Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia Eletrónica e de Computadores

Relatório Final

Inteligência Artificial para Sistemas Autónomos

Docente:

Luis Morgado

Aluno:

Ricardo Pinto N.º 47673

junho de 2024

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

Este relatório detalha um estudo abrangente e a implementação subsequente de um projeto que se encontra na intersecção entre a inteligência artificial para sistemas autónomos (IASA) e engenharia de software. O documento inicia com um enquadramento teórico sólido que visa oferecer uma visão clara sobre os conceitos fundamentais de I.A., e as práticas essenciais da engenharia de software que sustentam a construção de softwares robustos e eficientes.

Este relatório serve de complemento ao projeto, visto que junta todo o enquandramento teórico necessário para compreender o código. No entanto, este relatório não substitui o projeto em si, e deve ser lido (especialmente a seguir ao enquadramento teórico) acompanhado do projeto.

Foram realizadas 4 partes do projecto, sendo estas:

Parte 1 - Agente reactivo em Java. Desenvolvido o agente e ambiente. O agente atua sobre o ambiente de um jogo, onde o objectivo é fotografar animais.

Parte 2 - Biblioteca para desenvolvimento de agentes reactivos em Python. Modulação de um agente reactivo para encontrar alvos e evitar paredes.

Parte 3 - Biblioteca para desenvolvimento de agentes deliberativos com base em PEE em Python. Modulação de um agente deliberativo de contagem.

Parte 4 - Modulação de um agente deliberativo com base em PEE para para encontrar alvos e evitar paredes. Biblioteca para desenvolvimento de agentes deliberativos com base em PDM em Python. Modulação de um agente deliberativo com base em PDM para para encontrar alvos e evitar paredes.

No final o objectivo é ter uma biblioteca de inteligência artificial para sistemas autónomos bem implementada e testada capaz de ser usada para vários problemas diferentes.

2 Enquadramento teórico

2.1 Inteligência Artificial

A inteligência artificial é uma área científica que estuda o desenvolvimento de sistemas computacionais capazes de comportamento inteligente, que adopta duas principais perpectivas: analítica e sintética.

Analítica - por via empírica (baseada na prática, com experiências e observações) desenvolver modelos e teorias para explicar os fenómenos observados e reproduzir esses fenómenos e sistemas artificiais (dispositivos criados para o efeito).

Sintética - com base nesses modelos e teorias, desenvolver sistemas capazes de apresentar características e comportamentos associados ao conceito de inteligência.

Existem 3 principais paradigmas da inteligência artificial, sendo eles:

- Simbólico
- Conexionista
- Comportamental

2.1.1 Paradigma Simbólico

Um símbolo é um elemento discreto de representação de informação. Estes são centrais no processamento de informação e na representação de conceitos relacionados com o domínio de um problema sob a forma de estruturas simbólicas. Neste paradigma, a inteligência é o resultado da acção de processos computacionais sobre estruturas simbólicas.

É importante notar que os símbolos não têm um significado intrínseco. O seu significado deriva das relações entre estruturas simbólicas bem como das relações dessas estruturas com as entradas e saídas do sistema. Esta relação é chamada de ancoragem simbólica, pois o significado é ancorado nas observações e conceitos referenciados.

Num sistema artificial o significado é construído através de relacionamentos entre simbolos.

2.1.2 Paradigma Conexionista

A inteligência é uma propriedade emergente das interacções de um número elevado de unidades elementares de processamento interligadas entre si. Neste paradigma o processamento de informação é baseado em redes de unidades de processamento elementares (designadas neurónios) interligadas entre si, sendo a informação mantida e processada nessas redes de forma distribuída, em particular nas ligações entre neurónios, em vez de símbolos. Este paradigma é inspirado no conhecimento atual do funcionamento do cérebro humano.

2.1.3 Paradigma Comportamental

A inteligência resulta do comportamento individual e conjunto de múltiplos sistemas a diferentes escalas de organização tendo por base relações entre estímulos e respostas. No paradigma comportamental o processamento de informação é baseado em relações entre estímulos e respostas, modeladas sob a forma de reacções e de comportamentos (conjuntos estruturados de reacções).

2.1.4 Agente Inteligente

Um sistema autónomo descreve-se pela independência, ou seja, pela capacidade de existir e funcionar sem depender de outros sistemas.

Um sistema inteligente descreve-se por possuir cognição e racionalidade.

Cognição - É o processo pelo qual um sistema adquire, processa, armaneza e utiliza informação.

Racionalidade - É a capacidade de decidir no sentido de conseguir o melhor resultado possível perante os objectivos que se pretende obter. Ou seja, um sistema racional escolhe a acção que maximiza o valor esperado da medida de desempenho dado o conhecimento disponível sobre o ambiente, percepções e acções.

É de notar que para um sistema ser autónomo, este não necessita de ser inteligente, mas um sistema inteligente implica que seja também autónomo, visto que a autonomia é uma característica da inteligência.

Um sistema autónomo inteligente opera num ciclo realimentado percepção-processamento-acção, através do qual é realizado o controlo da função do sistema de modo a concretizar a finalidade desse sistema.

Um agente inteligente é a representação computacional de um sistema autónomo inteligente. Este segue o mesmo ciclo de percepção-processamento-acção. A percepção é feita através de sensores ligados ao ambiente, e a acção é feita sobre actuadores ligados ao ambiente.

Um agente inteligente pode ter as seguintes características, dependendo da sua arquitetura:

Autonomia - Capacidade de um sistema operar por si próprio, de modo independente de outros sistemas.

Reactividade - Capacidade de reagir sobre estímulos do ambiente.

Pró-Actividade - Capacidade de atuar sobre o ambiente, independente dos estímulos.

Sociabilidade - Capacidade de comunicar com outros agentes, de forma a concretizar objectivos individuais ou comuns a outros agentes.

Estas características estão associadas à concretização da finalidade do agente, ou seja, do seu objectivo, expresso na função que realiza.

2.1.5 Representação do Ambiente

A representação do ambiente é um elemento de suporte do processamento interno em alguns modelos de agente inteligente. Essas representações podem ter diferentes níveis de complexidade.

Um ambiente tem diversas propriedades:

Discreto/Contínuo - Se for discreto (como no caso de um ambiente virtual), este acontece num tempo não contínuo, ou seja, os seus pontos de informação têm um intervalo temporal. Se for contínuo (como no caso de um ambiente real), este acontece num tempo contínuo, ou seja, existem infinitos pontos de informação.

Determinístico/Estocástico - Se for determinístico, o ambiente opera sempre da mesma forma, ou seja, cada acção leva a um resultado expectável. Se for estocástico, o ambiente opera de forma aleatória, ou seja, mesmo sabendo toda a informação do ambiente, não é possivel determinar qual será o resultado de uma acção sobre este.

