Relatório de Projeto

Inteligência Artificial para Sistemas Autónomos

Licenciatura em Engenharia Informática e de Computadores

Autores:	**Francisco Engenheiro**
ISEL ID:	49428
Professor:	**Phd Luís Morgado**
Ano académico:	2023/2024

Índice

Capítulo 1

Introdução

Este relatório documenta o projeto desenvolvido no âmbito da unidade curricular de Inteligência Artificial para Sistemas Autónomos, da Licenciatura em Engenharia Informática e de Computadores.

O projeto tem como objetivo a aprendizagem de conceitos de inteligência artificial e a sua aplicação no desenvolvimento de sistemas autónomos inteligentes. O capítulo 2 descreve esses conceitos que servem de base aos restantes temas estudados e ao projeto realizado.

O projeto está dividido em duas partes:

- **Parte 1**: Desenvolvimento de uma biblioteca em Java para a implementação de agentes autónomos inteligentes; Desenvolvimento de um jogo em Java que integra essa biblioteca (ver capítulo 3).
- **Parte 2**: Desenvolvimento de um agente reativo em Python com módulos comportamentais para recolher alvos e evitar obstáculos num ambiente com dimensões fixas (ver capítulo 4).

Cada parte do projeto está organizada de forma a descrever o objetivo principal no contexto geral do projeto; fazer uma síntese dos conceitos estudados que se acharam relevantes para a sua implementação; caracterização do ambiente; descrever a implementação realizada, nomeadamente, a arquitetura do agente implementada; justificar as principais decisões tomadas e a sua relação com os conceitos estudados; e apresentar a estrutura do código desenvolvido.

Existe um capítulo dedicado à revisão do projeto realizado, onde são identificados e descritos os erros cometidos nas entregas realizadas, com a indicação do problema e da respetiva correção, justificada com base nos conhecimentos adquiridos.

O projeto foi realizado individualmente e as entregas foram realizadas semanalmente, de forma incremental, de acordo com o plano de trabalho definido.

Capítulo 2

Enquadramento Téorico

A inteligência artificial, um ramo da engenharia informática, concentra-se no desenvolvimento de algoritmos e sistemas que imitam a capacidade humana de raciocínio. Entre as suas diversas aplicações, destacam-se a visão computacional, o processamento de linguagem natural, a robótica e a aprendizagem automática (machine learning). No contexto da aprendizagem automática, mais concretamente para o desenvolvimento de sistemas autónomos, ao qual este projeto se insere, destacam-se os conceitos de inteligência e autonomia. A inteligência caracteriza-se pela relação entre a cognição (i.e. capacidade de realizar a ação adequada dadas as condições do ambiente) e a racionalidade (i.e., capacidade de decidir no sentido de conseguir o melhor resultado possível perante os objectivos que se pretende atingir e as condições do ambiente). Já a autonomia é a habilidade de um sistema operar de forma independente, sem intervenção externa, seja ela humana ou de outro sistema. Representa uma característica da inteligência e portanto ser autónomo não significa ser inteligente, no entanto, ser inteligente implica ser autónomo.

2.1 Agente Inteligente

O modelo de um sistema inteligente é uma abstracção que permite representar a interacção de um sistema com o seu ambiente. Este implementa um ciclo realimentado percepção-processamento-acção (ver figura 2.1), através do qual é realizado o controlo da função do sistema de modo a concretizar a finalidade desse sistema (e.g., a travagem automática de um automóvel quando deteta um obstáculo à sua frente).

Além da autonomia, as caracteristicas principais de um agente inteligente, que estão dependentes do tipo de arquitetura (ver secção 2.4), são:

- **Reactividade**: Capacidade de reagir aos diversos estímulos do ambiente;
- **Pro-actividade**: Capacidade de tomar a iniciativa em função dos seus objetivos;

- **Sociabilidade**: Capacidade de interagir com outros agentes em prol de atingir os seus objetivos, individuais ou coletivos;
- **Finalidade**: Propósito que o agente deve atingir e ao qual todas as suas características contribuem para a sua concretização.

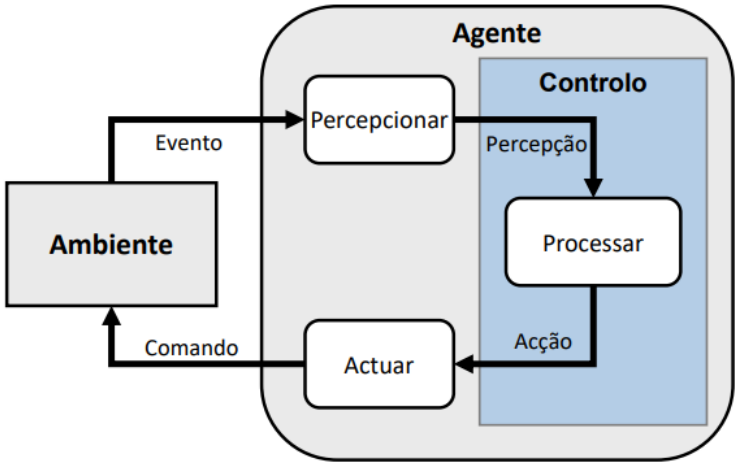

Figura 2.1: Representação conceptual da relação entre agente e ambiente.

