VitaLiberu, Lda.

Electronic Live Pets Art

Formica

Sistemas Livres

Inteligência Natural - Cérebro

Lisboa - Portugal, 2024

Índice

1 “ABSTRATO”

Será que não gostaríamos, todos nós, de ter um robot que nos fizesse companhia, conversasse, ajudasse nas tarefas, jogasse, etc.?! E não gostaríamos de o poder educar à nossa maneira, como se de um filho se tratasse, adquirindo a nossa educação? Melhor ainda, não seria ótimo se o pudéssemos partilhar com outros na educação que lhe dava-mos? De maneira a, pudermos trocar capacidades “robóticas”? Como por exemplo: adquirir um ficheiro dum outro robot para instalar no nosso e este ficar com a capacidade de lavar a loiça?! Ou melhor ainda: prestar ajuda mecânica ao carro empanado?! … enfim, qualquer capacidade ou habilidade imaginável, possível de imitar o ser humano, com as competências tecnológicas atuais. Sensores mais precisos dos que aos do ser humano, como infravermelhos, lasers, visão apurada. Nos motores e movimentos, ainda não igualamos o corpo humano em todos tipos de movimentos.

Se tivermos algum problema com isso podemos sempre mandar os robôs para o espaço sideral, Lua, Marte.

No meio dos cientistas de Inteligência Artificial fala-se de uma aposta de vencer, em futebol, os humanos em 2050. Pois estamos já em 2024 e ainda não estão perto de tal façanha.

Pois, “animem-se”… Estamos mais perto do que pensávamos.

As ideias deste projeto foram crescendo com a necessidade de saber o que é que um cérebro humano precisa para memorizar, procurar, aprender, ou como é a dinâmica do cérebro e seus prazeres.

Para isso acontecer, o robô também necessita de estar “vivo” ou imitar a vida se preferirem. Sem entrar em debates sobre o que é um ser vivo ou um ser inanimado. Precisa-se isso sim, de um “corpo” mecânico e de um “cérebro” elétrico para dar vida ao corpo. O cérebro do robô vai ser uma simulação do cérebro humano, já que é neste que cada um de nós pode fazer introspeção e tirar conclusões podendo imita-lo num robô.

Adaptámos isso às limitações do corpo do robô e aos seus movimentos que ainda limitados, os sensores, os motores e outras capacidades ainda reduzidas que não passam das capacidades de uma formiga.

Apresentamos-lhe aqui, a arquitetura de um sistema de processamento paralelo. Ou seja são vários programas a serem executados em simultâneo e em tempo real, constituindo o cérebro de um robô. Com sensores propriocetivos que são a sensação do próprio corpo. Saber que mexemos um braço, por exemplo.

Explicamos como um programa de Inteligência Natural com certos princípios implementados, deduzidos à medida em que se foi sentido a necessidade deles e introduzir modificações novas baseado e limitado ao que já se conhece sobre o cérebro.

Programa esse, de nome Liberu (vem do Latim, significa Livre) que serve como cérebro a ser executado num computador e um robô mecânico como corpo, comunicando-se por Bluetooth. Foi implementado em 13 classes escritas na linguagem de programação Java e a funcionar com o robô da Lego NXT MindStorm, ou com um robô de tecnologia Arduíno, ou ainda uma simulação de um robô num labirinto, e que ocupa cerca de uns reduzidos 115 KB, tamanho menor que una simples fotografia.

A escrita do programa em linguagem Java segue em anexo.

Quero portanto estudar o cérebro através da robótica.

2 Ideias Contemporâneas

Estávamos num mundo em que os animais e o Homem se encontravam em estatutos desumanamente diferentes. Mesmo entre, os seres humanos que se consideravam existentes de múltiplas capacidades e poderes inatos. Muitos super-heróis foram imaginados para nos libertar temporariamente das nossas indesejáveis limitações. Queríamos arduamente convencer-nos de que uma pessoa teria um destino e um objetivo para a sua vida.

Mas como saber quem merecia ser recompensado pelas suas capacidades e poderes especiais que nos permitiam ser mais ou menos importantes num destino e desígnio de algo ou alguém todo poderoso? Uns chamavam-lhe Deus, outros Buda, Alá, Poder Superior, entre muitos outros súbditos e mensageiros dessas entidades conhecedoras e cientes da nossa missão no planeta Terra.

Seria impensável existirmos só por existir, sem algo que nos fizesse justiça a praticar o Bem, a ser honesto, amigo ou mesmo altruísta, sem uma recompensa numa vida-pós-morte. Quer fosse fugir do inferno a caminho do paraíso, quer fosse todo o tipo de prazeres pós vida.

Assim o Homem, e dentro desta espécie da natureza, alguns mais ou menos elitistas do que outros, merecia ser considerado só “per si”, mais capaz! Quer tenham tido ascendências de poder, riqueza, inteligência, artística ou outros dotes de poderes genéticos para que os destacasse.

Tínhamos então todo o tipo de referências e marcas, nomes e testes, obras e sortes que nos distinguiam. Uns eram doutores, outros tinham um apelido de uma nobre família, ou gestores de uma grande empresa, um alto valor num teste de inteligência ou beleza, descoberto ou realizado algo notável, ou mesmo ter estado no sítio certo com a informação correta. Ou mesmo um pouco de tudo ou ficticiamente de tudo.

Classificávamos o indivíduo pelo que teria sido à nascença pelas conclusões e factos que o mesmo tinha alcançado na altura em que era observado ou avaliado. Se fosse alguém importante era porque já tinha essas predisposições ou poderes consigo. Logo, ou mesmo antes de dar entrada neste mundo.

Se a avaliação era negativa tratava-se sem dúvida de alguém possuidor de alguma deformidade inata e, portanto, não merecedora de atenção e muito menos de compaixão. Seria suficiente o nosso sentimento de pena e fazer-lho querer que nada havia a fazer se não ele próprio aceitar os desígnios da vida. Mesmo se fosse só uma fase descendente. Só poderia ter sido provocada por ele, pois já nasceu para esse tipo de sofrimento. Como que sendo predestinado pelos seus contemporâneos.

Nada mais errado. O destino ninguém o conhece e nós temos a capacidade de introspeção e de avaliação para tentarmos dar o nosso melhor em todas as circunstância. Mesmo que esse melhor não agrade a gregos e a troianos ao mesmo tempo. Os objetivos vamo-los tendo ao longo da vida e cada um tem os seus.

É claro que vivemos em sociedade e como tal, há uma Lei que nos permite não exagerar nas nossas liberdades como indivíduos que somos, todos nós seres vivos. E alteramos o nosso comportamento consoante o ambiente que nos rodeia (software), consoante o nosso corpo (hardware) e consoante foi o nosso passado como ser vivo e que somos capazes de recordar (firmware).

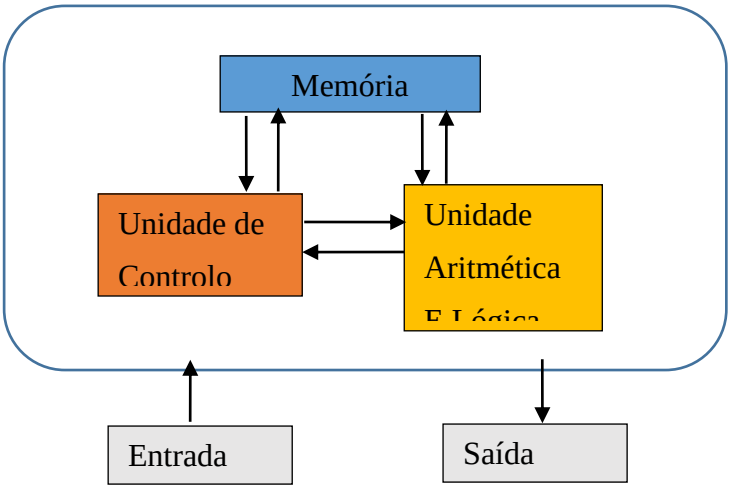
Somente estes 3 fatores influenciam o nosso comportamento, sem nenhuma alma ou entidade endógena a decidir o que o nosso cérebro deve ou não fazer, ele simplesmente faz. Assim sendo, o livre arbítrio não faz parte desta realidade.

Os cérebros dos mamíferos desempenham todos as mesmas funções. O porco tem todos os órgãos do seu corpo transplantáveis para um ser humano o que significa que também o cérebro do porco tenha as mesmas funções das do ser humano. As mesmas emoções e tudo de inteligência. O que difere é o corpo de cada mamífero.

É claro que o cérebro do porco não pode ser transplantado para o ser humano. Devido a não ser possível transplantar cérebros de um ser humano para outro. Porque cortar um nervo da coluna ou de outro sítio equivale a cortar um cabo de fios elétricos ou de fibra ótica com milhares ou milhões de fios cortados e muito difícil ou mesmo impossível voltar a uni-los todos como deveriam estar ligados.

Os computadores atuais, que se dizem com analogias do cérebro. Bem com o programa certo são uma imitação do cérebro, já que este programa –

Liberu está assente no suporte de uma máquina universal capaz de simular tudo e mesmo tudo, que são os computadores. E estes são constituídos por:



Uma Memória onde é guardada a informação em forma de bits 0 ou 1. A um grupo de bits designamos o que significar para o programa que vai à memória usá-los. Na Memória são armazenados as instruções dos programas e a informação em formato de letras, fotografias, música, vídeos e o que se quiser representar em forma de conjuntos de 0 e 1. Um conjunto de 8 bits chama-se Byte. A letra “A” é o Byte 0100 0001 para nós. O número 1 é o Byte 0000 0001 para o computador.

Uma Unidade Aritmética e Lógica onde são efetuados os cálculos aritméticos como a soma, subtração, multiplicação, divisão e as operações lógicas como saber se determinado valor é igual ou superior a outro.

Uma Unidade de Controlo inerente ao sistema, pois é aqui que existe o relógio do sistema e de período em período o sistema avança e são executados as instruções do programa.

E como é lógico um dispositivo de entrada, como o teclado e o rato e um dispositivo de saída como um monitor e impressora

Temos então um sistema genérico para manipular informação à vontade, fazendo executar programas armazenados, também em memória e interpretados pela unidade de controlo.

Ora este sistema em comparação com o cérebro, só tem a memória. Pois o nosso cérebro não tem uma unidade aritmética e lógica no interior. Não

faz cálculos como essa unidade. Faz cálculos por associações de ideias e através duma tabuada decorada. E a unidade de controlo não é igual ao do cérebro. Assim só resta a memória em comum com o cérebro.

No que respeita à Inteligência Artificial as ideias contemporâneas eram compostas pelas seguintes áreas:

Inteligência Clássica – Lógica; “Deductive reasoning”; “Expert Systems”; “Case-based reasoning” e “Simbolic machine learning systems”

Inteligência Computacional -

- Redes neuronais artificiais;
- Computação evolucionaria;
- Inteligência de enxames;
- Sistema imunitário artificial
- Sistemas “Fuzzy”

Nos sistemas clássicos as saídas são em função das entradas, ou seja, nas redes neuronais “Black Propagation” as entradas geram saídas em que as saídas comparadas com um objetivo se obtém um “erro” que se tenta minimizar com o ajuste dos “pesos” das ligações entre os nós da rede.

Nos Algoritmos genéticos geram-se saídas para se escolher as melhores entradas, e pelo algoritmo de seleção, para gerar melhores saídas.

Estas áreas de investigação tinham poucas ligações com as capacidades neurológicas do cérebro real. Faltava algo mais elementar.

No nosso caso temos uma enorme versatilidade e simplicidade, como iremos verificar:

- Permite representar qualquer rede ou ligações continuas entre nós
- Não há códigos nem padrões nos dados
- As instruções estão nas ligações
- Não temos a procura por padrões
- Não temos “pesos” nas ligações

Ou seja, não temos endereços, nem sistema de encaminhamento, nem padrões de dados, e nem procura por padrões.

Neste sistema, primeiro geramos saídas, para depois as associar com as entradas isto é: “Eu não sei como mover, ou o que tenho de fazer para mover, o braço… no passado tive consciência de como consegui mover o braço depois de o ter movido primeiro”.

A grande novidade é que “Nós não dizemos o que fazer, pois não temos objetivo, nós informamos e ouvimos”.

Ou como disse o presidente, dos Estados Unidos, Thomas Jefferson (1743-1826) “Se eu tivesse que escolher entre um governo sem jornais ou jornais sem governo, eu não hesitaria, um momento, em escolher o último”.

3 Motivação

Pois bem, todos nós, pessoas ou animais e mesmo plantas temos algo transcendente a nós próprios… isso é a vida.

O facto de nós estarmos vivos já é algo grandioso. É o que faz de nós especiais. O facto de sermos todos diferentes torna-nos únicos. O que faz de nós humanos especiais é o facto de termos o polegar destacado que nos permite agarrar coisas como o nosso “primo” macaco; o facto de termos cordas vocais que nos permitem ter uma linguagem completa de comunicação e por último, mas não menos importante, o facto de transmitirmos elevada informação de geração em geração, traduzindo-se assim numa enorme cultura e o poder de cada um de nós ter acesso a uma vasta aprendizagem que nos permite, não só não começar do zero em termos de conhecimentos de toda a humanidade, mas também de podermos acrescentar algo a essa cultura. A escrita permitiu-nos elevar o nosso culto e deverá fazer de nós umas pessoas melhores para com todos os outros. Sejam eles seres humanos ou animais, plantas e todo o planeta em geral, visto só termos um planeta.

O que me motivou a escrever este livro foi que depois de provar que os seres vivos não têm livre arbítrio, estão como que à deriva entre o ambiente, o seu corpo e o seu passado. Não têm livre escolha de fazerem o que querem sobre o ambiente e seu corpo. Essa escolha é determinada também pelo passado do ser vivo com cérebro. Nem os seres vivos têm por assim dizer um objetivo na vida. Todos os objetivos com que os seres vivos se deparam passam por, em última análise, pela satisfação do próprio corpo. Apesar disso são responsáveis pelos seus atos. Porque a cultura atual impõe limites à atuação dos seres vivos. É uma maneira de lutarmos por melhores condições de vida e de se propagar o que nos faz bem ou seja o que nos faz sentir bem naturalmente. E o que é bom para uns pode afetar outros negativamente. Deve-se, portanto, à necessidade de existirem regras em formato de Lei, a que nos devemos adaptar e respeitar pois são feitas com o intuito do bem comum.

4 “Estado da Arte”

Comecemos pelo que já sabemos. E o que sabemos sobre o cérebro humano?! Sabemos que tem muitos neurónios e que se comunicam entre si através das dendrites, axónios e sinapses. Há neurónios dedicados às entradas de informação (audição, por exemplo), outros dedicados à saída de informação ou de ação no corpo (músculos, por exemplo) e ainda outros neurónios dedicados às interligações entre os dois primeiros.

Sabemos também que o cérebro tem vontade própria, E não disse que tinha livre arbítrio, já que não acredito nesta última capacidade do cérebro, pois requer uma “alma”. O cérebro evita as situações de “dor” e procura “fontes de energia”, excluindo outras coisas que sabemos que o cérebro faz, mas, que não nos interessa neste caso por se tratar de um robô, como por exemplo: a procura de parceiro para acasalamento, comer/beber, respirar…

E sabemos também que o cérebro tem tendência para aprender com os erros do passado.

Além das duas capacidades mais evidentes que o cérebro faz e que são: fazer mexer o corpo e “sentir” o ambiente.

Para que as necessidades do corpo sejam alcançadas, é necessário aplicar princípios de “senso comum” básicos, tais como: lembrar-se (se possível) do caminho de volta à fonte de energia. Que os movimentos são dados em função do tempo. Que este, o tempo, não anda para trás, só para a frente. O senso de termos tendência a encontrar as coisas de que nos lembramos.

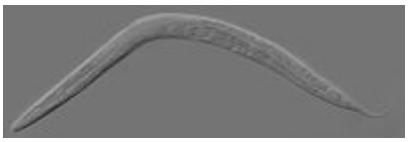
Por fim, e para o que nos interessa, o cérebro humano tem emoções que nos permitem detetar estados específicos do sistema, em nós, e nos outros. Estados internos de consistência ou inconsistência, que nos permitem, assim, comunicar melhor, uns com os outros, pois reconhecemos melhor o valor favorável ou de dor dado aos outros devido às nossas ações.

Eis uma tabela de animais com o seu correspondente número de neurónios:

Nome (Neurónios no cérebro/todo o sistema nervoso)

Formiga (10.000 Varia por espécie,

Na realidade:10.000-100.000)



Caenorhabditis elegans (302 Adulto)

Gato (300.000.000 córtex cerebral)

Barata (1.000.000)

Chimpanzé (6.200.000.000

Córtex cerebral)

Cão (160.000.000 córtex cerebral)

Elefante (200.000.000.000)

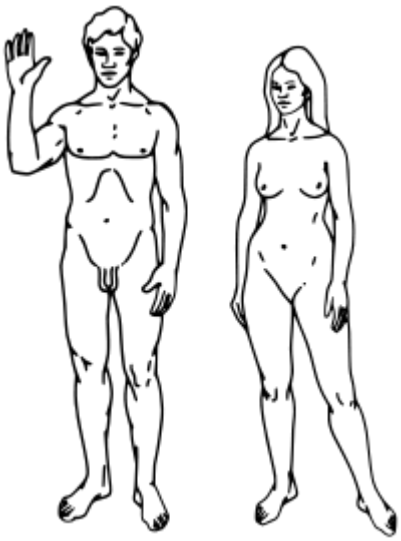
Sapo (16.000.000)

Mosca da fruta (100.000)

Abelhas (850.000)

Humano (100.000.000.000 total

11.000.000.000 (córtex cerebral)



para um adulto médio [8]

Rato (4.000.000 córtex cerebral)

Polvo (300.000.000)

Caracol (11.000) Comum

Ratazana (15.000.000 córtex cerebral)

Sea Slug (7.000 Espanhola)

Esponja (0)

Trichoplax (0)

Baleia (200.000.000.000)

Peixe Zebra (embrião 10.000)

4.1 ***Suposições***

Uma suposição é que os “sentidos” são informações do mundo dadas ao robô através de sensores elétricos específicos para cada “sentido”. Cada sensor envia uma estrutura de 3 dados (Classe Signal do programa em Java). Como a “identificação” do sensor com um número inteiro; uma “intensidade” - valor numérico - consoante o valor da intensidade do sinal; e um “prazer (desagrado)” - valor numérico - de satisfação (ou não) para o robô consoante o que está estipulado para esse sentido. O Prazer varia consoante a variação da intensidade entre dois pontos desse sentido no tempo. Por exemplo, se se aproximar da luz, sente prazer, se se afastar, sente desagrado. Ou então, sente dor quando luz é muito intensa ou quando choca com algo.

O sentido da visão que incluí a distância aos objetos em frente, por exemplo, tem uma “identificação” do sensor; uma “intensidade” em valor igual à distância em centímetros; e um “prazer positivo”, se a distância ao objeto se aproximou do robô ou diminuiu a distância por exemplo, ou tem um “prazer negativo” (desprazer, tristeza), se a distância ao objeto se afastou do robô ou aumentou a distância. Além disso, o robô “sente dor” se a distância for inferior uma determinada proximidade, por exemplo 30 cm. Choque eminente?!

Outra suposição, é de que o cérebro não tem poderes para associar 2 eventos reais, duas palavras por exemplo, em décimas de segundos e lembrar-se para sempre através das conexões que se estabeleceriam entre as células nervosas, no mesmo espaço de tempo. Assim, cada pedaço de informação do tipo descrito em cima ou seja do tipo <Signal>, será colocado num novo neurónio, nomeadamente em cada dendrite, que são as entradas dos neurónios. No axónio ou saída no neurónio saí um número único, entre os outros neurónios, que representa a “identificação” desse neurónio.

Por fim, a fé que houve, é a de que o cérebro contribui por inteiro, para satisfazer o corpo, como um todo. Como “Todos os neurónios por um e um por todos”.

5 Ideia

Tudo começou por volta dos meus 17 anos, ano 1980, quando estava na moda colocar as primeiras calculadoras programáveis a jogar umas contra as outras, para ver quem tinha feito o melhor programa. O programa que vencesse o jogo ganhava o desafio. E foi um primo meu que me lançou esse desafio. O jogo era o “4 em linha” e com esse jogo fiquei com a questão de fazer um programa que não só desse para jogar o “4 em linha” como qualquer outro jogo.

Ou seja, como é que nós pensamos, como é que é guardada a informação e que informação? A partir daí, e em 2004, descobri, vá lá, as “rodas do carro” ou seja, a difusão. A ideia de que era possível um cérebro com difusão da informação que seria partilhada para todos os neurónios. Em 2008 descobri o “motor do carro”, ou seja, a dinâmica do cérebro com o “tablier do carro”, ou sejam, as emoções. Em 2015 descobri o “Prazer do cérebro” ou seja, o “volante do carro”, que representa a escolha do melhor caminho a seguir. E em 2023 descobri a “Atenção do condutor” ou seja o pescoço do corpo. Olhar para o que chama a atenção, como por exemplo uma luz que se acende, um barulho estranho ou um movimento rápido são alguns casos. Aí o carro ficou completo e o funcionamento do cérebro também, pelo menos até ver... A “carroçaria do carro” equivale ao corpo eletromecânico do robô.

Em Dezembro de 2023, terminei a construção do programa Liberu, da minha empresa Vita Liberu, Unipessoal (“site” na Internet www.vitaliberu.pt). Neste site estão as fontes do programa Liberu que é feito em linguagem de programação Java e é constituído por 13 classes, 8 executáveis: Liberu (classe mãe); Interface; Mind; Remember; Emotions; Sleep; Cerebellum; Act. E 5 classes descritíveis dos dados: Brain; Cortex; Neuron; Signal, World. O programa Liberu que significa Livre em Latim, começou a ser escrito em 2004 e pretende ser uma simulação do cérebro, com um córtex cerebral para registar o seu passado, para qualquer animal que possua cérebro incluindo o ser humano.

A classe Liberu é onde arranca o programa. Onde é feita a ligação com o utilizador através da classe Interface. Na classe Liberu também se arranca

com a classe Mind que é a maestra da dinâmica do cérebro. Esta classe, a Mind, recebe do exterior do robô todos os sentidos, incluindo o sentido de si próprio (propriocetivo) e chama a classe Remember com 5.000 cópias a executarem-se em simultâneo, para procurar uma memória semelhante com as entradas dos sentidos. A classe Mind chama depois a classe Emotions que recebe da classe Remember o recordado. É também na classe Emotions que são detetadas e expressadas as emoções. Depois a classe Mind chama a classe Sleep com outras 5.000 cópias executáveis em simultâneo, onde o tempo é percorrido para trás e para a frente a velocidades rápidas, à procura de um caminho mais aprazível entre os percorridos. Gerando a emoção "Saudade" se o caminho mais aprazível for o caminho para trás e a emoção "Ansiedade" se têm mais "Prazer" em seguir o caminho para a frente. Essa pesquisa é para 6 passos ou seja seis ações efetuadas para a frente e para trás em tempo, do Passado vivido e reconhecido.

A seguir o Mind chama a classe Cerebellum que envia para a classe Act os sinais do sentido propriocetivo correspondente ao momento passado vindo do Remember ou vindo do Sleep e negativos se o sinal de Dor for detetado. A classe Act por sua vez inverte a ação se o sinal for negativo "de dor" ou executa a ação correspondente aos mesmos sinais propriocetivos guardados no Cerebellum do robô.

As classes Brain, Cortex, Neuron e Signal e World, são classes não executáveis e, por conseguinte, servem para definir tanto os sinais envolvidos, como definir um neurónio, assim como definir um córtex que pode ser salvaguardado num ficheiro de cerca de 6 MB e também para definir um cérebro (Brain) com a sua arquitetura e aonde é definido o número de passos que no nosso caso foi de 5.000 passos. Resultando em 6h 57min de janela de vida considerando um tempo de ação de cerca de 5 segundos por tempo de ciclo ou passo. Multiplicando o número de passos 5.000 pelo tempo de ciclo ou passo de 5 segundos obtemos o tempo de vida que o robô se recorda, ou seja, cerca de 6 horas e 57 minutos. A classe World, serve para representar um mundo virtual no ecrã do PC para questões da simulação. Pois o programa Liberu, pode comutar entre robôs e simulação.

Os 5.000 passos são definidos na constante NEURONS da classe Brain e é a única que pode ser alterada para cada robô definindo assim, com o tempo de ciclo das ações do robô, o tempo de passado possível de ser armazenado. Já que em cada ciclo o robô regista, no seu passado, um momento sentido pelos seus sensores.

Um ser humano, se, por exemplo, for possuidor de 14.191.200.000 neurónios no córtex. A dividir por 6 (5 sentidos mais 1 do propriocetivo ou o sentir do próprio corpo) temos, conjuntamente com um tempo de ciclo, de cerca de 1/5 de segundo que já é muito bom (é o tempo de reação mais rápido, só reagir, sem pensar – classe Sleep), temos portanto um possível tempo para armazenar um passado de precisamente 15 anos. Sensivelmente o mesmo tempo de duração de cada regeneração do corpo humano, que dizem ser de 15 em 15 anos. Ora a esperança média de vida anda pelos 82 anos, logo significa que temos uma janela de memória passada dos últimos 15 anos. Não significando, porém, que não nos recordemos de coisas mais antigas, em situações em que esses eventos vão sendo relembrados, entre os 15 anos passados anteriores, para uma memória presente. Lembrança essa que vai resistir outros 15 anos. Vai passando de geração em geração de memórias.

