Licenciatura Engenharia Informática e Multimédia

Instituto Superior de Engenharia de Lisboa

Ano letivo 2024 / 2025

Inteligência Artificial para Sistemas Autônomos

Turma: 41D

Name: Miguel Cordeiro Number: 49765

14 de junho, 2025

Docente: Luís Morgado

Índice

Índice de Figuras:

Introdução

Este relatório enquadra-se no âmbito da unidade curricular de Inteligência Artificial para Sistemas Autónomos (IASA), lecionada no quarto semestre da Licenciatura em Engenharia Informática e Multimédia no Instituto Superior de Engenharia de Lisboa. A disciplina tem como principal objetivo o estudo e implementação de agentes autónomos com capacidades de decisão, raciocínio e aprendizagem bem como o aprofundamento sobre a Engenharia de Software.

Ao longo destas treze aulas, o projeto foi desenvolvido em quatro partes, sempre com um grau de complexidade crescente, cada uma delas acompanhada por um conjunto específico de conteúdos teóricos. A primeira parte, mais introdutória, foi realizada em Java, onde se trabalhou a lógica de controlo reactivo básico. As restantes três fases foram desenvolvidas em Python, no ambiente Visual Studio Code, onde se evoluiu para arquiteturas deliberativas, mecanismos de planeamento e, por fim, modelos com suporte a processos de decisão sequencial e aprendizagem por reforço.

O projeto seguiu uma arquitetura modular, baseada nos diagramas UML fornecidos pelo docente, com pequenas alterações sugeridas pelo mesmo durante as aulas, sendo cada componente implementado com base nas aulas práticas e na teoria lecionada. Ao longo da implementação foram aplicados conceitos como o modelo do mundo, operadores, mecanismos deliberativos, planeamento automático com procura em espaço de estados e técnicas associadas a PDMs (Processos de Decisão de Markov).

O presente relatório organiza-se em vários capítulos que refletem tanto a evolução teórica como prática da cadeira. O foco passa por explicar de forma clara as decisões de implementação, os módulos criados e a relação com os conceitos de IA aplicada a agentes autónomos, nomeadamente os temas de reatividade, deliberação, procura e aprendizagem.

O trabalho inclui a integração de todos estes conceitos num agente que observa, delibera, planeia e aprende com a experiência, demonstrando de forma prática o que foi abordado nas aulas teóricas e laboratoriais.

(Página intencionalmente deixada em branco)

Enquadramento teórico

Fundamentos da Inteligência Artificial

A inteligência artificial, também abreviada como IA, é uma área das ciências informáticas que procura criar sistemas capazes de se comportar de forma inteligente, ou seja, que consigam tomar decisões, resolver problemas ou adaptar-se ao ambiente onde estão inseridos. De forma simples, podemos dizer que a Inteligência Artificial tenta construir máquinas que consigam "pensar" ou agir de maneira semelhante à forma humana, mesmo que, na prática, isso seja feito de forma muito diferente.

Existem duas formas principais de abordar a Inteligência Artificial:

- uma mais teórica, que procura perceber como funcionam os processos inteligentes
- outra mais prática, que tenta construir sistemas capazes de se comportar de forma inteligente.

Estas duas abordagens completam-se uma à outra. Entender como funciona os sistemas de Inteligência Artificial ajudam-nos a construir programas melhores, e construir programas que também nos ajudam a compreender melhor a inteligência.

Paradigmas da Inteligência Artificial

Ao longo do tempo, foram surgindo diferentes formas de olhar para a inteligência artificial, chamadas de paradigmas. Cada um tem a sua maneira de explicar como pode surgir um comportamento inteligente:

- **Paradigma simbólico:** É o mais tradicional e baseia-se na ideia de que a inteligência pode ser representada através de símbolos e regras. Aqui, os sistemas tomam decisões a partir de representações explícitas do conhecimento.
- **Paradigma conexionista**: Este inspira-se no funcionamento do cérebro humano. Em vez de símbolos, utiliza redes de neurónios artificiais, onde o conhecimento está espalhado pelas ligações entre unidades.
- **Paradigma comportamental**: Foca-se na relação direta entre estímulos e respostas. Um sistema é considerado inteligente se conseguir reagir de forma apropriada ao que acontece à sua volta, mesmo sem fazer raciocínios complicados ou ter representações internas.

O projeto que desenvolvemos nesta cadeira utiliza muito este último paradigma, o paradigma comportamental, principalmente na parte dos agentes reativos, que serão explicados mais à frente.

Paradigmas da Inteligência Artificial

Um agente inteligente é um sistema que consegue perceber o ambiente onde está, decidir o que fazer e agir conforme a necessidade. Pode viver num ambiente físico (como um robô) ou num ambiente virtual (como um personagem num jogo). O seu funcionamento baseia-se num ciclo que se repete continuamente: Percepcionar/Processar/Actuar (Figura 1).

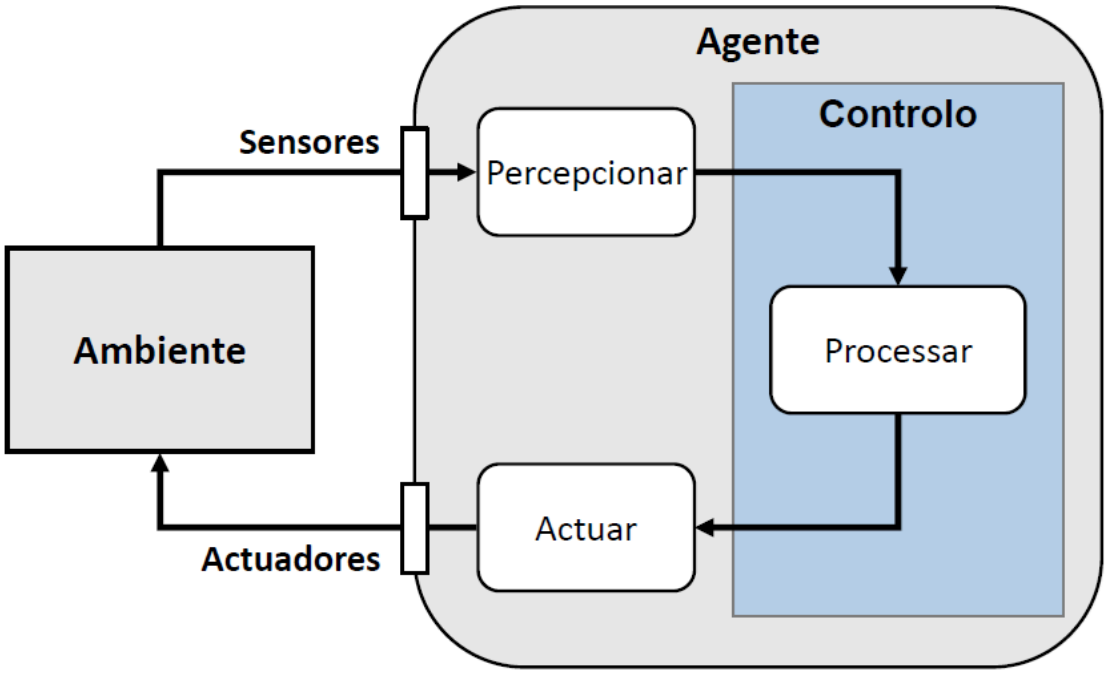


Figura 1 – Diagrama de um Agente Inteligente (Slides Professor Luís Morgado)

Para que um agente seja considerado inteligente, há algumas características importantes que deve ter:

- **Autonomia**: consegue funcionar sozinho, sem depender de ordens externas.
- **Reatividade**: responde de forma reactiva quando o ambiente muda.
- **Pró-atividade**: não está apenas à espera de estímulos, toma assim também iniciativas.

- **Sociabilidade**: pode comunicar e colaborar com outros agentes.

Estas características vão depender da forma como o agente é construído, ou seja, da sua arquitetura interna, que veremos já a seguir.

Percepção e Decisão

Ao falarmos da inteligência artificial, é importante entender também dois conceitos que aparecem muitas vezes:

- A **cognição** diz respeito à capacidade de adquirir e usar informação de forma útil. Nos agentes vê-se na forma como eles percebem o que se passa à volta e decidem o que fazer.

- A **racionalidade** está relacionada com a ideia de tomar boas decisões. Um agente é racional quando escolhe a ação que, dentro do que sabe, o aproxima mais dos seus objetivos. Isso pode ser feito com base em regras simples como nas reações estímulo-resposta ou com decisões mais complexas, planeadas com antecedência.

Arquiteturas de Agente

A arquitetura de um agente é a forma como ele está organizado internamente, ou seja, como é que os vários componentes (perceção, decisão, ação) estão ligados e funcionam em conjunto.

Existem três modelos principais:

- **Modelo Reativo**: o mais simples. O agente responde diretamente aos estímulos, sem pensar muito ou guardar memória, os objetivos são implícitos. É rápido, mas limitado.

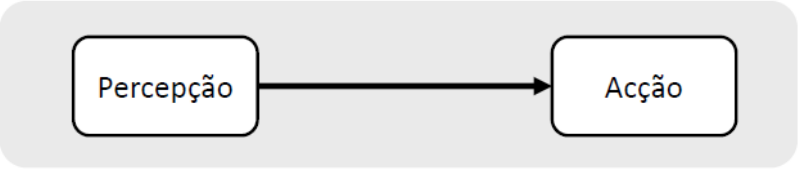


Figura 2 – Diagrama do Modelo Reactivo (Slides Professor Luís Morgado)

- **Modelo Deliberativo**: aqui o agente tem uma representação interna do mundo e consegue fazer planos com base em objetivos explícitos. É mais "inteligente", mas também mais lento.

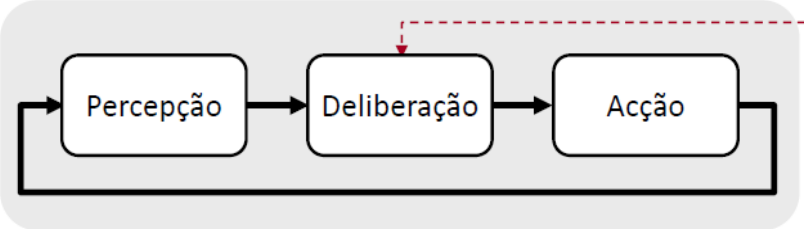


Figura 3 – Diagrama do Modelo Deliberativo (Slides Professor Luís Morgado)

- **Modelo Híbrido**: junta o melhor dos dois anteriores, responde rapidamente quando precisa, mas também consegue planear quando dá jeito, os objetivos são explícitos como no Modelo Deliberativo.

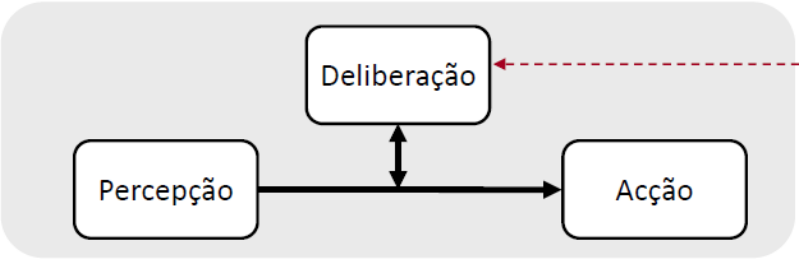


Figura 4 – Diagrama do Modelo Híbrido (Slides Professor Luís Morgado)

Engenharia de Software

Quando se desenvolve um sistema inteligente, como os agentes que implementámos neste projeto, não basta que ele funcione, é fundamental que o seu código seja bem organizado, fácil de manter e preparado para evoluir. É aqui que entra a **Engenharia de Software**, uma área que nos ajuda a lidar com a complexidade do desenvolvimento de software, usando métodos e boas práticas da programação que tornam todo o processo mais controlado e eficiente.