2.2 Ambiente

O espaço onde um agente opera é designado por ambiente, e que pode ser real (e.g., uma sala de aula) ou virtual (e.g., um jogo de computador). É caracterizado pelas seguintes propriedades:

- **Totalmente Observável vs. Parcialmente Observável**: Um ambiente é totalmente observável se o agente tem acesso a uma descrição completa do ambiente, que lhe permite resolver o problema em questão sem necessitar de guardar estado interno; caso contrário, é parcialmente observável;
- **Tipo de Agente**: Os ambientes podem ser de agente único (i.e., apenas existe um agente a atuar) ou multi-agente (i.e., existem vários agentes a atuar, iguais ou diferentes entre si);
- **Determinístico vs. Estocástico**: Um ambiente é determinístico se o estado seguinte é unicamente determinado pelo estado actual e pela acção do agente; caso contrário, é estocástico. Mais ainda, dizemos que um ambiente continua a ser determinístico, mas mais concretamente estratégico, caso estejamos num ambiente multi-agente onde o próximo estado pode estar também dependente das ações de outros agentes (e.g., num jogo de xadrez);
- **Episódico vs. Sequencial**: Um ambiente é episódico se a experiência do agente

é dividida em episódios independentes; caso contrário, é sequencial (i.e., pode ser representado por uma sequência de estados);

- **Estático vs. Dinâmico**: Um ambiente é estático se não se altera enquanto o agente está a tomar uma decisão; caso contrário, é dinâmico;
- **Discreto vs. Contínuo**: Um ambiente é discreto se o número de estados possíveis é finito; caso contrário, é contínuo.

Na figura 2.2 são apresentados exemplos de caracterização de ambientes.

Ambiente	Observável?	Agentes?	Determinístico?	Episódico?	Estático?	Contínuo?
Palavras Cruzadas	Completamente Observável	Agente Único	Determinístico	Sequencial	Estático	Discreto
Poker	Parcialmente Observável	Multi-Agente	Estocástico	Sequencial	Estático	Discreto
Mundo Real	Parcialmente Observável	Multi-Agente	Estocástico	Sequencial	Dinâmico	Contínuo

Figura 2.2: Exemplos de caracterização de ambientes. Retirado de [1].

2.3 Agente Relacional

Um agente racional é um tipo de agente que realiza as ações corretas, ou seja, deve conseguir, a partir da exploração e aprendizagem, descobrir estratégias para chegar ao seu objetivo de forma ótima. Para esse efeito, é escolhida a ação que maximiza o valor esperado da medida de desempenho segundo a informação que lhe é fornecida (i.e., perceções) e o conhecimento adquirido até ao momento (e.g., conhecimento disponível sobre o ambiente).

O conceito de recolha de informação entra neste contexto, visto que a exploração pode ser feita fora do âmbito do objetivo com vista a adquirir conhecimento (e.g., estado do ambiente, ações possíveis, recompensas associadas a cada ação, etc) e, assim, melhorar a tomada de decisão seguinte.

No entanto, a racionalidade não implica clarividência, isto é, a ação tomada pode não resultar no que pretendemos ou esperamos, visto que o raciocínio nem sempre leva ao sucesso (e.g., o planeamento de uma viagem de avião, que foi reservada com antecedência e com atenção às condições climatéricas, não garante que o voo não seja cancelado no dia da viagem ou a viajem não seja interrompida por qualquer motivo alheio, mesmo tendo sido tomadas todas as precauções possíveis com base na informação disponível).

2.4 Arquiteturas de Agente

A arquitetura de um agente é a estrutura que define quais os componentes do agente e a forma como estes interagem entre si. Um dos componentes de um agente é o módulo de

controlo, que é responsável por processar perceções e gerar ações. A forma como este módulo é implementado determina o modelo de arquitetura do agente, que pode ser:

- **Reativo**: Associado a um paradigma comportamental, é caracterizado por associações diretas entre perceções e ações. A finalidade deste modelo é a concretização de objetivos implícitos, presentes nas associações estímulo-resposta (i.e., as ações são diretamente ativadas em função das perceções);
- **Deliberativo**: Associado a um paradigma simbólico, é caracterizado pela existência de um módulo de deliberação que se situa entre a perceção e a ação. É neste módulo que ocorre o raciocínio e a tomada de decisão, com base em objetivos explícitos;
- **Híbrido**: Combinação dos modelos reativo e deliberativo, com o objetivo de tirar partido das vantagens que cada um oferece.

Capítulo 3

Projeto - Parte 1

Esta parte do projeto incidiu principalmente sobre desenvolvimento de uma biblioteca, em Java, para providenciar abstrações dos subsistemas que representam conceito gerais de Inteligência Artificial (e.g., agente, ambiente) e outros conceitos relacionados (e.g., máquina de estados).

Para tal, foi necessário definir uma arquitetura de software que permitisse a implementação dos diferentes subsistemas de forma independente e modular, seguindo as diretrizes (i.e., métricas (ver secção 3.1.1) e princípios (ver secção 3.1.2)) que garantem a qualidade da arquitetura.

A implementação da biblioteca foi feita com base na consulta e compressão de diagramas UML e de sequência de forma a garantir a correta implementação dos diferentes subsistemas.

Por fim foi desenvolvido um jogo (ver secção 3.4) que integra a biblioteca desenvolvida, permitindo a interação com o jogador por meio de comandos em texto.

3.1 Arquitetura de software

A arquitetura de software aborda a complexidade inerente ao desenvolvimento de software por meio de uma série de vertentes que estão interligadas.

3.1.1 Métricas

As métricas são medidas de quantificação da arquitectura de um software indicadoras da qualidade dessa arquitectura;

O acomplamento é uma métrica inter-modular que mede o grau de interdependência entre os módulos de um sistema. Pode ser medido através da:

- **Direção**: Unidirecional vs Bidirecional (uni representa menos acoplamento);
- **Visibilidade**: Quando menor for a visibilidade de um módulo, menor é o seu acoplamento;

- **Ordem**: (de menos acoplamento para mais) Herança → Composição → Agregação → Associação → Dependência.