Estático/Dinâmico - Um ambiente estático não varia, enquanto que um ambiente dinâmico varia.

Totalmente/Parcialmente observável - Se um ambiente for totalmente observável, um agente poderá ter toda a informação contida no ambiente. Se o ambiente for apenas parcialmente observável, apenas parte da informação do ambiente está disponível para o agente.

Agente único/Múltiplos agentes - Um ambiente poderá ter apenas um agente a atuar sobre este, um vários.

2.1.6 Raciocínio Automático

O raciocínio automático refere-se à capacidade de resolver de forma automática um problema, tendo como base uma representação de conhecimento do domínio do mesmo, produzindo uma solução a partir da exploração das diversas alternativas possíveis.

Para conseguir produzir uma solução, o processo de raciocínio automático envolve 2 tipos de atividades principais, sendo estas a exploração de opções possíveis e a avaliação das opções para decidir qual a melhor.

Exploração de opções possíveis - Deve usar um raciocínio prospectivo, ou seja, deve antecipar o que pode acontecer, e requer uma simulação interna do domínio do problema.

Avaliação de opções - As opções são avaliadas através do seu o custo, ou seja, os recursos necessários para a opção, e o seu valor, ou seja, os ganhos ou perdas da opção, que podem ser medidos, por exemplo, em termos de utilidade.

A representação do conhecimento do domínio do problema implica que haja uma conversão da informação concreta do problema, para estruturas simbólicas que servem como base à simulação para exploração e avaliação das opções possíveis.

Esta conversão envolve 2 processos: codificação e descodificação. A codificação trata-se da conversão da informação concreta do problema em estruturas simbólicas internas, enquanto que a descodificação é a operação inversa, ou seja, converte as estruturas simbólicas em informações concretas do problema.

Sendo necessário guardar estruturas simbólicas para realizar a exploração de opções, é necessário estabelecer como é que estas funcionam. Estas estruturas devem ter a informação necessária para resolver o problema, e são designadas de "modelo do problema". O modelo do problema é representado através de:

Estado - Representa uma configuração possível do problema. É único para um problema.

Operador - Representa uma acção que produz uma transformação do estado. Ao operar sobre um estado, é produzida uma transição que corresponde à geração de um novo estado.

Tendo esta informação, é possível descrever o raciocínio automático como tendo 4 requisitos:

Definição de Estado

Definição dos Operadores

Descrição do Problema - Um problema pode ser descrito através de 3 elementos:
Estado inicial
Operadores
Objectivos

Mecanismo de Raciocínio - Mecanismo que faz a exploração e avaliação das opções possíveis, usando como base a descrição do problema, e que mantém um modelo do problema interno.

Um espaço de estados é um conjunto de estados e de transições de estado representado sob a forma de um grafo, onde cada vértice corresponde a um estado e cada aresta corresponde a uma transição entre estados.

2.1.7 Procura em Espaços de Estados

A procura em espaços de estados é um método de resolução de problemas de planeamento, onde se pretende encontrar uma sequência de estados e operadores que levem de um estado inicial a um estado final (estado objectivo). Uma solução deste tipo de problemas é um percurso no espaço de estados, que liga o estado inicial a um estado objectivo.

O processo de procura ocorre por exploração do espaço de estados a partir do estado inicial. Esta exploração ocorre da seguinte forma:

Em cada passo do processamento, ocorre o seguinte:

- Primeiro é verificado se o estado atual é um estado objectivo. Caso seja, o processamento termina e é retornado o percurso do estado inicial ao estado atual. Caso não seja, continua o processamento.
- O estado atual é expandido, sendo gerados todos os estados sucessores por aplicação de todos os operadores possíveis.

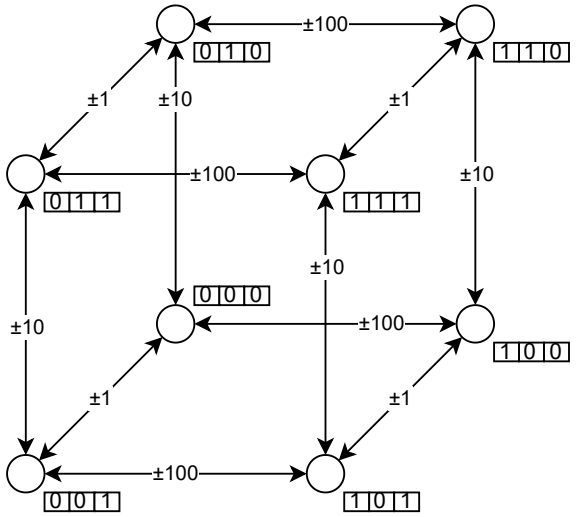

Figura 1: Exemplo de um espaço de estados, onde o problema corresponde à contagem de um número binário com 3 casas (ou seja, de 0 a 7), onde é possível incrementar/decrementar 1, 2 ou 4.

- É processado cada estado sucessor.
- Se não existiram mais estados sucessores, o processo de procura termina, não existindo solução.

Para guardar a informação gerada em cada passo da procura, é mantida uma árvore de procura quem mantém a informação relativa a cada transição de estado explorada. Esta árvore relaciona cada nó com o seu antecessor, e mantém a informação do estado correspondente a cada nó, e o operador que originou a transição do estado antecessor para o estado do nó.

É importante conhecer alguns conceitos sobre a exploração em árvores de procura:

Raiz da árvore - Nó correspondente ao estado inicial.

Fronteira de exploração - Estrutura de dados com relação de ordem que mantém os nós a explorar. O critério de ordenação da fronteira é o que determina a estratégia de controlo de procura.

Nó fechado - Nó que já foi expandido.

Nó aberto - Nó por expandir.

2.1.7.1 Métodos de procura

Existem vários métodos de executar a procura no espaço de estados. Alguns dos aspectos a considerar para cada método de procura são:

Completo - Método que garante que, caso exista uma solução, esta é encontrada.

Óptimo - Método que, existindo várias soluções, garante que encontra a melhor.

Complexidade Espacial - Memória necessária para encontrar uma solução.

Complexidade Temporal - Tempo necessário para encontrar uma solução.

Além disso, é necessário conhecer também os parâmetros de caracterização de um método de procura:

Factor de Ramificação (Branching Factor) - Número máximo de sucessores para um qualquer estado. Designado pela letra "b".

Profundidade da Procura (Search Depth) - Profundidade do nó objectivo menos profundo da árvore de procura, ou seja, a dimensão do percurso entre o estado inicial e esse estado objectivo. Designado pela letra "d".

2.1.7.1.1 Procura geral em grafos

Todos os métodos de procura estudados são métodos de procura em grafos, mais concretamento, procuras no grafo do espaço de estados.

