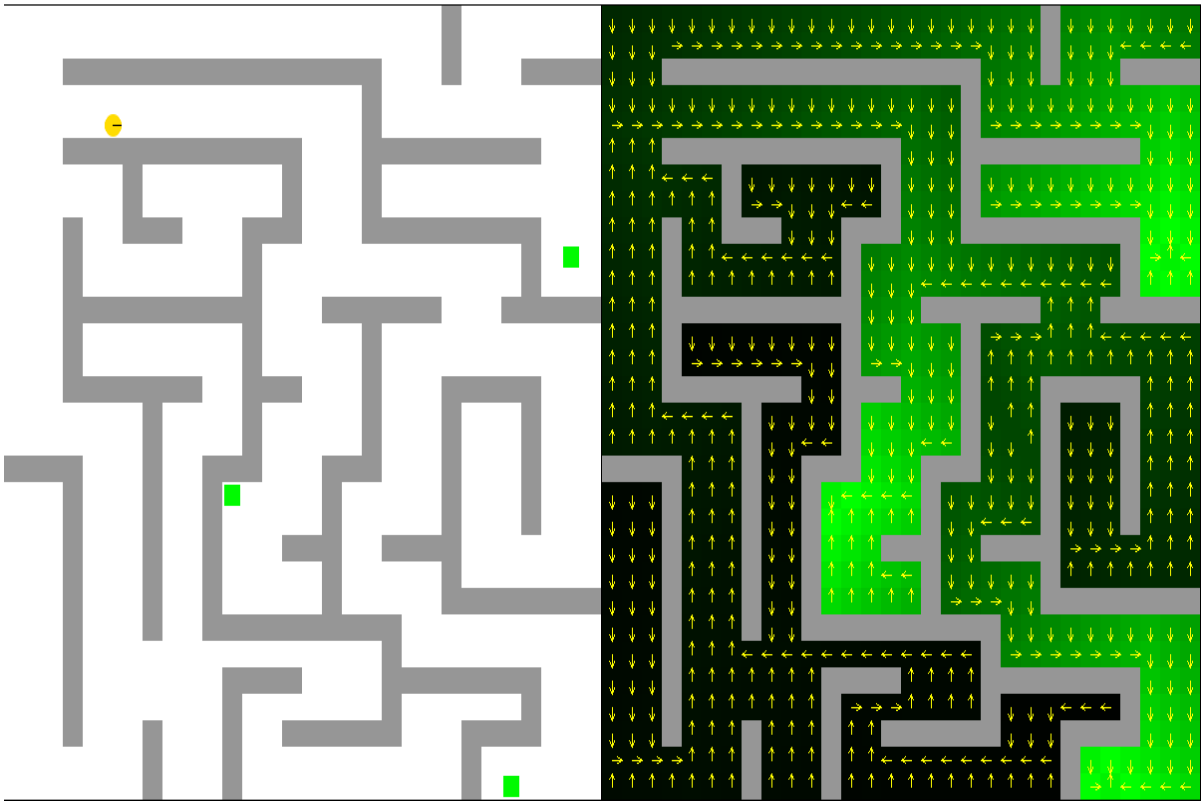
Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Inteligência Artificial para Sistemas Autónomos - 2022/2023 SV

Relatório Final



Docente Luís Morgado

Realizado por :
Pedro Silva 48965

3 de julho de 2023

Conteúdo

Lista de Figuras

1 Introdução

Este relatório tem como objetivo relatar o desenvolvimento do projeto realizado ao longo do semestre na disciplina de IASA. Esta disciplina e consequentemente, este projeto servem como uma excelente introdução e consolidação de conhecimentos de duas áreas da engenharia informática: Inteligência Artificial e Engenharia de Software. Assim, o semestre e o projeto ficaram divididos em 7 partes sendo estas:

1. INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL
2. INTRODUÇÃO À ENGENHARIA DE SOFTWARE
3. ARQUITETURA DE AGENTES INTELIGENTES
4. ARQUITETURA DE AGENTES REATIVOS
5. RACIOCÍNIO AUTOMÁTICO E TOMADA DE DECISÃO
6. ARQUITETURA DE AGENTES DELIBERATIVOS
7. APRENDIZAGEM POR REFORÇO

Antes de começarmos foi nos introduzido a linguagem UML que é uma linguagem visual padronizada que permite a modelagem e documentação de sistemas de software complexos. Os UMLs do projeto foram nos disponibilizados ao decorrer do semestre com o objetivo de analisar, projetar e comunicar sistemas de software de forma mais eficiente, reduzindo ambiguidades e erros de interpretação. Com estes e a ajuda do docente ficou montada a dinâmica das aulas, onde nos era dado modularmente trechos de matéria e um UML e nós tínhamos que passar para código essa informação.

As primeiras duas partes servem de introdução aos conceitos e temas que vamos abordar ao longo da disciplina, para isso foi nos pedido que criássemos um jogo, com a linguagem JAVA, que consiste num ambiente onde a personagem tem por objetivo registar a presença de animais através de fotografias. Esta interação dá-se através de uma máquina de estados o que nos ensinou sobre Estados/Transições e Eventos/Ações.

De seguida, aprendemos sobre a arquitetura de agentes inteligentes o que funcionou como uma ponte dos temas introdutórios para o sumo da disciplina. Estes agentes têm diversas características que dependem do tipo de modelo da sua arquitetura, sendo estes o modelo reativo, modelo deliberativo e modelo híbrido.

Entramos agora no foco desta cadeira que são os agentes reativos, o raciocínio automático e os agentes deliberativos. Passamos agora para a linguagem ”Python”e para o contexto que vai ficar connosco até ao final do semestre. Este vai ser uma simulação de um agente autónomo de recolha de alvos e desvio de obstáculos ou colisões. A diferença vai estar na implementação dos mecanismos de processamento onde estes vão envolver os tópicos referidos no início do parágrafo.

Por fim, foi nos apresentado e introduzido a aprendizagem por reforço. Esta é uma abordagem poderosa que permite que os agentes aprendam a tomar decisões autónomas através de interações com o ambiente, visando maximizar as recompensas obtidas.

2 Enquadramento Teórico

2.1 Introdução à Inteligência Artificial

A inteligência artificial é o campo que estuda a síntese e análise de agentes computacionais que agem de forma inteligente, que procura não só perceber, mas também construir entidades inteligentes. Pode ser dividida pelos seus principais paradigmas: simbólico, conexionista e comportamental.

Na inteligência artificial simbólica, a inteligência resulta da ação de processos computacionais sobre estruturas simbólicas, sem considerar os mecanismos responsáveis por esse processo, ou seja, permite capturar o conhecimento e o domínio de um problema de forma estruturada e compreensível. No entanto, o paradigma simbólico também apresenta desafios. A representação simbólica pode ser limitada na forma como lida com a incerteza e a imprecisão presentes em muitos problemas do mundo real. Além disso, a manipulação simbólica pode ser computacionalmente intensiva, especialmente em problemas complexos, o que pode limitar a escalabilidade e o desempenho dos sistemas.

Na inteligência artificial conexionista, a inteligência é uma propriedade emergente das interações de um número elevado de unidades elementares de processamento, e acredita-se que ao construir uma máquina que imite a estrutura do cérebro humano, esta apresentará inteligência. Este paradigma oferece vantagens como aprendizagem a partir de exemplos e tolerância a falhas, mas também enfrenta desafios relacionados com a quantidade de dados necessários e a interpretabilidade das redes neurais.

Por fim, na inteligência artificial comportamental, a inteligência resulta da definição de regras ou de associações estímulo-resposta, sem a necessidade de modelos internos complexos ou de representações simbólicas. O agente aprende a agir corretamente com base nas respostas que recebe do ambiente. Uma das limitações é a falta de explicabilidade do comportamento do agente. Como não há uma representação interna explícita do conhecimento ou do raciocínio, pode ser difícil entender o motivo pelo qual o agente tomou uma determinada ação.

2.2 Modelos de Dinâmica

O modelo de um sistema computacional é caracterizado por três perspetivas principais: estrutura, dinâmica e comportamento. A estrutura de um sistema é a sua organização no espaço, denota as partes e as suas relações num sistema, e apresenta estado (ou memória). A dinâmica de um sistema é a sua organização no tempo, são as regras que determinam como as partes e as suas relações evoluem ao longo do tempo, ou como as saídas evoluem a partir das entradas. O comportamento de um sistema é a transformação do estado com base em regras nas saídas, ou seja, a forma como o sistema reage a estímulos do ambiente.