A coesão é uma métrica intra-modular que determina o nível de coerência funcional de um subsistema/módulo, seja pela sua organização (i.e., cada modulo está organizado por conteúdo) ou pela sua funcionalidade (e.g., single responsibility principle - cada modulo tem uma única responsabilidade).

3.1.2 Princípios

Os princípios no contexto da arquitectura de software são um conjunto de convenções que orientam a sua definição, garantindo a qualidade de produção da mesma. Alguns exemplos são:

- **Abstração**: Define a forma como os componentes de um sistema são representados, permitindo a ocultação de detalhes de implementação;
- **Modularização**: Ao qual está associado a decomposição (e.g, divisão do sistema em sub-módulos) e o encapsulamento (i.e., ocultação de detalhes de implementação e/ou manutenção de estado privado e interno);
- **Factorização**: Onde a arquitectura é dividida em camadas, cada uma com um conjunto de responsabilidades bem definidas. Pode ser estrutural (e.g, Herança) e Funcional (e.g, Delegação);

3.2 Processo de Desenvolvimento de Software

O processo de desenvolvimento de software consiste na criação da organização de um sistema de forma progressiva, através de diferentes níveis de abstracção:

- **Modelo (Conceptual)**: Representação abstrata do sistema, que define o que o sistema deve fazer, sem especificar como;
- **Arquitetura (Modelo Concreto)**: Representação concreta do sistema, que define como o sistema deve ser implementado;
- **Implementação**: Código fonte que implementa o sistema definindo como o sistema deve ser executado.

Consiste num processo iterativo, em que as diferentes actividades de desenvolvimento são alternadas ao longo do tempo em função do conhecimento e do nível de detalhe envolvido. Essa alternância poderá ser circular (i.e., implementação → arquitectura → modelo → implementação).

3.2.1 Tipos de Implementação

Tabela 3.1: Tipos de Implementação

Tipo	**Modelo Associado**	**Designação**
Estrutural	UML	Define a estrutura de um sistema, ou seja, a forma como os componentes se relacionam entre si.
Comportamental	Diagrama de Sequência	Define o comportamento de um sistema, ou seja, a forma como os componentes interagem e comunicam entre si.

Mais detalhadamente, os diagramas de sequência ou atividade representam o fluxo de controlo de um sistema, ou seja, a sequência de atividades que um sistema executa e a sua ordem. Definem-se como modelos de interação com uma organização bidirecional (i.e., horizontal → tempo e vertical → estrutura) e são compostos por diferentes elementos de modelação (e.g., mensagens, operadores, linha de vida).

Já a linguagem de modelação unificada (i.e., UML - Unified Modeling Language) representa um modelo de comportamento com interação como perspetiva principal de modelação. Este tipo de modelação descreve a forma como as partes de um sistema interagem entre si e com o exterior para produzir o comportamento do sistema. No contexto do projeto, esta linguagem também ajudou a compreender os conceitos fundamentais associados a um sistema computacional (ver secção 3.3) através da representação de diagramas de transição de estado.

3.3 Modelação de um Sistema Computacional

Um sistema computacional geral, pode ser caracterizado de forma abstrata (ver figura 3.1), através de uma representação de estado, que define em cada momento a configuração interna do sistema, e de uma função de transformação que gera as saídas e o próximo estado em função das entradas e do estado actual do sistema [2].

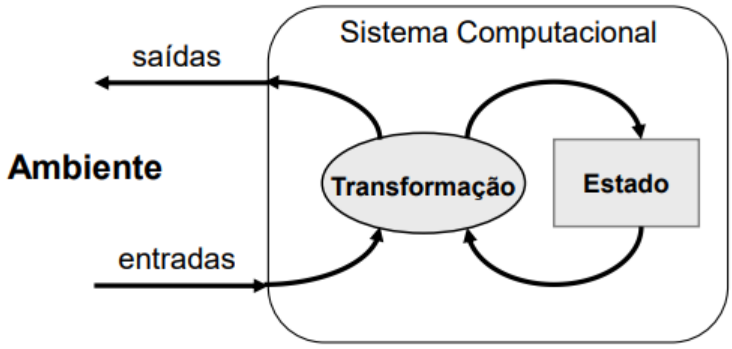

Figura 3.1: Modelo abstrato de um sistema computacional. Retirado de [2], slide 3.

Os conceitos fundamentais de um sistema computacional são:

- **Dinâmica**: Descreve os estados que um sistema pode assumir e a forma como eles evoluem ao longo do tempo, determinando o comportamento do sistema;
- **Comportamento**: Expressa a estrutura de controlo do sistema, que corresponde à forma como o sistema age (gera as saídas) perante a informação proveniente do exterior (entradas) e do seu estado interno;

3.3.1 Modelo formal de computação

A função de transformação, mencionada anteriormente, pode ser decomposta em duas funções distintas (ver tabela 3.2), em que **Q** representa o conjunto de estados possíveis do sistema (regularmente designado por espaço de estados), Σ representa o conjunto de entradas possíveis e **Z** representa o conjunto de saídas possíveis.

Tabela 3.2: Funções de Transformação de um Sistema Computacional

Designação	**Expressão**	**Descrição**
Função de Transição (δ)	$\delta : Q \times \Sigma \rightarrow Q$	Define a relação entre o estado do sistema e as entradas que recebe, e o próximo estado.
Função de Saída (λ)	$\lambda : Q \times \Sigma \rightarrow Z$	Define a relação entre o estado do sistema e as entradas que recebe, e a saídas que produz.