Para uma ideia nascer deve sujeitar-se aos seguintes passos:

- Passar pela religião; uma pessoa tem de ter fé em algo, pode ser algo que lhe facilite trabalho, algo necessário existir, um objeto, um ídolo, etc.

- Passar pela filosofia; estruturar ideias a ver se são possíveis e realistas.

- Passar pela ciência; formular as ideias.

- Passar pela engenharia; concretizar as ideias.

- Passar pela técnica; afinar as ideias.

- E por fim, vem o utilizador final, da ideia base de alguém que começou por ter fé.

Assim eu comecei por pensar e ter fé que os numerosos neurónios do cérebro estavam todos a trabalhar para o mesmo objetivo: o próprio corpo. Como que, “todos por um e um por todos” como nos mosqueteiros.

Portanto questões de segurança e privilégios de informações que existem no mundo dos computadores, não se colocavam no caso do cérebro. Todos os neurónios têm acesso a tudo e cada um é especialista numa coisa e uma coisa só. Como “cada macaco no seu galho”.

Foi assim que em 2004 comecei a fazer contas ao número de neurónios que tínhamos no cérebro, 100 mil milhões, para ver se dava para memorizar o meio ambiente através dos sentidos. Se, a esse número, dividirmos pela esperança média de vida de cerca de 80 anos, temos 39 neurónios por segundo. Ora era possível armazenar 39 coisas num segundo. Em que cada neurónio guarda nas suas dendrites ou entradas do neurónio. Ou seja, guarda os sinais respeitantes a cada momento sentido pelos neurónios de cada sentido humano, como a visão, audição, tato, paladar, olfato e o sentido propriocetivo. Sendo os sentidos 6, partindo do princípio que se regista um passado de momentos sentidos à razão de 1/5 de segundo ou 200 milissegundos, para dar tempo do corpo de agir nos músculos, temos cerca de 8 neurónios gastos por ciclo. Não necessitamos de gravar as 25 imagens por segundo que os olhos distinguem pois não têm influência a nível de decisão muscular que se faz ao ritmo do corpo. Pois o corpo precisa de mais tempo para se mover. Guardamos por isso uma amostragem de 5 imagens por segundo, uma por cada ciclo, registrando, cada imagem, nas dendrites ou entradas do neurónio atribuído para guardar o momento presente.

Logo, fiquei a saber que era possível termos memória suficiente no cérebro para registrar todo um passado.

O princípio de Dale na neurociência diz que “um único neurónio pode conter diferentes neurotransmissores, mas libertam os mesmos neurotransmissores em todas as sinapses." (William A. MacKay). Isto sugere a

possibilidade de cada neurónio ter uma identidade única distinta de todos os outros neurónios permitindo-nos que cada neurónio guarde nas suas dendrites os sinais de entrada e sinalizando à saída através da sua única identidade. Ou seja, cada neurónio só é responsável pela sua identidade. Pode ser um número natural único por exemplo. Um objeto real tem como representação no córtex, um conjunto de neurónios cada um com um detalhe do que vemos no objeto, constituindo uma memória para um passado vivido desse objeto.

6 Sinal - Neurónio

O neurónio serve para medir semelhanças entre, o sinal registado quando esse neurónio captou os sinais reais oriundos de neurónios sensitivos registrados no passado, e os sinais reais atuais presentes nos neurónios sensitivos. Ou seja, o neurónio sinaliza tanto mais, quanto a semelhança entre o momento passado, próprio desse neurónio registrado para sempre – "recPower", e o momento presente – "actPower". Assim para cada dendrite i ou entrada do neurónio, que regista um sinal, do conjunto de sinais somados, que definem um sentido (visão, audição, etc) temos 4 entidades de entrada do sinal "Signal": uma entidade "emitter" que representa o remetente da mensagem, única para cada neurónio; outra entidade "recPower", que regista o momento em que o neurónio começou a fazer parte do passado, com o registo da altura presente; outra entidade "actPower", que contém o mesmo sinal mas no momento presente, em que, quando o neurónio do presente regista pela primeira vez o presente, o "recPower" retêm o valor do "actPower" ; e ainda uma entidade "pleasure" ou prazer, que representa o que é Bom ou Mau para o corpo como um todo. Este prazer de Bom ou Mau é dado pela diferença entre a amplitude do sinal no momento presente e a amplitude do sinal no momento anterior ou ciclo anterior ou passo anterior, em que ciclo ou passo são tudo sinónimos nesta apresentação.

Na saída do neurónio temos um sinal com 3 entidades: o remetente "emitter" ou seja o seu próprio endereço, que no nosso caso é um número inteiro para cada neurónio; o "actPower" que é igual ao "actPower" da entrada e igual à amplitude do sinal sentido; e o "pleasure" que o neurónio sentiu.

A diferença, em valor absoluto ou positivo (Math.abs()), entre o valor dos dois sinais em amplitude "x", no momento da procura por semelhanças entre o presente "actPower" e passado "recPower", é tanto maior quanto a não semelhança, e é dado pelo valor "x". X é a distância à igualdade entre os sinais presente e passado. Se X=0 os sinais são idênticos.

```
x = Math.abs(getPerception(i).recPower - getPerception(i).actPower)
```

Para termos um valor, proporcional à semelhança entre os sinais gravados e reais, temos de inverter o sinal ou seja, negando o sinal que depois de racionalizado à escala de 100 por cento, temos o valor “activeCredit” que é proporcional à semelhança. A igualdade vale 100%. Ou seja activeCredit = 100. O “Brain.MaxInSense” é uma constante que representa o valor máximo, em amplitude, do sinal de entrada, em todos os sentidos, de valor igual a 255, no nosso caso.

```
activeCredit += (Brain.MaxInSense - x) * 100 / Brain.MaxInSense
```

Se definirmos um limiar de disparo em percentagem e de “não disparo” temos: se a semelhança for acima dos 95%, por exemplo, que sendo o “threshold” ou limiar de disparo do neurónio, limiar esse pré-definido para cada sentido. Então o neurónio dispara se tiver um activeCredit superior ao threshold, informando que disparou, sinalizando com a sua identificação ou o número único do neurónio.

if (activeCredit >= threshold) fire = **true**;

return activeCredit

Aqui, nesta função que testa se o neurónio do passado, dispara ou não, é retornado o valor de semelhança (return activeCredit). Que conjuntamente ou somado com os valores de todos os 6 sentidos, para cada momento no passado, é atribuído um valor de semelhança, para qualquer momento em geral, executado na classe Remember. Esta classe encontra o momento mais aproximado do presente. Ou seja, do último momento passado mais semelhante ao momento presente. E envia para a classe Emotions e para a classe Sleep que por sua vez, na classe Emotions, esse momento, pode ser chamado a entrar na classe Cerebellum para passar à classe Act, a que age. Este momento detetado ou lembrado só será executado na classe Cerebellum se na classe Sleep não for encontrado um momento melhor para ser executado, com mais prazer do que o prazer sentido no presente.

A ciência de computação diz-nos que qualquer algoritmo pode ser executado pela máquina universal de Alan Turing, o computador, e que essa mesma máquina poderia ser feita inteiramente de portas lógicas NANDs, negação da conjunção, e ORs, somas, pois com estas podemos fazer os ANDs, conjunção e NOTs, negação. Tendo assim todas as operações lógicas, à exceção da implicação que é da responsabilidade da classe Sleep, abordada mais tarde quando se analisar a Ansiedade e a Saudade. Pois bem o que o neurónio faz é um NAND (NOT (A AND B)) convertido em NOT A OR NOT B. já que no nosso neurónio temos o inverso das entradas x (NOT - que já por si são uma diferença ou uma distância) e o activeCredit += é incrementado (OR) com todos os valores de activeCredit das dendrites “i” dos vários sentidos, de todos os seis sentidos.

Como um sistema autónomo, o neurónio da memória vivida, contém uma terceira entidade que é o “prazer” . O “prazer”, é dado ao neurónio aquando este regista o presente, tornando-se passado. O valor do “prazer” é determinado pela diferença sentida em amplitude, de um sentido “visual”, “auditivo”, etc, do robô. Este valor pode ser positivo / prazer ou negativo / desconforto. Por exemplo, olhar para algo brilhante, dá-nos um certo prazer, ao contrário da escuridão que provoca um certo medo ou incomodo mental. Diferente da Dor explicada mais à frente. Os sons altos, mas não muito altos para não causar Dor, dão-nos conforto, ao contrário do silêncio.

7 Introdução aos sistemas

Tudo no universo, pode ser considerado como um sistema em termos de funcionamento. Pode-se escolher um conjunto de componentes e considerá-los isolados do resto do universo, desde que os nossos objetivos não sejam perturbados, apesar que esses componentes podem interagir com o exterior. Temos então um sistema que pode ser estudado tanto teoricamente como na prática. Entre os componentes do sistema pode haver uma interação, assim como em cada componente pode haver uma alteração do seu estado interno. Numa análise abstrata do sistema, podemos através de referenciais ou de tabelas de associações, quantificar ou qualificar o estado de um componente ou a interação entre dois componentes. Qualquer evento ou característica de algo que faça parte do sistema, matéria ou energia, pode ser analisado pela chamada informação estabelecida pela resultante dessas quantificações ou qualificações. Essa informação pode então ser recebida ou enviada entre um componente e outro. Pode também ser criada, alterada ou destruída num componente. Para isso é necessário que a informação esteja de certa forma codificada, segundo uma determinada tabela de codificação ou de um referencial.

A informação que origina de um referencial, como um ponto, um eixo, uma distância padrão, um tempo padrão, um estado de congelamento e de ebulição, etc., é uma informação quantitativa e pode ser representada por um número real, logo sendo uma entidade contínua.

Uma informação que tenha origem numa tabela de associações entre um código e uma representação real é uma informação qualitativa e pode ser representada por um número inteiro, sendo de natureza discreta.

Qualquer sistema pode ser constituído por três tipos de componentes: por matéria, objetos, algo palpável, o “hardware”; por energia, eventos, ações, algo imaterial, o “software”; e por ferramentas, instruções, comunicação, o “firmware”. A qualquer desses três componentes pode-se aplicar a informação tanto quantitativa como qualitativa. Por exemplo: para um bolo aparecer feito são necessários 3 componentes:

- Hardware: os ingredientes, materiais sólidos, líquidos, gasosos para o bolo ser feito.
- Sofware: os utensílios a usar incluindo o cozinheiro.
- Firmware: a receita do bolo, aprendizagem.

Outro exemplo, uma porção de água a ser aquecida, resulta numa transferência de energia entre a chama e a água. Quantidade essa que é representada por um número real e é algo imaterial. Por outro lado, a água vai aquecendo, resultando numa alteração do seu estado interno com uma subida de temperatura. Sendo, portanto, uma característica de um material e uma representação contínua. Quando a água entra em estado de ebulição podemos caracterizar esse estado de uma maneira qualitativa, em oposição ao estado líquido.

7.1 ***Sistemas com mensagens***

Em todos os casos podemo-nos abstrair da realidade e trabalhar com a informação resultante dessa realidade ou desse sistema. Deste ponto de vista, podemos criar o conceito de mensagem que não é mais do que um pedaço de informação que transita entre o emissor e o recetor. Essa mensagem destina-se alterar, ou não, o estado do recetor ou/e do emissor. Essa alteração é tanto maior quanto a novidade contida na mensagem.

Outro conceito no mundo das mensagens é a distinção entre vários possíveis recetores ou destinatários de uma mensagem. Para isso, podemos distingui-los por uma classificação qualitativa, quer com nomes, ou com números. Atribuirmos-lhes assim um endereço.

Por fim temos o caminho a tomar por essa mensagem para alcançar o destinatário. Esse caminho pode ser do conhecimento do emissor, ou não, ou do sistema em si, que direciona a mensagem para o destinatário, devido a um conhecimento prévio, ao longo do trajeto a partir de onde o emissor se encontra até ao destino. A esse conhecimento dá-se o nome de encaminhamento.

7.1.1 Com endereço e encaminhamento

Entre os vários sistemas que incorporam mensagens podemos começar por analisar um muito usado e em que se baseia o telefone, os correios, e a internet.

Neste sistema, o emissor tem de saber com quem é que quer comunicar quer por saber o número de telefone do destinatário, quer por saber a morada do destinatário, quer por saber o endereço de internet (URL) do computador e página do destinatário. No caso da internet, ao acedermos a um “link” o endereço do destinatário foi lá incorporado pela pessoa que desenvolveu a página. Para todos os efeitos o emissor sabe o endereço do destinatário, assim como podemos consultar o número de telefone que nos é dado pela lista telefónica. No caso dos e-mails temos de saber o endereço do destinatário.

Introduzindo o endereço no sistema, quer por discando o número no telefone, quer introduzindo a carta na caixa de correio, quer por clicando no “link” duma página de Web ou por dando o comando Enviar no correio eletrónico, o sistema toma conhecimento do destinatário e tem de saber o caminho a dar à mensagem pelo processo de encaminhamento. O caso de um telefonema, as centrais telefónicas descodificam o número e estabelecem uma ligação entre o emissor e o destinatário. No caso dos correios os carteiros, sabendo, orientam a carta consoante o endereço, para a morada escrita na carta. No caso da Internet, os computadores da rede “routers” orientam para outros computadores de rede, consoante o “link” ou endereço de e-mail, até que esse pedido chegue ao destino.

Ou seja, nestes sistemas temos que saber, nós emissores, com antecedência o endereço do destinatário e o sistema por si deve saber encaminhar a mensagem para esse mesmo destinatário, baseando-se nesse endereço.

7.1.2 Com endereço, sem encaminhamento

Noutros sistemas como por exemplo o usado nas redes locais de computadores, ou numa chamada de alunos, temos algo diferente.

Neste caso o emissor necessita de saber a quem se quer dirigir e chamar, por assim dizer, precisa de saber o destinatário. No caso das redes locais de computadores, o que quer estabelecer a comunicação, lança para a linha, comum a todos, o endereço do destinatário. No caso duma chamada pelo professor para identificar as presenças dos alunos, temos o professor a chamar pelo nome do aluno, difundido o nome perante toda a turma.

Posteriormente o computador destino ou aluno chamado identifica-se com o seu endereço ou nome, respetivamente, respondendo em conformidade. Posto isto verifica-se que o sistema em si não necessita de saber aonde é que se encontra o destinatário e, por conseguinte, não precisa de um encaminhamento das mensagens, uma vez que esta chega a todos os elementos possíveis de destinatários. Os elementos que não se identificam com o endereço ou nome de chamada, pura e simplesmente ignoram o emissor, nessa chamada.

7.1.3 Sem endereço nem encaminhamento

Por fim temos um tipo de sistema também muito conhecido, que se pode encontrar nos meios de comunicação, quer sejam os jornais, as revistas, livros, o rádio ou a televisão.

Aqui temos uma difusão da mensagem para todos os destinatários pertencentes ao sistema. Assim o emissor espalha a mesma mensagem por todos os que compram o jornal, a revista, livro, que sintonizam o posto da rádio ou o canal de televisão. Todos esses destinatários que se ligam ao sistema difusor recebem a mesma mensagem. Logo o emissor não se dirige a ninguém em particular, não precisando, portanto, de conhecer o endereço ou nome do destinatário.

Por seu lado o sistema, uma vez que todos recebem a mensagem, não necessita de saber encaminhar a mensagem para um específico destino, pois nem sequer temos o conceito de destinatário. Ou por outra, todos são destinatários, mas nenhum se distingue dos outros.

7.2 ***Sistemas com controlo***

Em todos os sistemas que se alteram com o tempo podemos considerar que sendo essas alterações provocadas por um ou vários elementos desse sistema, temos o conceito de controlo, que representa a ação de um elemento que provoca a alteração noutro elemento. Podemos dizer que o primeiro elemento controlou o segundo, pois alterou, de forma involuntária ou não o comportamento do segundo.

7.2.1 Controlo centralizado

A maior parte dos sistemas criados pelo Homem, incorporam uma unidade de controlo de certa forma centralizada, cujo objetivo é conduzir o sistema de determinada forma, face à vontade humana.

Um exemplo mais simples, poderá ser um aquecedor, em que nós definimos a temperatura através de um botão rotativo. O aquecimento tem um dispositivo que deteta a temperatura ambiente e se esta está abaixo do definido por nós o aquecedor trabalha. Caso a temperatura esteja acima do que foi previamente definido, o aquecedor desliga-se.

Um caso dum sistema mais complexo de controlo, é o computador. Aqui temos uma unidade de controlo que controla todo o computador mediante a vontade humana. Nós introduzimos os comandos na memória do computador e ordenamos a este que as execute. Quando isso acontece, essas ordens que se encontram na memória são entregues à unidade de controlo que realiza assim, na prática, essas ordens.

Como podemos colocar em memória variadíssimas sequências de ordens, que podem ser alteradas pelo utilizador, faz com que o computador seja, por excelência, o sistema de controlo mais versátil.

Pelo meio, temos por exemplo os carros que são controlados através do volante, pedais, mudanças e botões, numa zona de controlo centralizada, ou seja, o lugar do condutor.

Em qualquer dos casos, podemos visualizar estes sistemas como sendo compostos por várias camadas de comando, umas em cima das outras. Temos por exemplo no computador, e de uma forma sintetizada, a unidade de controlo. Esta fornece sinais elétricos aos fios que executam na prática as ordens vindas de uma camada superior que é o microcódigo. Esse microcódigo define os diversos comandos que são possíveis executar pelo processador. Este por sua vez recebe ordens vindas de uma camada superior que é a memória, em forma de comandos. Esses comandos formam o sistema operativo do computador, que por sua vez delega aos chamados programas executáveis que são introduzidos pelo utilizador e que são para todos os efeitos, também comandos percetíveis pelo processador. Por fim, temos a última camada que é o utilizador do computador que vai dando ordens, através do teclado ou rato, de maneira a executar os comandos que se encontram na memória.

Ou seja, aonde quero chegar é que, em última análise, a camada superior a todas as outras nestes sistemas é sempre o Homem. Tudo é feito sob a nossa vontade e o nosso controlo. Os sistemas elétricos, por exemplo, são controlados de uma forma centralizada no Homem.

Podemos então considerar, que os sistemas de mensagens em que o emissor escolhe um destinatário para receber a sua mensagem, propiciam a existência de sistemas com controlo centralizado, quanto mais não seja se considerarmos somente esses dois intervenientes e somente essa mensagem. Em contrapartida, um sistema que não define um destinatário em particular, como os sistemas de difusão sem destinatário específico, não temos um controlo centralizado sobre todos os intervenientes. Pois alguns destinatários poderão não ser afetados pela mensagem. E por outro lado muitos destinatários poderão ser afetados.

Na natureza se olharmos para os animais, incluindo o ser humano, cada um como um todo, temos como unidade centralizada de controlo, o cérebro. Este controla todo o organismo ou todo o corpo, em última análise.

É claro, também, mesmo em sistemas centralizados podem existir certas partes do sistema que funcionam de maneira autónoma, ou seja que não podem, nem são, controláveis pela considerada unidade central de controlo para funcionarem. Só comunicando por mensagens.

À centralização do controlo, chega-se quando a nossa análise termina no detalhe, e se estabelecem fronteiras, delegando a tomada de decisão a algo externo ao sistema considerado. Definindo a entrada da tomada de decisão externa, a esse sistema, como a unidade central de controlo. É, no entanto, um conceito que pode ser muito abstrato e dissolvido em algo descentralizado.

7.2.2 Controlo descentralizado

Se analisarmos toda a cadeia ou sequência de controlo de qualquer sistema chegamos à conclusão que não pode existir uma unidade centralizada de controlo que domine o sistema o tempo todo. Deve existir, isso sim, um controlo descentralizado em que diversos elementos do sistema vão tomando, à vez, as rédeas e o controlo de outras partes do sistema. Temos, portanto, uma interligação de elementos que se vão influenciando uns aos outros, sem nenhum poder ser considerado o controlador de tudo.

A natureza está cheia e repleta de exemplos desses. Os vírus, plantas, animais e seres humanos são todos influenciados em última análise, pelo ambiente em que se encontram e reagem em conformidade, no mesmo ambiente. O cérebro dos animais e dos seres humanos que controla o corpo dos mesmos, não é ele próprio mais do que um sistema de controlo distribuído. O controlo é feito pelos neurónios que o constituem. Não podemos dizer é que, há um ou um conjunto de neurónios, que centralizam o controlo de todo o resto do corpo. Podemos dizer então, que os neurónios se revezam, por assim dizer, no controlo do organismo, sendo a responsabilidade desse controlo repartida pelos inúmeros neurónios.

Sendo assim, a vontade própria dos seres vivos só pode ser entendida de uma forma macroscópica. Da mesma maneira que, se estivermos a assistir a um jogo de bilhar, em que não se conseguisse ver as tabelas da mesa de bilhar, nem as pessoas que jogam, ficaríamos a pensar que as bolas têm vontade própria, pois mudam de direção nas tabelas, que não as vemos, e iniciam movimentos do nada, já que também não veríamos as tacadas efetuadas na bola. No entanto sabemos que isso não é verdade.

Foi feita uma experiência, em que se dizia a uma pessoa que, frente a um relógio, decidisse premir um botão e dissesse quando tomou a decisão. Por exemplo, decidi carregar no botão aos 10 segundos. O que se verificou é que no cérebro já se tinham formado sinais que antecediam em, um curto intervalo de tempo, à altura em que o sujeito referiu ter tomado a decisão. Ou seja, a decisão consciente do indivíduo tinha sido precedida por uma influência inconsciente, cuja origem poderá ter sido externa ou interna à pessoa.

Por isso, a vontade própria é algo que só pode ser considerado a nível macroscópico em que a pessoa tem a sensação de que toma as decisões.

8 Sistema de comunicação

8.1 Endereços e Endereçamento

Que tipo de sistema é que precisamos? Como vimos no capítulo anterior e para sintetizar, e juntar o estado da arte, o cérebro humano tem células que se comunicam entre si. Logo trata-se de um sistema de comunicação entre elementos básicos semelhantes, formando um todo. Estes sistemas de comunicação podem ser classificados, em termos de endereços, e encaminhamento da comunicação. Ou seja, temos 4 sistemas de comunicação possíveis: caso o sistema tenha ou não endereços e caso tenha ou não encaminhamento.

Primeiro caso: com endereços e encaminhamento

- São o caso dos telefones, correios, internet e transportes.

- Neste caso, o sistema de encaminhamento necessita de tabelas de conversão entre os endereços destinatários e as direções de encaminhamento. Como por exemplo, precisamos dos números de telefone dos nossos contactos e routers para encaminhar esses números. Por vezes enormes tabelas. Logo este caso de sistema não nos interessa (biologicamente pouco viável).

Segundo caso: sem endereços, mas com encaminhamento

- Este caso é impossível, pois só faz sentido ter encaminhamento se existirem endereços destinatários para encaminhar.

Terceiro caso: com endereços e sem encaminhamento

- É o caso de um professor a fazer a “chamada de presenças dos alunos”.

- O endereço do destinatário, nome do aluno chamado pelo professor, chega a todos os alunos por difusão. Todos os elementos do sistema “ouvem” o professor e só responde o interessado. Ou seja, todos reconhecem o seu nome e quando o ouvem respondem. Os outros alunos permanecem calados como se não tivessem ouvido nada.

- Neste caso, os elementos emissores do endereço destinatário, ou seja, o professor, necessita de listas por vezes enormes de nomes de alunos e, acreditar que o aluno esteja a dizer a verdade, pois o professor pode não ter método de confirmação. O crédito ao aluno é dado pelo professor. Não é um sistema que se quer consistente, mas sim rodeado de incertezas por todos os lados, como no cérebro.

Sobra o quarto caso: sem endereços nem encaminhamentos

- São o caso da televisão, rádio, jornais, livros, distribuição elétrica, distribuição de água, distribuição de gás, radar ou sonar.

- A comunicação não é pessoal, mas sim para todos

- A mensagem não tem endereço de destinatário, destina-se a todos os elementos que fazem parte do sistema. Quem estiver interessado pela mensagem reage à mesma, os outros ignoram-na.

- Por exemplo, a televisão anuncia os números sorteados da Lotaria. Os vencedores são os mais interessados e contentes com a notícia agem em conformidade. Outros, interessados mas desiludidos com a perda no jogo ignoram a notícia. E os que não jogam nem ligam ao anúncio da Lotaria ignorando também a notícia.

- Este sistema é o que nos interessa. (Também, não sobra mais nenhuma hipótese).

9 ”Procura”

Como é que se pode proceder a uma procura num sistema sem endereços de destinatários nem endereçamento? Vamos ver uma analogia:

Se formos à procura de um livro sobre “Ferrari” numa biblioteca, como é que fazemos? Procuramos por temas! Mas isso pressupõe que existiu um trabalho prévio, de alguém que organizou os livros por temas, dessa biblioteca. E se esse alguém não existisse? como encontrávamos o livro? Bem, por mais que pense vou sempre chegar à conclusão que tenho de procurar “Ferrari” livro a livro por toda a biblioteca!?

