

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA



**ÁREA DEPARTAMENTAL DE ENGENHARIA DE
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES**

**Mestrado em Engenharia
Informática e Multimédia**

**Inteligência Artificial e
Sistemas Cognitivos**

Relatório Final

**Ano Lectivo:
2021/2022**

Aluno:
45414 - Arman FREITAS

Professor:
Eng. Luis MORGADO

06/02/2022

Contents

1	Introdução	4
2	Inteligência artificial e sistemas cognitivas	5
2.1	Inteligência Artificial	5
2.2	Redes Neuronais	6
2.2.1	Momento	6
2.2.2	Taxa de Aprendizagem	7
2.3	Técnicas de otimização	9
2.3.1	<i>Stochastic Hill Climbing</i>	9
2.3.2	<i>Simulated Annealing</i>	10
2.4	Algoritmos genéticos	12
2.5	Agentes Inteligentes	14
2.5.1	Arquitetura Reativa	15
2.5.2	Arquitetura Deliberativa	16
2.6	<i>Wavefront</i>	17
2.7	Procura RTAA*	17
2.8	Algoritmo <i>Dyna-Q</i>	18
3	Projeto - Objetivo 1	19
3.1	Função XOR	19
3.1.1	Resultados	20
3.1.1.1	Impacto do Momento	21
3.1.1.2	Impacto da Taxa de Aprendizagem	22
3.1.1.3	Efeito da apresentação das amostras de treino com ordem fixa ou aleatória	23
3.1.1.4	Efeito de utilização de uma codificação binária ou bipolar	24
3.2	Classificação de um padrão	25
3.3	Classificação de uma imagem (nuvem ou sol)	27
4	Projeto - Objetivo 2	29
4.1	<i>Hill Climbing</i> e <i>Simulated Annealing</i>	30
4.1.1	Implementação	31
4.1.2	Resultados	34
4.2	Algoritmos Genéticos	36
4.2.1	Implementação	37
4.2.2	Implementação por biblioteca externa	40
5	Projeto - Objetivo 3	41
5.1	Algoritmos	42
5.1.1	Procura em espaços de estados	42
5.1.1.1	<i>Wavefront</i>	42

5.1.1.2 <i>Real Time Adaptive A*</i>	43
5.1.2 Aprendizagem por reforço	46
5.2 Planeador	49
5.3 Agente	51
5.4 Representação gráfica do agente	53
6 A presença de inteligência artificial na Web 3.0 e no Metaverse	55
7 Conclusão	60

List of Figures

1 Cálculo dos incrementos dos pesos da rede	7
2 Descida de gradiente com Momento	7
3 <i>Hill Climbing</i>	9
4 Máximo local	9
5 Funções de decaimento da temperatura	10
6 <i>Simulated Annealing</i> vs <i>Hill Climbing</i>	11
7 <i>One Point Crossover</i>	12
8 <i>Multi Point Crossover</i>	12
9 <i>Bit Flip Mutation</i>	13
10 <i>Swap Mutation</i>	13
11 Diagrama de uma Arquitectura Reativa	15
12 Diagrama de uma Arquitetura Deliberativa	16
13 Processo de aprendizagem	18
14 Tabela de verdade - XOR	19
15 Referencial XOR	20
16 Estrutura da rede	20
17 Regiões de decisão das redes para os respetivos valores de momento 0, 0.5, 1	20
18 Momento = 0	21
19 Momento = 0.5	21
20 Taxa de aprendizagem	22
21 Ordem Fixa	23
22 Ordem aleatória	23
23 Dados bipolares	24
24 Dados binários	24
25 Padrões de entrada	25
26 Estrutura da rede proposta	26
27 Exemplo de uma nuvem	27
28 Exemplo de um sol	27
29 Diagrama de <i>Packages</i>	29
30 Diagrama UML - Algoritmos de otimização	30
31 Diagrama UML - Implementação dos problemas	31

32	Diagrama UML - Implementação dos operadores	32
33	Diagrama UML - Implementação do <i>Plot</i> para os diferentes problemas	33
34	Caixeiro Viajante	34
35	N-Rainhas	34
36	Caixeiro Viajante	35
37	N-Rainhas	35
38	Diagrama UML - Algoritmos Genéticos	36
39	Diagrama UML - <i>Crossover</i>	37
40	Diagrama UML - <i>Mutation</i>	38
41	Diagrama UML - <i>Selection</i>	38
42	Diagrama UML - <i>Problem</i>	39
43	Caixeiro Viajante - Resultados	39
44	N-Rainhas - Resultados	39
45	Caixeiro Viajante - <i>pyeasyga</i>	40
46	N-Rainhas - <i>pyeasyga</i>	40
47	Classe de implementação de <i>Wavefront</i>	42
48	Diagrama UML - Mecanismo geral de procura	43
49	Diagrama UML - Mecanismo de memória	44
50	Diagrama UML - Mecanismos específicos de procura	44
51	Diagrama UML - Definição do problema	45
52	Diagrama UML - Algoritmos de aprendizagem por reforço	46
53	Diagrama UML - Mecanismo de memória	47
54	Diagrama UML - Mecanismo de seleção de ações	47
55	Diagrama UML - Mecanismo de Aprendizagem	48
56	Organização dos <i>packages</i>	48
57	Diagrama UML - Modelo do problema	49
58	Diagrama UML - Planeadores	50
59	Diagrama UML - Arquitetura de agente	52
60	Organização dos <i>packages</i>	52
61	Representação gráfica do RTAA*	53
62	Representação gráfica do <i>Wavefront</i>	53
63	<i>DynaQ</i> em aprendizagem	54
64	<i>DynaQ</i> após aprendido o ambiente	54
65	Estrutura do <i>Metaverse</i>	57
66	Estrutura de uma Rede Neuronal Convulcional	60

1 Introdução

Este relatório engloba 3 objetivos realizados ao longo do semestre na Unidade Curricular de Inteligência Artificial e Sistemas Cognitivos:

- **Objetivo 1** - Redes Neuronais
- **Objetivo 2** - Raciocínio Automático para Otimização
- **Objetivo 3** - Raciocínio automático para planeamento, Aprendizagem por reforço e, Agentes Inteligentes

Ao longo do documento serão abordados os tópicos utilizados em cada objetivo teoricamente. Serão também apresentadas as formas utilizadas para resolução dos objetivos.

Será também apresentado o tema da presença da inteligência artificial na nova era da *internet*, como tema escolhido para pesquisa.

2 Inteligência artificial e sistemas cognitivas

De maneira sucinta serão abordados temas gerais sobre os quais se basearam as realizações dos trabalhos práticos.

2.1 Inteligência Artificial

É o campo que estuda a síntese e análise de agentes computacionais que atuam de forma inteligente. Um agente inteligente é um sistema que percebe seu ambiente e toma atitudes que maximizam sua hipótese de sucesso.

Existem 3 paradigmas principais de Inteligência Artificial:

- Simbólico: A inteligência é resultante da ação de processos sobre estrutura simbólicas
- Conexionalista: A inteligência é uma propriedade emergente das interações de um número elevado de unidades elementares de processamento
- Comportamental: A inteligência resulta da dinâmica comportamental individual e conjunta de múltiplos sistemas a diferentes escalas de organização

2.2 Redes Neuronais

Uma rede neuronal é uma rede que reflete o funcionamento do cérebro humano, fazendo com um computador consiga reconhecer padrões e resolver problemas.

Uma rede destas, possuí uma camada de entrada e, uma de saída. Toda ela é composta por neurônios, sendo que, podem existir várias camadas escondidas entre o *input* e o *output*.

2.2.1 Momento

O Momento (ou *Momentum*) é um conceito que vem da física mecânica e, é nada mais nada menos do que o produto da massa de um corpo pela sua velocidade, como diz a sua fórmula, $p = mv$. Quanto maior o momento de um objeto maior velocidade consoante a sua massa.

Na inteligência artificial este conceito permite transmitir um significado muito similar, dado que é aplicado ao algoritmo de descida de gradiente e, vem a ditar quão rápida ou forte é a descida/delta de cada iteração para encontrar o mínimo de uma função. Não só é mais fácil chegar à convergência pois as iterações têm mais inércia, como podem ser evitados mínimos locais de funções e, encontrados mínimos óptimos.

Isto é, o cálculo dos deltas de cada peso da rede que, sem qualquer momento, apenas é calculado através do produto da taxa de aprendizagem com o gradiente dos pesos, passa a adicionar parte do delta dos pesos da iteração anterior. Esta adição é o que dá impulso à convergência da rede. Em seguida é possível ver o cálculo dos deltas/incrementos dos pesos com e sem momento (fig. 1):

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

↑ weight increment ↑ learning rate ↘ weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

↗ momentum factor ↗ weight increment, previous iteration

Figure 1: Cálculo dos incrementos dos pesos da rede

A figura 2 mostra os deltas maiores nas descidas e, o encontro de um mínimo óptimo após a descoberta de um mínimo local.

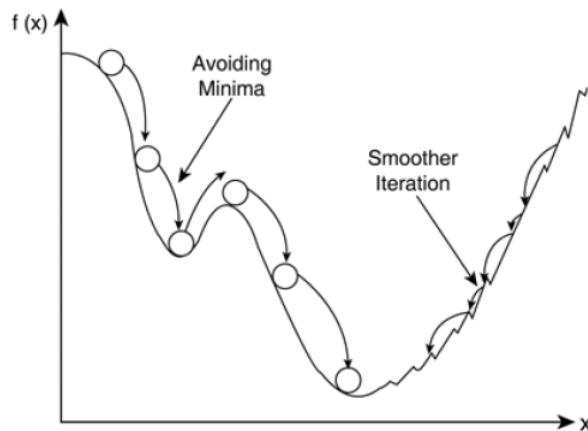


Figure 2: Descida de gradiente com Momento

2.2.2 Taxa de Aprendizagem

Tal como o nome diz, a taxa de aprendizagem, controla quanto rápido o modelo vai aprender. Ao contrário do que pode parecer, este parâmetro não determina que o nível de erro será menor caso a taxa seja maior.

A taxa de aprendizagem é o parâmetro que, conforme os erros encontrados na retro propagação, define quanto devem mudar os pesos do sistema atual. Maior a taxa de aprendizagem mais rápido converge o sistema. Ao contrário do termo de momento, este pertence diretamente à iteração em que se está a atualizar os pesos e, não tem a ver com pesos de iterações anteriores.

Este parâmetro insere-se na equação de atualização dos pesos mostrada anteriormente na figura 1.

2.3 Técnicas de otimização

Seram abordadas agora duas técnicas de otimização, o *Stochastic Hill Climbing* e, o *Simulated Annealing*.

2.3.1 Stochastic Hill Climbing

O *Stochastic Hill Climbing*, provém do algoritmo base *Hill Climbing*. O algoritmo começa num dado estado e, através de operadores, tenta chegar a um estado melhor que o anterior. De entre os estados gerados pelos operadores, é sempre escolhido aquele que mais valor te. Chama-se *Hill Climbing* pois o mesmo tenta encontrar máximos num dado espaço de estados. o pico da seguinte figura pode representar um máximo descoberto pelo algoritmo. Quando já não existe nenhum operador que forneça um estado de melhor valor, o algoritmo termina.

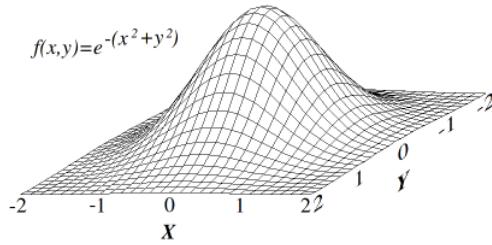


Figure 3: *Hill Climbing*

O principal problema é quando se chega a um máximo local. A figura 4 mostra o inicio de um caminho, que eventualmente chega a um máximo local que não tem saída.

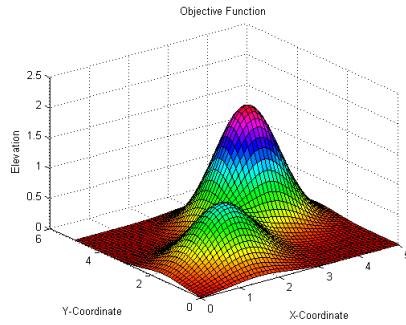


Figure 4: Máximo local

A variante *Stochastic* tenta agilizar este problema. Esta apenas dita que dos estados gerados pelos operadores, deve ser escolhido um estado aleatório que tenha mais valor que o estado presente. Como é de esperar, esta solução não corrige todos os problemas do *Hill Climbing*.

2.3.2 *Simulated Annealing*

O *Simulated Annealing* é baseado no processo físico de aquecer materiais e, lentamente arrefecer para mudar a estrutura do material.

Esta técnica é muito útil quando existem muitos máximos (ou mínimos) locais.

O algoritmo começa com um estado solução e, uma temperatura inicial. É utilizada uma função que defina o decaimento da temperatura ao longo das iterações. Alguns exemplos de funções encontram-se na seguinte figura (fig. 5)

- | | |
|------------------------------|---------------------------|
| 1. Linear Reduction Rule: | $t = t - \alpha$ |
| 2. Geometric Reduction Rule: | $t = t * \alpha$ |
| 3. Slow-Decrease Rule: | $t = \frac{t}{1+\beta t}$ |

Figure 5: Funções de decaimento da temperatura

A cada iteração do algoritmo, é escolhido aleatoriamente um vizinho de um estado. Se o estado sucessor seja melhor que o anterior, transita-se para o mesmo. Caso contrário é utilizada a seguinte probabilidade: $e^{\Delta c/T}$, para definir caso se transita ou não para esse estado. O Δc corresponde à diferença entre o custo do estado atual para o estado sucessor e, o T , a temperatura.

