ADDETC – Área Departamental de Engenharia Eletrónica e Telecomunicações
e de Computadores

LEIM -Licenciatura Engenharia informática e multimédia

Inteligência Artificial para Sistema Autónomos

Trabalho prático 1, 2 e 3

Turma:
LEIM-42D

Trabalho realizado por:
Miguel Távora Nº45102

Docente:
Luís Morgado

Índice

Índice ilustrações

1.Introdução

O primeiro trabalho prático tem como objetivo principal a introdução aos princípios base da Inteligência artifícial.

Desta forma, para implementar uma inteligência mais simples foi utilizada a interatividade de uma personagem virtual de um jogo. A personagem interage com um jogador humano. O objetivo principal da personagem é impedir que os inimigos entrem numa zona à sua guarda, de tal forma que, o ambiente e o jogador proporcionam as diferentes ações da personagem.

O segundo trabalho prático tem por objetivo a concretização de mecanismos de raciocínio automático. No caso presente para procura em espaços de estados.

De forma a concretizar o comportamento de procura irá ser implementada uma biblioteca com procuras em espaços de estados designada por pee. Esta biblioteca possui diversos tipos de procuras.

A partir da biblioteca implementada são feitos alguns testes para verificação do bom funcionamento da mesma. Os testes são nomeadamente um exemplo de ligações de uma rede de transportes e a realização automática do puzzle de 8 peças para as diversas procuras. A partir dos testes é feita uma análise ao desempenho da biblioteca no que toca à complexidade temporal e espacial.

O propósito do terceiro trabalho prático é a concepção de um agente inteligênte que opera numa plataforma de simulação com alvos e obstáculos. Este ambiente não sofre alterações ao longo do tempo, sendo que o objetivo é o agente recolher os alvo.

Para a conceção deste agente serão feitas múltiplas implementações de comportamentos do agente. Cada uma destas possui as suas características e benefícios.

De forma a tornar a implementação o mais eficiente possível, foi-nos dado uma estrutura pré-definida em forma de UML das classes e dos métodos. Desta forma, foi possível manter o nível de complexidade estável para o objetivo pretendido.

2. Desenvolvimento

2.1 Teoria

1. Aspetos relevantes

1.1 Significado de estímulo e ação

- Um estímulo são os elementos de ação relevantes para o sistema.
- Uma ação é a resposta produzida pelas diferentes componentes que constituem o sistema.

1.2 Significado de autonomia

- A autonomia é a capacidade de um sistema operar por ele próprio sem a necessidade de outros agentes.
- Um sistema autónomo pode não conter inteligênte, porém um sistema inteligente tem de ser autónomo.

1.3 Racicínio automático

- Realiza explorações de opções ou por raciocínio prospetor, isto é, por antecipação ou por simulação interna do mundo onde é necessário uma representação interna do mesmo.
- A avaliação das opções pode ser feita por custo ou utilizadade.

1.4 Arquitetura de subsunção

- O comportamento é organizado em camadas e responsável pela concretização independente de um objetivo.
- O resultado do comportamento pode ser a entrada de outro comportamento.
- Possibilidade do comportamento camadas superiores assumirem o controlo por: inibição, supressão, reinício.

- As camadas inferiores não tem conhecimentos das superiores.

1.5 Arquiteturas agentes

- Presente: Agentes reativos sem estado que apenas reagem.
- Passado-Presente: Agentes reativos com estado, isto é, memória.
- Passado-Presente-Futuro: Agentes deliberativos simulam situações.

2. Teoria primeiro trabalho prático

2.1 Modelo Comportamental simbólico

Existe um conjunto de símbolos designados por alfabetos, no qual temos o alfabeto de entrada, designado pelo símbolo $\sum$ e o alfabeto de saída, designado por Z. A função de saída, definida pelo simbolo lambda λ, que é responsável por mapear conjuntos entrada para elementos do conjunto de saída.

Função de saída:

$$\lambda : \Sigma \rightarrow Z$$

Para sistema com memória é necessário uma nova componente designada por estado interno. O estado interno do sistema representa um dos estados possíveis que o sistema pode assumir, este é simbolizado por *Q*. Desta forma, obtemos uma função de transição de estado que codifica as regras dinâmicas e a função de saída.