Neste tipo de procuras, é necessário eliminar os nós correspondentes a estados repetidos, e para este efeito, é necessário verificar se um novo nó sucessor corresponde a um estado que já foi previamente explorado, e se for, apenas o nó com o menor custo deve ser mantido, e os outros eliminados.

Para esta verificação, é preciso ter em conta tanto os nós abertos, como os fechados.

De modo a facilitar o processamento de nós repetidos, pode ser mantida uma única memória de nós previamentes explorados, que inclui ambos os nós abertos e os nós fechados. É importante notar que apesar de esta estrutura manter uma coleção dos nós abertos, esta não substitui a fronteira de exploração, servindo apenas para facilitar o processamento de nós repetidos.

Para um processamento mais eficiente, esta memória deve ser indexada por estado, para que haja um acesso eficiente na verificação dos nós já explorados.

2.1.7.1.2 Procura em profundidade (Depth-First Search)

Na procura em profundidade, o critério de exploração é a maior profundidade, ou seja, os nós mais recentes são explorados primeiro (os primeiros nós da fronteira), e os nós gerados são inseridos no início da fronteira. Isto representa uma fronteira do tipo LIFO (Last In, First Out), ou seja, onde o último nó a ser gerado é o primeiro a ser explorado.

Esta procura maximiza a profundidade do ramo corrente de procura, apenas mudando de ramo quando não for possível gerar mais estados sucessores para o estado atual.

Este tipo de procura ocupa menos memória que a procura por largura, mas não é nem óptima nem completa.

Complexidade Temporal: $O(b^m)$

Complexidade Espacial: $O(b * m)$

b - factor de ramificação.

m - profundidade máxima da árvore de procura.

Esta procura tem 2 sub-tipos:

Pocura em profundidade limitada (Depth-Limited Search) - Procura em profundidade que limita a procura a uma profundidade máxima.

Pocura em profundidade iterativa (Iterative Deepening Search) - Procura que realiza sucessivas procuras em profundidade limitada, com uma profundidade máxima iterativamente crescente.

2.1.7.1.3 Procura em largura (Breadth-First Search)

Na procura em largura, o critério de exploração é a menor profundidade, ou seja, os nós mais antigos são explorados primeiro (os últimos nós da fronteira), e os nós gerados são inseridos no fim da fronteira. Isto representa uma fronteira do tipo FIFO (First In, First Out), ou seja, onde o primeiro nó a ser gerado é o primeiro a ser explorado.

Esta procura explora exaustivamente cada nível de procura antes de explorar um nó a um nível de maior profundidade, apenas aumentando a profundidade quando não por possível gerar mais nós na profundidade atual.

Este tipo de procura ocupa mais memória que a procura por profundidade, mas em contra-partida, é óptima e completa.

Complexidade Temporal: $O(b^d)$

Complexidade Espacial: $O(b * d)$

b - factor de ramificação.

d - dimensão da solução.

2.1.7.1.4 Procura melhor-primeiro (Best-First)

Na procura melhor-primeiro,existe uma função f(n) que é capaz de avaliar um nó, dando a este uma prioridade. Através dessa avaliação, a fronteira de exploração é ordenada por ordem crescente da prioridade de cada nó.

Tipicamente, f(n) corresponde a uma estimativa do custo da solução através do nó. Ou seja, quanto menor o valor de f(n), mais promissor é o nó.

2.1.7.1.5 Procura de custo uniforme

A procura de custo uniforme trata-se de um caso particular da procura melhor-primeiro, onde f(n) corresponde ao custo do nó, ou seja, f(n) corresponde ao custo do percurso desde o estado inicial até ao estado do nó.

2.1.7.1.6 Comparação entre as diferentes procuras

Para avaliar as diferentes procuras, é possivel olhar para as diferentes métricas e decidir qual destas melhor se aplica ao problema em questão.

Método de Procura	Complexidade Temporal	Complexidade Espacial	Óptimo	Completo
Profundidade	$O(b^m)$	$O(b * m)$	Não	Não
Profundidade Limitada	$O(b^l)$	$O(b * l)$	Não	Não
Profundidade Iterativa	$O(b^d)$	$O(b * d)$	Sim	Sim
Largura	$O(b^d)$	$O(b^d)$	Sim	Sim
Custo Uniforme	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$	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 ($\epsilon > 0$)

Tabela 1: Comparação entre os diferentes métodos de procura

2.1.7.2 Métodos de Procura Informada

Todos os métodos até agora abordados foram métodos de procura não informada, visto que não utilizam o conhecimento do domínio do problema para otimizar a sua procura, sendo que apenas fazem uma exploração exaustiva do espaço de estados.

Pelo contrário, os métodos de procura informada fazem uma procura guiada, ou seja, tiram partido do conhecimento do domínio do problema para ordenar a sua fronteira de exploração, fazendo uma exploração selectiva do espaço de estados.

2.1.7.2.1 Procura melhor-primeiro (Best-First)

A procura melhor-primeiro pode também ser usada como um método de procura informada, definindo f(n) baseado em estimativas de custo, ou seja, com base em herísticas, que são métodos rápidos de estimação de valores ou de resolução de problemas.

Uma função de heurística (cujo simbolo é h(n)) representa uma estimativa do custo do percurso do nó até ao nó objectivo (sendo uma estimativa, o valor desta pode não ser igual ao valor real, cujo simbolo é g(n)).

É de valor notar que o valor é completamente independente do percurso anteriormente realizado, a heurística depende apenas do estado associado ao nó, e do objectivo.

Sendo assim, podemos definir 3 principais variantes da procura melhor-primeiro:

f(n) = g(n) - Procura de Custo Uniforme. Esta minimiza o custo acumulado até cada nó explorado, mas é uma procura não informada, logo não tira partido do conhecimento do domínio do problema expresso através da função h(n).

f(n) = h(n) - Procura Sôfrega. Esta minimiza a estimativa de custo para atingir o objectivo, não tendo em conta o percurso explorado.

f(n) = g(n) + h(n) - Procura A*. Esta minimiza o custo global, ou seja, o custo acumulado até ao nó + a estimativa de custo até ao objectivo.

2.1.7.2.2 Procura Sôfrega

Este tipo de procura apresenta uma menor complexidade computacional e é completa, mas pode apresentar uma solução sub-óptima.

2.1.7.2.3 Heurística Admissível e Consistente

Uma heurística admissível é uma heurística em que a estimativa de custo é sempre inferior ou igual ao custo efectivo mínimo, ou seja, o valor estimado deve estar entre 0 e o custo mínimo do nó até ao objectivo através do percurso óptimo.

No caso geral, é possivel obter uma heurística admissível através da remoção de restrições associadas ao problema.

