ISEL

DEETC – Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

MEIM - Mestrado Engenharia informática e multimédia

Inteligência Artificial e Sistemas Cognitivos

Projeto

Trabalho realizado por:
Duarte Domingues Nº45140

Docente:
Luís Morgado

Data: 12/06/2022

Índice

Índice de figuras

Introdução

Neste projeto foram realizados diferentes modelos e algoritmos de inteligência artificial e sistemas cognitivos.

O projeto dividiu-se nas seguintes etapas:

- Realização de um protótipo com base numa plataforma de redes neuronais para resolução de problemas e a aplicação de redes neuronais para um problema de classificação de imagens.
- Realização de um sistema de aprendizagem por reforço, com diferentes métodos de aprendizagem.
- Realização de um sistema de raciocínio automático para planeamento com diferentes métodos e algoritmos de otimização.

Ao longo do projeto foi definida a arquitetura da solução realizada para resolver os diversos problemas utilizando práticas e ferramentas de engenharia de software. Os sistemas realizados seguiram diferentes métricas de arquitetura. Teve-se como objetivo realizar sistemas de alta simplicidade, adaptabilidade, coesão e baixo nível de acoplamento.
Inicialmente começou-se por estudar e analisar diferentes conceitos de inteligência artificial e acabou-se por abordar um tema relevante de inteligência artificial, a criação de arte realista a partir de descrições de linguagem natural.

2. Desenvolvimento

Inteligência artificial é o campo que estuda a síntese e analise de agentes computacionais que agem de forma inteligente.

Agente

Estes agentes representam algo que age num ambiente, sendo este considerado inteligente quando realiza uma ação com base no seu ambiente, experiência e objetivos. O agente inteligente é flexível para ambientes e objetivos em mudança aprendendo a partir da sua experiência.

Sistemas cognitivos

O sistema cognitivo é um sistema capaz de utilizar a informação do ambiente, envolvente, de forma autónoma, para tomar decisões. Sistemas cognitivos conseguem perceber, aprender e realizar decisões. Estes sistemas estão ligados com a aplicação e compreensão da forma como humanos atuam perante situações complexas.
Dois conceitos essências para entender estes sistemas são cognição e racionalidade.

Cognição

No ser humano o conceito de cognição baseia-se na capacidade de conseguir processar informação e transformar esta informação em conhecimento, através do pensamento das experiências e sentidos.
No sistema computacional a cognição foca-se na capacidade de adquirir conhecimento sendo este conhecimento adquirido a partir de termos de representação e processamento de informação.

Racionalidade

Em relação ao conceito de racionalidade um sistema é racional se realiza a ação correta consoante o conhecimento que possui. Entretanto existem limites computacionais de recursos que podem impedir o agente de cumprir a ação correta. Estas limitações de computação determinam se o agente tem:

- Racionalidade limitada: O agente decide na melhor ação que pode encontrar dados os seus limites computacionais;

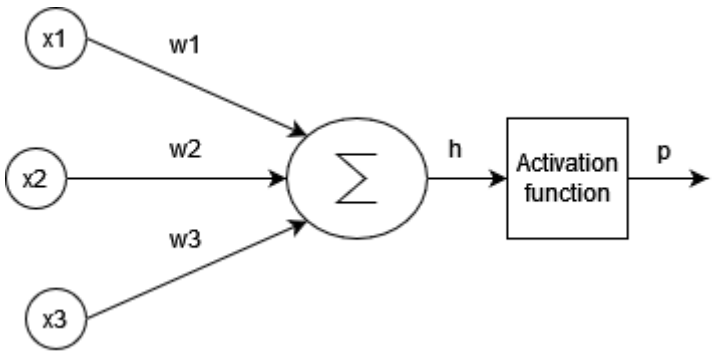
- Racionalidade ilimitada: O agente decide na melhor ação sem ter em consideração os seus recursos computacionais limitados.

Aprendizagem

Outro termo crucial para entender como desenvolver sistemas de inteligência artificial é a aprendizagem. A aprendizagem baseia-se na melhoria do desempenho de uma tarefa consoante a experiência.

Redes neuronais artificiais

Redes neuronais artificiais são um paradigma popular de aprendizagem automática, destacam-se por ser baseadas nos neurónios do cérebro. Em biologia os neurónios são ligados a outros neurónios a partir de dendrites e axónios e o ponto de ligação do neurónio às dendrites de outro neurónio é chamado sinapse. As forças das conexões das sinapses diferem consoante estímulo esterno, esta variabilidade de cada sinapse é crucial para a aprendizagem. Este mecanismo é simulado artificialmente nas redes neuronais.
Nas redes neuronais cada entrada a um neurónio é escalada com um peso, sendo a soma ponderada da soma das entradas aplicada uma função de ativação, o que é definido como o modelo do neurónio artificial o perceptrão, como pode ser observado na figura 1.



Redes neuronais multicamada

O perceptão tem muitas limitações. Estas limitação são ultrapassadas por redes neuronais multicamada, toda esta rede é composta por neurónios. Estas redes contêm uma camada de entrada com as entradas, uma ou mais camadas escondidas e uma camada de saída com o número de saídas corresponde ao número de classes.

Na rede neuronal artificial para cada instância de treino, x[n] é passada pela rede de forma a produzir uma saída, propagando os valores dos neurónios de entrada para os neurónios de saída, utilizando os pesos como parâmetros intermediários. A aprendizagem é realizada progressivamente através da alteração dos pesos.

Fases do modelo

Forward Propagation

Cada instância de treino, x[n] é passada pela network de forma a produzir um output. yh é o output predito dada pela função de ativação tangente hiperbólica no neutrão de output. No início são inicializados os pesos e as *biases* do sistema.

Cálculo da função de erro

A função de erro é uma estimativa do desempenho da network. O erro deve diminuir ao longo das EPOCHS até chegar a um valor baixo estável.

O erro é calculado a partir de diferentes funções possíveis como por exemplo: *Mean Squared Error* (MSE).

Retro propagação

Nesta fase o erro encontrado no modelo é utilizado para ajustar os pesos e *biases* que diminuem o valor da função de erro.
Começa-se com yh e tenta-se encontrar a taxa de mudança de erro em relação ao Wo. O algoritmo de retro propagação funciona da camada de saída para a camada de entrada, propagando os gradientes de erro.

Adaptação dos pesos

Os pesos (W) são adaptados por fim utilizando a fórmula da descida de gradiente.

Descida de gradiente

A descida de gradiente é um método iterativo para encontrar o mínimo de uma função. Este método funciona como andar numa rua em descida e dar sempre um passo na direção que desce mais. Este método de otimização realiza passos em cada direção proporcional à negativa da derivada parcial nessa direção.
Descida de gradiente é utilizada para a aprendizagem de parâmetros. Começa-se por introduzir os valores dos parâmetros iniciais como os pesos e as *biases* e de seguida o algoritmo utiliza cálculos para iterativamente ajustar os valores para que estes minimizem a função de custo.

O método é baseado numa função convexa e ajusta os seus parâmetros iterativamente para minimizar uma função para o seu mínimo local. Por exemplo, na seguinte imagem tendo os parâmetros (w e b) nos eixos x,y e a função de custo no eixo z. A descida de gradiente é utilizada para minimizar a função de custo e chegar ao mínimo local ajustado os parâmetros (w e b).

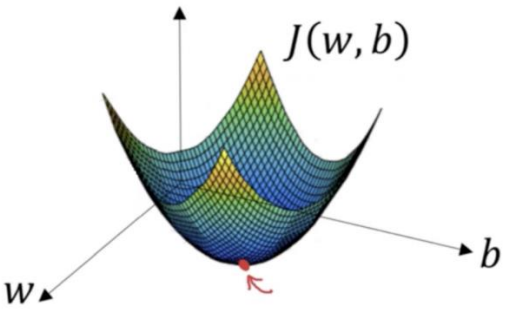


Figura 1 - descida de gradiente

Taxa de Aprendizagem

A taxa de aprendizagem vai ser responsável por controlar o quão rápido o modelo vai aprender, definindo o tamanho dos passos da descida de gradiente. Na descida de gradiente o modelo decide o quanto tem de mexer os pesos através do parâmetro da taxa de aprendizagem.

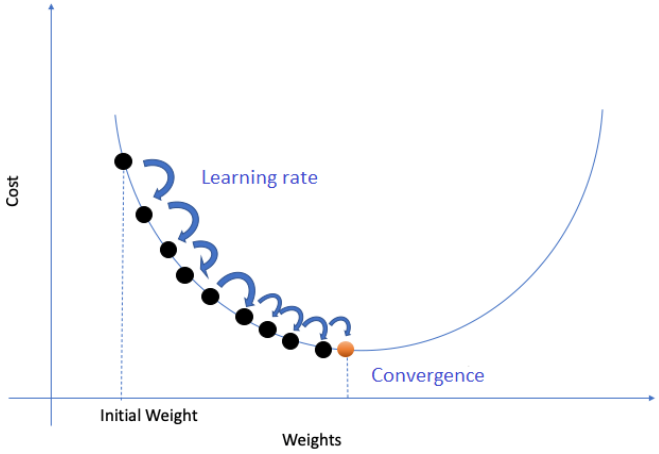


Figura 2 - taxa de aprendizaagem

Quanto maior for a taxa de aprendizagem, mais rápido irá o modelo convergir, porém tem de se ter cuidado a escolher o valor para a taxa de aprendizagem. Se os passos forem demasiado grandes, é possível que o sistema não chegue a mínimo local nenhum porque oscila sempre para a frente e para trás entre a função convexa da descida de gradiente. Se os passos forem demasiados pequenos a convergência é demasiado lenta.

Termo de momento

O termo de momento é aplicado na descida de gradiente ditando o quão rapida ou forte é a descida/delta de cada iteração para encontrar o minimo de uma função. Desta forma quanto maior for o termo de momento mais fácil é para o modelo para atingir convergência e evitar minimos locais, pois cada iteração tem inércia.

Descida de gradiente normal depende apenas da taxa de aprendizagem com o gradiente dos pesos.

$$\Delta w_i = -\eta \frac{\partial L}{\partial w_i}$$

Descida de gradiente com momento adiciona parte dos delta dos pesos das iterações anteriores, o que é responsavél pela adição de inércia à rede.

$$\Delta w_i^t = \alpha \Delta w_i^{t-1} - \eta \frac{\partial L}{\partial w_i}$$

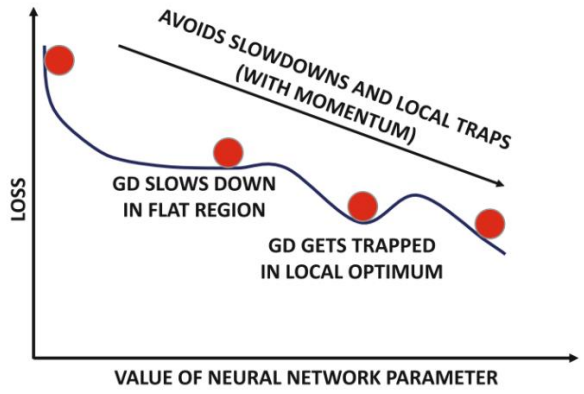


Figura 3 - termo de momento

Parte 1

Problema XOR

Na primeira etapa do projeto foi necessário através do desenvolvimento de redes neuronais resolver o problema da aprendizagem da função lógica XOR. A função lógica XOR é operação lógica entre dois operandos que resulta num valor positivo apenas se os dois operandos forem diferentes.
No projeto resolveu-se o problema com uma classificação binária ou bipolar e foram notadas as diferenças nos resultados.

x0	x1	y
0	0	0
0	1	1
1	0	1
1	1	0

x0	x1	y
-1	-1	0
-1	1	1
1	-1	1
1	1	0

Um modelo de perceptrão simples como apenas entradas e uma saída não vai conseguir resolver o problema XOR. Isto deve-se ao facto do XOR não ser um problema linearmente separável, como se pode observar na seguinte figura, sendo necessário duas barreiras para classificar corretamente dados num padrão XOR.