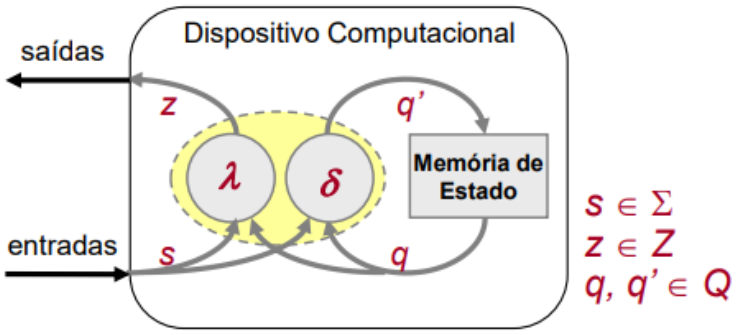

Figura 3.2: Modelo de um sistema computacional. Retirado de [2], slide 3.

Associado a um determinado modelo de um sistema computacional, está uma máquina de estados, que representa, de forma abstrata, o comportamento do sistema em função do tempo. Pode ser caracterizada por:

- **Estado**: Se o número de estados possíveis é finito, então a máquina de estados é finita; caso contrário, é infinita;
- **Determinismo**: Uma máquina de estados finita também pode ser subcaracterizada em determinística (existe apenas uma transição para o mesmo estado e entrada) ou

não determinística (existe mais do que uma transição para o mesmo estado e entrada). Para qualquer máquina de estados não determinística, existe uma máquina de estados determinística equivalente; [3, 4]

- **Função de Saída**: Se a função de saída λ depende das entradas Σ, então a máquina de estados é do tipo *Mealy* ($\lambda : Q \times \Sigma \to Z$); caso contrário, é do tipo *Moore* ($\lambda : Q \to Z$).

3.4 Desenvolvimento do Jogo

Tendo por base os conceitos anteriormente abordados, que consolidaram a aprendizagem inerente à implementação da biblioteca, foi desenvolvido um jogo. No seu processo de desenvolvimento, foi definido um conjunto de componentes que providenciam implementações concretas dos subsistemas expostos pela biblioteca, adaptados ao contexto específico do jogo:

- **Personagem**: Representa o agente do jogo;
- **Ambiente**: Representa o ambiente onde a Personagem opera;
- **Máquina de Estados**: Representa a máquina de estados associada ao controlo da Personagem. Caracterizada por ser finita, não determinística e do tipo *Mealy* (Ver figura 3.3).

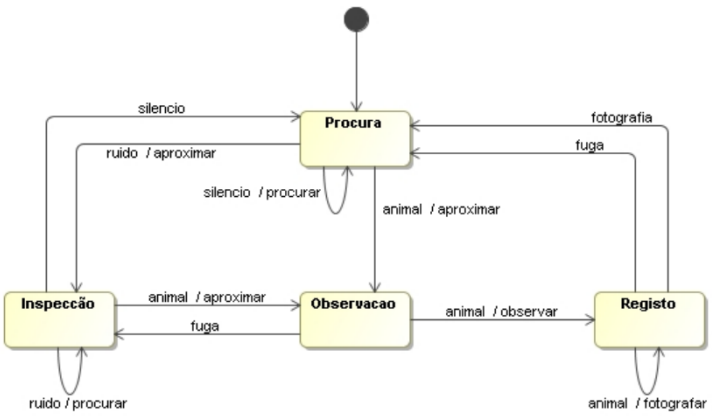

Figura 3.3: Máquina de estados associada ao controlo da Personagem. Retirado de [2], slide 14.

O jogo consiste num ambiente virtual onde a personagem tem por objectivo registar a presença de animais através de fotografias, simulando a exploração no mundo real. Pode ser caracterizado (ver secção 2.2) por ser:

- **Parcialmente observável**: A Personagem apenas sabe o que está no seu campo de visão, e por isso precisa de explorar o ambiente para descobrir o que está à sua volta, necessitando de guardar estado interno para tomar decisões (e.g., se observou um animal anteriormente, então pode registar a sua presença);

- **Agente único**: Considerou-se que a Personagem é a única a atuar no ambiente no contexto do problema, visto que os animais são tratados como objetos passivos e integrantes do ambiente;
- **Estocástico**: Porque o estado seguinte pode depender de eventos aleatórios (e.g., a fuga de um animal) e não somente do estado atual e da ação da Personagem;
- **Sequencial**: Pois existe uma linha de acontecimentos que se sucedem (e.g., não é possível inspecionar, registar ou observar algo sem ser feita primeiro uma procura);
- **Dinâmico**: Pois o ambiente pode mudar enquanto a Personagem está a tomar uma decisão (e.g., os animais podem fugir);
- **Discreto**: Pois o número de estados possíveis é finito (e.g., pode ser mapeado para uma máquina de estados finita).

Foi criada uma aplicação (ver figura 3.4) de cli (i.e., command-line interface) em Java, que possibilita a interação com o jogador por meio de comandos em texto. Tendo em conta que a interface gráfica não era o foco desta parte do projeto, foi emulada através de texto descritivo.