Uma heurística consistente (ou monótona) é uma heurística onde a estimativa de custo de n para objectivo é menor do que o custo real de transição de n para n' mais a estimativa de custo de n' para objectivo, para qualquer n. Esta heurística é obrigatóriamente admissível, mas uma heurística admissível pode não ser consistente.

2.1.7.2.4 Procura A*

Este tipo de procura é completo e óptimo, desde que utilize uma heurística consistente.

Esta procura garante que qualquer nó selecionado está num percurso óptimo, pois qualquer outro caminho terá um custo maior ou igual.

Apesar de esta procura ser de eficiência óptima, esta não resolve o problema da complexidade combinatória, isto é, o número de nós expandidos continua a ser uma função exponencial da dimensão do percurso até ao objectivo.

2.1.8 Arquitetura de Agentes

Existem 3 modelos usados na arquitetura de agentes:

Modelo Reactivo - Este modelo segue o paradigma comportamental. Neste modelo o comportamento do sistema é gerado de forma reactiva, com base em associações entre estímulos (referentes às percepções) e respostas.

Modelo Deliberativo - Este modelo segue o paradigma simbólico. Neste modelo o comportamento do sistema é gerado com base em mecanismos de deliberação (raciocínio e tomada de decisão), utilizando representações internas que incluem a representação explícita de objectivos.

Modelo Híbrido - Neste modelo o comportamento do sistema é gerado com base em processamento que integra as vertentes reactiva e deliberativa. Este modelo não é alvo de estudo da Unidade Curricular.

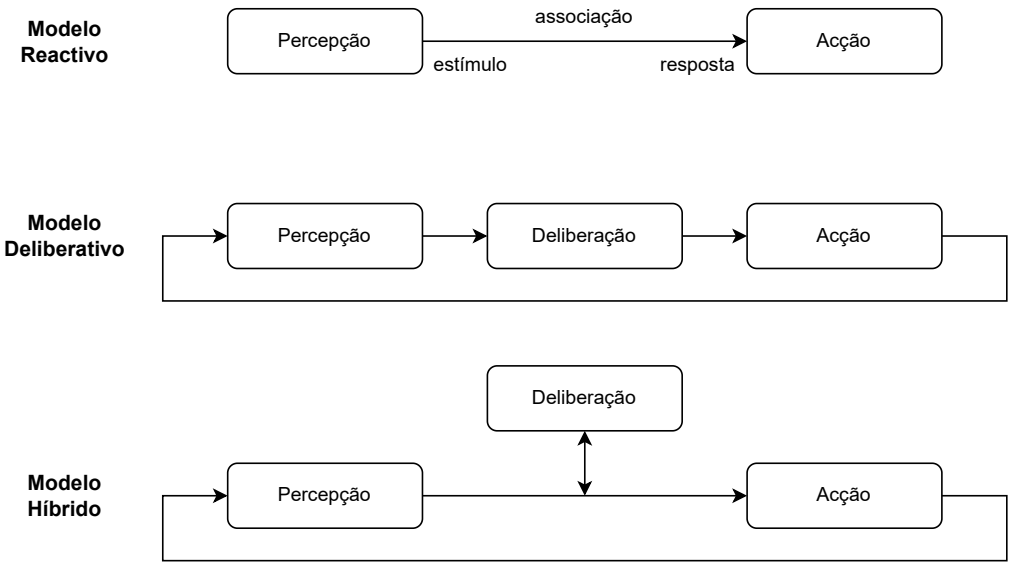

Figura 2: Diagramas dos 3 modelos usadas na arquitetura de agentes

2.1.8.1 Arquitetura de Agentes Reactivos

Neste tipo de arquitetura, o comportamento do sistema é gerado de forma reativa, baseando-se em associações entre estímulos. Os objetivos não estão representados de forma explicita, pelo contrário, encontram-se de forma implícita nas associações estímulo-resposta que definem o comportamento do agente. Existe um acoplamento forte com o ambiente, em alguns casos directo, através de associações entre os estímulos derivados das percepções e a respostas que produzem acções. Neste tipo de arquitetura, as acções são directamente activadas em função das percepções. Não são utilizadas representações internas do mundo, as respostas são rápidas mediante alterações no ambiente, e são fixas e predefinidas aos estímulos do ambiente, ou seja, para um mesmo estímulo, é sempre atuada a mesma acção.

No caso de reacções simples, as associações estímulo-resposta podem ser definidos através de regras SE-ENTÃO. Por exemplo, SE for avistado um animal, ENTÃO tirar fotografia. No entanto, as reacções podem envolver processamentos mais complexos incluindo a manutenção de estado interno com base em mecanismos de memória. Nessa situação, os vários elementos envolvidos numa reacção devem ser modularizados, de forma a poderem ser organizados de forma versátil e a encapsular a complexidade necessária à sua função. Em particular, devem ser definidos os seguintes elementos:

Estímulo - Define informação activadora de uma reacção.

Resposta - Define uma resposta a estímulos, em termos de acção a realizar e da respectiva prioridade.

Reacção - Módulo que associa estímulos a respostas.

Nesta arquitetura, o agente reactivo é composto por conjuntos de reacções, as quais devem ser organizadas de forma modular em módulos comportamentais designados comportamentos, de modo a reduzir a complexidade e a facilitar o desenvolvimento e manutenção do agente. Um comportamento é um conjunto de reacções relacionadas

entre si no sentido de produzirem um resultado específico, por exemplo, evitar um obstáculo. Um comportamento realiza de forma modular e coesa o encapsulamento das reacções internas, contribuindo assim para reduzir o acoplamento e a complexidade da arquitectura de um agente reactivo. Um comportamento pode também ser composto por outros comportamentos, tratando-se assim de um comportamento composto. Este tipo de comportamentos necessita de um mecanismo de selecção de acção para determinar a acção a realizar em função das respostas dos vários comportamentos internos. Existem vários mecanismos de selecção, que podem executar acções em paralelo, escolher acções por prioridade, ou até combinar acções. Dois tipos de mecanismos de selecção de acção que escolhem uma acção por prioridade são: a hierarquia, e a prioridade.

Hierarquia - Os comportamentos estão organizados numa hierarquia fixa de subsunção (supressão e substituição)

Prioridade - As respostas são seleccionadas de acordo com uma prioridade associada que varia ao longo da execução.