Importância da Engenharia de Software na IA

Sistemas com inteligência artificial tendem a ser bastante complexos, porque muitas vezes lidam com ambientes imprevisíveis e precisam de tomar decisões em tempo real. Isto exige uma estrutura bem pensada, que permita:

- Gerir a complexidade à medida que o sistema cresce;
- Fazer alterações sem comprometer o que já está feito;
- Reaproveitar código de forma inteligente;

- Testar e validar comportamentos.

Durante o projeto, aplicámos vários destes princípios, especialmente quando seguimos os diagramas UML propostos nas aulas com as alterações específicas comunicadas pelo professor.

Abstracção, modularização e Factorização

Três conceitos fundamentais da Engenharia de Software que usámos bastante foram a abstracção, modularização e a factorização.

- **Abstração** significa focarmo-nos apenas no que é relevante, escondendo detalhes que, naquele momento, não interessam. Por exemplo, um módulo de comportamento pode apenas dizer "avançar", sem precisar de explicar como é que isso acontece exatamente.

- **Modularização** é a divisão do sistema em partes mais pequenas (módulos), cada uma com uma função bem definida. Isto facilita o desenvolvimento e permite-nos trabalhar em diferentes partes do projeto sem criar confusão. No nosso caso, cada reação, estímulo ou resposta foi implementada como um módulo separado.

- **Factorização**: ajuda a eliminar código repetido, substituindo-o por partes comuns reutilizáveis. Isto evita redundância, torna o código mais limpo e reduz o risco de erros difíceis de manter

Estes três conceitos geram a complexidade devida para o projeto, eliminando assim a complexidade desorganizada e controlar melhor a complexidade organizada necessária para o funcionamento do sistema.

Diagramas UML (Unified Modeling Language)

A linguagem UML é uma ferramenta usada para modelar o sistema antes de o programarmos, como referido no conceito "Modularização". Em vez de escrever diretamente código, usamos **diagramas** para representar e facilitar a implementação com boas práticas:

- As classes e as relações entre elas (diagrama de classes).

- O comportamento do sistema em diferentes situações (diagrama de estados).

- As interações entre componentes ao longo do tempo (diagrama de sequência).
- O fluxo de decisões ou atividades (diagrama de atividades).

Durante o projeto, os diagramas UML ajudaram-nos a planear melhor os comportamentos e perceber como cada parte do agente se relacionava com as outras. Também foram úteis para garantir que a implementação seguia uma estrutura coerente e flexível.

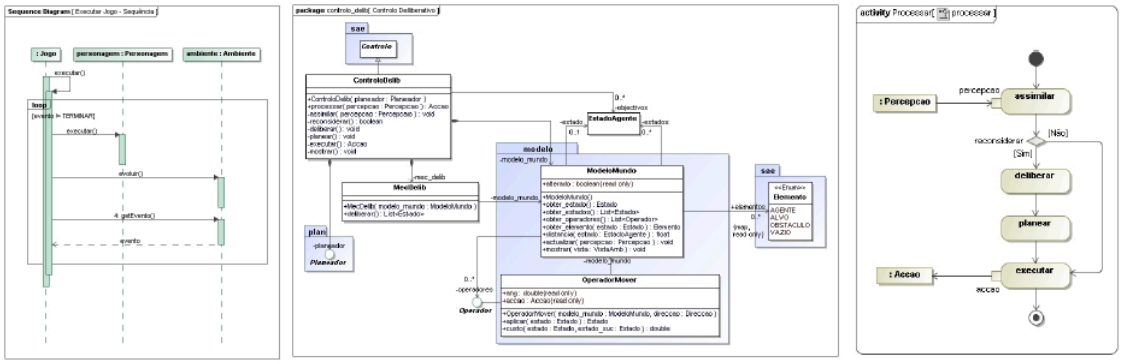


Figura 5 – Exemplos de Diagramas UML (Slides Professor Luís Morgado)

Organização, qualidade e manutenção

Outro aspecto importante foi o cuidado com a qualidade do código: seguir uma boa arquitetura não é só uma questão de organização, mas também dr garantir que o sistema é fácil de manter e expandir. Isto tornou-se evidente nas fases mais avançadas do projeto, quando começámos a adicionar memória, mecanismos de planeamento ou comportamentos mais complexos, se a base não estivesse bem estruturada, tudo se tornaria muito mais difícil.

Usámos também conceitos como:

- **Acoplamento**: mede o grau de dependência entre diferentes partes do sistema. Quanto mais acoplado estiver o código, mais difícil será alterá-lo sem provocar efeitos noutros lados. O ideal é manter o acoplamento baixo, usando, por exemplo, encapsulamento e interfaces bem definidas.

- **Coesão**: diz respeito a quão bem relacionadas estão as funções dentro de um módulo. Um módulo com alta coesão tem partes que servem todas para o mesmo propósito, o que facilita a leitura, manutenção e reutilização do código.

- **Simplicidade**: está relacionada com a facilidade com que se percebe como o sistema está organizado. Uma arquitetura simples facilita a comunicação entre os membros da equipa e reduz o tempo necessário para perceber ou modificar o código.

- **Adaptabilidade**: avalia quão fácil é modificar a estrutura do sistema para acomodar novos requisitos. Um sistema adaptável está preparado para evoluir sem que isso exija reescrever tudo de raiz.

Estas métricas são especialmente úteis quando queremos avaliar a qualidade do nosso sistema à medida que ele cresce e se torna mais complexo.

Processo de Desenvolvimento

O desenvolvimento de software segue normalmente um processo iterativo, que passa por três fases principais: modelação, arquitetura e implementação. Cada uma destas fases permite refinar o sistema de forma progressiva, adaptando-se ao que vamos aprendendo durante o desenvolvimento.

Durante o projeto, este processo foi bastante visível. Começámos por seguir os modelos UML fornecidos (modelos conceptuais), depois organizámos a arquitetura com base nesses modelos, e por fim, fomos implementando os comportamentos e reações, sempre de forma incremental.

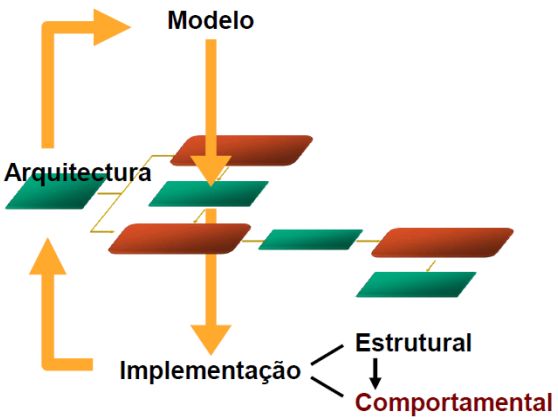


Figura 6 – Processo de Desenvolvimento de Software (Slides Professor Luís Morgado)

Arquitetura de Agentes Reactivos

Os agentes reativos foram o primeiro tipo de agente que estudámos e implementámos neste projeto. A sua principal característica é o facto de reagirem diretamente ao ambiente, ou seja, ao que percebem à sua volta, sem necessidade de representações internas complexas ou planeamento a longo prazo.

Este tipo de agente é bastante eficaz em ambientes dinâmicos e imprevisíveis, porque consegue responder rapidamente a estímulos, mesmo que essa resposta não seja a mais "inteligente" no sentido clássico. É uma arquitetura simples, mas bastante poderosa quando bem organizada.

Processo de Desenvolvimento

Na arquitetura reativa, o comportamento do agente é construído através de associações entre estímulos (percepções) e respostas (ações). Isto significa que, quando o agente detecta um determinado estímulo (como um obstáculo ou um alvo) responde de forma automática com uma ação pré-definida, como rodar ou avançar.

Não existe um processo de deliberação nem um modelo do mundo guardado internamente. O agente atua em tempo real com base no que "vê" naquele momento.

Reacções, Estímulos e Respostas

As reações podem envolver processamento mais complexo, incluindo a manutenção de estado interno com base em mecanismos de memória

Assim os vários elementos envolvidos numa reacção devem ser modularizados na seguinte sequência de elementos:

- **Estímulo**: define o que o agente está à procura numa determinada direção (exemplo: "existe um obstáculo à frente?", "existe um alvo a norte?").

- **Resposta**: define a ação a realizar (exemplo: rodar, avançar, apanhar alvo).
- **Reação**: associa um estímulo a uma resposta.

Cada reação pode ser vista como uma regra do tipo "se acontecer X, então faz Y", e a combinação destas reações permite definir comportamentos mais complexos.

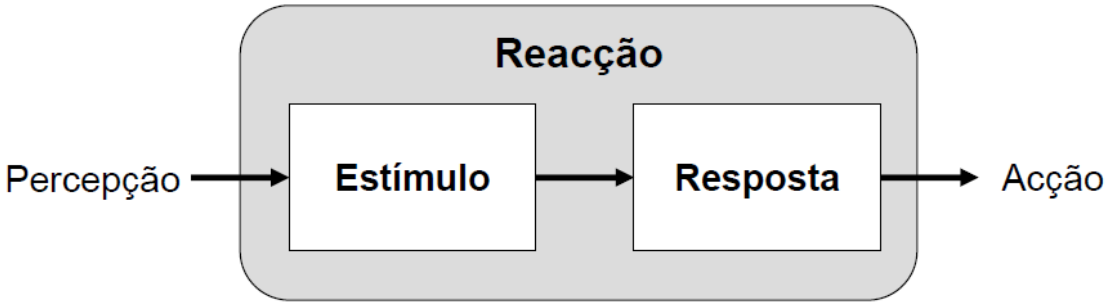


Figura 7 – Regra Estímulo / Resposta (Slides Professor Luís Morgado)

Comportamentos e Hierarquias

As reações individuais são organizadas em comportamentos, que agrupam reações relacionadas com o mesmo objetivo, como por exemplo:

- **EvitarObst**: reações para evitar obstáculos em várias direções.
- **AproximarAlvo**: reações para se aproximar de um alvo detectado.
- **Explorar** e **ExplorarMem**: ações de exploração aleatória, com ou sem memória.

Depois, estes comportamentos podem ser organizados em comportamentos compostos, onde há uma hierarquia ou uma lógica de coordenação. Um exemplo claro é o comportamento Recolher, que inclui:

- AproximarAlvo (prioridade mais alta),
- EvitarObst (intermédio),
- Explorar e ExplorarMem (último recurso).

Seleção da Acção

Quando vários comportamentos geram respostas em simultâneo, é preciso decidir qual executar. Os nossos agentes usam diferentes estratégias de seleção de ações:

- **Hierarquia**: onde as ações ou regras são organizadas em níveis de prioridade. O agente avalia os estímulos recebidos e verifica, em ordem hierárquica, qual comportamento é aplicável.
- **Prioridade**: executa-se a ação do comportamento com maior importância naquele momento.
- **Combinação**: em alguns casos, podem combinar-se ações (por exemplo, somando vetores de direção).

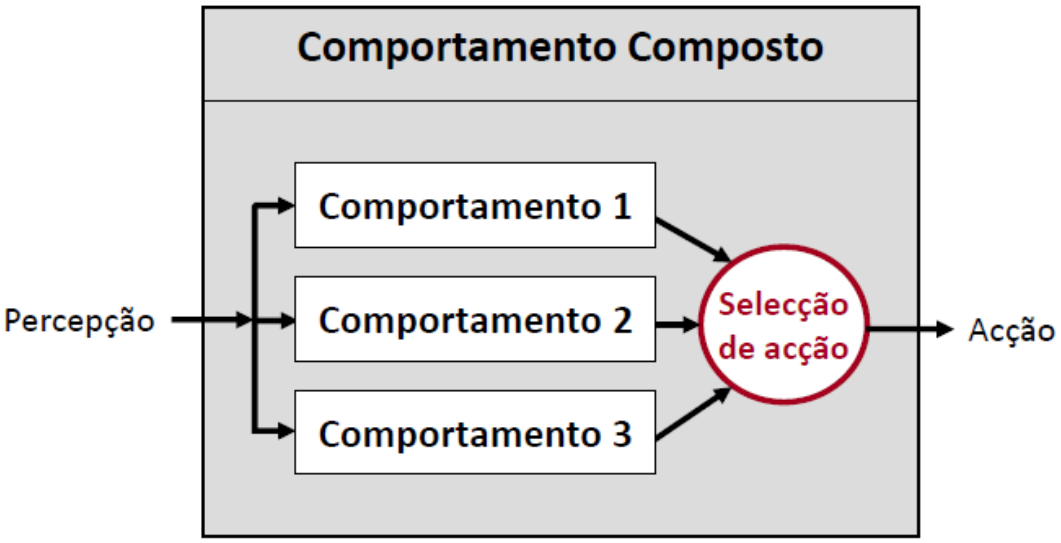


Figura 8 – Seleção da Acção (Slides Professor Luís Morgado)

Comportamentos com Memória

Até aqui, vimos como os agentes reativos funcionam exclusivamente com base no que percebem no momento. No entanto, este tipo de funcionamento apresenta limitações quando aplicamos os agentes a tarefas mais complexas, como explorar ambientes desconhecidos, onde seria útil "lembrar-se" do que já foi percorrido para evitar caminhos repetidos ou becos sem saída.