A dinâmica pode ser expressa como uma função de transformação que, perante o estado atual e as entradas atuais, produz o estado seguinte e as saídas seguintes.

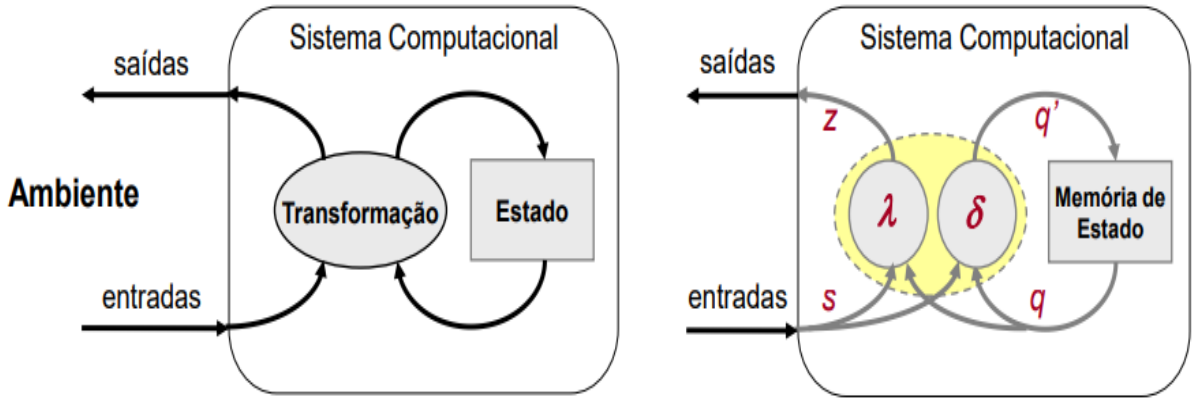


Figura 1: Representação da dinâmica de um sistema - Modelos de Dinâmica - Luís Morgado - página 4

Em termos de um sistema abstrato, as entradas e saídas são representadas por conjuntos de símbolos que podem ocorrer nelas. Esses conjuntos de símbolos são designados como alfabeto de entrada (representado por Σ) e alfabeto de saída (representado por Z).

Além disso, o sistema possui um conjunto de estados possíveis, representado por Q, que descreve o estado interno do sistema. A função de transformação do sistema é descrita com base em duas funções distintas: a função de transição de estado (representada por δ) e a função de saída (representada por λ).

A função de transição de estado, δ, mapeia um par composto por um estado atual do sistema (em Q) e um símbolo de entrada (em Σ) para um novo estado do sistema. Formalmente, δ: Q × $\Sigma \rightarrow$ Q.

A função de saída, λ, mapeia um par composto por um estado do sistema (em Q) e um símbolo de entrada (em Σ) para um símbolo de saída correspondente (em Z). Formalmente, λ: Q × $\Sigma \rightarrow$ Z.

Estas funções são usadas para descrever como o sistema se comporta quando recebe uma entrada específica e como gera uma saída correspondente com base no seu estado interno atual e no símbolo de entrada.

2.3 Engenharia De Software

A engenharia de software é uma área de engenharia orientada para a especificação, desenvolvimento e manutenção de software, que tem por objetivo o desenvolvimento, operação e manutenção de software de modo sistemático e quantificável. Sistemático no sentido de ser uma forma organizada e previsível de desenvolver software e quantificável ao usar o métodos e processos adequados para garantir a qualidade e o desempenho do software produzido, bem como a criação de documentação e a monitorização do processo de desenvolvimento.

Mas apresenta dois grandes problemas: a sua complexidade que tem tendência a aumentar, e a mudança que vai sofrendo. Quanto mais informação, mais complexo, e maior o esforço necessário para a geração de ordem nesse sistema. A

mudança é expressa no ritmo crescente a que o software necessita de ser produzido, ou modificado, para satisfazer as necessidades dos respetivos contextos de utilização.

Podemos medir em termos de acoplamento – grau de interdependência entre subsistemas (caráter inter-modular), coesão – nível de coerência funcional de um subsistema (caráter intra-modular), simplicidade – nível de facilidade de compreensão, e adaptabilidade – nível de facilidade da alteração da arquitetura para incorporação ou alteração de requisitos, por outras palavras, a facilidade com que evolui.

Um bom sistema é aquele que maximiza fatores como a segurança, robustez e facilidade de manutenção, e está estruturado de forma lógica com as suas funções devidamente separadas em módulos independentes, mantendo um baixo nível de acoplamento e alta coesão. A sua arquitetura deve seguir os princípios de:

- Modularidade – decomposição em partes coesas, com detalhes internos de cada parte isolados em relação ao exterior, reduzindo interdependência;
- Factorização – redução de redundâncias;
- Abstração – realçar apenas o essencial e omitir detalhes não necessários.

Ao longo do semestre foi desenvolvida a capacidade de desenvolver software robusto e bem documentado. Este processo deve ser iterativo, para obter o melhor balanço entre a modularidade de um projeto e a sua complexidade, e saber até onde decompor uma funcionalidade.

2.4 Agente Inteligente

Um agente inteligente perceciona o seu ambiente, processa e gera uma ação como resposta. Apresenta autonomia, reatividade, proatividade e sociabilidade, em prol da sua finalidade. Escolhe a ação que maximiza o valor esperado da medida de desempenho, dado o conhecimento disponível sobre o ambiente, perceções e ações. Estes agentes têm diversas características que dependem do tipo de modelo da sua arquitetura, sendo estes o modelo reativo, modelo deliberativo e modelo híbrido. No modelo reativo o comportamento do sistema é gerado de forma reativa, com base em associações entre estímulos (referentes às perceções) e respostas (referentes às ações). No modelo deliberativo o comportamento do sistema é gerado com base em mecanismos de deliberação (raciocínio e tomada de decisão), utilizando representações internas que incluem a representação explícita de objetivos. No modelo híbrido temos uma junção saudável dos dois em cima.

2.5 Agentes Reativos

Num agente reativo, o seu comportamento depende apenas da reação a estímulos no presente. As suas ações são diretamente ativadas em função das perceções, estando de acordo com o Paradigma Comportamental. Não simulam internamente estados futuros, não calculam os benefícios de uma determinada ação, nem têm uma representação interna do seu ambiente. Ganham o nome por agirem consoante o estado em que se encontram, sem ter em conta consequências a médio ou longo prazo, só reagem. Assim, se o ambiente sofrer alguma alteração, as respostas são rápidas e espontâneas. A sua natureza reativa implica que o agente tenha um conjunto de reações fixas e predefinidas para o seu contexto operacional.

No Paradigma Comportamental, um comportamento existe como forma de concretizar um objetivo e pode ser composto por um conjunto de outros comportamentos. Relaciona padrões de perceção com padrões de ação, mas o que interessa é a ação em si. O comportamento, internamente, pode ter reações a vários estímulos, pelo que encapsulamos o que está lá dentro seguindo o conceito de caixa preta, sabendo que funciona, mas sem saber exatamente como.

Uma perceção é capaz de ativar várias reações, disparando múltiplas ações, por isso é necessário um processo de seleção de ação. O sistema tem de ser capaz de escolher, entre um conjunto de ações, qual executar, utilizando um critério de racionalidade para escolher a ação de maior sucesso. Esta seleção pode ser feita de três formas: hierarquia, organizando os comportamentos numa hierarquia de supressão; prioridade, selecionando a resposta de acordo com a prioridade associada, podendo ser fixa ou dinâmica; ou fusão, combinando as respostas numa composição.

Estas formas de seleção de ação permitem ao sistema comportamental adaptar o seu comportamento de acordo com o ambiente e as circunstâncias, escolhendo as ações mais adequadas para alcançar o objetivo desejado. Dessa forma, o sistema pode lidar com situações complexas e dinâmicas, selecionando a ação mais apropriada com base em critérios de eficiência, relevância ou outros fatores relevantes.

2.6 Procura em Espaço de Estados

O raciocínio automático desempenha um papel fundamental no desenvolvimento de sistemas inteligentes capazes de resolver problemas complexos. Ele orienta-nos na representação e processamento de um problema para gerar soluções, através do que é conhecido como modelo do problema. Este modelo é composto por estados que descrevem a identificação única do problema e por operadores que representam ações ou transformações de estado.