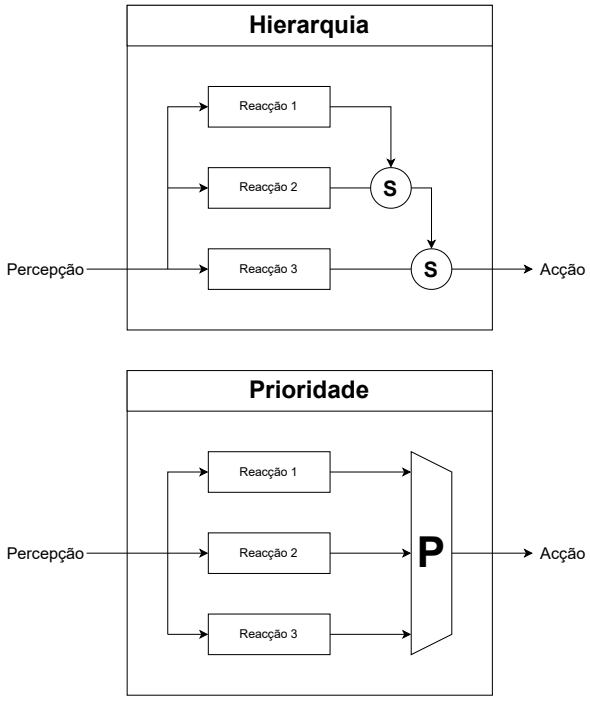

Figura 3: Diagrama dos mecanismos de seleção: hierarquia e prioridade

Na concretização de uma arquitectura de agente, o processamento interno que relaciona percepções com acções, pode ser modularizado com base num módulo de controlo. No caso da arquitectura de um agente reactivo, esse controlo será um controlo reactivo, em que o processar das percepções é realizado com base num módulo comportamental, também designado comportamento, o qual representa o comportamento geral do agente que pode ser constituído por diferentes sub-comportamentos.

Para modelação dos comportamentos de um agente reactivo, deve ser realizada a análise do domínio do problema a resolver para identificação de diferentes aspectos, como a informação que o agente recebe do ambiente (para modelação das percepções),

que tipo de acções a realizar e, em particular, qual a finalidade do agente, definida em termos de objectivos concretos. A concretização dos objectivos identificados, deve posteriormente ser modelada sob a forma de comportamentos, os quais, por sua vez, podem ser definidos em termos de sub-objectivos que serão realizados sob a forma de sub-comportamentos, de forma modular.

A arquitetura reativa pode ser implementada com memória, e nesse caso, as reacções para além de dependerem das percepções, dependem também da memoria de percepções anteriores para gerar as ações, as reacções podem envolver não apenas percepções, mas também estado interno (memória). As vantagens de manter o estado interno são:

- pode produzir todo o tipo de comportamento.
- Pode representar dinâmicas temporais, evoluindo o estado, e a resposta ser definida tendo em conta percepções anteriores.
- Possibilidade de comportamentos mais complexos baseados na evolução de estado.
- Capacidade de lidar com situações de falha por exploração de acções não realizadas anteriormente.

Algumas desvantagens são:

- Elevada complexidade espacial (memória).
- Elevada complexidade computacional (manter os estados).
- Mesmo com a manutenção de estado, as arquitecturas reactivas não suportam representações complexas, nem exploram planos alternativos de acção.

2.1.8.2 Arquitetura de Agentes Deliberativos

Como foi anteriormente explorado, os agentes baseados em arquitecturas reactivas não são capazes de anticipar o futuro, sendo esta uma habilidade dos agentes deliberativos.

Numa arquitectura deliberativa, é a memória que desempenha o papel central na geração do comportamento do agente. É a memória que é o suporte de representação do mundo e dos mecanismos de deliberação, que são nomeadamente, mecanismos de raciocínio e de tomada de decisão.

Esta arquitectura o comportamento é gerado com base em processos de planeamento suportados pelo modelo do mundo.

Aplicando os conceitos do raciocínio automático ao conceito do agente autónomo, encontramos o raciocínio prático. Este tipo de raciocínio é orientado para a acção, e são utilizadas representações simbólicas dos objectivos a atingir, das acção que os agente pode realizar e do ambiente, sendo posteriormente gerados planos de acção que vão ditar o comportamento do agente.

Este tipo de raciocínio é caracterizado por 2 principais componentes, sendo estas: a deliberação, ou seja, o raciocínio sobre os fins a atingir, e que tem como resultado um conjunto de objectivos a concretizar, e o planeamento, ou seja, o raciocínio sobre as acções e recursos necessários para atingir os fins, que tem como resultado planos de acção para conseguir concretizar os objectivos determinados pela deliberação.

Estas duas componentes são suportadas pelo modelo do mundo, que é atualizado pelas percepções do ambiente, mas também por efeito de mecanismos de raciocínio prático, estes que também podem ser influenciados pelas percepções, em específico, através de eventos que influenciam os mecanismos de deliberação e planeamento.

Nestas arquiteturas, é o raciocínio prático que suporta o processo geral de tomada de decisão que determina o comportamento do agente, ou seja, quais as acções a realizar dadas as percepções obtidas e o estado do modelo do mundo.

O processo de tomada de decisão e acção tem os seguintes passos:

1. Observar o mundo (mais concretamente, gerar percepções)
2. Actualizar o modelo do mundo
3. Se for não for necessário reconsiderar, saltar para o passo 6.
4. Deliberar o que fazer (mais concretamente, gerar um conjunto de objectivos)
5. Planear como o fazer (ou seja, gerar um plano de acção)
6. Executar o plano de acção.

Neste tipo de arquiteturas, o modelo do mundo e os mecanismos de raciocínio prático podem ser organizados num controlo deliberativo, que executa todo o processo de tomada de decisão e acção, recebendo percepções, e gerando acções.

2.1.9 Processos de Decisão Sequencial

No problema geral de tomada de decisão de um agente deliberativo, os estados e acções do agente vão alternando ao longo do tempo, em função das decisões do agente, das acções selecionadas e dos resultados das mesmas. Sabendo isto, é necesário existir uma forma de prever e controlar as acções do agente ao longo do tempo, de modo a atingir um objectivo a longo prazo.

Devido a esta necessidade, surgem os processos de decisão sequencial, onde para cada estado, cada uma das decisões de acção podem levar a múltiplas ramificações de estados e decisões futuras. A utilidade desses estados futuros pode não ser conhecido de imediato, podendo ser diferido no tempo, em função do encadeamento de estados e decisões futuras. É também de frisar que a própria evolução dos estados observados em função das acções realizadas pode não ser determinista, o que quer dizer que o resultado de uma acção pode não ser completamente previsível, ou seja, uma acção aplicada num estado, pode ter mais do que 1 resultado possível (o que resulta em existirem várias transicções possíveis para a mesma acção).

A utilidade resulta de um conjunto de decisões, onde podem existir ganhos e perdas, sendo que o valor é cumulativo. Sendo que o ambiente pode não ser determinista, é necessário ter em conta a recompensa (valor que representa o ganho ou perda que terá uma transição) de cada transição possível para a acção.

2.1.10 Processos de Decisão de Markov (PDM)

Um tipo de processo de decisão sequencial é o processo de decisão de Markov.

Neste tipo de processos, a probabilidade de transição para um estado depende exclusivamente do estado atual.

