação interna do problema

2.10 Procura em Espaço de Estados

Procura em Espaço de Estados consiste numa exploração sucessiva do espaço de estados, tendo como início no estado inicial e fim no estado final.

- Recorre ao conceito de **estado** que define uma configuração do problema
- **Operador** que define uma transformação de estado (transição de estado) e este pode ser associado um custo (representação simbólica da ação)
- A definição do **problema** que consiste num estado inicial, na definição de uma função objetivo (função que indica quais são os estados objetivos) e um conjunto de operadores
- Por fim, através da exploração do espaço de estados é produzida uma **solução**, que corresponde a um percurso (plano de ação).

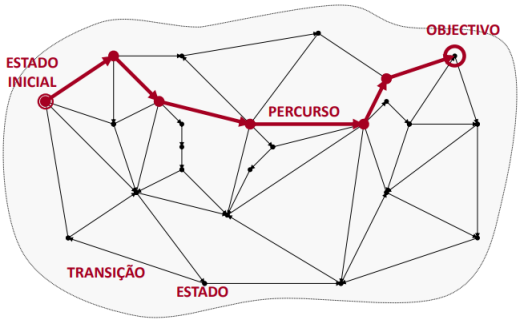

Figure 11: Ilustração da procura em espaço de estados

Na figura acima, podemos associar a procura dos estados como sendo uma "árvore". Essa árvore constitui a estrutura de dados que suporta um processo de procura, e esta, por sua vez, possui como elemento estruturante o nó. Um estado do espaço de estados é representado por um nó e esse por sua vez representa uma etapa de procura (um passo na procura). A etapa de procura corresponde ao efeito da aplicação de um operador a um estado que, por sua vez, produz uma transição para o próximo estado. A raiz da árvore é o nó que corresponde ao estado inicial. As folhas das árvores são nós não expandidos, ou seja, nós que ainda não possuem nós sucessores

(não foram aplicados operadores aos estados a que esses nós correspondem). Para que seja possível estruturar árvores, os nós têm de estar interligados. Assim, cada nó possui informação sobre o seu nó antecessor. É de salientar que numa árvore podem existir diferentes nós que correspondem ao mesmo estado. Assim, o percurso desde uma folha(nó objetivo) até o nó raiz corresponde à solução do problema.

Para a avaliação de um método de procura deve considerar se os seguintes critérios:

- **Completo** ou seja, garante que, caso exista solução, esta será encontrada
- **Ótimo** garante que, existindo várias soluções, a solução encontrada é a melhor
- **Complexidade** relativa ao esforço para encontrar solução que pode ser dividida em
    1. Complexidade temporal(tempo): tempo necessário para encontrar a solução
    2. Complexidade espacial(espaço): memória necessária para encontrar uma solução

É de mencionar também os parâmetros de caracterização de um método de procura:

- **Fator de Ramificação(b)**: número máximo de sucessores para um qualquer estado
- **Profundidade de procura(d)**: profundidade do nó objetivo menos profundo na árvore de procura. Consiste na dimensão do percurso entre o estado inicial e o estao objetivo

Quando se trata de definir a maneira de efetuar o processo de procura existem duas hipóteses:

- Explorar primeiro os nós mais recentes
- Explorar os nós mais antigos primeiro

Nas figuras 12 e 13, é possível de ver uma representação gráfica de ambas as situações em estudo.

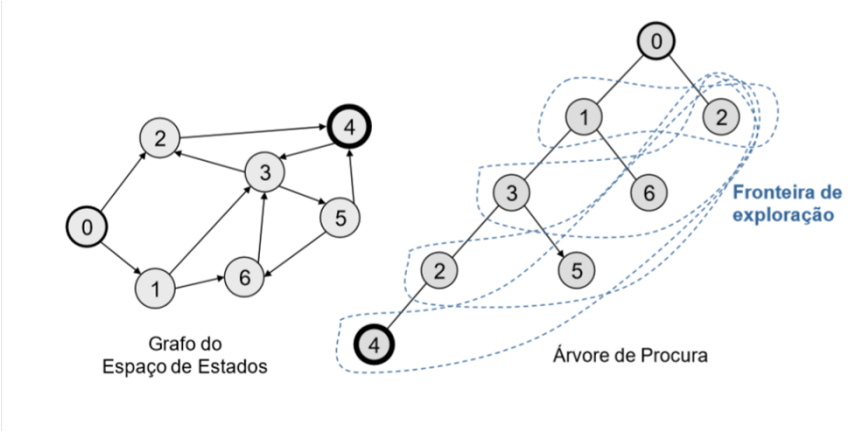

Figure 12: Exploração dos estados, tendo como cenário explorar os nós mais recentes primeiro

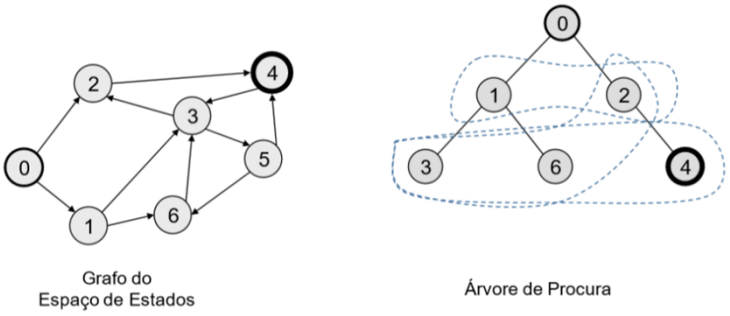

Figure 13: Exploração dos estados, tendo como cenário explorar os nós mais antigos primeiro

2.11 Métodos de Procura

2.11.1 Procure Melhor-Primeiro

No método de procura melhor-primeiro é uma instância do algoritmo geral de procura em grafos com ciclos em que um nó é selecionado para expansão com base numa função de avaliação ($f(n) \geq 0$) . Tipicamente, esta função, representa uma estimativa do custo da solução (função de custo), em que quanto menor o valor de f(n), mais promissor é o nó n.

Neste método de procura a fronteira de exploração é ordenada por ordem crescente de f(n).

No método de procura melhor-primeiro, a utilização do algoritmo geral de procura em grafos (versus algoritmo de procura em árvore) é mesmo necessário, pois é fulcral garantir a eliminação de nós repetidos, mantendo apenas o nó com menor custo (melhor custo produzido pela função f(n)).

Os métodos de procura podem-se dividir pelos seguintes grupos:

- Procura Não Informada - Consiste em estratégias de exploração do espaço de estados que não tiram partido do conhecimento do domínio do problema para ordenar a fronteira de exploração. Não se sabe como chegar ao objetivo, ou seja, consiste numa procura nao guiada(exaustiva). Os tipos de procura aqui contidos são :
  1. Profundidade
  2. Largura
  3. Custo Uniforme
- Procura Informada - Consiste em estratégias de exploração do espaço de estados que tiram partido do conhecimento do domínio do problema para ordenar a fronteira de exploração.Sabe-se como chegar ao objetivo, ou seja, consiste numa procura guiada(seletiva). Os tipos de procura aqui contidos são:
  1. Sôfrega
  2. Procura A*

2.11.2 Procura em profundidade

Critério de exploração: maior profundidade

Esta procura opta sempre por explorar **os nós mais recentes** na fronteira. Querendo isto dizer que o primeiro nó da fronteira a ser processado é o nó que mais recen-

temente foi colocado na fronteira, e o resultado é dependente da ordem dos operadores. Deste modo é realizada uma procura extensiva em profundidade. Esta procura aprensenta uma desvantagem, que ao entrar num ramo sem fim, nunca vai ser encontrada solução. Com isto é possível de compreender que numa árvore nunca há ciclos, no entanto o grafo pode ter ciclos e assim a árvore pode ter ramos infinitos.