Função transição de estado:

$$\delta : Q \times \Sigma \rightarrow Q$$

Função de saída:

$$\lambda : Q \times \Sigma \rightarrow Z$$

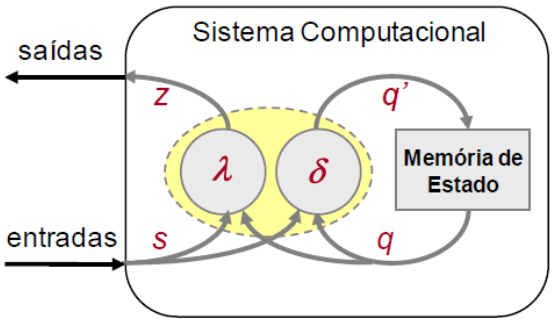

Figura 1 - Função transformação

2.2 Arquiteturas Reativas

Existem três arquiteturas de agentes, porém nesta fase iremos somente dar destaque a arquiteturas reativas. Este tipo de arquitetura dá ênfase ao acoplamento com o ambiente, o mecanismo utiliza por base a regra estímulo-resposta.

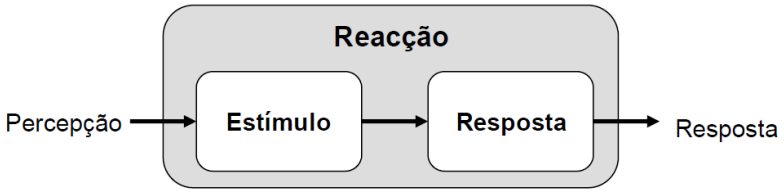

Figura 2 - Reacão de regra estímulo | resposta

De acordo com a arquitectura, é possível construir seguidamente o modelo de representação, no qual para diferente estimulos tem-se diferentes respostas. Na tabela em baixo representa um exemplo para um tipo genérico de comportamento com arquitetura reativa.

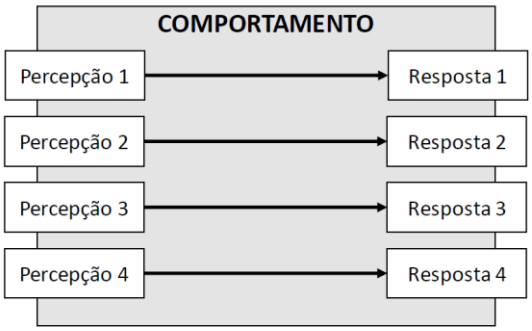

Figura 3 - Mecanismo de reação

Contudo, este tipo de regras não mantém nenhuma representação interna do estado do mundo. Nesta arquitetura existe uma conexão direta entre a perceção e a ação, surgindo assim o problema da necessidade da manutenção do estado. Para solucionar esse problema é assim necessário implementar uma arquitetura reactiva com memória. Desta forma, é possível representar dinâmicas temporais, nomeadamente evolução de estado. A dinâmica desta arquitectura pode ser expressa como uma transição perante o estado atual e as entradas atuais, produzindo o estado seguinte e as saídas seguintes.

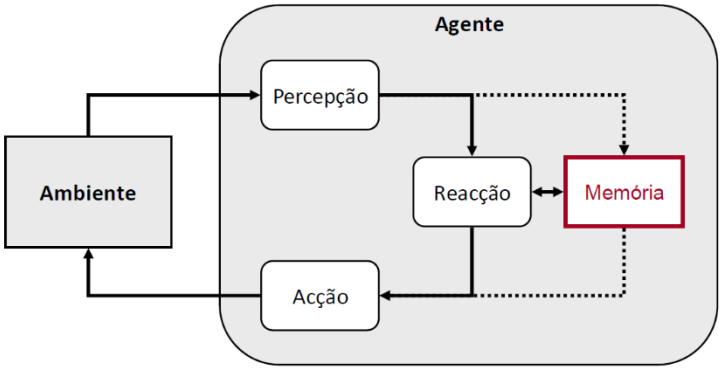

Figura 4 - Arquitetura reativa com memória

3.Teoria trabalho prático 2

3.1 Processo de exploração

O processo de exploração é feita por explorações sucessivas de espaço de estados, onde cada etapa de procura é representada por um nó na forma de uma árvore de procura. A raiz dessa árvore é o estado inicial.

As componentes base para realizar a exploração são: estado, operador, problema e solução. Um estado define uma situação possível que o problema pode assumir. Um operador define uma transformação onde a transformação pode ser feita por utilidade, isto é, quando todos os nós possuem o mesmo custo, ou por custo onde cada nó possui um custo associado diferente dos restantes. O problema é o que define o estado inicial, o objetivo ou estado final e também os operadores. A solução é o percurso realizado desde o estado inicial até ao estado final.