Pois bem…, no entanto, imaginemos que os livros têm “vida” própria!! Que reconhecem o seu próprio tema e respondem em conformidade, dando a sua localização. Ou seja, chegávamos à biblioteca e gritávamos pelo tema “Ferrari?” e o livro (ou livros) que têm como tema “Ferrari” reconhecia(m) o nome “Ferrari” e respondia(m) “Estou aqui neste sítio!” os outros calavam-se. A resposta está à “distância” temporal duma pergunta, pois esta destina-se ou é difundida a todos os elementos (ou livros) do sistema (ou biblioteca).

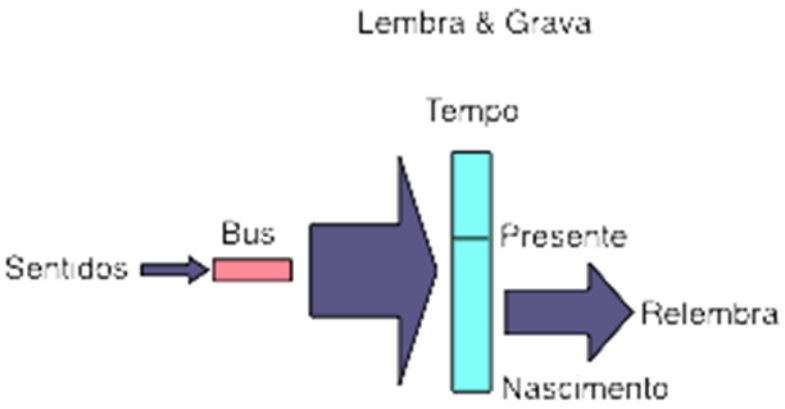


Fig. 1

Na figura 1, vemos a classe de Java de nome Remember em que as perceções entram nos 6 “buses” dos sentidos (os 5 sentidos mais 1 do propriocetivo) e destes são lidos por todos os neurónios da memória (Tempo)

em paralelo ou ao mesmo tempo. Disparam (relembram) os neurónios cujo o seu registo passado “recPower” se assemelha à entrada vinda do sentido correspondente, com o “actPower” real e presente no Bus, que envia para todos os neurónios do Tempo. Tempo esse, de janela temporal de retenção no passado, que é igual a “NEURONS” ou o número de passos vezes o tempo médio de cada passo. Recomeçando no nascimento, mas já com um passado totalmente preenchido quando acaba o futuro ou seja quando se esgotam os passos em número igual à constante “NEURONS”.

Não há propriamente, uma procura assim dita. Há um processamento em paralelo em que cada livro ou neurónio tem um programa (ou uma Thread em Java) que analisa e reconhece o seu tema. De facto, cada pedaço de dados a “procurar” tem esse programa associado que está atento à entrada da biblioteca ou fila de chegada ou “bus” da informação (uma ConcurrentLinkedQueue em Java). E a classe Remember compara o que chega com o que tem em memória de um passado vivido.

Na nossa implementação usamos perto de 10.000 programas, 5.000 programas para o Remember cada um com 6 sentidos ou com 6 neurónios, mais 5.000 programas para o Sleep. Sem contar com os programas executáveis do programa Liberu. No Cerebellum contabilizamos, na salvaguarda ou cópia do sentido propriocetivo, 5.000 neurónios para cada um dos 2 hemisférios do Cerebellum, esquerdo e direito. Na ação, classe Act, temos 12 neurónios ou ações. Temos portanto um total de 40.012 neurónios (as formigas têm entre 10.000 e 100.000 neurónios – logo, daí o nome do projeto do robô ser Formica – do Latim, formiga) e o robô tem um ciclo de procura, pensar e agir de um tempo médio de 2 segundos (num processador Intel i7 de 8 núcleos a 3,6 GHz de velocidade). Não quero com isto dizer, que é mais rápido que as procuras em árvores binárias, ou em outros sistemas de procura já existentes, mas é “biologicamente” falando, mais viável de existir, e independente da quantidade de livros ou neurónios existentes.

É do nosso interesse procurar as memorias semelhantes com o que é percecionado ou sentido como realidade.

10 Aprendizagem

Mas como é que um sistema destes aprende? Para aprender, aprende-se com algo de fora de que se quer aprender, é algo que vem de fora. Para isso vou revelar um velho hábito dos Eletrotécnicos, chamado: o método da "caixa negra" ("black box"):

Quando se tinha em mãos um aparelho elétrico desconhecido (caixa negra), introduzia-se sinais nas entradas e, tentava-se descobrir, pelo que se obtinha nas saídas do aparelho, como é que esse aparelho poderia funcionar. Ora é precisamente o que se vai usar aqui, mas, com um truque: não é o programador que vai descobrir como é que o mundo funciona e representa-lo num robô, mas sim implementar um robô que aprenda a conhecer o mundo e a adaptar-se. Logo, temos de por o cérebro do robô a descobrir como é que o "Mundo", e corpo do próprio robô funcionam.

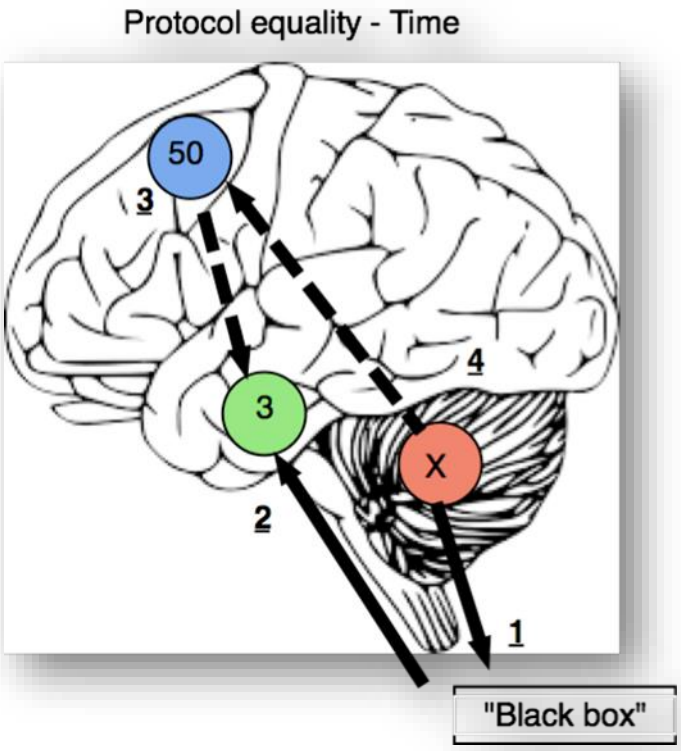


Fig. 2

Na figura 2, o sistema cerebral produz, no passo 1, através das saídas do cérebro “X”, a identificação do seu sinal, seja ele qual for, produzindo ações nos músculos. Estas ações dos músculos agem no corpo e no mundo real que, por analogia, apresentada em cima, representam as “entradas” do aparelho ou caixa negra. Depois, no passo 2, as perceções recebidas pelos sentidos, como por exemplo o neurónio com o identificador de sensor número “3”, ficam registadas no neurónio “3” e representam a “saída” do mundo real ou por analogia a “saída” da caixa negra. A caixa negra neste caso, é o corpo e o ambiente do robô como um todo. Por isso não precisamos de nos preocupar mais em saber como é que o mundo funciona para se poder ter um robô adaptável. Nem tão pouco, o próprio robô, como sistema cerebral necessita de saber como é que o corpo funciona. Ou seja, o neurónio de saída vai aprender com o neurónio da entrada, guardando essa informação e sendo o único detentor dessa informação.

Por sua vez, no passo 3, o neurónio “50”, intermediário, regista a identificação do neurónio que disparou a seguir, com o número 3 já que foi o neurónio “3” que disparou. No passo 4, o neurónio “X”, que está em “escuta” pelo(s) neurónio(s) que disparar(em) a seguir a ele que, neste caso, será o neurónio “50”, regista por fim a identificação, deste último, com o número 50.

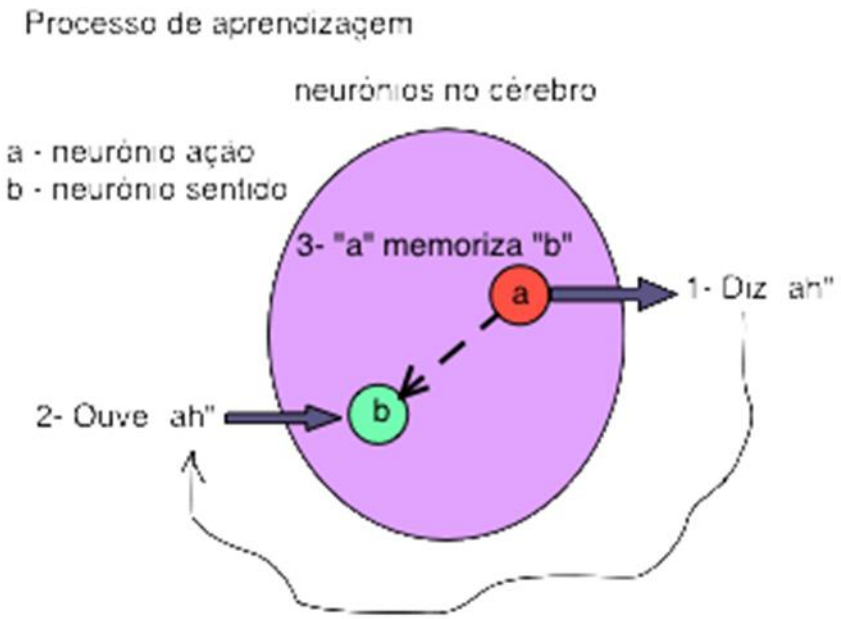


Fig. 3

Outro exemplo, como se pode ver pela figura 3, o neurónio de saída “a” dedicado a agir sobre um músculo da fala, “dispara” ou seja, entra em ação e o robô produz o som “ah”. Depois disso, fica constantemente à “escuta” dos registos de entradas. Diz-se que fica em “Standby” ou à espera.

De seguida o nó “b” (ou nós) de entrada dedicado(s) à audição, de uma determinada frequência, deteta o som “ah” e como tal dispara(m) ou seja, coloca(m) no(s) registo(s) de saída somente 3 informações (da classe Signal): o seu número de identificação “b” (em Java inicializou-se cada neurónio - com a sua estrutura de dados, e Thread ou programa associado - com um número distinto para cada neurónio); a intensidade do sinal; e o seu prazer/depressão em relação ao que foi ouvido no ciclo anterior. Maior intensidade do sinal em relação ao anterior, mais prazer, menor intensidade em relação ao anterior, maior depressão.

Por fim, o neurónio de saída “a”, que estava em standby, deteta registo(s) na entrada e guarda, nas suas dendrites - em Java temos um ArrayList<Signal> - a informação dada pelo neurónio posterior, obtendo-se um padrão específico de neurónios de entrada para o som “ah”. Passando isso, o neurónio de saída “a” fica “sensível” à identificação(ões) do(s) neurónio(s) de entrada “b”, posterior(es) em tempo real ao seu. Quando o neurónio de saída deteta outro padrão igual “b”, ou semelhante dentro duma margem pré-definida para cada sensor, provocado pelo som vindo de outro lado externo que não o próprio “a”, o neurónio “a” sinaliza um disparo e o som “ah” volta a ouvir-se no mundo real, ou não, devido à vontade própria do robô de satisfazer o que lhe dá prazer e evitando a depressão e a dor.

11 Passado

Tudo o que o cérebro faz, em última análise visível, é mover músculos. Pois bem, todos os movimentos são em função do tempo. Não podemos passar do ponto A ao ponto B sem passar por todos os pontos intermédios, entre um caminho possível, entre A e B.

Os movimentos do robô, logo, são em função do tempo. O tempo prossegue num único sentido, do presente para o futuro.

A única estrutura de dados que mantém o fator tempo, se bem que de uma forma discreta do tempo, com uma amostragem o mais rápido que se conseguir, no nosso caso temos uma amostragem sincronizada com o ciclo do robô, ou seja, cerca de 5 segundos no caso da simulação no ecrã, é um formato em linha. Em oposição às estruturas de dados, em forma de árvore com as suas ramificações. Já que as estruturas em árvore perdem o fator tempo. Ou seja, não se fica a saber de imediato se um acontecimento num ramo da árvore aconteceu primeiro ou não, em relação a outro ramo. Por outro lado, se tivermos uma linha temporal isso é fácil de saber.

Os dados estão armazenados como que numa pilha de pratos sempre a subir. Cada prato representa 6 neurónios, um por cada sentido, com os seus sinais de entrada guardados. A pilha de pratos significa a linha temporal sempre a subir para o futuro. Entre dois pratos passam-se 5 segundos ou o tempo de reação. O prato do topo representa o ponto presente. E a perceção dum objeto real, por exemplo, pode estar contida em vários pratos, um para cada pormenor do objeto. O robô vivo implementado com nome de código Liberu pode ter 5.000 pratos ao todo. Já que a tecnologia presente não permite muito mais.

Se o cérebro do robô tem 5.000 camadas de neurónios e cada camada ou 6 neurónios é preenchida com perceções a cada 5 segundos, temos uma janela temporal de vida, de cerca de 6 horas e 57 minutos.

Com esta estrutura de dados em linha, cria-se um “passado” do robô com uma pilha de dados que vai crescendo (retângulo azul na figura 1). E contém toda a vivência ou experiência do robô. Todos os sinais sensoriais, captados de 5 em 5 segundos, desde o seu nascimento e sempre que estiver ligado e a funcionar.

Por outro lado, um ser humano adulto tem, de altura máxima, 2 metros desde o cérebro até à ponta dos pés, ou seja, 4 metros num caminho de ida e volta. Como um sinal nervoso percorre a uma velocidade máxima de cerca de 100km/h, logo demora 0.144s a percorrer os 4 metros. Ao inverso de 0.144s, temos a frequência máxima do ciclo temporal, ou seja, cerca de 7 neurónios por segundo. De modo a que o sistema de comunicação não se “engasgue” com dados em atraso. Verificou-se porém, que temos uma velocidade de reação de cerca 200 milissegundos (a minha velocidade de reação é de 203 milissegundos) ou seja 5 neurónios por segundo, o que está abaixo do previsto de 7 neurónios por segundo. O elefante e a baleia têm mais neurónios que o ser humano, mas também vivem mais tempo.

Mas nós dissemos que o robô tinha vida teoricamente infinita. Isso deve-se ao facto de quando a pilha completou os 5.000 pratos, voltarmos a gravar no prato do fundo da pilha. Vai-se, assim, ter uma janela temporal máxima de cerca de 7 horas, ou seja, tem-se um passado recente de cerca de 7 horas. O que não significa que, não nos lembremos de algo mais antigo pois, basta relembrar um ato, que esse ato passa para o presente. E por conseguinte, para a janela temporal seguinte.

12 Decisão

Quando queremos saber aonde é que estacionámos o carro, não agimos em termos de probabilidade estatística. Se não, começávamos a dirigíamo-nos quase sempre na mesma direção. Em último caso, procuramos em função do que os sentidos nos esclarecem.

Ora, normalmente relembramo-nos em função do tempo. Em outras palavras, recordamos o mais recente (na implementação a variável Emotions.Recent) episódio em que estivemos no carro. Daí relembramo-nos onde estacionámos o carro. É um pensamento automático. É o que o nosso instinto nos diz à primeira.

Mas nem para todos os casos nos interessa lembrar do último episódio em que nos ocorreu algo. Interessa-nos antes, saber se determinado caminho temporal é possível ou não. Isso é feito pelo mesmo mecanismo que é usado para o sono (classe Sleep). Aí o tempo anda para trás e para a frente, procurando desde o relembrado no paragrafo anterior, até ao futuro ou passado vivido durante uma determinada profundidade, que no robô foi de profundidade igual a 6. Aí se o futuro, já vivido, for mais prazeroso fica-se com “ansiedade” acelerando as ações. Se for o passado, em relação ao lembrado, o mais prazeroso fica-se com “saudade” e mais lento nas ações.

Se tivermos “Ansiedade” o robô dá um primeiro passo nesse caminho mais prazeroso. Se tivermos “Saudade” invertemos o caminho. Ou seja temos um avança ou recua. Como o lutar ou fugir.

Em caso de se sentir “Dor” física temos uma atitude de reflexo contrária à real ou atual ação. Ou seja age-se em sentido oposto ao que provocou a dor.

13 Cerebelo

Ao contrário de termos a propagação de sinais da entrada, processamento e para a saída, temos da saída, entrada e para o processamento.

Da saída para a entrada, pelo exterior, começamos por gerar ações. Essas ações podem ser aleatórias, por instintos genéticos (como quando um bebé move os seus braços quase aleatoriamente) ou por ações induzidas por algo externo ao sistema (como ajudar o robô a pôr-se de pé). As ações fazem mover o corpo do robô e este por sua vez, interage com o mundo. Os sinais devolvidos, na interação do robô com o mundo, são dados pelos sentidos, nomeadamente o sentido propriocetivo, e são gravados no passado dentro das memórias do presente, que se vão sempre acrescentando no tempo.

Por outro lado, o sentido propriocetivo entra no cerebelo (classe Cerebellum) e nos neurónios de saída (classe Act) que estes últimos também fazem parte do cerebelo. No Cerebellum são registados os sinais propriocetivos por ordem temporal, como no passado. Esses sinais vão ser, mais tarde, enviados para todos os neurónios de saída por difusão e que depois de comparados, com o que registaram e se adaptaram, vão reagir ou não à chamada.

Ou seja, a adaptação funciona de modo a anular sobreposição de sinais no sentido propriocetivo. À medida que a mesma ação é executada, o neurónio nas suas dendrites guarda os sinais propriocetivos e mantém aqueles que se repetem e ignora os esporádicos. Sabendo que o neurónio apesar de tudo, se reconhece a ele próprio. Esses sinais que se mantêm, são associados com a ação executada. Pois cada neurónio atua com uma ação diferente dos outros neurónios.

É como fazem os astrónomos para fotografar uma região do céu de longa exposição. As luzes que permanecerem nessa região do céu serão estrelas as outras são interferências ocasionais e são eliminadas, por computador, da imagem. Eles, astrónomos, partem do princípio que nesse espaço de tempo em que é tirada a fotografia, não nasce nem morre nenhuma estrela. O que é o mais provável que aconteça.

É na altura do período aleatório que se dá a adaptação dos neurónios de saída, aos sinais propriocetivos ou a chamada aprendizagem por sobrevivência. Deve-se porém garantir que todos os neurónios de saída, e suas ações sejam executadas as vezes necessárias a eliminar sinais esporádicos. Ou seja, que as ações aleatórias sejam disjuntas umas das outras pelo menos uma vez. Podendo mesmo serem executadas várias ações em simultâneo.

Outra aprendizagem, permite uma pessoa levantar o robô pelos braços e o robô, mais tarde, pôr-se de pé sozinho. Pois temos só sinais propriocetivos e nenhuma ação efetuada. É uma aprendizagem induzida, por um agente externo (pessoa), ao robô.

14 Arquitetura

O sistema cognitivo Liberu e sua arquitetura podem ser vistos na figura 4. Aqui podemos distinguir 5 módulos que correspondem às respetivas classes em Java. Juntamente, com a classe de topo Mind, temos as 6 classes executáveis que constituem a dinâmica do robô. As 2 classes executáveis que faltam, dizem respeito à própria aplicação Liberu: classes Liberu e Interface.

A classe Mind é o maestro das 5 classes da figura, Remember, Decision ou Emotion, Sleep, Cerebellum e Act. A primeira classe a ser executada são os 5.000 programas da classe Remember que leem os sentidos e recordam-se de acontecimentos semelhantes, nomeadamente o mais recente.

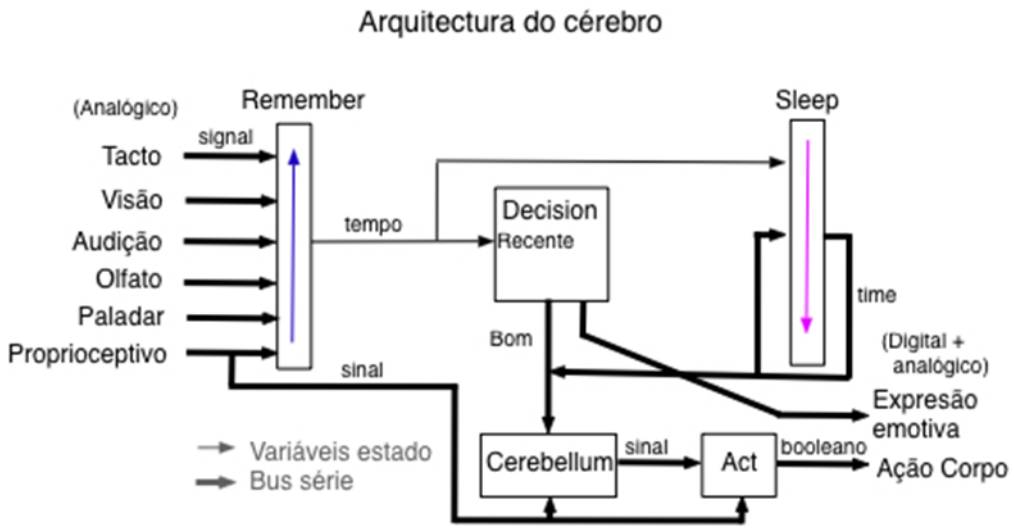


Figure 4. Arquitetura do cérebro do Robot

Esses acontecimentos semelhantes alteram variáveis de estado, próprias de cada neurónio.

A seguir é executado o módulo Emotions (Decision na figura). Segue-se o módulo Sleep, com os seus outros 5.000 programas. No módulo Emotions

(um só programa), são gerados os números aleatórios e expressas as emoções detetadas em vários pontos do ciclo.

No Sleep temos a procura, num ciclo mais acelerado (6 passos de profundidade), dos caminhos para as metas ou objetivos mais “prazerosos”. Sejam objetivos de longo ou curto prazo à profundidade de 6 passos.

O Cerebellum é executado no fim do Sleep, e recebe o resultado instintivo do Emotions ou o mais elaborado resultado do Sleep.

Finalmente o Act representa os neurónios de saída que no nosso caso são 5 (Frente, Direita, Esquerda, Energia e Home). Ou seja, são 5 programas ou tarefas melhor dizendo a funcionar em simultâneo. Este módulo, recebe sinais propriocetivos e do Cerebellum e expressa ações.

No geral entram sinais analógicos e saem sinais digitais (músculos distintos) com uma componente analógica (intensidade do estímulo).

Esta arquitetura é independente do corpo do robô e é biologicamente viável. Eu, pessoalmente, acredito que todos os animais possuem a mesma arquitetura de cérebro. Sendo que, o que varia, para os animais terem diferentes poderes, é que o seu corpo, não os seus órgãos, varia. Mas varia também a cultura do animal ou seja toda a informação que passa de geração em geração, para além do ambiente, em que se encontram, também variar.

15 Dor, Prazer, Motivações e Aleatório

15.1 ***. Dor***

Sempre que o robô sente um sinal de “Dor” (detetada pelo tato), o robô reflexivamente reage com a ação contrária à última ação sentida. Temos um ato reflexo. Além disso incorpora no sistema um sinal de prazer negativo, ou seja, de depressão ou “desprazer”.

15.2 ***. Prazer***

O prazer é dado por um botão de energia usado pelo utilizador como que “um bombom” que se dá ao cão. Ou quando o robô tem falta de energia são gerados momentos de prazer, quando lhe é fornecida energia. Serve para o robô saber que esses momentos de prazer lhe pertencem e vão servir como objetivos a alcançar como no caso da aquisição de energia, nas baterias do robô.

15.3 ***. Motivações***

As motivações, tais como para receber energia quando falta, são um sinal que entra no sentido propriocetivo e vai fazer com que o robô se lembre de situações semelhantes, durante essa motivação. Com a ajuda do Sleep (sono) coloca-se, também, como objetivos os sítios que o robô conhece como sendo locais de abastecimento elétrico ou mais “prazerosos”. E se o robô no seu passado encontrou o abastecimento, vai encontra-lo, em princípio, no futuro.

15.4 ***. Aleatório***

Ações aleatórias, são geradas no início enquanto o sistema está no útero (“Womb”). Essas ações constituem um início de passado aleatório. Ajudam os neurónios ou nós de saída (classe Act) a eliminarem o ruído.

Servem para um primeiro arranque, como o motor de arranque nos carros.

A ação aleatória deve desaparecer para que o robô não desaprenda um hábito sem querer. No nosso caso, são só os primeiros 30 passos do robô, ou

seja, os primeiros 150 segundos de vida do robô. Como só temos 5 ações distintas, não são necessários muitos passos de ações aleatórias, 30 bastam. Devem ser o suficiente para gerarem todas as 5 ações necessárias para que o robô não fique sem conseguir no futuro repetir essas ações não geradas aleatoriamente.

16 Dinâmica - Sono

O sistema funciona da seguinte maneira: os sentidos captam perceções em forma de sinais. Esses sinais lembram, ao robô, por difusão aos nós do passado, alguns episódios temporais no Remember. Além disso, os mesmos sinais são gravados no presente.