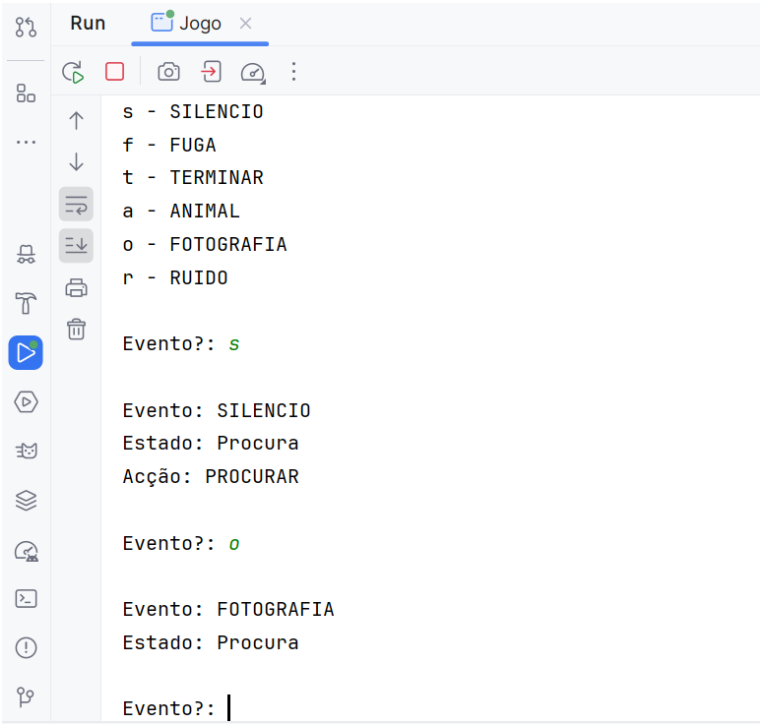

Figura 3.4: Utilização da aplicação do jogo.

3.5 Estrutura do Projeto

No processo de desenvolvimento de software associado a esta fase do projeto, foi definida a seguinte estrutura em módulos e que está presente na pasta iasa_jogo/src:

- *agente*: Integra classes e interfaces que definem o Agente e os seus componentes (e.g., módulo de controlo);
- *ambiente*: Agrega interfaces que representam o Ambiente e os seus componentes (e.g., comandos, eventos);
- *maqest*: Contém classes que definem o conceito de Máquina de Estados e os seus componentes (e.g., estados, transições);
- *jogo*: Agrega os detalhes da implementação do jogo, onde são integrados os módulos anteriores e definidas implementações concretas dos mesmos, adequadas ao contexto a que o jogo se insere.

Capítulo 4

Projecto - Parte 2

Nesta fase do projeto, o objetivo foi criar, em python, um sistema autónomo inteligente capaz de se movimentar num espaço com dimensões fixas onde existem obstáculos. A finalidade do agente prospetor é recolher os alvos e evitar os obstáculos presentes no ambiente (ver figura 4.1). O agente pode mover-se nos quatro sentidos cardeais (i.e., norte, sul, este e oeste).

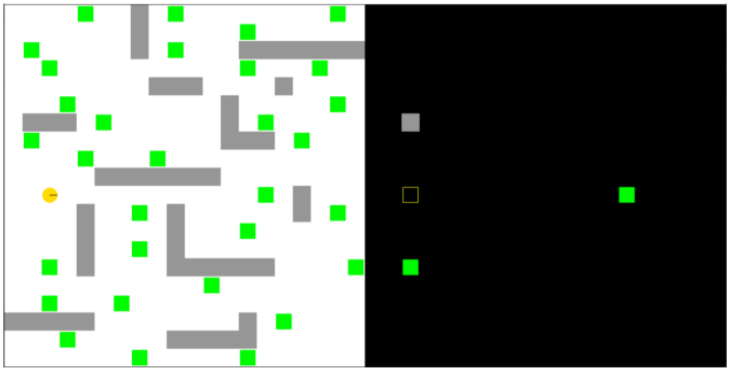

Figura 4.1: Representação visual desta fase do projeto.

Para que o agente possa atingir os objetivos propostos, foi necessário implementar (ver secção 4.6) um agente reativo e os módulos comportamentais associados, que lhe permitam reagir a estímulos e gerar respostas de forma a alcançar tais objetivos.

Tal como em fases anteriores, a implementação foi feita com base na consulta e compressão de diagramas UML e de sequência de forma a garantir a correta implementação dos diferentes subsistemas.

4.1 Agentes Reativos

Na secção da arquitetura de agentes (ver secção 2.4), foi referido que uma arquitetura reativa é caracterizada pela associação direta entre perceções e ações.

4.1.1 Reação

Inerente a arquitetura de agentes reativos, está o conceito de mecanismo de reação e que apresenta os seguintes elementos:

- **Estímulo**: Define a informação que é extraída de uma percepção, de forma a ser utilizada na ativação de uma resposta. Vários estímulos podem ser ativados por uma única perceção. Estão regularmente associados a um parâmetro de intensidade.
- **Resposta**: Representa a geração de uma resposta a estímulos, inerentemente ligada a uma ação a ser executada e à sua respetiva prioridade. Além de poder ser ativada por um estímulo, uma resposta pode ser ativada diretamente por uma percepção, de forma a garantir restrições de ativação (i.e., guardas) se necessário.
- **Reação**: Associa estímulos a respostas.

Um agente reativo é composto por um conjunto de reações, que devem ser organizadas de forma modular em módulos comportamentais designados por comportamentos. Isto permite as reações internas do agente sejam encapsuladas, facilitando a sua manutenção e escalabilidade.

4.1.2 Comportamento

Um comportamento (ver figura 4.2) define um módulo comportamental associado a um agente reativo. Encapsula um conjunto de reações relacionadas entre si, possivelmente com uma sequência temporal, de forma a atingir um ou ou mais objetivos (e.g., evitar obstáculos, recolher alvos, etc). Associado a um objetivo podem existir sub-objetivos.