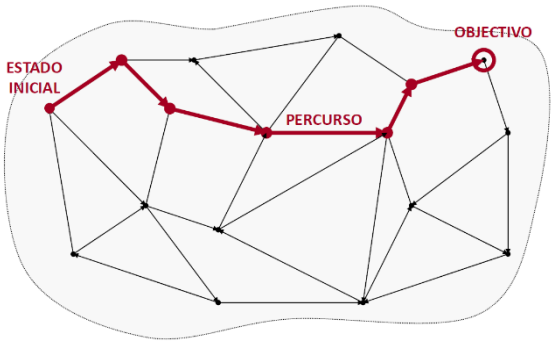

Figura 5 - Componentes do processo exploração

3.2 Estados repetidos na árvore procura

Quando num processo de procura existe reversibilidade nas ações de que correspondem às transições estado, o grafo do espaço estados apresenta ciclos provocando um desperdício de recursos tanto de tempo como de memória.

Para solucionar o problema surge a memória de nós processados. Para nós gerados mas não expandidos ficam na fronteira de exploração como abertos e nós já expandidos são fechados.

3.3 Procura Melhor-Primeiro

Esta procura tem como objetivo tirar partido de uma avaliação do estado, utilizando uma função f para avaliação de cada nó n gerado. A função f(n) é sempre maior ou igual a 0 e f(n) representa uma estimativa do custo da solução através do nó n. A fronteira de explorados é ordenada por ordem crescente de f(n).

3.4 Métodos de Procura Informada

Os métodos de procura informada tiram proveito do conhecimento do domínio do problema para facilitar a procura, tornando-a numa procura guiada.

3.5 Função heurística h(n)

A função heurística é uma função que estima o custo do percurso desde o nó n até ao nó objetivo. Da mesma forma que a procura Melhor-Primeiro reflete conhecimento acerca do domínio do problema para guiar a procura. O seu valor é independente do percurso até n pois depende somente do estado objetivo, por outro lado a procura Melhor-Primeiro utiliza função f(n) para avaliar cada nó n gerado.

4. Teoria trabalho prático 3

4.1 Agente Reativo

4.1.1 Arquitetura de subsunção

O agente realizado será um agente sem memória onde a sua arquitetura possui uma implementação com base numa sequência de ativação fixa de procedimentos. Este tipo de arquitetura é uma alternativa a abordagens simbólicas referidas anteriormente, onde define um conjunto de comportamentos. A sua implementação é robusta e relativamente simples de implementar.

4.2 Agente Deliberativo

4.2.1 Raciocínio prático

O raciocínio prático é um raciocínio orientado para a ação, onde as questões que se fazem são: o que fazer e como fazer. Os elementos de suporte a este raciocínio são: a representação dos objetivos a atingir, a representação das ações realizáveis e representação do mundo (ambiente).

4.2.2 Raciocínio meios - fins

O raciocínio fim de finalidade define o que se pretende atingir e qual o objetivo. Os meios definem qual o meio para atingir o fim noemadamente a ação e o plano. Na deliberação, ou raciocínio sobre fins é feita a decisão sobre o que fazer, isto é, as opções e o resultado ou objetivos do problema. No planeamento, ou raciocínio sobre meios é feita a decisão das ações e quais os planos para o resultado.

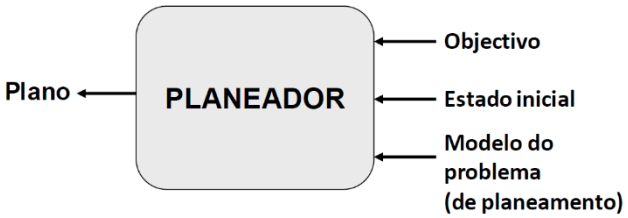

Figura 6 - Planeamento automático

4.2.3 Processo de tomada de decisão e ação

1. Observar o mundo
2. Atualizar crenças
3. Caso se reconcidere
   4. Deliberar
   5. Planear
6. Executar o plano

4.2.4 Propriedade de Markov

Um processo tem a propriedade de Markov se a distribuição probabilidade condicional dos estados futuros de um processo depender exclusivamente do estado presente. A partir desta propriedade é possivel criar uma representação do mundo sob a forma de um PDM.