Assim, que algum critério de paragem seja encontrado, o algoritmo pára, ou caso chegue ao número máximo de iterações.

A figura 6, representa a vantagem do *Simulated Annealing* comparado com o *Hill Climbing*. O *Simulated Annealing* permite que possam haver saltos maiores que levem a máximos (ou mínimos, dependendo do problema) globais.

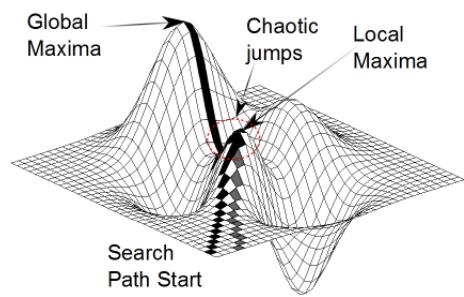


Figure 6: *Simulated Annealing vs Hill Climbing*

2.4 Algoritmos genéticos

Algoritmos genéticos são algoritmos de pesquisa com base em conceitos como seleção natural e genética.

Antes de proceder, é necessário estabelecer as seguintes terminologias:

- **População** - Conjunto de possíveis soluções.
- **Cromossomas** - É uma das soluções pertencente à população.
- **Gene** - É um elemento de um dado cromossoma.

A forma como funciona é a seguinte, é inicializada uma população (conjunto de possíveis soluções), selecionados alguns dos cromossomas, através de técnicas de seleção, e, é calculada a função *fitness* para cada cromossoma. A **função fitness** caracteriza um cromossoma com um valor, costuma-se relacionar com o custo.

É realizada a operação de ***Crossover***. Esta consiste na combinação de cromossomas. Tipicamente são "cortados" dois cromossomas (com um ou dois cortes) e, são criadas combinações, gerando novos sucessores. As figuras 7 e 8, representam dois tipos de *Crossover*

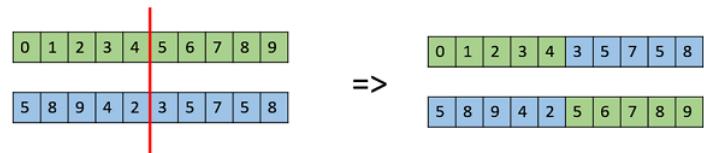


Figure 7: One Point Crossover

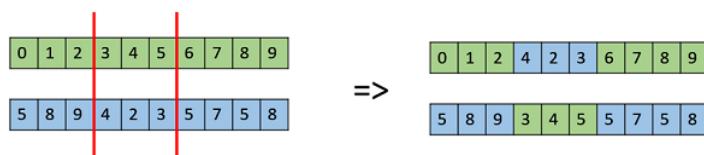


Figure 8: Multi Point Crossover

Após a criação de sucessores, serão selecionados aleatoriamente alguns que irão ser mutados. Uma mutação consiste numa pequena diferença nalgum(ns) gene(s) do cromossoma. Alguns tipos de mutação podem ser a *Swap Mutation* e, a *Bit Flit Mutation*.

Ambas encontram-se descritas nas figuras 9 e 10.

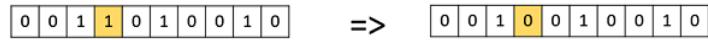


Figure 9: *Bit Flip Mutation*

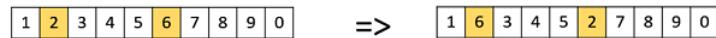


Figure 10: *Swap Mutation*

Após as mutações (caso haja alguma), o sistema volta a repetir o processo, sendo a nova população, os cromossomas sucessores da geração passada.

Caso não seja definido nenhum critério de paragem, o sistema apenas terminará de correr o algoritmo caso chegar ao número máximo de iterações.

2.5 Agentes Inteligentes

Um agente inteligente deve possuir autonomia, reatividade, pro-atividade e sociabilidade onde tudo isto deve levar o agente a um propósito.

A autonomia do agente é a capacidade do mesmo de operar por si próprio e isto é uma característica de inteligência, logo, inteligência implica autonomia.

Existem três tipos de arquiteturas para um agente:

1. Reativas (comportamentais).
2. Deliberativas (cognitivas).
3. Híbridas.

2.5.1 Arquitetura Reativa

Este tipo de arquitetura associa uma resposta a um estímulo. Para isto, existe uma percepção do meio envolvido e é realizada uma ação predefinida à percepção do ambiente. Múltiplas percepções podem potencialmente activar múltiplas respostas.

A seleção de ações pode dar-se de 3 maneiras:

1. Execução paralela de ações, onde estas não interferem entre si e são executadas em paralelo
2. Combinação de ações, onde diferentes ações ativadas por diferentes percepções são combinadas e gerem uma única ação
3. Precedência de ações, as ações interferem entre si onde as ações mais importantes possuem precedência.

Fica aqui uma imagem explicativa de um agente reativo:

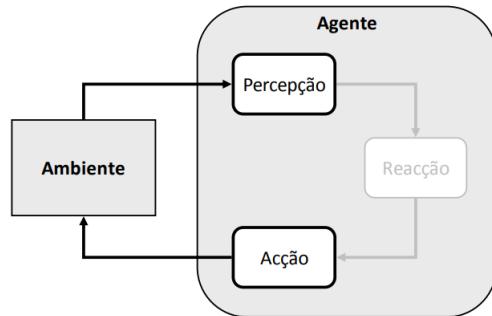


Figure 11: Diagrama de uma Arquitectura Reativa

2.5.2 Arquitetura Deliberativa

Como o nome indica, uma arquitetura deliberativa não toma uma ação, transitando de estado, baseada apenas na percepção do agente. A seleção de ação passa agora a ser um processo deliberativo. Este processo é muito dependente do modelo utilizado.

Este tipo de arquitetura tem como base um sistema que encontra uma **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 **objectivo**.

Neste projecto o processo de decisão implica uma avaliação com base em dois factores:

- **Custo** de transição de estado
- **Utilidade** relativamente a atingir o objectivo

Estas avaliações levam a que o agente encontre a melhor solução possível dentro do ambiente que o rodeia com base no seu **mecanismo de procura**.

Fica aqui uma imagem explicativa de um agente deliberativo:

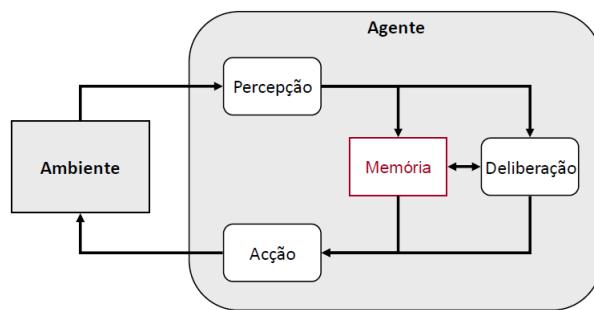


Figure 12: Diagrama de uma Arquitetura Deliberativa

2.6 *Wavefront*

O *Wavefront* (ou Frente-de-onda), é um algoritmo de planeamento que planeia o caminho de um agente até ao seu objetivo.

A peculiaridade deste método, é o facto de começar a pesquisa pelo objetivo (ou objetivos) até chegar ao agente. A pesquisa a partir do objetivo é feita em largura, ou seja, são explorados todos os nós mais antigos sempre, fazendo uso de uma memória *FIFO* (*First In First Out*). Por cada nó expandido, é dado um valor ao mesmo. Estes valores vão incrementando até chegar ao agente.

Após a pesquisa se ter alastrado até ao agente, o mesmo apenas necessita de seguir o gradiente dos valores criados pelo *Wavefront* até chegar ao nó objetivo.

2.7 Procura RTAA*

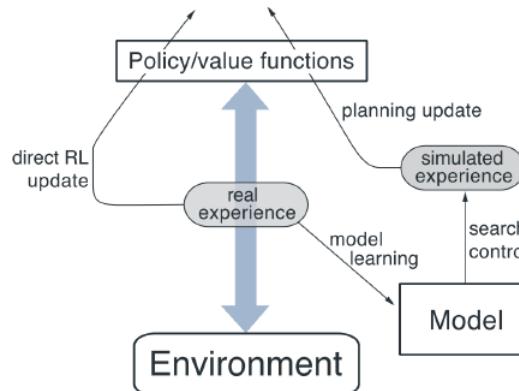
A procura RTAA* (*Real Time Adaptive A**) é uma variante da procura A*. No entanto, esta consegue resultados mais rapidamente. O funcionamento da mesma segue as seguintes normas, primeiramente é realizada uma procura A* com uma profundidade máxima de nós a explorar (*Lookahead*). Após feita a pesquisa, são atualizados os valores de heurística de todos os nós fechados. Por fim, basta apenas seguir o caminho de custo menor para o nó onde a procura A* deixou e, ir fazendo procura A* e, atualizando as heurísticas até chegar ao nó objetivo.

2.8 Algoritmo Dyna-Q

O algoritmo *Dyna-Q*, pertence ao ramo de aprendizagem por reforço. A aprendizagem por reforço é o treino de agentes de forma a fazerem uma sequência de decisões. O agente aprende a obter um objetivo num ambiente desconhecido.

O *Dyna-Q* ilustra como a experiência real e simulada conseguem ser combinadas na construção de uma política. É utilizada experiência de simulação criada por um modelo, para encontrar ou atualizar a política para interagir com o ambiente.

O algoritmo implementa ambos a aprendizagem com um modelo do mundo e, aprendizagem direta, como dita a figura 13.



[Sutton & Barto, 2020]

Figure 13: Processo de aprendizagem

3 Projeto - Objetivo 1

O objetivo 1 consiste numa série de problemas de classificação redes neuronais.

3.1 Função XOR

Pretende-se implementar uma rede neuronal que capaz de realizar operação lógica XOR. A operação lógica XOR consiste numa operação entre dois bits de entrada e, um de saída. A sua tabela de verdade é a que se apresenta de seguida (fig. 14):

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Figure 14: Tabela de verdade - XOR

Parece um problema fácil de resolver, dado que apenas são dadas duas entradas à rede (os bits a realizar a operação) e, apenas é necessária uma saída (0 ou 1). No entanto, como podemos verificar na figura em seguida, para classificar um operador XOR não necessitamos de apenas uma barreira de decisão, mas sim duas. As coordenadas de cada ponto no referencial representam os neurónios de entrada na rede. Neste caso é uma codificação bipolar, isto é, invés de termos entradas e saídas entre 0 e 1 temos entre -1 e 1.

É de notar que uma rede pode tanto convergir para a figura acima, assim como convergir para um resultado parecido mas simétrico a esse, mas no geral a convergência deve ser algo parecido para um *score* de 1. Sabendo então que, para resolver o problema temos de traçar duas retas, é necessário no mínimo uma rede com dois neurónios na camada escondida. A estrutura da rede deverá ter o seguinte aspeto (fig. 16):

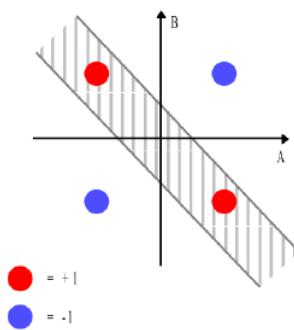


Figure 15: Referencial XOR

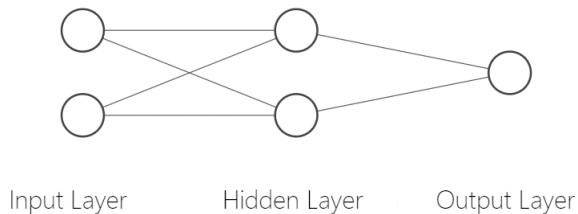


Figure 16: Estrutura da rede

3.1.1 Resultados

Os resultados da rede, com uma *loss*, sempre menor que 0.001, deram origem a regiões de decisão como as seguintes:



Figure 17: Regiões de decisão das redes para os respetivos valores de momento 0, 0.5, 1

Todos eles, apesar de valores de momento variados, se assimilam à figura 15, demonstrada no planeamento da rede.

3.1.1.1 Impacto do Momento

$\alpha = 0$	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	306651	90508	52759	32269	17473
2	308355	90677	52475	30538	17419
3	305032	90581	52721	30193	17286
4	304451	90493	53325	30784	17400
5	305531	90806	56445	30552	17379
6	305142	90896	52776	30264	17478
7	305767	165859	52985	50113	17313
8	306832	90866	52431	30611	17344
9	305712	90585	52819	30274	17474
10	306286	90713	52610	30395	17392
11	305552	90818	52409	30458	17653
12	306278	90784	52720	30435	17334
13	307486	91007	52953	30307	17213
14	305071	90474	53120	30653	17270
15	306753	90370	53047	30344	17665
Média:	306010,4	95695,8	53039,67	31879,33	17406,2

Figure 18: Momento = 0

$\alpha = 0.5$	r = 0.05	r = 0.25	r = 0.5	r = 1	r = 2
1	181965	52617	30372	17441	9937
2	487868	52701	30470	17434	9930
3	183702	52802	30436	17487	9847
4	182254	52580	30642	17977	9789
5	184494	52701	33132	17243	9788
6	181930	52760	30665	17379	9845
7	182997	52926	30301	17302	9809
8	182928	52990	30705	17285	9833
9	184990	55115	30519	17335	9847
10	183433	53317	30444	17383	9990
11	183359	52578	30366	17275	9910
12	183081	53002	30246	17411	9819
13	183090	53192	30289	17300	9828
14	182740	53031	31192	17496	9798
15	182618	52451	30722	17385	9789
Média:	203429,9	52984,2	30700,07	17408,87	9850,6

Figure 19: Momento = 0.5

Como é possível verificar nos dados recolhidos, a introdução de um termo de momento, diminui o número de iterações que a rede demora a convergir. Este fenómeno pode ser explicado pelo ponto 2 do documento.