A ausência de memória torna o agente totalmente dependente dos seus sensores e, por consequência do ambiente onde se encontra. Isto significa que, em certos contextos, o agente pode acabar a repetir os mesmos comportamentos indefinidamente, sem conseguir alcançar o objetivo de forma eficiente.

Para resolver este problema, é possível estender a arquitetura reativa com um módulo de memória, que guarda informação sobre percepções anteriores. Esta memória colabora com o módulo de reação e permite ao

agente tomar decisões tendo em conta não só o que está a acontecer no momento, mas também o que já aconteceu. Desta forma, é possível definir ações que combinam percepções atuais e passadas.

A nível prático, esta abordagem leva à criação de reacções com estado interno, ou seja, reacções que guardam alguma informação entre ativações. A lógica de ativação dessas reacções pode então ser gerida através de máquinas de estados, que organizam a evolução do comportamento ao longo do tempo.

Teriam vantagens como:

- Permite ao agente desenvolver comportamentos mais ricos e variados;
- As respostas podem evoluir com base na experiência passada;
- Torna possível implementar estratégias de exploração mais eficientes.

Embora desvantagens como:

- A complexidade do sistema aumenta, tanto a nível computacional como de memória;
- A representação interna ainda é limitada (mesmo com memória), o agente continua sem capacidade para representar objetivos ou planear alternativas.

Arquitetura de Subsunção

Com a introdução de memória e dos mecanismos de seleção de ações, também se torna necessário organizar melhor os comportamentos do agente. Uma forma de o fazer é através da arquitetura de subsunção.

Neste modelo, os comportamentos são organizados por camadas, onde cada camada representa um nível de competência. A camada mais elevada tem prioridade total e pode anular ou substituir o comportamento das camadas inferiores. Esta hierarquia garante que apenas os comportamentos mais adequados são ativados em cada momento, reduzindo conflitos.

Cada camada funciona de forma independente, sem conhecimento do que se passa acima. Isto ajuda a manter a complexidade do sistema sob controlo, mesmo quando o número de comportamentos aumenta.

Mecanismos de controlo usados nesta arquitetura:

- **Inibição**: impede que certos módulos comuniquem, dependendo do estado da camada superior;

- **Supressão**: desativa todas as camadas inferiores se uma camada superior estiver ativa;

- **Reinício**: permite reiniciar o estado interno de um comportamento sempre que necessário.

Além da hierarquia, cada comportamento mantém a sua estrutura reativa, respeitando a lógica estímulo/resposta. No entanto, a coordenação entre eles é feita de forma mais controlada, com base numa máquina de estados aumentada, que interpreta as percepções recebidas e decide qual o comportamento mais adequado a executar.

Importa referir que, uma vez definida a hierarquia dos comportamentos, esta estrutura mantém-se fixa durante a execução do agente, ou seja, a prioridade entre comportamentos não muda dinamicamente, o que traz previsibilidade, mas também alguma rigidez.

Raciocínio Automático

À medida que os sistemas autónomos evoluem, torna-se necessário equipá-los com a capacidade de resolver problemas de forma mais inteligente, indo além das simples reações ao ambiente. O raciocínio automático trata da resolução de problemas com base em conhecimento representado internamente e em processos de inferência.

Em termos simples, podemos dizer que um sistema com raciocínio automático é capaz de pensar nas possíveis opções, simular resultados, avaliar alternativas e, por fim, escolher o caminho mais adequado para atingir um determinado objetivo.

O que é o Raciocínio Automático?

Raciocinar automaticamente significa que o sistema, ao receber um problema, é capaz de representar esse problema internamente, conseguir explorar as diferentes possibilidades de resolução, consecutivamente avaliar os resultados de cada opção e, por fim, chegar a uma solução adequada de forma autónoma.

Este processo exige que o sistema tenha uma representação simbólica do domínio do problema, ou seja, uma forma de descrever internamente as

situações, as ações e os objetivos. A manipulação dessas representações, para tirar conclusões ou prever resultados, é descrevida como inferência.

Explorar e avaliar opções

O raciocínio automático é composto por duas grandes atividades principais:

- **Exploração**: o sistema analisa diferentes caminhos possíveis, de forma antecipada, tentando prever o que aconteceria se tomasse determinada ação (raciocínio prospectivo).
- **Avaliação**: cada opção é analisada com base em critérios como custo (esforço) e valor (ganhos). Esta análise ajuda o sistema a decidir qual a melhor ação a seguir.

Este tipo de raciocínio é essencial em tarefas como navegação autónoma, planeamento robótico, jogos ou controlo de personagens em ambientes virtuais.

Representar o problema internamente

Para que um sistema raciocine, tem de conseguir representar internamente:

- O **estado atual** (como estão as coisas no momento);
- Os **operadores** (ações possíveis que alteram o estado);
- O **objetivo** (estado que queremos atingir).

Estas representações criam o que se chama de modelo do problema. A partir daí, o sistema pode simular ações e prever os estados seguintes, até encontrar um caminho que o leve do estado inicial até ao objetivo.

Espaço de Estados

Quando todas as situações possíveis (estados) e todas as ações possíveis (operadores) são representadas, formam um espaço de estados. Este espaço pode ser representado como um grafo, onde:

- Cada nó representa um estado;

- Cada ligação representa uma ação (operador) que transforma um estado noutro.

A procura de uma solução para o problema passa por navegar neste grafo até encontrar o caminho que leva ao estado objetivo. Esta navegação pode seguir diferentes estratégias.

Procura em Espaço de Estados

Depois de compreender o que é o raciocínio automático, para o agente encontrar uma solução dentro de todas estas possibilidades a resposta será nas técnicas de procura em espaço de estados, uma parte essencial da inteligência artificial que nos permite explorar diferentes caminhos até chegar a um objetivo.

Este tipo de procura é fundamental sempre que temos de tomar decisões com base em múltiplas opções possíveis, seja para resolver um puzzle, planear um caminho ou reorganizar objetos num ambiente.

O que é um Espaço de Estados?

O espaço de estados é um modelo que representa todas as situações possíveis de um problema, assim como as ações que permitem passar de uma situação para outra. Um mapa em que cada ponto representa uma configuração do sistema (um estado), e as estradas entre os pontos são as ações (ou operadores) que permitem mudar de um estado para outro.

Este espaço é normalmente representado como um grafo, onde:

- Os nós são os estados;
- As arestas são os operadores (ações);
- O estado inicial é o ponto de partida;
- O estado objetivo é o destino que queremos atingir.

A tarefa do agente é encontrar um caminho válido e eficiente desde o estado inicial até ao estado objetivo.

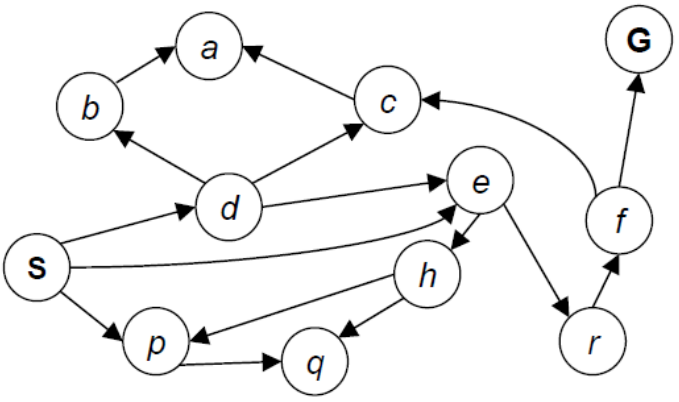


Figura 9 – Espaço de Estados (Slides Dr. Gabor Hullam)

Árvore de Procura

Durante o processo de procura, é importante manter o registo da informação gerada em cada passo, nomeadamente os estados por onde o agente já passou e as ações que executou para lá chegar. Para isso, utiliza-se uma estrutura chamada árvore de procura.

Esta árvore organiza-se em nós, onde cada nó guarda:

- O estado atual;
- O operador (ou ação) que foi aplicado para chegar a esse estado;
- O nó anterior (ou seja, o estado de onde se partiu);
- A profundidade do nó (quantos passos o separa da raiz);
- E o custo acumulado (se a procura considerar custos).

A principal utilidade desta estrutura é permitir reconstruir o caminho desde o estado inicial até a um estado qualquer, uma vez que cada nó está ligado ao seu antecessor. Além disso, a profundidade e o custo ajudam a controlar a forma como o espaço de estados está a ser explorado.

Diferença entre Grafo de Espaços e Árvore de Procura

Quando falamos em problemas de procura, é importante distinguir dois conceitos que, apesar de relacionados, são diferentes: o grafo de espaço de estados e a árvore de procura.

- O grafo de espaço de estados representa todas as possíveis configurações do problema (os estados) e as ligações diretas entre

eles (as ações ou operadores). Cada estado aparece apenas uma vez, e os caminhos entre estados refletem as possibilidades reais de transição.

- Já a árvore de procura é a estrutura construída pelo agente à medida que explora o problema. Aqui, cada nó representa um caminho completo desde o estado inicial até ao estado atual. Isso significa que o mesmo estado pode aparecer várias vezes na árvore, se for alcançado por caminhos diferentes.

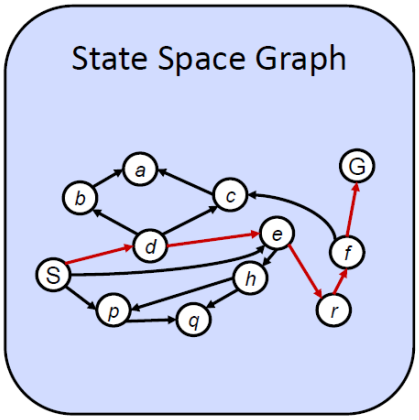


Figura 10 – Grafo de Espaços

(Slides Dr. Gabor Hullam)

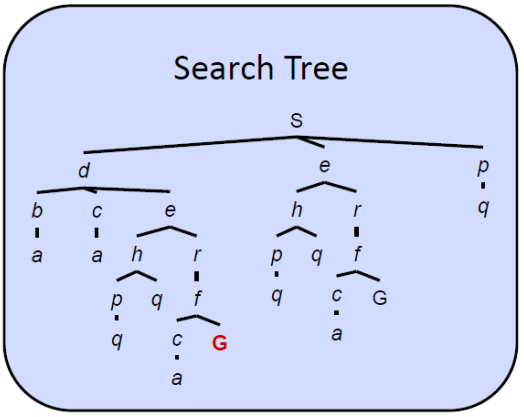


Figura 11 – Árvore de Procura

(Slides Dr. Gabor Hullam)

Componentes da procura

Para aplicar uma estratégia de procura, o agente precisa de:

- Um estado inicial;
- Um conjunto de operadores (ações permitidas);
- Um ou mais estados objetivo (ou uma função que os reconheça);
- Um método para gerar sucessores, ou seja, os estados que resultam de aplicar cada operador ao estado atual;
- Uma estratégia de exploração, que define a ordem em que os estados são visitados.