4.2.5 Representação do mundo sob a forma de um PDM

- S : conjunto de estados do mundo
- A (S) : conjunto de ações possíveis
- T (S, A, S') : probabilidade de S transitar para S' através de A

- R(S, A, S') : retorno esperado na transição de S para S' através de A
- **γ :** taxa de desconto para recompensas diferidas no tempo

4.2.6 Utilidade

A utilidade possui um efeito cumulativo da evolução da situação. Conforme evolui o sistema numa sequência de estados estes possuem ganhos e perdas. Cada estado pode ter uma recompensa de ganho ou perda conforme determinado o estado, este valor tem de ser finito e pode ser positivo ou negativo.

4.2.7 Política comportamenteal

A política comportamental é nada mais que a forma de representação de um comportamento do agente, isto é, qual a ação realizada pelo agente em cada estado. A política pode ser determinística ou não determinística.

- Política deterministica π: S -> A(s); s ∈ S
- Política não determinista π: S * A(s) -> [0,1]; s ∈ S

4.2.8 Principio da solução ótima

Esta metodologia é uma programação dinâmica que requer primeiramente a decomposição do problema em sub-problemas. Num PDM isso deriva de assumir a indepêndencia dos caminhos, as utilidades dos estados podem ser determinadas em função das utilidades dos estados sucessores.

4.3 Aprendizagem por reforço

4.3.1 Conceito Aprendizagem

Tendo por base o conceito de aprendizagem este é a melhoria de desempenho para uma dada tarefa ganha através da experiência. Onde é possível melhorar o desempenho para uma dada tarefa T, com base numa medida de desempenho D e na expeciência E. A aprendizagem surge como a generalização e a formação de abstrações que podem ser decompostas em protótipos, conteitos e padrões comportamentais.

4.3.2 Aprendizagem conceptual e comportamental

A aprendizagem automática divide-se em duas vertendes a aprendizagem conceptual e comportamental.

Apredizagem conceptual é a aprendizagem do conceito, pode ser supervisionada ou não.

Aprendizagem comportamental é a aprendizagem sobre o comportamento, será nesta apredizagem que será o foco visto que o objetivo é aprender comportamentos.

4.3.3 Conceitos Aprendizagem por Reforço

A aprendizagem é feita a partir da interação com o ambiente com estados, ações e o reforço com ganhos e perdas. A aprendizagem de comportamentos é o que fazer e a relação entre as situações e as ações. A aprendizagem por reforço também utiliza política comportalmental e a utilidade que possui recompensas aditividas e descontadas conforme as ações.

4.3.4 Como determinar o valor Q

O valor Q é o valor de cada ação, para isso é necessário testar onde o foco será na ação feita de forma incremental.

Feito de forma incremental onde cada tentativa cada vez conta menos para o conhecimento do problema:

$$Q_n^k = Q_{n-1}^k + \frac{1}{n}\left[r_n^k - Q_{n-1}^k\right]$$

Esta metodologia permite decresce incremental o seu valor, pois no início o conhecimento é 0 sendo necessario aproveitar todo o conhecimento e conforme as tentativas cada vez fica menos relevante o novo conhecimento. No qual a função concreta que tem por base o conhecimento da aprendizagem por reforço é:

$$Q_n^k = Q_{n-1}^k + \alpha[r_n^k - Q_{n-1}^k]$$

$\alpha \in [0,1]$ - Factor de aprendizagem

4.3.5 Dilema Explorar/ Aproveitar

A questão presente é a altura onde o agente aprendeu o suficiente para aproveitar o conhecimento adquirido e começar a realizar ações sobre o que aprendeu. Então surge o dilema de explorar ou aproveitar.

A exploração é escolher uma ação que permite explorar o mundo para melhorar a aprendizagem.

O aproveitamento é quando se escolhe uma ação que leva á melhor recompensa de acordo com a aprendizagem, que é essencialmente uma procura sôfrega. Contudo esta pode ainda não ser a política ótima.

Para solucionar este dilema o que é feito efetivamente é que o agente realiza ambos contudo conforme o passar do tempo a exploração vai tendo cada vez menos peso na ação do agente, e este tende cada vez mais para a procura sôfrega.

4.3.6 Aprendizagem Associativa

Esta aprendizagem pode evoluir ao longo do tempo através de:

- Estados observados : s Є S

- Ações realizadas : a Є A
- Reforços obtidos: r Є |R
- Valor de num estado realizar uma ação: Q(s,a)

4.3.7 Processo Aprendizagem

Existem dois tipos de aprendizagem: política por seleção de ação única e política seleção de ações diferenciadas.

A política seleção ação utiliza a mesma política de seleção de ações para comportamento e propagação de valores, onde explora todas as ações (política da procura e-Greedy).