Este tipo de procura contém duas variantes:

- **Procura em profundidade limitada:** quando a profundidade limite for atingida. Caso seja alcançada não é realizada mais expansão de nós e a fronteira fica vazia o que faz com que termine a sua resolução
- **Procura em profundidade iterativa:** É uma procura que combina as vantagens da procura em Profundidade e da Largura. Limita a procura a uma profundidade máxima, resolvendo o problema de encontrar um ramo sem fim e é criada devido à procura em largura consumir muitos recursos. Este método de procura vai ter uma complexidade espacial inferior ao método de procura e de largura, e ligeiramente superior à procura em profundidade, pois apresenta a mesma complexidade temporal que a procura em largura e é um método de procura ótimo e completo.

  Este método de procura consiste em realizar uma procura em profundidade onde a limitação da profundidade máxima é alterada iterativamente. Deste modo garante-se que se encontra a solução à menor profundidade possível e garante-se que, caso exista, a solução é sempre encontrada.

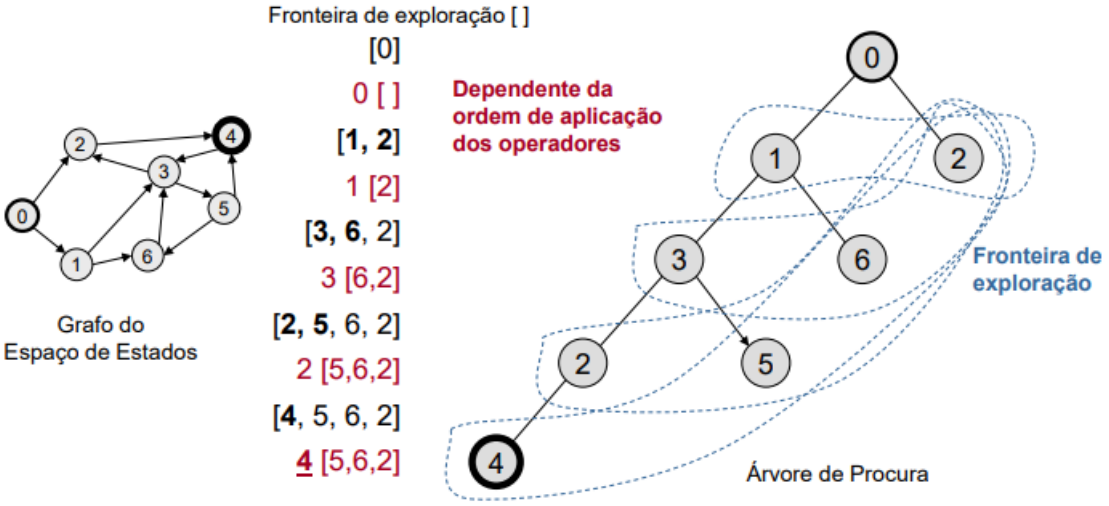

Figure 14: Estratégia de controlo da procura em profundidade

2.11.3 Procura em Largura

Critério de exploração: menor profundidade

Este tipo de procura permite explorar primeiro os nós mais antigos, o que isto significa que os nós colocados primeiramente na fronteira sejam expandidos antes dos que foram colocados mais recentemente nesta fronteira. Deste modo só se processam nós na profundidade seguinte caso já todos os estados da profundidade atual tenham sido explorados.Se a solução estiver no nível da profundidade atual é logo encontrada, ou seja, o resultado é independente da ordem dos operadores.

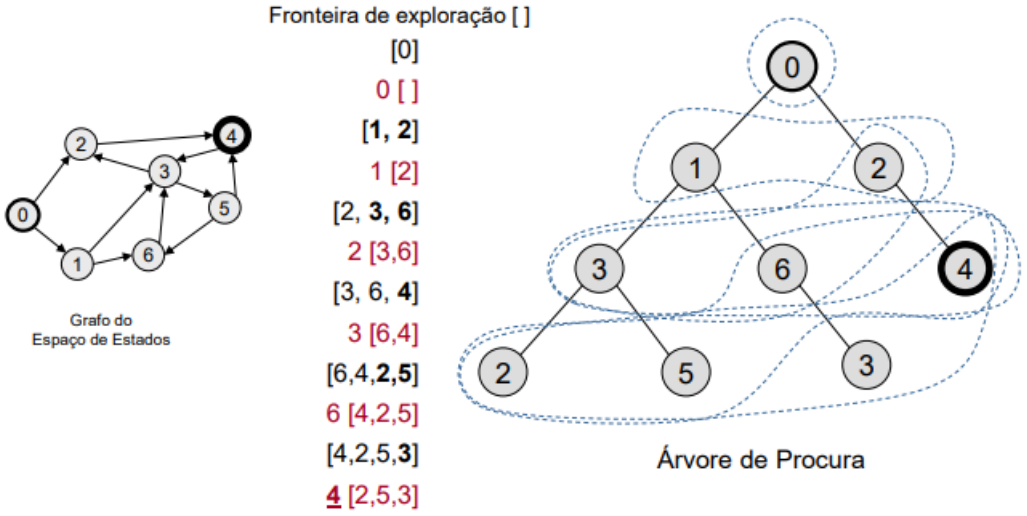

Figure 15: Estratégia de controlo da procura em largura

2.11.4 Procura Custo Uniforme f(n) = g(n)

Critério de exploração: custo da solução Este tipo de procura, é do tipo melhor-primeiro onde explora primeiro os caminhos com o menor custo garantindo assim uma procura ótima (f(n) = g(n))

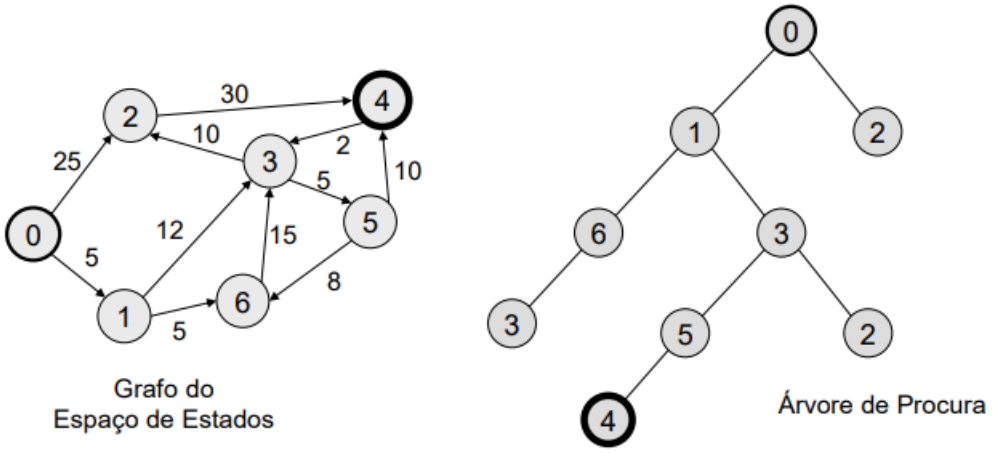

Figure 16: Estratégia de controlo da Procura Custo Uniforme

2.11.5 Comparação das procuras não informadas

Comparando os métodos de procura, a procura em profundidade não é completa, não é ótima (pode não encontrar solução), a complexidade temporal é exponencial e a complexidade espacial é linear. A procura em largura é ótima, completa, a complexidade temporal é exponencial, no entanto, a complexidade espacial é igualmente exponencial. Por sua vez, o método procura uniforme é completo e ótimo.

A procura em profundidade em termos de complexidade espacial é boa, uma vez que a mesma é linear, no entanto, não é ótima, nem completa. Assim, para que seja possível recorrer a uma procura em profundidade, tirando partido da complexidade espacial linear, foi desenvolvida a procura em profundidade interativa. Basicamente, este tipo de procura limita a profundidade de procura máxima de forma interativa. Ou seja, em cada iteração, se a solução não for encontrada, a profundidade de procura máxima é aumentada. Na procura em profundidade iterativa não podemos manter a informação sobre as procuras anteriores, caso contrário, a complexidade espacial torna-se exponencial. Como tal, em cada interação a procura em profundidade iterativa repete sucessivamente todas as procuras.