Estes elementos são comuns a todas as formas de procura, sendo a diferença o modo como se prioriza a visita a cada estado.

Tipos de Procura

Existem várias estratégias que o agente pode usar para explorar este espaço. As mais simples são chamadas de **procura não informada**, porque não usam qualquer conhecimento sobre o problema além da sua estrutura. As mais sufisticadas são a **procura informada**, onde se tenta guiar a exploração com base em algum tipo de estimativa ou heurística.

Procura Não Informada

As estratégias de procura não informada são aquelas em que o agente não tem qualquer conhecimento adicional sobre o problema além da sua estrutura. Isto significa que o agente não faz ideia de onde está o objetivo, limita-se a explorar o espaço de estados de forma sistemática.

Estas estratégias garantem que, se existir uma solução, ela será eventualmente encontrada. No entanto, nem sempre o fazem da forma mais eficiente em termos de tempo ou memória.

Procura em Largura (Breadth-First Search)

Explora primeiro todos os nós ao nível da raiz antes de descer na árvore, por camadas (Figura 12), e garante encontrar a solução com menor número de passos (ótima se todos os passos tiverem o mesmo custo), no entanto pode consumir muita memória porque guarda todos os nós do nível atual.

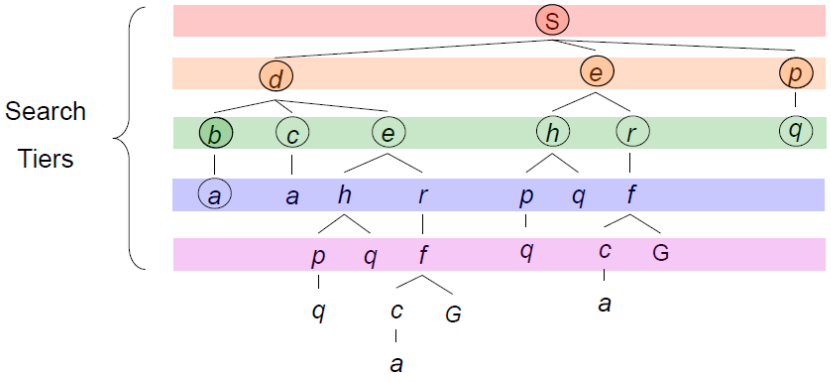


Figura 12 – Exploração por Camadas

(Slides Dr. Gabor Hullam)

Procura em Profundidade (Depth-First Search)

Vai o mais fundo possível num ramo antes de voltar atrás (Figura 13), usando pouca memória, mas pode entrar em ciclos ou seguir caminhos muito longos e desnecessários, por fim não garante encontrar a solução mais curta.

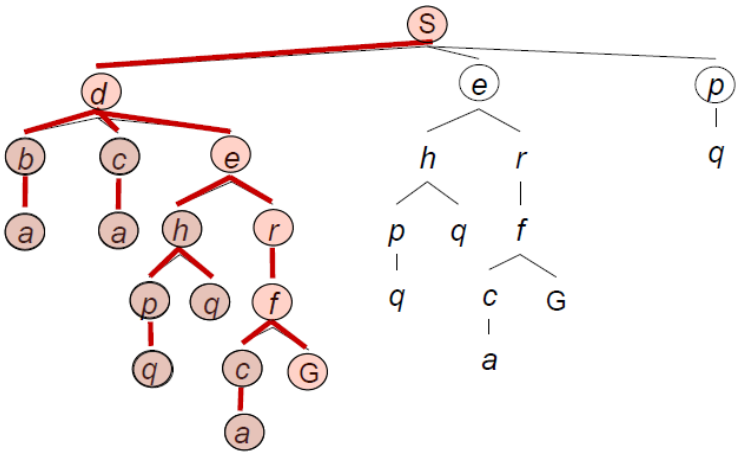


Figura 13 – Exploração total de um Ramo

(Slides Dr. Gabor Hullam)

Procura em Profundidade Limitada

Igual à procura em profundidade, mas com um limite máximo de profundidade (Figura 13), evita alguns dos problemas da Procura em Profundidade (como os ciclos infinitos), mas pode falhar se a solução estiver para além do limite.

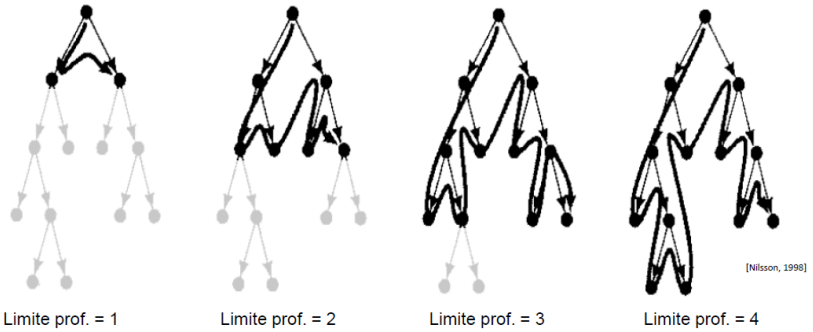


Figura 14 – Exploração com Limite de Profundidade

(Slides do Professor Luís Morgado)

Procura em Profundidade Iterativa

Combina as vantagens da procura em largura e da procura em profundidade, fazendo várias buscas em profundidade, começando com profundidade 0, depois 1, depois 2, e assim por diante (Figura 14), garantindo completude e otimidade (como a procura em largura), mas com menos uso de memória.

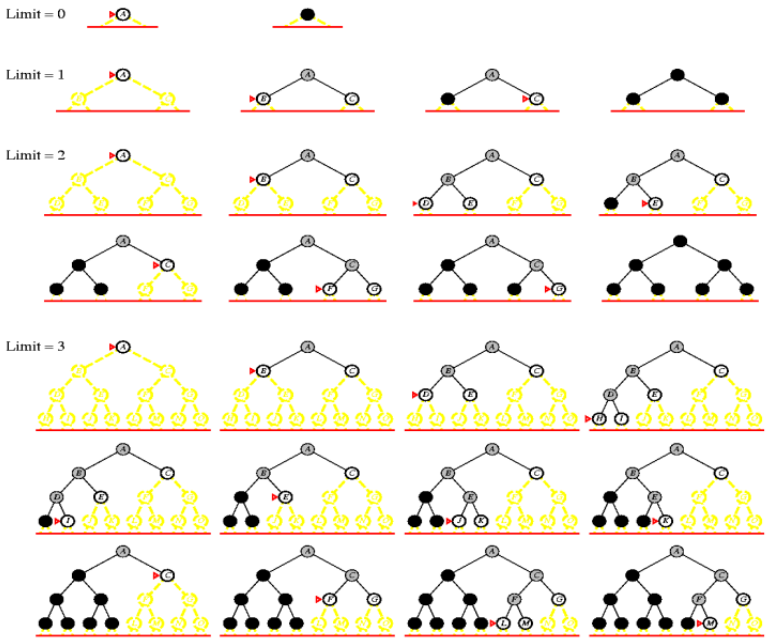


Figura 15 – Procura em Profundidade Iterativa

(Slides do Professor Lucas Cambuim)

Heurística e Custo

Antes de avançarmos para as estratégias de procura informada, é importante perceber dois conceitos fundamentais: heurística e custo de caminho.

Heurística

Uma heurística é uma função que faz uma estimativa de quão longe estamos do objetivo.
É uma aproximação, não uma garantia, mas ajuda o agente a orientar a sua procura.
Exemplos clássicos incluem:

- **Distância de Manhattan** (usada em grelhas);

- **Distância Euclidiana** (para movimentos contínuos ou diagonais).

Cada heurística é desenhada à medida do problema, e a sua qualidade influencia diretamente a eficiência da procura.

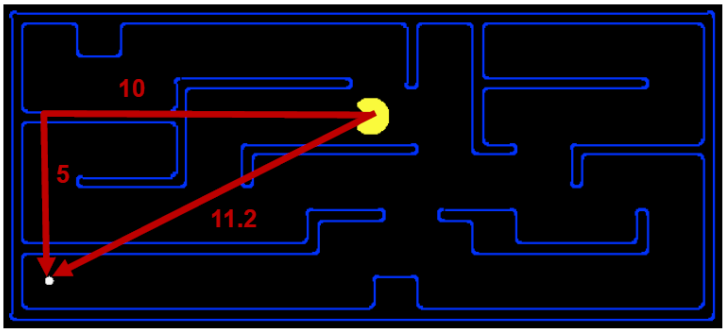


Figura 16 – Função da Heurística

(Slides Dr. Gabor Hullam)

Custo

O custo representa o valor para passar de um estado para outro. Pode medir a distância física, tempo, energia, ou outro recurso relevante para o problema.

Algumas estratégias dão prioridade a caminhos com menor custo total acumulado, independentemente da profundidade ou da distância ao objetivo.

Procura Informada

A procura informada usa conhecimento adicional sobre o problema para orientar a procura. Este conhecimento serve como uma função heurística, que estima quão longe estamos do objetivo.

Esta abordagem costuma ser mais rápida e eficiente, especialmente em espaços de estados muito grandes, porque evita explorar caminhos que claramente não levam a lado nenhum.

Nesta procura temos um grande grupo chamado o grupo da Procura Melhor Primeiro (Best-First Search), este grupo inclui várias estratégias que, a cada

passo, escolhem expandir o nó que parece mais promissor, com base numa função de avaliação.

Procura em Custo Uniforme (Uniform Cost Search)

Expande sempre o nó com o menor custo acumulado desde a raiz, garantindo a solução mais barata, independentemente da profundidade. É uma versão da Procura em Largura onde se considera o custo real do caminho.

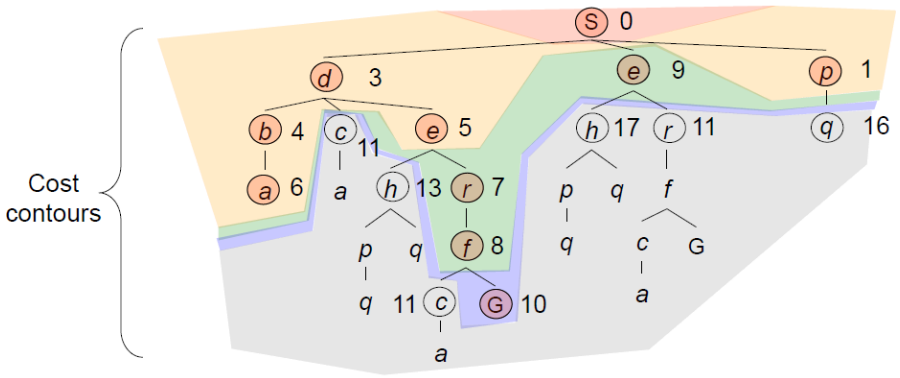


Figura 17 – Custo Uniforme com contornos de custo acumulado

(Slides Dr. Gabor Hullam)

Procura Sôfrega (Greedy Search)

Usa apenas a heurística (estimativa da distância até ao objetivo), escolhendo o nó que parece mais próximo da solução, esta tem a vantagem de poder ser muito rápida, mas não garante que a solução encontrada seja a melhor.

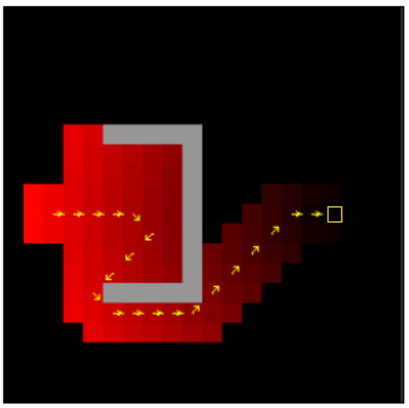


Figura 18 – Procura Sôfrega

(Slides do Professor Luís Morgado)

Procura A* (A-Star Search)

Combina o custo já percorrido com a estimativa até ao objetivo:
f(n) = g(n) + h(n)

onde:

- "f(n)" é a função de avaliação usada para decidir o próximo nó a expandir.
- "g(n)" é o custo real até ao nó;
- "h(n)" é a heurística (estimativa do que falta);

Quando usada com uma heurística admissível, encontra a solução ótima, sendo uma das estratégias mais poderosas e populares na Inteligência Artificial.