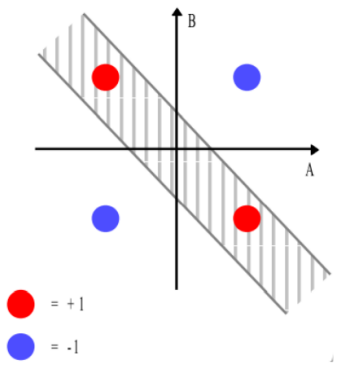


Figura 4 - problema XOR

De forma a resolver o problema XOR vai ser necessário uma rede com entradas, uma camada escondida com pelo menos dois neurónios e com uma saída.

Como critério de convergência definiu-se que a rede tem de ter sempre uma *loss* superior a 0.1.

A rede neuronal realizada tem as seguintes características fixas:

- Função de ativação: '*relu'*, função linear que é definida pela fórmula f(x) = max(0,x)
- Solver para otimização dos pesos: '*sgd'*, implementa descida de gradiente estocástica
- Número máximo de iterações até convergência: 100000

Através de diferentes testes foi estudado o desempenho da rede com diferentes valores do termo de momento, taxa de aprendizagem e a apresentação das amostras de treino com ordem fixa ou aleatória.

Resultados obtidos

Efeito da taxa de aprendizagem (*alpha*=0)

Na seguinte tabela pode se observar o efeito da taxa de aprendizagem no número de iterações necessárias até ser atingida convergência na rede.

Execução	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	7466	3100	2067	1480	805
2	5470	2349	1653	1154	805
3	5546	3023	1663	1162	1052
4	7334	3011	1665	1162	1071
5	5722	2375	2110	1158	811
6	5804	2386	2097	1484	804
7	5542	2395	1657	1491	1058
8	7306	2365	2151	1496	811
9	5548	2425	2095	1499	801
10	5473	2408	1648	1486	1068
Média:	6121.1	2583.7	1880.6	1357.2	908.6

Figura 5 - efeito da taxa de aprendizagem

Quanto maior for a taxa de aprendizagem mais rapidamente a rede converge como seria esperado. Entretanto se o valor da taxa de aprendizagem continuasse a aumentar muito mais seria possível que o sistema não chegasse a mínimo local nenhum, aumentado o número de iterações necessárias para convergência. Vai ser agora estudada a adição de termo de momento.

Efeito da introdução de um termo de momento

α = 0.5

Execução	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	3868	2082	1154	1063	775
2	3853	1640	1184	1056	767
3	4191	1654	1167	795	773
4	5034	1678	1476	1045	785
5	3878	2104	1473	799	556
6	3871	1631	1482	803	555
7	3913	1639	1147	795	568
8	3954	2094	1510	810	551
9	3940	1673	1145	813	551
10	3810	2097	1474	1071	557
Média:	4031.2	1829.2	1321.2	905	643.8

α = 1

Coluna1	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	87	53	36	33	23
2	77	59	39	34	25
3	90	50	40	29	24
4	80	49	36	38	28
5	84	48	35	33	28
6	86	60	42	33	28
7	84	43	36	31	23
8	87	55	35	36	26
9	91	44	41	29	23
10	92	44	36	30	23
Média:	85.8	50.5	37.6	32.6	25.1

Figura 6 - Efeito introdução de termo de momento

Como referido anteriormente, com a adição de momento a rede converge com base nos deltas das iterações do passado, adicionando inércia. A partir da observação dos resultados é possível observar que consoante maior for o valor de α mais rápido é a convergência da rede e evitar mínimos locais.

Efeito da introdução de ordem de apresentação aleatória

$\alpha = 1$

Coluna1	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	108	39	42	31	24
2	108	43	35	29	22
3	127	41	44	34	24
4	95	44	46	27	28
5	87	57	38	28	24
6	124	51	34	27	22
7	91	51	49	28	23
8	109	52	33	29	25
9	85	46	33	34	23
10	101	66	36	26	25
Média:	103.5	49	39	29.3	24

Figura 7 - Efeito de introdução de ordem aleatória de apresentação

Pretende-se agora analisar a diferença entre os dados de treino serem aleatórios ou não. A ordem de apresentação aleatória tem a vantagem de reduzir variância nos dados e certificar que o modelo entra menos em sobre aprendizagem.

A partir da observação dos resultados é possível notar que a aleatoriedade nos dados de entrada não causou resultados muito diferentes que os normais. É de notar que possivelmente não se observou grande diferença entre a ordem de apresentação aleatória ou não aleatória devido ao conjunto de entrada ser muito pequeno.

Efeito da codificação binária

α = 1

Coluna1	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	120	62	47	35	29
2	120	74	50	36	28
3	190	58	51	39	36
4	128	58	57	31	38
5	154	48	46	42	34
6	141	70	47	56	36
7	123	51	43	34	30
8	178	57	43	39	26
9	123	71	61	39	25
10	78	95	44	41	33
Média:	135.5	64.4	48.9	39.2	31.5

Figura 8 - Efeito da codificação binária

Nesta etapa foi analisada a diferença nos resultados entre codificação binária e bipolar. A partir dos resultados pode-se observar que a codificação binária ter tido um desempenho ligeiramente pior do que a bipolar. Os dados bipolares podem ter tido melhor resultados pois usando dados binários um nó de entrada pode ter valor 0, então o seu peso é delta 0. Usando representação bipolar este problema é resolvido porque os nós nuca têm valor de 0.

Regiões de decisão

Na seguinte figura é observada a região de decisão de uma rede sem introdução de um termo de momento, contra uma rede com termo de momento igual a 1.

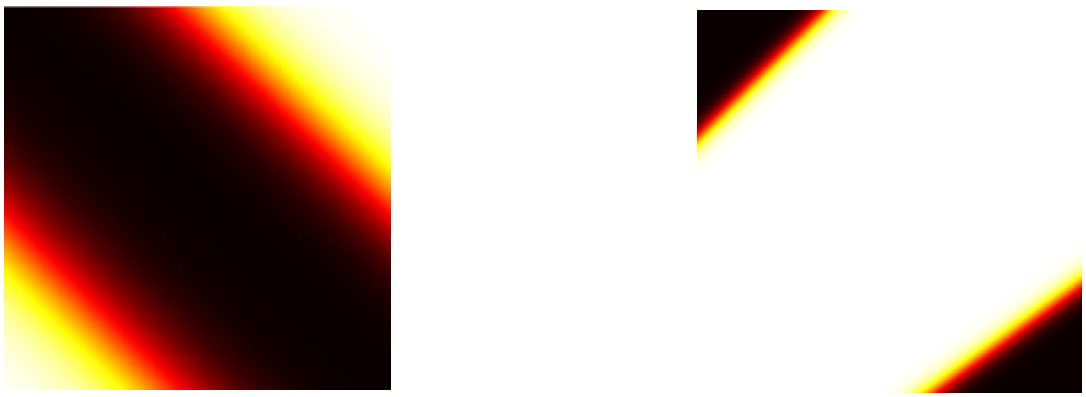


Figura 9 - Regiões de decisão

É possível observar que com a adição de termo de momento, a região de decisão está dividida de forma mais evidente.

Aprendizagem de padrões

De seguida foi desenvolvida uma rede neuronal para a aprendizagem dos padrões das imagens seguintes:

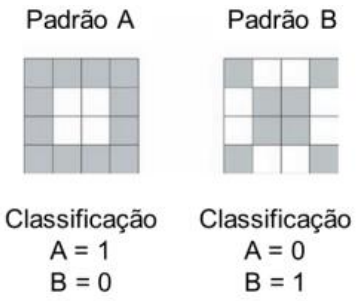


Figura 10 - Padrões XOR

Os padrões pertencem a duas classes:

- Padrão A: padrão retangular
- Padrão B: padrão axadrezado

Cada padrão é 4x4, sendo cada "pixel" da imagem 0 ou 1. Devido aos padrões terem esta dimensão a rede desenvolvida irá ter 16 neurónios na camada de entrada. A saída do problema tem dois valores um para a classe A e outro para a classe B, logo irão ser necessários dois neurónios na camada de saída.
Na camada escondida foram utilizados dois neurónios, poderia ter sido utilizado apenas um, mas a utilização de dois neurónios teve melhor desempenho.

O conjunto de dados foi dividido em treino e teste. Para o conjunto de treino foram utilizados 9 exemplos e para o teste três exemplos.

Como critério de convergência definiu-se que a rede tem de ter sempre uma loss superior a 0.1.
A rede criada tem os seguintes parâmetros:

- Solver para otimização dos pesos: 'sgd', implementa descida de gradiente estocástica;
- Número máximo de iterações até convergência: 100000;

- Termo de momento (*alpha*): 1;
- Taxa de aprendizagem (r): 0.05;
- Foram testadas diferentes funções de ativação: '*identity'*, '*logistic'*, '*tanh'*, '*relu'*.

A rede neuronal elaborada teve a seguinte configuração:

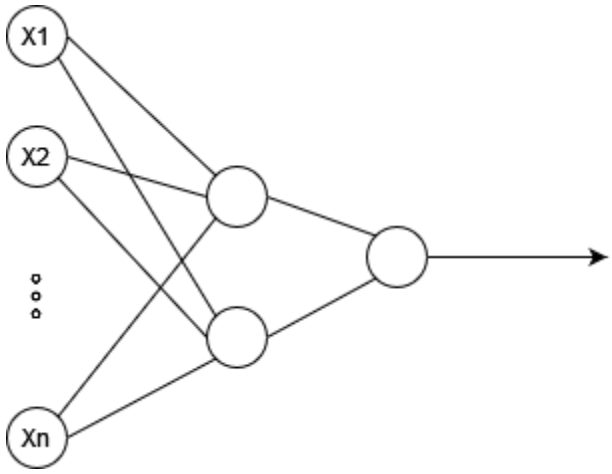


Figura 11 - Rede neuronal

Resultados obtidos

Os resultados obtidos para o conjunto de teste com as diferentes funções de ativação foram os seguintes:

- A precisão dos resultados independentemente da função de ativação foram sempre 1.

As iterações necessárias para a convergência para cada rede consoante a função de ativação foram os seguintes:

Identity	logistic	tanh	relu
iterações	iterações	iterações	iterações
30	112	75	88
32	116	73	66
41	120	62	75
37	123	70	73
36	116	72	83

Figura 12 - Resultados obtidos

Classificação de imagens

Nesta etapa vai ser realizada a classificação de imagens do conjunto de dados Fashion-MNIST. Este conjunto de dados consiste num conjunto de treino de 60000 imagens e um conjunto de teste de 10000 imagens de roupa.

Cada imagem tem o formato de 28x28 pixéis. Cada pixel da imagem tem um valor de 0 a 255, indicando o nível de cinzento do pixel. Na seguinte figura pode ser observado o exemplo de uma imagem no conjunto de dados.

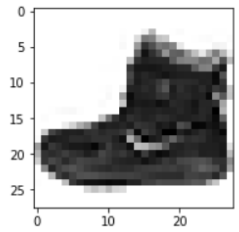


Figura 13 - Exemplo de imagem

As imagens estão divididas em 10 classes, na seguinte figura pode ser observado 10 exemplos de imagens para cada classe.