A representação do mundo sob a forma de PDM é feita da seguinte forma:

S - conjunto de estados do mundo.

$A(s)$ - conjunto de acções possíveis no estado $s \in S$.

$T(s, a, s')$ - probabilidade de transição de s para s' através de a.

$R(s, a, s')$ - recompensa esperada na transição de s para s' através de a.

γ - taxa de desconto para recompensas diferidas no tempo.

t - tempo discreto (0,1,2,3,...).

O raciocínio automático com base em PDM envolve o cálculo da função de utilidade, que define o valor associado a cada estado do problema, a qual serve de suporte ao cálculo da política ótima de acção, que determina a seleção das ações que maximizam o valor da acção em cada estado.

A política comportamental refere-se à estratégia de acção do agente, ou seja, esta define qual a acção que deve ser realizada em cada estado. A política pode ser determinista ou não determinista.

Política determinista - Cada estado está associado a uma ação.

Política não determinista - Para cada par estado-acção, existe uma associação com um valor entre 0 e 1, ou seja, a probabilidade dessa acção ser escolhida. Os pares estado-acção devem ser o produto cartesiano entre o conjunto dos estados e o conjunto das acções.

Para encontrar a solução ótima, é preciso decompor o problema em sub-problemas, assumindo a independência entre caminhos.

A utilidade de cada estado é calculada a partir das utilidades dos estados sucessores de acordo com as equações de Bellman, que têm a 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

Figura 4: Equações de Bellman - [1]

2.2 Engenharia de Software em I.A.

2.2.1 O que é?

A engenharia de software é uma área de engenharia orientada para a especificação, desenvolvimento e manutenção de software, que tem por objectivo o desenvolvimento, operação e manutenção de software de modo sistemático e quantificável.

Sistemático - Significa a capacidade de realizar o desenvolvimento de software de forma organizada e previsível, de modo a garantir a satisfação dos requisitos definidos, incluindo tempo e recursos necessários.

Quantificável - Significa a capacidade de avaliar os meios envolvidos e os resultados produzidos no desenvolvimento de software, utilizando métodos e processos adequados para garantir a qualidade e o desempenho do software produzido, bem como a criação de documentação e a monitorização do processo de desenvolvimento.

O desenvolvimento de software de forma sistemática e quantificável é importante para garantir que os requisitos do software sejam cumpridos, bem como para prever e gerir os recursos necessários para o desenvolvimento e operação do software produzido. Além disso, a aplicação das boas práticas da engenharia de software leva a software mais robusto e produzido com maior rapidez.

Existem dois principais obstáculos que crescem ao longo do desenvolvimento: a complexidade e a mudança.

2.2.2 Complexidade

Expressa na crescente dificuldade de desenvolvimento, operação e manutenção de software, na crescente sofisticação do software produzido, bem como na quantidade crescente de recursos envolvidos, nomeadamente, em termos de capacidade de processamento e de memória utilizada.

O crescimento da complexidade de um sistema de forma não adequadamente controlada, pode tornar o esforço de desenvolvimento muito elevado, a ponto de comprometer a viabilidade do respetivo projecto. Isto deve-se ao facto da complexidade crescer exponencialmente, ou seja, um sistema com duas vezes mais partes é muito mais do que duas vezes mais complexo.

Para melhor entender o nível de complexidade, é possível dividir a complexidade em dois tipos:

Complexidade organizada - Resulta de padrões de inter-relacionamento entre as partes correlacionáveis no espaço e no tempo.

Complexidade desorganizada - Resulta do número e heterogeneidade das partes de um sistema. As partes podem interagir entre si, mas a interacção é irregular.

2.2.3 Mudança

A mudança é expressa no ritmo crescente a que o software necessita de ser produzido, ou modificado, para satisfazer as necessidades dos respetivos contextos de utilização, quer a nível de funcionalidades disponibilizadas, quer a nível das tecnologias utilizadas.

2.2.4 Arquitetura de Software

A arquitectura de software tem por objectivo abordar o problema da complexidade com base numa organização adequada do software a produzir. Para abordar este problema, necessitamos de 3 vertentes: métricas, princípios e padrões.

2.2.4.1 Métricas

Existem várias métricas para quantificar a qualidade da arquitetura:

Acoplamento - O conceito de acoplamento refere-se ao grau de dependência entre diferentes partes de um sistema, sendo tanto maior quanto maior a dependência entre as partes de um sistema. Pode ser medido utilizando diferentes métricas, como o número de associações entre elementos ou o grau de complexidade de uma interface. Quanto menor, melhor.

Coesão - O conceito de coesão refere-se à forma como os elementos de um sistema estão agrupados de forma coerente entre si, será tanto maior quando mais relacionados entre si forem os elementos agrupados em cada módulo. Pode ser medida utilizando diferentes critérios, por exemplo, por tipo de função das partes agrupadas, sendo designada neste caso por coesão funcional. Quanto maior, melhor.

Simplicidade - Nível de facilidade de compreensão/comunicação da arquitectura.

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

2.2.4.2 Princípios

Abstração - Ferramenta base para lidar com a complexidade, é o processo de descrição de conhecimento a diferentes níveis de detalhe e tipos de representação. Por exemplo, ao modular uma maçã e uma banana, é possivel abstrair conceitos comuns para um terceiro tipo, neste caso fruta, do qual ambos a maçã e a banana derivam/extendem.

Modularização - O conceito de modularização refere-se à capacidade de organização de um sistema em partes coesas, ou módulos, que podem ser interligados entre si para produzir a função do sistema, está relacionado com dois aspectos principais, decomposição e encapsulamento.

- **Decomposição** - Partir um sistema em partes coesas. Por exemplo, dividir as representações de percepções e de acções em diferentes módulos.
- **Encapsulamento** - Isolamento dos detalhes interiores de parte do sistema para as restantes partes, permitindo o acesso apenas através da sua interface (contracto funcional).

Factorização - A decomposição das partes de um sistema de modo a eliminar redundância. Por exemplo, se duas funções partilham algum código, é possivel realizar uma terceira função que execute apenas o código comum, para que esse código esteja presente numa única função.

2.2.5 Unified Modeling Language - UML

O desenvolvimento de sistemas com base em inteligência artificial é caracterizado pela elevada complexidade desses sistemas. Como tal, requer métodos adequados de engenharia de software, nomeadamente, no que se refere à modelação de sistemas, bem como a linguagens de modelação adequadas, como é o caso da linguagem UML.

3 Projecto – Parte 1

A primeira parte do projeto trata-se de um agente reactivo, e de um ambiente sobre o qual este actua. Esta primeira parte foi desenvolvida em Java. O agente reactivo encontra-se num jogo, com um ambiente virtual. O seu objectivo é fotografar os animais que conseguir encontrar.