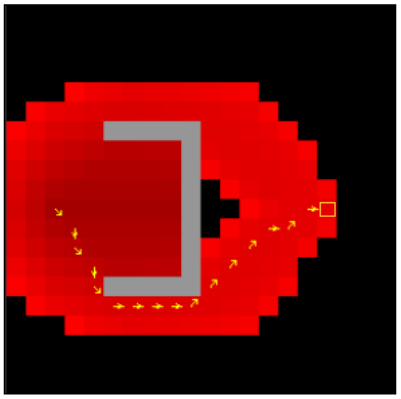


Figura 19 – Procura A*

(Slides do Professor Luís Morgado)

Diferenças entre as Procuras Informadas

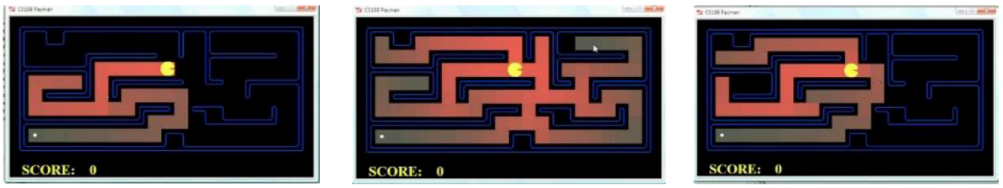
Apesar de todas seguirem o princípio da procura informada, cada uma destas estratégias apresenta características distintas em termos de eficiência, garantia de solução e uso de recursos.

A procura em custo uniforme tem como principal vantagem o facto de garantir sempre a solução com menor custo, desde que os custos dos operadores sejam positivos. No entanto, pode tornar-se lenta, pois tende a explorar muitos caminhos com o mesmo custo antes de chegar ao objetivo.

Já a procura sôfrega é geralmente mais rápida, porque se guia apenas pela heurística, tentando sempre aproximar-se do objetivo. Contudo, essa rapidez tem um preço. Esta estratégia não garante a melhor solução e pode

facilmente seguir caminhos enganadores, se a heurística não for bem escolhida.

Por sua vez, a procura A* combina o melhor dos dois mundos, ao considerar tanto o custo já percorrido como a estimativa do que falta. Quando usada com uma heurística admissível, A* consegue garantir a solução ótima, sendo ao mesmo tempo mais eficiente do que a procura em custo uniforme. Ainda assim, o seu maior ponto fraco está no consumo de memória, que pode crescer rapidamente em problemas mais complexos.



Greedy Uniform Cost A*

Figura 20 – Comparação das 3 Pesquisas Informadas

(Slides Dr. Gabor Hullam)

Arquitetura de Agentes Deliberativos

À medida que os desafios enfrentados pelos agentes autonomos se tornam mais exigentes, é necessário que estes sejam capazes de mais do que apenas reagir ao ambiente. Precisam de pensar, ponderar alternativas, planear passos futuros e tomar decisões fundamentadas. Diferente da arquitetura reactiva, podemos usar então a arquitetura deliberativa, uma estrutura interna que permite ao agente agir com base em raciocínio, objetivos e simulação do futuro.

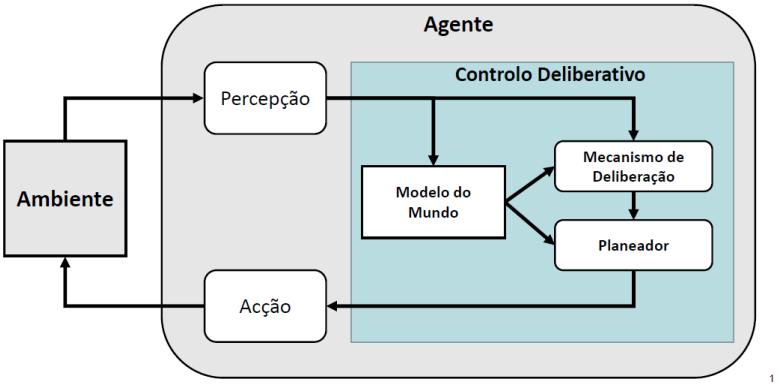


Figura 21 – Arquitetura Deliberativa

(Slides do Professor Luís Morgado)

Raciocínio Prático (Pensar para Agir)

O raciocínio dos agentes deliberativos é chamado de raciocínio prático porque está diretamente orientado para a ação.

- **Deliberação:** o agente decide o que quer fazer, escolhendo os objetivos que pretende atingir;
- **Planeamento:** o agente define como vai fazer, escolhendo as melhores ações para alcançar esses objetivos.

Conceito Modelo do Mundo

O modelo do mundo é a representação interna que o agente constrói com base no que observa do ambiente. Este modelo guarda informação relevante como a sua própria posição, a localização de obstáculos, alvos, zonas visitadas, entre outros, permitindo ao agente tomar decisões com base no que já sabe.

O modelo do mundo é essencial para agentes deliberativos, pois serve de base tanto para escolher objetivos (deliberação), como para planear ações futuras (planeamento).

Ciclo Deliberativo

1. **Observar:** captar percepções através dos sensores;
2. **Atualizar:** integrar a nova informação do modelo do mundo com o conhecimento já existente;
3. **Deliberar:** escolher quais os objetivos que vale a pena atingir;
4. **Planear:** gerar um plano de ações concreto;
5. **Executar:** meter o plano em prática.

Exploração e Avaliação de opções

Para planear, o agente precisa de conseguir simular algumas opções antes de agir. Isto envolve dois processos:

- Exploração de opções: imaginar diferentes caminhos possíveis, com base nas ações disponíveis e no modelo do mundo;
- Avaliação de opções: analisar cada alternativa com base em critérios como:

    - Custo;
    - Valor (utilidade da solução).

Este raciocínio antecipado é o que torna os agentes deliberativos especialmente úteis em contextos onde não basta reagir, mas tambem tomar decisões estratégicas.

Planeamento Automático

O planeamento automático é um processo deliberativo que permite ao agente construir uma sequência de ações Este planeamento é feito com base em raciocínio automático, recorrendo a técnicas como a procura em espaço de estados ou modelos de decisão.

Quando o agente usa a procura em espaço de estados (PEE), o planeador trabalha sobre uma versão simplificada do mundo, onde cada estado representa uma configuração possível, e as ações do agente são vistas como operadores que transformam esses estados.

Processos de Decisão Sequencial

Em muitos problemas do mundo real, as consequências de uma decisão podem surgir mais tarde e nem sempre de forma garantida. Podemos referir assim os processos de decisão sequencial, onde a ação correta depende não só do estado atual, mas também da sequência de ações futuras que o agente venha a executar.

Num processo de decisão sequencial o objetivo do agente é encontrar uma estratégia (política) que maximize os ganhos acumulados ao longo da sua trajetória.

Passos relativos a este processo:

- O agente está num estado actual;

- Escolhe uma ação entre as várias disponíveis;
- Essa ação leva-o a um novo estado, com alguma probabilidade de sucesso ou de falha;
- Recebe uma recompensa, positiva ou negativa;
- E o ciclo repete-se ao longo do tempo.

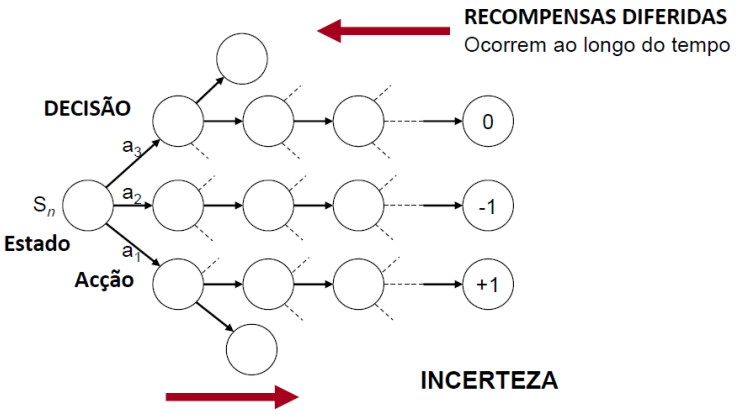


Figura 22 - Decisão Sequencial

(Slides do Professor Luís Morgado)

Lidar com Incerteza nas Decisões

Quando o agente não obtém todas as informações relativas ao domínio do problema enfrentamos incerteza.
Assim pode haver vários desfechos possíveis, cada um com uma certa probabilidade.

Este raciocínio torna-se fundamental onde sensores imperfeitos e ambientes dinâmicos tornam o comportamento do mundo menos previsível.

Propriedade de Markov

A maioria dos modelos usados nestes problemas baseia-se na chamada propriedade de Markov, criada por Andrey Markov (Matemático Russo), que afirma que o futuro depende apenas do estado atual, e não da sequência de estados anteriores.

Isto permite simplificar o raciocínio e usar modelos chamados Processos de Decisão de Markov (PDM), que irão ser referidos ainda neste capítulo.

Recompensa e Utilidade

A cada ação podem estar associadas recompensas que indicam se a ação foi boa ou má:

- Recompensas positivas = ganhos;
- Recompensas negativas = perdas.

A utilidade será o valor de um determinado estado para realizar o objetivo pretendido, pode ser medido pelo valor total esperado de seguir uma certa sequência de ações a partir de um estado.

Cálculo da Utilidade: $U(s_t) = R(s_t) + R(s_{t+1}) + R(s_{t+2}) + \ldots$

Uma maneira prática de entender o cálculo da utilidade será associar este a soma das recompensas ao longo da sequência de estados.

Política (Tomadas de Decisão)

A política é representada por uma função que indica qual ação tomar em cada estado.

Uma política poderá ser:

- **Determinista**: sempre executa a mesma ação, no mesmo estado;
- **Não determinista**: escolhe ações com certas probabilidades.

O objetivo será encontrar a política ótima, aquela que representa o maior valor esperado a longo prazo. Esta política é definida com base na função de utilidade de cada estado.

Processos de Decisão Markov (PDM)

Um PDM é um modelo que descreve as situações em que um agente tem de tomar decisões num ambiente incerto e ao longo do tempo. Baseia-se na ideia de que o próximo estado depende apenas do estado atual e da ação escolhida, e não do historial completo

O modelo inclui:

- Um conjunto de estados do mundo (S);
- Um conjunto de ações disponíveis num estado (A);
- Probabilidade de “s” para “s’” através de “a” T(s,a,s’);
- Recompensa esperada de “s” para “s’” através de “a” R(s,a,s’).
- Tempo discreto t = 0, 1, …

Planeamento com base em PDM

Planeamento com base em Processos de Decisão de Markov referem-se a quando aplicamos esta lógica ao planeamento. Neste caso, o agente utiliza um modelo do mundo probabilístico para prever os efeitos das suas ações e calcular a melhor sequência de decisões.

O planeador precisa dos mesmos elementos do modelo PDM, adicionando a gama "γ", que representa um fator de desconto.

O raciocínio assenta no cálculo da função de utilidade, que atribui um valor a cada estado do problema. Esse valor é depois usado como base para definir a política, ou seja, a estratégia que o agente deve seguir. Em particular, o objetivo é encontrar a "política ótima", que indica quais as ações que maximizam o valor esperado em cada estado.

Aprendizagem por Reforço

Neste último capítulo teórico iremos abordar o tema da aprendizagem por reforço, esta entende-se quando o agente não conhece o modelo do mundo nem os resultados exatos das suas ações, mas precisa de aprender com base na experiência. Esta forma de aprendizagem baseia-se na ideia de que o agente explora o ambiente, recebe recompensas ou penalizações, e ajusta o seu comportamento ao longo do tempo.