O conjunto de estados e as transições entre eles formam o espaço de estados. O processo de procura nesse espaço é realizado de forma sucessiva, iniciando pelo estado inicial, também conhecido como raiz. A partir desse estado, são gerados nós aplicando os operadores, formando uma árvore de procura. Esses nós são organizados numa lista de possíveis nós seguintes, criando assim a fronteira de exploração que determina a estratégia adotada na procura.

Existem diferentes estratégias de exploração, que variam de acordo com a abordagem utilizada. Por exemplo, é possível explorar primeiro os nós mais recentes, utilizando uma fronteira LIFO (Last In, First Out), mantendo em memória o ramo e os estados seguintes, o que caracteriza a procura em profundidade. Por outro lado, pode-se optar por explorar os nós mais antigos, utilizando uma fronteira FIFO (First In, First Out), mantendo tudo em memória, caracterizando assim a procura em largura.

Os métodos de procura também variam em relação ao seu nível de conhecimento do domínio, o que influencia a ordenação da fronteira de exploração. Uma procura não informada é exaustiva e não utiliza critérios específicos para selecionar o espaço de estados, não tirando partido do conhecimento disponível. Já uma procura informada é guiada e seletiva, escolhendo o caminho mais benéfico com base nas informações do domínio.

2.6.1 Procura em Profundidade

A procura em profundidade tem como critério de exploração a maior profundidade, o que implica que a pesquisa é feita desde a raiz até ao mais longe possível em cada ramo, antes de retroceder.

Existem duas variantes da procura em profundidade. A primeira é a procura em profundidade limitada, onde a exploração é restrita a uma profundidade máxima pré-definida. Isso significa que, ao atingir o limite de profundidade, a procura retorna e explora outros ramos. Esta abordagem é útil para evitar a exploração infinita de caminhos longos ou quando se tem conhecimento prévio de que a solução desejada não está além de uma determinada profundidade.

A segunda variante é a procura em profundidade iterativa, que aplica a procura em profundidade limitada de forma iterativa, aumentando gradualmente o limite de profundidade a cada iteração. Esta abordagem permite explorar o espaço de estados em diferentes níveis de profundidade, começando com limites mais baixos e aumentando-os progressivamente. Desta forma, é possível encontrar soluções mais profundas a cada iteração e identificar casos em que a solução esteja além da profundidade atualmente explorada.

2.6.2 Procura em Largura

A pesquisa em largura tem como objetivo explorar todos os nós vizinhos inexplorados e, em seguida, explorar os nós vizinhos desses nós e assim por diante, até que o nó objetivo seja encontrado. É uma pesquisa não informada, o que significa que não usa informações específicas sobre o problema para orientar a exploração.

Ao contrário da procura em profundidade, a procura em largura não segue um critério de profundidade, mas sim uma ordem de exploração por níveis. Começando pelo nó inicial, a procura expande-se para os nós vizinhos desse nó, explorando-os completamente antes de passar para os vizinhos dos vizinhos e assim por diante. Esta estratégia garante que todos os nós sejam visitados no mínimo uma vez, permitindo uma procura exaustiva.

Uma vantagem da pesquisa em largura é que ela encontra a solução mais próxima do nó inicial, uma vez que explora de forma sistemática todos os caminhos em camadas. No entanto, esta abordagem pode requerer um elevado consumo de recursos computacionais, uma vez que todos os nós visitados precisam ser armazenados em memória durante o processo de busca.

2.6.3 Procura em Grafos

Nesse caso, ocorre um desperdício de tempo e memória. Para evitar isso, a pesquisa em grafos implementa um mecanismo para lidar com essa situação. Quando um nó sucessor é gerado, verifica-se se já está presente na lista de nós abertos (fronteira de exploração) ou na lista de nós fechados (já explorados). Se o nó sucessor não estiver nos nós abertos, é adicionado a essa lista. Se o nó sucessor já estiver nos nós abertos, mas tiver sido alcançado por um caminho mais curto, é atualizado na lista de nós abertos, mantendo apenas o caminho com o menor custo. Por fim, se o nó sucessor já estiver nos nós fechados, mas tiver um custo menor, é removido dos nós fechados e adicionado aos nós abertos. Desta forma, evita-se a repetição desnecessária da análise de estados e garante-se a eficiência do algoritmo de busca.

2.6.4 Procura Melhor-Primeiro

A procura melhor-primeiro é um tipo de pesquisa que utiliza uma função de avaliação para determinar a ordem de exploração dos nós em um espaço de busca. Esta função de avaliação estima o custo da solução através de cada nó. O objetivo é selecionar o nó mais promissor em termos de alcançar a solução de forma eficiente. Existem várias variantes da procura melhor-primeiro, incluindo procuras não informadas(pesquisa de custo uniforme), e informadas(pesquisa sôfrega e A*).

2.6.5 Procura de Custo Uniforme

A procura de custo uniforme é um tipo de procura melhor-primeiro que seleciona o nó com o menor custo acumulado para expandir. Utiliza uma função de avaliação que representa o custo acumulado para alcançar cada nó no espaço de busca. O objetivo é encontrar a solução com o menor custo total. Útil quando o custo é uma consideração importante na resolução do problema. Por exemplo, em problemas planeamento de rotas, onde se pretende encontrar a rota com o menor custo possível. No entanto, é importante notar que esta abordagem pode exigir muitos recursos computacionais, pois todos os nós visitados precisam de ser armazenados em memória durante o processo de busca.

2.6.6 Procura Sôfrega

Até aqui todas as procuras têm sido do tipo não informadas, em contraste desta e da próxima. Estas redefinem a função de avaliação para cada nó fazendo uso de uma função heurística, que é um método de auxílio à resolução do problema que representa uma estimativa do custo do caminho do nó atual até ao nó objetivo. Sendo uma estimativa, por si só não é suficiente para obter resultados ótimos de forma eficiente, podendo transmitir informação imprecisa.

A procura sôfrega (ou Greedy) não tem em conta o custo do percurso explorado, a sua procura é orientada para os nós que têm uma heurística melhor para o objetivo, minimizando o custo local, o que faz com que a sua função de avaliação iguale esta estimativa. Útil quando se deseja encontrar uma solução rapidamente, mesmo que não seja necessariamente a solução ótima. Pode ser aplicada em problemas onde a estimativa heurística é confiável e fornece uma boa indicação da proximidade do objetivo.

2.6.7 Procura A*

A procura A* é um algoritmo de procura informada que combina os benefícios da procura em largura e da procura sôfrega. Utiliza uma função heurística para estimar o custo total de um caminho, que é a soma do custo acumulado até ao nó atual e da estimativa heurística do custo restante até o objetivo.

A função heurística fornece uma estimativa do custo restante para alcançar o objetivo a partir de um determinado nó. Com base nessa estimativa, a procura A* seleciona o nó com o menor custo total (custo acumulado + estimativa heurística) e continua a expansão a partir desse nó.

A sua grande vantagem é que ela garante a obtenção de uma solução ótima, desde que a função heurística seja admissível, ou seja, nunca superestime o custo real até o objetivo. Além disso, evita a expansão excessiva de nós desnecessários, focando nos caminhos mais promissores.

No entanto, a eficiência da procura depende da qualidade da função heurística. Se a estimativa heurística for precisa e informar bem sobre a proximidade do objetivo, o algoritmo pode encontrar rapidamente a solução ótima. Caso contrário, pode se tornar mais lenta ou até mesmo não garantir a otimalidade.

2.7 Agentes Deliberativos

Um agente deliberativo é composto por uma representação interna do domínio do problema, que funciona como um mapa cognitivo do ambiente em que atua. Essa representação permite ao agente compreender o contexto e avaliar as opções disponíveis. Com base nessa representação, o agente elabora um plano de execução, que consiste numa sequência de ações a serem tomadas para atingir um objetivo desejado. A deliberação e o planeamento são processos cruciais nessa abordagem, pois envolvem a inferência das consequências das ações e a seleção da melhor estratégia a ser seguida.

No entanto, os agentes deliberativos enfrentam desafios relacionados aos recursos computacionais limitados. A memória e o tempo de execução desses agentes são restritos, o que requer otimização dos processos para lidar eficientemente com a complexidade das simulações internas. Além disso, o ambiente em que os agentes operam é dinâmico e sujeito a mudanças, o que pode afetar a validade dos planos elaborados. Nesses casos, é necessário um mecanismo de reconsideração das opções e ajuste do plano antes da execução.

