Instituto Superior de Engenharia de Lisboa

Departamento de Engenharia Eletrónica e de Computadores

Relatório Intermédio

Inteligência Artificial para Sistemas Autónomos

Docente:

Luis Morgado

Aluno:

Ricardo Pinto N.º 47673

abril de 2024

Conteúdo

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., especialmente focados no desenvolvimento de agentes reativos, e as práticas essenciais da engenharia de software que sustentam a construção de softwares robustos e eficientes.

Foram realizadas duas 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.

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.

2.2 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.

2.2.5 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.

2.2.6 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.

2.2.7 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.

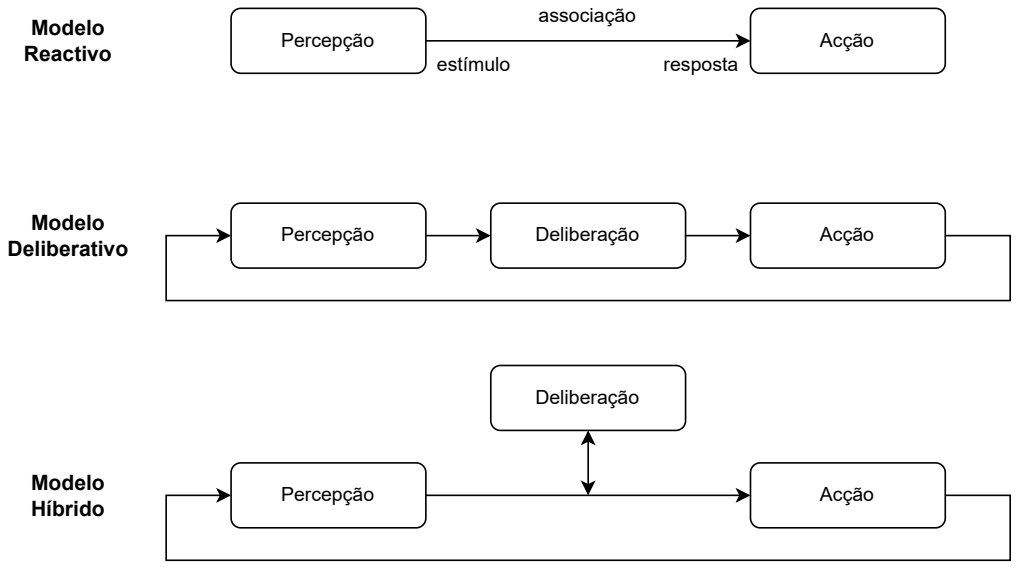
2.3 Arquiteturas de Agente

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.



2.3.1 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.

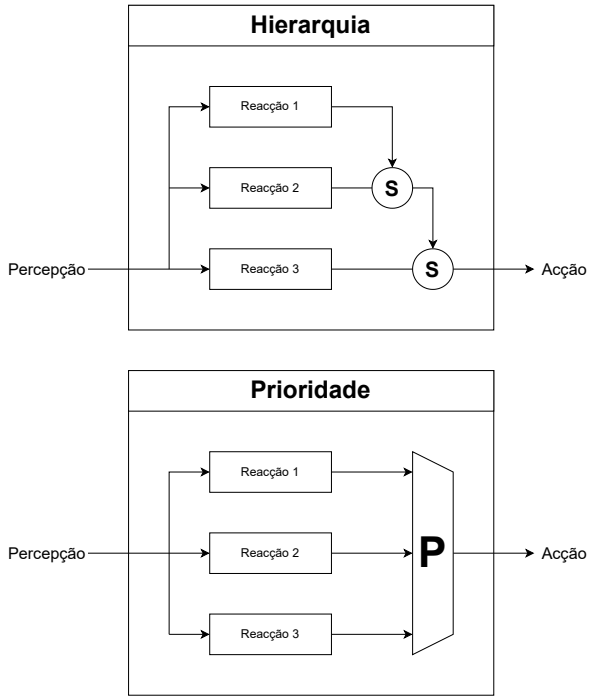
Resposta - Define uma resposta a estímulos, em termos de acção a realizar e da respectiva prioridade.

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.



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.

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.

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.

5 Projecto – Parte 3

Ainda não incluída.

6 Projecto – Parte 4

Ainda não incluída.

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). De resto, as entregas foram entregues sem erros de código.

8 Conclusão

O projeto até ao momento abordou vários tópicos fundamentais da Inteligência Artificial para Sistemas Autónomos e da Engenharia de Software, tais como os agentes reactivos e o desenvolvimento de codigo com suporte em UML. Com a conclusão da parte 1 e 2 da unidade curricular, é possivel concluir que está desenvolvido, na teoria e prática, uma forte base para a aprendizagem dos seguintes tópicos da unidade curricular, tais como a procura em espaço de estados.

Referências

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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