A seguir, o módulo Remember envia para o Emotions, o mais recente episódio recordado ou com mais impacto, ou seja, o mais parecido com os acontecimentos reais – nível de confiança. Assim, como se exprime emocionalmente, com as seguintes emoções: novidade; alegria; tristeza; medo; dor; além do estado normal.

O módulo Sleep entra então em ação e procura, dentro do recordado, em sentido contrario e a favor do tempo, se há um caminho mais “aprazível”?! De facto, cada neurónio, do passado recordado, guarda a informação do(s) neurónio(s) temporal(is) que disparara(m) no ciclo seguinte. Pode-se, portanto, disparar os objetivos na entrada dos neurónios do Sleep, e andar para trás (saudade) ou para a frente (ansiedade) no tempo e encontrar o caminho, desde o momento presente até ao mais “prazeroso”.

Se for encontrado um caminho, a entrada do Cerebellum é substituída pelo novo momento temporal. Esse momento temporal entra no Cerebellum que envia para os neurónios de saída, Act, com os sinais propriocetivos correspondentes ao momento temporal, já que no Cerebellum está uma cópia do passado sensorial propriocetivo sentido no presente.

O Act, por sua vez, faz equivaler esses sinais a ações reais no corpo do robô, implicando e interagindo com o meio ambiente. Pois durante o período aleatório já fez corresponder os sinais propriocetivos às ações distintas do robô.

Isto perfaz um passo num ciclo infinito de passos possíveis.

17 Emoções

As emoções são particularidades do sistema, como por exemplo, quando nos lembramos de ter sentido algo novo, em que é expressa a emoção de novidade. Essas emoções servem também para o robô distinguir algo bom de algo errado.

As emoções são expressas geneticamente no módulo Emotions e, depois de expressas, retornando sinais propriocetivos que vão ser gravados no presente do robô. Passando o robô a lembrar-se de eventos emocionais semelhantes, especialmente recorda-se da última ocasião de emoção semelhante.

Pois nós temos tendência para quando simulamos o riso ocorrer-nos recordações de riso e rimo-nos genuinamente.

Neste sistema temos as emoções: alegre/triste; novidade; dor e medo (medo quando sentimos uma redução no “prazer”).

O alegre e triste tem a ver com os níveis de prazer ou depressão que o robô está a sentir.

Temos também a “Ansiedade” e a “Saudade”: quando se encontrou um caminho melhor, para a frente ou melhor para trás, de aonde estamos, respetivamente a ansiedade se for para a frente ou a saudade se for para trás. E o robô dá um passo nessa direção, sabendo nós que, para um carro ir do ponto A ao ponto B não necessita de saber todo o caminho, basta saber o próximo passo à distância dos faróis do carro.

18 TESTE

Os testes efetuados funcionaram dentro das expectativas. Temos de concordar que o robô da Lego NXT Mindstorm, é um pouco limitativo em termos de sensores. Mas o esperado é que o robô aprenda com o passado e saiba caminhar entre objetivos dele (robô), como por exemplo, carregar a sua própria bateria.

Foi também testado com um robô de tecnologia Arduíno (construído por mim, chamado Formica - de formiga em Latim) e ainda uma simulação no ecrã do computador (minhoca C.Elegans). Os robôs comunicam com o computador, por Bluetooth, ao programa Liberu.

18.1 DESCRIÇÃO

O corpo do robô tem, para o caso do robô da Lego NXT Mindstorm:

- 4 Sensores que captam as perceções exteriores (Luz, Distância, Contacto, e Som).

- Sensores que captam as sensações interiores, propriocetivas, como o estado dos motores e da bateria.

- 2 Motores para se mover (Frente, Esquerda, e Direita).

- Emissor de som.

O cérebro tem as seguintes características:

- Nunca para de funcionar, exceto se a pedido do “dono”.

- Capacidade de aprender.

- Capacidade de sobrevivência com o tempo e educação para não ficar sem energia e não se prejudicar. Ou seja, para satisfazer as necessidades do corpo.

- Capacidade de detetar sensações novas e avisar o utilizador. Como música em alto volume pela primeira vez.

- 12 Classes em Java. 4 tipos de estruturas: Signal, Neuron, Cortex e Brain; e 8 executáveis: Liberu, Interface, Mind, Remember, Emotions, Sleep, Cerebellum e Act. A classe Liberu contém o método de arranque, main().

- 1 Classe em Java da estrutura do mundo de um labirinto para o C. Elegans, chamada World, podendo ser alterada. Não é necessária para aquando se usa o robô da Lego ou o de tecnologia Arduíno.

18.2 ***Entradas***

Os 4 sensores têm um sinal cada um na entrada do sistema, cada sinal variando entre 0 e 255. E o sentido propriocetivo tem 12 sinais correspondendo a 5 ações, 5 emoções, energia e motivação.

18.3 ***Saídas***

As saídas são 5. As três direções de movimento, a motivação para voltar para casa e a voz, “speak”, que é um zumbido.

19 CONCLUSÕES

Nenhuma informação transita de um passo para o outro à exceção do último movimento.

Outra conclusão, foi uma ideia já abandonada que se revelou ser verdadeira. Para uma pessoa reconhecer um nome de alguém que lhe é conhecido tem, de ser enviado um pedido de procura por imagem da pessoa e em retorno ao mesmo sítio, a resposta é assim verificada ou reconhecida. Ou seja, a informação anda em círculo. Por isso é imperativo que os neurónios motores do cerebelo, se reconheçam a si próprios como únicos entre os outros.

Uma aplicação prática poderá ser para segurança doméstica, na deteção de algo diferente nas perceções aprendidas pelo robô, que passeia pela casa e avisa o dono de possíveis perigos. Como deteção de sons estranhos, imagens novas, ou mesmo temperaturas exageradas.

20 “Trabalho a fazer”

Estive a instalar o programa Liberu, para um robô de tecnologia Arduino. Esse robô ainda está em fase de arranque, mas espera-se ter sensores mais aproximados aos dos sentidos humanos, nem que seja em qualidade e não em quantidade de neurónios especializados.

Adicionando um sentido apurado de audição ao robô, este pode receber “comandos” de voz e executá-los, se assim o entender. O número de passos possíveis a introduzir, só depende da capacidade de processamento do computador que executa o programa Liberu. Neste momento, o máximo para um PC regular é de cerca de 10.000 passos dependendo do sistema operativo e da capacidade do processador. Alterando este número, e com um corpo apropriado, poderíamos ter um ser humano artificial com muitas mais capacidades.

Com a arquitetura de computadores a 64bits e, se o Java permitisse que todos os números inteiros pudessem ser substituídos por 64bits, teríamos o equivalente na melhor das hipóteses e com uma capacidade de cálculo incalculável para esta arquitetura de computadores, cerca de 1,4 mil milhões de vidas humanas num único só robô.

Para mais informações, descarregar as fontes/executáveis do programa Liberu em Java, entrar em contacto com o Autor, entre outras coisas, pode consultar o site www.VitaLiberu.pt ou simplesmente vitaliberu.pt ou por último procurar pela palavra VitaLiberu na Internet.

21 Créditos e Comportamentos

Na execução do programa Liberu, descobri a possibilidade de existirem 2 tipos de atribuição de crédito, ao que o próprio fez das suas ações, e com isto permite a possibilidade de existirem 2 tipos de reações diferentes. No programa pode ser alterado o modo de agir do robô entre esses 2 tipos de reação, através dum botão cujo o nome é “Thumb” do inglês Polegar. Esse polegar pode ser o polegar direito, que fica por cima ou o polegar esquerdo, que fica por cima. A alteração pode ser feita em qualquer altura em que o robô está a funcionar, só muda o tipo de ação final a ser executada, tudo o resto funciona igual. No robô reparei que os de polegar esquerdo em cima são mais energéticos e os de polegar direito em cima são mais ponderados.

Mas o que é essa história do polegar esquerdo ou direito em cima… de quê?

Quando se cruzam os dedos das mãos, ou seja, quando os dedos entrelaçados ficam alternados, de maneira natural, sem parecer que nos estamos a sentir desconfortáveis dessa maneira, cruzando entre os da mão esquerda e os da mão direita…por outras palavras como se estivéssemos a rezar a Deus. Fixamos qual é o polegar da mão que fica por cima. Se é o esquerdo ou o direito que fica por cima… por esta ordem de ideias o dedo mindinho que fica por baixo, será o da mão oposta. Já que estão todos os dedos entrelaçados um por um.

Ora isto nasce com a pessoa, por isso, temos dois tipos de créditos a ações na natureza que também existem nos animais racionais, independentemente do sexo do animal, deste ser fêmea ou macho. A que na gíria temos por exemplo entre os seres humanos 4 tipos de comportamentos típicos na sociedade:

Polegar esquerdo por cima, designa-se por uma pessoa feminina EMOTIVA ou masculina EMOTIVA.

Polegar direito por cima designa-se por uma pessoa feminina RACIONAL ou masculina RACIONAL.

As relações mais positivas e com maior surpresa para ambos os membros do casal, ou de amigos, ou entre familiares são as que, entre os dois, haja um EMOTIVO e outro RACIONAL. Só assim se complementam e são mais

poderosos. Porque a qualidade de uns são os defeitos dos outros e vice-versa. Ou seja, o que um não sabe, sabe o outro o que um não consegue fazer, consegue o outro…etc.

Por exemplo, os racionais são pessoas dadas a caminhos nas explicações de direções… dizem algo como, virar na segunda à esquerda, na quarta à direita, definindo um caminho contínuo ou analógico… são, portanto, pessoas tipicamente “racionais” seguem a lógica do possível.

Os emotivos são pessoas dadas a referências ou nomes nas explicações das direções… dizem algo como, virar depois do prédio amarelo, quando chegar às bombas de gasolina vira à esquerda, etc… definindo um conjunto de referências para chegar ao destino. Tendo um pensar discreto são, portanto, pessoas tipicamente “emotivas”. Os emotivos, pela minha experiência muito agradável (já que sou “racional”), têm, na sua maneira de pensar, o “dom” de atravessar paredes pela sua lógica, converter água em vinho e construir um prédio pelo telhado (perdendo o chão pelo caminho) …, mas o que eu me “rio” com eles (emotivos) e elas (emotivas). Não me “rio” deles o que é muito diferente. Rio-me devido a muitas vezes não termos o mesmo tipo de discurso, nem de entendimento…

Características do polegar esquerdo:

Importa: saber exprimir-se, saber comunicar

Importa: o porquê de ajudar – agir, no que é importante

Orientação: de referência em referência (nomes)

Pensamento: discreto

Atrai-se: pela personalidade

Importante: quantidade

Discurso: mais adjetivos (mais cor à vida)

Sistemas Livres – Paulo Roque Silva

Discurso: saltitam entre assuntos

Não se consegue por na posição do outro

Gosta de ser reconhecido pelos outros

Tendência para gestão, artes, advocacia

Normalmente são mais bonitos

Características do polegar direito:

Importa: mensagem a transmitir (ideia)

Importa: como pode ajudar – agir, o procedimento

Orientação: por caminhos contínuos

Pensamento: analógico

Atrai-se: pela beleza física

Importante: qualidade

Discurso: mais advérbios

Discurso: mantêm o objetivo a esclarecer

Consegue-se por na posição do outro

Confia em si próprio e não necessita do reconhecimento dos outros

Tendência para ciências

Normalmente são mais rudes na beleza

22 Demonstração Empírica

1. Cada neurónio memoriza “algo” (nas dendrites e sinapses) e todos são “diferentes” quando disparam ou comunicam (através do axónio).
2. O sistema de comunicação, como foi analisado, é por difusão “total”.
3. Podemos imaginar 1 anel com “todos” os neurónios, em que, quando um neurónio “fala” ou dispara os outros todos “ouvem” e estabelece-se um “ciclo”.
4. No começo “algum” neurónio do “anel” vai disparar com a sua “identificação”.
5. O(s) neurónio(s) que vai(ão) disparar em “segundo” lugar, dispara(m) por algum “motivo”?!
6. Há 3 “MOTIVOS” para o(s) “segundo(s)” neurónio(s) disparar(em) e por isso há 3 “ANEIS” de neurónios possíveis de existirem: hemisfério cerebral esquerdo, direito e cerebelo.
    a. 1º “motivo”: os neurónios do anel disparam por “semelhança” com o que registaram “algo” através dos “sentidos” no passado.
        i. Obtém-se: uma identificação “diferente” temporal (mais recente ou melhor no passado) dos neurónios dos “sentidos” mais “semelhante” com o real “sentido” (no presente) (HEMISFÉRIO CEREBRAL REAL).
    b. 2º “motivo”: os neurónios do anel disparam porque os neurónios quando dispararam a primeira (ou não) vez registam a ou as “identificações” dos neurónios que dispararam “logo de seguida” e antes do próximo “ciclo”. Logo disparam se reconhecerem a “identificação” no registo de “identificações” dos neurónios (lista curta, devido à repetição pelo evoluir do tempo – seria outro neurónio a registar devido ao primeiro motivo) que dispararam “logo de seguida”.
        i. Obtém-se: a possibilidade de andar para frente no tempo (HEMISFÉRIO CEREBRAL IMAGINATIVO).

c. 3º “motivo”: os neurónios do anel disparam porque “se reconhecem” a eles próprios.
    i. Obtém-se: a possibilidade de andar para trás no tempo (HEMISFÉRIO CEREBRAL IMAGINATIVO).
    ii. Obtém-se: a ação correspondente a esse neurónio (movimento muscular, hormonas, etc.) ou a ação “inversa” no caso de se sentir “DOR”, em que o local da “DOR” corresponde a determinada ação, assim como a “inversa” (ato reflexo), desenvolvido à nascença (CEREBELO).

7. No início “todos” os neurónios do cerebelo disparam pelo menos 1 vez (3º “anel” / “motivo”).
    i. Obtém-se: um início de “um passado” valorizado de bom/mau pelos “sentidos” “no primeiro anel/motivo” (hemisfério cerebral do real/ “passado”).
    ii. Obtém-se: uma “equivalência unívoca” entre os neurónios do cerebelo (cópia do propriocetivo) e os neurónios do hemisfério cerebral real nomeadamente os do “sentido” propriocetivo, ou seja, o retorno do próprio corpo físico e da “DOR”. Ou seja, a ação, por exemplo, nº 325 vai corresponder no sentido propriocetivo, também ao nº 325, da mesma ação (os números dos neurónios, no computador, podem ser subconjuntos de determinados neurotransmissores nas sinapses. Mas “todos” os conjuntos / “identificação” de cada neurónio “diferentes”.
8. Passado o “período” do passo 7 (comportamento “aleatório”) inicia-se o período “motivador” e continua assim por toda a vivência que se realiza por “ciclos” temporais de “receção e recordar” (anel/motivação 1), de “pensamento” (anel/motivação 2) e de “ação” (anel/motivação 3):
    a. Ciclo de “receção e recordar” – Hemisfério cerebral real: regista, por exemplo, 1/5 de segundo a 1/5 de segundo em cada neurónio de cada um dos 6 “sentidos” (audição, tato,

paladar, olfato, visão e propriocetivo) “valores” de “amplitude” correspondente aos neurónios sensitivos. Esses “valores” dividem-se em 3 tipos: “identificação” do neurónio sensitivo (desnecessário neste caso, do neurónio sensitivo, mas necessário como “valor” que saí no axónio quando o neurónio do ponto de vista do emissor, dispara – dando-nos uma linha ou um circulo ou um formato disperso, temporal passado, criado pelos “ciclos” reais registados); “amplitude” do sinal detetado pelos sensores; e um “valor” “epigenético” de BOM/MAU para o próprio corpo físico. Devolve o “tempo” mais “semelhante” com a realidade sentida como presente, e o “tempo” em que foi “melhor” (BOM/MAU) para o corpo dentre os que foram recordados ou dispararam.

b. Ciclo de “pensamento” – Hemisfério cerebral Imaginativo: anda, entre ações executadas na realidade, “imaginariamente” para a frente e para trás no tempo, entre o mais “semelhante” e o “melhor” dado pelo ciclo de “receção e recordar”. Podem-se executar alguns destes ciclos, por exemplo, 10 ciclos “imaginários” / “sono” para a frente (ansiedade) e para trás (saudade) no mesmo “ciclo” temporal real (deste passo 8). Aqui calcula-se, aditivamente o BOM/MAU, entre ciclos imaginários, para determinar o “melhor caminho”. Já que, se regista neste anel o que vai sendo executado/acionado e acrescentando dendrites do que foi executado/acionado a seguir a esse neurónio imaginário que tem o mesmo número temporal como “identificação” devolvida no final dos ciclos imaginados. Ficamos com um possível “melhor caminho” dado por um valor “temporal”.

c. Ciclo de “ação” – Cerebelo: no inicio (comportamento “aleatório”) são eliminados os registos de, ruídos no sentido propriocetivo, e erros de desenvolvimento, que não correspondem à própria ação (já que cada um “se reconhece” a ele próprio). O cerebelo tem uma cópia do sentido propriocetivo igual ao do hemisfério cerebral real. E quando

vem do ciclo de “receção e recordar” para poder “REPETIR” a execução ou ação do “semelhante” devolvida que fez no passado. Ou quando vem do ciclo de “pensamento” para agir ou executar prioritariamente pelo que lhe vai fazer melhor para o corpo físico. Em ambos os casos o que vem do hemisfério cerebral esquerdo ou direito (real ou imaginativo) é uma informação temporal para ser recordada na cópia que se encontra no cerebelo no mesmo instante de tempo real passado. Dessa forma envia para o anel/motivação 3 do cerebelo os sinais propriocetivos que o anel vai reconhecer pelo período aleatório e executar ou movimentar, algo no corpo físico ou hormonal. Neste anel de ação só se reconhece o que se tem para executar. E existem 2 valores possíveis de “crédito” para dar potencial à ação. Daí o passo 9.

9. O “crédito” dado às coisas, é dado por cada um de nós, seres vivos racionais. Não é por dizerem que “algo” é verdade, que nós vamos acreditar. Ou, podemos acreditar, mas somos nós que damos ou avaliamos e, em última análise, decidimos e executamos em conformidade do que nos faz mais capazes. Existem 2 “maneiras” de executarmos o crédito que damos às coisas. E cada um tem a sua “maneira” predefinida à nascença que se distingue pelo cruzar dos dedos – das duas “maneiras” possíveis de cruzar os dedos. E, pela lógica, dois indivíduos, com ambas as “maneiras”, são, no conjunto, mais poderosos do que dois indivíduos de “maneiras” iguais. Dois indivíduos com “maneiras” iguais entendem-se melhor.
10. As emoções são características do próprio sistema. São como o painel de luzes do tablier dum carro. Por exemplo, o detetar “algo” novo num determinado sentido e incorporar esse “sinal” de “novidade” no sentido propriocetivo no próximo “ciclo”
    i. Obtém-se: estados diferentes para cada emoção (se simularmos o riso ficamos mais contentes). Detetei 5 emoções: uma física, do corpo, a “DOR”, as outras 4,

mentais; “novidade”, “alegria”, “tristeza” e “medo”. Com algumas combinações possíveis. Como rir, falar.

23 Programa

Começando pelo princípio quando eu tinha 17 anos (1980) e programei uma calculadora em linguagem de computador Basic o jogo do 4 em linha. Jogo esse que consistia colocar um circulo numa fileira para conseguir 4 círculos seguidos. Aí pensei que o que era útil, era conseguir posições privilegiadas, atribuindo-lhes um peso maior para a jogada seguinte.

Depois, passando vários anos pensei no que seria que um cérebro humano precisava para pensar e como guardar a memória. Pensei inclusivamente em árvores binárias ou seja decisões de sim ou não.

Ao longo dos anos seguintes fui anotando peças do puzzle que me faziam sentido.

Em 2004 (já tinha 41 anos de idade) ocorreu-me pensar no ditado popular “Um por todos e todos por um” e dizer que o cérebro e todas as suas células estão todas pelo mesmo corpo e o corpo está para todo o cérebro.

Então se o corpo está para todo o cérebro é porque o cérebro todo recebe notícias do corpo. Havia um difusão da informação. Para isso era melhor que as células do cérebro estivessem todas alinhadas com as suas competências e ouvirem tudo o que o corpo dizia. Mas alinhadas como?

E pensei que, se definir o tempo, ao contrário das redes binárias que, não sabem quando definiram o tempo da decisão de sim ou não, teria um movimento contínuo, como os nossos músculos se regem por um tempo continuo. Ou seja como um gravador. Que no ser Humano a amostragem é de cerca de 1/5 de segundo como rapidez de reflexos e por conseguinte a velocidade de gravação do registo de memória. Por isso registamos todos os 6 sentidos (Visão, Audição, Tato, Paladar, Olfato e Propriocetivo – sentimento do próprio corpo) 5 vezes por segundo e agimos em conformidade.

Então como é que se ia guardar essa informação consecutiva e em função do tempo? Só poderia ser, ao contrário das redes binárias, uma memória em linha reta, temporal. Assim teremos uma linha reta em função do tempo em que todos os neurónios ou memórias recebem em difusão do corpo os sinais recebidos dos sentidos.

Assim sabendo nós que os neurónios disparam ou transmitem informação em consequência de ter ultrapassado determinado nível de disparo comecei a definir uma classe de programação do neurónio chamada Neuron.

Temos assim o início da programação, em 2004, da classe Neuron. Comecei por definir a informação necessária para um neurónio do cérebro. Essa classe foi modificada, entre muitas vezes, em 2014 com o acrescento do prazer de um sinal do neurónio.

```
/** Creates a new instance of Neuron */
public Neuron(int neuronId, int th) {

        pointer = 0;
    step = 0;
    end = false;
    who = neuronId;
    threshold = th;
    quality = 0;
    finalQuality = 0;
    fire = false;
    standby = false; // Espera por disparo do aleatório
    standby2 = false; // Espera disparo do Mind
    perception = new ArrayList<Signal>(Nperceptions);
    learn = new ArrayList<Signal>(Nperceptions);
}
```

Vemos na classe Neuron a inicialização de algumas memórias do programa ou variáveis.

Temos: a variável “pointer” que representa o número do passo ou “step” actual; a variável “who” que é a própria identificação do neurónio, diferente de todos os outros neurónios, portanto único; a variável “threshold” define o limiar em percentagem do nível de disparo do neurónio, ativando o neurónio pelo que sentiu por difusão da sensação nas entradas do neurónio e sinalizando o seu disparo enviando a sua identificação ou seja a memória “who”; quando o neurónio dispara a variável “fire” fica como Verdadera (“true” e “false” em linguagem Java) significando que o neurónio disparou ou seja que foi ultrapassado o limiar de “threshold”; a variável “quality” guarda a qualidade da informação das entradas do neurónio ou seja o prazer com valor de um número positivo, ou desconforto com valor de um número negativo, inferior a zero; a variável “finalQuality” define o prazer total de determinado passo (variável “pointer”) dos 5 sentidos humanos mais o propriocetivo ou seja o resultado dos 6 sentidos ou 6 neurónios ficando registado na variável “finalQuality” do

neurónio do sentido propriocetivo; as variáveis standby e standby2 servem quando verdadeiras ou seja, quando o os neurónios musculares ou ativos ou de saída disparam ficando à espera do próximo passo, a variável “standby” e a variável “standby2” para a noção de tempo do passo ou o número do passo previamente executado, ficando à espera do passo executado a seguir; e as variáveis “perception” e “learn” são para guardar as entradas do neurónio, em forma explicada na próxima classe que vamos ver que é na classe Signal.

Continuando na classe Neuron temos nas seguintes linhas de linguagem Java. A declaração das variáveis necessárias para essa classe.

```
public class Neuron implements Serializable {
    public static final long serialVersionUID = 2023L;
    // Versão dos dados
    private static final int Nperceptions = 10;
    // perceptions atribuidas por defeito
    static boolean                  end;
    // Se verdadeiro o programa do neurónio é para acabar
    int                             pointer, // atual passo
                                    step; // total de passos dados
    int                             who, // Identificação própria
                                    quality, // Bom ou mau
                                    finalQuality, // quality total
                                    threshold;
    // Nivel de reação (em percentagem 0-100)
    boolean                         fire, // Neurónio disparou
                                    standby,
    // Para a aprendizagem por sobrevivência no Act
                                    standby2;
    // Para a aprendizagem Out-Wait-In do Sleep
    ArrayList<Signal>         perception;
    // Sinais a que o neurónio ouve ou age
    ArrayList<Signal>         learn; // Sinais para o Sleep
```

Podemos ver em anexo na classe Neuron que contem varias funções ou procedimentos preparados para manipular as diversas variáveis. Por fim há 2 procedimentos que vale a pena saber, “powerOfFire” e “powerToAct”, mas antes vamos conhecer como os sinais de todo o programa são constituídos ou seja a classe Signal, para entendermos os procedimentos “powerOfFire” e “powerToAct”.

A classe Signal é onde são definidas as variáveis ou memórias dos sinais de entrada e saída dos neurónios.