É, por natureza, uma aprendizagem de interação com o ambiente, onde o conhecimento é adquirido passo a passo com base nas consequências das ações tomadas.

Na aprendizagem por reforço, o agente:

- Observa o estado atual do ambiente;
- Escolhe uma ação;
- Executa essa ação e observa o resultado (novo estado);
- Recebe um reforço (recompensa) associada a essa transição;
- Atualiza o seu conhecimento e tenta melhorar as escolhas futuras.

Este processo repete-se continuamente, permitindo ao agente descobrir, por tentativa e erro, quais são as melhores decisões a tomar para maximizar as recompensas ao longo do tempo.

Projecto – Parte 1

Na primeira parte do projeto introduziu-se a construção de sistemas autónomos em Java. O principal objetivo foi modelar o comportamento de um agente reativo simples, inserido num ambiente simulado, utilizando máquinas de estados finitas como mecanismo de controlo. Esta fase focou-se em compreender os fundamentos da inteligência artificial aplicada a agentes, ao mesmo tempo que aplicámos boas práticas de Engenharia de Software.

A arquitetura do projeto foi dividida em vários subsistemas, organizados para refletir os principais elementos envolvidos num agente autónomo:

- **Agente** – Este pacote reúne as classes principais do agente inteligente, nomeadamente:
  - "Agente": implementa o ciclo perceção,processamento, ação;
  - "Controlo": uma interface que define como o agente decide o que fazer;
  - "Percepcao": encapsula a informação que o agente obtém do ambiente;
  - "Accao": representa as ações que o agente pode executar.

  Estas classes representam de forma abstrata o comportamento de qualquer agente autónomo.

- **Ambiente** – responsável por representar o mundo onde o agente atua, neste pacote são definidas as interfaces gerais do ambiente:
  - "Ambiente": descreve o contrato para qualquer tipo de ambiente;
  - "Comando": interface para ações que afetam o ambiente;
  - "Evento": interface para representar percepções que vêm do ambiente.

  Estas interfaces funcionam como **contratos genéricos**, permitindo desacoplar completamente a lógica do agente da implementação concreta do mundo onde este atua.

- **Jogo** – sistema que coordena a interação entre todos os elementos, este é o pacote onde se encontra a implementação específica do cenário de simulação. Divide-se em duas subpastas:

  "jogo.ambiente":

  - "AmbienteJogo": simula um ambiente interativo onde o utilizador gera eventos por teclado;
  - "ComandoJogo" e "EventoJogo": implementações concretas das interfaces Comando e Evento, usando enumerados para representar ações e eventos possíveis;

  "jogo.personagem":

  - "ControloPersonagem": implementa um controlo baseado numa máquina de estados;
  - "Personagem": representa a personagem do jogo e herda da classe Agente.

  Contém a classe main responsável por inicializar o ambiente e a jogo.personagem e executa a lógica do jogo.

- **Máquina de Estados** – estrutura lógica que define a dinâmica de comportamento da personagem:
  - "Estado": representa um estado possível da personagem;
  - "Transicao": descreve as transições entre estados, acionadas por eventos;
  - "MaquinaEstados": gere o estado atual e calcula a próxima ação com base nas transições definidas.

  Este módulo abstrai a lógica do controlo reativo através de uma estrutura versátil e reutilizável, alinhada também com os modelos de Mealy e Moore.

  Máquinas de Mealy: a função de saída depende das entradas.

  Máquinas de Moore: a função de saída não depende das entradas

O agente, representado pela classe "Agente", segue um ciclo relativo a percecionar o ambiente recebendo um "Evento", processa a percepção através do "Controlo", que decide qual ação tomar e atua sobre o ambiente com a ação retornada.

Toda esta lógica está desacoplada, permitindo que o mesmo agente funcione com diferentes tipos de controlo, bastando mudar a instância de Controlo.

O comportamento da personagem foi implementado através da classe “ControloPersonagem”, que utiliza uma máquina de estados para decidir que ação executar consoante o evento detetado no ambiente. Esta máquina de estados é composta por quatro estados principais: Procura, Inspeção, Observação e Registo. Cada um destes representa um modo específico de atuação da personagem, refletindo o tipo de estímulo recebido.

Foram criados eventos de mudança da Maquina de Estados (Transição):

- Animal: Procurar/Inspeção, muda para a observação.
- Ruído: Procura, muda para a inspeção.
- Silêncio: Inspeção, muda para a procura.
- Fuga: muda para a inspeção/procura.
- Fotografia: muda para procura.

A lógica de controlo segue o modelo de máquina de Moore, onde a resposta do agente é determinada pelo estado atual e pelo evento que desencadeia a transição. Esta abordagem permite criar comportamentos coerentes e previsíveis, mantendo a simplicidade estrutural e a clareza de execução.

A classe “AmbienteJogo” representa o mundo simulado onde o agente atua. Este ambiente permite ao utilizador interagir com o sistema de forma direta, introduzindo eventos através do teclado, como por exemplo a tecla “s” para SILENCIO, “r” para RUIDO, “a” para ANIMAL, “f” para FUGA, “o” para FOTOGRAFIA e “t” para TERMINAR.

Estes eventos são armazenados e disponibilizados ao agente sobre a forma de percepções, o que permite testar o comportamento reativo da personagem em tempo real. Apesar da simplicidade da interface, esta implementação revelou-se muito eficaz para validar a lógica de controlo e garantir que as reações do agente estavam de acordo com os eventos gerados.

```
Digite um evento: (s)SILENCIO, (r)RUIDO, (a)ANIMAL, (f)FUGA, (o)FOTOGRAFIA, (t)TERMINAR:
s
Evento: SILENCIO
Estado: Procura
Comando: Comando: PROCURAR
Digite um evento: (s)SILENCIO, (r)RUIDO, (a)ANIMAL, (f)FUGA, (o)FOTOGRAFIA, (t)TERMINAR:
a
Evento: ANIMAL
Estado: Observação
Comando: Comando: APROXIMAR
Digite um evento: (s)SILENCIO, (r)RUIDO, (a)ANIMAL, (f)FUGA, (o)FOTOGRAFIA, (t)TERMINAR:
o
Evento: FOTOGRAFIA
Estado: Observação
Digite um evento: (s)SILENCIO, (r)RUIDO, (a)ANIMAL, (f)FUGA, (o)FOTOGRAFIA, (t)TERMINAR:
t
Evento: TERMINAR
Estado: Observação

Process finished with exit code 0
```

Figura 23 – Consola relativa à Parte 1

(Página intencionalmente deixada em branco)

Projecto – Parte 2

Nesta segunda fase do projeto, deixámos para trás a implementação em Java e avançámos para o desenvolvimento tudo na linguagem de programação Python, com foco total num agente reativo. O objetivo foi aplicar, de forma prática, a nova aprendizagem de todos os conhecimentos em Engenharia de Software, relativa a esta linguagem, os conceitos de arquitetura reativa, em que o agente toma decisões com base direta nas percepções do ambiente, sem recurso a planeamento.

Foi também nesta fase que começámos a utilizar o simulador fornecido pelo docente, o package "sae", com o ambiente e agente definidos em estruturas reutilizáveis, e que permitiram evoluir progressivamente para um comportamento mais complexo.

O objetivo desta parte foi criar um agente funcional que respondesse automaticamente a estímulos detectados, reagindo de forma simples e eficaz, como previsto no modelo clássico de reatividade.
A arquitetura seguida foi inspirada nos esquemas teóricos apresentados nas aulas, estes definem:

- Um ciclo de controlo baseado em percepção/processamento/ação;
- Um módulo de controlo responsável por ativar comportamentos consoante os estímulos;
- Um conjunto de reacções organizadas em comportamentos simples ou compostos.

O centro da decisão foi implementado na classe "controlo_react", que liga as percepções do ambiente ao comportamento selecionado. Esta estrutura permite ao agente reagir de forma automática e dinâmica no ambiente, garantindo uma resposta rápida para estímulos.

Os comportamentos foram definidos usando uma abordagem modular. Por exemplo:

- A classe "Explorar" move-se com movimentos aleatórios sem necessitar de um estímulos (explorar);
- A classe "Recolher" (AproximarAlvo > EvitarObst > Explorar) utilizada no agente final, combina diferentes reações e comportamentos compostos com propriedades (recolher).

A classe "AgenteReact" define o agente e o seu ciclo de execução (agente_react):

1. Percebe o que se passa à sua volta (percepcionar);

2. Processa essa percepção através do controlo (ControloReact);
3. Executa a ação resultante (actuar).

Para organizar melhor as diferentes reações, foram utilizados comportamentos compostos com hierarquia, como definido na classe "Recolher". Este comportamento inclui:

- AproximarAlvo: aproxima-se de um alvo detectado;
- EvitarObst: evita obstáculos nas várias direções;
- ExplorarMem: evita zonas já visitadas;
- Explorar: realiza movimentos aleatórios quando nada mais se aplicar.

Esta estrutura hierárquica permite ao agente priorizar ações com base na sua urgência ou relevância.

Iremos ter dois tipos de reações:

- **Estímulo**: deteta algo numa direção (ex.: "EstimuloAlvo" ou "EstimuloObst");
- **Resposta**: executa uma ação adequada (ex.: "RespostaMover", "RespostaEvitar")

Estas reações foram combinadas com diferentes estratégias de ativação (prioridade, hierarquia, memória, a memória irá ser referida no próximo ponto desta parte 2.

Como já referido foi criado um "Explorar", que usava movimentos aleatórios, posteriormente foi introduzido um "ExplorarMem", que guarda percepções passadas para evitar repetir caminhos. Isto permite uma melhoria de desempenho sem deixar de ser reativo, já que o agente continua a reagir apenas ao que vê, mas agora com alguma noção do que já viu.

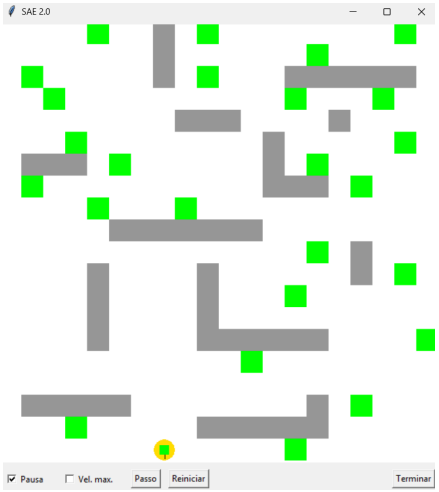


Figura 24 - Simulador relativo à Parte 2

Projecto – Parte 3

Na terceira fase do projeto, aprimoramos o comportamento do agente, deixando para trás a reatividade pura para começar a trabalhar com deliberação e planeamento automático.
O foco foi a introdução de um mecanismo de Procura em Espaço de Estados (PEE), que permitisse ao agente raciocinar sobre o ambiente e planear ações antes de agir.

Nesta fase, começámos verdadeiramente a construir um agente com capacidade de analisar a situação, definir objetivos e procurar a melhor forma de os atingir.

Até aqui, o agente reagia diretamente aos estímulos. Agora, com o planeamento por procura, o agente passou a:

- Construir um modelo interno do mundo;
- Deliberar com base em objetivos;
- Escolher o melhor caminho para lá chegar.

Este processo é feito através da classe “PlaneadorPEE”.

Para aplicar a procura, foi necessário primeiro modelar o problema. Isso foi feito com a classe “ProblemaPlan”, que define o estado inicial, estados objetivos e operadores disponíveis.
Para guiar a procura, foi utilizada a heurística “HeurDist”, que calcula a distância euclidiana entre estados.

Testámos várias estratégias de procura para comparar o desempenho:

- Procura de Custo Uniforme: garante sempre a solução com menor custo;
- Procura A*: combina custo acumulado com heurística, permitindo soluções mais rápidas mantendo a qualidade dos planos.