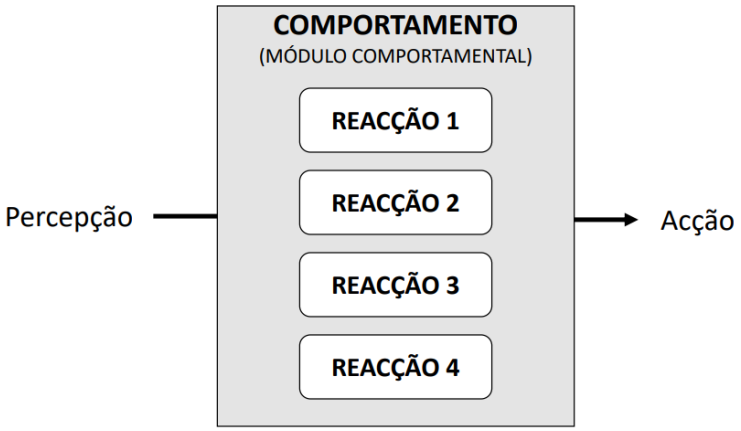

Figura 4.2: Módulo comportamental de um agente reativo. Retirado de [5], slide 11.

Tipos de Comportamento:

- **Comportamento Fixo**: Caracterizado por ter uma resposta fixa, pois gera uma ação em permanência;
- **Comportamento Simples**: Reação;
- **Comportamento Composto**: Agrega sub-comportamentos, ao qual está associado um mecanismo de seleção de ação de forma a determinar a ação a realizar em função das respostas dos mesmos.

4.1.3 Comportamento Composto

Como em comportamentos compostos, uma perceção pode ativar múltiplas reacções, as quais geram diferentes ações, é necessário definir um mecanismo de seleção de ação. Alguns dos mecanismos de seleção de ação mais comuns são:

- **Execução paralela**: As ações são executadas em paralelo porque não interferem entre si;
- **Seleção por prioridade**: As ações são selecionadas em função de uma prioridade. É possível distinguir dois tipos de comportamentos compostos com seleção por prioridade:
  - **Prioridade**: Representa uma forma de comportamento composto com um mecanismo de seleção de ação de prioridade dinâmica, e que portanto a prioridade dos comportamentos associados varia consoante o tempo (e.g., o comportamento "evitar obstáculo" tem prioridade sobre "explorar" quando o agente se encontra próximo de um obstáculo, mas "explorar" tem prioridade sobre "evitar obstáculo" quando não existem obstáculos próximos).
  - **Hierarquia**: Representa uma forma de comportamento composto com um mecanismo de seleção de ação de prioridade por hierarquia fixa de subsunção (i.e., as camadas superiores controlam as inferiores, através da inibição, supressão e/ou reinício dos comportamentos associados). Os comportamentos mais altos na hierarquia têm prioridade sobre os mais baixos (correção de acção), e não variam consoante o tempo (e.g., o comportamento "carregar bateria", tem prioridade mais alta sobre os comportamentos "explorar" ou "evitar obstáculo", pois representa um comportamento mais básico e fundamental para a sobrevivência do agente);
- **Combinação**: As ações são combinadas numa única resposta por composição (e.g., soma vectorial);
- **Seleção aleatória**: As ações são selecionadas aleatoriamente.

4.2 Controlo Reativo

Como foi apresentado na secção de arquiteturas de agentes (ver secção 2.4), um agente tem associado um módulo de controlo que é responsável por processar perceções e gerar ações. No caso de um agente reativo, o módulo de controlo é composto por um conjunto de comportamentos que são ativados em função de estímulos e que geram respostas (ver figura 4.3).

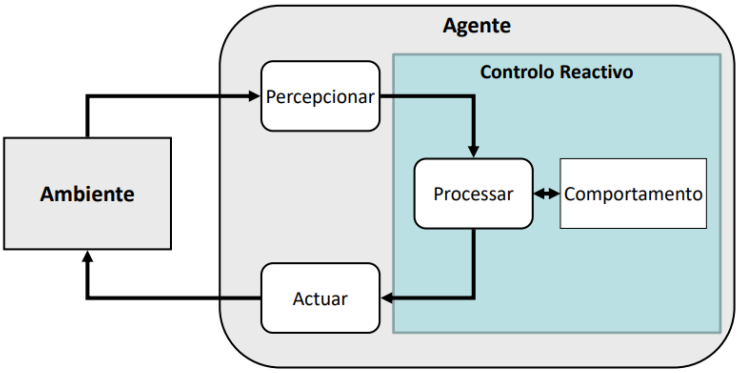

Figura 4.3: Módulo de controlo reativo. Retirado de [6], slide 11.

4.3 Arquitetura Reativa com Memória

Numa arquitectura reactiva com memória, as reações dependem não só das perceções, mas também da memória de perceções anteriores (ou de informação delas derivada) para gerar as ações. Para esse efeito é necessário manter internamente memória, a qual é atualizada a partir das perceções e das reações ativadas, influenciando essas mesmas reações, bem como as ações geradas [7].

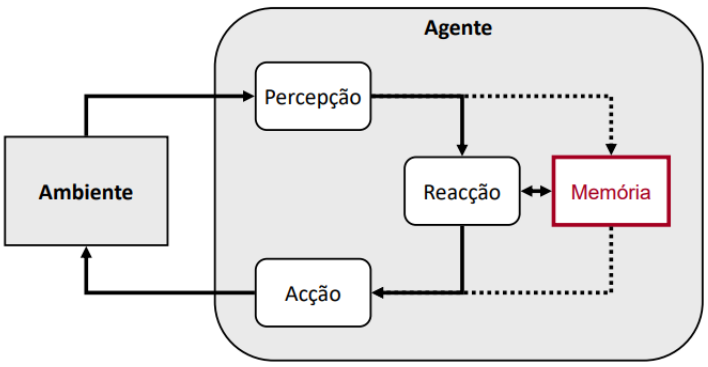

Figura 4.4: Arquitetura reativa com memória. Retirado de [7], slide 5.