A figura 17 mostra também que, com o aumento do termo de momento, os limiares de decisão encurtam bastante. Isto pois, como os incrementos dos pesos, durante a retro propagação, são muito maiores os saltos de uma região para outra são muito mais súbitos.

3.1.1.2 Impacto da Taxa de Aprendizagem

$\alpha = 0$	r = 0,05	r = 0,25	r = 0,5	r = 1	r = 2
1	306651	90508	52759	32269	17473
2	308355	90677	52475	30538	17419
3	305032	90581	52721	30193	17286
4	304451	90493	53325	30784	17400
5	305531	90806	56445	30552	17379
6	305142	90896	52776	30264	17478
7	305767	165859	52985	50113	17313
8	306832	90866	52431	30611	17344
9	305712	90585	52819	30274	17474
10	306286	90713	52610	30395	17392
11	305552	90818	52409	30458	17653
12	306278	90784	52720	30435	17334
13	307486	91007	52953	30307	17213
14	305071	90474	53120	30653	17270
15	306753	90370	53047	30344	17665
Média:	306010,4	95695,8	53039,67	31879,33	17406,2

Figure 20: Taxa de aprendizagem

Tal como esperado, à medida que se vai aumentando a taxa de aprendizagem, mais rápido converge o sistema, apenas com base nas iterações do presente e, não do passado como faria o momento.

3.1.1.3 Efeito da apresentação das amostras de treino com ordem fixa ou aleatória

$\alpha = 1$		$r = 0.05$	$r = 0.25$	$r = 0.5$	$r = 1$	$r = 2$
1		141	56	38	31	28
2		131	56	57	35	24
3		95	49	52	30	27
4		127	50	40	28	31
5		110	69	48	28	24
6		116	64	48	55	24
7		95	60	40	33	24
8		99	86	55	40	27
9		95	56	36	34	27
10		103	49	47	37	27
11		106	54	42	31	26
12		100	54	41	32	25
13		118	56	42	30	28
14		109	46	40	32	26
15		130	63	49	29	30
Média:		111,6667	57,86667	45	33,66667	26,53333

Figure 21: Ordem Fixa

Execução	$r = 0.05$	$r = 0.25$	$r = 0.5$	$r = 1$	$r = 2$
1	94	65	46	43	26
2	125	54	44	35	27
3	93	51	44	37	27
4	129	58	55	34	33
5	115	60	44	32	27
6	94	56	45	32	31
7	95	53	55	37	31
8	103	58	53	32	27
9	112	80	52	34	24
10	115	53	45	31	28
11	97	52	48	32	29
12	103	52	35	30	32
13	163	53	47	34	27
14	102	68	40	33	27
15	110	70	59	34	29
Média:	110	58,86667	47,46667	34	28,33333

Figure 22: Ordem aleatória

Aqui, diferencia-se amostras de treino com ordem fixa ou aleatória. Isto é, caso os dados tenham sempre a mesma ordem, tem ordem fixa e, caso se opte por baralhar os dados antes de do treino é ordem aleatória.

Ambos os resultados têm valores muito semelhantes, pelo que se percebe que no caso presente ordem fixa e aleatória não causa efeitos nos testes. A razão deste acontecimento pode ter a ver com a dimensão da amostra ser demasiado pequena para se notar qualquer diferença.

3.1.1.4 Efeito de utilização de uma codificação binária ou bipolar

$\alpha = 1$	$r = 0.05$	$r = 0.25$	$r = 0.5$	$r = 1$	$r = 2$
1	141	56	38	31	28
2	131	56	57	35	24
3	95	49	52	30	27
4	127	50	40	28	31
5	110	69	48	28	24
6	116	64	48	55	24
7	95	60	40	33	24
8	99	86	55	40	27
9	95	56	36	34	27
10	103	49	47	37	27
11	106	54	42	31	26
12	100	54	41	32	25
13	118	56	42	30	28
14	109	46	40	32	26
15	130	63	49	29	30
Média:	111,6667	57,86667	45	33,66667	26,53333

Figure 23: Dados bipolares

Execução	$r = 0.05$	$r = 0.25$	$r = 0.5$	$r = 1$	$r = 2$
1	116	54	60	30	37
2	110	73	43	38	30
3	125	62	48	30	25
4	132	67	51	34	25
5	118	48	46	47	26
6	98	54	44	44	26
7	126	60	40	36	29
8	149	68	43	50	31
9	99	60	59	38	25
10	115	75	38	32	26
11	123	59	45	43	26
12	102	67	54	35	29
13	155	59	41	36	30
14	88	85	51	37	30
15	98	63	50	31	27
Média:	116,9333	63,6	47,53333	37,4	28,13333

Figure 24: Dados binários

A codificação bipolar, a ser utilizada, significa que os dados podem ser previstos como sendo 1 e -1 e, a codificação binária como 1 e 0.

Mais uma vez, as gamas de valores não variam muito pelo que pode ter a ver com a pequena dimensão da amostra.

3.2 Classificação de um padrão

O segundo desafio proposto foi uma rede que, consoante uma imagem com um determinado padrão ('X' ou 'O') adivinhe a que classe esse padrão pertence. Os padrões fornecidos pelo docente são os seguintes:

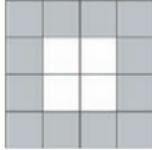
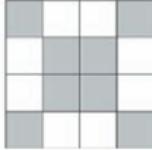
Padrão A	Padrão B
	
Classificação	Classificação
$A = 1$	$A = 0$
$B = 0$	$B = 1$

Figure 25: Padrões de entrada

Como se pode ver, a imagem consiste numa matriz 4x4, onde cada célula pode ou não estar activa. A entrada na rede deverá ser então 16 neurónios, cada um simboliza uma pixel da imagem.

Trata-se de um problema com duas classes, a A e a B. Como existem apenas estes dois padrões, a separação das regiões de decisão torna-se linear, o que nos leva facilmente à determinação de apenas um neurónio na camada escondida da rede.

O *output* da rede contém dois neurónios que, cada um corresponde às respetivas classes, A e B.

A rede ficou com a seguinte estrutura (fig. 26):

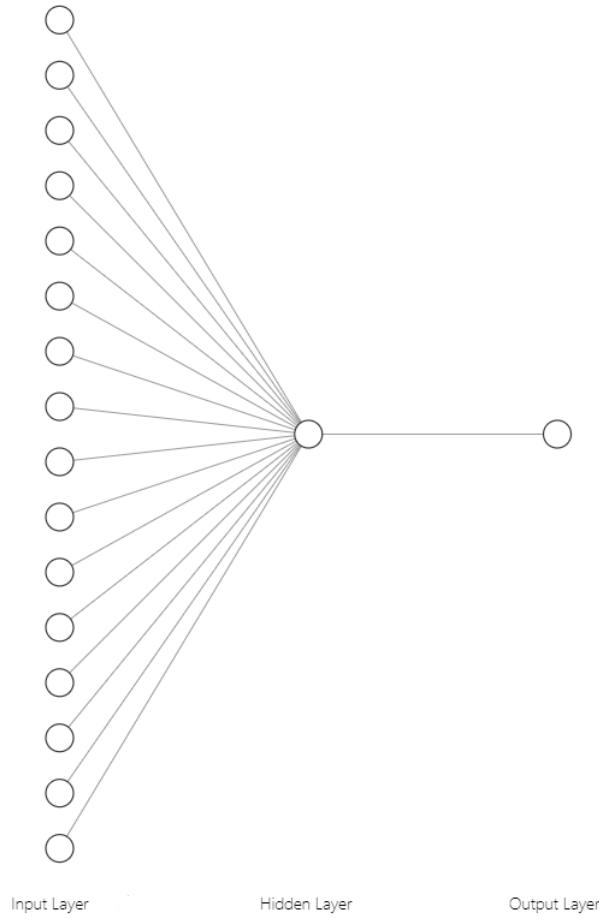


Figure 26: Estrutura da rede proposta

Através dos seguintes resultados obtidos, conseguimos ver que a função de ativação que minimiza o termo *loss* e o número de iterações para convergir da rede é a função identidade.

Identidade		Relu		Tangente Hiperbólica		Logística	
Iterações	Loss	Iterações	Loss	Iterações	Loss	Iterações	Loss
142	0.00072	428	0.00632	340	0.00582	665	0.01569
159	0.00057	374	0.00375	394	0.00608	700	0.01563
132	0.0007	380	0.00289	362	0.00770	624	0.01680

3.3 Classificação de uma imagem (nuvem ou sol)

O ultimo exercício, desafia os alunos a aplicarem redes neurais a um problema, de forma a resolvê-lo. Aqui, foi escolhida a classificação de imagens, a partir de um *dataset* com imagens de nuvens e sóis. É um problema parecido ao do ponto 3.2. No entanto, invés de apenas um padrão com apenas 4x4 células, o *dataset* disponibiliza um vasto número de imagem com resolução de 64x64 pixels.

De seguida, estão dois dos exemplos de imagens facultadas pelo *dataset*:



Figure 27: Exemplo de uma nuvem



Figure 28: Exemplo de um sol

Trata-se mais uma vez de uma classificação binária, mas neste caso, de imagens o que aumenta a complexidade do problema. A rede terá como entrada cada pixel da imagem. Esta contém 64x64 pixels, o que é 4096 que, multiplicado por cada canal de cores (RGB) dará 12288 neurónios de entrada.

Com um valor tão grande, a rede iria demorar bastante a correr cada iteração. Para isso, trata-se de cada imagem, deixando apenas os aspetos que se necessita, as chamadas *features*. Para classificar caso uma imagem é um sol ou uma nuvem não é de todo necessário saber as cores. Facilmente conseguimos dizer as diferenças entre uma nuvem e um sol pela sua forma.

Assim, são removidos os canais de cores normais (RGB) e, passa-se apenas a ter uma imagem a preto e branco. O nível de entradas baixa drasticamente, ficam 1/3 do que estava antes, mais precisamente. Por fim, 64x64 se calhar pode ser demais apenas para distinguir duas formas. Dando *resize* à imagem é possível baixar ainda mais a complexidade do problema. Neste caso cada imagem de teste e treino final ficou com 30x30 pixels.

Como explicado anteriormente, é um problema binário e que, se calhar apenas precisa de uma reta como limiar de decisão. Logo, como camadas escondidas foi apenas escolhida uma com um neurónio e, também um neurónio de saída.

4 Projeto - Objetivo 2

O objetivo 2 consiste na realização de uma biblioteca de métodos de raciocínio automático para otimização, com implementação do método *Simulated Annealing* e o *Stochastic Hill Climbing* assim como uma de algoritmos genéticos.

É também pedido o uso desta biblioteca para a resolução do problema do: Caixeiro viajante e N-Rainhas.

Caixeiro Viajante : O problema do caixeiro viajante faz a seguinte pergunta: "Dada uma lista de cidades e distâncias entre cada par delas, qual é o caminho mais pequeno que consegue passar por todas as cidades?".

N-Rainhas : Este problema faz a seguinte pergunta: "Dado um tabuleiro N*N com N rainhas, qual é a configuração que as rainhas têm de tomar para que nenhuma faça cheque a mais nenhuma".

O diagrama de pacotes de ambas as bibliotecas ficou com a seguinte aparência (fig. 29).

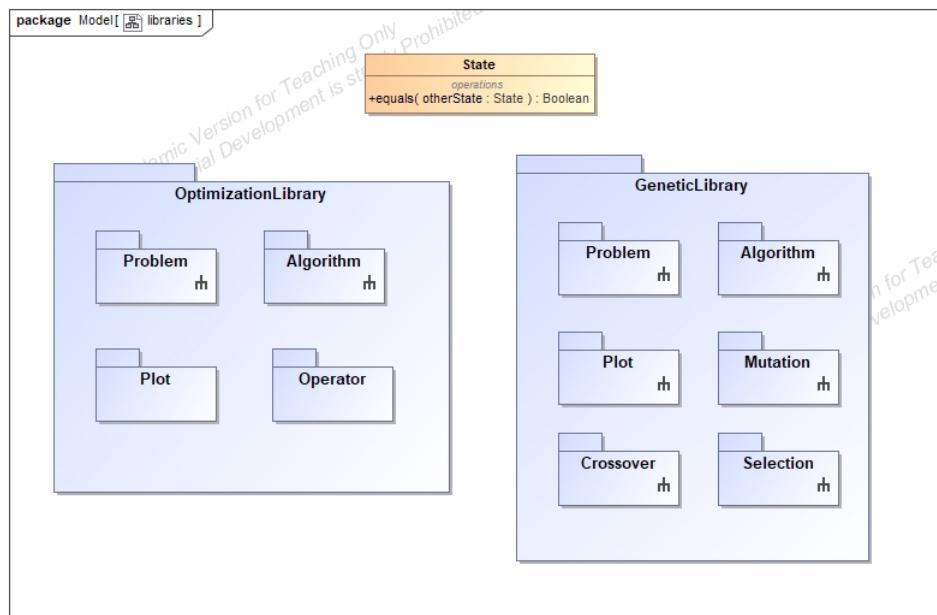


Figure 29: Diagrama de *Packages*

4.1 Hill Climbing e Simulated Annealing

A biblioteca de otimização de *Hill Climbing* e *Simulated Annealing* encontra-se nas seguintes partes, de forma a criar desacoplamento:

- **Algoritmo** - Algoritmo desacoplado.
- **Problema** - Definição do problema a resolver.
- **Operador** - Operador de mudança de estado.
- **Plot** - Vizualização do estado.