O processo de tomada de decisão e ação em agentes deliberativos pode ser resumido em seis etapas principais:

1. Observar o mundo: O agente recolhe informações sobre o ambiente através de sensores, percebendo o estado atual do mundo ao seu redor.
2. Atualizar o modelo do mundo: Com base nas informações recolhidas, o agente atualiza o seu modelo interno do mundo, que representa o seu conhecimento sobre o ambiente, as suas características e os possíveis estados futuros.
3. Se Reconsiderar: O agente pode reavaliar as suas crenças, objetivos e restrições, tendo em conta as informações mais recentes. Isto permite que o agente ajuste o seu raciocínio e tome decisões mais adequadas.
   (a) Deliberar: O agente utiliza o seu modelo do mundo atualizado para analisar as opções disponíveis e os possíveis resultados de diferentes ações. Ele avalia o custo, benefício e risco associados a cada opção, tendo em conta os seus objetivos e restrições.
   (b) Planear: Com base na deliberação, o agente desenvolve um plano de ação detalhado. O plano descreve uma sequência de ações que o agente pretende executar para alcançar os seus objetivos.
4. Executar plano de ação: O agente inicia a execução do plano de ação, realizando as ações planeadas de acordo com a sequência definida. Durante a execução, o agente continua a observar o mundo e atualizar o seu modelo do mundo, adaptando-se a mudanças e ajustando o plano, se necessário.

Este processo de tomada de decisão e ação permite que o agente deliberativo seja capaz de perceber e compreender o ambiente, atualizar o seu conhecimento, refletir sobre as suas decisões, planear estrategicamente e executar ações de forma a alcançar os seus objetivos de maneira eficiente e eficaz.

2.8 Planeamento Automático com Base em PEE

O mecanismo de planeamento num agente autónomo é responsável por gerar planos de ação com base nos objetivos estabelecidos pelo mecanismo de deliberação. No caso de um planeador baseado em procura em espaços de estados, é necessário ter em consideração os seguintes elementos:

1. Modelo do problema de planeamento: O modelo do problema descreve as características e restrições do ambiente onde o agente está inserido, assim como as possíveis ações que o agente pode executar e como essas ações afetam o estado do ambiente. O modelo do problema é essencial para determinar as transições de estado e as consequências das ações.
2. Heurística (se necessário): Em alguns casos, pode ser útil utilizar uma heurística para orientar a procura e melhorar a eficiência do planeador. A heurística fornece uma estimativa do custo ou utilidade de um determinado estado, o que ajuda a priorizar a exploração de caminhos mais promissores durante a procura.

3. Mecanismo de procura: O mecanismo de procura é responsável por explorar de forma sistemática o espaço de estados, com o objetivo de encontrar um plano de ação que conduza à satisfação dos objetivos.

O planeador utiliza o modelo do problema, a heurística (se aplicável) e o mecanismo de procura para gerar um plano de ação que seja viável e eficiente para alcançar os objetivos estabelecidos pelo agente.

2.9 Processos de Decisão Sequencial

No caso de um problema determinístico, um problema é reduzido ao planeamento das ações e as suas consequências são imutáveis, ou seja, sabe-se que o resultado de uma determinada ação num determinado estado é sempre o mesmo. Num espaço de estados não determinístico (estocástico) a consequência de uma ação pode não corresponder à pretendida, havendo neste ambiente uma incerteza na decisão. Para implementar um agente deliberativo eficaz desta natureza, é importante que o seu processo de deliberação tenha em consideração fatores de não-determinismo do problema. Assim, a simulação interna do agente deve simular estas incertezas para chegar a uma solução ótima, apesar da sua natureza estocástica.

O Processo de Decisão de Markov (PDM) é um bom exemplo deste tipo de raciocínio. Efetua o cálculo do valor de utilidade de todos os possíveis estados de um problema e produz uma política comportamental que procure maximizar esse valor. Este método destaca-se porque em qualquer estado que se encontre, sabe sempre qual a ação ótima. Isto quer dizer que o agente nunca se encontra numa situação em que não sabe que ação tomar, desde que a política calculada seja ótima. A previsão dos estados seguintes só depende do estado presente.

2.9.1 Planeamento Automático com Base em PDM

Sobre este processo o mundo é representado por vários componentes que formam a Cadeia de Markov, que corresponde a uma sequência de encadeamentos ação-estado, com recompensas positivas ou negativas. Estas componentes são:

- S – conjunto de estados do mundo;
- A(s) – conjunto de ações possíveis no estado s;
- T(s,a,s') – probabilidade de transição de s para s' através de a;
- R(s,a,s') – recompensas esperada na transição de s para s' através de a;
- γ - taxa de desconto para recompensas diferidas no tempo;

Utilidade com base num modelo $U^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^{\pi}(s')]$

Figura 2: Utilidade com base num modelo - Processos de Decisão Sequencial - Luís Morgado - página 14

Neste domínio a utilidade é o efeito cumulativo da evolução da situação, e a recompensa é um ganho (ou perda) num determinado estado dada por um valor finito positivo (ou negativo), e começa sempre a 0. Iterando pelas utilidades, para todo o estado pertencente ao grupo de estados, atualiza-se o valor da utilidade do estado, maximizando-lhe o valor. As recompensas podem ser aditivas, quando a taxa de desconto não se aplica (γ=1), ou descontadas (γ¡1). A política é a forma de representar o comportamento do agente, definindo a ação a ser realizada em cada estado. Este γ na prática, significa que valores próximos a 0 prioritizam a obtenção de recompensas imediatas, enquanto valores próximos a 1 incentivam o agente a considerar também as recompensas futuras. Pode ser determinista se garantir uma ação, ou não-determinista se cada estado vier agregado a uma probabilidade.

O planeamento com base neste processo usa uma política, como já referido, que é no fundo a estratégia de decisão que indica para cada estado a ação a realizar e que maximiza a utilidade. A utilidade é a atribuição de valor a cada estado que indica quão bom é o estado para efeitos de concretização dos objetivos do sistema.

2.10 Aprendizagem por Reforço

Apesar de não termos desenvolvido nenhum programa que utilize aprendizagem por reforço esta foi-nos introduzida e lecionada com o intuito de nos mostrar o futuro no campo da Inteligência Artificial. A aprendizagem por reforço permite a um agente aprender a tomar decisões ótimas através da interação com um ambiente. Ao contrário de outros métodos de aprendizagem, esta não requer exemplos ou um modelo explícito do ambiente. Em vez disso, o agente aprende a partir de tentativa e erro, recebendo feedback na forma de recompensas ou punições pelo seu comportamento. É frequentemente usada em situações onde o agente deve enfrentar um problema de decisão sequencial, em que suas ações afetam o estado do ambiente e, por sua vez, influenciam as recompensas futuras. Essa abordagem é especialmente útil em jogos, robótica, otimização de recursos e muitas outras aplicações.

Dois dos algoritmos que aprendemos foram o Q-Learning e o SARSA. O processo de Q-learning envolve o agente explorar o ambiente, tomar ações com base na política atual e atualizar os valores de Q com base nas recompensas obtidas. A atualização dos valores de Q é feita por uma fórmula de atualização que leva em consideração a recompensa instantânea, o desconto futuro e o valor máximo de Q para o próximo estado. No processo SARSA, ao contrário do anterior, atualiza os seus valores com base na recompensa instantânea, na ação real tomada, no próximo estado e na próxima ação escolhida de acordo com a política atual.

Ambos utilizam a Estratégia ε-greedy que funciona da seguinte forma: com uma probabilidade ε (taxa de exploração), o agente escolhe uma ação aleatória, explorando o ambiente em busca de novas informações. Com uma probabilidade de 1 - ε (taxa de exploração), o agente escolhe a ação com o maior valor de Q, explorando o conhecimento adquirido até o momento.

3 Projeto Realizado

3.1 Jogo

Este primeiro projeto serve de introdução aos conceitos e temas que vamos abordar ao longo da disciplina e tem como objetivo a implementação de um agente reativo por meio de jogo com uma personagem virtual que interage com um jogador humano. O jogo consiste num ambiente onde a personagem tem o objetivo de registar a presença de animais através de fotografias.