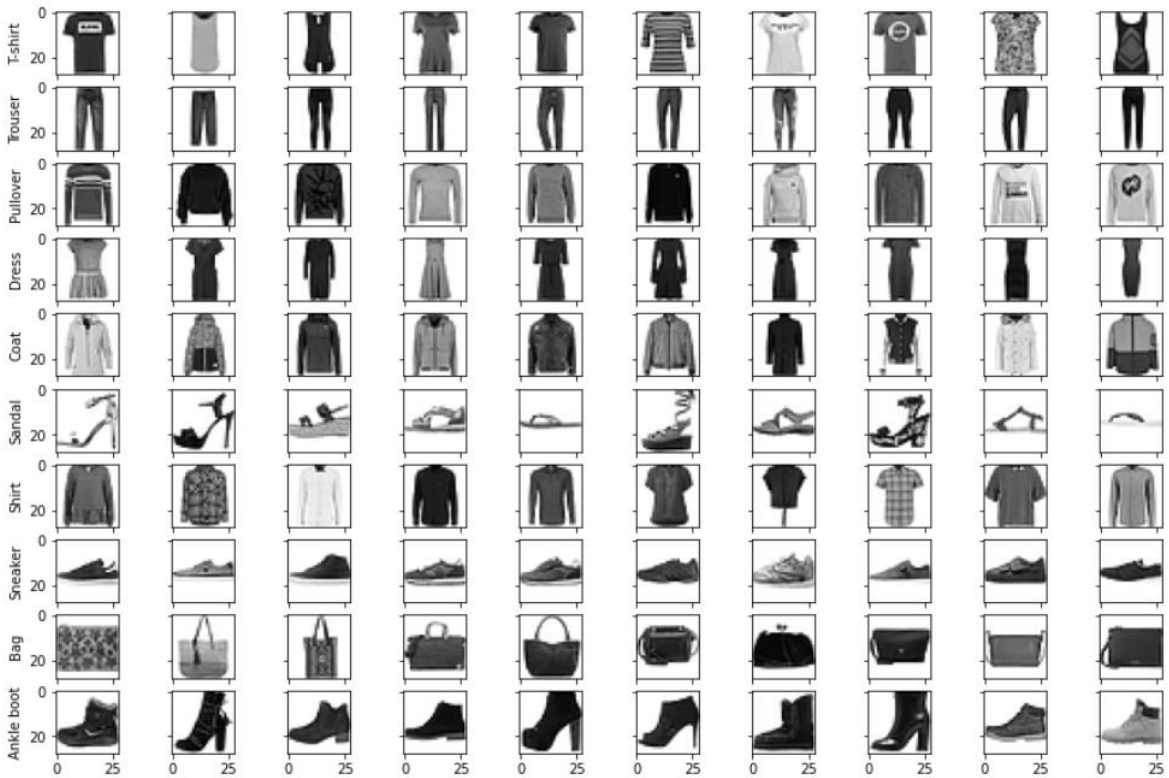


Figura 14 - Imagens divididas por classe

Pré-processamento dos dados

Inicialmente foi necessário pré-processar os dados para um formato que possa ser utilizado para uma rede neuronal. Começou-se a realizar um *reshape* de (28x28) para (784) e conversão dos dados para *float*. De seguida os dados foram baralhados. Por fim os dados foram categorizados *com one-hot-encoding*, que é uma representação de variáveis categóricas como vetores binários.

Treino da rede neuronal

A rede sequencial é composta por um conjunto de camadas emparelhadas.

A rede neuronal treinada é uma rede sequencial com as seguintes características:

- 1000 unidades na camada de entrada
- 5 camadas escondidas (primeira com 1000 unidades e outras com 505)
- camada output com 10 unidades
- Função de ativação das camadas escondidas - *ReLU*
- Função de ativação para a camada de saída - *softmax*
- Otimização – Nadam

Resultados obtidos

De seguida apresentam-se os resultados obtidos:

Precisão de treino: 0.96

Precisão de teste: 0.88

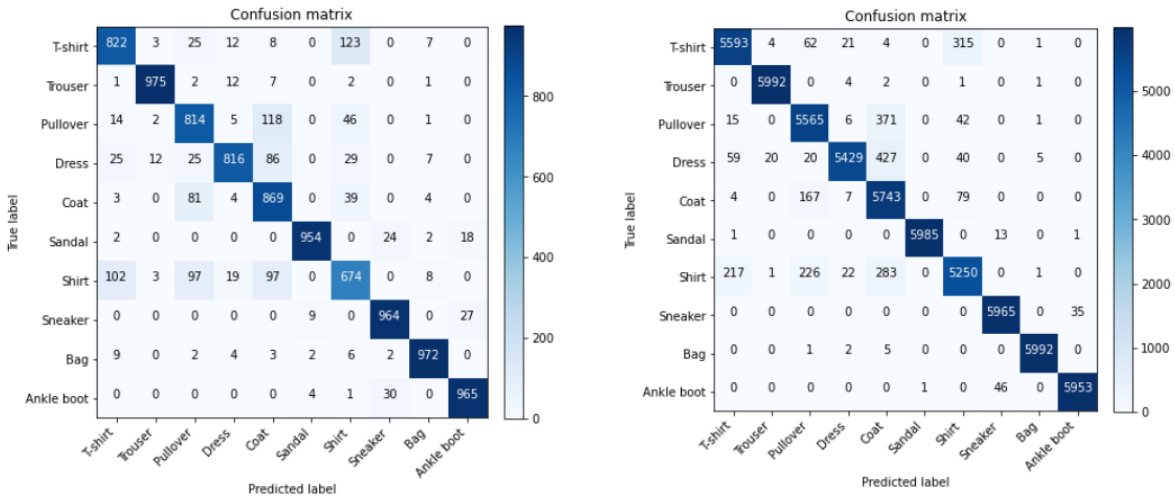


Figura 15 - Matrizes de confusão

Parte 2

Aprendizagem por reforço

Aprendizagem por reforço baseia-se na aprendizagem do agente a partir da interação com o ambiente. O agente recebe uma recompensa por cada ação realizada num estado. Ao longo do tempo o agente aprende a maximizar as recompensas positivas, ajustando as suas ações.

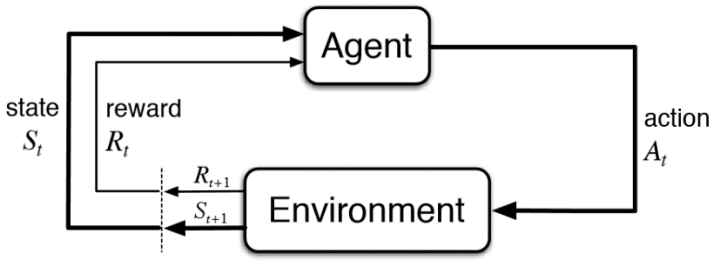


Figura 16 - Diagrama de aprendizagem por reforço

Este tipo de aprendizagem baseia-se numa política comportamental, uma estratégia de ação que define para cada estado que ação deve ser realizada. Esta política pode ser determinística ou não determinística.

- **Política determinística** um estado define que uma ação especifica seja obrigatoriamente realizada.
- **Política não determinista** são definidas diferentes ações possíveis de acontecer num estado, consoante as suas probabilidades.

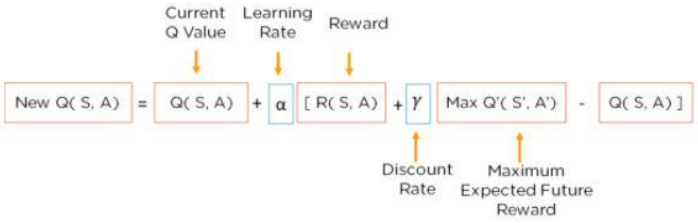
Para escolher uma ação existe o dilema entre escolher uma ação que beneficie a **exploração** do ambiente e uma ação valorize o **aproveitamento** que leve à melhor recompensa conforme a aprendizagem (ação sôfrega). Para convergir para um valor ideal tem de se explorar e aproveitar, nunca podendo parar de explorar.

A aprendizagem por reforço tem como problemas, possíveis tempos elevados para ocorrer convergência e complexidade do espaço de estados. Soluções possíveis é a utilização de memórias episódicas, utilização de modelos do mundo e arquiteturas híbridas.

O processo de aprendizagem é definido por memórias de pares estado-ação com um valor Q associado a cada par estado-ação. Sendo a aprendizagem realizada incrementalmente consoante a experiência aumenta.

Q-Learning

Com este método pretende-se maximizar a recompensa de longo prazo. Este método utiliza um função Q(s,a), para determinar o quão boa é uma ação tomada num estado em particular. A ação é escolhida de acordo com o estado com base numa política derivada de Q. De seguida é atualizada a função Q com base na recompensa e transição de estado realizada, a partir da seguinte equação:



A equação soma o valor de Q(s,a) atual mais a recompensa obtida com a máxima recompensa futura esperada menos a estimativa anterior de Q(s,a). A taxa de aprendizagem determina o quão rápido o modelo vai aprender. O novo valor de Q(s,a) é guardado numa memória de aprendizagem com formato de tabela. Cada linha e coluna da tabela são os pares- estado ação e os valores são os valores Q. Esta tabela permite encontrar a melhor ação para cada estado no ambiente. Este algoritmo é repetido por cada episódio. Um episódio acaba quando o agente chega a um estado terminal.

Este algoritmo *é off-policy*, ou seja, segue a utilização de políticas de seleção de ação distintas para comportamento e para propagação de valor.

Q-Learning com memória episódica

Este método é uma variação do *Q-Learning* tradicional. A atualização da função Q neste método não se baseia apenas nas últimas transações de estado e recompensas, tendo também uma memória de experiência. Esta memória armazena as transições de estado e recompensas e utiliza-as para atualizar a função Q.

Este método permite que o agente aprenda a partir de experiências passadas, e utilize estas experiências para auxiliar a sua a aprendizagem.

A memória de experiência pode ter dimensões muito elevadas, é necessário então utilizar métodos para redução de dimensão da memória. No projeto foi definido uma dimensão máxima para a memória, sendo o elemento mais antigo da memória removido caso esta dimensão seja alcançada.

Dyna-Q

O algoritmo *Dyna-Q* baseia-se no algoritmo *Q-Learning* com a adição de um **modelo de mundo** e **simulação de experiências**. O modelo do mundo é a representação das características relevantes do domínio do problema.

O algoritmo *Dyna-Q* aprende a partir de experiências reais como no *Q-Learning*, atualizando o valor Q para os pares estado-ação. Entretanto, o algoritmo *Dyna-Q* possui um modelo, que permite ao agente simular ações e futuras recompensas baseadas no modelo do mundo e atualizar a política para interagir com o ambiente com base nestas simulações. Isto permite a aprendizagem indireta a partir de um modelo interno, tendo como vantagem tornar o processo de aprendizagem mais rápido. Pode ser observado o funcionamento do método deste método de aprendizagem na seguinte figura:

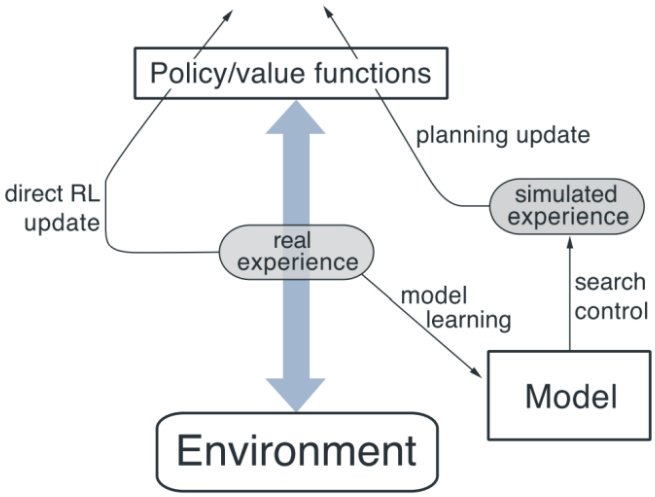


Figura 17 - Diagrama Dyna-Q

Implementação

Na aprendizagem por reforço foram implementados os seguintes métodos de aprendizagem:

- *Q-Learning*
- *Q-Learning* com memória episódica
- *Dyna-Q*

Estrutura

A implementação seguiu a seguinte estrutura:

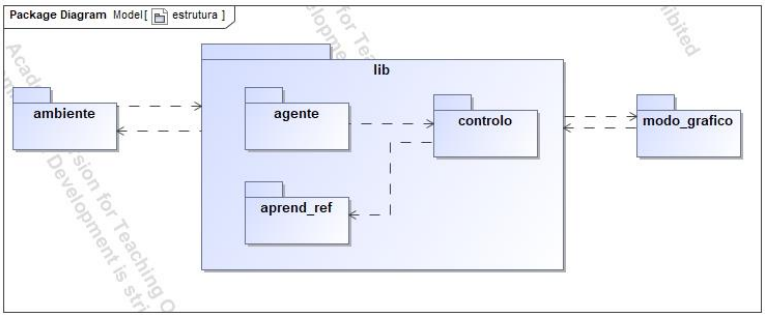


Figura 18 - Estrutura

O *package* ambiente contém as classes responsáveis por criar o ambiente e gerir o ambiente.