As variáveis são: o “emitter” que guarda, que neurónio é que enviou o sinal, isto é a variável do neurónio “who” do remetente; a variável “active” que indica

se o sinal está ativo ou não; a variável “pleasure” que retém o prazer da sensação, um valor positivo (aproximar da luz, sons mais altos, são exemplos) ou negativo dando desprazer ao robô resultado duma experiência má e este sente medo (aproximar do escuro, redução do som, por exemplo); a variável “recPower” que regista a primeira sensação, em amplitude, do neurónio e nunca é apagada essa informação, exceto para neurónios motores que aprendem por persistência do mesmo sinal presente, ignorando com o tempo os outros impulsos de possível ruído; e por fim a variável “actPower” que guarda durante o tempo de um passo (“step”) a informação dos sensores, em amplitude, no presente e é propagado para todos os neurónios das Classes Remember, Emotion, Cerebellum e Act.

```
public class Signal implements Serializable {
    public static final long    serialVersionUID = 2023L;
    // Versão dos dados
    int                                 emitter;
    // Identificação do nó que emitiu o sinal
    boolean                             active;
    // Se o sinal está ou não presente
    int                                         pleasure;
    // Positivo se prazer, negativo se desprazer
    float                               recPower,
    // Amplitude do sinal original
                                        actPower;
    // Atual ou real amplitude

    /** Creates a new instance of Signal */
    public Signal(int em, float pwr, int plsr) {

        emitter  = em;
        active   = false;
        pleasure = plsr; // Entrada do prazer
        recPower = pwr; // Nunca muda excepto os nós do Act
                            // pela adaptação de sobrevivência
        actPower = pwr; // A atualizar para um novo valor
    }
```

O resto da classe Signal são procedimentos para manusear as diferentes variáveis dessa classe.

Voltando à classe Neuron, aqui estão os procedimentos “powerOfFire” e “powerToAct.”.

```
    // Para os neurónios no Remember
    public float powerOfFire(int sense) {
        float     activeCredit,
                  nActives, // Número de sinais ativos
                  rtrn ;
```

```
      int         i; // index
      float   x; // Diferença absoluta entre amplitudes

        nActives = 0;
      activeCredit = 0;
      rtrn = 0;
        for (i=0; i<perception.size(); i++)
            if (getPerception(i).active ||
                        getPerception(i).recPower >
            Brain.MinSense) {
                  // Calcula activeCredit
                  nActives += 1;
                x = Math.abs(getPerception(i).recPower -
                        getPerception(i).actPower);
                if (x > Brain.MaxInSense)
                  x = Brain.MaxInSense;
                activeCredit += (Brain.MaxInSense - x) *
                        100 / Brain.MaxInSense;
            }
      fire = false;
      if (nActives == 0) nActives = 1;
      activeCredit = activeCredit / nActives;
        if (activeCredit >= threshold) {
            fire = true;
            rtrn = activeCredit;
        }
        return rtrn;
    }
```

Na rotina “powerOfFire” selecionamos as entradas em amplitudes dos 6 sentidos que estão presentes ou ativas ou seja, as que têm sinal superior à sensação mínima (“Brain.minSense”). Depois vê--se a amplitude da diferença entre o gravado na primeira vez (“recPower”) e o presente (“actPower”) e coloca-se o resultado na forma de uma percentagem na variável “activeCredit”. A seguir faz-se um média em relação todas as entradas ativas. E compara-se com o limiar de disparo “threshold”. Se ultrapassar o “threshold” o neurónio dispara colocando a variável “fire” como Verdadeira. O procedimento acaba por informar em retorno a variável “activeCredit”, que nos diz a grande ou menos grande, já que ultrapassa o “treshold”, de aproximação entre o presente e o lembrado, na Classe Remember.

```
// Para os neurónios do Act
    public int powerToAct() {
        int         activeCredit,
                    totalCredit, // Valor total das percepções
                    credit, // Credito dado
                    i; // index

      activeCredit = 0;
        totalCredit = 0;
        for (i=0; i<perception.size(); i++) {
            if (getPerception(i).active)
```

```
                activeCredit += getPerception(i).recPower;
            totalCredit += getPerception(i).recPower;
        }
      fire = false;
      if (totalCredit == 0) totalCredit = 1;
       // Média percentual dos ativos
      credit = (int)(100 * activeCredit / totalCredit);
      System.out.println(credit);
        if (credit >= threshold) {
            fire = true;
            standby = true;
        }
        return credit;
    }
}
```

Na rotina “powerToAct” selecionamos as entradas em amplitudes, dos 6 sentidos que estão presentes ou ativas ou seja, as que têm sinal superior à sensação mínima (“Brain.minSense”). A seguir, como este procedimento é para ser usado na classe Act ou programa Act que recebe do programa Cerebellum as entradas propriocetivas ou dos músculos ou parte motora, e reage com uma ação motora. Por isso, vai-se ver o total da soma de todos os sinais ativos do “recPower” ou seja o que sentiu quando foram guardados pela primeira vez pelo “activeCredit”. Vê-se também a soma de todos os “recPower´s” quer ativos ou não, “totalCredit”. Fazemos então na variável “credit” a percentagem do “activeCredit” em relação ao “totalCredit” colocando na variável “credit” e comparando com o limiar “threshold”. Se ultrapassar o “threshold” o neurónio dispara colocando a variável “fire” como Verdadeira. O procedimento acaba por informar, em retorno, a variável “credit” que representa como no “powerOfFire” a semelhança entre o que recebeu e o que esse neurónio motor tem memorizado depois da aprendizagem por sobrevivência explicado mais à frente com a variável “standby”.

Na classe Neuron é definido o procedimento da “atenção” (última descoberta só no ano de 2023) do robô “attentionOfFire()”. Trata-se de dar prioridade à novidade nomeadamente no nosso caso à novidade no sentido da visão “Brain.Vision” e no sentido do paladar “Brain.Taste”. Ou seja quando se deteta uma novidade num desses sentidos o “pescoço” do robô no caso da visão dá prioridade virando-se para a novidade e no caso do paladar o robô passa a “comer” em casa ou “beber” da energia.

Passamos agora para a classe Cortex em que é definida toda a informação necessária para poder ser guardada na memória do computador quando se fizer um “Save Cortex…”.

```
/** Creates a new instance of Cortex */
public Cortex(int t, int nAct, int nPast) {
    int i, n;     // indexes

    Name = new String();
    EmailPhone = new String();
    Password = new String();
    Senses    = t;
    nNeuronsAct = nAct;
    nNeuronsTime = nPast;
    actual = 0;
    timeLife = 0;
    bornTime = 0;
    timeBefore = 0;
    exec = 0;
    lastExec = 0;
    beforeLastExec = 0;
    lastQuality = 0;
    time = new Neuron[nNeuronsTime][Senses];
      for (i=0; i<Senses; i++)
             for (n=0; n<nNeuronsTime; n++)
                       time[n][i] = new Neuron(n,
                                   threshold[i]);
    actNode = new Neuron[nNeuronsAct];
    for (n=0; n<nNeuronsAct; n++)
        actNode[n] = new Neuron(n, ThresholdAct);
    hemisphereLeft = new Neuron[nNeuronsTime];
    hemisphereRigth = new Neuron[nNeuronsTime];
      for (n=0; n<nNeuronsTime; n++) {
        hemisphereLeft[n] = new Neuron(n,
                    ThresholdProSelf);
        hemisphereRigth[n] = new Neuron(n,
                    ThresholdProSelf);
      }
      Warn = false;
}
```

Este Cortex tem essencialmente a definição das 4 redes de neurónios. A primeira é a rede que é em linha reta e definida por time[][].perception. A segunda é definida por time[][].learn e forma um rede complexa com continuidade e sem pontas soltas. A terceira rede é a do Cerebelo com os seus dois hemisférios, hemisphereLeft[] e hemisphereRigth[] que são uma cópia do sentido propriocetivo igual ao registado na primeira rede e portanto também de forma linear. Por fim a quarta rede é formada pelos neurónios atuantes sendo uma rede discreta ou sem continuidade ou sem correlação entre os neurónios da rede.

Nesta classe Cortex são definidos os parâmetros de limiar de disparo ou ativação dos neurónios, Threshold, em percentagem para cada sentido, aonde poderão ser efetuadas afinações conforme os sensores utilizados no robô.

Para que possamos trabalhar com a estrutura do Cortex temos a classe Brain. Aqui é que é definido o número de neurónios por cada rede, NEURONS igual a 5000 e ACTS igual ás 5 ações. Define-se também quantos sentidos SENSES igual a 6, e o número de movimentos ou expressões emotivas MOVES igual a 12.

Outra aspeto é o número de sinais individuais que cada sensor produz dado por Nanalog para cada sentido e para cada robô por exemplo no caso da simulação do C.Elegans temos para o sentido paladar ou “Taste” o valor 3 que equivalem ao Terreno, Casa e Energia.

A classe Brain define então rotinas próprias de inicialização das classes executivas, se são as classes Remember, Emotions, Sleep, Cerebellum e Act.

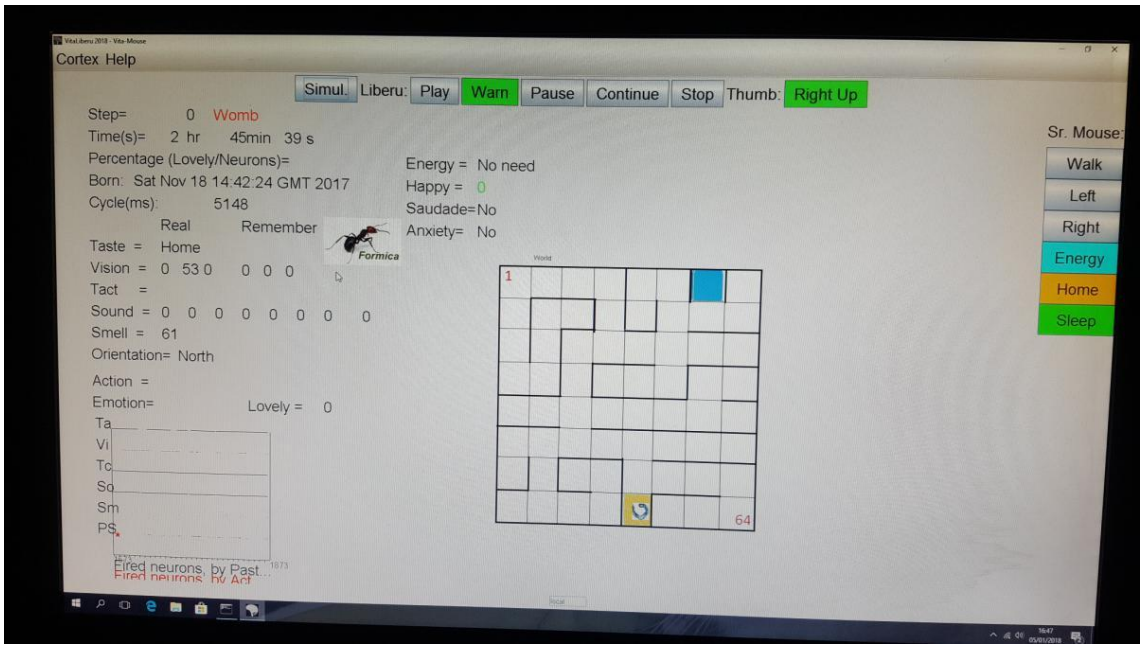
Estão também definidos procedimentos para lidar com a difusão na classe Slep, como “clearBuses”, poolSleep”, “transfer” e “decide”.

Nesta classe é por onde dão entrada os diversos sentidos e motivações para os diversos robôs. São procedimentos como “motivation”, “feel” e “proSelf”.

A última classe não executável ou só de estruturas de dados como as estudadas até agora a lembrar a Signal, Neuron, Cortex e Brain, temos a classe que só serve para a simulação do C. Elegans no labirinto. Essa classe chama-se World e define o esquema do labirinto para o C. Elegans andar. E pode ser alterada para outro labirinto à escolha dentro de um tabuleiro de Xadrez.

Passamos agora a outra categoria de classes que são as classes executáveis como Liberu, Interface, Mind, Remember, Emotions, Sleep, Cerebellum e Act. A classe Liberu e Interface são necessárias para correr o programa no sistema Windows da Microsoft. E servem para, por exemplo, na classe Liberu que sendo a classe mãe começa por lançar todas as outras classes executáveis e mostra a classe onde é desenhado o único ecrã descrito na classe Interface. Estas duas classes Liberu e Interface não representam o

conceito de cérebro aqui exposto, mas para quem sabe ler na linguagem de programação Java pode entender facilmente que não tem nada de mais do que servir de interface homem-máquina representado na seguinte figura.



Cortex - Para novo Robô; para abrir ou guardar Córtex do Robô e para SAIR

Help - Para referência e ajuda

Simul, - Comuta entre Simulação no ecrã do Robô, um Robô de tecnologia Arduino por USB ou Bluethoot e um Robô NXT MindStorm da Lego por Bluethoot
Liberu - Nome do Programa
Play - Inicia o andamento do Robô
Warn - Quando ativo, avisa que sentido detetou algo novo para o Robô
Pause - Pára o Robô
Continue - Repõe o Robô a andar
Stop - Pára o Robô para se poder substituir de Cortex de outro Robô
Thumb - Quando se cruzam as mãos, qual o dedo Polegar que está por cima (Esquerdo, Direito)?
Right Up - Troca de Polegar de cima, entre Esquerdo (Emotivo) e Direito (Racional)

Sr. **Mouse** - Comandos dados ao Robô
Walk - Força um passo de andar para a frente
Left - Força um passo para virar à esquerda
Right - Força um passo para virar à direita
Energy - Dá prazer ao Robô
Home - Para treinar ou reproduzir um treino (normalmente para vir para casa - quadrado Laranja ou Verde atualmente)
Sleep - Pára o Robô e põe a zero o Happy ou a felicidade – em desuso

Step - Contabiliza os passos dados pelo Robô
Womb - Aparece quado o Robô está na fase inicial (30 passos) de movimentos Aleatórios. Aqui o Robô deve executar pelo menos uma vez cada uma das 5 ações, porque se não executar uma ação nesta fase perde o poder de a vir executar no futuro

Time(s) - Tempo de vida do Robô
Percentage (Lovely/Neurons) - Percentagem de neurónios que dispararam em relação à totalidade de neurónios (#sentidos x NEURONS)
Born - Data de nascimento do Robô
Cycle(ms) - Tempo entre passos do Robô em milissegundos

Imagem Formica - Logótipo

Taste, **Vision**, **Tact**, **Sound**, **Smell** - Sentidos do Robô
Orientation - Meramente informativo
Remember - Os sentidos da memória recordada pelo Robô

Action - Os movimentos que o Robô vai tendo (Eat, Drink, Walk, Left, Right) de cores (Vermelho-Aleatório, Verde-Comando ou Atenção, Preto-Normal, Azul-Saudade, Laranja-Ansiedade)
Emotion - As emoções que o Robô vai tendo (Dor, New, Sad, Happy, Afraid)
Lovely - Número de neurónios que dispararam

Ta, **Vi**, **Tc**, **So**, **Sm**, **PS** - Gráfico dos últimos neurónios que dispararam por Sentidos (Ta - Taste, Vi - Vision, Tc - Tact, So - Sound, Sm - Smell, PS - Proprioceptivo ou interno)
Gráfico - A linha do X representa o Tempo e a linha do Y estão os sentidos. Representa os neurónios que dispararam por semelhança
Asterisco vermelho - Representa qual a ação é que o Robô executou
Cruz Vermelha - Indica o tempo na memória do passado em que foi executada a ação do Robô

Energy - Se tem a bateria do Robô com pouca carga, avisa
Happy - Nível de felicidade (número positivo) e Tristeza (número negativo)
Saudade - Se o Robô sente vontade de voltar para trás (anda mais devagar)
Anxiety - Se o Robô sente vontade de seguir um caminho (anda mais depressa)

Homeostase (não visível na figura) - Média por neurónio, da totalidade de felicidade/tristeza dos neurónios usados

Input local - Para introduzir um número entre 1 e 64 para posicionar o Robô nessa casa do tabuleiro

World - Mundo do Robô
Quadrado Azul - Energia
Quadrado Laranja ou Verde - Home
C. **Elegans** – Robô

Vamos ver então as classes executáveis que nos interessa Mind, Remember, Emotions, Sleep, Cerebellum e Act.

Comecemos pela classe Act que é a primeira a agir nas ações aleatórias. Aqui é efetuada a seleção de sinais para os neurónios atuantes que são 5 através da aprendizagem por sobrevivência através da variável “standby” quando está no útero “Womb”.

```
if (neuro.standby && neuro.step <
            Emotions.ActRandom) {
      // Aprendisagem por sobrevivência
      signalCount = 0;
      listen = Brain.BusInProSelf.iterator();
      if (neuro.perception.size() == 0)
        while (listen.hasNext()) {
            signInput = listen.next();
            neuro.addPerception(
                        signInput.emitter,
                        signInput.actPower,
                        signInput.pleasure);
            System.out.println(
                        neuro.who); // REMOVER
            System.out.print(
                  "Add dendrite Body fired=");
            // REMOVER
            System.out.println(
                        signInput.actPower);
            // REMOVER
            signalCount += 1;
        }
      else
        while (listen.hasNext()) {
            signInput = listen.next();
            if (signInput.actPower >=
                        Brain.MinSense)
                  if (signalCount != neuro.who)
                        neuro.getPerception(
                          signalCount).setRecPower(
                                    0);
            // ActPower not used
            signalCount += 1;
        }

      neuro.standby = false;
}
```

Esta classe Act é um programa para cada neurónio de saída sendo 5 programas em paralelo logo aplica-se a cada neurónio. Cada neurónio que tinha disparado, como vimos na classe Neuron, colocou a variável “standby” verdadeira e como tal na classe Act é guardado os sinais “ouvidos” ou lidos no bus “Brain.InProself”. Se o neurónio não tinha ainda nenhum registo guarda todos os sinais ouvidos no bus. Se o neurónio já tem alguns sinais retira os que não lhe dizem respeito ou que não são iguais a ele próprio.

É aqui que os neurónios agem “writeBus()” sobre o corpo e o mundo com um vigor, cada um, que depende do género, no que respeita ao polegar no topo, com uma pequena diferença em vigor ou amplitude, que mais tarde se vai notando diferença nos géneros. Os Emotivos são mais energéticos que os Racionais.

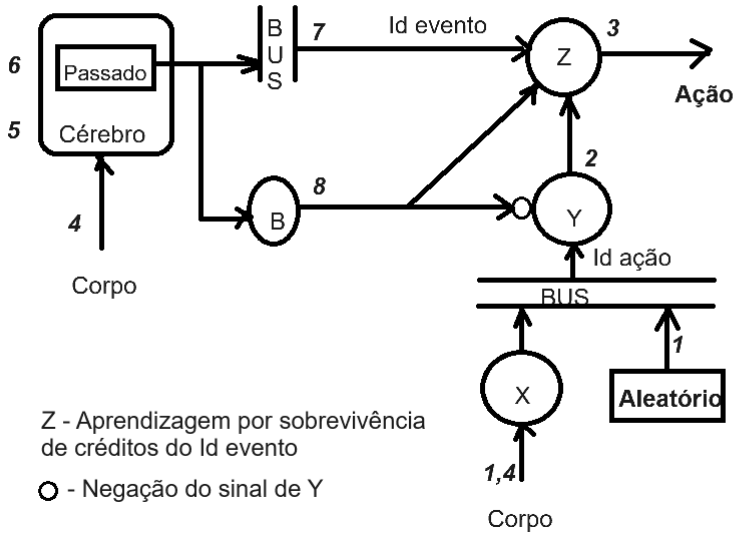
Por fim temos a definição do inverso de uma ação ou seja em sentido contrário à anterior para o caso de Dor e em sentido contrário ao lembrado para o caso de Saudade. Em que a ação a executar fica negativa.

Terminando a classe Act passemos para a classe que a antecede, ou seja a classe Cerebellum. Vamos assim estudar as classes desde a última a ser executada até a primeira a ser executada pela classe Liberu que é a classe Mind. Vamos por isso analisar por ordem, Act, Cerebellum, Sleep, Emotions, Remember e a classe que as mete em ordem, a classe Mind, o maestro do ciclo infinito de ações.

Na classe Cerebellum é feito uma cópia do sentido propriocetivo para que se, se receber no cerebelo, a identificação do neurónio a ser executado pela classe Act, o cerebelo fica a saber que sinais é que esse neurónio tem nas suas dendrites. Isto é com um número temporal de identificação pode o cerebelo saber que neurónios da classe Act vai “chamar”. E basicamente é só isso.

Podemos só analisar um esquema redesenhado do livro “Neurofisiologia sem Lágrimas”, página 187, sobre o cerebelo.

Cerebellum



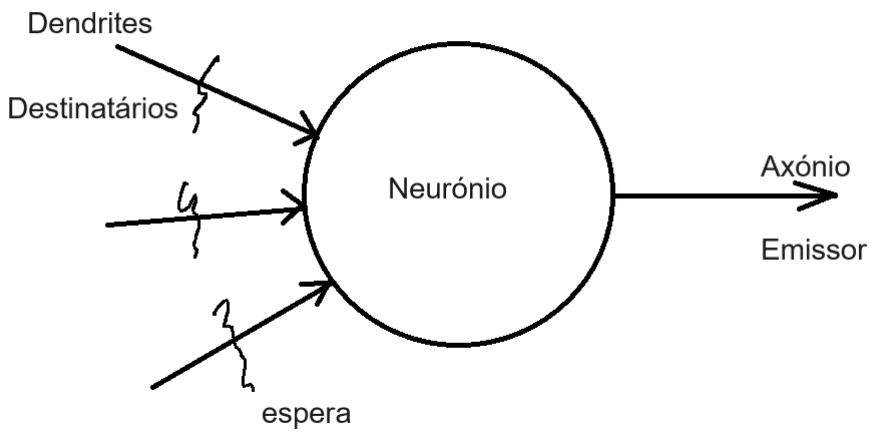
Podemos ver as seguintes etapas dinâmicas.

No início é gerado aleatoriamente, ou com influência genética, identificações de neurónios de ação, etapa 1, lançando as identificações no bus para fazer a difusão. Na etapa 2 o neurónio Y responde e conduz o neurónio Z a agir. Logo depois no próximo ciclo do cérebro é recebido, na etapa 4, um sinal propriocetivo dos músculos tanto no cerebelo X como no sentido propriocetivo do cérebro ou seja no passado, nas etapas 5 e 6. Aqui no passado é enviado para o bus do cerebelo a identificação do neurónio a atuar, “Id evento”. Por fim o neurónio Z aprende a responder à identificação “Id evento”, etapa 7, ao mesmo tempo que o neurónio B inibe o neurónio Y e favorece o neurónio Z, etapa 8.

Podemos imaginar o caso de o bebé habituar-se a fazer as suas necessidades no bacio e largar os instintos primários, por associações sentidas quando deve ir ao bacio.

Passemos então para a classe Sleep. Nesta classe de um sono “acordado”, faz-se um percurso no tempo, para trás, Saudade e para a frente, Ansiedade. É também nesta classe que a rede de ações, executadas no tempo, vai sendo construída.

Vamos pensar que, um neurónio guarda nas suas entradas, dendrites, os endereços ou identificação dos neurónios que já terão sido executados no momento seguinte ou ciclo seguinte ou seja guarda os seus destinatários, como se vê na figura.



Sendo assim, se recebermos o neurónio de início e enviarmos para a rede todas as suas informações dos destinatários que estão nas dendrites “learn” ou entradas guardadas no neurónio, obtemos a resposta dos neurónios que tiverem essa identificação. Passamos depois ao “write2” o testemunho ao neurónio destinatário de colocar na rede os seus destinatários. Progredimos assim no tempo.

```
listen = Brain.BusSleepPreviews2.iterator();
if (listen.hasNext())
      while (listen.hasNext()) { // Lê entrada bus
            signInput = listen.next();
            if (signInput.emitter == me) // "Ouvindo"
            for (i=0; i<time[me][0].learn.size();
                        i++)
                        write2(time[me][0].getLearn(i).
                                    emitter, signInput.actPower,
                                    signInput.pleasure +
                                    time[me][Brain.ProSelf].
                                    finalQuality);
      }
```

Para andar para trás, envia-se a identificação do neurónio inicial, e quem possuir nas suas dendrites essa identificação significa que o neurónio inicial esta à posteriori dos que estiveram a “ouvir”. Assim os neurónios que ouviram a identificação do neurónio inicial enviam no “write”, por sua vez, a sua identificação e o ciclo repete-se até se querer. No nosso caso foi de uma profundidade de 6 ciclos.

```
listen = Brain.BusSleepPreviews.iterator();
if (listen.hasNext())
      while (listen.hasNext()) { // Lê entrada bus
            signInput = listen.next();
            if (!time[me][0].newLearn(
                        Math.abs(signInput.emitter)))
                  // "Ouvindo"
                  write(me, signInput.actPower,
                              signIput.pleasure +
                              time[me][Brain.ProSelf].
                              finalQuality);
      }
```

A classe Sleep vai construindo a rede com todas as identificações dos neurónios que foram sendo executados e com a ajuda da variável lógica do neurónio “standby2” ficam guardados na rede de nome “learn”.

```
if (time[me][0].standby2) { // Se em espera
            time[me][0].standby2 = false;
      time[me][0].addLearn(Math.abs(Cortex.lastExec),
                  0, time[Math.abs(Cortex.lastExec)]
                              [Brain.ProSelf].finalQuality);
      System.out.println(Cortex.lastExec); //REMOVER
}
```

Analisada a classe Sleep passamos por ordem à classe Emotions. A classe Emotions recebe da classe Remember a identificação do neurónio mais semelhante e recente com o evento presente. Na classe Remember também se sabe se houve alguma situação de Dor, “ActualPain”, pelo que foi detetado na classe Mind. Sendo o caso de Dor a classe Emotions coloca a identificação como negativa para originar uma ação oposta na classe Act.

```
  // ######  Recente decisão ######
if (ActualPain && past.time[0][0].step > ActRandom)
      //
      Good = -Good; // Ato oposto de reflexo de Dor
      // ##############################
```

Na classe Emotions é definido o número de passos aleatórios “ActRandom” de valor 30 que é o suficiente para geral, quase sempre, todas as 5 ações possíveis.