Foram projetadas as seguintes classes para os mecanismos de cada entidade:

- Jogo – classe que contém o main, por isso executa o jogo;
- Personagem – classe do agente a ser simulado, a personagem que vai interagir com o ambiente, perceciona e atua, conforme um determinado controlo;
- Ambiente – estrutura evolutiva com quem a personagem vai interagir;
- Perceção – atualiza a perceção da personagem com base no evento corrente;
- Evento – enumerado de possíveis eventos no decorrer do jogo, produzem transições de estado, compõem o alfabeto de entrada no sistema, E;
- Ação – enumerado de possíveis ações a ser realizadas pela personagem, constituem o alfabeto de saídas do sistema, Z.

E as seguintes para o controlo da personagem, regras e objetivos:

- Estado – indica a situação da personagem;
- Transição – representa o que acontece a um estado, dado um determinado evento;
- MaquinaEstados – processar um evento, para saber a transição a seguir;
- Controlo – implementa a MaquinaEstados e é responsável por definir para cada estado a sua transição.

A máquina de estados tem o seguinte aspeto:

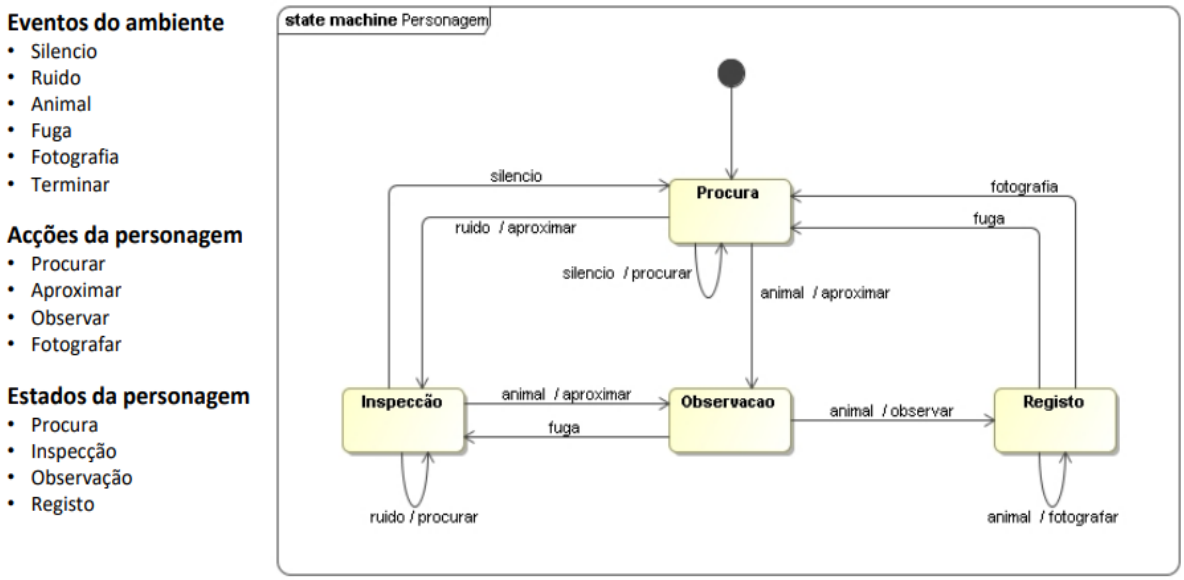


Figura 3: Máquina de Estados - iasa-jogo-2 - Luís Morgado - página 7

3.1.1 Teste Jogo

```
Estado: Procura
Evento?a
Evento: ANIMAL
Estado: Observação
Accao: APROXIMAR
Evento?f
Evento: FUGA
Estado: Inspecção
Evento?r
Evento: RUIDO
Estado: Inspecção
Accao: PROCURAR
Evento?a
Evento: ANIMAL
Estado: Observação
Accao: APROXIMAR
Evento?a
Evento: ANIMAL
Estado: Registo
Accao: OBSERVAR
Evento?o
Evento: FOTOGRAFIA
Estado: Procura
Evento?t
Evento: TERMINAR
```

Figura 4: Teste ao Jogo

Para testarmos o Jogo executámos a classe Jogo e respondemos de acordo com a imagem acima. Iniciamos em Estado de Procura o Animal aparece logo Aproximamos-nos e entramos no estado de Observação. Este foge logo passamos para um estado de Inspeção, de seguida ouvimos um Ruído logo continuamos a Inspecionar. Aparece outra vez o Animal, então Aproximamos-nos e Observamos. O Animal mantém-se em posição, logo passamos para o estado de Registo até que conseguimos Fotografa-lo. Como podemos verificar, os resultados vão de acordo com o expectável, pelo que se pode concluir que o agente reativo apresentou uma adaptabilidade às mudanças no ambiente, e obteve sempre as soluções que devia.

3.2 Agente Reativo

Neste projeto o objetivo também é criar um agente reativo, mas desta vez por meio de um agente cujo objetivo é recolher alvos e desviar-se de obstáculos. Este trabalho aumenta o nível de complexidade, isto deve-se o facto desta nova arquitetura implementar mais subpartes, sendo preciso uma maior modularização dos componentes. Para isso dividimos os módulos nos seguintes packages:

- controlo_react - faz com que o agente tenha uma reação com base num estímulo, contém todos os comportamentos, reações e estímulos associados ao agente. Está subdividido em packages de aproximar, evitar, explorar, recolher e resposta.
- ecr - esquemas comportamentais reativos, contém todas as classes que servem de base, ou seja, que integram as todas abstrações e métodos comuns para as classes do controlo reativo, incluindo comportamento, reação, comportamento composto, hierarquia, prioridade, estímulo e resposta.
- teste - package usado também em futuros projetos que contém as classes de teste aos agentes criados.

Numa fase inicial foram implementadas as classes da ecr, descrevendo todos os elementos necessários a um agente reativo cujo processo de tomada de decisão, como previamente explicado, ocorre em resposta a estímulos do ambiente, tendo como base um conjunto de regras estímulo-resposta. Assim é preciso descrever o que é um estímulo caso haja um alvo ou um obstáculo, a resposta que serve como resultado ao estímulo, e os diferentes tipos de comportamentos, tal como a escolha destes. Antes de poder testar o funcionamento do programa falta implementar as reações em si, organizadas cada uma na sua divisão, e que estão munidas do esquema comportamental reativo que as descrevem. Aproximar e evitar agregam especificações de estímulos, respostas direcionais e comportamentos, explorar é definido por um comportamento, recolher é definido como uma hierarquia onde a maior prioridade é a aproximação do alvo, seguida do desvio de obstáculos e por fim a exploração, e resposta mover gera a resposta numa dada direção.

3.2.1 Teste Agente Reativo

Para testar criámos o ficheiro teste_react onde vamos passar ao Simulador fornecido pelo docente uma instância do ControloReact que tem como parâmetro uma lista de comportamentos apenas com o Recolher.

Durante a simulação o agente, como descrito anteriormente, tem como prioridade máxima a aproximação ao alvo. Este alvo é escolhido através da classe EstimuloAlvo, em que para cada uma das direções possíveis esta deteta um dos alvos no mapa e devolve a intensidade desse estimulo de acordo com a distância entre os dois. A próxima prioridade é o evitar obstáculos que é tratada pela classe EstimuloObst que para cada direção possível devolve 1 de intensidade se houver contacto com esse movimento se não devolve 0 o que nos vai indicar quais são as direções possíveis. Por fim, se o agente não vir nenhum alvo nem tenha que evitar um obstáculo ele escolhe uma direção ao acaso. Esta solução não apresenta um percurso ótimo sendo que entre fases onde não tenha um alvo

no campo de visão, é deixado a explorar ou a evitar obstáculos, mas mais uma vez o objetivo de um agente reativo não é a chegada a soluções ótimas, mas sim a adaptabilidade a mudanças no ambiente, o que foi atingido neste teste.