2.11.6 Função Heurística

Uma função heurística h(n) representa uma estimativa do custo do percurso desde o nó n até ao objetivo. Consiste numa forma rápida de produzir uma solução. O seu valor é independente do percurso até n, depende apenas do estado associado a n e do objetivo.

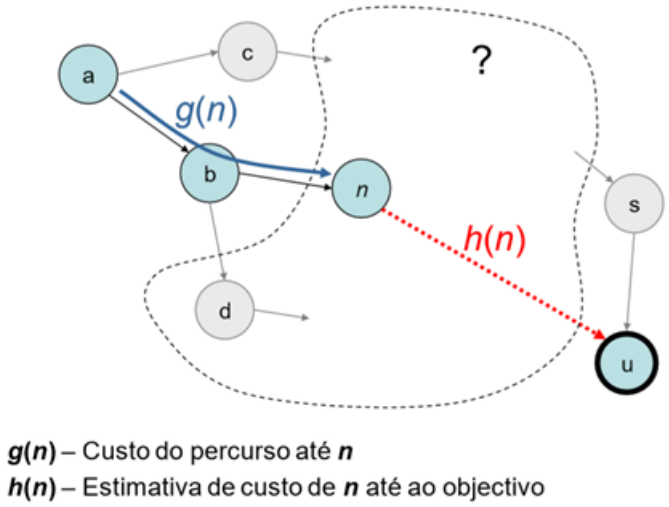

Figure 17: Método de procura informada, com a funcao heuristica aplicada

2.11.7 Procura Sôfrega (f(n) = h(n))

Procura que não tem em conta o custo do percurso explorado, tem apenas a função heurística. Faz uma minimização de custo local e tem soluções sub-ótimas(problema dos ótimos locais) no qual f(n)=h(n).

2.11.8 Procura A* (f(n) = g(n) + h(n))

Até a este momento apenas foi discutida a qualidade dos diferentes modos de procura por via de uma métrica computacional, considerando os conceitos de tempo e memória. No entanto, ainda existirá uma outra métrica a discutir, o custo. O custo é uma métrica usada na avaliação de situações reais, sendo, portanto importante uma procura que prioruize este parâmetro, Este tipo de procura é denominado como uma procura "Best-First".

A procura "Best-First" apresenta três variações uma procura de custo uniforme, onde não se tira partido do conhecimento do domínio do problema. A procura Sôfrega não tem em conta o custo que a solução pode apresentar e apresenta soluções sub-ótimas. A procura A* é uma procura que minimiza o custo global.

Este tipo de procura é uma heurística admissível, ou seja a estimativa de custo realizada é sempre inferior ou igual ao custo mais baixo possível.

Esta função de avaliação de estado é igual à combinação da função heurística e do custo do percurso explorado. A combinação destes destas 2 funções produz, aquilo que se designa, por um método de procura ótimo. É considerado um método ótimo porque é ótimo, completo e produz o menor esforço computacional possível para encontrar uma solução ótima. Estas caraterísticas apenas ocorrem se a heurística for admissível.

2.11.9 Procuras - Complexidade Computacional

Na tabela a seguir, é possível de ver as diferentes procuras.

Método de Procura	Tempo	Espaço	Óptimo	Completo
Profundidade	$O(b^m)$	$O(bm)$	Não	Não
Largura	$O(b^d)$	$O(b^d)$	Sim	Sim
Custo Uniforme	$O(b^{[C^*/\varepsilon]})$	$O(b^{[C^*/\varepsilon]})$	Sim	Sim
Profundidade Limitada	$O(b^l)$	$O(bl)$	Não	Não
Profundidade Iterativa	$O(b^d)$	$O(bd)$	Sim	Sim

b – factor de ramificação
d – dimensão da solução
m – profundidade da árvore de procura
l – limite de profundidade
C^* – Custo da solução óptima
ε – Custo mínimo de uma transição de estado ($\varepsilon > 0$)

Figure 18: Tabela com as procuras

2.12 Arquitetura Deliberativa

Este tipo de arquitetura segue a abordagem clássica da Inteligência Artificial, onde os agentes atuam com pouca autonomia e possuem modelos simbólicos explícitos dos seus ambientes. Esta arquitetura também tem por base o tempo passado, presente e futuro, ao contrário da reativa que apenas tem em conta o presente e possivelmente o passado.

A memória é um dos conceitos mais importantes quando estamos a estudar esta arquitetura, pois é este que permite recordar o passado, percecionar e atuar no presente e antecipar o presente. O agente para explorar as várias opções tem de ter uma representação interna do mundo e conseguir antecipar a próxima ação através de memória. Assim na escolha da melhor opção é tido em conta o custo e a utilidade

Neste tipo de arquitetura a memória desempenha o papel principal no processamento interno do agente. Pois é na memória que ocorre a simulação do futuro, antecipando o que pode acontecer, e esta suporta o raciocínio interno, ou seja, a deliberação, que, por sua vez,determina ação. O modelo base da arquitetura de agentes deliberativos é apresentada na Figura 19.

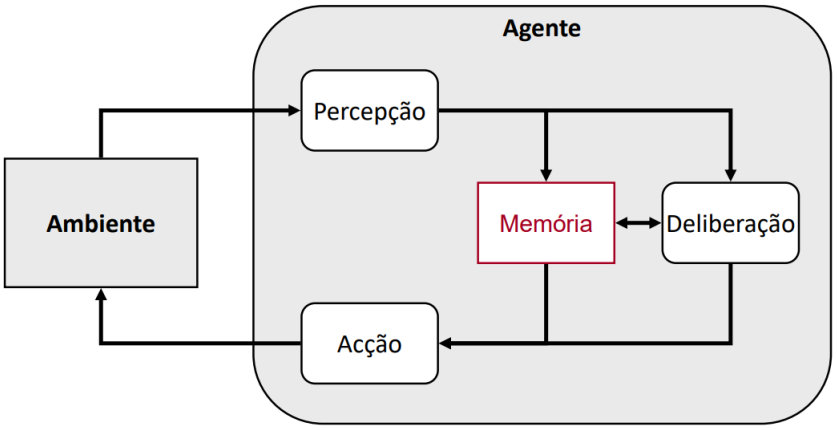

Figure 19: Arquitetura de Agentes Deliberativos

O raciocínio de um agente deliberativo ocorre por exploração de opções a partir de uma situação atual (por exemplo, processos de decisão de Markov). De seguida, essa simulação é suportada numa representação interna, onde é necessário avaliar as diversas opções (por exemplo, avaliação por custo ou utilidade).

O raciocínio prático é um raciocínio muito importante quando nos referimos a agentes autónomos e constitui um caso particular de raciocínio, uma vez que está

orientado para a ação. O raciocínio orientado para a ação envolve dois aspetos principais:

- **Saber o que fazer**
- **Saber como fazer**

. O raciocínio prático apresenta elementos de suporte que incluem:

- **Representação dos objetivos a atingir:** representados como uma árvore, onde no tipo está o objetivo prioritário e nos ramos os sub-objetivos;
- **Representação das ações realizáveis:** consistem nos operadores de movimento;
- **Representação do mundo**

Por fim, o resultado deste tipo de raciocínio são os planos de execução que permitem concretizar os objetivos a atingir.

Os aspetos principais do raciocínio orientado para a ação originam duas formas de raciocínio:

- **Raciocínio sobre fins (deliberação):** a deliberação pondera sobre múltiplas opções (situações a concretizar) e seleciona aquelas que são efetivamente para concretizar (objetivos).
- **Raciocínio sobre meios (planeamento):** o planeamento decide que ações vão ser realizadas (como fazer) e o resultado é um plano com as sequências de ações a executar

O esquema da arquitetura deliberativa com a inclusão do raciocino prático e da representação interna pode ser observado na Figura 20.