Aqui também estão todas as expressões emotivas definidas, a nomear: expressar sons; emoção de novidade; emoção normal; emoção de feliz; emoção de triste; emoção de dor; emoção de medo e duas emoções compostas de “rir” e “falar”.

Esta classe Emotions têm uma segunda parte que é executada perto do fim do ciclo para incrementar a noção de passo dado ou relógio para o sistema funcionar, acrescentado uma unidade ao passo anterior, guardado em cada neurónio “step”. Nesta segunda parte são também aqui gerados os números aleatórios quando no útero. No caso de o robô sentir Dor não é feito o incremento e esse último passo será substituído por um passo sem Dor. Assim não se guardam as sensações e ações que provocaram Dor.

Passemos agora à classe Remember. Nesta classe são guardados todos os sentidos nos neurónios do presente e os outros neurónios chamam a rotina

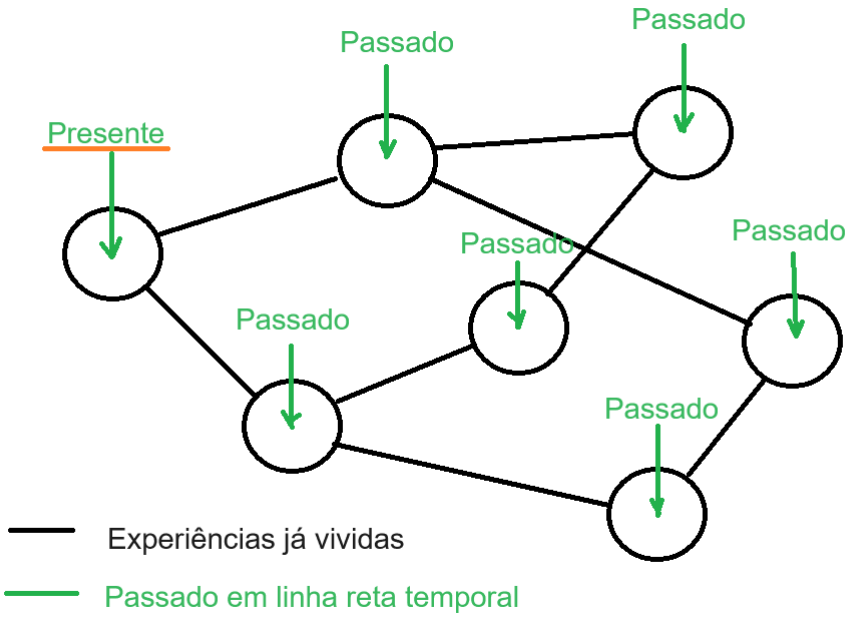
já vista “powerOfFire” para saber a identificação do neurónio temporal mais semelhante e recente comparando com o presente. E basicamente é só o que faz.

Para terminar entramos na classe Mind que não é mais do que o programa que mete todo o cérebro a funcionar. Aqui define-se a profundidade “PROFUNDITY” do deslocamento de tempo para a classe Sleep para saber quanto para a frente e para trás o sono acordado progride. No nosso caso temos uma profundidade de seis.

Aqui na classe Mind configura-se as entradas nos sentidos para cada robô com as suas noções de prazer (prazer esse que representa a primeira derivada do sinal, a segunda derivada do sinal serve para na sua transição para negativo, implicação Falsa, detetar o resultado da Ansiedade/Saudade na rede “learn” da classe Sleep). Por exemplo, se no sentido auditivo dá mais prazer ao robô os sons mais altos ou os mais baixos, se o robô no sentido visual gosta mais de determinada cor que outra, e por aí fora.

Temos toda a estrutura sincronizada através de barreiras “CyclicBarrier” que esperam que os diversos módulos sejam executados.

Terminada a análise com a classe Mind poderemos visualizar a estrutura de rede que vai sendo montada na classe Sleep com as ligações de aprendizagem “learn”.



Aqui vemos os neurónios da rede “learn” com as suas possíveis ligações de ações já vividas. Mas a qualquer momento, devido ao que sai do relembrado, classe Remember, podemos ser colocados em qualquer parte da rede, sabendo porém que tanto a rede linear do Remember como a rede dispersa do “learn” são redes contínuas no tempo, não há pontas soltas na rede.

Escolhido um ponto da rede, só se pode progredir num sentido. Sabendo o sentido correto não se pode voltar para trás no sentido entre dois neurónios. Pois para a frente e para trás são dois caminhos diferentes, sensações diferentes, seria outro nó ou neurónio da rede.

Temos ainda, para terminar o capítulo, que esclarecer as prioridades que o robô dá às ações que vai executar:

1 – mais prioritária, as ações do dono ao robô ou o que despertar à atenção do robô.

2 – se o robô sentir Dor.

3 – se o robô sentir Ansiedade ou Saudade.

4 – menos prioritária, o instinto do robô dado pela aprendizagem.

Devido aos neurónios guardarem quem são e para onde vão, pode-se perder neurónios que os restantes continuam a funcionar, ficando o todo com menos potencialidades.

Diz-se que na doença de Alzheimer os neurónios vão-se perdendo desde o presente para trás, no seu passado, a caminho para a juventude.

24 Ideias Soltas

Muitas das vezes, nós temos decisões influenciadas pelo inconsciente e nem nos apercebemos conscientemente. Daí nós limitamo-nos 'desesperadamente' a justificar as nossas decisões aos outros.

Assistimos, por assim dizer, ao desenrolar do nosso destino.

Poucas são as pessoas que sabem lidar com a felicidade, mais são as que sabem lidar com a adversidade.

Os jovens pensam mais rápido do que os idosos porque tem menos opções de respostas aprendidas no seu passado.

Nem sempre são as melhores opções e respostas comparadas com as de um mais experiente idoso. Não interessa a quantidade de ideias mas sim a qualidade das mesmas...

Somos, todos nós seres vivos com cérebro, mais inteligentes e espertos do que todos os outros pensam, que somos... os outros somos nós!

Nós é que não damos crédito facilmente aos outros... o que é pena?!

A dor de uma formiga é igual à de um ser humano pois basta um neurónio para a sentir e não é algo que dependa do tamanho do corpo! É discreta e não analógica...

Se queres conhecer a morte apresento-te o sono profundo...

Como a população de seres humanos ia aumentando, pensou-se e muito bem (dados os dados da época) que antigamente existiriam menos pessoas resultando em um Adão e uma Eva. Formava-se então uma pirâmide em que no topo se encontravam esses dois seres e na base da pirâmide se encontrava a atualidade.

Pois agora, com os nossos conhecimentos a pirâmide está de pernas para o ar. Ou seja em baixo da pirâmide de um único elemento (cada um de nós, seres vivos) encontra-se o milagre da vida, a consciência relativa de cada ser perante o que o rodeia e sente devido a milhões e milhões de experiências de

tentativa e erro, capacidade de adaptação e número de gerações que cada ser já teve (exemplo, se os ascendentes reproduzirem todos mais tarde - instinto materno e paterno lento - têm menos gerações e por consequência menos cultura corporal do que se os que se apressaram).

Assim no topo da pirâmide invertida estão de início, os átomos que nos compõem, depois as moléculas, depois estruturas biológicas, órgãos e corpos que são todos felizmente diferentes, pois na diversidade e quantidade de antepassados geracionais faz de nós, seres, mais ou menos hábeis...

Hoje fui procurar a palavra 'relação' no dicionário de português-inglês. E não é que abri o livro na página correta à primeira. E não só abri na página correta mas era a palavra que aparecia no topo da página como referência. Ou seja que grande coincidência...então estou a pensar na palavra e de repente leio-a logo...

Eu acredito em coincidências divinas, que não passam de coincidências sem causas artificiais ou mão humana. Não significam nada, só que o nosso cérebro captou como associação de eventos semelhantes!!

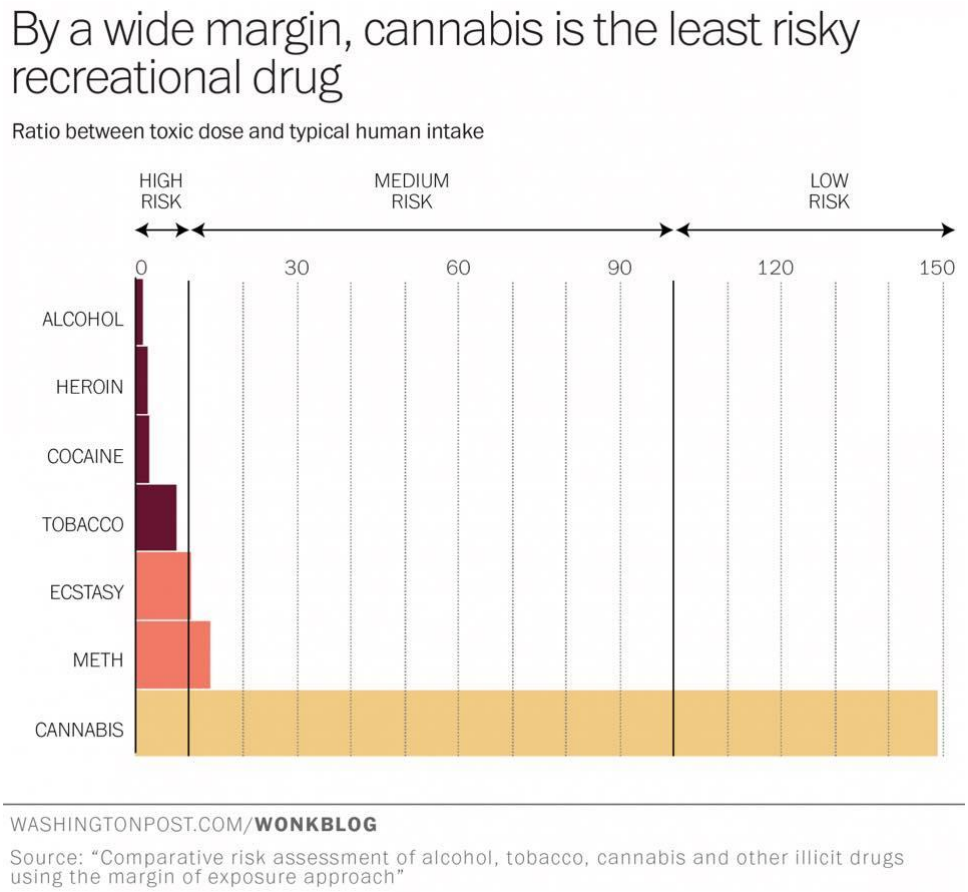
Quatro cabeças diferentes chegam mais longe que duas cabeças diferentes, duas cabeças diferentes chegam mais longe que uma cabeça e uma cabeça com mil livros chega mais longe que mil cabeças com um livro cada uma...

Deve-se educar os outros, incluindo animais, pelo bom, ou bem feito, e não pelo castigo, pelo mal feito. Errar é animalesco...

Só para refletir:

"O sábio não procura o prazer, mas sim a ausência de sofrimento".

Ao contrário do Cérebro...



O cérebro pulsa como o coração, respira como os pulmões ou seja tem um ciclo. O ciclo dos pulmões é de alguns segundos, do coração cerca de 1 segundo. O ciclo do cérebro é igual à rapidez com que se consegue carregar num botão depois de aceder uma luz (há programas para testar essa rapidez - a minha reação foi de 203 milissegundos).

Por isso, cerca de 5 vezes por segundo o cérebro: memoriza todos os 5 sentidos mais o sentido do próprio corpo (propriocetivo), que ficam registadas para sempre no Passado do cérebro (Cultura); sente prazer ou desconforto; exprime emoções; e age, instintivamente idêntico ao no Passado para situações semelhantes no ambiente sentido pelo cérebro, pela genética, que só serve para construir o corpo, e da Saúde do corpo no presente imediato.

Ou seja as nossas ações dependem de 4 coisas: genética; ambiente; passado/Cultura; e Saúde.

O cérebro é um animal de hábitos. Repete sempre um passo do seu passado mais semelhante ao presente. Por isso devemos alimentar o corpo

evidentemente, e o cérebro com memórias prazerosas. Um corpo bonito ou forte ou culto são os mais cobiçados.

Os neurologistas estudam o Hardware do cérebro, os engenheiros no projeto www.vitaliberu.pt estudam o Firmware do cérebro, os psicólogos estudam o Software do cérebro e os sociólogos estudam o Freeware do cérebro.

Eu sei que Deus existe porque vejo a Sua obra. A Natureza. Mas não o vejo como figura Humana, mas como amor, paixão, cuidar, educar, curar, melhorar, crescer, entre infinitos sentimentos que influenciam os meus pensamentos e ações.

Se quiserem, cruzem ou entrelacem os vossos dedos, como se estivessem a pedir perdão a Deus.

Os que tiverem o polegar direito por cima e o dedo mindinho esquerdo por baixo são os racionais.

Os que tiverem o polegar esquerdo por cima e o dedo mindinho direito por baixo são os emotivos.

Os casais com química são quando se juntam os opostos, racional-emotivo.

Os casais emotivos-emotivos ou racionais-racionais são como um casal como duas chaves-de-fendas ou com dois martelos.

Enquanto que os casais racionais-emotivos são como um casal com um martelo e uma chave-de-fendas. Logo mais poderosos e felizes, porque se surpreendem um ao outro. O que um não sabe, sabe o outro e o que o outro não sabe, sabe o primeiro. Logo vão mais longe.

Conclusão vale mais ter 2 ferramentas diferentes do que 2 ferramentas iguais.

Sejam felizes, vão sempre a tempo. Bela natureza: duas personalidades diferentes para dois géneros.

O instinto é a certeza, que não se esqueceu...

O cérebro/mente é composto por 10 classes escritas em linguagem Java..

(Dinâmicas - 6 classes - 3 imutáveis e 3 variáveis com os gostos e expressões emocionais de cada ser)

Mente (gostos)

Relembrar

Emoções (expressões emocionais)

Sono

Cerebelo

Ação (que também se encontra no cerebelo) - (move músculos e realização das expressões emocionais)

(Não dinâmicas - 4 classes - 3 imutáveis e 1 variável com característica por espécie)

Cérebro (variável)

Córtex

Neurónio

Sinal

E é composto por 5 partes de construção...

1 - difusão

2 - dinâmica

3 - prazeres

4 - atenção

Veja no site VitaLiberu.pt a implicação das 4 partes…

DADOS do corpo humano.... 3 variáveis:

Supondo que a velocidade dos impulsos nervosos é de 100km/h

E que temos um tempo de resposta de um neurónio de 40ms podemos alcançar 2,7m de altura máxima...

Supondo que o ser humano tem em altura máxima de 2m e que o tempo de reação do neurónio é de 40ms basta-nos ter uma velocidade dos impulsos inferior de 180km/s.

E por último, se o ser humano tem uma altura de 2m e a velocidade dos impulsos de 100km/h basta-nos ter um tempo de resposta do neurónio inferior a 72ms.

Por isso temos uma grande margem de diversificações para podermos ser todos diferentes, mas todos iguais...

A Inteligência Artificial das redes neuronais não passam de adivinhos!? A mesma informação de entrada pode mudar de categoria ao longo do tempo!? E é com base desses Adivinhos que se tomam decisões muitas vezes vitais...

Temos algo de mais concreto em que não é a informação que entra, é processada e age. Mas sim a informação que saí, depois entra, e só no fim é que é processada!!! É como não ser o Sol que anda à volta da Terra, mas sim o contrário... é como o eco, radar, sonar..

Eu acho que os humanos vão desaparecer por perda de cultura. Pois vai deixar de existir alguém que saiba reparar um robô ou máquina ou mesmo semear alimentos, por se entregarem aos prazeres da vida...

Faz lembrar o filme da Máquina do tempo em que no futuro os monstros subterrâneos (que sacrificavam um “elfo” de vez em quando) são os robôs que fazem tudo e o “elfos” serão os humanos... a biblioteca que aparece no filme é toda a cultura perdida... só que o sistema de sobrevivência não dura para sempre... logo...logo... só Deus sabe...

Quem é que nunca teve um sonho em que queria correr ou gritar e ficar desesperado, pois “sentimos” ou melhor não nos sentimos a correr ou a sair a voz para o grito?!

Pois bem, a explicação para isso é extremamente simples de entender…

Ao músculos do nosso Corpo estão “paralisados” durante o sono, logo o cérebro não recebe informação de retorno desses músculos (inclusivamente os músculos vocais). Apesar de ter tido a vontade de quere correr ou gritar, não houve a confirmação habitual, de retorno, por parte dos músculos e até o retorno das nossas próprias ações, quer vermo-nos a correr ou ouvirmo-nos a gritar. Temos então um certo desespero.

O cérebro já está formado antes do Universo existir. Pois a linguagem de Deus é única que existe sempre, a Matemática, e um programa do cérebro, como todos os outros programas, fazem parte da Matemática. Logo é intrinsecamente único para todos os seres do Universo e funcional.

O cérebro humano guarda uma imagem e perde quatro imagens. Vemos 25 imagens por segundo e temos um tempo de resposta do cérebro de cerca de 200 milissegundos ou um quinto de segundo, logo só memorizamos 5 imagens por segundo e perdemos 20 imagens.

Se a velocidade de reação do cérebro for de 200 milissegundos temos, com cada neurónio de 40 milissegundos de reação, um total mínimo de 5 neurónios a disparar. Que no nosso sistema poderão ser: 2 no cérebro, “sentido” e no Remember; 2 no cerebelo, Cerebellum e Act; e 1 no corpo, músculos.

Os cérebros funcionam todos iguais mas com inúmeras configurações de “gostos” ou o que lhes dá prazer. Uns gostam do verde ou uma determinada comida outros gostam do vermelho e outro tipo de comida, etc.

Os cérebros são todos “adictos” aos “gostos”. Uns dão para os objetos, transtorno obsessivo ou compulsivo. Outros dão para a comida, trabalho, co dependência, drogas naturais e químicas, dinheiro, desporto, compras, jogos, narcisismo, etc.

Tudo isso depende muito dos “gostos” e da educação dada, a esse cérebro, e exemplos presenciados.

A consciência é uma dádiva da natureza biológica de nos fazer perceber e sentir ou melhor assistir ao resultado prévio das decisões do cérebro. E não afetando, a consciência, as ações posteriores do cérebro.

Os aparelhos eletromecânicos não têm consciência, só cérebro.

Os cérebros têm como que múltiplas personalidades consoante as emoções ou necessidades naturais ou induzidas artificialmente por drogas. Pois é como se só tivéssemos acesso ao subconjunto de memórias correspondentes a essa personalidade sem ligações com as outras personalidades, favorecendo a falta de recordação entre elas.

A doença de Alzheimer deixa de ter memórias desde o presente evoluindo para trás no tempo. Neste sistema com a disposição da memória em linha temporal é fácil de encaixar esse entendimento da doença.

Ingerir álcool potencia nesta arquitetura de sistema ao não registo dos sentidos presentes (classe Remember) sem influenciar o funcionamento do cérebro, pois nós para agir só dependemos do passado e não do presente.

O córtex pré frontal dos bebés não tem muita atividade, pelo que se sabe, e ajusta-se a este sistema pois o passado quando se nasce está muito reduzido. Sobrando ainda muito por preencher.

Neste sistema, não transita nenhuma informação de um passo ou ciclo para o outro. É como se estivesse sempre pronto para o imprevisto sem remorsos. É um sistema aberto que se fecha com o mundo e corpo em redor.

Este livro dá a conhecer a inteligência NATURAL, como novo paradigma contrário ao da Inteligência Artificial. A Inteligência Artificial cria uma inteligência fictícia resolvendo problemas complicados como jogar xadrez, linguística, contabilidade, etc. Até mesmo problemas mais simples, mas todos com o ponto de vista do problema a resolver, ensinando nós ao computador, o problema e como o ser humano resolve esse problema ou seja, imita artificialmente o resultado do comportamento do cérebro/corpo.

Ora a Inteligência NATURAL resolve, e é uma descoberta e não uma invenção, o, que até agora foi um enigma, comportamento do próprio cérebro e da sua essência que, neste caso, é como no cérebro real, não de natureza biológica mas, de tecnologia matemática, programação, eletrónica e eletromecânica. Só então será possível, esta descoberta, poder aprender a por exemplo: jogar xadrez. Depois de aprender as regras e conseguir jogar xadrez, jogará, bem ou mal, dependendo do treino.

As verdades são como peças do Puzzle, e quantas mais peças tivermos mais facilmente veremos a Verdade de Deus… e se conseguirmos Olhar o quadro do grande Puzzle vemos que é, o que Deus representa em nós através de toda a influência de todo o Universo, e do que o nosso Corpo e Cérebro absorve, demonstrando-O através das nossas ações.