Primeiro, foi desenvolvido o ambiente, comando e evento. O ambiente modulado é capaz de evoluir, observar um evento e executar um comando. Tanto o ambiente, como o evento e comando tratam-se de interfaces, que tal como referido anteriormente, diminuem o acoplamento da arquitetura. O uso de interfaces é um bom exemplo de encapsulamento, visto que os detalhes da implementação são ocultados pelo contrato funcional (interface) que é definido. Neste subsistema, é já possivel observar uma dependência, visto que Ambiente depende tanto de Evento como de Comando. A relação entre estas classes é apenas uma dependência, visto que o único uso é como parâmetro num método. Este tipo de relação é o tipo com menor acoplamento.

De seguida, foi então modulado o agente, o controlo, a percepção e a acção. Neste subsistema, já é possivel ver um exemplo de uma associação (por exemplo, de Percepção para Evento), que tem maior acoplamento que uma dependência. Esta associação acontece visto que a percepção contém uma referência para um evento. Também é possível observar uma composição (de Agente para Controlo), que tem ainda maior acoplamento que uma associação. A composição acontece quando uma parte é composta por outras parte que só existem no contexto da parte composto, neste caso, Agente é composto por Controlo, e um Controlo só existe no contexto de um Agente. É também de notar que neste sistema existe também pela primeira vez a necessidade de existirem atributos *read only* (apenas de leitura). Em Java, isto é possivel através do encapsulamento do atributo (ou seja, o atributo é privado e existem *getters* e *setters*, neste caso apenas existe o *getter*, visto que é *read only*).

De seguida foi modulado o subsistema do ambiente do jogo, modulando as classes AmbienteJogo, EventoJogo e ComandoJogo. Neste subsistema, temos um exemplo de uma realização de uma interface (por exemplo, AmbienteJogo realiza Ambiente). Esta é a relação com maior acoplamento (juntamente com a generalização). Neste caso específico, o ambiente do jogo evolui através da geração de novos eventos. Esta geração acontece através do *input* do utilizador, podendo este escolher entre os eventos possiveis.

De seguida, foi modulado o subsistema da personagem, desenvolvendo a classe Personagem. Neste subsistema, aparece finalmente uma generalização (de Personagem para Agente), que tal como foi referido anteriormente, é uma das relações com maior acoplamento.

De seguida, foi modulado o subsistema Jogo, que apenas incorpora a classe Jogo. Esta classe tem o método main.

De seguida, foram moduladas as classes MaquinaEstados, Estado e Transição.

Para finalizar a primeira parte do projeto, foi então modulada a classe ControloPersonagem.

Após finalizadas todas as classes, foi possivel testar e concluir que o programa estava a funcionar corretamente, sem a necessidade de corrigir algum tipo de código. Isto foi apenas possivel graças à aplicação de boas práticas da engenharia de software durante o desenvolvimento do código, passando por exemplo, pelo desenvolvimento do código estrutural antes do desenvolvimento do código comportamental.

Esta primeira parte do projeto está diretamente relacionada com os seguintes tópicos teóricos:

- 2.1.4 Agente Inteligente
- 2.1.5 Representação do Ambiente
- 2.1.8.1 Arquitetura de Agentes Reactivos
- 2.2 Engenharia de Software em I.A.

Estando concluída a primeira parte do projeto, foi então iniciada a segunda parte.

4 Projecto – Parte 2

A segunda parte do projeto trata-se de uma biblioteca para o desenvolvimento de agentes reactivos (ecr), e da modulação de um agente para percorrer um ambiente, onde este deve recolher os alvos. Esta parte (e as seguintes) foi desenvolvida em Python.

Para esta parte do projeto (e para as seguintes partes também), foi utilizado o sae (Simulador de Ambiente de Execução), biblioteca que representa um ambiente. Esta foi fornecida pelo docente.

Dando inicio à segunda parte do projeto, começamos por desenvolver a biblioteca ecr.

Após o desenvolvimento da biblioteca, foi então desenvolvido o controlo reactivo, o comportamento Explorar, a resposta RespostaMover e a hierarquia Recolher. Com estas classes desenvolvidas, foi possivel testar uma versão simples do agente, usando apenas o comportamento Explorar dentro da hierarquia Recolher. Foi verificado que o agente movia-se aleatoriamente, e que por vezes recolhia os alvos, como esperado.

De seguida, foram desenvolvidas as classes AproximarDir, EstimuloAlvo, AproximarAlvo, EvitarDir, EstimuloObst, EvitarObst e RespostaEvitar. Com estas classes desenvolvidas, é agora possivel incluir não só o comportamento Explorar na hierarquia Recolher, como também os comportamentos AproximarAlvo e EvitarObst. Foi então testado a nova versão do agente, com os novos comportamentos, e verificado que este já não tentava avançar contra paredes, e que ia ao encontro dos alvos quando os percepcionava.

No final desta parte do projeto, e como foram também aplicadas as boas práticas da engenharia de software durante o desenvolvimento do código, foi apenas necessário corrigir no ControloReact, para que este mostrasse a informação sensorial. Sendo verificado que o comportamento agora era o expectável, foi então dada como concluida a segunda parte do projeto.

Esta segunda parte do projeto está diretamente relacionada com os seguintes tópicos teóricos:

- 2.1.4 Agente Inteligente
- 2.1.5 Representação do Ambiente
- 2.1.8.1 Arquitetura de Agentes Reactivos
- 2.2 Engenharia de Software em I.A.

5 Projecto – Parte 3

Na terceira parte do projeto, foi desenvolvido todo o suporte para o raciocínio automático com base em procura em espaços de estados, sendo possível utilizar vários tipos de procura, tais como: procura em profundidade, procura em profundidade limitada, procura em profundidade iterativa, procura em largura, procura de custo uniforme, procura sôfrega e procura A*.

Sendo desenvolvido este módulo, foi necessário testar a sua implementação. Para tal, foi modulado um problema de contagem, onde a contagem é iniciada a 0, e é pretendido chegar ao valor 9, sendo possível incrementar 1 ou 2 valores em cada passo, com o custo de cada passo igual ao quadrado do valor a incrementar (por exemplo, ao incrementar 2, o custo seria $2^2 = 4$). A heurística escolhida foi a diferença absoluta entre o valor do estado e o valor objectivo (neste caso, 9).

Sendo modelado o problema, foi possível testar todas os tipos de procura implementados, sendo verificado que todos estavam funcionais.

Esta terceira parte do projeto está diretamente relacionada com os seguintes tópicos teóricos:

- 2.1.6 Raciocínio Automático
- 2.1.7 Procura em Espaços de Estados
- 2.1.8.2 Arquitetura de Agentes Deliberativos

6 Projecto – Parte 4

A quarta e última parte do projeto começou com a implementação dos módulos necessários para utilizar o raciocínio automático baseado em procura em espaços de estados num agente deliberativo.