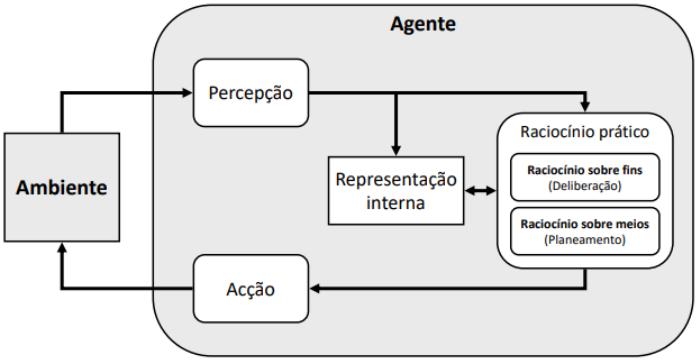

Figure 20: Arquitetura de Agentes Deliberativos com raciocínio prático

O raciocínio prático é concretizado a partir do processo geral de tomada de decisão. Assim, a tomada de decisão de um agente autónomo num ambiente estático ocorre através de 5 passos:

1. **Observar o mundo**
2. **Atualiza o modelo do mundo (representação interna)**
3. **Deliberar sobre o que fazer (quais os objetivos a concretizar)**
4. **Planear como fazer (como concretizar os objetivos selecionados)**
5. **Executar o plano de ação.**

No entanto, é importante ter em consideração que o raciocínio pode ser um processo demorado, pois envolve simular e avaliar múltiplas hipóteses possíveis de evolução futura para os múltiplos estados que o agente pode apresentar. Como o ambiente pode ser dinâmico, o resultado do raciocínio pode não ser consistente com a situação atual do ambiente. Nesse sentido, é necessário adicionar outro passo no processo de tomada de decisão, nomeadamente, a reconsideração de opções, que corresponde à reavaliação das opções com eventual mudança de planos, implicando uma nova deliberação e um novo planeamento.

2.13 Planeamento Automático com base em PEE

Podemos associar raciocínio automático a um planeador, onde é orientado para a geração de estratégias ou planos de ação (sequências de ação), em particular para execução de agentes inteligentes.

Na figura 21, é de notar que com um planeador, fica mais fácil o agente conhecer o(s) objetivo(s), e um modelo de planeamento, que representa um estado (sendo o inicial).

Para este tipo de planeamento podemos associar diferentes métodos de planeamento automático, nomeadamente mecanismos de procura em espaços de estados e processos de decisão de Markov.

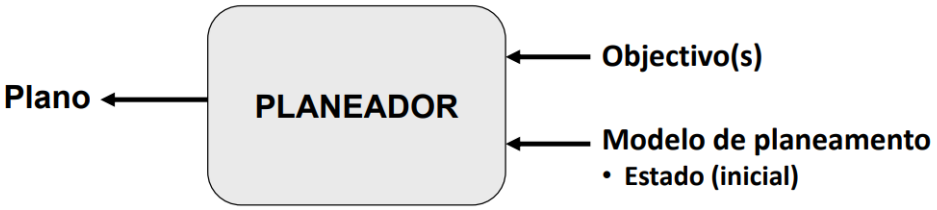

Figure 21: Planeador em PEE

2.14 Processo De Decisão Sequencial

Num processo de decisão sequencial o objetivo é conseguir prever e controlar o desenrolar da interação entre o agente e o ambiente ao longo do tempo, no sentido de otimizar a realização de um objetivo a longo prazo. No caso geral, um processo de decisão sequencial representa um processo de decisão em que o objetivo não está definido com um ou vários estados concretos finais, mas estão relacionados com a maximização de uma medida de valor, normalmente quantificada em termos de utilidade

2.14.1 Processos de Decisão de Markov

A propriedade de Markov consiste na definição de um sistema em que a previsão dos estados seguintes apenas depende do estado presente. Esta propriedade pode ser incorporada nos processos de decisão sequencial, dando origem aos processos de decisão de Markov (PDM). Assim, no PDM a previsão dos estados futuros e de recompensas futuras só depende do estado atual.

No caso do agente, não se pode ter em conta o custo, mas sim a utilidade que depende da sequência de decisões tomadas pelo agente, existindo possibilidade de ganhos e perdas. Ou seja, o agente tem incerteza na decisão, e de acordo com a sua decisão irá ter uma recompensa que pode ser positiva ou negativa, onde no final existe um efeito acumulativo que representa a utilidade.

Nota:A utilidade é o valor do efeito acumulativo. A recompensa nunca pode ser o guia de tomada de decisão!

De acordo com o PDM a previsão dos estados seguintes só depende do estado presente e o mundo está representado da seguinte forma:

- S – Conjunto de estados;
- A(s) – Conjunto de ações possíveis no estado s pertencente a S;

- T (s, a, s') – Probabilidade de transição de s para s' através de a;
- R (s, a, s') – Retorno ou recompensa esperado na transição de s para s' através de a;
- γ - Taxa de desconto para recompensas diferidas no tempo, ou seja, é a perda de oportunidade. Só toma valores 0 ou 1 [0,1] representando o fato de desconto temporal;
- t – Tempo discreto: 1,2,3...

A utilidade é o efeito acumulativo da evolução da decisão, ou seja, é uma estimativa do que se espera ganhar ou perder ao longo do tempo. Sintetizando,é o efeito acumulativo das recompensas, sendo que as recompensas podem ser positivas ou negativas possuindo um fator de desconto. É calculada pela seguinte expresão:

$\text{Uh}([s0, s1, s2, ...]) = R(s0) + \gamma R(s1) + \gamma 2R(s2) + ...$

Esse fator de desconto, está relacionado com a preferência do agente por recompensas correntes em relação a futuras, ou seja, um fator de desconto muito pequeno, leva o agente a ser mais "medroso", isto é, ele fica sempre no mesmo local, porque é melhor estar ali do que se mover e prejudicar-se. Já um fator de desconto grande, provoca um aumento no número de iterações necessárias, sendo por isso importante, variar esse fator tendo em conta o objetivo final. A Política Comportamental é uma forma de representação do comportamento do agente e define uma estratégia global de ação(ação que deve ser realizada em cada estado). Existem dois tipos de política:

- Política Determinista - $\pi : S \rightarrow A(s)$; $s \in S$
- Política Não Determinista – $\pi : S \rightarrow A(s)[0:1]$; $s \in S$

O Princípio da Solução Ótima requer a decomposição em sub-problemas, onde as utilidades dos estados podem ser determinadas em função das utilidades dos estados sucessores. Este princípio pode ser descrevido em função das equações de Bellman da seguinte forma:

$$U^{\pi}(s) = \mathrm{E}\left\langle r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots \right\rangle$$
$$= \mathrm{E}\left\langle r_1 + \gamma U^{\pi}(s') \right\rangle$$
$$= \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma U^{\pi}(s') \right]$$

Equações de Bellman

Figure 22: Equação de Bellman para o princípio da solução

Uma política ótima (π^*) produz a utilidade mais alta possível:

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

Figure 23: Equação de Bellman, com utilidade mais alta

E uma utilidade de estado para uma política ótima π^*:

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

Figure 24: Equação de Bellman com utilidade de estado e uma política ótima

2.15 Aprendizagem

Com base na aprendizagem automática, a aprendizagem é uma melhoria de desempenho(D) para uma determinada tarefa(T), com a experiência(E). Onde por exemplo, em jogar xadrez a tarefa seja jogar xadrez, o desempenho a percentagem de jogos ganhos, e a experiência os jogos realizados. O conceito de aprendizagem é totalmente diferente de memorização. A aprendizagem baseia-se na generalização, ou seja, na formação de abstrações que consistem em modelos.