A política seleção de ações diferenciadas utiliza políticas de seleção de ações distintas para comportamento e propagação de valor, produz a otimização da função valor Q(s,a).

2.2 Concretização

2.2.1 Concretização trabalho prático 1

1. Implementação personagem como agente reativo

Na implementação é preciso primeiramente ter em conta as três primeiras componentes base que são: o Jogo, a Personagem e o Ambiente.

1.1 Ambiente

O ambiente será responsável por desencadear eventos e mostrar os mesmos eventos. Desta forma o ambiente possui três classes diferentes uma que representa o próprio ambiente, este responsável por gerar eventos, evoluir, terminar e mostrar os acontecimentos. Possui ainda dois enumeradores que são os nomes dos eventos possíveis e o nome da ação que foi desencadeada pelo evento.

1.2 Personagem

A personagem será responsável por percecionar o estímulo do ambiente, realizar o processamento do estímulo no qual resulta uma ação e de seguida realiza essa ação. A classe personagem possui os métodos percecionar, processar e atuar, que realizam os comportamentos referidos anteriormente. Para além da classe personagem existe ainda um conjunto de classes associadas ao comportamento. Cada classe representa uma reação distinta que a personagem poderá realizar e também uma classe para inicialização de todos esses comportamentos, organizando estes em estados.

1.3 Jogo

O jogo será responsável por correr o programa, neste caso será a classe com o método main que chama a função executar para começar o jogo.

2. Biblioteca Reacao e maqest

Os agentes reativos como foi dito anteriormente percecionam uma alteração no ambiente que desencadeia uma reação e o agente atua sobre a mesma. Esta biblioteca tem como interesse principal fazer de forma clara e precisa a transição entre um estímulo e uma ação. Esta biblioteca interage com outra biblioteca designada maqest. A biblioteca maqest é utilizada para agentes reativos com memória incorporada.

Durante a implementação do código, foi necessária a implementação de uma máquina computacional para processar os símbolos que são encriptados e codificar a dinâmica. Tudo isto feito pelo método processar.

Implementar tipos genéricos que determinam a informação que é conhecida após aparecerem, quando instanciados, o seu valor é mapeado no tipo que lhe foi passado.

O comportamento da personagem é representada pelo diagrama que se segue:

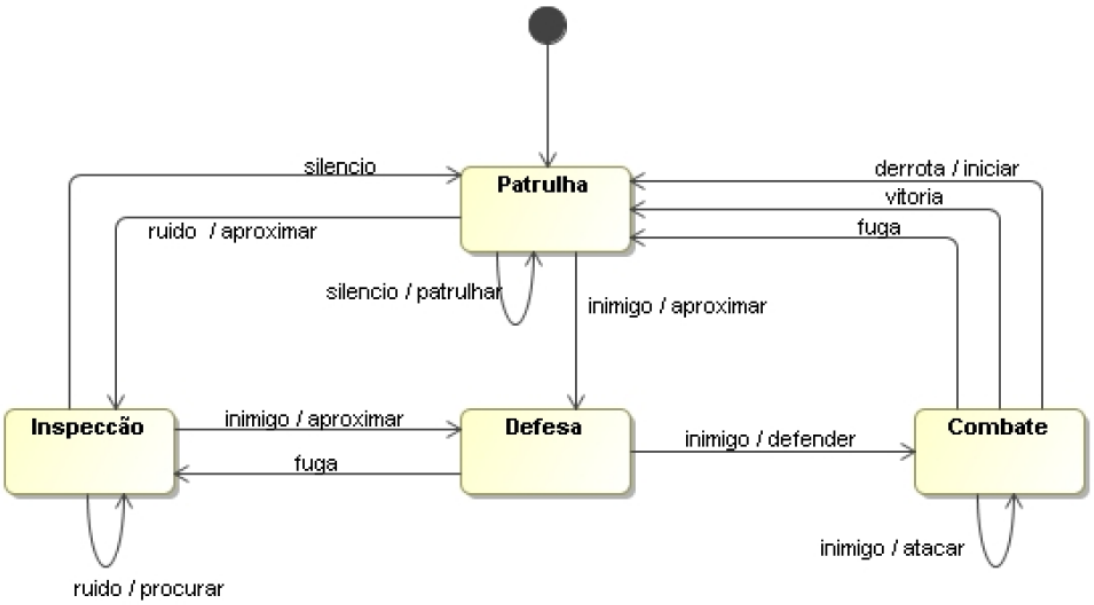

Figura 7 - Dinâmica da personagem

2.2.2 Concretização trabalho prático 2

1. Implementação métodos procura espaços dados

1.1 Implementação conceitos base

1.1.1 Estado

O estado corresponde a uma situação possível de um problema, isto é, define uma situação. Em termos de biblioteca a classe abstrata Estado permite em termos funcionais saber se um estado é o objetivo do problema e qual a identificação desse mesmo estado.