O plano gerado pelo agente foi representado na classe “PlanoPEE”, que transforma a sequência de estados/ações numa estrutura passo a passo.

Esta abordagem garante que o agente se adapta caso haja mudança de mundo, mantendo o comportamento consistente com a realidade.

O agente foi refeito na classe “AgenteDelib”, passando a usar o controlo deliberativo. Este agente é agora capaz de:

- Atualizar o seu modelo do mundo com base nas percepções;

- Deliberar com base em alvos visiveis;
- Avaliar se precisa de gerar um novo plano;
- Planear a melhor sequência de ações para o objetivo;
- Executar o plano passo a passo até atingir o objetivo ou precisar de reconsiderar.

Para validar o funcionamento dos mecanismos de procura, foi criado o ficheiro “teste_contagem.py”, onde implementámos vários testes com a classe “ProblemaContagem”.

Estes testes garantiram-nos a boa continuidade do projeto,a partir da confirmação do sistema de procura e geração de planos (custo, número de nós processados, máximo de nós armazenados em memória e número de estados não repetidos).

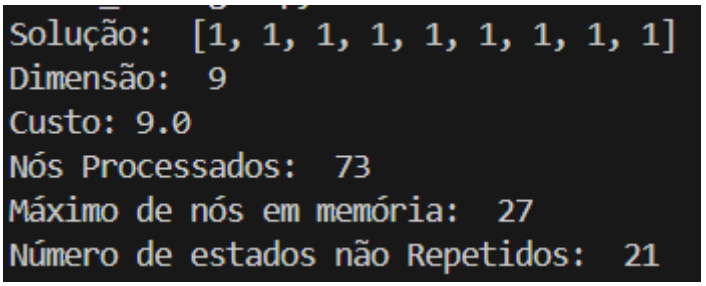


Figura 25 – Teste Procura Custo Uniforme

Para testar o comportamento deliberativo do agente no ambiente simulado foi elaborado um segundo teste.
Foram utilizados testes “assert” para garantir que:

- O modelo do mundo “ModeloMundo” se atualiza corretamente com base nas percepções;
- Os estados válidos e operadores são detectados com precisão;
- O mecanismo deliberativo “MecDelib” identifica corretamente os alvos como objetivos;
- O planeador “PlaneadorPEE” gera um plano com a dimensão esperada;
- O controlo deliberativo “ControloDelib” reage corretamente à percepção e seleciona a ação adequada.

Este teste valida a integração entre todos os módulos do agente deliberativo, caso errados retornem uma exceção na consola.

```
def teste_modelo_mundo(percepcao):

    modelo_mundo = ModeloMundo() # Criar instancia do Modelo do
Mundo
```

```
    modelo_mundo.actualizar(percepcao) # Atualizar modelo do mundo

    assert modelo_mundo.alterado == True # Verifica se o modelo
reconheceu alterações após a perceção



    estado = modelo_mundo.obter_estado()

    assert estado.posicao == (0, 0) # Verifica se a posição do
agente foi corretamente registada no modelo



    estados = modelo_mundo.obter_estados()

    #print(len(estados))

    assert len(estados) == 671 # Confirmar que o modelo detetou
todas as posições válidas do ambiente

    # ("671" depende do ambiente definido, neste caso o Ambiente
Numero 4)



    operadores = modelo_mundo.obter_operadores()

    assert len(operadores) == 4 # Confirmar que foram criados
operadores para as 4 direções possíveis



    estado = EstadoAgente((28, 9))

    elemento = modelo_mundo.obter_elemento(estado)

    assert elemento == Elemento.ALVO # Verifica se o modelo sabe
que na posição escolhida existe um alvo



    return modelo_mundo



def teste_mec_delib(modelo_mundo):
```

```
    mec_delib = MecDelib(modelo_mundo)


    objectivos = mec_delib.deliberar()
    assert len(objectivos) == 3 # Verifica se o mecanismo foi capaz
de identificar 3 alvos como objetivos válidos


    return mec_delib, objectivos


def teste_planeador_pee(modelo_mundo, objectivos):
    planeador = PlaneadorPEE()
    plano = planeador.planear(modelo_mundo, objectivos)
    assert plano.dimensao == 37


    return planeador


def teste_controlo_delib(planeador, percepcao):
    controlo = ControloDelib(planeador)


    accao = controlo.processar(percepcao)
    assert accao.direccao == Direccao.SUL


# Activação do Teste
if __name__ == "__main__": # "__name__" nome do modulo interno
    num_amb = 4
    ambiente = Ambiente(DEF_AMB[num_amb]) # Instancia ambiente
```

```
    # Para obter percepcao precisamos de uma instancia de
Transdutor

    transdutor = Transdutor()

    transdutor.iniciar(ambiente)

    percepcao = transdutor.percepcionar()



    # Teste do modelo do mundo

    modelo_mundo = teste_modelo_mundo(percepcao)



    # Teste do mecanismo deliberativo

    mec_delib, objectivos = teste_mec_delib(modelo_mundo)



    #Teste do planeador

    planeador = teste_planeador_pee(modelo_mundo, objectivos)



    # Teste do Controlo Deliberativo

    teste_controlo_delib(planeador, percepcao)



    print("\nTeste concluido com sucesso\n")



    Simulador = Simulador(4, AgenteDelib(), vista_modelo=True)

    Simulador.executar()
```

Figura 26 – Testes Assert

Visualmente, no simulador, foi possível observar o agente a identificar os alvos, a tracejado amarelo mostrar os melhores trajetos e recalcular planos quando o ambiente mudava.

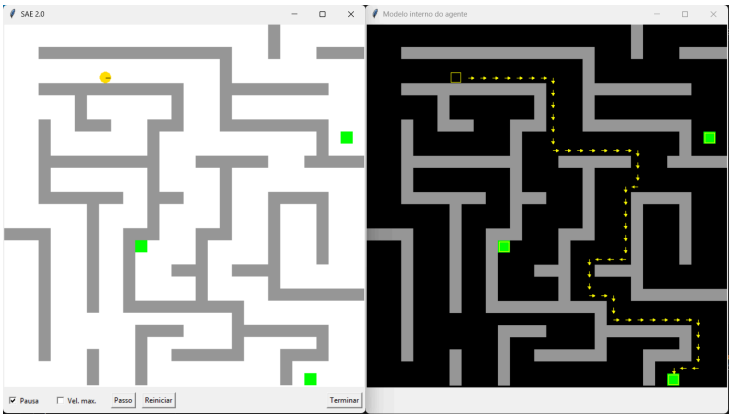


Figura 27 – Simulador Parte 3 (1)

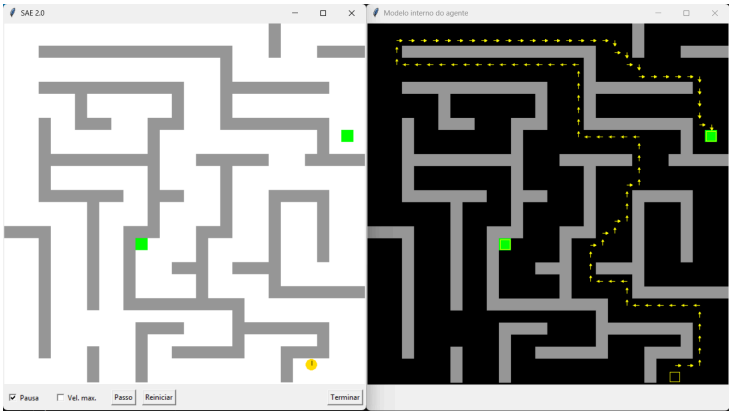


Figura 28 – Simulador Parte 3 (2)

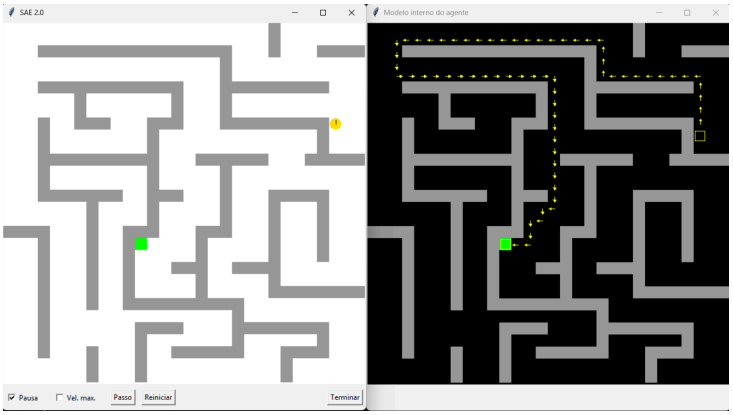


Figura 29 – Simulador Parte 3 (3)

Projecto – Parte 4

Na quarta e última fase do projeto de IASA, o agente deliberativo foi evoluído para um estado final, desta vez com a introdução de um modelo de decisão mais sufisticado (Processos de Decisão de Markov (PDM)). Esta abordagem permitiu que o agente deixasse de apenas seguir planos fixos e passasse a tomar decisões com base na utilidade esperada ao longo do tempo, mesmo em situações com incerteza e consequências a um certo prazo.

Nesta fase, substituímos o planeador baseado em procura (PEE) por um novo planeador "PlaneadorPDM", que tem por base um modelo de política ótima. Em vez de procurar uma sequência de ações até um objetivo, o agente agora:

- Avalia qual o estado mais vantajoso a seguir;
- Toma decisões com base em recompensas acumuladas;
- Ajusta o seu comportamento de forma mais estratégica.

Este planeador usa um modelo do mundo específico "ModeloPDMPlan" e um mecanismo de decisão (Processos de Decisão Markov) que calcula para cada estado a melhor ação a tomar de acordo com a política ótima.

Começamos por desenvolver um Modelo Exemplo (Ambiente 7x1) para testarmos o novo Modelo do Mundo com os processos de decisão de markov e entendermos melhor o funcionamento deste novo planeador.

Para permitir esta abordagem, foi necessário criar uma ponte entre o modelo do mundo do agente e o formato exigido pelo planeador PDM. Isso foi feito através da classe "ModeloPDMPlan", que implementa as funções:

- S(): Lista de Estados;
- A(): Lista de Operadores (acções);
- T(s, a, s'): Modelo de Transição;
- R(s, a, s'): Modelo de Recompensa;
- suc(s, a): Sucessores de um estado.

Neste modelo, as ações que conduzem a estados objetivo recebem recompensas máximas (rmax), enquanto ações em estados que já atingiram o objetivo são inibidas para evitar repetições desnecessárias.

O cálculo das decisões ideais passou a ser feito pelo "PDM", que organiza iteração de valor através do módulo "MecUtil". Esta iteração permite

atualizar, de forma progressiva, a utilidade de cada estado, tendo em conta os modelos de transição (T), e das recompensas atribuídas (R) e o fator de desconto (gama). Com base nas utilidades calculadas, o sistema extrai no final uma política ótima (π*), que define qual é a melhor ação a tomar em cada estado.

Para aplicar esta nova forma de planear, foi desenvolvido um novo agente deliberativo "AgenteDelibPDM" para uso no simulador. Apesar de manter a estrutura geral do agente deliberativo da fase anterior, este novo agente deixa de depender de uma sequência fixa de passos, agora segue uma política dinâmica e adapta-se às mudanças no ambiente e maximizando sempre a utilidade esperada a longo prazo.

Durante a simulação com o ambiente SAE, foi possível observar com clareza os efeitos desta nova abordagem.