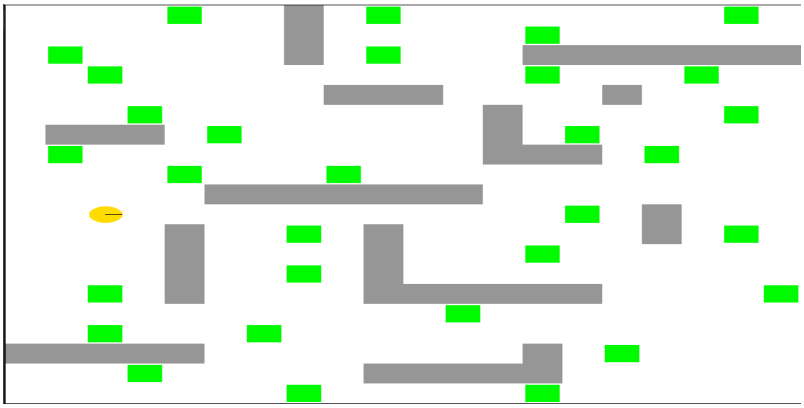


Figura 5: Simulador do agente Reativo

3.3 Procura em Espaço de Estados

Vamos outra vez aumentar o grau de complexidade e implementar um agente deliberativo, com um mecanismo de raciocínio, assim como a implementação de registos de dados da simulação. Inicialmente adicionamos à arquitetura os seguintes packages:

- mod – modelo de um problema, engloba as classes estado, operador e problema;
- pee – composto pelos módulos mec_proc, prof, larg, melhor_prim e classes que descrevem a solução encontrada;
- controlo_delib – contém as classes necessárias à implementação do processo de tomada de decisão.

O processo de desenvolvimento das funcionalidades nesta fase dividiu a estrutura da resolução do problema no seu modelo, diferentes mecanismos de procura (mec_proc), controlo do agente e os componentes necessários à sua resposta. O objetivo do package de mec_proc foi possibilitar a implementação de vários tipos de mecanismos de raciocínio sem ter de refazer o código não relacionado com os processos de resolução de um problema.

Para todos os mecanismos de procura a descrição modelo do problema é composta por um estado, operadores e o problema que os engloba e descreve o que se quer resolver. Cada mecanismo de procura é também composto por uma fronteira, e tem como objetivo chegar a uma solução, que vai ser composta pelos vários passos tomados para chegar da raiz ao objetivo. O primeiro passo foi implementar estas generalidades de um mecanismo de procura, para no próximo passo os descrever a cada um individualmente já havendo a base para cada um.

Foram então implementados os seguintes mecanismos de procura:

- Profundidade (DFS - Depth-First Search): Procura em profundidade, explora o grafo descendo o máximo possível antes de retroceder. Não é ótimo nem completo;
- Profundidade Limitada (DLS - Depth-Limited Search): Variação do DFS com um limite máximo de profundidade para evitar ciclos infinitos. Pode resultar em soluções subótimas ou falhar se o limite for muito restritivo;
- Profundidade Iterativa (IDS - Iterative Deepening Search): Combina a procura em profundidade com limites crescentes, é ótima e completa;
- Largura (BFS - Breadth-First Search): Procura em largura, explora o grafo por níveis. É ótima e completa, mas pode ser pesada em termos de recursos computacionais.
- Custo Uniforme (UCS - Uniform Cost Search): Procura com base no menor custo acumulado até ao momento. Garante otimalidade, mas não lida bem com custos infinitos;
- Sôfrega (Greedy Search): É um algoritmo de procura informada que seleciona o nó que parece mais promissor com base numa heurística. A heurística avalia a qualidade de um estado em relação ao objetivo, sem considerar o custo real para alcançá-lo. O algoritmo prioritiza a procura em direção ao objetivo, não garante ser completo nem ótimo;
- A* (A* Search): É um algoritmo de procura informada que combina a procura de custo uniforme com uma heurística adicional. Seleciona o nó com o menor custo total estimado, que é a soma do custo acumulado e da estimativa heurística. O A* é ótimo, completo e eficiente quando a heurística é admissível (não superestima o custo real). É um dos algoritmos mais amplamente utilizados em problemas de procura.

Para suportar estes mecanismos, foram implementados os tipos de memórias de procura, ou fronteiras, das quais as LIFO e FIFO suportam os métodos de procura não informados, e a de Prioridade que é compatível com o uso de uma função avaliador para todas as suas procuras, e faz uso de uma função heurística para ordenar a fronteira de exploração das procuras informadas.

O planeamento para estes métodos de procura foi dividido em três testes disjuntos, o planeador de trajeto, o planeador PEE e o planeador sequência. Os três encapsulam a resolução do problema, mas o primeiro foi usado para testar as procuras sem avaliador e não informadas, retratando o caminho por meio dos nós explorados, representados pelos seus estados, e o segundo foi usado para testar as procuras com avaliador, com e sem recurso a uma heurística mostrando o percurso de um agente a capturar diversos alvos num dado ambiente. O terceiro teve o mesmo objetivo que o segundo mas desta vez tivemos que o construir assim como todos os seus componentes sem a ajuda do docente ao contrário do que viemos a fazer até agora.

O planeador de trajeto é implementado com os seus estados e operadores integrados no problema. Ele recebe um estado inicial e final, sendo que cada estado é representado como uma localidade. Cada operador é visto como uma ligação entre dois estados, com um custo de transição associado. O objetivo do planeador de trajeto é encontrar o melhor caminho entre os dois estados, de acordo com o método de procura escolhido.

Já o planeador PEE é implementado com um modelo do mundo, e os alvos que tem de apanhar, recebendo no seu problema também a representação do mundo, e desta vez apenas o estado final, podendo ou não receber também uma heurística de distância a esse estado. Tem como objetivo recolher todos os alvos, da forma mais eficiente que lhe for possível.

O planeador sequência é uma junção dos dois anteriores, onde temos que chegar a uma determinada sequência de números a partir doutra. Cada operador é visto como uma troca de posição entre dois números, com um custo igual à diferença entre as posições dos dois números. O objetivo do planeador de trajeto é encontrar o melhor caminho entre os dois estados, de acordo com o método de procura escolhido.

3.3.1 Teste Planeador de Trajetos

A tabela com a informação sobre os trajetos entre localidades e os seus custos é a seguinte:

Localidade inicial	Localidade final	Custo
Loc-0	Loc-1	5
Loc-0	Loc-2	25
Loc-1	Loc-3	12
Loc-1	Loc-6	5
Loc-2	Loc-4	30
Loc-3	Loc-2	10
Loc-3	Loc-5	5
Loc-4	Loc-3	2
Loc-5	Loc-6	8
Loc-5	Loc-4	10
Loc-6	Loc-3	15

Figura 6: Planeador de Trajetos entre Localidades - iasa-pee-3 - Luís Morgado - página 3

No teste pedido pelo docente o objetivo foi com o uso da Procura Custo Uniforme ir da Localização 0 até à Localização 4, e os resultados foram os seguintes:

```
Loc-0 : None  Custo :  0
Loc-1 : Loc-0 -> Loc-1  Custo :  5
Loc-3 : Loc-1 -> Loc-3  Custo :  17
Loc-5 : Loc-3 -> Loc-5  Custo :  22
Loc-4 : Loc-5 -> Loc-4  Custo :  32
```

Figura 7: Simulador Trajetos

Como podemos observar, de acordo com esta procura, o agente chegou ao resultado correto pois este de facto é o caminho mais eficiente em termos de custo. Em termos de trajetos o mais eficiente seria ir da Localização 0 para a 2 e desta para a 4, o que iria equivaler a um custo de 55, este é provavelmente o resultado duma Pesquisa por Largura.

3.3.2 Teste Planeador PEE

Neste teste foi nos pedido que usássemos o simulador disponibilizado pelo docente para descobrir qual das três: Procura Custo Uniforme, Procura Sôfrega e Procura A* é a mais eficiente quando damos a um agente um modelo mundo com obstáculos e 3 objetivos. Este é o modelo mundo:

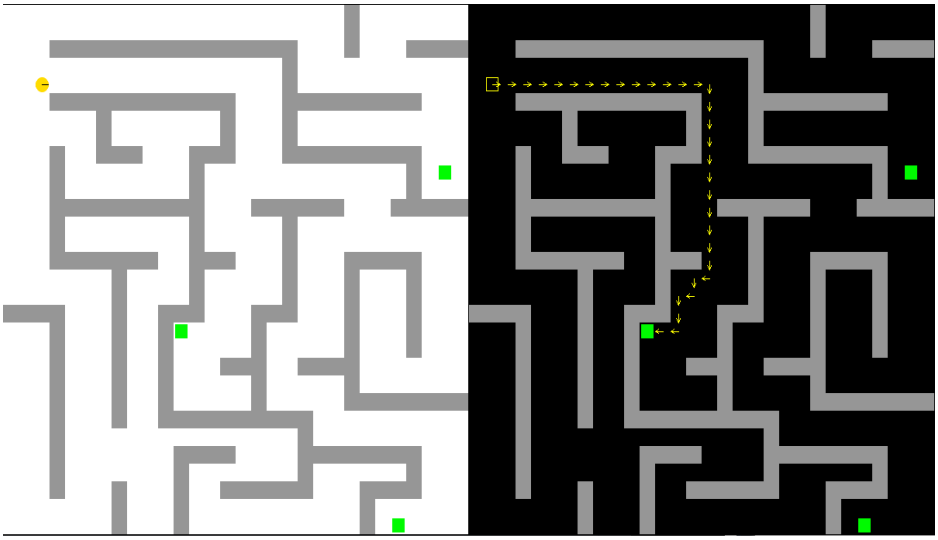


Figura 8: Simulador PEE

Os resultados foram os esperados com a procura menos eficiente a ser a Procura Custo Uniforme e a mais eficiente a Procura A*, sendo a Procura Sôfrega um intermédio.