Para tal, foi necessário construir os módulos necessários para o controlo deliberativo. Depois, foi necessário modelar o nosso problema de recolha de alvos. A heurística escolhida neste caso foi a distância entre o estado e o estado objectivo.

Com estas duas tarefas concluídas, foi possível observar o agente a realizar o raciocínio automático com base em PEE, e comparar visualmente os diferentes tipos de procura.

De seguida, foi iniciada a última implementação do projeto, sendo esta a implementação de um processo de decisão sequencial, mais especificamente o processo de decisão de Markov.

Nesta implementação, é de notar a classe ModeloPDMPlan, onde foram implementadas ambas as interfaces ModeloPlan e ModeloPDM, e dessa forma, este modelo pode ser utilizado nos módulos existentes do agente deliberativo, mas também nos novos módulos de suporte ao processo de decisão de Markov. Além disso, neste módulo é possivel verificar o uso de uma técnica ainda não utilizada no projeto: a delegação de tarefas. Esta classe guarda num atributo privado um ModeloPlan, delegando depois as suas funções associadas a essa interface para esse atributo.

É também de notar que o ambiente utilizado é um ambiente deterministico, logo a probabilidade de transição é sempre 0% ou 100%. (100% se for possível transicionar para aquele estado com aquela acção, 0% se não)

A recompensa de transição é igual ao oposto do custo da ação, ou se o estado sucessor for um estado objectivo, o valor máximo de recompensa, que é recebido no construtor da classe.

Estando estes módulos desenvolvidos, foi testado o agente deliberativo com base num planeador com base em PDM. Foi verificado que para os ambientes 1, 2, 3 e 7 o agente funcionava perfeitamente. No entanto, para os ambientes 4, 5 e 6 foi necessário aumentar o valor do gama (que por omissão é 0.85), visto que os objectivos encontravam-se mais longe do agente. Ao aumentar o valor do gama para 0.98, foi possível reparar que apesar do agente demorar mais tempo a calcular a política ótima, esta conseguia agora chegar aos objectivos.

Tendo sido testado com sucesso esta última funcionalidade, foi dado como concluído o projeto da Unidade Curricular.

Esta quarta parte do projeto está diretamente relacionada com os seguintes tópicos teóricos:

- 2.1.6 Raciocínio Automático
- 2.1.7 Procura em Espaços de Estados
- 2.1.8.2 Arquitetura de Agentes Deliberativos
- 2.1.9 Processos de Decisão Sequencial
- 2.1.10 Processos de Decisão de Markov (PDM)

7 Revisão do projecto realizado

Durante as diversas entregas ocorreram poucos erros, sendo o mais notável a falta de documentação e justificação teórica para o código desenvolvido. Esta deve-se à diferença entre a documentação/justificação teórica (e a forma como esta é pedida, sobre a forma de comentários no código) nesta Unidade Curricular, com a documentação habitual entregue nas restantes unidades curriculares de LEIC (e mesmo quando é necessário algum tipo de justificação teórica, esta é entregue separadamente). No entanto, durante a 3ª e 4ª parte do projeto, foi tido mais cuidado para tentar incluir toda a documentação necessária.

Na entrega do projeto feita junto com o relatório, as únicas mudanças foram ao nível dos comentários, visto que o código já estava realizado e testado na última entrega.

8 Conclusão

Durante todo o projeto foi possível abordar diversos temas, como agentes reactivos, mecanismos de procura (não informada e informada), agentes deliberativos com base em PEE e em PDM, e diversas vertentes da engenharia de software, como por exemplo, o desenvolvimento de código com suporte em UML. Tendo sido concluído o projeto, é notável o aproveitamento da Unidade Curricular, tendo sido possível adquirir informação sobre os diversos temas. Além disso, existe agora uma biblioteca bastante completa que pode ser usada para diversos problemas

Referências

[1] Luis Morgado. *Processos de decisão sequencial*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1241346/mod_resource/content/2/15-pds.pdf.

[2] Luis Morgado. *Introdução à inteligência artificial*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1232247/mod_resource/content/1/02-introd-ia.pdf.

[3] Luis Morgado. *Introdução à engenharia de software - Parte 1*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1232248/mod_resource/content/1/03-introd-eng-soft-1.pdf.

[4] Luis Morgado. *Introdução à engenharia de software - Parte 2*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1232250/mod_resource/content/1/04-introd-eng-soft-2.pdf.

[5] Luis Morgado. *Introdução à engenharia de software - Parte 3*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1232251/mod_resource/content/1/05-introd-eng-soft-3.pdf.

[6] Luis Morgado. *Arquitectura de agentes reactivos - Parte 1*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1237114/mod_resource/content/1/06-arq-react-1.pdf.

[7] Luis Morgado. *Arquitectura de agentes reactivos - Parte 2*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1237119/mod_resource/content/1/07-arq-react-2.pdf.

[8] Luis Morgado. *Arquitectura de agentes reactivos - Parte 3*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1238392/mod_resource/content/1/08-arq-react-3.pdf.

[9] Luis Morgado. *Raciocínio automático*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1239030/mod_resource/content/2/09-rac-aut.pdf.

[10] Luis Morgado. *Procura em espaços de estados - Parte 1*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1239031/mod_resource/content/3/10-pee-1.pdf.

[11] Luis Morgado. *Procura em espaços de estados - Parte 2*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1239033/mod_resource/content/1/11-pee-2.pdf.

[12] Luis Morgado. *Procura em espaços de estados - Parte 3*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1239034/mod_resource/content/1/12-pee-3.pdf.

[13] Luis Morgado. *Arquitectura de agentes deliberativos*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1240633/mod_resource/content/1/13-arq-delib.pdf.

[14] Luis Morgado. *Planeamento automático com base em procura em espaços de estados*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1240634/mod_resource/content/1/14-plan-pee.pdf.

[15] Luis Morgado. *Planeamento automático com base em PDM*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1241347/mod_resource/content/1/16-plan-pdm.pdf.

[16] Luis Morgado. *Aprendizagem por reforço*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1241988/mod_resource/content/1/17-aprend-ref.pdf.

[17] Luis Morgado. *Projecto - Parte 1*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1232249/mod_resource/content/2/P1-iasa-proj.pdf.

[18] Luis Morgado. *Projecto - Parte 2*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1237118/mod_resource/content/1/P2-iasa-proj.pdf.

[19] Luis Morgado. *Projecto - Parte 3*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1239199/mod_resource/content/1/P3-iasa-proj.pdff.

[20] Luis Morgado. *Projecto - Parte 4*. 2024. URL: https://2324moodle.isel.pt/pluginfile.php/1240912/mod_resource/content/5/P4-iasa-proj.pdf.