Existem dois tipos de aprendizagem:

- **Aprendizagem Conceptual:** baseia-se em saber o conceito das coisas e pode ser supervisionada ou não supervisionada;
- **Aprendizagem Comportamental:**baseia-se em realizar comportamentos, ou seja, ações através do reforço;

2.16 Aprendizagem por reforço

A Aprendizagem por reforço baseia-se numa aprendizagem que tem por base a interação com o ambiente e da realização de comportamentos de forma a ganhar experiência. A partir de um determinado estado o agente escolhe a ação para mudar para o estado seguinte. Essa ação gera um reforço (aquilo que na prática concretiza a motivação através de um incentivo) positivo ou negativo. Ou seja, o agente vai aprendendo ao longo do tempo de acordo com a experiência que ganha.

Ao contrário dos processos de decisão sequencial o agente quando implementa uma apredizagem por reforço é como se andasse às "escuras", so tendo noção da posição em que se encontra e se atingiu algo.

A recompensa na apredizagem por reforço é diferida, ou seja, não é imediata o que gera incerteza no agente. Sendo que o objetivo é maximizar a recompensa a longo prazo. Como o agente escolhe as ações experimentando-as, o que interessa saber é o valor da ação que corresponde à média:

$$Q_n(a) = \frac{r_1^a + r_2^a + \cdots + r_n^a}{n}$$

Figure 25: Valor da Ação (Cada tentativa produz uma recompensa rn)

$$Q_n(a) = Q_{n-1}(a) + \frac{1}{n}[r_n^a - Q_{n-1}(a)]$$

Figure 26: Valor da Ação(forma incremental)

Sendo que:

- n = nº de iteração, ou seja, as vezes que o agente passou no mesmo estado;
- α = fator que permite ver se o agente tem tudo necessário para a sua aprendizagem;

2.17 Processos de Aprendizagem

Existem 2 tipos de processos de aprendizagem:

- **Política de seleção de ação única (On-Policy):** usa a mesma política de seleção de ação para comportamento e para programação de valor, explorando todas as ações (política épsilon-greedy).
- **Política de seleção de ação diferenciadas(Off-policy):** usam políticas de seleção de ação distintas para comportamento e propagação de valor, otimizando a função de valor Q(s,a).

2.18 Explorar vs Aproveitar

O agente depois de aprender tem que aplicar o que aprendeu, mas, no entanto, é difícil de saber quando é que o agente já aprendeu o suficiente para aplicar. Como tal, o agente pode explorar ou aproveitar:

- O agente ao explorar, escolhe uma ação que permite explorar o mundo de forma a melhorar a aprendizagem
- O agente ao aproveitar escolhe a ação que leva à melhor recompensa de acordo com a aprendizagem. Segue uma estratégia Greedy que corresponde a uma ação Sôfrega. Se o agente aproveita muito fica mais "medronho" por não arriscar;

2.19 Aprendizagem por Diferença Temporal

A aprendizagem por diferença temporal, baseia-se na atualização de uma estimativa de valor de estado com base na mudança (diferença temporal) entre instantes sucessivos.

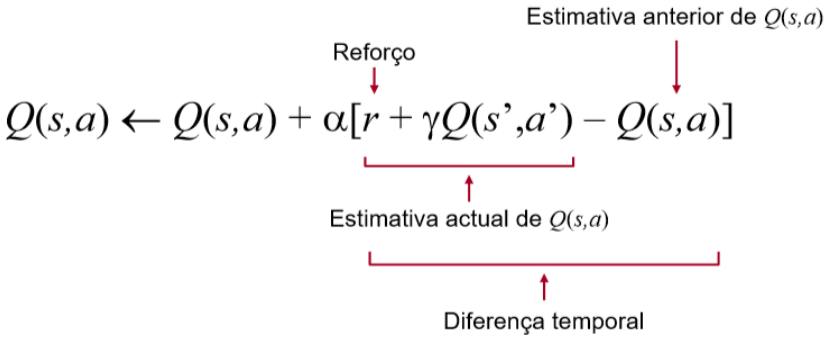

Figure 27: Diferença Temporal

2.20 Algoritmo SARSA

Este algoritmo previne grandes perdas e os caminhos podem não ser os melhores por ser uma exploração aleatória.

2.21 Algoritmo Q-Learning

Este caminho escolhe os melhores caminhos, pois os melhores são aqueles que maximizam a função Q, ou seja, a decisão é ótima.

3 Projeto Realizado

Esta seção aborda todo o projeto realizado, com as suas diversas conclusões. Visto que foram estudadas duas linguagens de programação, esta seção vai ser dividida na parte do projeto realizada em *Java* e a outra realizada em *Python*.

3.1 Projeto realizado em Java

Esta parte do projeto consiste na introdução ao estudo da Inteligência Artificial e da Engenharia de Software. Para a realização do mesmo foi criado um jogo com uma personagem virtual que interage com um jogador humano.

3.1.1 Ambiente

Para o ambiente vamos conter as seguintes características:

- **Discreto**: Só reage quando há *input* do utilizador
- **Determinístico**: o evento que interpreta uma ação é determinado pelo utilizador
- **Estático**: o ambiente muda apenas consoante um evento
- **Totalmente observável**: o agente conhece todas as ações
- **Agente único**:apenas uma personagem, que realiza ações consoante os eventos do ambiente

Para o jogo conhecer os diferentes eventos que este ambiente vai conter, foi criada e implementada a classe ***Ambiente***. No entanto para ser gerado um evento, é necessário o *input* do utilizador, e para isso foi criado o método ***private Evento gerarEvento()***.

3.1.2 Máquina de estados da Personagem

Visto que a personagem não será um agente capaz de aprender sozinho, esta irá conter uma máquina de estados associada. Para o desenvolvimento da máquina de estados, vai-se ter em conta os seguintes conceitos:

- **Estado**:representa uma situação de um problema. O estado atual representa o estado em que o agente se encontra no instante atual e de um estado é possível transitar para outro caso este não seja o estado final.
- **Evento**: representa o evento do ambiente
- **Ação**: representa a ação que vai executar consoante um determinado evento

Através dos conceitos acima, foi usada como referência a figura 28 para a implementação da classe ***MaquinaEstados***. No entanto, é necessário haver uma relação entre um **evento** e uma **ação**, ou seja, consoante o tipo de evento escolhido pelo *input* do utilizador, o agente vai gerar uma ação. Para isso foi criada a classe ***Transicao*** que realiza esta condição.