A seguinte figura demonstra o diagrama UML da estrutura do algoritmo (fig. 30).

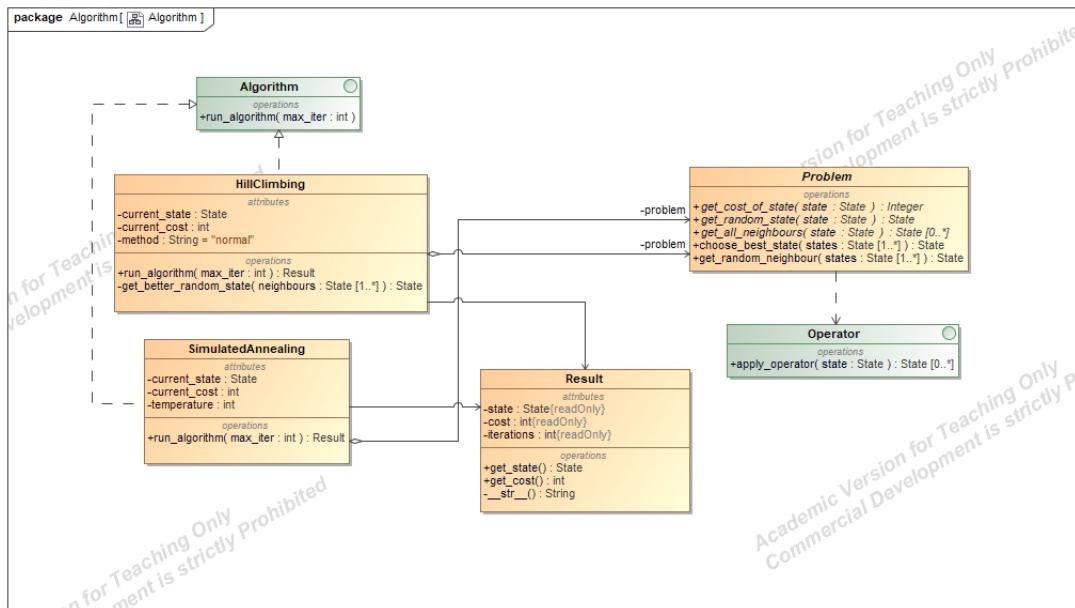


Figure 30: Diagrama UML - Algoritmos de otimização

Um algoritmo foi definido como sendo um conceito constante. Isto é, o algoritmo é completamente independente do problema a resolver. Para isso, foi necessária a criação de uma classe abstrata com os métodos necessários para qualquer problema. Apenas os métodos de escolha

de vizinhos aleatórios e escolha do melhor estado são implementados nesta classe, pois foram considerados iguais para qualquer problema.

É também possível averiguar a presença de uma interface *Operator*, que representa a modificação de um estado, de forma a chegar a um estado vizinho.

Os algoritmos seguem o mesmo contrato (*Algorithm*), permitindo caso necessário adições à biblioteca. É de notar também que qualquer algoritmo retorna um resultado do tipo *Result*, para manter coerência entre todos.

4.1.1 Implementação

Como se pode ver, os algoritmos estão criados na biblioteca. Todavia, o utilizador deve sempre criar um problema assim como um operador de mudança de estado.

Para o problema do Caixeiro Viajante e das N-Rainhas, foram criados os seguintes problemas e operadores (figs. 31 e 31).

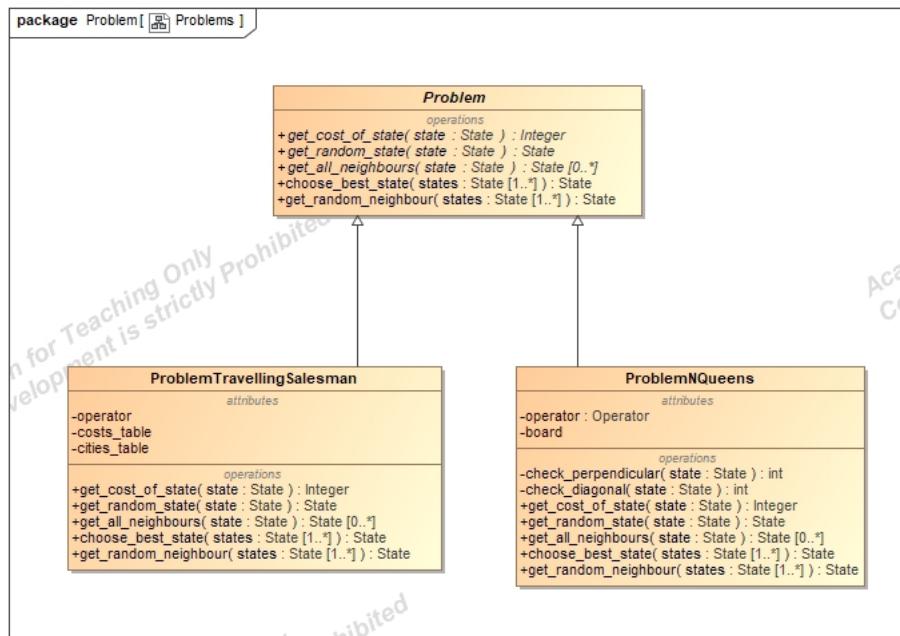


Figure 31: Diagrama UML - Implementação dos problemas

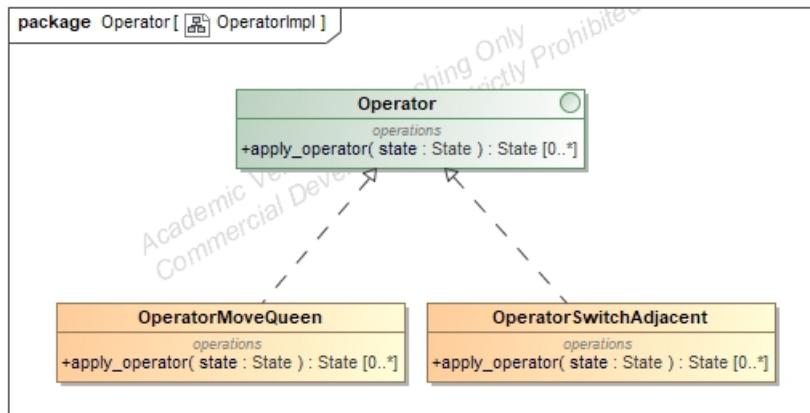


Figure 32: Diagrama UML - Implementação dos operadores

Para o problema do Caixeiro Viajante:

- O operador representa trocar a ordem de cidades adjacentes.
- A função de custo é a distância de entre todas as cidades adjacentes num determinado estado.

Para o problema das N-Rainhas:

- O operador representa a mudança das peças das rainhas de sítio
- A função de custo é o número de cheques existentes no tabuleiro

A biblioteca possibilita também a visualização de um estado, dada a interface *Plot*, à qual o utilizador pode especificar uma classe *Plot*. Os seguintes tipos de visualização utilizam a biblioteca *matplotlib* para demonstrar um estado (fig. 33):

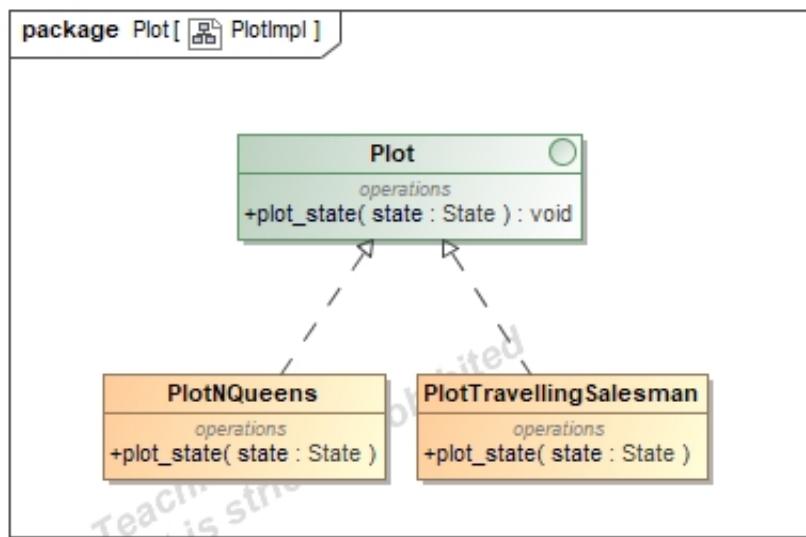


Figure 33: Diagrama UML - Implementação do *Plot* para os diferentes problemas

4.1.2 Resultados

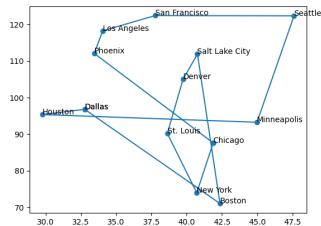


Figure 34: Caixeiro Viajante

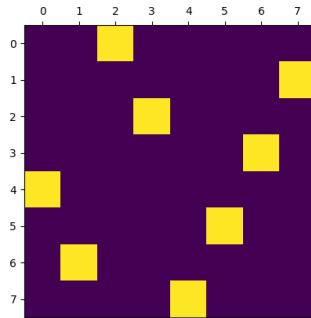


Figure 35: N-Rainhas

Stochastic Hill Climbing

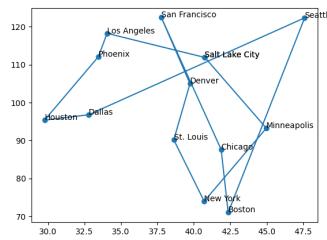


Figure 36: Caixeiro Viajante

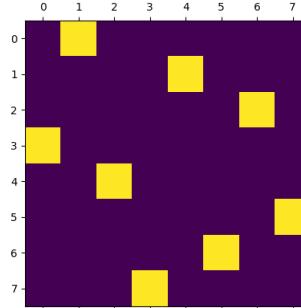


Figure 37: N-Rainhas

Simulated Annealing

É possível concluir que o *Simulated Annealing* gera, a maior parte das vezes um *output* melhor que o *Hill Climbing*, devido a ficar ”preso” em máximos (ou mínimos) locais menos vezes.

4.2 Algoritmos Genéticos

Embora pareça bastante parecida à biblioteca descrita anteriormente, a biblioteca de Algoritmos Genéticos distingue-se em alguns aspectos.

Como se pode ver a seguir (fig. 38), esta biblioteca é bastante similar à anterior, tendo a estrutura do algoritmo muito parecida.

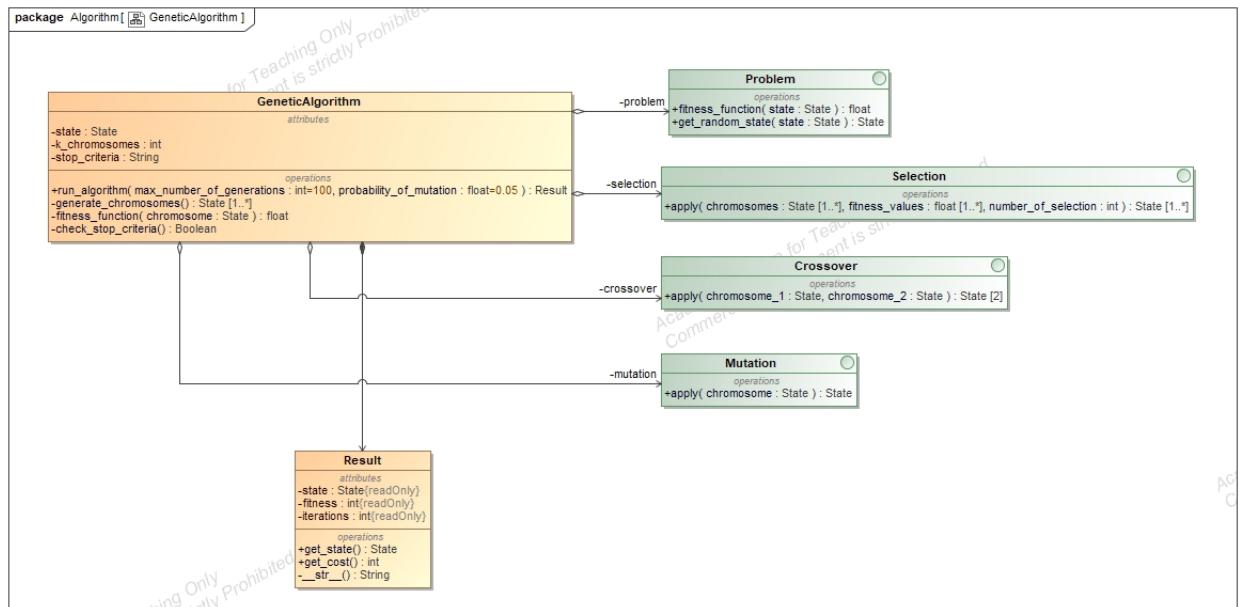


Figure 38: Diagrama UML - Algoritmos Genéticos

O Problema e o Algoritmo têm o mesmo intuito. No entanto, a *Selection*, o *Crossover* e a *Mutation* são diferentes operadores que os algoritmos genéticos utilizam.

É possível notar que, neste caso, o problema é uma interface e, não uma classe abstrata. Isto pois nos algoritmos genéticos não ter sido encontrado qualquer método que seja sempre o mesmo para qualquer problema.

O algoritmo utiliza as interfaces de cada operador (*Selection*, *Mutation* e *Crossover*), para resolver o problema (interface *Problem*). Cada operador apenas é aplicado, gerando um ou mais estados sucessores e, a função *fitness* faz parte do problema.