O *pakage* lib contém a biblioteca de aprendizagem por reforço que contém os métodos de aprendizagem e todas as classes e métodos necessários.

O package agente contém a classe e métodos do agente. O agente perceciona o mundo através de **perceções**, a partir das perceções e do **controlo** são definidas as ações que o agente deve tomar no **ambiente**.

O package controlo contém a classe e métodos do controlo, é aqui que são tomadas as decisões sobre as ações que o agente deve realizar.

O package modo gráfico contém as classes e métodos necessários para a visualização gráfica do comportamento do agente. O modo gráfico recebe o ambiente e o agente.

Mecanismos

Inicialmente começou-se por definir o package lib, realizando a biblioteca de aprendizagem por reforço.

Foram definidos os mecanismos da biblioteca. Os diferentes métodos de aprendizagem por reforço estendem da classe abstrata "`AprendRef`", contém o método "aprender" responsável por realizar o processo de aprendizagem, atualizando a memória de aprendizagem definida na classe "`MemoriaAprend`" e selecionando uma ação a partir da classe "`SelAccao`". A classe "`MecanismoAprendRef`" é o mecanismo principal da aprendizagem por reforço, onde são passadas as diferentes ações possíveis do sistema.

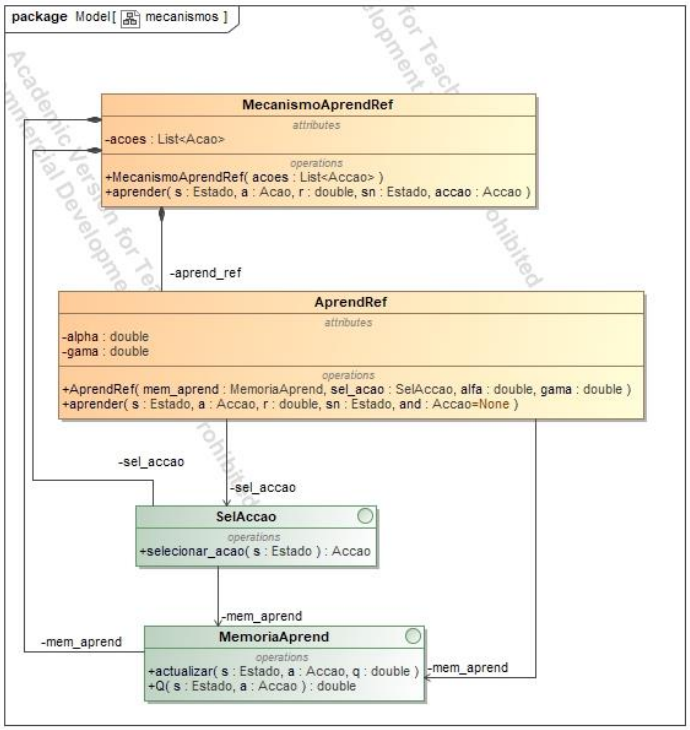
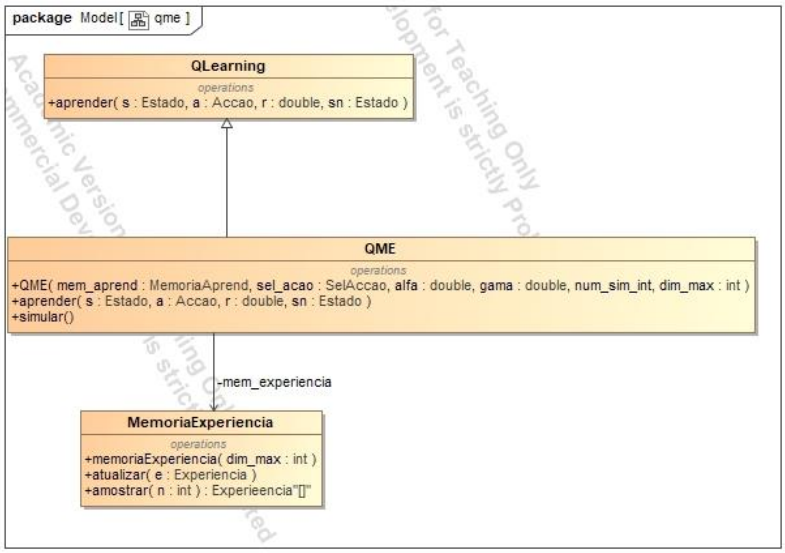


Figura 19 - Mecanismos

Métodos de aprendizagem

Como se pode observar na seguinte figura, *QLearning* com memória episódica é uma generalização de *QLearning*, contendo uma memória de experiência e também um método simular adicional. Num determinado instante t, a experiência do agente é definida por um *tuple*: e = (st , a , r, sn). O processo de seleção de amostras é realizando aleatoriamente, consoante um valor n que define o número de amostras.



A classe "`DynaQ`" é uma generalização de *QLearning*, tendo um modelo definido pela classe "`ModeloTR`". No método de aprendizagem *Dyna-Q* a aprendizagem ocorre como no *QLearning* tendo como diferença no final da aprendizagem, a atualização do modeloTR e a realização da simulação.

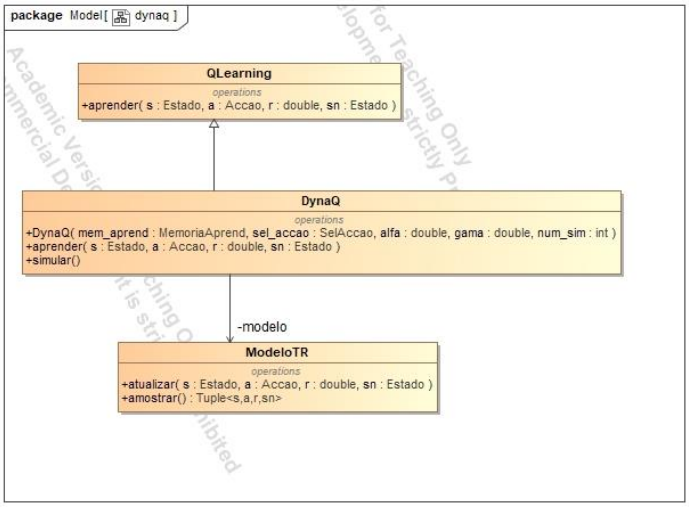


Figura 20 - Dyna-Q

Ambiente

De modo a criar o ambiente foi criado um método que permite ler o ficheiro de texto que contém a informação do ambiente e retornar uma instância da classe "`Ambiente`". A classe Ambiente tem diferentes atributos que representam a informação do ambiente, respetivamente:

- Largura e Altura
- Ponto de origem
- Direção inicial do agente
- Alvo
- Obstáculos

Interação com o ambiente

De forma ao agente conseguir percecionar o seu ambiente foi criada a classe "`Percecao`" e para o agente se poder mover no ambiente foi criada a classe "`Mover`".
A classe "`Percecao`" permite obter a posição do agente a partir do ambiente, obter os obstáculos, verificar se o agente atingiu um obstáculo, atingiu um alvo e verificar o custo do movimento. Movimentos diagonais têm o dobro do custo de movimentos normais

A classe "`Mover`" recebe uma ação e verifica se a realização da ação resulta na colisão com um obstáculo, caso ocorra colisão a posição do agente no ambiente não é alterada.

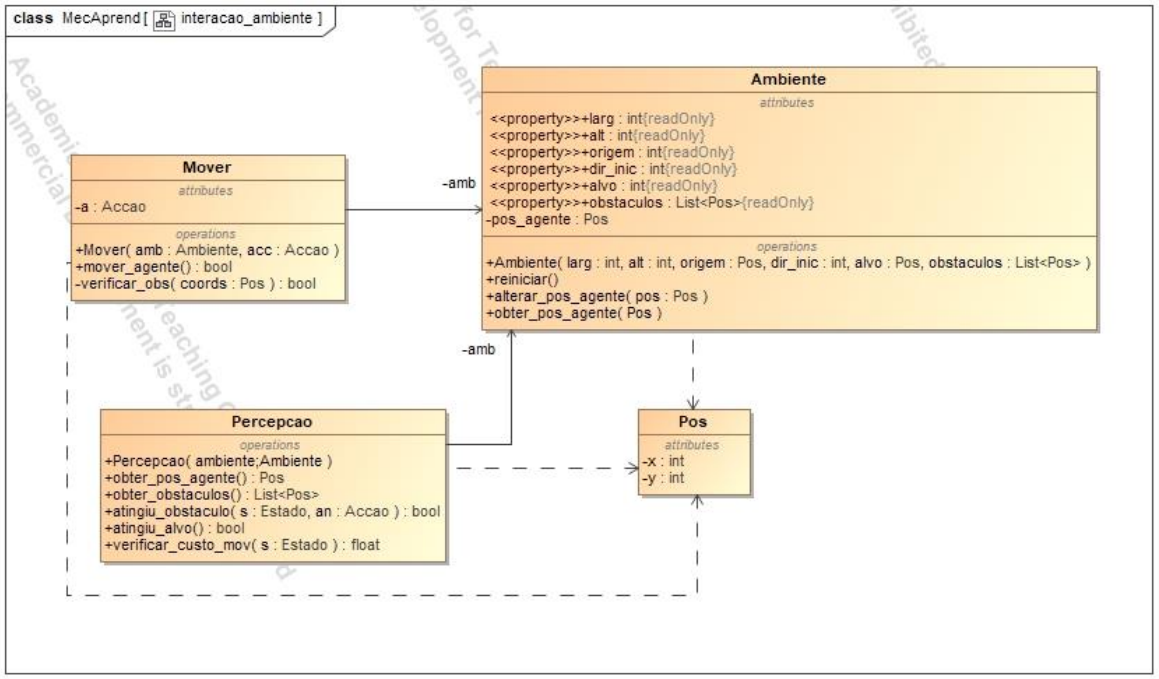


Figura 21- Interação com o ambiente

Controlo

O controlo é implementado na classe "`ControloAprendRef`", no método "`processar`" desta classe, através de uma perceção é selecionada a melhor ação para o agente realizar, através do mecanismo de aprendizagem utilizado. De seguida é obtido a recompensa para esta ação e é chamado o método de aprender consoante o mecanismo de aprendizagem.

Foi definido que a recompensa máxima tem o valor 100, ao agente atingir o alvo. A recompensa mínima é -100 para quando o agente atinge um obstáculo. Para cada movimento está associado um custo de forma a minimizar o número de passos necessários para a solução ser atingida, o custo do agente realizar um único movimento foi definido como 0.1.