O conceito de memória associado a um comportamento pode ser implementado de várias formas, como, por exemplo:

- **Estado**: As perceções anteriores que sejam relevantes são armazenadas num estado interno do comportamento, que é atualizado e consultado em função das perceções atuais e que influencia a geracão de ações;
- **Máquina de Estados**: Além de ter em conta as perceções atuais e as reações correspondentes, o comportamento pode ter em consideração entradas ou até um mecanismo de reset, que influenciam a próxima transição de estado e a consequente geração de ações e possíveis saídas. As entradas e saídas representam, neste contexto, interligações entre comportamentos.

Tabela 4.1: Vantagens e desvantagens da manutenção de estado em arquiteturas reativas [7].

Vantagens	**Desvantagens**
• Poder produzir todo o tipo de comportamentos; • Representar a evolução temporal do comportamento do agente; • Representar comportamentos complexos baseados na evolução temporal das perceções (e.g., agir devido à inexistência de um alvo após um determinado número de passos); • Lidar com falhas, explorando novas ações (e.g., por este novo caminho chegarei ao alvo?).	• Aumento da complexidade espacial; • Aumento da complexidade computacional (i.e., manter as representações de estado); • Limitações no suporte de representações complexas e exploração de planos alternativos de ação.

4.4 Biblioteca SAE

Para esta fase do projeto foi disponibilizada uma biblioteca em python que apresenta e extende os conceitos expostos pela biblioteca realizada na primeira fase do projeto (ver capítulo 3). Além disso, a biblioteca SAE (Simulador de Ambiente de Execução) permite a execução de um agente autónomo num ambiente simulado bidimensional, composto por alvos e obstáculos [8]. Oferece, além de outras funcionalidades, a possibilidade de utilizar sete configurações de ambiente predefinidas.

A interface gráfica é composta por duas áreas de visualização (ver figura 4.5): a primeira área (à esquerda) apresenta a configuração do ambiente escolhida e com a indicação do agente; e a segunda área (à direita) é uma área de visualização de informação interna do agente.

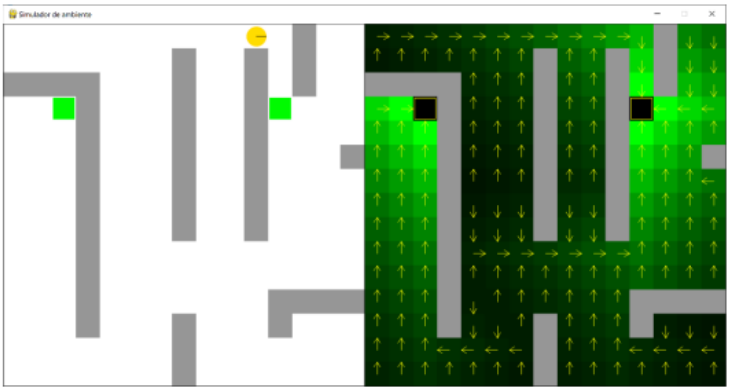

Figura 4.5: Interface gráfica do SAE. Retirado de [8], pág. 3.

4.5 Caracterização do Ambiente

O ambiente onde o agente reativo se movimenta é virtual, bidimensional, com dimensões fixas e composto por alvos e obstáculos estáticos. Pode ser caracterizado (ver secção 2.2) por ser:

- **Totalmente observável**: O agente tem acesso a toda a informação do ambiente;
- **De agente único**: O ambiente contém apenas um agente a atuar;
- **Determinístico**: O próximo estado do ambiente é determinado pelo estado atual e pela ação do agente;
- **Sequencial**: As ações do agente afetam as etapas consequentes (e.g., a recolha de um alvo implica a sua remoção do ambiente);
- **Estático**: O ambiente não muda enquanto o agente está a decidir a próxima ação a realizar;
- **Discreto**: O ambiente é composto por um número finito de estados possíveis.

4.6 Implementação do Agente Reativo

No âmbito desta fase do projeto, foram realizadas as seguintes tarefas:

1. Definição de uma biblioteca denominada ECR (Estímulos, Comportamentos e Respostas) que expõe os mecanismos base de Comportamento, Reação, Estímulo, Resposta e Comportamento Composto;
2. Implementação de um conjunto de comportamentos e subcomportamentos que permitem ao agente autónomo atingir os objetivos propostos (i.e., recolher alvos e evitar obstáculos) utilizando a biblioteca ECR;

3. Observação do comportamento do agente e da sua interação com o ambiente através da interface gráfica da biblioteca SAE;

4. Adição de comportamentos mais complexos ao controlo do agente reativo de forma a melhorar o seu desempenho tendo em conta os objetivos propostos (e.g., inicialmente o agente apenas explorava numa direção aleatória, mas posteriormente foram adicionados comportamentos compostos de recolha de alvos e evitação de obstáculos); Ocorreu em simultâneo com o ponto **3**.

Em relação ao ponto **2**, foram implementados os seguintes comportamentos:

- **Recolher Alvo**: Representa um comportamento composto com um mecanismo de seleção de ação por hierarquia fixa de prioridade, que corresponde ao objetivo do agente prospector de recolher alvos. O agente deve, por esta ordem de prioridade, ter em consideração os seguintes subcomportamentos: (1) aproximar alvo; (2) evitar obstáculos; e (3) explorar o ambiente.

- **Aproximar Alvo**: Representa um comportamento composto com um mecanismo de seleção de ação por prioridade dinâmica, que corresponde ao objetivo do agente prospector de se aproximar de um alvo. Como a distância entre o agente e os diversos alvos varia consoante a posição do agente no ambiente (i.e., a intensidade do estímulo varia), o agente deve selecionar o alvo mais próximo (i.e., prioritizar o estímulo que apresenta maior intensidade) e por isso é que é usado um mecanismo de seleção por prioridade dinâmica (ver secção 4.1.3).