4.2.1 Implementação

Na implementação foram implementadas duas formas de *crossover*, o *Multi Point Crossover*, e o *One Point Crossover*. Em termos de mutações, foi implementada a *Swap Mutation* que apenas muda dois elementos de um estado. Já para a seleção foi utilizada a dada em aula, a seleção por roleta. Na classe de problema (classe *Problem*) para o caixeiro viajante e para o N-Rainhas foi definida uma função de *fitness* diferente para cada um. No caso do caixeiro viajante, a função ficou $1/custodoestado$, e, para o N-Rainhas a função ficou $1/custodeestado + 1$. Assim, os valores de *fitness* concentram-se entre 0 e 1.

Adicionalmente, foi adicionado um critério de paragem para o problema de N-Rainhas, visto que se sabe que o melhor estado possui *fitness* igual a 0.

As seguintes figuras (figs. 39, 40, 41, 42) representam a implementação dos objetos descritos.

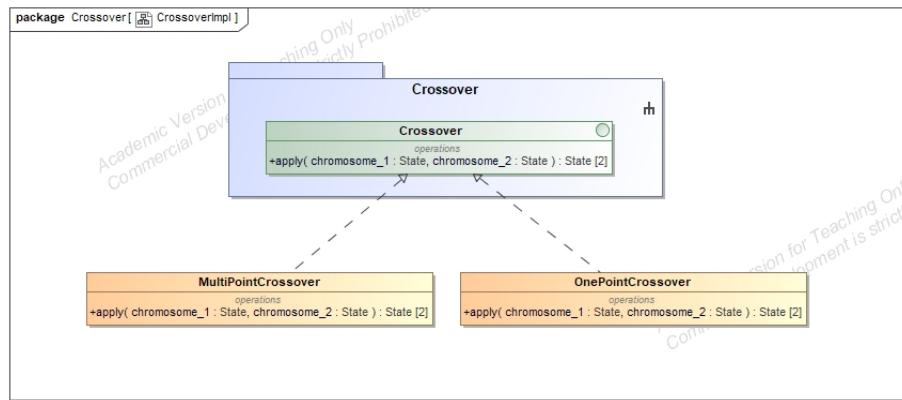
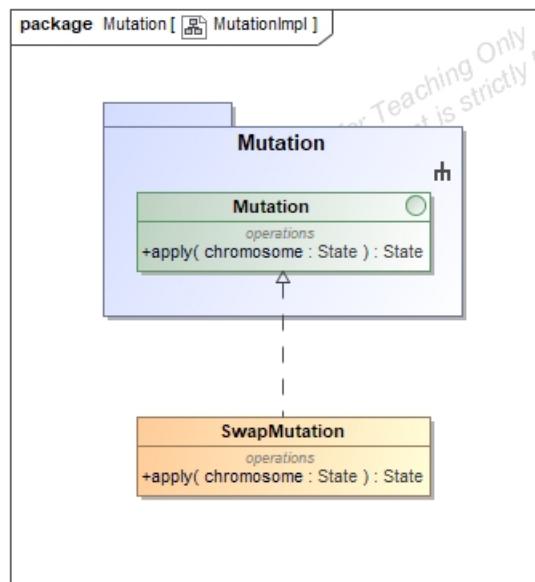
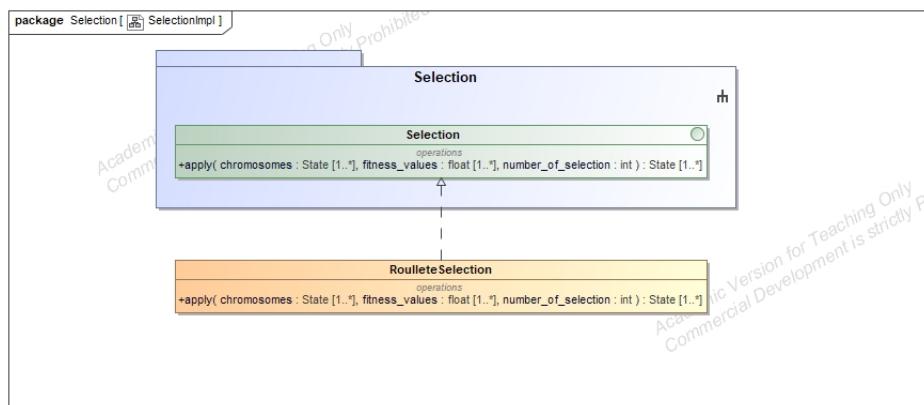
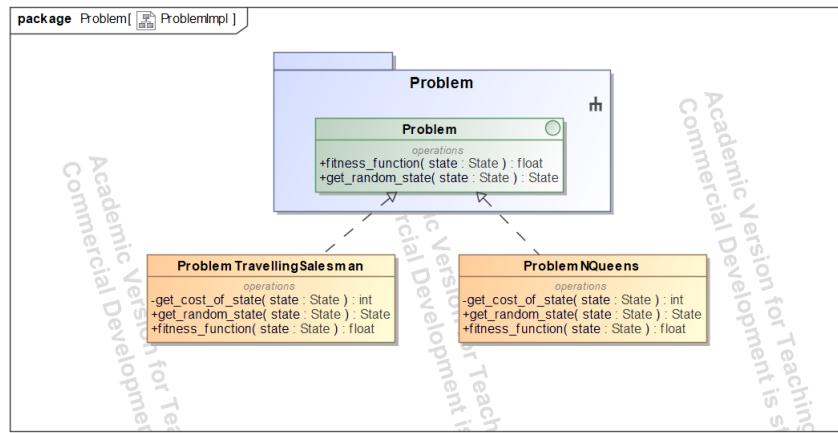


Figure 39: Diagrama UML - *Crossover*

Figure 40: Diagrama UML - *Mutation*Figure 41: Diagrama UML - *Selection*

Figure 42: Diagrama UML - *Problem*

Resultados As figuras (fig. 43 e 44) representam as resoluções que o algoritmo encontrou no problema do Caixeiro Viajante e das N-Rainhas, respectivamente.

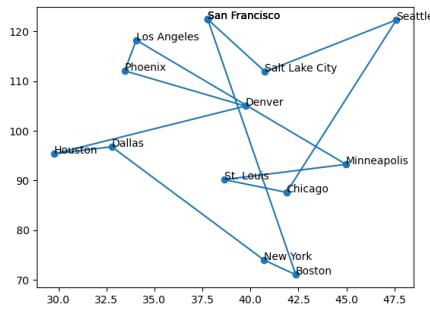


Figure 43: Caixeiro Viajante - Resultados

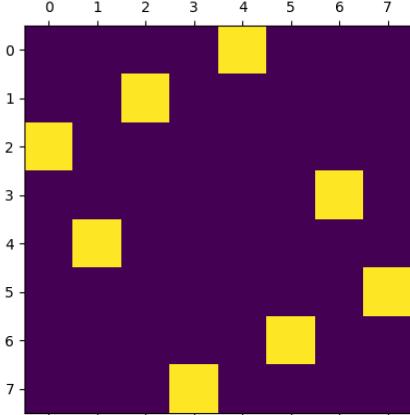


Figure 44: N-Rainhas - Resultados

4.2.2 Implementação por biblioteca externa

Como termo de comparação, foi utilizada uma biblioteca externa, chamada *Pyeasyga*, para resolver os problemas que foram resolvidos pela biblioteca criada no projeto.

Os resultados são os seguintes. Observe-se que são resultados bastante melhores que os retirados acima. Isto pode-se dever a uma série de fatores, mas nomeadamente com o facto da biblioteca poder estar a fazer uma série de operações, tal como o Elitismo, que mantém os melhores cromossomas de cada geração. É natural que os resultados sejam melhores, assim como mais eficientes a correr.

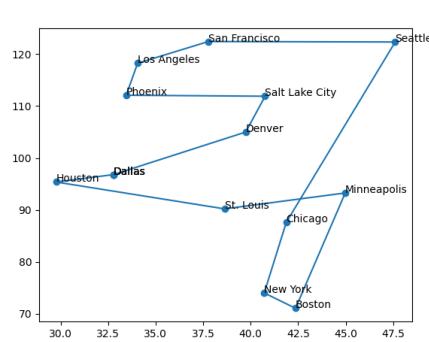


Figure 45: Caixeiro Viajante - *pyeasyga*

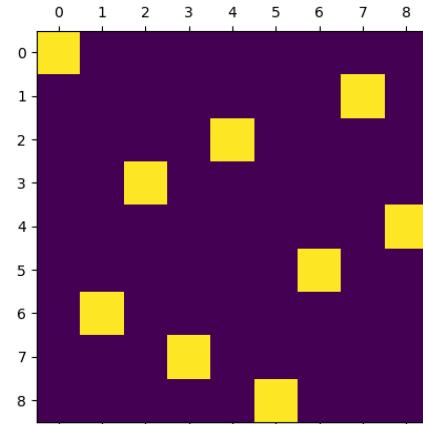


Figure 46: N-Rainhas - *pyeasyga*

5 Projeto - Objetivo 3

Este objetivo pretende a realização de um agente capaz de navegar num espaço de dimensões discretas, com obstáculos e um alvo, desviando-se dos obstáculos e recolhendo o alvo, tendo por base:

- O algoritmo **Wavefront**.
- O algoritmo **RTAA***.
- O algoritmo **Dyna-Q**.

O problema foi dividido nos seguintes constituintes:

- **Algoritmos.**
- **Planeador.**
- **Agente.**
- **Representação gráfica do agente.**

Neste objetivo mantém-se os termos de operador e estado. Relembrando que um operador é aplicado a um estado de forma a gerar um novo estado.

5.1 Algoritmos

A primeira parte consiste nos algoritmos utilizados. Foram construídas duas bibliotecas, uma para procura em espaços de estados e, outra para aprendizagem por reforço.

5.1.1 Procura em espaços de estados

A biblioteca de procura em espaços de estados contém os algoritmos *Wavefront* e RTAA* assim como outros tal como a procura A*, apenas utilizada para auxiliar no desenvolvimento do RTAA*.

5.1.1.1 *Wavefront*

O algoritmo *Wavefront*, foi incrivelmente simples de implementar, sendo que apenas faz uso de uma classe (*Wavefront*). Esta mesma classe, como se pode ver a seguir (fig. 47) tem o método *run_algorithm()* que corre o algoritmo, retornando um dicionário com os valores de cada estado, para todos os estados do mundo.

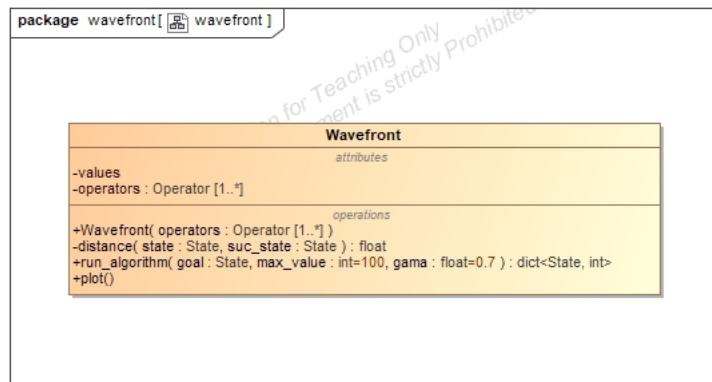


Figure 47: Classe de implementação de *Wavefront*

5.1.1.2 Real Time Adaptive A*

O desenvolvimento do algoritmo RTAA* foi deveras mais complexo. Começou-se por criar uma base que suportasse a procura A*, na qual a RTAA* se baseia.

Para tal foi utilizado um mecanismo de procura (classe *SearchEngine*), que utiliza memória (classe *SearchMemory*), para navegar num espaço de nós (classe *Node*) associados a estados e operadores (classes *State* e *Operator*, figs. 48 e 50).

Podem existir vários tipos de memória, pelo que a procura A* e RTAA* exigem uma memória prioridade (*Priority Memory*, figs. 49). Esta memória retira elementos da mesma de acordo com a sua prioridade.

Mais uma vez o conceito de problema está presente outra vez, aqui, o mesmo guarda a função heurística, assim como o estado inicial e final (fig. 51).