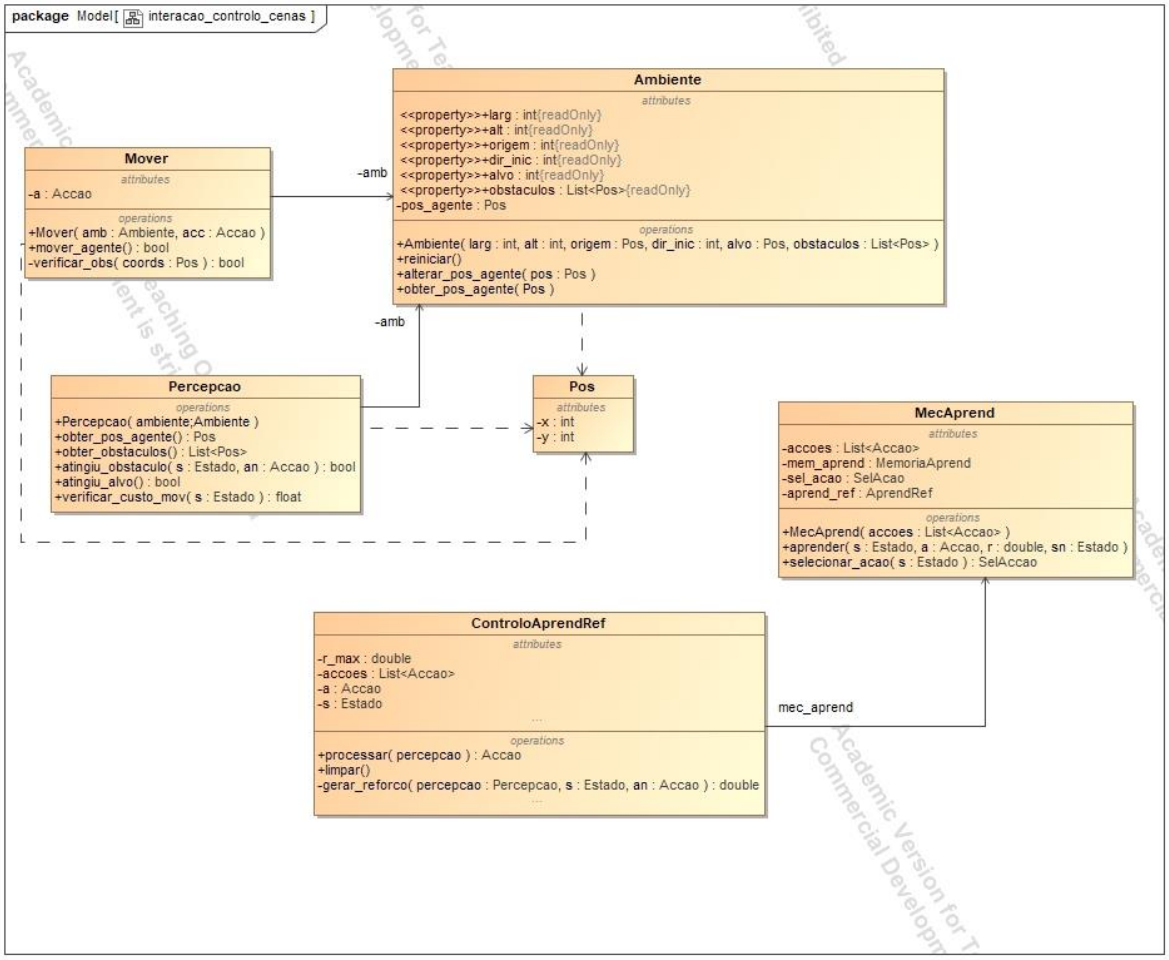


Figura 22 - Controlo

Agente

O agente foi definido na classe "`AgenteProspector`", nesta classe é realizado todo o processo de execução a partir do método "`executar`".

O agente começa por realizar uma perceção, avalia o ambiente em que se encontra o agente, procura alvos e identifica obstáculos.

De seguida o agente processa a perceção, através do método de processar do controlo e é escolhida a ação que deve ser tomada, sendo obtida uma ação.

Finalmente o agente atua sobre o ambiente, a partir da classe "`Mover`".

Caso a agente atinga o alvo o ambiente é reiniciado e o estado e ação atual são limpos. O agente volta à sua posição inicial.

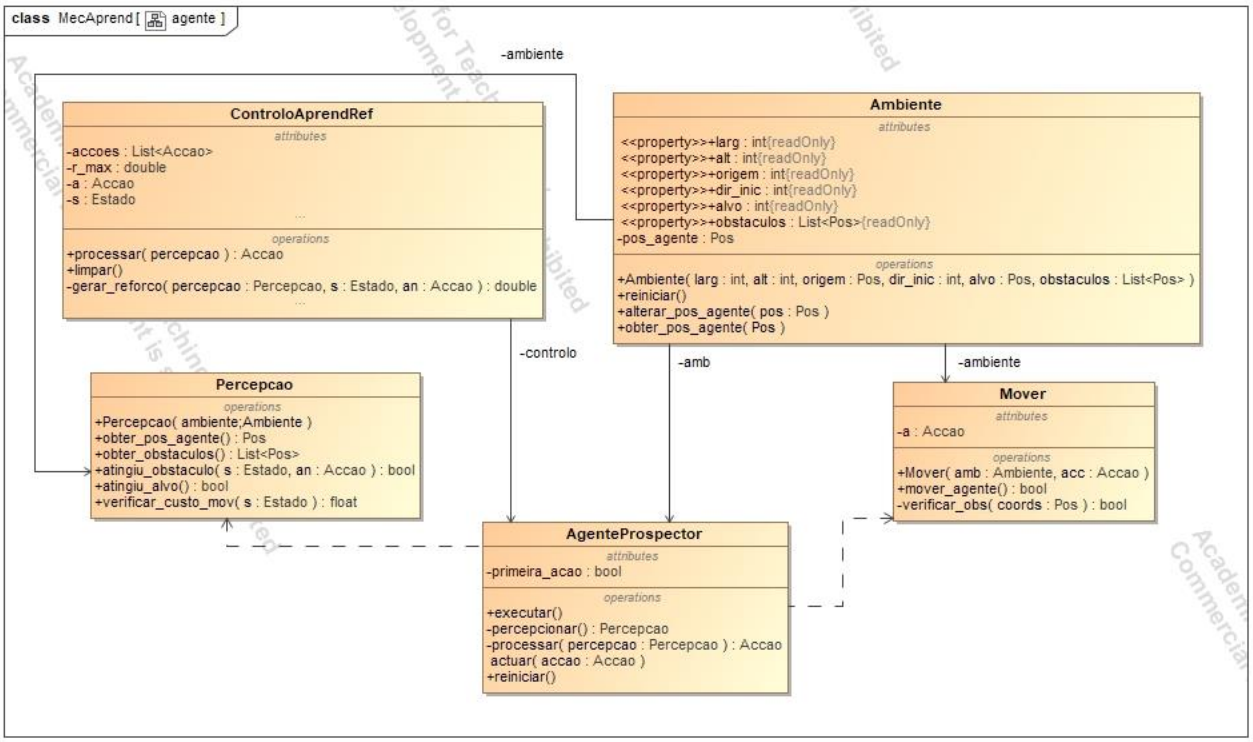


Figura 23 - Agente

Modo gráfico

O modo gráfico foi realizado utilizando a biblioteca *Pygame*. Na seguinte figura pode ser observado um exemplo da representação gráfica dos movimentos do agente no ambiente. O número de *frames* por segundo e as dimensões da janela podem ser ajustadas consoante a necessidade.

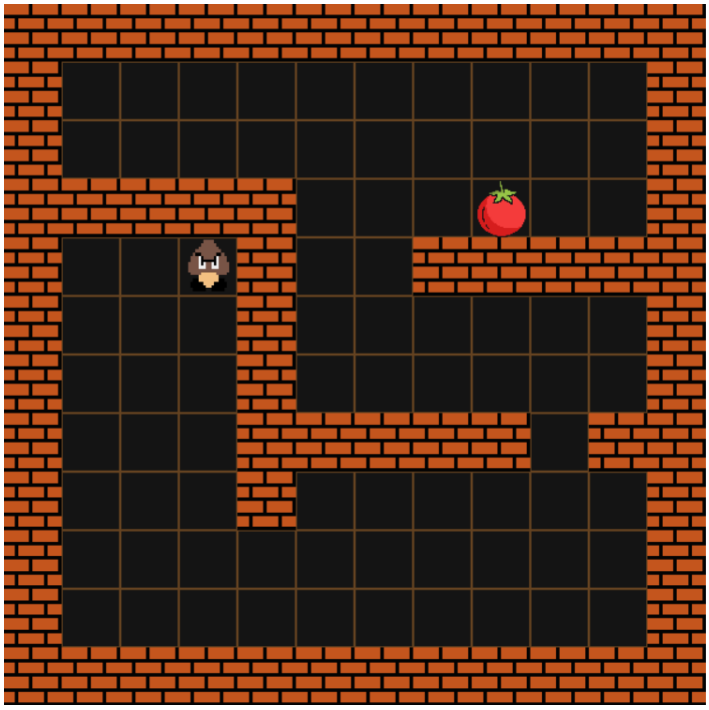


Figura 24 - Modo gráfico

Resultados obtidos

É de notar que o tempo de aprendizagem necessário para o método de aprendizagem *Dyna-Q* foi consideravelmente menor comparado com os outros métodos. Deve-se à exploração do ambiente através de simulações, além de aprender por experiências reais tendo assim vantagem em relação aos métodos de *Q-Learning*.

O método *Q-Learning* com memória episódica foi o segundo método mais rápido, enquanto o método *Q-Learning* normal foi o mais lento por uma margem considerável. Isto deve-se ao agente conseguir aprender a partir de experiências passadas, tornando o processo de aprendizagem mais rápido.

Parte 3

Arquitetura deliberativa

Arquitetura deliberativa é uma arquitetura em que a tomada de decisão se baseia num processo consciente e intencional, tendo ênfase nas representações internas do mundo. Esta arquitetura encontra a solução para um problema, através de simulações internas de todas as ações possíveis no sistema com base num mecanismo de procura de forma a atingir um objetivo. O processo de decisão tem em consideração:

- **Custo** de transição de estado
- **Utilidade** relativa a atingir o objetivo

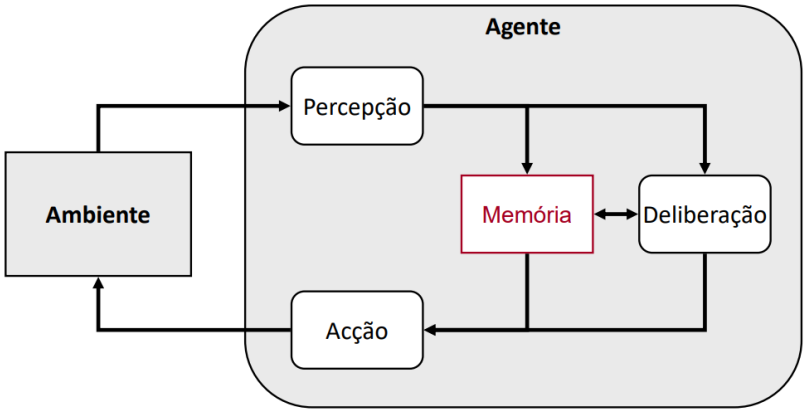


Figura 25 - Agente deliberativo

Procura em espaço de estados

A procura em espaço de estados baseia-se numa representação interna do mundo, em que os estados representam a configuração interna de um problema. Tem-se como objetivo encontrar o estado final a partir de um estado inicial, a partir de um percurso planeado. O espaço de estados pode ser representado por um grafo, em que os nós representam estados e as arestas as transições de estado válidas.

Dois critérios base da exploração do grafo de espaço de estados são:

- **Procura em profundidade** primeiro são explorados os nós com maior profundidade
- **Procura em largura** primeiro são explorados os nós com menor profundidade

Procura Melhor primeiro

A procura melhor primeiro, é um tipo de procura em espaço de estados que prioriza os estados com a melhor avaliação de acordo com uma função de avaliação ***f***.
A fronteira de exploração é ordenada por ordem crescente, onde são expandidos primeiros os estados com melhor avaliação. Este é um método de procura informada pois tira conhecimento do domínio do problema para ordenar a fronteira de exploração.

Procura A* ponderada

A procura A* é uma variante da procura melhor primeiro. Esta procura utiliza uma função heurística admissível ***f(n)*** como função de avaliação. Esta função de avaliação permite obter uma estimativa do custo do percurso desde o nó n até ao nó objetivo. Neste tipo de heurística a estimativa do custo é sempre inferior ou igual ao custo mínimo efetivo.

A procura A* ponderada difere da procura A* tradicional através da adição de um fator de ponderação *w*. A fórmula para a função *f(n)* é a seguinte:

$$f(n) = g(n) + w * h(n)$$

g(n): Custo do percurso até n
h(n): Estimativa do custo de n até ao objetivo dado por uma função heurística admissível
w: Fato de ponderação

Como na procura melhor primeiro, são primeiro expandidos os nós mais promissores, através da ordenação da fronteira de exploração.

Os nós que já foram expandidos e não têm de ser revistos novamente são guardados numa lista de nós fechados. Os nós são adicionados à lista de nós fechados depois de serem expandidos e

removidos da lista de nós abertos. Isto tem a vantagem de reduzir a lista de nós abertos, reduzindo a complexidade e necessidade de armazenamento.