A Procura Custo Uniforme sendo um algoritmo de busca em que o custo total do caminho percorrido até o momento é levado em consideração, ou seja, fez com que o algoritmo precise de visitar muitos nós antes de encontrar a solução ótima, o que não é bom tendo em conta o tamanho desta simulação

A Procura Sôfrega é uma estratégia que explora o espaço de busca indo o mais longe possível em cada ramificação antes de retroceder. Esta não é a melhor procura para esta simulação devido ao facto de esta funcionar como um labirinto,ou seja, existem vários longos caminhos que não vão dar em nada, o que faz com que grande quantidade de nós sejam explorados desnecessariamente.

A Procura A* combina a procura custo uniforme e uma heurística que estima o custo do caminho restante até a solução. Logo devido à heurística não é necessário explorar todos os caminhos ao contrario da Procura Custo Uniforme.

3.3.3 Teste Sequência

Por fim, neste último teste foi nos pedido para criar todo o modelo problema assim como o planeador sozinhos. Este tem o objetivo de, a partir de uma sequência inicial e final fazer essa transição usando a procura escolhida. A sequência inicial foi [3, 6, 4, 2, 5, 1] e a final [1, 2, 3, 4, 5, 6] e as procuras utilizadas foram a Procura Custo Uniforme e a Procura A*. Os resultados foram os seguintes:

```
Solução ProcuraCustoUnif:          Solução ProcuraAA:                 Solução ProcuraSofrega:
[3, 6, 4, 2, 5, 1] None            [3, 6, 4, 2, 5, 1] None            [3, 6, 4, 2, 5, 1] None
[3, 6, 2, 4, 5, 1] Troca(2, 3)     [3, 6, 2, 4, 5, 1] Troca(2, 3)     [3, 6, 2, 4, 5, 1] Troca(2, 3)
[3, 2, 6, 4, 5, 1] Troca(1, 2)     [3, 2, 6, 4, 5, 1] Troca(1, 2)     [3, 2, 6, 4, 5, 1] Troca(1, 2)
[3, 2, 1, 4, 5, 6] Troca(2, 5)     [3, 2, 1, 4, 5, 6] Troca(2, 5)     [6, 2, 3, 4, 5, 1] Troca(0, 2)
[1, 2, 3, 4, 5, 6] Troca(0, 2)     [1, 2, 3, 4, 5, 6] Troca(0, 2)     [1, 2, 3, 4, 5, 6] Troca(0, 5)
Dimensão:  5                       Dimensão:  5                       Dimensão:  5
Custo:  7                          Custo:  7                          Custo:  9
Complexidade temporal:  699        Complexidade temporal:  30         Complexidade temporal:  5
Complexidade espacial:  720        Complexidade espacial:  237        Complexidade espacial:  54
```

Figura 9: Simulador Sequência

Apesar de ambas as procuras terem a mesma dimensão e o mesmo custo a Procura A* é claramente melhor pois observando ambas as complexidades (temporal e espacial) conseguimos ver que esta tem menores valores comparando com a Procura Custo Uniforme, logo é mais eficiente. Este era o resultado esperado pois vai de acordo com tudo o que temos estudado até aqui. Para uma maior diferença, também fiz o teste com a Procura Sôfrega, que teve de longe as complexidades mais eficientes o que era o esperado devido à sua natureza ”greedy”. Isto vem em detrimento do custo que subiu ao contrário das outras procuras.

3.4 Processo de Decisão de Markov

Este último projeto teve como objetivo, também, a implementação de um agente deliberativo, mas desta vez implementando o raciocínio do processo de decisão de Markov. Esta arquitetura, fazendo também parte de um agente deliberativo faz uso do controlo deliberativo já criado previamente. Para este projeto bastou criar 2 packages:

- pdm – na biblioteca, contém a classe PDM que implementa o processo e o modelo abstrato que vai ser usado;
- plan_pdm – onde estão o planeador deste processo e a classe que vai implementar o modelo do mundo do agente, que sofre de uma injeção de dependências, partilhando o modelo de plano usado e testado na fase anterior, com o modelo situado no package pdm que obriga à implementação dos métodos dos componentes específicos ao PDM.

Neste tipo de processo, a transição de um estado para outro é determinada pela propriedade de Markov, que estabelece que a probabilidade de transição depende apenas do estado atual e da decisão tomada, sendo independente do histórico de eventos passados. Isso significa que o estado futuro depende apenas do estado atual e da ação escolhida, sem ser afetado pelo caminho percorrido até o momento.

A primeira parte deste projeto foi implementar a classe do método de procura em si e o modelo que este obriga que seja implementado. Desta forma são introduzidas a utilidade e a política, juntamente com os outros componentes necessários à cadeia de Markov: gama, estados do mundo, ações e transições possíveis, e recompensas para uma transição. Por fim, foi implementado o planeador PDM, que faz uso das classes criadas previamente, e no fundo resolve o problema, retornando um plano válido de execução.

3.4.1 Teste PDM

Na classe teste_delib_pdm, assim como na teste_delib_pee, o agente têm que percorrer um labirinto à procura dos seus alvos. Este é o aspeto do simulador, em que a cor verde indica o custo, ou utilidade, e a seta a direção da ação, ou política, para cada estado:

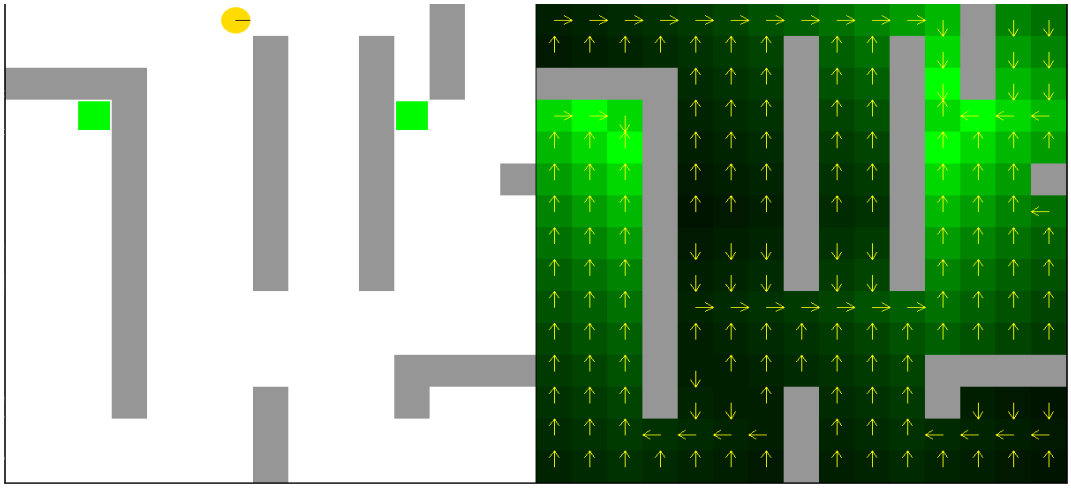


Figura 10: Simulador PDM 3

Como foi dito no enquadramento teórico, uma grande vantagem deste processo é que o agente em qualquer estado que se encontre sabe sempre a solução ótima, ou seja, sabe sempre o caminho que deve seguir, o que se pode comprovar pela política comportamental demonstrada no lado direito da imagem. A utilidade também é tanto maior quanto mais perto do objetivo se encontrar o estado, que também vai de acordo com as características do processo de Markov.