1.1.2 Operador

O operador permite definir transformações, essas transformações são aplicados aos estados para provocar uma transição no mesmo. Na implementação o Operador é uma interface que permite aplicar um operador a um estado e provocar um novo estado e também permite saber o custo entre dois estados.

1.1.3 Problema

O problema define as peças fundamentais do problema tal como o estado inicial, o objetivo e quais os operadores a aplicar. Em termos funcionais a classe Problema tal como foi dito possui um estado que é o estado inicial, um *array* de operadores que foram aplicados ao estado e recebe o estado final ou objetivo.

1.1.4 Solução

A solução é constituída pelo percurso entre o estado inicial e o estado final. Em termos práticos a Solucao é um iterador que irá percorrer os diferentes passos solução para construir o caminho desde o objetivo até ao nó raiz.

2. Métodos de Procura

2.1 Procura Profundidade

Procura que explora primeiro os nós com maior profundidade, isto é, explora primeiros os nós mais recentes. Esta procura depende da ordem de aplicação dos operadores. A implementação da classe ProcuraProf permite iniciar a memória do tipo LIFO(*last in first out*) onde os primeiros nós a ser procurados são os nós mais recentes.

2.2 Procura Largura

Procura que explora primeiramente os nós com menos profundidade, ou seja, são explorados primeiro os nós mais antigos. A expansão para o próximo nível de profundidade só é feita quando todos os nós do nível anterior já tiverem sido expandidos. A implementação da classe ProcuraLarg á semelhança da profundidade inicia a memória neste caso FIFO(*firt in first out*) onde os primeiros nós serão os primeiros a ser explorados.

2.3 Procura Profundidade iterativa

Procura consiste numa procura em profundidade, contudo com algumas limitações. Essas limitações são tais que ele realiza procura em profundidade até ao nível requerido, caso realize toda a procura até esse nível é aumentado o nível de profundidade e assim sucessivamente até ao limite de profundidade máxima. A classe permite definir o incremento sucessivo de passos e a definição da profundidade máxima da procura, possuindo assim o seu próprio método resolver.

2.4 Procura Custo Uniforme

A procura de custo uniforme é uma procura informada dispõe portanto de informação do domínio do problema. A estratégia de procura é explorar primeiro os caminhos com menor custo. A classe ProcuraCustoUnif possui somente o método f que avalia cada nó n gerado retorna o custo do nó. A partir deste é possível construir o melhor caminho tendo por base o custo.

2.5 Procura Sôfrega

Procura que utiliza a função heurística como base da procura. Como utiliza a função heurística possui conhecimento do domínio do problema. Para realizar a procura sôfrega é feita a concretização da classe ProcuraHeur que obriga a calcular a heurística tendo o conhecimento do problema. A característica da procura sôfrega é tal que a procura tenta sempre ir direto para a solução. Este tipo de procura não tem em conta o custo da solução em cada nó. A classe ProcuraSofrega possui apenas o metodo f que retorna a heurística do problema.

2.6 Procura A*

Esta procura é um compromisso entre a procura de custo uniforme e a procura sôfrega. Esta procura faz o custo da minimização do custo global da procura e produz resultados ótimos

e completo. A classe ProcuraAA á semelhança das anteriores possui também somente o método f onde retorna a heurística com o comprimisso entre custo em cada nó e o custo global estimado da heurística.

2.2.3 Implentação trabalho prático 3

1. Agente prospetor

A arquitetura implementada tem por base um agente prospetor. Este agente prospetor é um agente genérico que executa ações genéricas. O algoritmo de funcionamento deste base do agente é receber uma percepção, que é processada e posteriormente é feita uma atuação sobre a mesma. Em termos de implementação é concretizado pela classe AgenteProspetor que possui o método executar que executa o algoritmo referido anteriormente. A classe possui um atributo controlo, o controlo é uma interface que será utilizada para representar as várias concepções através da função processar.

1.2. Agente reativo

1.2.1 Controlo Reativo

A classe ControloReact é um controlo, este controlo possui a função processar que realiza a conexão entre percepções e a ações. A classe utiliza um comportamento, este comportamento é uma interface que faz a abstração entre o controlo e o comportamento do agente reativo feito pela biblioteca ecr.