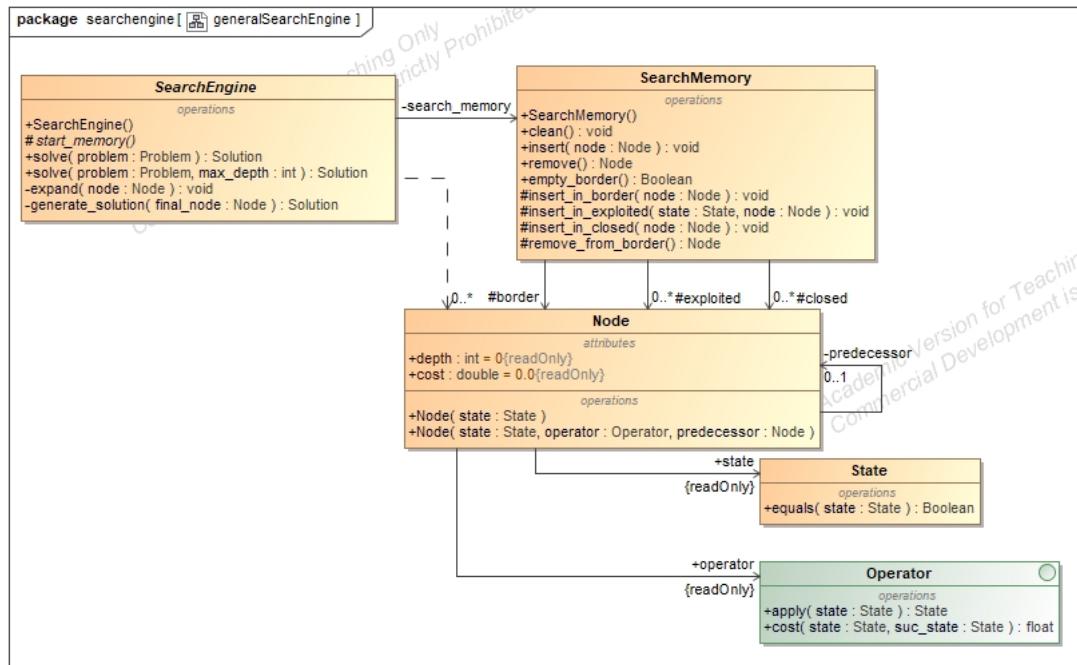


Figure 48: Diagrama UML - Mecanismo geral de procura

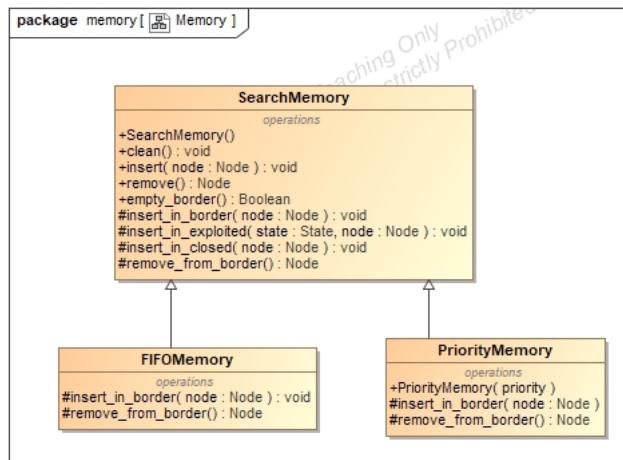


Figure 49: Diagrama UML - Mecanismo de memória

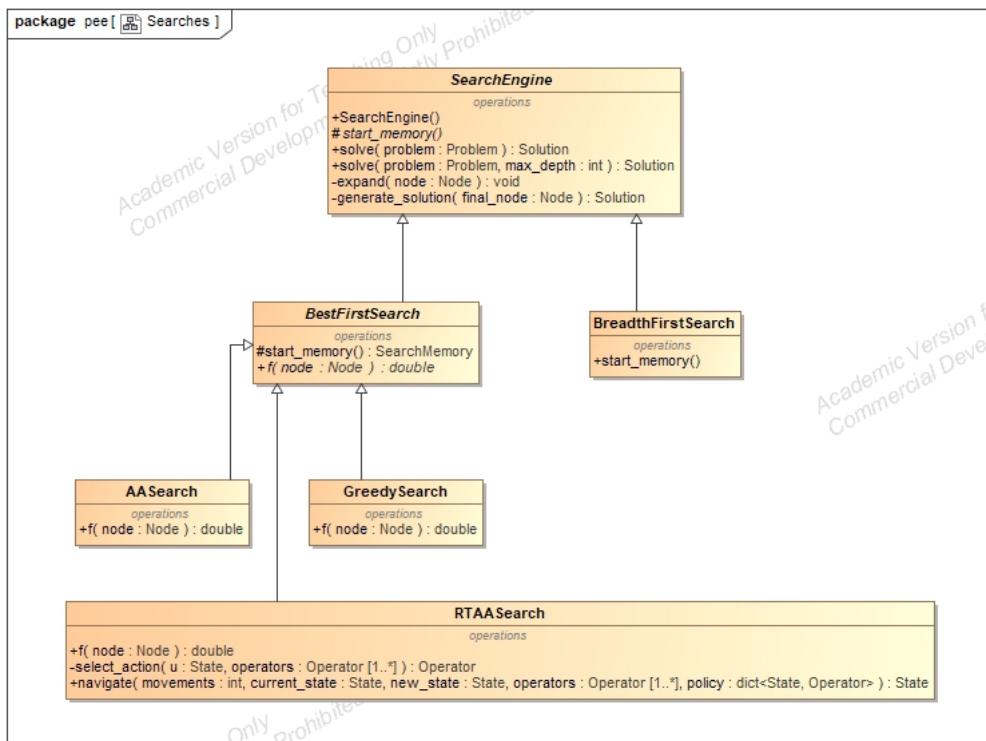


Figure 50: Diagrama UML - Mecanismos específicos de procura

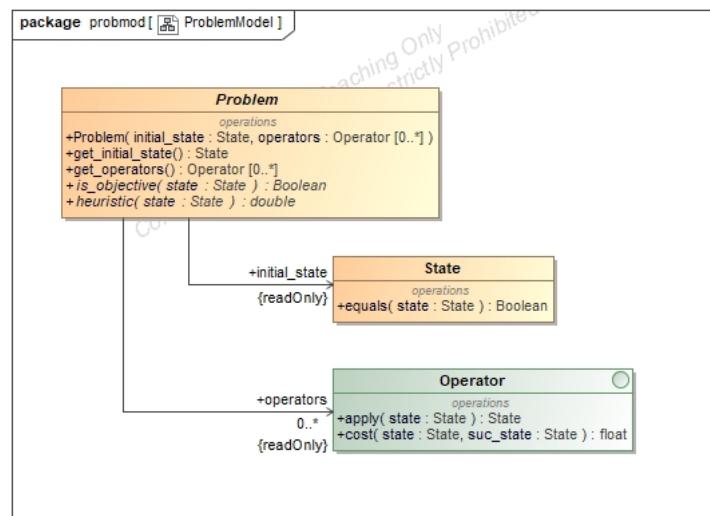


Figure 51: Diagrama UML - Definição do problema

5.1.2 Aprendizagem por reforço

A aprendizagem por reforço contém o algoritmo *Dyna-Q* e outros como auxiliares como o mesmo pode ser construído a partir do clássico *Q-Learning*.

A base do algoritmo é a classe *LearningRef*, uma classe abstrata apenas com o método *learn()*. Todos os algoritmos estendem esta mesma classe.

O objetivo aqui, é calcular os valores de cada estado que o agente já passou sempre que uma ação é realizada pelo agente. Esta "tabela" de valores do mapa encontra-se na memória esparsa (*SparseMemory*). O processo de aprendizagem, ou atualização dos dados em memória dá-se no objeto *DynaQ*, com a ajuda do ModeloTR (classe *TRModel*).

As seguintes figuras (figs.), representam a implementação do algoritmo *Dyna-Q*.

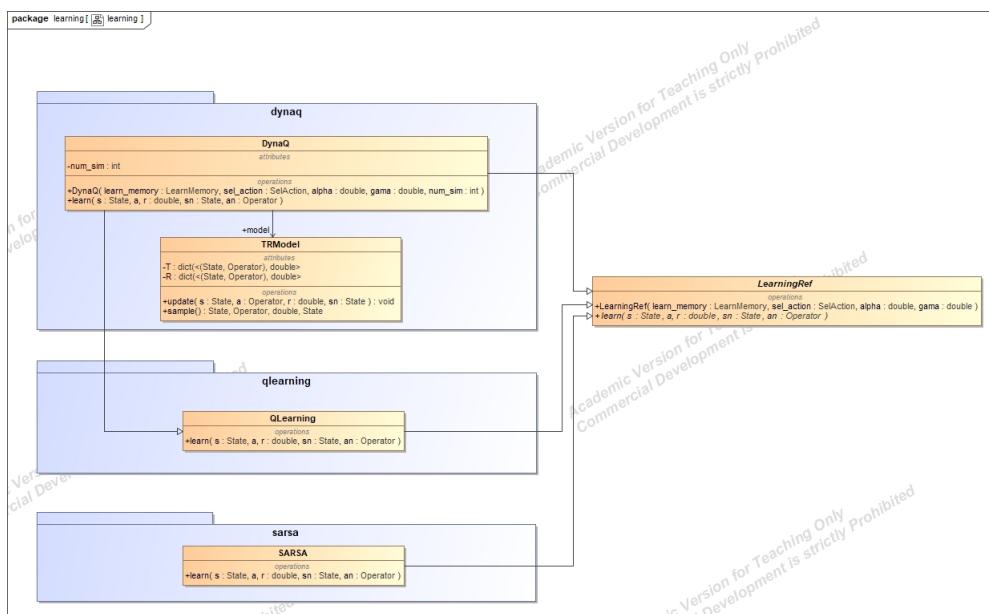


Figure 52: Diagrama UML - Algoritmos de aprendizagem por reforço

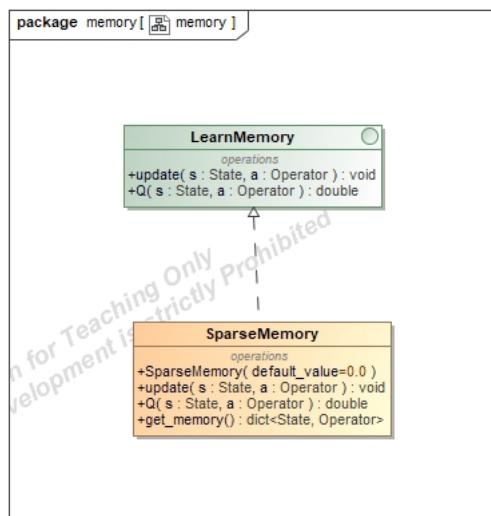


Figure 53: Diagrama UML - Mecanismo de memória

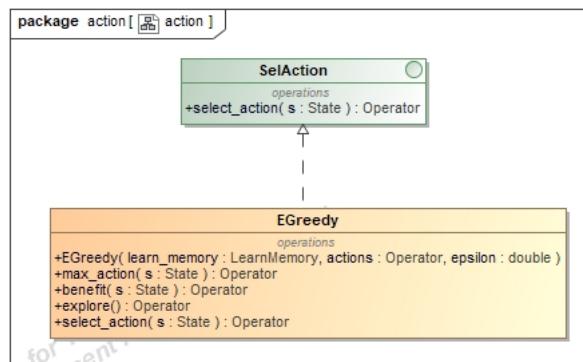


Figure 54: Diagrama UML - Mecanismo de seleção de ações

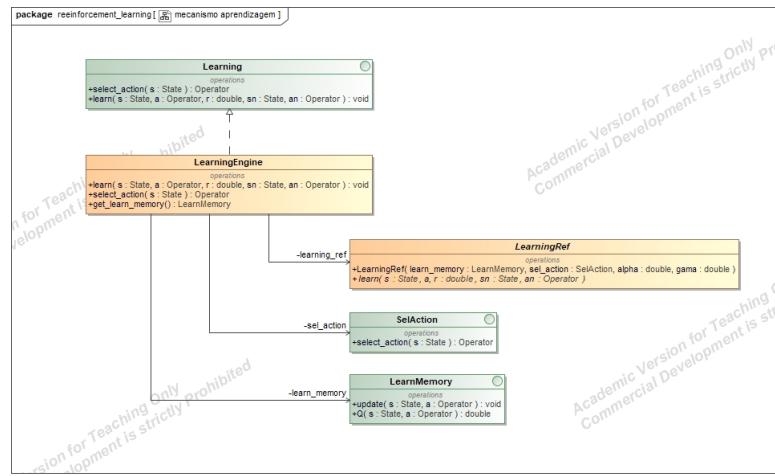
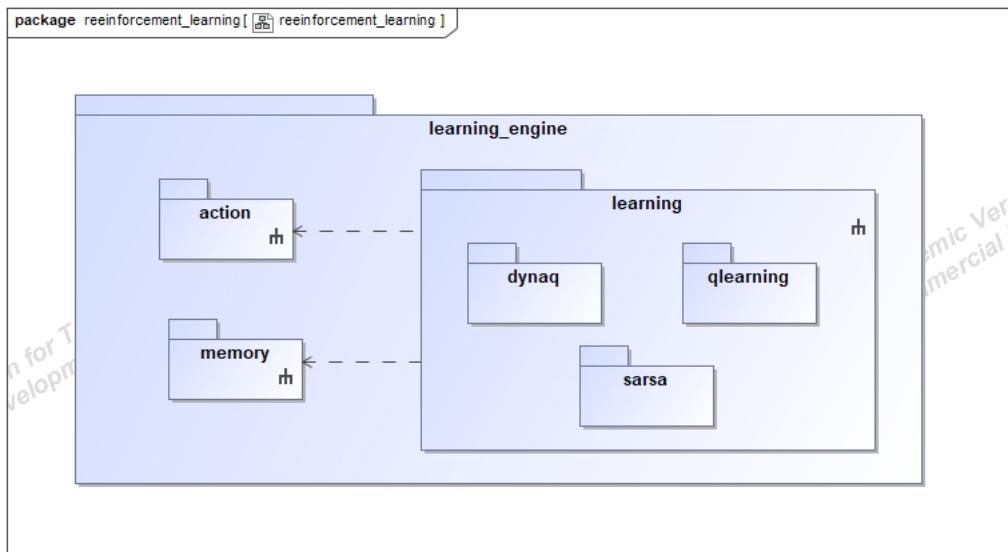


Figure 55: Diagrama UML - Mecanismo de Aprendizagem

A organização dos *packages* tem o seguinte aspecto (fig. 56):

Figure 56: Organização dos *packages*

5.2 Planeador

Com os algoritmos desenvolvidos, segue-se para o domínio do problema. Para isso, resolveu-se criar planeadores. Um planeador planeia um trajeto através de um dado algoritmo. É no planeador que se especificam os operadores, problema, estado e mundo.

A seguinte figura (fig. 57), procura representar o estado em concreto (classe *PlanState*), o operador (classe *MoveOperator*), neste caso o operador é o que faz o agente andar (verticalmente, horizontalmente e nas diagonais), e, a definição do problema e o modelo do mundo, que armazena toda a informação observada no mundo.