Foi nos sugerido uma gama = 0.85 que funciona perfeitamente com o simulador da imagem acima mas quando tentamos o próximo algo estranho acontece.

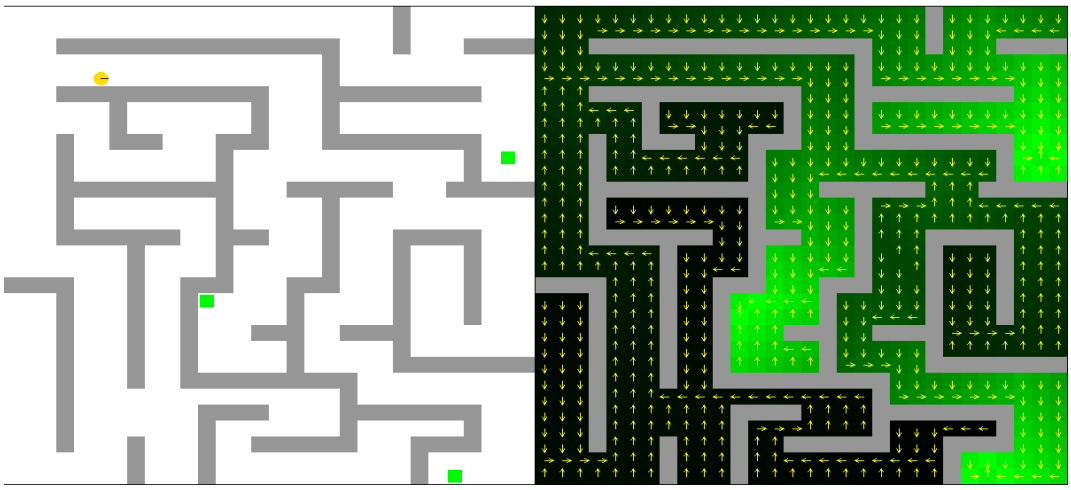


Figura 11: Simulador PDM 4

O agente percorre o caminho esperado até ao primeiro alvo mas quando o apanha isto acontece:

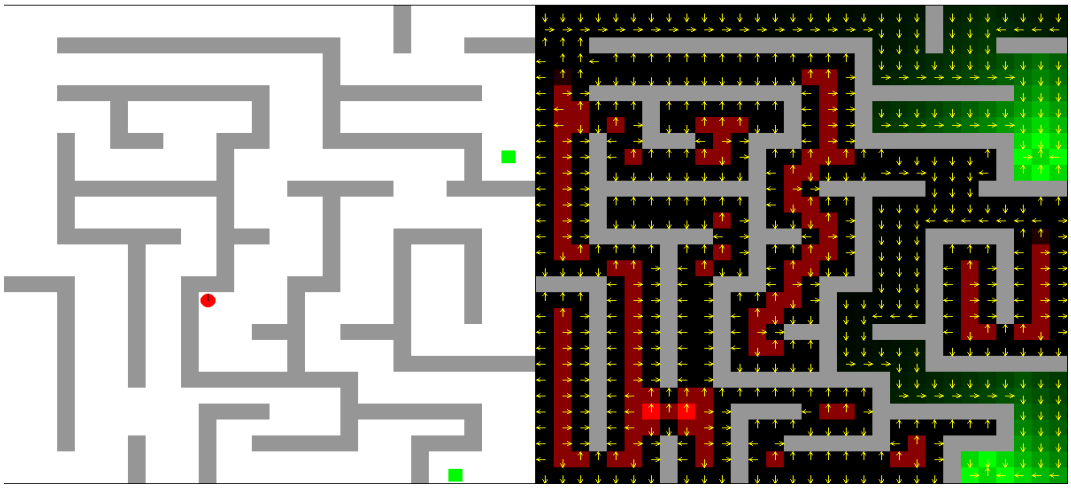


Figura 12: Simulador PDM 4 Erro

Este comportamento deve-se ao facto de, devido ao tamanho do simulador ser consideravelmente maior que o anterior a gama também tem que ser ajustada pois este 0.85 não é suficiente para o agente considerar que valha a pena ir atrás do alvo, logo fica onde está. Isto pode ser remediado se aumentarmos a gama para um valor em que o agente já considere favorável ir atrás do alvo, neste caso usámos 0.95 o que embora necessite de mais recursos da máquina, o que influenciou o tempo de espera a abrir o programa, fez com que o agente fosse atrás de todos os alvos. Esta gama simboliza então o valor descontado ao agente em cada passo, ou seja, é inversamente proporcional à gama.

4 Revisão do Projeto Realizado

O desenvolvimento dos vários projetos foi em, em si, um processo de aprendizagem. Torna-se importante consolidar os conhecimentos adquiridos ao longo do semestre. Esta secção dedica-se a propostas de alteração nas entregas realizadas.

Nas entregas inicias houve várias em que não consegui entregar tudo o que o docente pediu sendo a maior parte dessas as explicações teóricas. Isto foi algo que eu sempre remendei, ao enviar outra entrega e substituir a incompleta algumas horas depois mas no mesmo dia. Tirando os comentários também existiram alguns erros nas entregas como o engano nos nomes de métodos ou classes.

O erro mais importante, surgiu no desenvolvimento da classe ControloDelib que foi concebida na parte 9 do trabalho e apenas foi resolvida na última entrega. Este erro está no método __planear onde obtemos o plano através do planeador passando-lhe o modelo mundo e os objetivos algo que só acontece se existirem objetivos. O problema está no facto de não ter sido desenvolvido uma resposta ao caso de não existirem mais objetivos, como por exemplo se o agente apanhasse-os todos. De modo a resolver este problema acrescentámos a condição que se não existirem mais objetivos passamos o plano para None. Para ir de acordo com esta solução no método seguinte __executar adicionámos a verificação que só tentamos obter o operador se existir um plano, ou seja, este não é nulo. Isto vai de acordo com a teoria pois, se não existem objetivos não existe a necessidade de criar um plano. Fica em baixo uma imagem dos métodos em questão.

```
def __planear(self):
    """
    Tem como objetivo obter o plano através do planeador passando lhe
    o modelo do mundo e os objetivos
    Isto só acontece se existirem objetivos
    Se não colocamos o plano a None

    """
    if self.__objetivos:
        self.__plano = self.__planeador.planear(self.__modelo_mundo,
                                                 self.__objetivos)
    else:
        self.__plano = None

1 usage
def __executar(self):
    """
    Obtém-se, chamando o método obter_accao() do plano, o operador que deverá se
    no modelo do mundo.
    Se este não for nulo e existirem objetivos retornamos a ação a este ligada.

    Returns: ação que o agente deverá executar

    """
    if self.__plano:
        operador = self.__plano.obter_accao(self.__modelo_mundo.obter_estado())
        if operador:
            return operador.accao
```

Figura 13: Classe ControloDelib métodos __planear e __executar

5 Conclusões

Os conceitos explorados ao longo do desenvolvimento destes projetos são de grande importância no dia-a-dia de qualquer pessoa envolvida não só no mundo da engenharia de software, mas de programação em geral. Com o aparecimento de tecnologias cada vez mais avançadas, nota-se também um aumento na complexidade nestes projetos, e um aumento das equipas que os desenvolvem. O elevado número de pessoas a trabalhar num mesmo projeto obriga à sistematização do processo de implementação de software desde a sua projeção e conceção, até à sua mobilização, incentivando a eficiência, organização, e uma boa documentação.

No que toca a engenharia de software os conceitos chave a reter são os seus três princípios: modularização, factorização e abstração. No que toca à inteligência artificial, e o desenvolvimento de agentes inteligentes e autónomos, os projetos realizados potencializaram a aquisição de novas competências nesta área, compreendendo diferentes tipos de arquiteturas e modos de implementação. Num contexto real, as aplicações destas noções variam, podendo ser implementados na condução autónoma de um veículo, motores de videojogos, jogos de tabuleiros, entres outros.

Em suma, esta disciplina foi uma excelente forma de melhorar as minhas capacidades com projetos deste nível e que é notório a minha evolução olhando para o inicio do semestre em que tinha dificuldade em acabar o trabalho da aula para o final onde acabava muito antes do fim e tinha tempo para verificar e comentar tudo ao pormenor. Por fim, creio que com os projetos desenvolvidos e este relatório, demonstrei o domínio que tenho sobre a matéria lecionada.