1.2.2 Biblioteca ecr

O agente reativo desenvolvido é feito de acordo com uma arquitetura de subsunção, sendo um agente sem memória e onde o comportamento é dado pelo forte acoplamento entre percepção-reação.

Na biblioteca existem duas classes que estendem de comportamento, a classe Reaccao

e a classe ComportComp. A classe Reaccao é o comportamento responsável por comportamentos de deteção de estimulo e geração de resposta. Esta classe possui ainda a função ativar do Comportamento que devolve uma resposta a partir de uma percepção, utilizada na classe ControloReat.

A classe ComportComp define a seleção de ações do agente. Esta classe possui uma lista de comportamentos que serão organizados pelo tipo de seleção de ação. Organização esta implementada por um seletores de ações através de uma função abstrata. Visto que a classe também é um comportamento também possui o método ativar, que a partir da lista de comportamentos retirada a resposta de acordo com o tipo de seletor de ação.

A classe de seleção de ação são a Hierarquia e Prioridade que estendem de ComportComp e implementão a organização pela função selecionar resposta. A hierarquia organiza os comportamentos numa hierarquia fixa. A hierarquia é ordenada por ordem de chegada devolvendo sempre a primeira resposta. A prioridade é a seleção de acordo com uma prioridade, prioridade essa obtido pela classe Resposta.

A classe Resposta é uma classe de leitura que permite obter a accao e a prioridade de uma resposta.

1.2.3 Reaccões

Para finalizar é necessário a implementação das reações.

A reação Aproximar realiza a funcionalidade de aproximar de um alvo caso detetado, a classe divide-se em três classes AproximarDir. A aproximação direcional aproxima-se de um alvo de acordo com uma direção específica frente, direita ou esquerda. A classe Aproximar está no topo da hierarquia de reações.

As classes Evitar e Contornar permitem evitar obstáculos, o Evitar serve para obstáculos á frente e o contornar para a esquerda e direita.

A classe Explorar está no fim da hierarquia. Comportamento responsável por explorar o mapa por movimentos aleatórios.

A classe que cria a hierarquia entre todos estas reações é a classe Recolher.

2. Agente deliberativo

2.1 Controlo deliberativo

Para esta concepção, da mesma maneira que o agente reativo, possui uma classe de controlo designada ControloDelib. Esta classe implementa o algoritmo de processo de tomada decisão e ação. Inicialmente é observado o mundo e atualizadas as crenças. De seguida verificasse a necessidade de reconciderar, se sim deliberasse, planeiasse e posteriormente executasse, senão passasse diretamente ao passo executar que devolve uma ação. A classe possui funções privadas que ajudam na realização do algoritmo. Para o funcionamento do algoritmo é necessário um planeador realizado pela interface Planeador e uma representação interna do mundo feito pela classe ModeloMundo. A classe ModeloMundo é a classe responsável pela criação de uma representação interna do mundo necessária para o agente. A classe OperadorMover representa as ações realizáveis pelo agente.

2.2 Planeamento automático

Para criar um planeamento é necessário um planeador de modelo de caixa preta. Este planeador recebe um modelo de planeamento, um estado inicial e os objetivos e constrói um plano de ação. Este comportamento é feito pela realização da interface Planeador.

Como o Planeador necessidade de uma representação do modelo do planeamento surge a interface ModeloPlan que no caso do trabalho será o ModeloMundo.

3. Planeador utilizando pee

Para realizar o planeamento com base em procura de espaço de estados foi utilizada como ferramenta a biblioteca pee. Para isso foi criado um planeador para o pee designado PlanPEE. Esta classe recebe uma procura no espaço de estados. Esta classe cria uma lista com um Plano com os nós que deve percorrer até á solução e um ProblemaPlan para acesso a componentes chave para resolução do problema.

A classe ProblemaPlan é a classe que permite á classe PlanPEE obter o objetivo do problema e a heurística.

4. Planeador utilizando pdm

O planeamento PDM tem por base a propriedade probabilísta de Markov. Para criar um planeador com base em pdm criou-se a classe PDM. O PDM possui dois atributos o gama e o delta max, o gama representa a taxa desconto temporal e o delta max é o valor mínimo da utilidade para continuar a aprender. A classe permite obter as componentes base do pdm como política utilidade entre outros.