- **Evitar Obstáculo**: Representa um comportamento composto com um mecanismo de seleção de ação por hierarquia fixa de prioridade, que corresponde ao objetivo do agente prospector de evitar obstáculos. É usado este mecanismo de seleção de ação, porque a próxima direção livre de obstáculos é calculada em função da posição do agente e dos obstáculos presentes à sua volta, de forma aleatória. E portanto, num dado momento, não existe uma direção livre mais prioritária (no sentido literal) que outra, mas sim, uma hierarquia onde estão definidas, pela ordem de descoberta aleatória, as direções livres de obstáculos encontradas.

- **Explorar**: Define um comportamento fixo sem memória (i.e., representação interna de perceções anteriores) e que, por isso está condenado à repetição. Está associado ao objetivo "explorar" e caracteriza-se por ter uma resposta fixa, pois gera uma ação em permanência, que consiste em mover-se numa direção aleatória. Não depende de nenhum estímulo para ser ativado.

Além dos comportamentos descritos, para os comportamentos compostos, foram implementados estímulos, respostas e reações, de forma a garantir a descrição e execução correta dos comportamentos

associados.

Em paralelo ao desenvolvimento da implementação descrita e de forma a introduzir os conceitos de arquitetura reativa com memória, foi introduzido um comportamento com a finalidade de contar os passos do agente e tomar decisões com base no número de passos efetuados. Mas tal comportamento não consta na implementação final reativo, pois não é relevante para os objetivos propostos.

4.7 Estrutura do Projeto

No processo de desenvolvimento de software associado a esta fase do projeto, foi definida a seguinte estrutura em módulos e que está presente na pasta iasa_agente/src:

- *agente*: Integra a implementação do agente reativo e do seu controlo, bem como os componentes dos módulos comportamentais associados;
- *lib/ecr*: Integra a implementação da biblioteca ECR;
- *lib/sae*: Integra a implementação da biblioteca SAE;
- ficheiro *teste.py*: Executa o agente reativo no ambiente simulado providenciado pela biblioteca SAE.

Capítulo 5

Conclusão

O projeto permitiu adquirir conhecimentos sobre inteligência artificial e a implementação de agentes autónomos inteligentes, com recurso a diferentes arquiteturas.

Outras temáticas como a arquitetura de software e as suas vertentes, bem como o processo de desenvolvimento de software, foram também abordadas, aprendidas e aplicadas ao longo do projeto, o que permitiu a aquisição de competências em áreas como a análise, o design e a implementação de sistemas de software.

Foram adquiridas competências de programação em Java e Python, especialmente vocacionadas para a programação orientada a objetos, o que permitiu modular, de forma incremental, os diferentes subsistemas das arquiteturas implementadas.

Em relação as diferentes partes do projeto, na primeira parte, foi desenvolvida uma biblioteca em Java onde foram implementados os mecanismos bases relacionados com os conceitos gerais de inteligência artificial (e.g., agente, ambiente) e um jogo que integra essa biblioteca. O jogo consistia num ambiente virtual onde a personagem tinha por objectivo registar a presença de animais através de fotografias, simulando a exploração no mundo real. O agente foi implementado com uma arquitetura reativa com memória, mais concretamente, criou-se uma máquina de estados finita no seu módulo de controlo, que permitia a tomada de decisões com base no estado anterior, em eventos pre-definidos do ambiente e na acção a executar pelo agente. Com esta arquitetura observou-se que o agente conseguia atingir os objetivos propostos, mas não conseguia reagir a situações inesperadas (i.e., eventos não previstos que podessem ocorrer no ambiente).

Na segunda parte do projeto, foi desenvolvido um agente reativo em Python com módulos comportamentais para recolher alvos e evitar obstáculos num ambiente virtual com dimensões fixas. O agente foi implementado com uma arquitetura reativa simples (i.e., sem memória), onde um comportamento composto (que englobava os subcomportamentos: aproximar alvo, evitar obstáculo, explorar) foi integrado no módulo de controlo. Esta integração permitiu a

escolha da ação a executar com base num mecanismo de seleção de ação, mais concretamente, por hierarquia fixa de prioridade. Com esta arquitetura observou-se, através da interface gráfica da biblioteca SAE, que o agente conseguia atingir os objetivos propostos, mas não de forma ótima (i.e., o caminho mais rápido para recolher todos os alvos, evitando os obstáculos, não era seguido).

Após análise dos resultados obtidos, concluiu-se que ainda não se encontrou a arquitetura ideal para o agente autónomo inteligente no contexto destes problemas, e a resposta possa estar nas arquiteturas deliberativas e híbridas que não foram abordadas neste projeto.

Referências

[1] LEIC - Departamento de Engenharia Informática, Instituto Superior Técnico. Agentes - Resumos LEIC, 2024. [Online; accessed April 16, 2024].

[2] Luis Morgado. Introdução à Engenharia de Software - Parte 3, 2024. [Online; accessed April 23, 2024].

[3] Wikipedia contributors. Finite-state machine, 2024. Online; accessed April 23, 2024.

[4] Eddie Jaoude. State Machines: Basics of Computer Science. *freeCodeCamp.org*, 2019. [Online; accessed April 23, 2024].

[5] Luis Morgado. Arquitetura de Agentes Reativos - Parte 1, 2024. [Online; accessed April 23, 2024].

[6] Luis Morgado. Arquitetura de Agentes Reativos - Parte 2, 2024. [Online; accessed April 24, 2024].

[7] Luis Morgado. Arquitetura de Agentes Reativos - Parte 3, 2024. [Online; accessed April 24, 2024].

[8] Luis Morgado. Simulador de Ambiente de Execução - Documentação, 2024. [Online; accessed April 24, 2024].