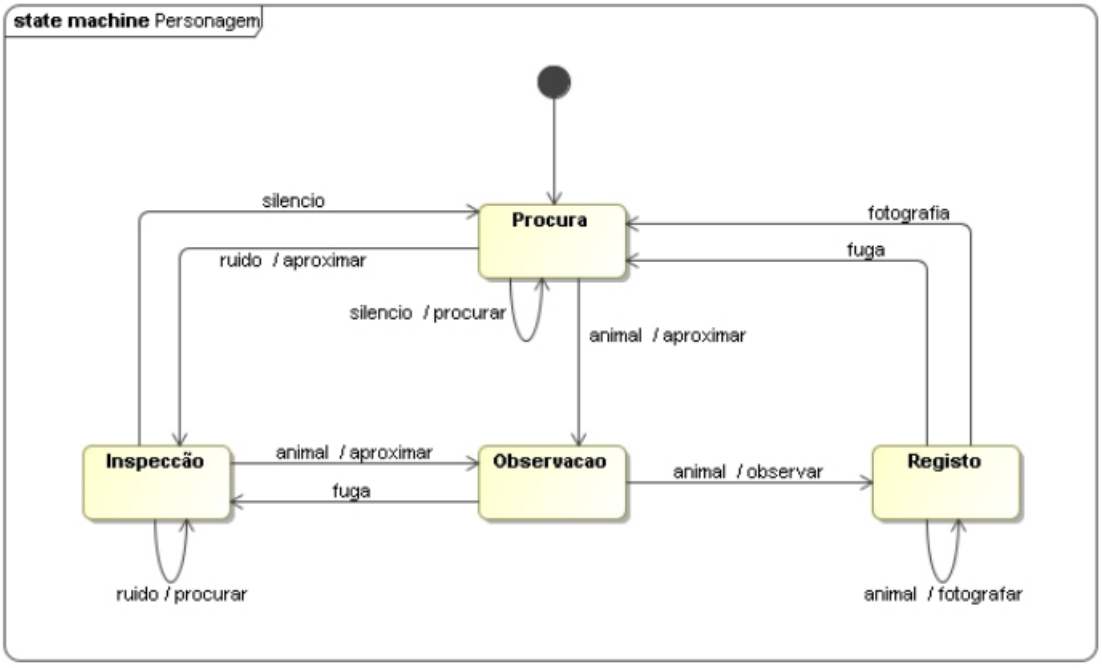

Figure 28: Máquina de estados do agente

Esta máquina de estados conhece 6 eventos e 4 ações. Em cada um dos estados a personagem encontra-se à espera do evento para realizar a ação necessária.

3.1.3 Personagem

Para finalizar a implementação do jogo, foi criada a classe ***Personagem***, no qual conhece o Ambiente e um **controlo**. Este controlo é aquele que permite controlar as ações, através da percepção das mesmas.

De seguida, passou-se para o cenário de testes. Para mostrar o correto funcionamento do jogo, colocou-se o estado como sendo "Procura".

O resultado final do jogo pode ser visto na figura 29

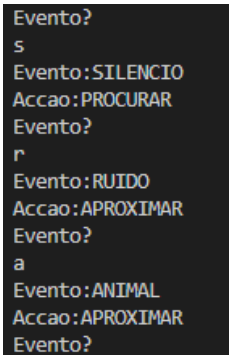

Figure 29: Resultado obtido quando o estado é "Procura"

Com esta parte do trabalho, foi possível consolidar as bases fundamentais da **inteligência artificial**. Foram compreendidos os conceitos de Estado e Ação, e como é possível desenvolver uma máquina de estados, com base em certos eventos. Um dos aspetos a destacar neste trabalho foi a **capacidade de abstração** tendo sido algo o docente se debruçou bastante, pois é fundamental um engenheiro de software pode observar diferentes problemas e em todos modular uma estratégia.

3.2 Projeto realizado em Python

3.2.1 Implementação de uma arquitura reativa com um agente reativo

No trabalho anterior, para o agente executar ações era necessária o *input* do utilizador, ou seja, o agente não mostrava um certo tipo de autonomia na execução das ações.

Para introduzir autonomia foi estudado o conceito de "arquitetura de agentes". A primeira arquitetura implementada neste trabalho consistia numa arquitetura reativa, podendo ser vista na figura 2. Este tipo de arquitetura vai permitir ao agente atuar consoante uma percepção, realizando uma determinada ação.

No entanto, é necessário perceber como esta arquitetura funciona. Um dos conceitos a ter em conta é **reação**. Uma reação permite ativar uma percepção, detetar um estímulo e ativar uma resposta, consoante a intensidade passada pelo estímulo, produzindo uma ação. Foi criada e implementada a classe ***Reaccao*** com o método ***activar(percepcao)***.

No entanto, dependendo do tipo de estímulo o agente reage de forma diferente. Assim, podemos dizer que um agente reativo pode ter múltiplas reações, e através do Mecanismo de seleção de ação é possível escolher de entre várias reações ativas ao mesmo tempo a reação que melhor se adequa à situação atual do agente. Foram

também criadas e implementadas dois tipos de mecanismos de seleção de ação: Hierarquia(figura 7), através da classe ***Hierarquia*** e Prioridade (figura 8), através da classe ***Prioridade***. Ambas as classes partilham o método ***selecionar_accao(accoes)*** e como o próprio nome indica, seleciona uma ação de entre uma lista de ações possíveis que o agente eventualmente vai realizar.

Para finalizar a implementação desta arquitetura, foram introduzidos comportamentos para o agente executar, esses comportamentos foram: **Recolher**, implementando a classe ***Recolher*** onde o agente recolhe os alvos à medida que se movimenta no ambiente, e **Explorar** implementando a classe ***Explorar*** onde o agente explora o ambiente, podendo encontrar alvos.

Na figura 30 é possível de ver o resultado obtido, quando o agente executa o comportamento Recolher.

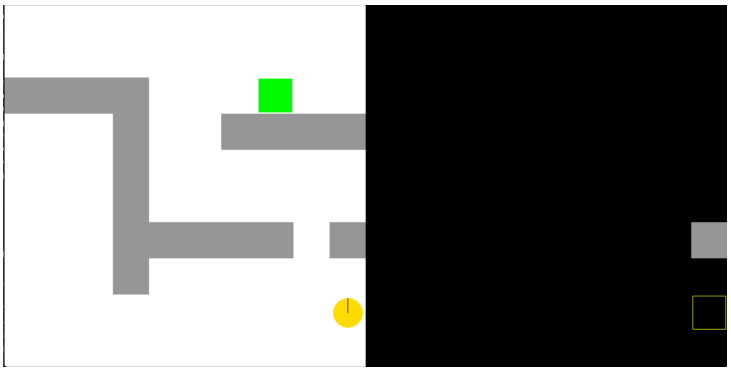

Figure 30: Resultado obtido quando o agente executa o comportamento "Recolher"

Em suma, foi implementada com sucesso a arquitetura reativa, assim como a compreender o conceito de reação, e como é possível efetuar um mecanismo de seleção de ação quando uma percepção pode ativar uma ou mais reações. Foram inseridos diferentes comportamentos para o agente realizar e ter uma noção do tipo de ambiente que se movimenta.

3.2.2 Implementação da procura espaços de estados

Após a implementação da arquitetura reativa, foi implementada a "Procura em espaços de estados" e os diferentes tipos de mecanismos de procura associados.

O funcionamento principal que vai permitir a PEE funcionar será a implementação do mecanismo de procura. Para isso foi criada a classe ***MecanismoProcura*** com o método ***resolver(problema)***.Este mecanismo será partilhado por todos as procuras, e para isso é **iniciada a fronteira de exploração**, onde cada procura

irá **resolver** o seu **problema**, contendo um **estado inicial** e um **estado final**, até ser encontrada a sua respetiva **solução**. Esta solução consiste numa **árvore** de **nós**, onde serão inseridos numa **fronteira** de nós que foram **explorados**.

De seguida foram implementados 5 tipos de mecanismos de procura, cada um associado a um método de procura que tipicamente diferem na estratégia de controlo.A implementação destes métodos de procura difere, tipicamente, no modo como os nós são inseridos na fronteira de exploração. Os tipos de procura implementados foram os seguintes:

- ProcuraLargura(classe ***ProcuraLarg***): os nós a explorar são inseridos no final da fronteira de exploração (memória FIFO (classe ***FronteiraFIFO***)
- ProcuraProfundidade(classe ***ProcuraProf***): os nós a explorar são inseridos no início da fronteira de exploração (memória LIFO (classe ***FronteiraLIFO***)
- ProcuraProfundidadeIterativa(classe ***ProcuraProfIter***): os nós a explorar são inseridos no início da fronteira de exploração. Este método de procura é uma extensão do método de procura em profundidade que limita a profundidade de procura máxima de forma iterativa
- ProcuraMelhorPrimeiro(classe ***ProcuraMelhorPrim***): neste método de procura a fronteira de exploração é ordenada por ordem crescente da função custo (prioridade), sendo explorado em primeiro lugar o nó com menor função custo (fronteira prioridade(classe ***FronteiraPrioridade***)).