25 "Act"

```
/*
 * Act.java
 *
 * Created on 24 of June 2007, 1:15
 */

/**
 * web site: www.vitaliberu.pt
 * email: sysliberal@gmail.com
 * @author Paulo Roque Silva
 */
import java.util.Iterator;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;

public class Act implements Runnable {
    public static boolean     Thumb;
    // Se verdade polegar esquerdo em cima ou
    // se falso polegar direito em cima
    Neuron          neuro; // Neurónio
    CyclicBarrier   bar; // Barreira de passos
    Iterator<Signal> listen; // Lê do bus
    Signal              signInput; // Lê sinal
    Signal              []outSignal; // Saída de ações
      boolean                 invert;
      public static boolean speak;

    /** Creates a new instance of Act */
    public Act(CyclicBarrier barr, Neuron ne,
            Signal []outSig) {

        bar       = barr;
      neuro     = ne;
        outSignal = outSig;
        Thumb = false;
        speak = false;
    }

    public int invertActDor(int actID) {

      switch (actID) { // Muda a ação
      case Brain.DRINK:
            return(Brain.TURN_LEFT +
                        (int)Math.rint(Math.random()));
      case Brain.WALK: {
            return(Brain.TURN_LEFT +
                        (int)Math.rint(Math.random()));
      }
      case Brain.TURN_LEFT:
            return(Brain.TURN_RIGHT);
      case Brain.TURN_RIGHT:
            return(Brain.TURN_LEFT);
      case Brain.EAT:
            return(Brain.TURN_LEFT +
                        (int)Math.rint(Math.random()));
      default:
```

```
            System.out.print("ERROR Act (actID)...");
            return(Brain.WALK);
      }
    }

    public void writeBus(int idAct, float apwr) {

            outSignal[idAct].setActive(true);
            outSignal[idAct].setActPower(apwr);
      System.out.print("ActPower= "); // REMOVER
      System.out.println(apwr); // REMOVER
      speak = false;
      if (Liberu.Robot == 0 && Cortex.nLastSpeak >=
                  Emotions.NSpeakRandom) {
            if (idAct == Brain.Vogal1) {
                  Mind.clip.setMicrosecondPosition(0);
                  Mind.clip.start();
              speak = true;
            }
            if (idAct == Brain.Vogal2) {
                  Mind.clip2.setMicrosecondPosition(0);
                  Mind.clip2.start();
              speak = true;
            }
            if (idAct == Brain.Vogal3) {
                  Mind.clip3.setMicrosecondPosition(0);
                  Mind.clip3.start();
              speak = true;
            }
      }
    }

    public void run() {
      int         signalCount, // Conta os sinais do bus
                  signalEmitter, // Guarda o emissor
                  credit, // Crédito
                  actE, // Crédito
                  actR; // crédito actPower

        while (!Neuron.end && !bar.isBroken()) {
            try {
                bar.await();
            }
            catch (InterruptedException ex) {
                return;
            }
            catch (BrokenBarrierException ex) {
                return;
            }
            if (neuro.standby && neuro.step <
                        Emotions.ActRandom) {
                  // Aprendisagem por sobrevivência
                  signalCount = 0;
                  listen = Brain.BusInProSelf.iterator();
                  if (neuro.perception.size() == 0)
                    while (listen.hasNext()) {
                        signInput = listen.next();
                        neuro.addPerception(
                                    signInput.emitter,
                                    signInput.actPower,
                                    signInput.pleasure);
```

```
                System.out.println(
                            neuro.who); // REMOVER
                System.out.print(
                      "Add dendrite Body fired=");
                // REMOVER
                System.out.println(
                            signInput.actPower);
                // REMOVER
                signalCount += 1;
            }
        else
            while (listen.hasNext()) {
                signInput = listen.next();
                if (signInput.actPower >=
                            Brain.MinSense)
                      if (signalCount != neuro.who)
                            neuro.getPerception(
                              signalCount).setRecPower(
                                                  0);
                signalCount += 1;
            }

        neuro.standby = false;
    }
    // ######################## Act ################
    neuro.clearSignals();
    signalCount = 0;
    signalEmitter = 0;
    invert = false;
    actE = 0;
    actR = 0;
    listen = Brain.BusInAct.iterator();
    while (listen.hasNext()) {
        signInput = listen.next();
        signalEmitter = signInput.emitter;
        if (signInput.emitter < 0)
              invert = true;
        if (neuro.perception.size() > 0)
              if (signInput.actPower >
              Brain.MinSense) {
                    neuro.getPerception(signalCount
                                ).setActive(true);
              if (signalCount == neuro.who) {
                    actR = (int)signInput.actPower;
                    actE = (int)signInput.pleasure;
              }
              }
        signalCount += 1;
    } // Bus vazio

    neuro.powerToAct();
    if (neuro.fire) {
        System.out.print(
                    "NeuroAct Body fired............");
        // REMOVER
      System.out.println(neuro.who);
      // REMOVER
      System.out.println(signalEmitter);
      // REMOVER
      if (Thumb)
        credit = actE;
```

```
                else
                  credit = actR;
                System.out.println(credit); // REMOVER
                if (invert)
                  writeBus(invertActDor(neuro.who), credit);
                else
                    writeBus(neuro.who, credit);
            }
        } // Fim ciclo While
    }
}
```

26 “Brain”

```
/*
 * Brain.java
 *
 * Created on 6 of December 2006, 1:34
 */

/**
 *
 * @author Paulo Roque Silva
 */

import java.awt.Color;
import java.util.Iterator;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.ConcurrentLinkedQueue;
import javax.swing.*;
import icommand.nxt.Motor;
import icommand.nxt.Battery;
import icommand.nxt.LightSensor;

public class Brain {
 // Constantes do cérebro
    static final int  NEURONS = 5000; // Número passos
    static final int  SENSES = 6; // Número sentidos
    // com propriocetivo guardados no passado
    static final int  ACTS   = 5; // Número de ações robô
    static final int  MOVES = 15; // Número de movimentos
 // Constantes do corpo
    static final float MaxInSense = 255.0F;
    // Máxima amplitude nos sensores e na ação
    static final float MinSense = 1.0F;
    // Mínima sensação analisada
    static final int  NanalogTaste = 3,
    // Número de sinais para cada sentido
                      NanalogVision= 3,
                      NanalogTact  = 1,
                      NanalogSound = 7,
                      NanalogSmell = 64,
                      NanalogProSelf=MOVES;
    static final int  NanalogTasteR = 1,
                          NanalogVisionR= 2,
                          NanalogTactR  = 1,
                          NanalogSoundR = Mind.BANDS,
                          NanalogSmellR = 3;
    static final int  NanalogTasteN = 1,
                          NanalogVisionN= 1,
                          NanalogTactN  = 1,
                          NanalogSoundN = Mind.BANDS,
                          NanalogSmellN = 0;
    static final int  AWALK     = 0,
                          ABACK     = 1,
                          BWALK     = 2,
                          BBACK     = 3,
                          SPEAK     = 4,
                          //  FIM DOS ACTRND
                          Vogal1    = 5,
```

```
                        Vogal2     = 6,
                        Vogal3     = 7,
                        ENERGY     = 8,
                        PAIN       = 9,
                        New        =10,
                        Happy      =11,
                        Sad        =12,
                        Afraid     =13,
                    Motive     =14,
                        //  FIM DOS MOVES

                    // Variáveis para os sentidos
                        NO_TACT    = 0, // Não sente obstáculo
                        TACT            = 1, // Sente obstáculo

                        Taste           = 0,
                        Vision     = 1,
                        Tact            = 2,
                        Sound           = 3,
                        Smell           = 4,
                        ProSelf    = 5;

    static final int  DRINK      = 0,
                    EAT        = 1,
                    WALK       = 2,
                    TURN_LEFT  = 3,
                    TURN_RIGHT= 4;
                    //  FIM DOS ACTRND
 // Variáveis do Brain
    Cortex              past; // Rede de memória
    Emotions          decision;
    static ConcurrentLinkedQueue<Signal>
                        BusInTaste,
                        BusInVision,
                              BusInTact,
                              BusInSound,
                              BusInSmell,
                              BusInProSelf,
                        BusSleepPreviews,
                        BusSleepOut,
                        BusSleepPreviews2,
                        BusSleepOut2,
                              BusInAct;
    static Boolean      BackHome; // Se a caminho de casa
    static int          Best; // Melhor caminho

// Variáveis do corpo
    public static Signal []bodyAction;
    // Ações no corpo do robô
    public static boolean Energy; // Motivação para energia
      Iterator<Signal>  listen; // Ouve sinais dos buses
      Signal                    signInput; // Guarda o sinal


// Variáveis da simulação
    CyclicBarrier     stepBarrierAct;
    CyclicBarrier     stepBarrierDecision;
    CyclicBarrier     stepBarrierDecision2;
    CyclicBarrier     stepBarrierPast;
    CyclicBarrier     stepBarrierCerebellum;
```

```
    CyclicBarrier      stepBarrierSleep;

    JTextField         texto; // Tecla primida
    static boolean     Saudade, Ansiedade;

    /** Creates a new instance of Brain */
    public Brain(JTextField txt) {
      int i;

        texto          = txt;
      past           = new Cortex(SENSES, ACTS, NEURONS);
      bodyAction     = new Signal[MOVES];
        for (i=0; i<bodyAction.length; i++)
            bodyAction[i] = new Signal(i, 0, 0);
        setEnergy(false);
        BackHome       = false;

        BusInTaste    = new ConcurrentLinkedQueue<Signal>();
        BusInVision   = new ConcurrentLinkedQueue<Signal>();
        BusInTact     = new ConcurrentLinkedQueue<Signal>();
        BusInSound    = new ConcurrentLinkedQueue<Signal>();
        BusInSmell    = new ConcurrentLinkedQueue<Signal>();
        BusInProSelf = new ConcurrentLinkedQueue<Signal>();
        BusInAct       =
                  new ConcurrentLinkedQueue<Signal>();
        BusSleepPreviews  =
                  new ConcurrentLinkedQueue<Signal>();
        BusSleepOut              =
                  new ConcurrentLinkedQueue<Signal>();
        BusSleepPreviews2 =
                  new ConcurrentLinkedQueue<Signal>();
        BusSleepOut2       =
                  new ConcurrentLinkedQueue<Signal>();
        Ansiedade = false;
        Saudade = false;
    }

    public CyclicBarrier initiatePastTime() {

      // Cria os processos do Remember
        stepBarrierPast    =
                  new CyclicBarrier(past.nNeuronsTime+1);
          for (int n=0; n < past.nNeuronsTime; n++)
            new Thread(new Remember(stepBarrierPast,
                        past.time[n],
                        decision)).start();
        return stepBarrierPast;
    }

    public CyclicBarrier initiateDecision() {

      // Cria os processos do Emotions
        stepBarrierDecision   = new CyclicBarrier( 2 );
        stepBarrierDecision2 = new CyclicBarrier( 2 );
        decision = new Emotions(stepBarrierDecision,
                  stepBarrierDecision2, past, bodyAction,
                  texto);
          new Thread(decision).start();
        return stepBarrierDecision;
    }
```

```
    public CyclicBarrier initiateSleep() {

    // Cria os processos do Sleep
      stepBarrierSleep  =
                new CyclicBarrier(past.nNeuronsTime + 1 );
        for (int n=0; n < past.nNeuronsTime; n++)
          new Thread(new Sleep(stepBarrierSleep,
                        past.time, n)).start();
      return stepBarrierSleep;
  }

  public CyclicBarrier initiateCerebellum() {

    // Cria os processos do Cerebellum
      stepBarrierCerebellum  = new CyclicBarrier( 2 );
        new Thread(new Cerebellum(stepBarrierCerebellum,
                past.hemisphereLeft, past.hemisphereRigth,
                        past, past.nNeuronsTime)).start();
      return stepBarrierCerebellum;
  }

  public CyclicBarrier initiateAct() {

    // Cria os processos do Act
      stepBarrierAct  =
                new CyclicBarrier(past.nNeuronsAct + 1);
    for (int i=0; i<bodyAction.length; i++)
          bodyAction[i] = new Signal(i, 0, 0);
        for (int n=0; n < past.nNeuronsAct; n++)
          new Thread(new Act(stepBarrierAct,
                        past.actNode[n], bodyAction)).start();
      return stepBarrierAct;
  }

  public CyclicBarrier initiateDecision2() {

      return stepBarrierDecision2;
  }

  public void clearBusesIn(boolean proSelf) {

    while (!BusInTaste.isEmpty())
          BusInTaste.poll();
    while (!BusInVision.isEmpty())
          BusInVision.poll();
    while (!BusInTact.isEmpty())
          BusInTact.poll();
    while (!BusInSound.isEmpty())
          BusInSound.poll();
    while (!BusInSmell.isEmpty())
          BusInSmell.poll();
    if (proSelf)
          while (!BusInProSelf.isEmpty())
                BusInProSelf.poll();
  }

  public void clearBusSleepPreviews() {

    while (!BusSleepPreviews.isEmpty())
          BusSleepPreviews.poll();
  }
```

```
    public void clearBusSleepPreviews2() {

      while (!BusSleepPreviews2.isEmpty())
            BusSleepPreviews2.poll();
    }

    public void pollSleepOut() {

      while (!BusSleepOut.isEmpty()) // Esvazia bus
            BusSleepOut.poll();
    }

    public void pollSleepOut2() {

      while (!BusSleepOut2.isEmpty()) // Esvazia bus
            BusSleepOut2.poll();
    }

    public void transferOutIn() {

      while (!BusSleepPreviews.isEmpty()) // Esvazia bus
            BusSleepPreviews.poll();

      listen = BusSleepOut.iterator(); // Transfere
      while (listen.hasNext()) { // L□ todo o bus
            signInput = listen.next();
            BusSleepPreviews.add(new Signal(
                        signInput.emitter, signInput.actPower,
                        signInput.pleasure));
      }
      pollSleepOut();
    }

    public void transferOutIn2() {

      while (!BusSleepPreviews2.isEmpty()) // Esvazia bus
            BusSleepPreviews2.poll();

      listen = BusSleepOut2.iterator(); // Transfere
      while (listen.hasNext()) { // L□ todo o bus
            signInput = listen.next();
            BusSleepPreviews2.add(new Signal(
                        signInput.emitter, signInput.actPower,
                        signInput.pleasure));
      }
      pollSleepOut2();
    }

    public void decide(int pr) {  // Saudade

      Signal                        signInput;
        Iterator<Signal>      listen;

      listen = BusSleepOut.iterator();
      while (listen.hasNext()) {
            signInput = listen.next();
            if (signInput.pleasure / pr > Best) {
                  Best = signInput.pleasure / pr;
                  Mind.setBom((int)signInput.actPower);
                  Saudade = true;
```

```
            }
        }
    }

    public void decide2(int pr) {    // Ansiedade

      Signal                          signInput;
        Iterator<Signal>      listen;

      listen = BusSleepOut2.iterator();
      while (listen.hasNext()) {
            signInput = listen.next();
            if (signInput.pleasure / pr > Best) {
                  Best = signInput.pleasure / pr;
                  Mind.setBom((int)signInput.actPower);
                  Ansiedade = true;
            }
        }
    }

    public int addStep() {

      return past.addDataActual();
    }

    public void idAct1(int idAction) {

      decision.idAct1Action(idAction);
    }

    public static boolean getEnergy() {

      return Energy;
    }

    public static void setEnergy(boolean state) {

      Energy = state;
    }

    public static void setBackHome(boolean state) {

      BackHome = state;
    }

    public void motivationNXT(int []senseProSelf, int pw) {

        if (BackHome)
            senseProSelf[Motive] = pw;
    }

    public void feelTaste(int []sense, int []plsr) {
      int       m;

      for (m=0; m < sense.length; m++)
            BusInTaste.add(new Signal(m, sense[m],
                        plsr[m]));
    }

    public void feelVision(int []sense, int []plsr) {
      int       m;
```

```
    for (m=0; m < sense.length; m++)
                BusInVision.add(new Signal(m, sense[m],
                            plsr[m]));
  }

  public void feelTact(int []sense, int []plsr) {
    int       m;

    for (m=0; m < sense.length; m++)
          BusInTact.add(new Signal(m, sense[m],
                      plsr[m]));
  }

  public void feelSound(int []sense, int []plsr) {
    int       m;

    for (m=0; m < sense.length; m++)
          BusInSound.add(new Signal(m, sense[m],
                      plsr[m]));
  }

  public void feelSmell(int []sense, int []plsr) {
    int       m;

    for (m=0; m < sense.length; m++)
          BusInSmell.add(new Signal(m, sense[m],
                      plsr[m]));
  }

  public void feelProSelf(int []sense, int []plsr) {
    int       m;

    for (m=0; m < sense.length; m++)
          BusInProSelf.add(new Signal(m, sense[m],
                      plsr[m]));
  }


  public void act(Signal result[]) {


    for (int i=0; i<bodyAction.length; i++) {
          result[i].setActive(bodyAction[i].active);
          result[i].setActPower(bodyAction[i].actPower);
    }
  }

  public int proSelfNXT(int []senseProSelf,
          LightSensor light, int pw) {
    int   plsr;

    plsr = 0;
      if (bodyAction[ENERGY].active &&
                Battery.getVoltageMilliVolt() <= 8000) {
          plsr = pw;
          }
      if (Battery.getVoltageMilliVolt() <= 8000) {
          bodyAction[ENERGY].setActive(true);
          plsr = pw;
          }
```

```
        if (bodyAction[EAT].active)
            senseProSelf[EAT] = pw;

        if (bodyAction[ENERGY].active)
            senseProSelf[ENERGY] = pw;
      if (bodyAction[ENERGY].active &&
                  past.time[0][0].step >=
                  Emotions.ActRandom) {
            Energy = true;
            Liberu.jButtonDAct.setBackground(Color.RED);
      }

      if (Motor.A.getTachoCount() > 0 &&
                  Motor.B.getTachoCount() < 0) { // em graus
            senseProSelf[TURN_LEFT] = (int)bodyAction[Brain.
                                        TURN_LEFT].actPower;
      }
      if (Motor.B.getTachoCount() > 0 &&
                  Motor.A.getTachoCount() < 0) {
            senseProSelf[TURN_RIGHT] = (int)bodyAction[Brain.
                                        TURN_RIGHT].actPower;
      }
        if ((Motor.A.getTachoCount() > 0) &&
                  (Motor.B.getTachoCount() > 0)) {
            senseProSelf[WALK] = (int)bodyAction[Brain.
                                      WALK].actPower;
        }
        Motor.A.resetTachoCount();
        Motor.B.resetTachoCount();

        if (bodyAction[Brain.Motive].active)
                  senseProSelf[Brain.Motive] = pw;

        for (int i=0; i<bodyAction.length; i++) {
            bodyAction[i].setActive(false);
            bodyAction[i].setPleasure(0);
        }
        return plsr;
    }

    public int proSelfRobot(int []senseProSelf, int pw) {
      int   plsr;
      boolean tact;
      String s = "0";

            try {
            Liberu.output.write((byte)99);
            Liberu.output.write('\n');
            while (!Liberu.input.ready())
                Thread.sleep(1);
            s = Liberu.input.readLine();
            }
      catch (Exception e) {
            System.err.println(e.toString());
      }
            tact = false;
            if (!s.isEmpty())
                  tact = (s.charAt(0) == 1);

      plsr = 0;
```

```
        if (BackHome && tact) {
            bodyAction[ENERGY].setActive(true);
            plsr = pw;
            }

        if (Energy) {
            plsr += (pw - Mind.local) * 10;
            Liberu.jButtonDAct.setBackground(Color.RED);
            bodyAction[ENERGY].setActive(true);
        }

        if (bodyAction[ENERGY].active) {
            bodyAction[ENERGY].setActive(false);
            senseProSelf[ENERGY] = pw; // Sente energia
        }
      if (bodyAction[ENERGY].active &&
                  past.time[0][0].step >=
                  Emotions.ActRandom) {
            Energy = true;
            Liberu.jButtonDAct.setBackground(Color.RED);
      }

        if (bodyAction[Brain.Motive].active) {
                  senseProSelf[Brain.Motive] = pw;
            plsr += pw;
        }

        for (int i=0; i<bodyAction.length; i++) {
            if (bodyAction[i].active) {
                  senseProSelf[i] = (int)bodyAction[i].
                              actPower;
                bodyAction[i].setActive(false);
            }
            bodyAction[i].setPleasure(0);
        }
        return plsr;
    }

    public int proSelf(int []senseProSelf,int []senseTaste,
            int pw) {
      int   plsr=0;

        // Come quando tem fome e sente comida
        if (bodyAction[EAT].active  &&
                  senseTaste[World.FOOD-1] == pw ||
                  bodyAction[EAT].active &&
                  past.time[0][0].step <=
                  Emotions.ActRandom) {
            senseProSelf[EAT] = pw; // Sente comida
            plsr += pw;
            bodyAction[EAT].setActive(false);
            }
        if (Energy) {
            plsr += (pw - Mind.local) * 10;
            Liberu.jButtonDAct.setBackground(Color.RED);
            bodyAction[ENERGY].setActive(true);
        }

        if (bodyAction[ENERGY].active) {
            bodyAction[ENERGY].setActive(false);
            senseProSelf[ENERGY] = pw; // sense energy
```

```
        }
      if (bodyAction[ENERGY].active &&
                  past.time[0][0].step >=
                  Emotions.ActRandom) {
            Energy = true;
            Liberu.jButtonDAct.setBackground(Color.RED);
      }

        if (bodyAction[Brain.Motive].active) {
                  senseProSelf[Brain.Motive] = pw;
            plsr += pw;
        }

        for (int i=0; i<bodyAction.length; i++) {
            if (bodyAction[i].active) {
                  senseProSelf[i] = (int)bodyAction[i].
                              actPower;
                bodyAction[i].setActive(false);
            }
            bodyAction[i].setPleasure(0);
        }
        return plsr;
    }

    public void endLiberu(boolean endL) {

      Neuron.end = endL;
    }
}
```

27 “Cerebellum”

```
/*
 * Cerebellum.java
 *
 * Created on 27 of February 2013, 17:45
 */

/**
 *
 * @author Paulo Roque Silva
 */
import java.util.Iterator;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;

public class Cerebellum implements Runnable {

      static final int ESCape = Brain.NEURONS + 100;
      // Mentor ou aleatório
      Cortex          past; // Todo o cortex
      Neuron              []neuroLeft;
      Neuron              []neuroRigth;
    CyclicBarrier   bar; // Barreira de passos
    Iterator<Signal> listen; // Ouve bus
    Signal              signInput; // Guarda sinal
    int                       nNeurons,
                        inv; // inverte a ação

    public Cerebellum(CyclicBarrier barr, Neuron []he,
            Neuron []hd, Cortex p, int nNTime) {

      bar       = barr;
      neuroLeft = he;
      neuroRigth= hd;
      past      = p;
      nNeurons  = nNTime;
      }

    public void writeBus(int time, int pwr) {
      int i, // indexe
            absTime;

        // Escreve no bus do Act
      absTime = Math.abs(time);
      System.out.print("time Cerebellum= "); // REMOVER
      System.out.println(time); // REMOVER
      if (Emotions.IdAct1 == ESCape &&
                  past.time[0][0].step >= Emotions.ActRandom)
            for (i=0;
                        i<neuroLeft[absTime].perception.size();
                        i++) {
                  Brain.BusInAct.add(new Signal(
                              inv * neuroLeft[absTime].
                              getPerception(i).emitter,
                              (int)neuroLeft[absTime].
                              getPerception(i).actPower,
                              pwr));
                  System.out.println(neuroLeft[absTime].
                              getPerception(i).actPower);
```

```
                    // REMOVER
            }
    }

    public void run() {
      int aPwr, temp;

      while (!Neuron.end && !bar.isBroken()) {
            try {
                bar.await();
            }
            catch (InterruptedException ex) {
                return;
            }
            catch (BrokenBarrierException ex) {
                return;
            }
            aPwr = 0;
            neuroLeft[neuroLeft[0].pointer].
            clearPerceptions();
            listen = Brain.BusInProSelf.iterator();
            while (listen.hasNext()) { // L□ todo o bus
                  signInput = listen.next();
                  neuroLeft[neuroLeft[0].pointer].
                  addPerception(signInput.emitter,
                  signInput.actPower, signInput.pleasure);
            }
            while (!Brain.BusInAct.isEmpty())
                  Brain.BusInAct.poll();

            if (Remember.RecentMost + 1 <
                        neuroLeft.length)
                  temp = Remember.RecentMost + 1;
            else
                  temp = 0;
            if (neuroLeft[neuroLeft[0].pointer].
                        perception.isEmpty())
                  aPwr = 0;
            else
                  aPwr = (int)neuroLeft[neuroLeft[0].
                      pointer].getPerception(0).
                        actPower;
            inv = 1;
                  if (Emotions.Good < 0) {
                  inv = -1;
                  writeBus(Emotions.Good, aPwr);
                  }
                  else
                  if (!Brain.BackHome && Emotions.IdAct1 ==
                              Cerebellum.ESCape) {
                        if (Act.Thumb) {
                              if (Brain.Saudade) {
                                    writeBus(Cortex.exec, aPwr);
                                    Cortex.exec =
                                                Math.abs(Cortex.exec);
                                    Mind.veloc = 0.75F; // Menos
r□pido
                                    Interface.Ink = 3;
                              }
                              else
                                    if (Brain.Ansiedade) {
```

```
                                        writeBus(Cortex.exec, aPwr);
                                     Mind.veloc = 1.5F;//Mais rápido
                                        Interface.Ink = 4;
                                  }
                                  else {
                                        writeBus(temp, aPwr);
                                        Cortex.exec = temp;
                                  }
                            }
                            else {
                                  if (Brain.Ansiedade) {
                                        writeBus(Cortex.exec, aPwr);
                                        Mind.veloc = 1.5F; // Mais rápido
                                        Interface.Ink = 4;
                                  }
                                  else
                                        if (Brain.Saudade) {
                                              writeBus(Cortex.exec, aPwr);
                                              Cortex.exec =

      Math.abs(Cortex.exec);
                                              Mind.veloc = 0.75F;
                                              Interface.Ink = 3;
                                        }
                                        else {
                                              writeBus(temp, aPwr);
                                              Cortex.exec = temp;
                                        }
                                  }
                      }
                      else {
                            writeBus(temp, aPwr);
                            Cortex.exec = temp;
                      }
            } // Fim ciclo While
      }
}
```

28 “Cortex”

```
/*
 * Cortex.java
 *
 * Created on 4 of August 2004, 22:21
 */
/**
 *
 * @author Paulo Roque Silva
 */
import java.io.*;

public class Cortex implements Serializable {
    public static final long serialVersionUID = 2024L;
    // Versão dos dados
    static String            Name; // Nome
    static String            EmailPhone; // Email - Telefone
    static String            Password; // Senha
    static final int        ThresholdTaste   = 90;
    static final int        ThresholdVision  = 98;
    static final int        ThresholdTact    = 80;
    static final int        ThresholdSound   = 90;
    static final int        ThresholdSmell   = 80;
    static final int        ThresholdProSelf = 80;
    static final int        ThresholdAct  = 90;
    static int              Senses; // Número de sentidos
    public static boolean    Warn; // Avisa se verdadeiro
    int                                 []threshold = {
                                  ThresholdTaste,
                                  ThresholdVision,
                                  ThresholdTact,
                                  ThresholdSound,
                                  ThresholdSmell,
                                  ThresholdProSelf
                                  };
    public static int        exec; // Executado
    public static int        lastExec;
                                  // Executado antes do exec
    public static int        beforeLastExec;
                                  // Executado antes do lastExec
    public static int        lastQuality;
    public static int        nLastSpeak;
    int                                 nNeuronsAct,
                                  // Número de neurónios no Act
                                  nNeuronsTime,
                                  // Número de neurónios do
                                  // Remember e Sleep
                                  actual; // Passo atual
    long                          timeLife,
    // Tempo em milisegundos de vida do Liberu executando
                                  bornTime,
                                  // Data de nascimento - (ms)
                                  timeLifeReal,
                                  // Tempo de vida do robô - (ms)
                                  timeBefore;
    Neuron                        [][]time;
                                  // Rede do passado - Remember
    Neuron                        []actNode;
```

```
                                        // Rede dos que agem - Act
    Neuron                              []hemisphereLeft;
    //Hemisfério esquerdo do Cerebellum
    Neuron                              []hemisphereRigth;
    //Hemisfério direito do Cerebellum

    /** Creates a new instance of Cortex */
    public Cortex(int t, int nAct, int nPast) {
        int i, n;     // indexes

        Name = new String();
        EmailPhone = new String();
        Password = new String();
        Senses    = t;
        nNeuronsAct = nAct;
        nNeuronsTime = nPast;
        actual = 0;
        timeLife = 0;
        bornTime = 0;
        timeBefore = 0;
        exec = 0;
        lastExec = 0;
        beforeLastExec = 0;
        nLastSpeak = 0;
        lastQuality = 0;
        time = new Neuron[nNeuronsTime][Senses];
          for (i=0; i<Senses; i++)
                 for (n=0; n<nNeuronsTime; n++)
                           time[n][i] = new Neuron(n,
                                        threshold[i]);
        actNode = new Neuron[nNeuronsAct];
        for (n=0; n<nNeuronsAct; n++)
            actNode[n] = new Neuron(n, ThresholdAct);
        hemisphereLeft = new Neuron[nNeuronsTime];
        hemisphereRigth = new Neuron[nNeuronsTime];
          for (n=0; n<nNeuronsTime; n++) {
            hemisphereLeft[n] = new Neuron(n,
                        ThresholdProSelf);
            hemisphereRigth[n] = new Neuron(n,
                        ThresholdProSelf);
          }
          Warn = false;
    }

    public int addDataActual() {

      actual += 1;
      return actual;
    }

    private long getTime() {

      return System.currentTimeMillis();
    }

    public long bornTime() {

        timeLife = getTime();
      return bornTime = timeLife;
    }
```

```
    public long addTime() { // Retorna tempo vida real (ms)

      return timeLifeReal += getTime() - timeLife;
    }

    public long startTime() { // Marca o ponto de execução

      timeBefore = timeLife;
      return timeLife = getTime();
    }

    public long getCicle() {

      return (timeLife - timeBefore);
    }
}
```