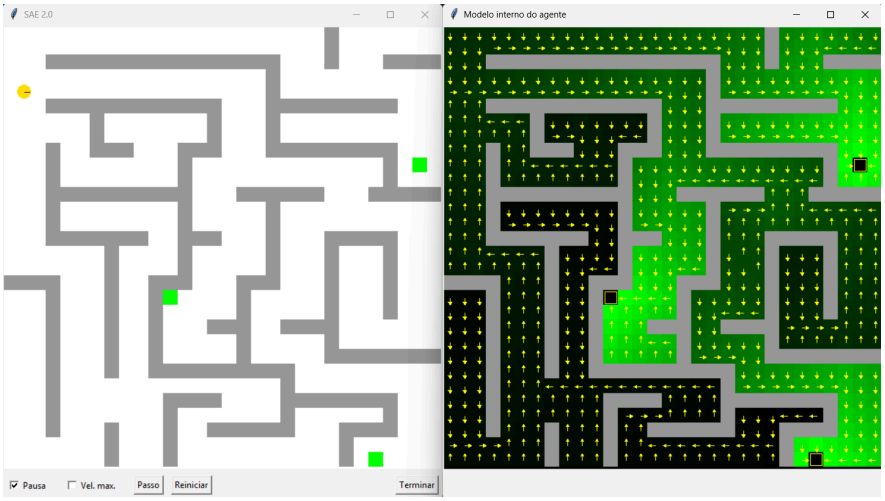


Figura 30 – Simulador Parte 4 (1)

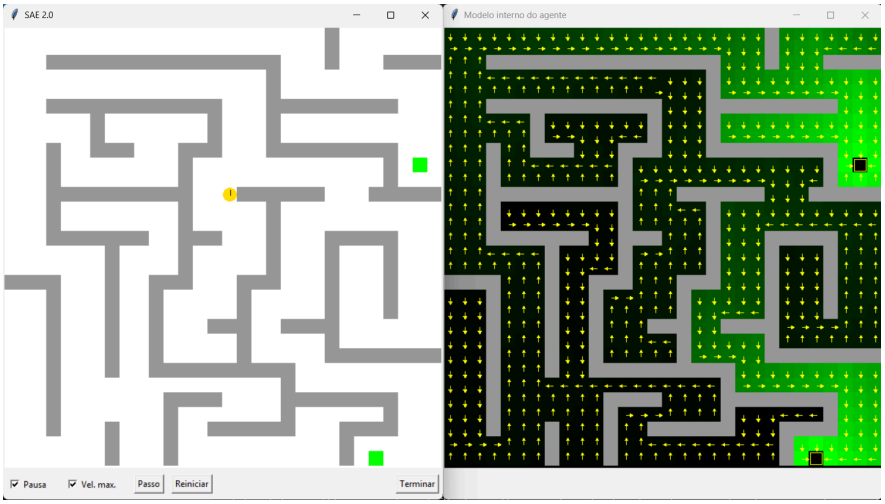


Figura 31 – Simulador Parte 4 (2)

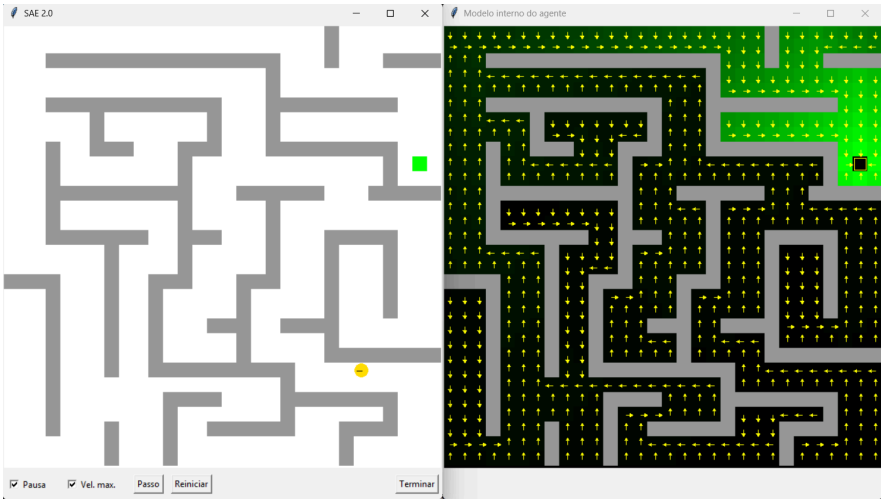


Figura 32 – Simulador Parte 4 (3)

A requisito do professor Luís Morgado, foi pedido para implementarmos uma nova estratégia, ou seja, em vez do agente encontrar os alvos mais próximos, encontrar prioritariamente sempre o mais afastado.

Para isso alteramos apenas duas linhas de código:

- No “ControloDelib”: Acrescentou-se [0] no “self.__objectivos” para que apenas seja usado o primeiro objetivo da lista para planear.
- No “MecDelib”: Adicionamos o “reverse = True” para inverter a ordem criada pelo parâmetro “distância”.

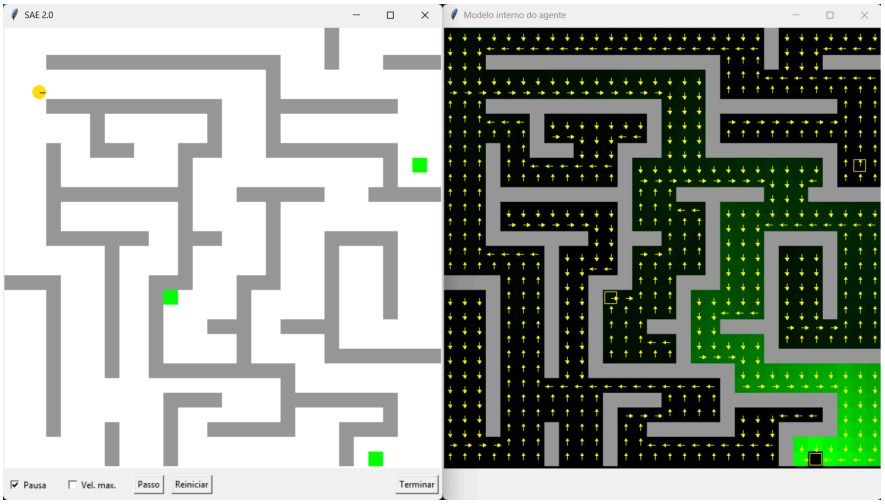


Figura 33 – Simulador Parte 4 – Extra (1)

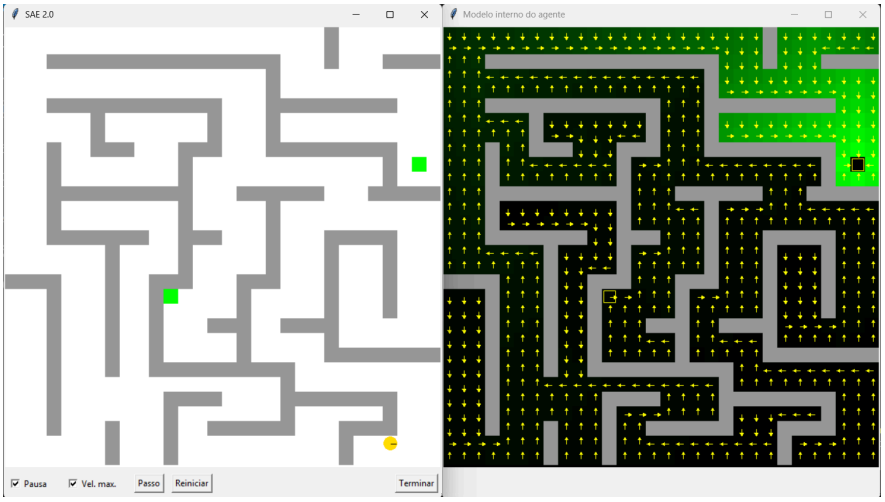


Figura 34 – Simulador Parte 4 – Extra (2)

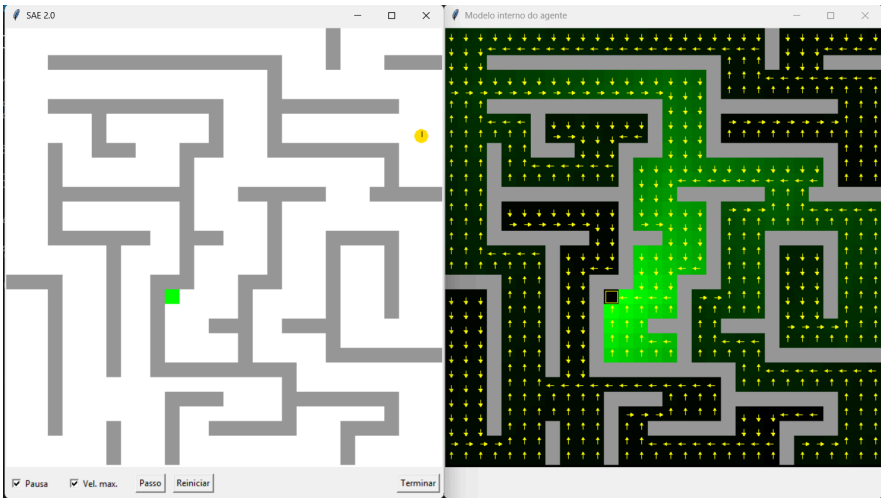


Figura 35 – Simulador Parte 4 – Extra (3)

Revisão do projecto realizado

Ao longo destas quatro fases do projeto da unidade curricular de Inteligência Artificial para Sistemas Autónomos (IASA), foi possível acompanhar uma evolução progressiva e estruturada do comportamento do agente.

Começámos com uma implementação simples em Java, focada em agentes puramente reativos e baseados em máquinas de estados. Esta primeira abordagem permitiu aprender, de forma prática, os fundamentos da reatividade.

Na segunda fase, a transição para Python marcou o início de um novo ciclo, com o reforço da arquitetura reativa e boas práticas de Engenharia de Software. Foram explorados conceitos como estímulos, respostas, comportamentos compostos e seleção de ações com hierarquia. O agente começou a adaptar-se melhor, reagindo a obstáculos, alvos e zonas exploradas.

Com a terceira fase, introduzimos planeamento deliberativo com base em procura em espaço de estados (PEE). O agente passou a ser capaz de raciocinar antes de agir, usando heurísticas e estratégias como a Porcura A* ou Procura Custo Uniforme para encontrar trajetos ótimos até aos seus objetivos. Esta camada deliberativa representou um salto significativo em termos de autonomia e inteligência do sistema.

Por fim, na quarta parte, foi incorporado um modelo ainda mais avançado (os Processos de Decisão de Markov PDM) que permitiram ao agente tomar decisões sob incerteza, considerando recompensas acumuladas ao longo do tempo. Com isso, o agente parou de executar planos fixos e passou a seguir uma política ótima que maximiza utilidade, tornando-se capaz de antecipar consequências e adaptar-se a cenários mais complexos.

(Página intencionalmente deixada em branco)

Conclusão

O projeto de IASA permitiu consolidar, de forma prática e progressiva, os principais conceitos ligados à inteligência artificial aplicada a agentes autônomos. Desde a reatividade básica até ao planeamento com raciocínio sequencial, cada fase do desenvolvimento serviu de aprendizagem para a construção de um sistema inteligente que observa, decide, planeia e aprende.

Para além da aplicação dos conteúdos teóricos, o projeto também reforçou boas práticas de engenharia de software (algo muito referido durante toda esta cadeira), com uma arquitetura modular, reutilizável e fácil de se expandir. A utilização de comentários orientados para a teoria, a realização de testes e a observação do comportamento do agente em simulação foram aspetos fundamentais para validar as escolhas feitas e garantir a consistência das implementações e aprendizagem.

Referências

- Slides teóricos do Professor Luís Morgado

  Material disponibilizado na plataforma moodle da cadeira respetiva ao projeto, Instituto Superior de Engenharia de Lisboa (ISEL).

- Slides teóricos do Dr. Gabor Hullam

  Material disponibilizado na plataforma Microsoft Teams da cadeira que ingressei na faculdade da Hungria, Budapest University of Technology and Economics (BME).

- Slides teóricos do Professor Lucas Cambuim:

  Material encontrado na internet para melhor compreensão de certos assuntos e disponibilização de mais imagens ilustrativas relevantes para o relatorio, acesso: https://www.cin.ufpe.br/~lfsc/cursos/introducaoainteligenciaartificial/IA-Aula4-BuscaCega.pdf.