A lista de nós explorados é semelhante a de nós fechados. Os nós explorados são os nós que já foram avaliados, foram adicionados à lista de nós abertos e a sua função ***f(n)*** foi calculada, entretanto não chegaram a ser explorados.

No projeto, a função h(n) utilizada foi a distância Euclidiana entre o nó e nó final.

$$h_1(n) = \sqrt{(x_n - x_{obj})^2 + (y_n - y_{obj})^2}$$

Procura Frente-Onda

A procura Frente-Onda é utilizada para encontrar o caminho mais curto entre um ponto inicial e um ponto final num ambiente. Esta procura baseia-se na expansão de nós a partir de um ponto inicial e associá-los com o custo estimado até ao ponto final. Em cada iteração o nó com menor custo estimado é expandido. A procura continua até o nó final ser alcançado ou todos os nós sejam expandidos.

Implementação

Foi implementada uma biblioteca de raciocínio automático para planeamento, com implementação dos seguintes métodos:

- Procura em espaço de estados A* ponderada
- Planeamento automático com base no método Frente-Onda

Procura em espaço de estados A * ponderada

Primeiro começou-se por implementar a procura em espaço de estados A* ponderada. As classes e métodos associados à biblioteca de raciocínio automático foram guardados num *package* chamado "pee*"* (procura em espaço de estados).

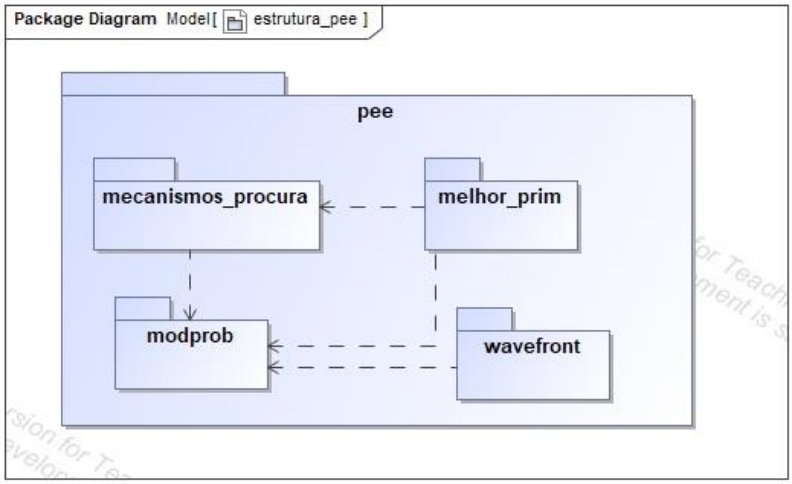


Figura 26 - estrutura

Inicialmente foi necessário definir as classes e métodos dos elementos do modelo do problema, respetivamente:

- **Estado**: representa configuração de um problema
- **Operador**: Vetor de transformação de estado, tem a função aplicar que permite realizar a transformação de estado, e a função custo que retorna o custo do operador.
- **Problema**: Definição do problema a resolver, definido o estado inicial, operadores, objetivo e heurística do problema.
- **Solução**: Definição da solução do problema, a solução é composta por um conjunto de nós.

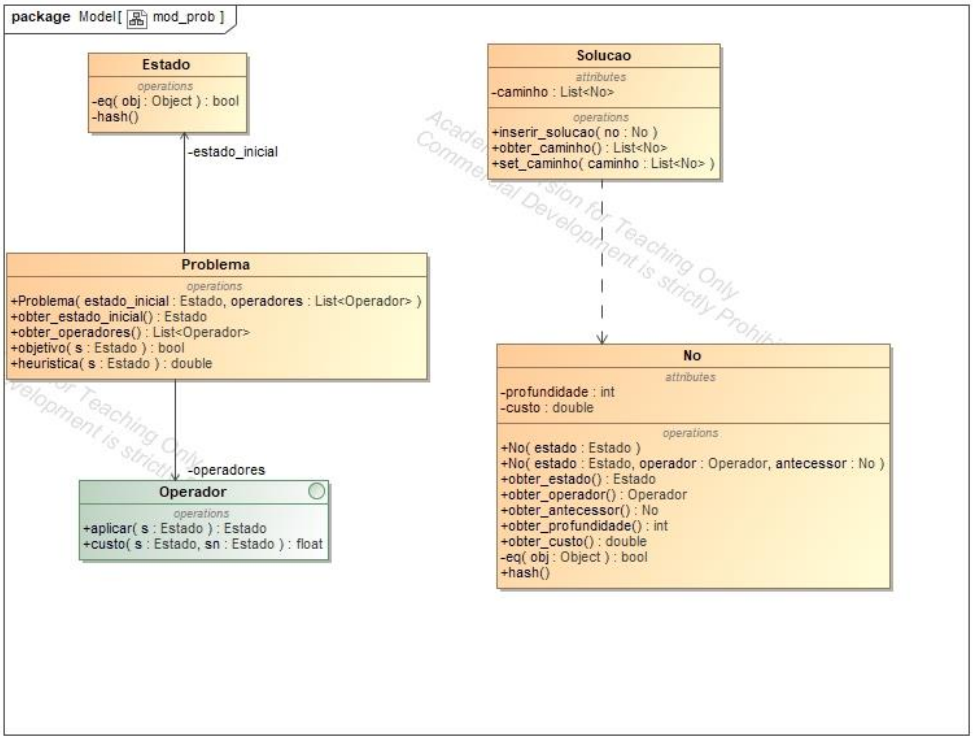


Figura 27 - Modelo do problema

De seguida, foi necessário definir os mecanismos necessários para a procura em espaço de estados, foi definida a classe "`MecanismoProcura`" e a classe "`MemoriaProcura`" que representa a memória de procura.

A classe "`MecanismoProcura`" expande os nós, resolve o problema recebido com limitação de profundidade e gera a solução para um problema apos ter sido achada uma solução num determinado nó. Tendo como ponto de partida o nó final, irá navegar pela arvore até atingir a sua raiz, devolvendo o caminho percorrido como a solução para o problema em causa.

A memória de procura armazena os nós para serem explorados, os nós já visitados e explorados e o nós fechados. Tem também métodos para adicionar e remover nós das respetivas listas. De seguida foi definida a ordem em que os estados são guardados na fronteira.

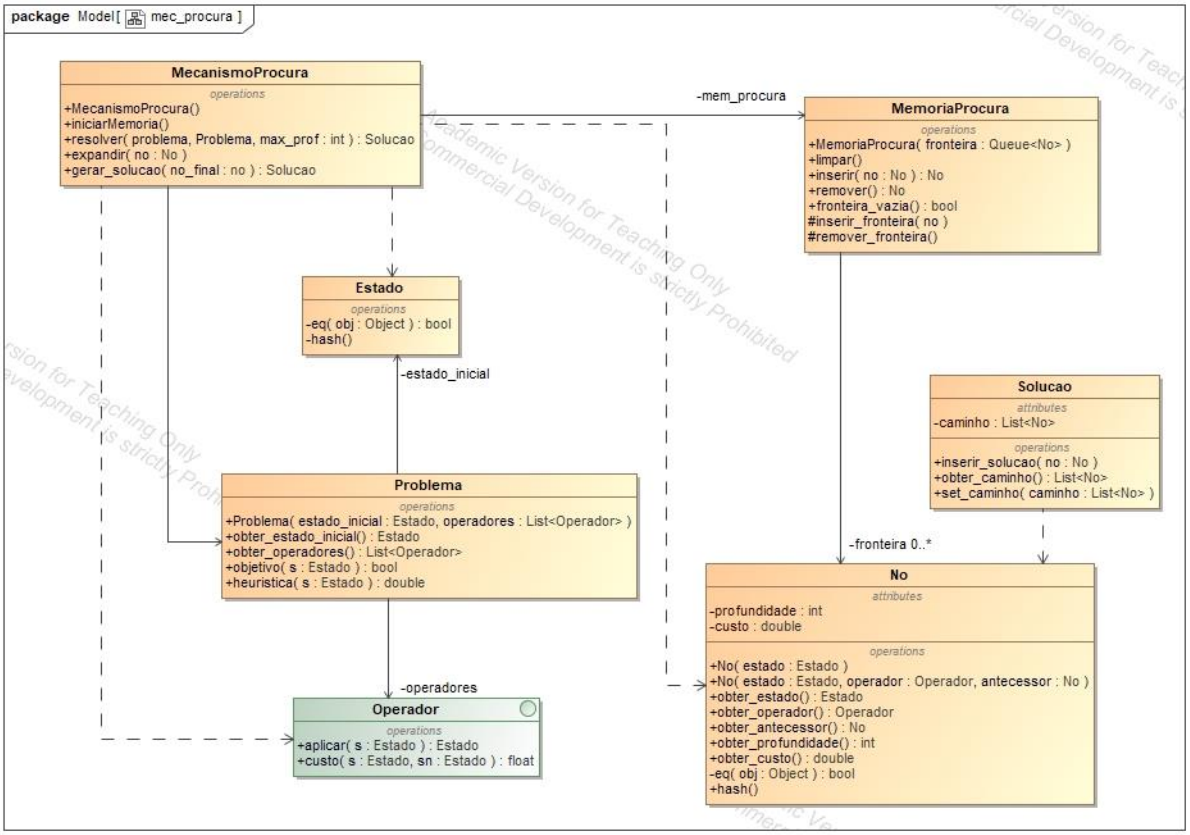


Figura 28 - mecanismos de procura

Memória

A memória de prioridade define a ordem pela qual os eles da fronteira são ordenados. Na memória de prioridade como se está a lidar com uma procura A* ponderada, os nós vão ser ordenados consoante uma ordem de prioridade. A prioridade é definida através de uma função de avaliação heurística.

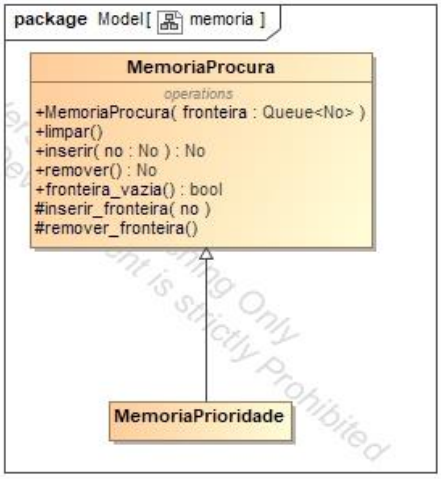


Figura 29 - Memória

Procura

A procura A* é uma variação da procura melhor primeiro, no método f da classe da procura A* é definida a função heurística. Como mencionado anteriormente a heurística utilizada no projeto foi a distância euclidiana entre dois nós.

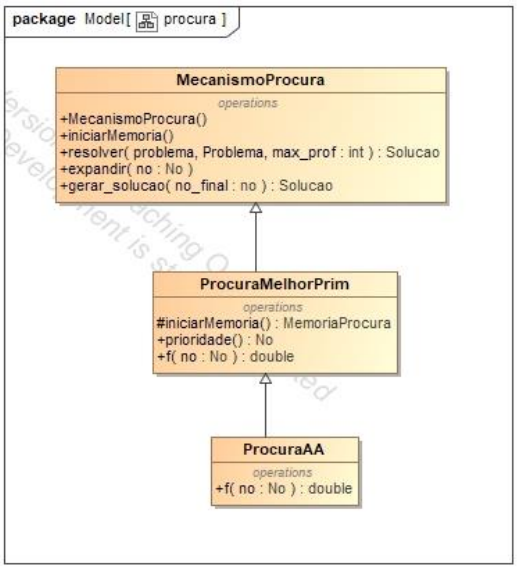


Figura 30 - Procura

De seguida foi implementada a adaptação do plano de espaço de estados definido ao problema em questão. Começou-se por definir o estado, operador e problema do trajeto.

- **Estado do trajeto** : Estado simples que contém as coordenadas x e y do estado.
- **Operador do trajeto** : Altera a posição do agente e retorna o custo associado ao movimento.
- **Planeador do problema**: Define o objetivo e heurística do problema.