A ProcuraAA e ProcuraSofrega estendem a ProcuraInformada e ProcuraCustoUniforme estende a ProcuraMelhorPrimeiro e implementam a função de valor (f(no) que corresponde ao método ***prioridade(no)***). Tal como descrito no capítulo 2.11.4 a procura de custo uniforme a função de valor corresponde ao custo acumulativo até ao estado do nó (valor real), na procura sôfrega (capítulo 2.11.7) corresponde apenas à heurística para o estado do nó e a procura A* (capítulo 2.11.8) corresponde à soma entre a heurística para o estado do nó e o custo acumulativo até ao estado do nó.

Para finalizar e visualizar os resultados, foi implementado um planeador de trajetos, para visualizar a solução desta procura. Foram usadas "Localidades" tendo como referência uma localidade inicial (estado inicial) e uma localidade final (estado final). Na figura 31 é possível de ver um dos cenários de teste para a procura testada.

Figure 31: Planeador com procura custo uniforme

3.2.3 Implementação da arquitetura deliberativa

Após ter sido implementado o planeador de trajetos, passou-se para a PEE com raciocínio automático. E para isso foi implementada uma arquitetura deliberativa, que pode ser vista na figura 19.

Para a implementação desta arquitetura, foi criada a classe ***ControloDelib*** onde existem os seguintes métodos que seguem a lógica que pode ser vista no capítulo 2.12 na lista 1:

- Observa o mundo e atualiza o modelo do mundo através da percepção (***_assimilar(per)***)
- Verifica se deve reconsiderar (***_reconsiderar()***), ou seja, se o modelo do mundo foi alterado (o número de objetivos) ou se não existe plano, o agente deve:
  1. Deliberar (***_deliberar()***)
  2. Planear (***(_planear()***)
- Por fim, o agente deve executar (***(_executar()***), recorrendo ao plano de ação.

Para finalizar a implementação desta classe, foi criado o método ***processar(percepcao)*** que permite ao agente processar as ações.

No entanto, para o agente se movimentar no mundo, deve conhecer a sua posição (posições X e Y). Foi implementada a classe ***OperadorMover***, e de entre os métodos implementados, método ***aplicar(estado)*** vai ser aquele que atualiza a posição do agente, e consoante essa atualização, é aplicado um novo estado.

Para finalizar a arquitetura foi implementado a classe de nome ***PlanPEE***. Este planeador recebe o tipo de procura que vai resolver o problema. Para testes, foi testada a procura A*, passando uma heurística. Foi também criado um método para verificar se o plano é válido, ou seja, permite verificar se existe uma solução e se o primeiro passo da solução corresponde ao estado atual do agente.

Na figura 32 é possível de ver o resultado obtido quando corremos um dos ambientes,

com mais do que um alvo. É de notar que na figura, a parte preta pode ser vista o agente com o percurso(solução) até recolher o alvo.

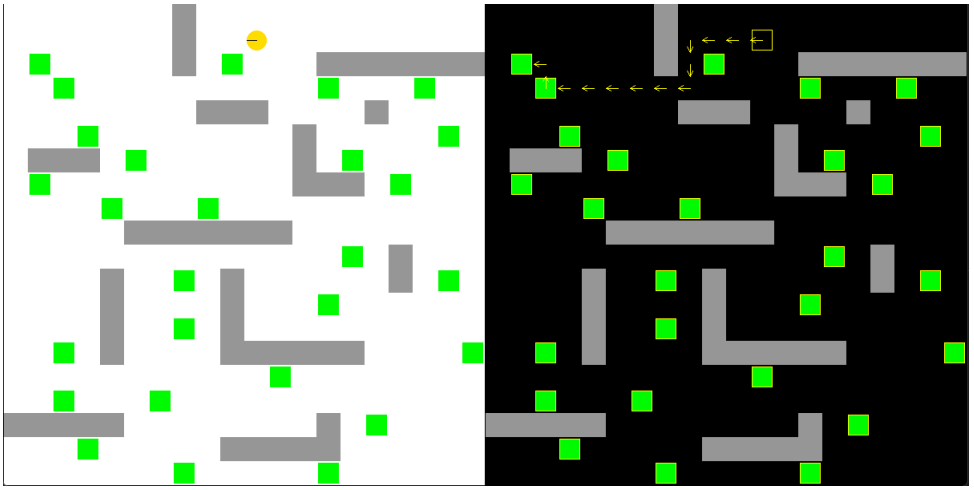

Figure 32: Procura em espaços de estados usando a procura A*

3.2.4 Implementação do processo de decisão de markov (pdm)

De seguida foi implementado o processo de decisão de markov. Nesta parte do projeto, foram construídos métodos para fazer o cálculo da política e da utilidade. A política vai servir para indicar a ação que o agente deve executare a utilidade permite representar o estado para onde o agente se deve movimentar (funcionando como uma probabilidade).

Foram implementadas as classes ***PDM*** e ***ModeloPDMPlan*** que implemetam as equações mencionadas no capítulo 2.14.1

De seguida foi criada a classe ***PlaneadorPDM*** para planear o agente com este processo de decisão. Na tentativa de testar esta parte, alguns dos ambientes demoravam imenso tempo a mostrar os resultados, pois o valor da utilidade estava a ser calculado para muitos alvos e diferentes caminhos. Por isso foi testado no ambiente onde existem poucos alvos para o agente recolher.

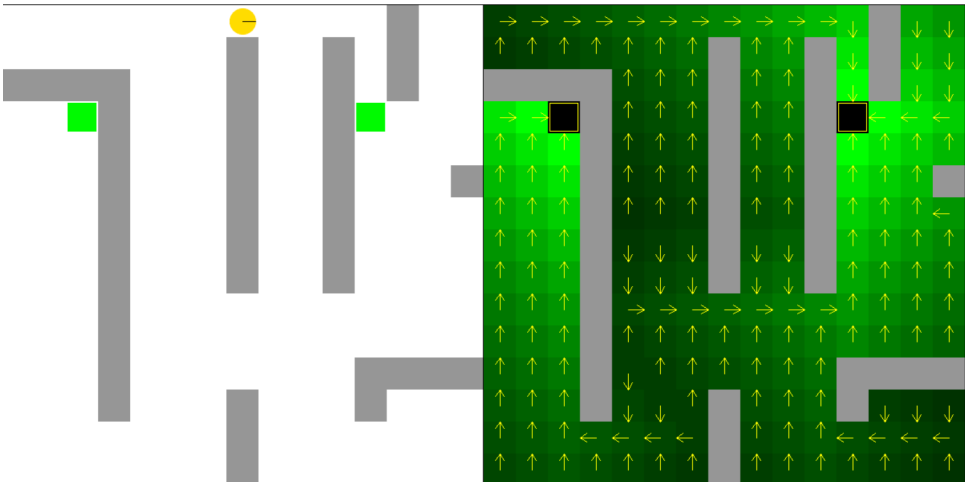

Figure 33: Resultado obtido em PDM

Como é possível de ver na figura 33 o valor de utilidade para os estados com alvos está representado a preto, uma vez que a utilidade é nula e os estados com maior utilidade são os adjacentes ao estado com alvo.

Também podemos observar que o agente desvia-se dos obstáculos, isto acontece porque a informação de que podem existir desvios nas ações foi incorporada no processo de decisão de Markov. Assim, ele aprendeu que se houver desvio e este estiver junto dos obstáculos, pode existir colisão.

Para concluir, este algoritmo mostrou bons resultados. Efetuando os diferentes cálculos da utilidade, foi possível ao agente usufruir do melhor caminho até ao alvo mais próximo, ao contrário da procura em espaços de estado, onde este recolhia os alvos consoante a sua ordem na lista dos objetivos.

No entanto, o PDM apresenta uma desvantagem, que consiste no nível computacional. O cálculo da utilidade pode tornar-se demorado e lento quando existe mais do que um alvo no ambiente. Já a procura em espaços de estados, como foi dito, contém uma lista de objetivos, tornando-se mais rápida a sua execução.