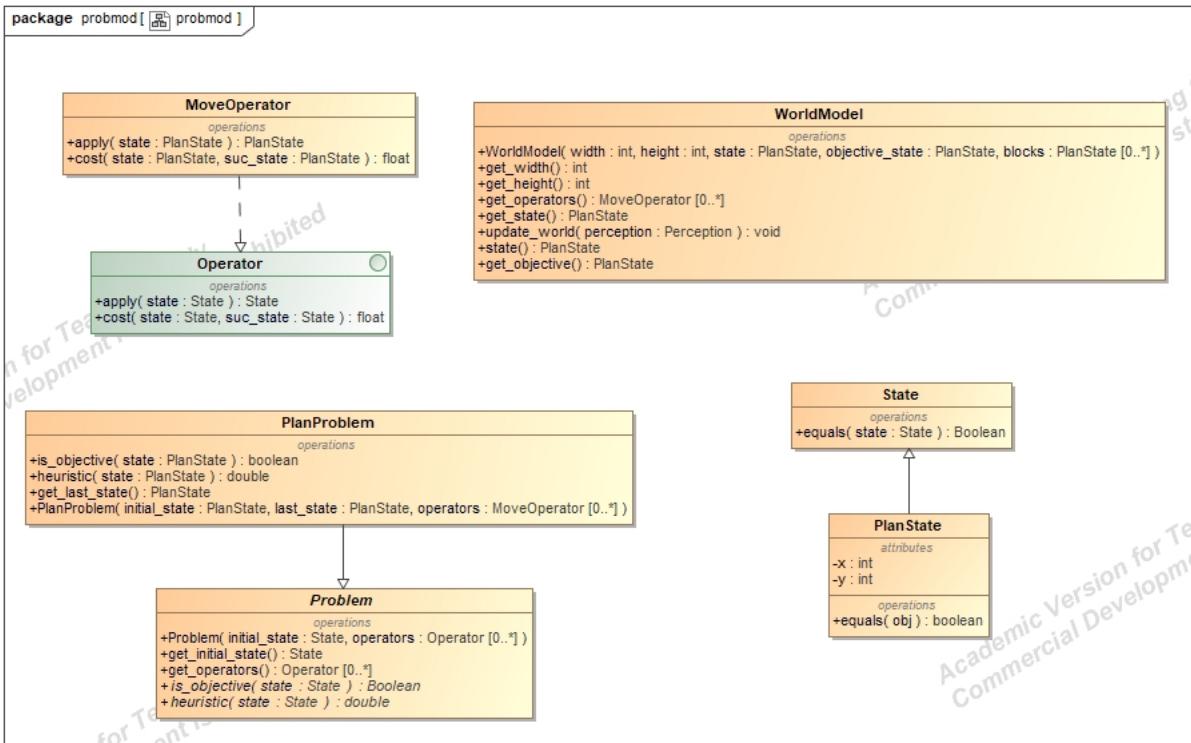


Figure 57: Diagrama UML - Modelo do problema

Por fim, o diagrama em seguida mostra as classes para os algoritmos a resolver. É de notar que a aprendizagem por reforço não utiliza planeador. Isto pois a mesma não calcula um trajeto mas sim tenta sempre chegar ao objetivo.

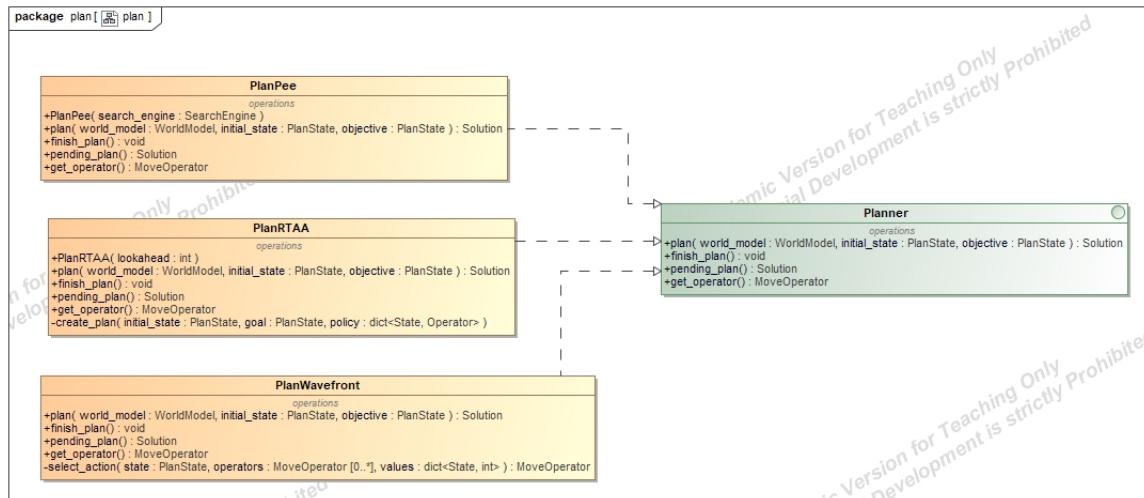


Figure 58: Diagrama UML - Planeadores

5.3 Agente

O agente é que corre o algoritmo (independentemente de qual seja). Foi desenvolvido um agente prospetor (classe *Prospector Agent*) que, através de Controlo (interface *Control*), consegue reagir à resposta do algoritmo.

Como se verificou no subcapítulo anterior, a aprendizagem por reforço não possuí qualquer tipo de planeador. Isto faz com que, o controlo do agente quando o mesmo necessita de utilizar aprendizagem por reforço seja outro. Foram criados dois tipos de controlo, um controlo deliberativo (classe *DeliberativeControl*) e um controlo de aprendizagem de reforço (classe *LearnRefControl*).

Para possibilitar ao agente a observação do que o rodeia, foi criado um sensor (classe *Sensor*), que capta percepções (classe *Perception*) que o agente tem. Com estas percepções o mesmo pode atualizar o modelo do mundo que tem em memória.

O agente é também capaz de andar pelo mundo através de um atuador (classe *Effector*), que o deixa aplicar um operador à posição onde se encontra.

É no controlo que é atualizado o mundo através das percepções, sendo também tomadas as decisões sobre o que o agente tem de fazer.

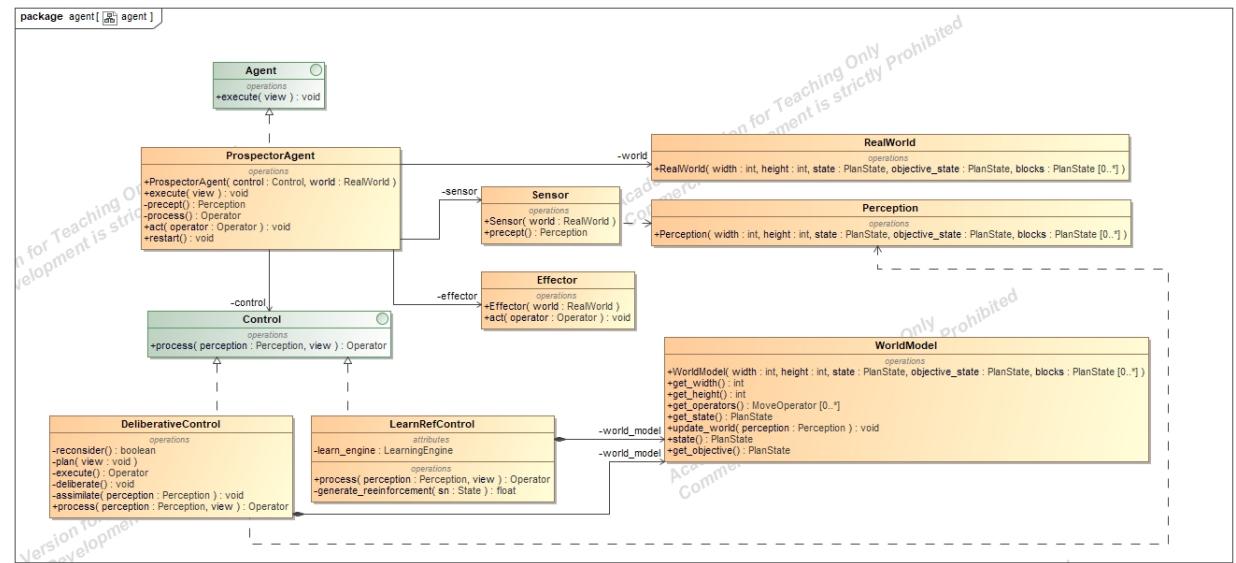
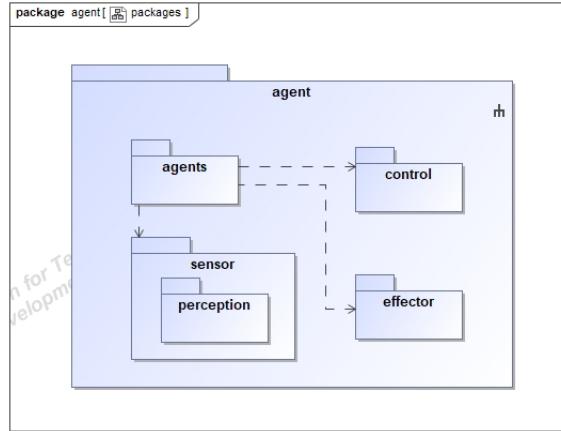


Figure 59: Diagrama UML - Arquitetura de agente

A estrutura da arquitetura do agente é a seguinte (fig. 60):

Figure 60: Organização dos *packages*

5.4 Representação gráfica do agente

De forma a representar graficamente o agente foi utilizada a biblioteca *Pygame* presente na linguagem de programação *Python*. O uso da mesma deve-se ao facto de já ter sido abordada noutras unidades curriculares.

Seguem-se algumas das representações em diferentes ambientes (fornecidos pelo docente), para o algoritmo *Dyna-Q*, *Wavefront* e, RTAA*, respetivamente.

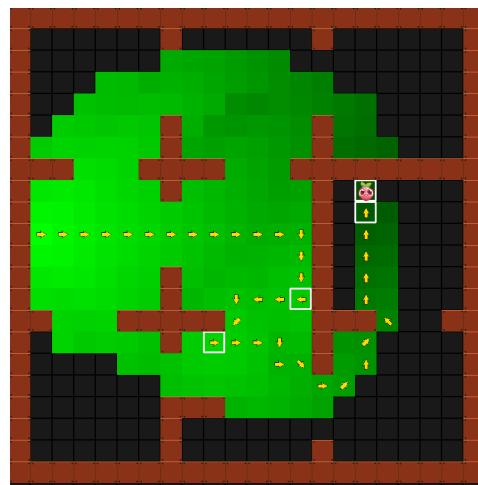


Figure 61: Representação gráfica do RTAA*

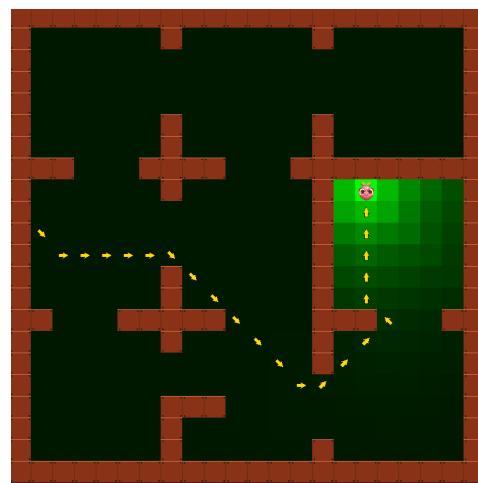
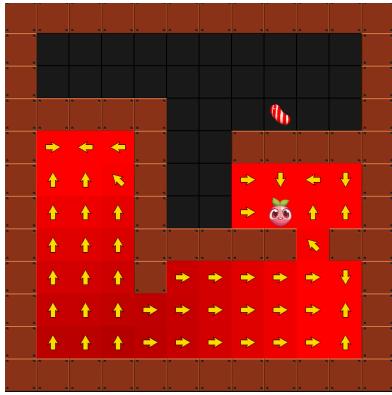
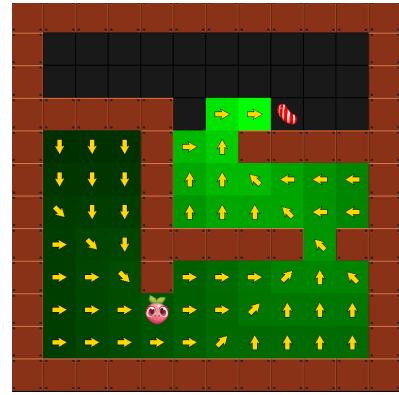


Figure 62: Representação gráfica do *Wavefront*

Figure 63: *DynaQ* em aprendizagemFigure 64: *DynaQ* após aprendido o ambiente

O algoritmo ***Dyna-Q*** contém a função valor de cada estado representada como a cor de cada estado. Caso tenha valor negativo o estado encontra-se vermelho e, caso seja positivo, verde. É possível também ver a política do agente, isto é, o sentido que em cada estado o mesmo deve tomar (tendo em conta o melhor reforço) representado como setas.

No algoritmo ***Wavefront*** é também possível verificar as funções de valor de cada estado, sendo que neste caso apenas é representada com diferentes tons de verde. Neste não está presente a política mas sim o caminho tomado pelo agente (com as setas).