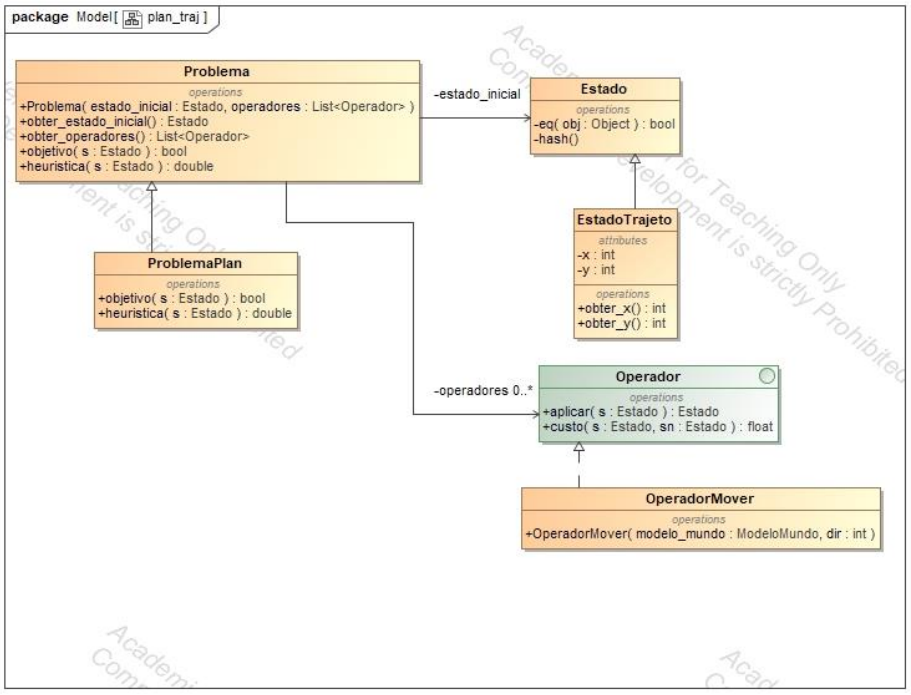


Figura 31 - Planeamento do trajeto

Planeamento

De forma a conseguir planear o trajeto do agente utilizou-se planeadores, que seguem um algoritmo específico para chegar a uma solução. No planeador foram definidos os operadores, o estado inicial e o modelo do mundo.

O modelo do mundo contém a informação do mundo como a posição dos obstáculos, do objetivo e do agente.

É necessário realizar um planeador para a procura A* e para o método Frente-Onda, visto que ambos têm algoritmos de forma de planeamento diferentes.

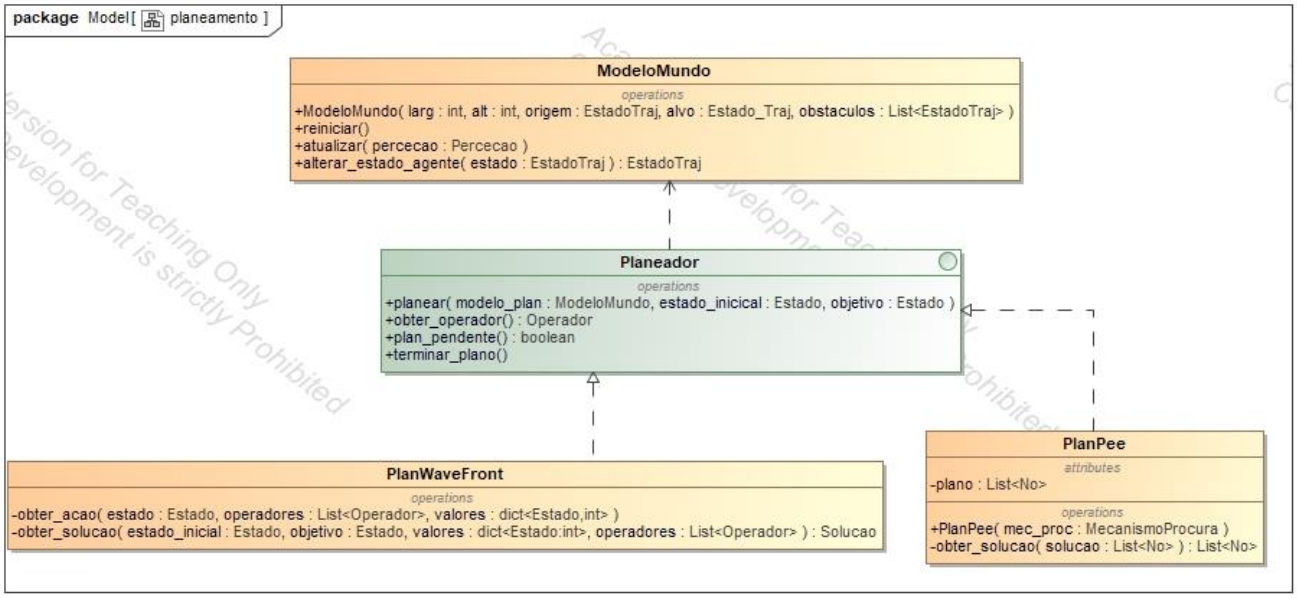


Figura 32 - Planeador

Controlo

O controlo permite ao agente processar o estado do ambiente e devolver a melhor ação a ser tomada. O controlo realiza através do seguinte processo:

1. Assimilar: É atualizado o modelo do mundo a partir de uma perceção recebida
2. Reconsiderar: O controlo verifica se o mundo foi alterado.
3. Deliberar: É definido o objetivo do problema.
4. Planear: Permite ao agente calcular o melhor percurso de ação através do planeador.

Agente

O agente realiza ações sobre o mundo consoante os operadores recebidos do controlador. O funcionamento do agente é o seguinte:

1. Percecionar: O agente avalia o ambiente em que se encontra o agente, procura alvos e identifica obstáculos.

2. Processar: Através da perceção do ambiente e o controlador, o agente recebe a ação que deve ser tomada.
3. Atuar: Agente atua a ação recebida no ambiente.

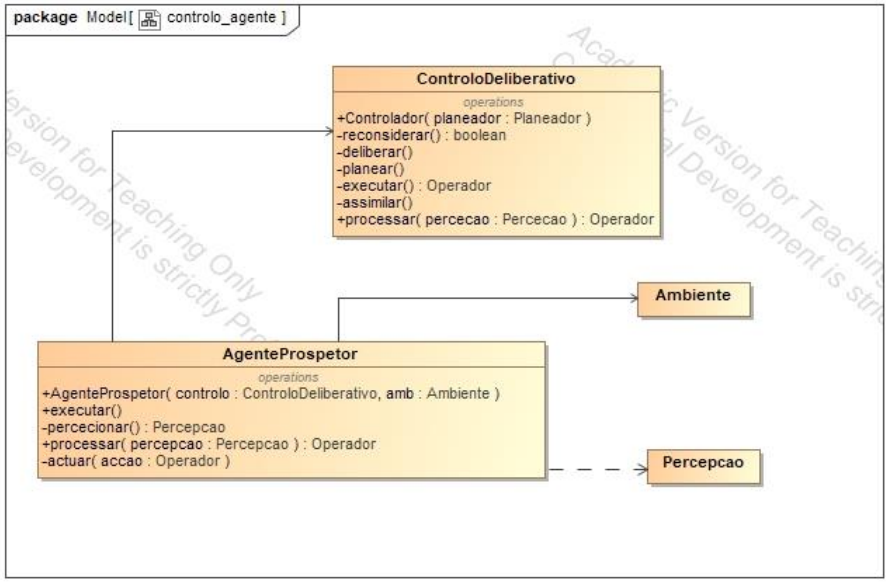


Figura 33 - Controlo e Agente

Modo gráfico

Foi realizado um modo gráfico em que pode ser visualizado o trajeto do agente. Pode ser também visualizado o plano desde o estado inicial até ao estado objetivo representado através de setas.

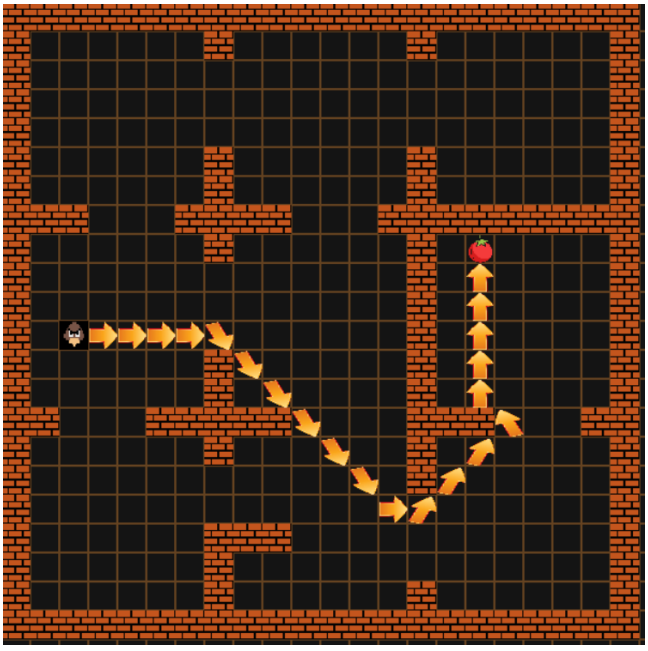


Figura 34 - Modo gráfico

Resultados obtidos

Notou-se que na procura de espaço de estados A* e no método Frente-Onda, o agente consegue chegar à solução através do percurso mais ótimo. Ambos os métodos foram eficazes para resolver o problema de o agente ter de chegar ao alvo no menor número de passos possíveis.

A procura de espaço de estados A* ponderada teve a vantagem de evitar expandir estados que não oferecem valor. O método Frente-Onda teve a vantagem de ter uma implementação mais simples comparado com a procura A* ponderada.

Otimização

A otimização baseia-se na seleção da melhor opção de um conjunto de opções possíveis a partir de um critério de avaliação de opções. Os métodos de otimização realizados no projeto são métodos meta-heurísticos. Sendo formas de procura em espaço de estados.

Na resolução de problemas de otimização pretende-se saber quais os parâmetros ótimos de modo a maximizar uma função de valor, isto resulta em uma configuração de parâmetros (estado).

Hill-Climbing

O método de otimização *Hill-Climbing* é a técnica de procura local mais simples. Em cada passo o nó corrente é substituído pelo melhor vizinho, até não ser encontrado nenhum vizinho com valor superior.
Este algoritmo tem o problema de ficar preso em ótimos locais, sendo não completo e ótimo.

A variante do *Hill-Climbing* utilizada no projeto, foi o *Hill-Climbing* estocástico, esta variante tem as seguintes características:

- Escolha aleatória entre sucessores que aumentam o valor do estado.
- Convergência mais lenta que *Hill-Climbing.*
- Pode encontrar melhores soluções.

Simmulated Annealing

Também foi implementado o método de otimização *Simmulated Annealing*. Este método utiliza o conceito de variação de temperatura para encontrar a melhor solução para um problema. Esta abordagem é útil para evitar mínimos locais e encontrar soluções otimizadas. Inicialmente é definido um estado inicial e uma temperatura inicial.

Por cada iteração do algoritmo é escolhido um estado aleatoriamente de um vizinho deste estado. Avalia-se se o estado escolhido é melhor, caso seja transita-se para este estado. Caso contrário, é utilizada a fórmula e **c/T, para definir se ocorre ou não transição de estado.

Implementação

Inicialmente foi definida uma biblioteca de otimização em que são definidos os dois métodos anteriores, *Simmulated Annealing e Hill-Climbing*.

Começou-se por definir uma classe abstrata "`Problema`" em que foram definidos os métodos que têm de ser implementados em ambos os problemas.