O modelo é representado por meio de uma interface designada ModeloPDM que simboliza o modelo do mundo na forma de um PDM. Onde a função S é o conjunto de estados do mundo, o A o conjunto de ações possíveis no estado s pertentencente ao conjunto de estados S, T a probabilidade transição de um estado(s) para um outro estado (s') através de uma ação e R o retorno esperado na transição de (s) para (s') através de uma ação.

A classe PlanPDM cria um plano de ação a partir dos valores obtidos pelo pdm. Sendo um classe semelhante ás classes de planeamento por possuir os mesmos métodos, com a diferença de possuir um atributo gama, delta max, utilidade e política precisas para utilização do PDM.

Para finalizar é criada a classe ModeloPDMPlan que estende de ModeloPDM e possui também um ModeloPDM que implementa as funções S, A, T e R referidas anteriormente e

também os estados e operadores do ModeloPDM no seu interior.

5. Aprendizagem por reforço

Para se fazer a aprendizagem por reforço é necessário uma memória de aprendizagem com o carácter comportamental, onde a aprendizagem é feita a partir da interação com o ambiente através de um um estado, ação e reforço. Assim como o planeador por pdm o reforço também utiliza uma política e utilidade. Para isso primeiramente é criada a AprendRef que aprende a partir de uma memória de aprendizagem e de uma seleção de ação.

Desta forma surgem duas interfaces a MemoriaAprend para representar a memória de aprendizagem e SelAccao para a seleção da ação.

Para concretizar a MemoriaAprend é criada a MemoriaEsparsa. Esta memória serve para atualizar o estado e a ação e permite saber o valor do reforço de um par estado ação.

Para a concretização da seleção de ação é feita pela classe SelAccaoEGreedy esta estratégia de seleção é feita por um valor epsilon que dita a taxa de aprendizagem do agente. A seleção de ações é feita com base no que o agente aprende e numa seleção de ações aleatórias. Como a seleção é E-Greedy o valor que determina quando o valor é aleatório ou não é o epsilon, sendo que a procura greedy tem probabilidade 1-epsilon. As funções representam este mesmo comportamento onde a seleção de ação é feito com base num valor aleatório onde para valores inferiores a epsilon é aletório senão é sôfrega, e os respetivos métodos para procura sôfrega e aleatória.

Para concluir o mecanismo de aprendizagem falta somente uma classe que implemente o método aprender da classe AprendRef. Posto isto a classe AprendQ implementa a função aprender com o algoritmo Q-Learning.

Desta maneira já temos o mecanismo de aprendizagem todo concluído, falta somente juntar todas as componentes numa única que é a classe MecAprend que cria uma memória

esparsa, seleção-ação E-Greedy e uma aprendizagem por Q-Learning. Sendo então abstração à implementação das funções e espondo apenas as funções de contrato com as interfaces.

Por fim falta o controlo para o agente prospetor realizar o comportamento por reforço. A classe ControloAprendRef sendo o controlo implementa a função processar e utiliza a classe MecAprend para aprender e selecionar uma ação. A classe também implementa um passo importante no algoritmo que é o geramento de reforço.

3. Conclusões

Nos presentes trabalhos práticos foi possível de uma forma prática aplicar os conhecimentos teóricos dados durante o decorrer das aulas.

Esta aplicação prática permitiu aprender de uma forma mais consistente a matéria, vista que para a concretização do código é necessário uma base forte dos conceitos teóricos.

Durante o decorrer dos trabalhos práticos foram aprendidos os seguintes conceitos:

- O significado do paradigma simbólico e a utilidade da associação entre símbolos e o seu significado.
- Foi aprendido o significado do que é um agente reativo, quais as suas vantagens e desvatagens de utilização.
- O que significa o processo de procura em espaços de estados e quais as diferentes procuras existentes.
- As características de cada procura bem como as suas vantagens e desvantagens comparado a outras procuras.
- Qual é o significado de um agente deliberativo e a forma de atuação sobre um plano.
- Processos de decisão sequêncial com a propriedade de Markov.
- Como é feita a representação do mundo na forma de um PDM.
- Conceitos por de trás da aprendizagem por reforço.

Foi possível de verificar como fazer código de forma incremental e modelar facilita a implentação posterior do mesmo. Tendo um planeamento prévio com modelos é possível criar código de forma simples com uma baixa complexidade. Desta maneira torna-se simples de implementar novos comportamentos.

4. Bibliografia

[Textos Apoio] - Luís Morgado, IASA,ISEL, 2019-2020

[AIMA-1] - 2011