Por fim, o algoritmo ***RTAA**** contém em si também a função valor, todavia, este algoritmo não pesquisa todos os estados necessariamente, sendo que apenas contém a função valor para os estados explorados. Estes são também demonstrados através de diferentes tons de verde. A caminho tomado para atingir o objetivo é também demonstrado, outra vez com ajuda das setas amarelas e, adicionalmente, o estado final de cada sub-percurso do algoritmo tem uma borda branca, conseguindo o espetador ver o caminho tomado.

6 A presença de inteligência artificial na Web 3.0 e no Metaverse

Olhando para trás alguns anos, bastantes foram os aspectos das nossas vidas que mudaram. Ainda ontem toda gente assistia séries na televisão, onde apenas atores, apresentadores e entrevistadores eram os protagonistas. Se calhar a *internet* continha menos anúncios porque o *marketing* era feito em revistas e cartazes. Parece que já caminhámos tanto e não há muito mais por onde ir, mas todos os dias são dados mais passos no desenvolvimento tecnológico.

Um pilar das nossas vidas que, hoje em dia muitas pessoas não conseguem sequer viver sem isso, é a *internet*. Não conheço ninguém que não faça uso da *internet* pelo menos uma vez por dia. A *internet* foi algo que se estranhou no início, mas o ser humano rapidamente se moldou a ela e, hoje muitos não sabem o que fariam sem ela.

Este pequeno artigo pretende uma visão ampla da importância do ramo da inteligência artificial no que diz respeito à Web 3.0 e *Metaverse*.

Tudo começou com a Web 1.0, mesmo no início da *internet*, pode-se mesmo chamar da primeira iteração da *internet* na década de 90. A *internet* servia apenas como forma de obter informação. Não existiam redes sociais, jogos na *internet*, nada. Apenas páginas estáticas com informação. A web era meramente uma enciclopédia, com bastantes páginas com apenas conteúdo textual, informação útil, mas com nenhuma forma de prazer ao ser humano.

Desde o início dos anos 2000 deu-se a nova era da *internet* inicializada, a web 2.0. Como é possível imaginar, foi aqui que se começou a investir em formas de manter as pessoas "coladas" aos ecrãs. Agora era possível não só ler artigos, mas também ver vídeos, partilhar fotografias com amigos, "gostar" de publicações, comentar, etc. Começou a aparecer comércio *online* (*e-commerce*) e *influencers* de redes sociais. A web 2.0 dominou completamente o mundo e, de certa forma ajudou no desenvolvimento de dispositivos móveis atuais, sendo estes todos moldados a *internet*.

Tudo começou com uma ”grande enclopédia”, até as pessoas conseguirem tirar prazer da *internet*. E agora? Com o aumento do uso de tecnologias na área de Inteligência Artificial e *Blockchain*, a *internet* vai evoluir ainda mais.

O futuro da *internet* é uma plataforma massivamente grande, persistente, interativa e interoperacional em tempo real, composta por mundos virtuais interconectados onde as pessoas podem trabalhar, jogar, criar e socializar.

Foi-nos introduzido recentemente, pela antiga empresa *Facebook* (agora chamada *Meta*), o conceito de *Metaverse* juntamente com *Blockchain*.

O *Metaverse* é eventualmente a evolução longínqua da Web 3.0. Uma versão da web que mantém o controlo da mesma descentralizado.

É um universo completamente virtual, onde utilizadores vivem uma vida, mas num universo diferente. Toda gente poderia fazer tudo o que faz na vida normal, mas no *Metaverse*, com moedas virtuais (criptomoedas). O ponto é misturar o mundo real com o mundo virtual, de tal forma que não haja distinção.

A figura (fig. 65) representa uma possível estrutura de camadas presentes no *Metaverse*.

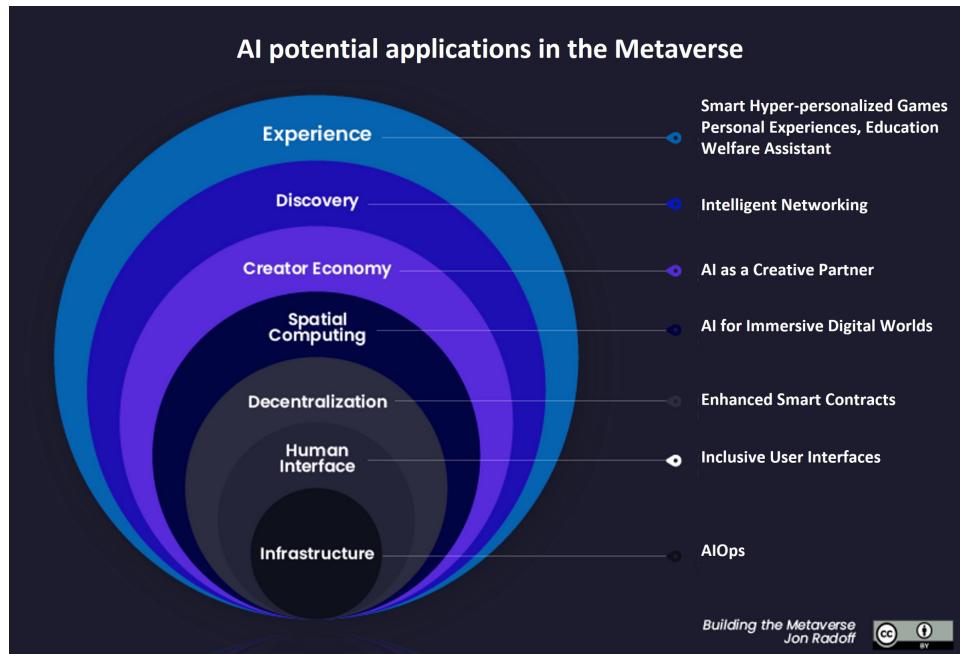


Figure 65: Estrutura do *Metaverse*

O núcleo tem o nome de **AIOps**. AIOps é a combinação de *big data* e *machine learning* de forma a automatizar processos de TI, incluindo deteções de anomalias e determinação de causalidade. Esta camada é chave para a robustez da infraestrutura do *Metaverse*.

Interfaces de utilizador inclusivas são também uma camada bastante importante, trazendo o conceito de Inteligência Artificial para área da acessibilidade. Pessoas que se calhar não estão muito acostumadas às tecnologias devem conseguir fazer uso do *Metaverse*. Algumas das tecnologias nesta camada poderiam ser:

- Reconhecimento de imagem para pessoas com desabilitações visíveis.
- Tradução automática.
- Exoesqueletos inteligentes para interagir com o mundo virtual

A categoria ***Spatial Computing*** é o uso de Inteligência Artificial para o desenvolvimento de mundos virtuais. Uma ferramenta que já implementa este conceito a níveis muito elevados é

o *Omniverse* da empresa *NVIDIA*. O *Omniverse* contém um conjunto de ferramentas que ajudam na criação de ambientes virtuais, animações por voz, etc. É realmente impressionante a capacidade de tecnologia que a plataforma *Omniverse* da *NVIDIA* possuí.

Creator Economy tem a ver com o uso de Inteligência Artificial como um companheiro criativo. Já existem implementações de assistentes de Inteligência Artificial que geram texto como se fosse um humano a escrever, como por exemplo o GPT-3.

A ideia de **Intelligent Networking** é verificar como é que a Inteligência Artificial pode melhorar a capacidade social do *Metaverse*. Alguns dos aspetos já se encontram em uso em redes sociais atuais como por exemplo a prevenção de discursos de ódio, ou detetar publicações explícitas.

A camada mais superficial é a **Experiência**. Construindo um universo destes, camada a camada como descrito, são explorados os limites da inteligência artificial (caso haja algum). Esta camada é a mais externa mas procura mostrar que o este novo universo deve fornecer uma experiência única a todos e quaisquer utilizadores desta nova plataforma. O que vem a fornecer esta experiência tão personalizada é um conjunto de várias tecnologias, mas todas nelas presente o inteligência artificial.

Esta análise às possíveis constituintes do *Metaverse* alerta para o facto de poder chegar um mundo virtual irresistível para o ser humano.

Uma pergunta bastante importante que o tópico traz é, será é possível que este mundo virtual se torne indistinguível do real? Será que é possível uma pessoa trocar completamente a sua vida pessoal por uma no *Metaverse*?

Algo que nos permite sempre perceber se estamos a jogar um videojogo, é sempre a inteligência artificial dos personagens. Os detalhes em videojogos atuais já são bastante semelhantes ao real e, através de uma fotografia pode até ser indistinguível. Apenas quando o utilizador interage com NPC's (*Non Playing Characters*) é que se apercebe que está num jogo. Nada do que lá está a jogar é real, os personagens têm todos um *script*. Noutras palavras, ex-

iste sempre um limite para o que um agente num videojogo consegue fazer. Se calhar alguns personagens só têm a mesma fala, outros apenas dois ou três estados.

Esta análise ao *Metaverse* abre os olhos às diferentes descobertas no campo de inteligência artificial que já existem. Um universo com uma combinação dessas técnicas pode nunca chegar a ser indistinguível do real, mas vão sempre haver pessoas que, mesmo sabendo que é um mundo virtual, o querem viver, ao invés do real a que somos expostos todos os dias.

Com dados mesmo de hoje em dia é possível concluir que já existem pessoas que vivem uma vida apenas virtual. No Taiwan, à uns anos um homem morreu devido a jogar um jogo durante 3 dias seguidos sem qualquer pausa.

Muita gente pergunta se vivemos numa simulação. A meu ver, o problema não é viver numa simulação, mas sim estar viciado na simulação. O homem que faleceu por jogar não deveria acreditar que o jogo era real, se calhar foi um mero acontecimento de vício.

O que me preocupa na área de inteligência artificial não é o facto de existirem mundos virtuais indistinguíveis do real, mas sim existirem formas cada vez mais aliciantes, descobertas por este ramo tecnológico, de transmitir ao ser humano a sensação de prazer momentâneo sem qualquer tipo de esforço.

A implementação do *Metaverse* ainda se encontra em construção e, pode levar bastante tempo, mas o que é certo e que vem para ficar. Esta análise permitiu ver tanto pontos positivos como negativos que possam acontecer.

É claro que a evolução a nível tecnológico é importante e, este ramo de inteligência veio para resolver uma vasta gama de problemas sejam físicos ou psicológicos. Mas é sempre estranho pensar o que pode acontecer caso o uso de um ramo como a inteligência artificial, um ramo que ainda contém muito por desvendar, com capacidades extremas, seja utilizado como forma de cativar as pessoas para uma vida não real. Hoje já são utilizados mecanismos de inteligência para recolha de dados, então o que poderá vir a seguir...

7 Conclusão

Concluindo, apesar da complexidade dos objetivos ir incrementando ao longo do tempo, foi-se capaz de manter uma entropia menor, utilizando técnicas como a coesão aprendidas noutras unidades curriculares como Engenharia de *Software*. Caso os problemas não tivessem sido abordados primeiro em alto nível, refinando cada vez mais diagramas UML, a complexidade aumentaria exponencialmente.

Apesar da grande ajuda dos princípios de Engenharia de *Software*, houve alguns percalços pelo caminho.

No **objetivo 1**, na resolução do tema à escolha, inicialmente era para ter sido utilizado um *dataset* com fotografias de cães e gatos, com o propósito de os classificar binariamente. Todavia, as imagens eram demasiado grandes, contendo vários píxeis. Tentando diminuí-las para tamanhos mais razoáveis a rede não chegava a resultado nenhum. Isto pois, ao diminuir o tamanho, a maior parte dos detalhes que descrevem os animais já não eram visíveis nem ao olho humano. Resolveu-se então utilizar o *dataset* das núvens e sóis demonstrado no relatório, mas apenas depois de verificar como se poderia eventualmente classificar imagens daquele género.

Após alguma pesquisa, reparou-se que a melhor forma era utilizar um CNN, uma Rede Neuronal Convulcional. Algo que não chegou a ser abordado nas aulas teóricas muito a fundo. Em geral, a ideia de uma rede destas é retirar apenas as características das imagens, importantes para a sua classificação. Uma Rede Neuronal Convulcional seguiria a seguinte estrutura (fig. 66):

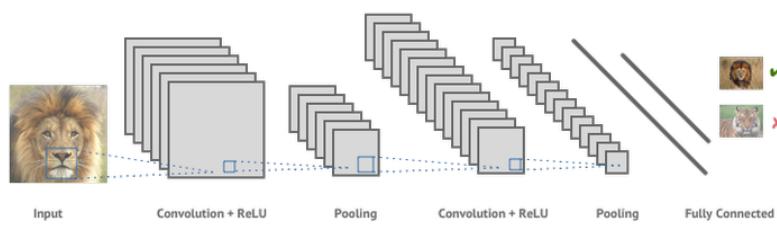


Figure 66: Estrutura de uma Rede Neuronal Convulcional

No **objetivo 3**, no desenvolvimento da arquitetura do algoritmo *Wavefront* pensou-se inicialmente que, como este é uma variante da procura em largura, se podia apenas estender a classe de Procura em Largura (*BreadthFirstSearch*) e, utilizar os mesmos conceitos mas fazer a pesquisa do objetivo para o agente. A implementação do método desta forma gerou muito mais complexidade, desnecessária. Assim, voltou-se à arquitetura, estruturou-se de outra forma e, com o algoritmo de *Wavefront* isolado da procura em largura implementada (*BreadthFirstSearch*), a implementação do mecanismo foi simples e super eficaz, assim como de mais fácil leitura.

Em boa verdade, o projeto abriu os olhos para uma extrema variedade de coisas que a inteligência artificial pode fazer. A pesquisa sobre o tema da *Web 3.0* e do *Metaverse*, complementou bastante bem o conteúdo lecionado na unidade curricular. Foi possível compreender o que realmente já se encontra em uso e, o que podemos esperar de tecnologias presentes neste ramo.