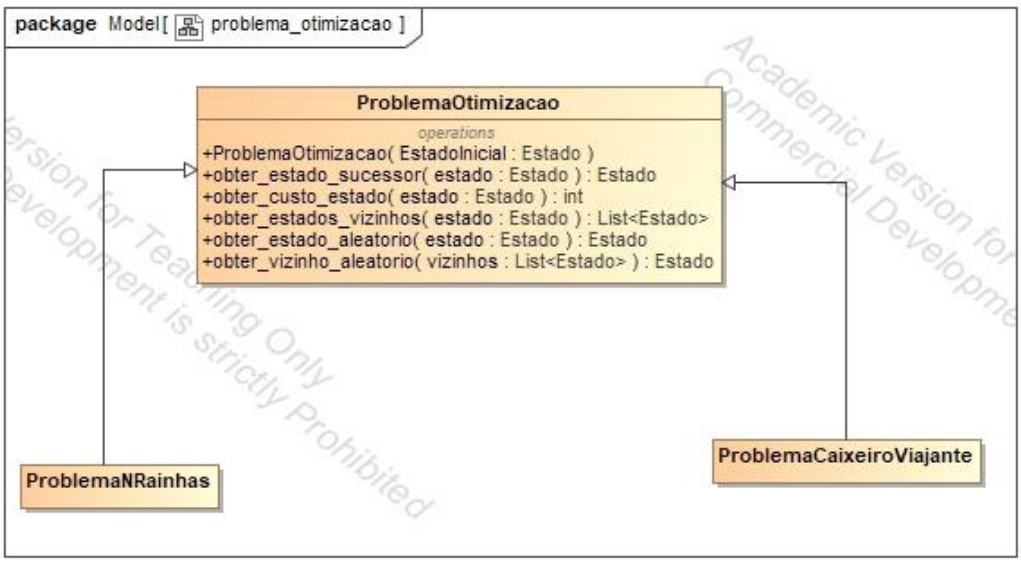


Figura 35 - Problema de Otimização

Foram também definidos os algoritmos, os algoritmos recebem uma instância da classe "`Problema`" e geram a solução.

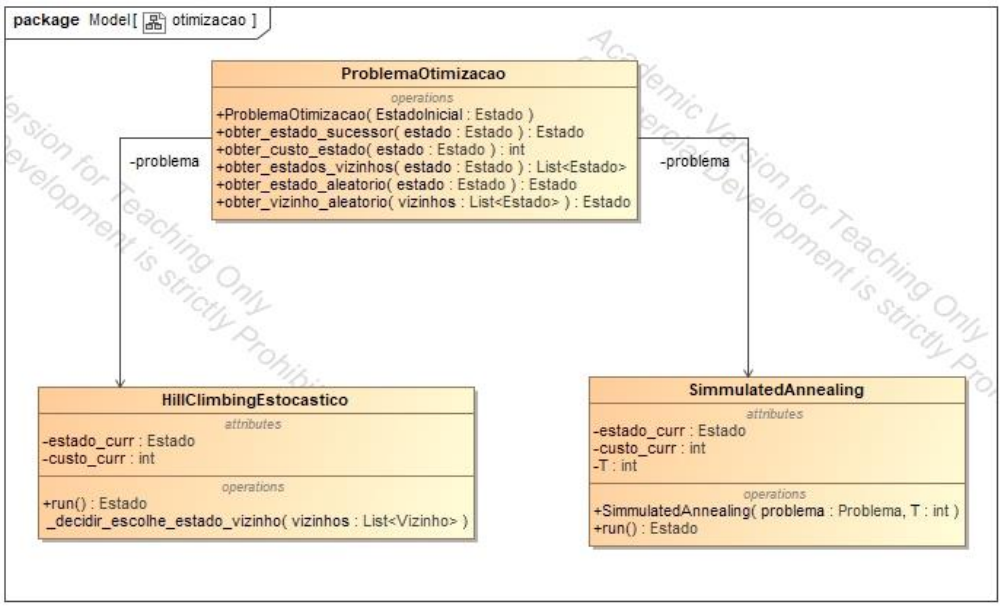


Figura 36 - Otimização

Problemas resolvidos

Os problemas que foram resolvidos com estes métodos de otimização foram:

- **Problema N-Rainhas:** Consiste em posicionar rainhas num tabuleiro de xadrez de tamanho N x N, de forma a que nenhuma das rainhas possa atacar a outra.
- **Problema Caixeiro viajante:** Consiste em encontrar o caminho mais curto em que um conjunto de cidades possam ser visitadas.

Problema N-Rainhas

Para resolver o problema N-Rainhas foi necessário obter o custo de cada estado. Cada estado é composto por *tuples* 2d (x,y), em que x é a coordenada das linhas e y a coordenada das colunas do tabuleiro de xadrez.

Foi necessário encontrar o custo de cada estado, para isto foram realizados métodos que permitem verificar num estado a quantidade de vezes que ocorre ataques entre as rainhas. Pretende-se que o número de ataques das rainhas seja zero.

Na seguinte figura pode ser observado um exemplo do resultado obtido para um tabuleiro com dimensão n = 4.

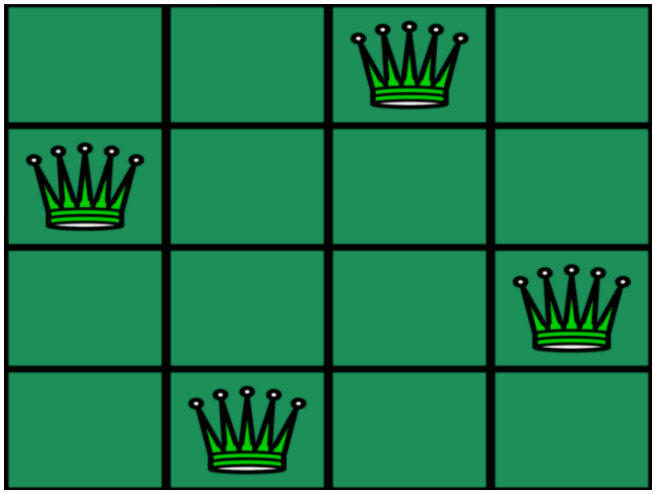


Figura 37 - Exemplo N-Rainhas

Problema Caixeiro Viajante

Para resolver este problema foi necessário inicialmente definir a lista de cidades que iria ser utilizada e o custo para viajar entre as diferentes cidades. A lista definida foi composta por um conjunto de cidades Europeias, incluindo Lisboa e Paris.

De seguida foi necessário realizar os métodos para obter o custo de cada estado e os seus vizinhos. O custo é obtido através da distância entre cidades.

Um exemplo de um resultado obtido neste problema foi:

'Madrid' -> 'Amsterdam' -> ‘Budapest' -> 'Rome' -> 'London' -> 'Lisbon' -> 'Vienna' -> 'Paris' -> 'Berlin' -> ‘Brussels' -> ‘Madrid'

Resultados obtidos

Foi de notar que o método *Simmulated Annealing* teve melhor desempenho em ambos os problemas. Neste método foram retornadas soluções mais ótimas e consistentes do que com o *Hill-Climbing*.

De seguida apresentam-se os resultados obtidos para o problema N-Rainhas com a configuração:

Configuração: `[(0,0),(2,0),(1,0),(3,0),(4,1),(1,2),(5,2),(5,3)]`

Método	Custo	Última iteração
Hill-Climbing	1	4
Simmulated-Annealing	1	999

Figura 38 - Resultado obtido

Ambos os métodos tiveram custo semelhante, mas o método *Hill-Climbing* exigiu um menor número de iterações.

Os resultados obtidos para o problema Caixeiro Viajante com a seguinte configuração, teve estes resultados:

Configuração: `["Berlin", "Paris", "London", "Madrid", "Amsterdam", "Vienna", "Budapest", "Brussels", "Lisbon", "Rome"]`

Método	Custo	Última iteração
Hill-Climbing	5279	12
Simmulated-Annealing	5009	999

Figura 39 - Resultado obtido

O método *Simmulated Annealing* teve resultados ligeiramente superiores, mas exigiu um maior número de iterações.

DALL·E

DALL-E é um sistema de inteligência artificial que foi criado pela OpenAI. Este sistema é caraterizado por conseguir gerar texto e imagens a partir de uma descrição de linguagem natural. Este sistema tem andado a ganhar imensa popularidade nos últimos meses.

O modelo deste sistema foi treinado a partir de dados disponíveis na Internet, o que permite uma elevada compreensão de assuntos e tópicos relevantes na Internet. O número exato de dados utilizados no treino do DALL-E não se encontra exposto publicamente. Entretanto, sabe-se que foi treinado utilizando milhares de exemplos de textos de diferentes fontes. Tendo também sido treinado para resolver tarefas como geração de texto, tradução e adaptabilidade a erros do utilizador.

O sistema é não supervisionado, foi treinado sem a definição de etiquetas e classes para os dados de treino.

As imagens geradas pelo DALL-E são caraterizadas pelo seu alto detalhe e realismo, sendo por vezes díficil conseguir distinguir uma imagem gerada por este sistema com uma imagem real.

Na seguinte imagem pode-se observar exemplos de imagens geradas pelo DALL-E para diferentes descrições:

Figura 40 - Imagens geradas a partir de DALL-E

Entretanto com um sistema tão eficaz a simular uma tarefa que muito recentemente era apenas realizada por humanos, é inevitável que surgem questões filosóficas.
Cada vez mais artistas têm se manifestado contra a introdução de inteligência artificial no processo de criação artística. Medos e receios de sistemas como o DALL-E substituírem e diminuírem o valor de obras criadas por humanos são uns dois maiores argumentos contra o DALL-E. Entretanto existem argumentos a favor, como o aumento da eficiência, estas ferramentas podem permitir que artistas consigam criar obras de arte mais rapidamente e facilmente.

Este é um tópico altamente controverso, sendo difícil dar uma opinião concreta e absoluta sobre estas questões. Ambos os lados têm fortes argumentos, sendo este um bom exemplo da luta constante entre a valorização e manutenção de valores tradicionais contra a inovação tecnológica.

Conclusões

Em suma neste projeto foram realizados diveros modelos e arquiteturas de inteligência artificial. Inicilamente começou-se por abordar e estudar redes neuronais. Redes neuronais são um conceito muto importante na área de inteligência artificial. Baseiam-se no sistema nervoso humano e são capazes de realizar multiplas tarefas e a aprender a partir de dados. Redes neuronais são muito utilizadas em problemas de reconhecimento de informação, classificação e reconhecimento de padrões.

A aprendizagem por reforço foi o segundo tópico abordado. Este é um tipo de aprendizagem em que o agente aprender a realizar ações num ambiente consoante o objetivo de ter um valor máximo de recompensa. Aprendizagem por reforço é muito utilizada em áreas como video-jogos, robótica e controlo de sistemas automáticos. A aprendizagem por reforço tem a grande vantagem de conseguir aprender em problemas sem supervisão.

A última área de inteligência artificial abordada no projeto foi o raciocinio automático. Esta etapa do projeto foi a mais complexa e demorada. Nesta etapa foi implementado um agente deliberativo, que utiliza processos de tomada de decisão lógicos para atingir os seus objetivos. Foram abordados problemas de procura em espaço de estados. A utilização de procura em espaço de estados teve a vantagem de conseguir encontrar uma solução ótima de forma mais eficiente e rápida do que na aprendizagem por reforço. Finalmente foram abordadas técnicas de otimização, tendo sido resolvidos dois problemas conhecidos de inteligência artificial. A otimização tem a vantagem de permitir encontrar soluções ótimas ou muito próximas disso, de forma rápida e eficiente.

Bibliografia

[1] - Luís Morgado, Inteligência Artificial e Sistemas Cognitivos, 02-Inteligência artificial e sistemas cognitivos, 2023

[2] - Luís Morgado, Inteligência Artificial e Sistemas Cognitivos, 03-Redes neuronais artificiais, 2023

[3] - Luís Morgado, Inteligência Artificial e Sistemas Cognitivos, 05-Aprendizagem por reforço, 2023

[4] - Luís Morgado, Inteligência Artificial e Sistemas Cognitivos, 08-Raciocínio automático, 2023

[5] - Luís Morgado, Inteligência Artificial e Sistemas Cognitivos,11-Arquitecturas cognitivas, 2023