3.2.5 Implementação da aprendizagem por reforço (algoritmo Q-Learning)

A última parte do projeto focou-se no estudo da aprendizagem por reforço, e para isso foi implementado o algoritmo *Q-Learning*. Também surgiu a possibilidade de implementar o algoritmo *Sarsa*, no entanto o *Q-Learning* é aquele que apresenta melhores resultados, visto que é aquele algoritmo que encontra o melhor e mais curto caminho até o agente recolher o alvo.

Foi criada a classe ***AprendQ*** com o método ***aprender(s, a, r, sn)*** onde pode ser visto a realização desse algoritmo.

De seguida, visto que o agente já aprendeu o suficiente, foi introduzido o dilema do *Explorar/Aproveitar*. Para isso foi criada a classe ***SelAccaoEGreedy*** onde foi criado o método ***seleccionar_accao(s)***. Este método retorna uma a ação que o agente vai realizar, neste caso vai poder explorar ou aproveitar.

Para finalizar o processo de aprendizagem por reforço, foi criada a classe ***ControloAprendRef*** onde vai ser possível processar o controlo do agente. Foi criado o método ***processar(percepcao)*** onde vai ser feito o processamento do agente. Neste método é conhecido as posições do agente(estado), seleciona a ação e gerar reforço. Sempre que o agente encontra um estado e uma ação, ele vai aprender consoante esses estados e ações.

No entanto, é necessário fornecer ao agente um reforço. Para isso foi criado o método ***_gerar_reforco(,percepcao)***. Este método permite gerar reforço, consoante a situação. Caso ele encontre o alvo, gera reforço positivo, caso encontre um obstáculo, gera reforço negativo.

Com as classes finalizadas, passou-se para a parte de testes. Na figura 34 é possível de ver a aprendizagem por reforço a ser testado num dos ambientes.

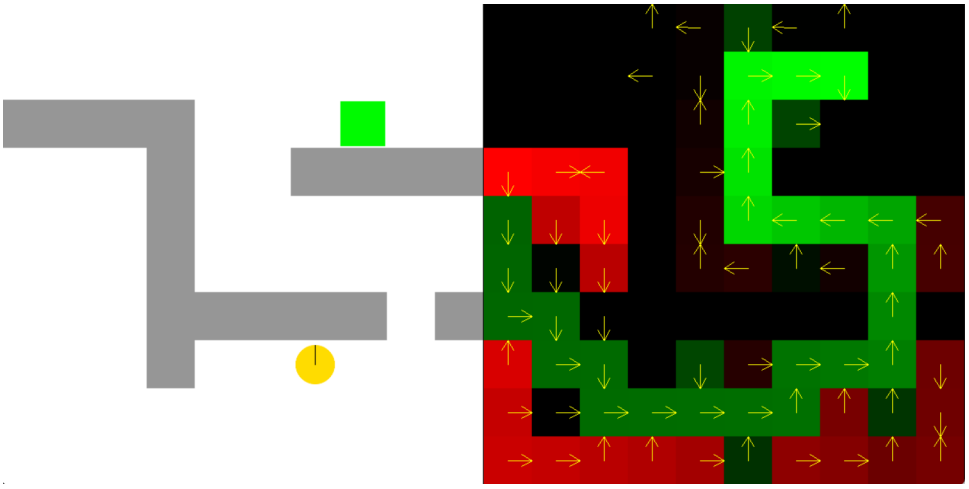

Figure 34: Resultado obtido na aprendizagem por reforço

Em suma, este algoritmo mostrou bons resultados. O agente no início como não conhece nada do mundo, está a **explorar** até encontrar o alvo. Sempre que encontra um obstáculo à sua frente, aprende e opta por explorar outro caminho. Assim que encontra o alvo, o agente reinicia o seu processo de aprendizagem,descobrindo o

melhor caminho até ao alvo. Sempre que encontra o alvo, o agente reincia o seu processo de aprendizagem, este tipo de processo chama-se **episódio**. Se fosse preciso ao agente explorar com mais detalhe o ambiente em seu redor, podia-se alterar o valor do *epsilon*. Como se pode ver na figura, a parte a verde, representa o melhor trajeto que o agente aprendeu para chegar ao alvo.

4 Propostas de revisão do projeto realizado

Em termos de propostas de revisão, surgiram algumas entregas onde foi efetuado apenas o código estrutural. A implementação de alguns métodos não foram realizados da melhor maneira possível, pois tinha surgido um problema que na altura não havia possibilidade de resolver.

No entanto, sempre que surgia uma nova entrega, o projeto era sempre melhorado e atualizado de acordo com a respetiva matéria lecionada nas aulas. O código efetuado para todo o projeto tinha sempre o objetivo obter o melhor desempenho e visualizar o resultado pretendido.

Relativamente às aulas dadas pelo docente, sempre foram interessantes e motivantes para a realização do projeto. Todas as dúvidas eram esclarecidas sempre que havia necessidade para tal.

Gostaria de apontar também o facto de não haver muito material de apoio. Os *powerpoints* fornecidos para a disciplina de Inteligência Artificial Sistemas Autónomos, apesar de ajudarem bastante no estudo e apoio ao projeto, podiam conter mais alguma informação, tornando-se por vezes, difícil de incorporar a matéria dentro do projeto.

5 Conclusão

Dada a conclusão de todos os trabalhos práticos da unidade curricular de Inteligência Artificial para Sistemas Autónomos foram adquiridos importantes conhecimentos e conceitos relacionados com a Inteligência artificial.

Foram desenvolvidas capacidades de identificação de conceitos basilares de uma das maiores áreas científicas e computacionais da atualidade. Foram definidas as noções de inteligência e todo o processo envolvente, sendo ainda definido este conceito.

Estudaram-se modelos de representação interna, deliberação e raciocínio no contexto de arquitetura de agente. Foi ainda estudada a concretizaçãao de conceitos de arquitetura deliberativa e de raciocínio prático no contexto desse modelo.

Com os conhecimentos adquiridos foi possível resolver problemas com base no conceito de agente, tendo como base arquiteturas ora deliberativas, ora reativas.

Foi desenvolvida a representação e resolução de problemas tendo por base a procura em espaço de estados, aplicando diversos métodos de procura não informados, como a procura em profundidade, procura em largura, procura em profundidade iterativa e procura custo uniforme. Estes métodos foram estudados tendo sido caraterizados em termos computacionais bem como as respetivas limitações.

Importantes conceitos como o de função heurística foram descritos no contexto da procura em espaço de estados, sendo atribuído valor seguindo a esta função. Dentro do mesmo contexto foram também definidos métodos de procura informada, nomeadamente a procura A*.

Foram estudados Algoritmos como o SARSA e Q-Learning como suporte de aprendizagem por reforço. Todo este processo levou à correta representação e resolução de problemas sob a base da aprendizagem por reforço, bem como a caracterização.

Deste modo é seguro de dizer que todos os conhecimentos adquiridos e estudados ao longo da unidade curricular de Inteligência Artificial para Sistemas Autónomos foi adquirida com sucesso.

6 Bibliografia

- L. Morgado, Engenharia de Software: Introdução, Lisboa: ISEL, 2021.
- L. Morgado, Arquitetura de Agentes Deliberativos, Lisboa: ISEL, 2021.
- L. Morgado, Linguagem UML: Introdução, Lisboa: ISEL, 2021.
- L. Morgado, Agentes Inteligentes: Introdução, Lisboa: ISEL, 2021.
- L. Morgado, Planeamento Automático com base em PEE, Lisboa: ISEL, 2022.
- H.-E. Ericksson, M. Penker, B. Lyons e D. Fado, UML 2 Toolkit, Indianapolis: Wiley, 2004, pp. 1, 11-13, 24-31