29 “Emotions”

```
/*
 * Emotions.java
 *
 * Created on 21 of January 2010, 16:42
 */

/**
 *
 * @author Paulo Roque Silva
 */

import java.util.Iterator;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;
import icommand.nxt.Sound;
import javax.swing.*;
import javax.swing.event.*;

public class Emotions implements Runnable {
      static final int                    ActRandom   = 30;
      // Número de passos aleatórios
      static final int                    NSpeakRandom = 15;
      // Número de speaks aleatórios
      static final int                    NSpeaks = 3;
      // Número de Vogais
      static int                                Normal = 0;
      private static final short[]  noteNew     = {5, 40};
      private static final short[]  noteDor     = {2349,

      40};
      private static final short[]  noteRir     = {500,

      40};
      private static final short[]  noteChorar  = {100,

      40};
    public static int                     QF, // Qualidade final
                                                QP,
                                                // Qualidade presente
                                                QR,
                                                // Qualidade lembrada
                                                Good,
                                          // Identificação recordado
                                                IdAct1, // Ação
                                                PwEm; // Amplitude ação
    public static boolean                 EmotionNew, //
                                                // Se novidade
                                                ActualPain,
                                                // Se sente Dor
                                                WARNING;
    public static Signal              []BodyAct;
                                          // Ações no corpo do robô
    private Cortex                              past; // Rede do córtex
    Iterator<Signal>                      listen; // Ouve bus
    CyclicBarrier                             stepBarrDecision;
                                          // Barreira do Emotions
    CyclicBarrier                             stepBarrDecision2;
```

```
                                        // Barreira 2 do Emotions
public static boolean                   emotional,
                                              rir,
                                              choro;
                                        // Se fica emocional
JTextField                              text; // Texto lido
DocumentListener                        listener; // Ouve bus
String                                        key; // Tecla primida

public Emotions(CyclicBarrier barrD,
        CyclicBarrier barrD2, Cortex me,
        Signal []bodyA, JTextField txt) {

  stepBarrDecision = barrD;
  stepBarrDecision2 = barrD2;
  past          = me;
  BodyAct       = bodyA;
      text            = txt;
  PwEm          = Mind.Power;
      emotional       = false;
      rir                   = false;
      choro           = false;
  ActualPain    = false;
  IdAct1              = Cerebellum.ESCape; // Sem ação
    Liberu.Already = true;
    WARNING = false;
    key = " ";
      listener      = new LearnListener();
    text.getDocument().addDocumentListener(listener);
}

public void idAct1Action(int idAction) {

        IdAct1 = idAction;
}

private static void express(short[] note) {
  int i;

  if (Liberu.Robot == 2)
        for (i = 0; i < note.length; i += 2) {
              final short w = note[i + 1];
              final int n = note[i];
              Sound.playTone(n, w * 10);
        }
  if (Liberu.Robot == 1)
        try {
            Liberu.output.write((byte)110);
            Liberu.output.write('\n');
        }
        catch (Exception e) {
            System.err.println(e.toString());
        }
}

  public void emotionSpeak() {
        short []happySound = {1963, 80};
        int         idSound;

        Cortex.nLastSpeak += 1;
        if (Liberu.Robot == 1 || Liberu.Robot == 2) {
```

```
            happySound[0] = (short)QF;
            if (happySound[0] > 3000)
                  happySound[0] = 3000;
            express(happySound);                //NXT
      }
      if (Liberu.Robot == 0 &&
                  Cortex.nLastSpeak < NSpeakRandom) {
            if (IdAct1 == Cerebellum.ESCape) {
                  idSound = Brain.Vogal1 +
                              (int)Math.rint(Math.random() *
                              (Emotions.NSpeaks - 1));
                  BodyAct[idSound].setActive(true);
                  BodyAct[idSound].setActPower(PwEm);
                  if (idSound == Brain.Vogal1) {
                        Mind.clip.setMicrosecondPosition(0);
                        Mind.clip.start();
                  }
                  if (idSound == Brain.Vogal2) {
                        Mind.clip2.setMicrosecondPosition(0);
                        Mind.clip2.start();
                  }
                  if (idSound == Brain.Vogal3) {
                        Mind.clip3.setMicrosecondPosition(0);
                        Mind.clip3.start();
                  }
            }
      }
      System.out.println("SPEEK...  :-)"); //REMOVER
      }

      public void emotionNew(boolean empty) {

            // Emoção de novidade - Genetica
            if (empty) {
                  emotional = true;
                  EmotionNew = true;
            if (Liberu.Robot == 2)
                  express(noteNew); // comando NXT
            if (Liberu.Robot == 1)
                        try {
                        Liberu.output.write((byte)106);
                        Liberu.output.write('\n');
                        }
                  catch (Exception e) {
                        System.err.println(e.toString());
                  }
                  System.out.println(
                              "Emotion NEW...    :-0"); //REMOVER
                  BodyAct[Brain.New].setActive(true);
                  WARNING = true;
            }
      }

      private void emotionNormal() {

            emotional = false;
      ActualPain = false;
            EmotionNew = false;
            rir                = false;
            choro        = false;
            System.out.println(
```

```
                    "Emotion Normal... :-|"); //REMOVER
if (Liberu.Robot == 1)
            try {
            Liberu.output.write((byte)107);
            Liberu.output.write('\n');
            Liberu.output.write((byte)108);
            Liberu.output.write('\n');
            Liberu.output.write((byte)109);
            Liberu.output.write('\n');
            }
      catch (Exception e) {
            System.err.println(e.toString());
      }
}

public void emotionHappy() {
      short []happySound = {1963, 40};

      emotional = true;
BodyAct[Brain.Happy].setActive(true);
// Expressão genética de felicidade
if (Liberu.Robot == 2) {
      happySound[0] = (short)(QF * 10 + 1000);
      if (happySound[0] > 3000)
            happySound[0] = 3000;
      express(happySound); // NXT
}
if (Liberu.Robot == 1)
            try {
                  Liberu.output.write((byte)104);
              Liberu.output.write('\n');
            }
            catch (Exception e) {
              System.err.println(e.toString());
            }
System.out.println(
            "Emotion HAPPY...  :-)"); //REMOVER
}

public void emotionSad() {
      short []sadSound = {0403, 40};

      emotional = true;
BodyAct[Brain.Sad].setActive(true);
// Expressão genética de tristeza
if (Liberu.Robot == 2) {
      sadSound[0] = (short)(QF * 10 + 1000);
      if (sadSound[0] < 20)
            sadSound[0] = 20;
      express(sadSound);
}
if (Liberu.Robot == 1)
            try {
            Liberu.output.write((byte)105);
            Liberu.output.write('\n');
            }
      catch (Exception e) {
            System.err.println(e.toString());
      }
      System.out.println(
                  "Emotion Sad...    :-("); //REMOVER
```

```
}

public void emotionPAIN() {

ActualPain   = true;
emotional = true;
if (BodyAct[Brain.Sad].active)
      if (Liberu.Robot == 1 || Liberu.Robot == 2)
            express(noteDor);
BodyAct[Brain.PAIN].setActive(true);
// Expressão genética de Dor
      System.out.println(
                  "Emotion PAIN...   ;-("); //REMOVER
}

public void emotionRIR() {

emotional = true;
rir           = true;
if (Liberu.Robot == 1 || Liberu.Robot == 2)
      express(noteRir);
if (Liberu.Robot == 0) {
      Mind.clip4.setMicrosecondPosition(0);
      Mind.clip4.start();
}
System.out.println(
                  "Emotion RIR...   ;-)"); //REMOVER
}

public void emotionCHORAR() {

emotional = true;
choro   = true;
if (Liberu.Robot == 1 || Liberu.Robot == 2)
      express(noteChorar);
if (Liberu.Robot == 0) {
      Mind.clip5.setMicrosecondPosition(0);
      Mind.clip5.start();
}
      System.out.println(
                  "Emotion CHORAR...   :-#"); //REMOVER
}

public void emotionAfraid() {

emotional = true;
BodyAct[Brain.Afraid].setActive(true);
      System.out.println(
                  "Emotion Afraid...   ,-("); //REMOVER
}

public void runEmotions() {
      int i, j; // indexes

try {
      stepBarrDecision.await(); // Sincroniza
  }
  catch(InterruptedException e) {
          e.printStackTrace();
      }
  catch (BrokenBarrierException ex) {
```

```
            return;
        }
    // ################################ DECIDE #######
    Interface.Ink = 0;
      for (i=0; i<Brain.SENSES; i++) {
          if (past.time[0][0].step > ActRandom) {
                emotionNew(Remember.NewSense[i]);
                if (i == Brain.Taste &&
                            Remember.NewSense[Brain.Taste] &&
                            !ActualPain)
                      IdAct1 = past.time[Good][i].
                      attentionOfFire(i);
                if (i == Brain.Vision &&
                            Remember.NewSense[Brain.Vision] &&
                            !ActualPain)
                      IdAct1 = past.time[Good][i].
                      attentionOfFire(i);
          }
          }

    QP = past.
                time[past.time[0][0].
                     pointer][Brain.ProSelf].finalQuality;
    QR = past.
                time[Math.abs(Good)][Brain.ProSelf].
                finalQuality;
          System.out.println(QP);     // RETIRAR
          System.out.println(QR);     // RETIRAR
    QF = QP + QR;
    Interface.QF = QF;
    if (Normal >= 0)
          if (QF < 0 && past.time[0][0].step > ActRandom)
                emotionSad();
          else {
                if (QF > 2 * Normal &&
                            past.time[0][0].step > ActRandom)
                emotionHappy();
          }
    else
          if (QF < 2 * Normal &&
                      past.time[0][0].step > ActRandom)
                emotionSad();
          else {
                if (QF > 0 &&
                            past.time[0][0].step > ActRandom)
                emotionHappy();
          }
    if (QP < Cortex.lastQuality &&
                past.time[0][0].step > ActRandom) {
          emotionAfraid();
          if (BodyAct[Brain.Happy].active)
                emotionRIR();
          else
                if (BodyAct[Brain.Sad].active)
                      emotionCHORAR();
                else
                      emotionSpeak();
    }
          Cortex.lastQuality = QP;
      // ######  Recente decis►o ######
    if (ActualPain && past.time[0][0].step > ActRandom)
```

```
            //
            Good = -Good; // Ato oposto de reflexo de Dor
            // ##############################
      for (i=0; i<past.nNeuronsTime; i++)
            for (j=0; j<past.time[i][0].learn.size();
                        j++) {
                  if (Math.abs(past.time[i][0].getLearn(j).
                              emitter) == past.time[i][0].
                              pointer)
                        past.time[i][0].removeLearn(j);
            }
      // ################################ ACT ##########
            System.out.print("Key****= "); // RETIRAR
            System.out.println(key); // RETIRAR

      while (!Brain.BusInAct.isEmpty())
            Brain.BusInAct.poll();

        // ################## ALEATÓRIO - MENTOR ##########
      try {
            stepBarrDecision2.await(); // sincroniza
        }
        catch(InterruptedException e) {
                e.printStackTrace();
            }
        catch (BrokenBarrierException ex) {
            return;
        }
        if (IdAct1 != Cerebellum.ESCape ||
                  past.time[0][0].step < ActRandom) {
            if (IdAct1 != Cerebellum.ESCape) {
                  BodyAct[IdAct1].setActive(true);
                  BodyAct[IdAct1].setActPower(PwEm);
                  System.out.print("MentorA= "); //REMOVER
                  System.out.println(IdAct1); //REMOVER
                  Interface.Ink = 2; // Verde
            }
            else {
                  IdAct1 = (int)Math.rint(Math.random() *
                              (Brain.ACTS-1));
                if (past.time[0][0].step == 0) {
                  IdAct1 = Brain.WALK; // Primeira ação
                }
                  BodyAct[IdAct1].setActive(true);
                  BodyAct[IdAct1].setActPower(PwEm);
                  System.out.print("Random1= "); //REMOVER
                  System.out.println(IdAct1); //REMOVER
                  Interface.Ink = 1; // Vermelho
            }
            past.actNode[IdAct1].standby = true;
            IdAct1 = Cerebellum.ESCape; // Sem ação
        }

            // ##################### R E L Ó G I O ############
        System.out.print("step= "); // REMOVER
            System.out.println(past.time[0][0].pointer);
                                                      // REMOVER
      if (!ActualPain) { // Incremento do passo sem Dor
            for (j=0; j<Cortex.Senses; j++)
                  for (i=0; i<past.nNeuronsTime; i++) {
                        past.time[i][j].addStep();
```

```
                        // Relógio de vida do robô
                        past.time[i][j].addPointer();
                        // Relógio da janela temporal do robô
                  if ( past.time[i][j].pointer >
                  past.nNeuronsTime - 1)
                        past.time[i][j].setPointer(0);
                  }
            for (i=0; i<past.nNeuronsTime; i++) {
                  past.hemisphereLeft[i].addStep();
                  // Relógio de vida do robô
                  past.hemisphereLeft[i].addPointer();
                  // Relógio da janela temporal do robô
                  if ( past.hemisphereLeft[i].pointer >
                  past.nNeuronsTime - 1)
                        past.hemisphereLeft[i].setPointer(0);
                  past.hemisphereRigth[i].addStep();
                  // Relógio de vida do robô
                  past.hemisphereRigth[i].addPointer();
                  // Relógio da janela temporal do robô
                  if ( past.hemisphereRigth[i].pointer >
                  past.nNeuronsTime - 1)
                        past.hemisphereRigth[i].setPointer(0);
            }
      }
      // ############ Fim relógio #######################

        if (emotional) // Regresso ao estado normal
            emotionNormal();
      }

    public void setLearn() {
      int temp;

            key = (String)text.getText().trim();
            Liberu.InterFace.repaint();
            System.out.print("Key= "); // RETIRAR
            System.out.println(key); // RETIRAR
            temp = Integer.parseInt(key);
            if (temp > 0 && temp < 65)
                  if (key != "")
                        Mind.local = temp;
            Liberu.InterFace.repaint();
            System.out.print("Local= "); // RETIRAR
            System.out.println(Mind.local); // RETIRAR
    }

    private class LearnListener implements
    DocumentListener {

      public void insertUpdate(DocumentEvent e)
      { setLearn(); }
      public void removeUpdate(DocumentEvent e)
      { setLearn(); }
      public void changedUpdate(DocumentEvent e) {}
    }

    public void run() {

            while(!stepBarrDecision.isBroken() &&
                        !stepBarrDecision2.isBroken() &&
```

```
                        !Neuron.end) {
            runEmotions();
      }
        System.out.println("End of Emotions Thread...");
    }
}
```

30 “Interface”

```
/*
 * Interface.java
 *
 * Created on 8 of February 2006, 2:04
 */
/**
 *
 * @author  Paulo Roque Silva
 */
import javax.swing.*;
import java.awt.*;
import java.util.Date;

public class Interface extends JPanel {

    public static final long    serialVersionUID = 2023L;
    // Vers►o do Interface
    static final int    MaxDisplay = 300; // Janela (passos)
      public static int   Ink;
      // ink cor 0=preto 1=vermelho 2=verde 3=azul 4=laranja
      public static int []Taste;
      public static int []Vision;
      public static int []Sound;
      public static int []Smell;
      public static int   Lovely;
      public static int   QF;
    public static boolean Warned; // Se em aviso
    public static boolean WarnSense[]; // Se avisado
      Brain                   mind; // Daos do cérebro
    boolean             execute; // se em execução
    Color               colorBefore;
    int                 i, n; // Índexes
    long                      ciclo;
    Image                     imge, imgeW;
    Image               imgWorld, imgE, imgElegans2,
      imgElegans3, imgElegans4, imgElegans2n, imgElegans3n,
      imgElegans4n, imgElegans2w, imgElegans3w, imgElegans4w,
      imgElegans2e, imgElegans3e, imgElegans4e;
    int                 []value = new int[100];
    // Memória auxiliar

    /** Creates new form Interface */
    public Interface() {
      int i;

        imge = Toolkit.getDefaultToolkit().
                  getImage("formicaLogo120pxl.jpg");
        imgeW = Toolkit.getDefaultToolkit().
                  getImage("World8x8.jpg");
        execute = false;
        ciclo = 0;
      Ink = 0;
      Taste = new int[Brain.NanalogTaste];
        for (i=0; i<Taste.length; i++)
            Taste[i] = 0;
      Vision = new int[Brain.NanalogVision];
        for (i=0; i<Vision.length; i++)
```

```
            Vision[i] = 0;
      Sound = new int[Brain.NanalogSound];
        for (i=0; i<Sound.length; i++)
            Sound[i] = 0;
      Smell = new int[Brain.NanalogSmell];
        for (i=0; i<Smell.length; i++)
            Smell[i] = 0;
        Lovely = 0;
        QF = 0;
        Warned = false;
        WarnSense = new boolean[Brain.SENSES];
        for (i=0; i < Brain.SENSES; i++)
            WarnSense[i] = false;

        Toolkit kit = Toolkit.getDefaultToolkit();
        imgWorld    = kit.getImage("World.jpg"); // Mundo
        imgElegans2 = kit.getImage("c_elegans2.jpg");
        imgElegans3 = kit.getImage("c_elegans3.jpg");
        imgElegans4 = kit.getImage("c_elegans4.jpg");
        imgElegans2n = kit.getImage("c_elegans2n.jpg");
        imgElegans3n = kit.getImage("c_elegans3n.jpg");
        imgElegans4n = kit.getImage("c_elegans4n.jpg");
        imgElegans2w = kit.getImage("c_elegans2w.jpg");
        imgElegans3w = kit.getImage("c_elegans3w.jpg");
        imgElegans4w = kit.getImage("c_elegans4w.jpg");
        imgElegans2e = kit.getImage("c_elegans2e.jpg");
        imgElegans3e = kit.getImage("c_elegans3e.jpg");
        imgElegans4e = kit.getImage("c_elegans4e.jpg");
        for (i=0; i<value.length; i++)
            value[i] = 0;
    }

    public void displayRobot(Brain m, boolean ex,
            boolean w) {

      mind = m;
        execute = ex;
      Warned = w;
    }

    public void saveValue(int n, int val) {

        value[n] = val;
    }

    public void paintComponent(Graphics g) {
      int n, m, rct, bom, aux,
            inicioJanela,
            fimJanela,
            pointerJanela,
            temp; // Define janela
            float t=0;
            Font fDefault, f;

        super.paintComponent(g);
        g.drawImage(imge, 450, 210, null, null);
      if (Liberu.Robot == 0) {
            g.drawString("World", 785, 55);
            g.drawImage(imgeW, 720, 65, null, null);
      }
        colorBefore = g.getColor();
```

```
//##################################################

if (Liberu.Robot == 0) {
    imgE = imgElegans2;
    if (value[4] == World.NORTH)
          imgE = imgElegans2n;
    if (value[4] == World.WEST)
          imgE = imgElegans2w;
    if (value[4] == World.EAST)
          imgE = imgElegans2e;
    if (value[0] == 61) {
          imgE = imgElegans3;
          if (value[4] == World.NORTH)
                imgE = imgElegans3n;
          if (value[4] == World.WEST)
                imgE = imgElegans3w;
          if (value[4] == World.EAST)
                imgE = imgElegans3e;
    }
    if (value[0] == 7) {
          imgE = imgElegans4;
          if (value[4] == World.NORTH)
                imgE = imgElegans4n;
          if (value[4] == World.WEST)
                imgE = imgElegans4w;
          if (value[4] == World.EAST)
                imgE = imgElegans4e;
    }
    if (value[0] != 0) g.drawImage(imgE,
                735 + 60 * ((value[0]-1) % 8),
                      80 + 60 * ((value[0]-1) / 8),
                       null);
}
if (Liberu.Robot == 0) {
    switch (value[2]) {
          // value 2          Elegans
    case World.EARTH: g.drawString(
                "Earth", 150, 230);
        break;
    case World.WATER: g.drawString(
                "Water", 150, 230);
        break;
    case World.FOOD: g.drawString(
                "Home", 150, 230);
        break;
    default: g.drawString("Error", 150, 230);
        break;
    }
    if (Liberu.Robot == 0) {
        g.drawString("Orientation= ", 40, 405);
        switch (value[4]) {
                // value 4
        case World.NORTH: g.drawString(
                "North", 150, 405);
            break;
        case World.EAST: g.drawString(
                "East", 150, 405);
            break;
        case World.SOUTH: g.drawString(
                "South", 150, 405);
```

```
                    break;
                case World.WEST: g.drawString(
                        "West", 150, 405);
                    break;
                default: g.drawString("Error", 150, 405);
                    break;
                }
            }
        }
    //#####################################################

        if (Emotions.WARNING && Warned) {
            g.setColor(Color.RED);
            for (n=0; n < Brain.SENSES; n++) {
                  if (Remember.NewSense[n])
                        if (!WarnSense[n])
                        WarnSense[n] =
                        Remember.NewSense[n];
            }
        }
        for (n=0; n < Brain.SENSES; n++)
            if (WarnSense[n])
                  switch (n) {
                  case Brain.Taste: {
                    g.drawString("WARNING - New event!" +
                  " TASTE", 600, 585 + 0 * 30);
                    break;
                  }
                  case Brain.Vision: {
                    g.drawString("WARNING - New event!" +
                  " VISION", 600, 585 + 1 * 30);
                    break;
                  }
                  case Brain.Tact: {
                    g.drawString("WARNING - New event!" +
                  " TACT", 600, 585 + 2 * 30);
                    break;
                  }
                  case Brain.Sound: {
                    g.drawString("WARNING - New event!" +
                  " SOUND", 600, 585 + 3 * 30);
                    break;
                  }
                  case Brain.Smell: {
                    g.drawString("WARNING - New event!" +
                  " SMELL", 600, 585 + 4 * 30);
                    break;
                  }
                  case Brain.ProSelf: {
                    g.drawString("WARNING - New event!" +
                  " SELF", 600, 585 + 5 * 30);
                    break;
                  }
                  default:
                    g.drawString("ERROR case...", 600,
                              585 + 6 * 30);
                  }
      g.setColor(colorBefore);

        g.drawString("Step= ", 40, 20);
        g.drawString(Integer.toString(mind.past.actual),
```

```
                150, 20);
        if (mind.past.actual <= Emotions.ActRandom) {
            g.setColor(Color.RED);
            g.drawString("Womb", 200, 20);
            g.setColor(colorBefore);
        }

        g.drawString("Time(s)= ", 40, 55);
      t = (float)mind.past.timeLifeReal/1000/60/60;
      // t em horas
        g.drawString("hr", 150, 55);
        g.drawString(Long.toString((long)Math.floor(t)),
                130, 55);
      t = (float)(t - (int)Math.floor(t));
      // t = resto em horas
        g.drawString("min", 230, 55);
        t = t * 60; // t em minutos
        g.drawString(Long.toString((long)Math.floor(t)),
                210, 55);
        g.drawString("s", 310, 55);
        t = (float)(t - (int)Math.floor(t));
        // t = resto em minutos
        g.drawString(Long.toString((long)Math.floor(t*60)),
                290, 55); // t em segundos

        g.drawString("Percentage (Lovely/Neurons)= ",
                40, 90);
        if (Liberu.Already)
            g.drawString(Integer.toString(
                        (int)(Lovely/(Brain.SENSES*
                                    (float)mind.past.nNeuronsTime)*
                                    100)), 230, 90);

        g.drawString("Born: ", 40, 125);
        if (!execute)
            g.drawString((new Date()).toString() ,
                        140, 125);
        else
            g.drawString((new Date(
                        mind.past.bornTime).toString()) ,
                        140, 125);

        g.drawString("Cycle(ms): ", 40, 160);
        if (execute)
            ciclo = mind.past.getCicle();
        g.drawString(Long.toString(ciclo) , 200, 160);
        //  ##############################################
        if (execute) {
            g.drawString("Remember" , 250, 195);
            g.drawString("Real" , 150, 195);
        }
        // TASTE
        if (Liberu.Already)
            if (Remember.NewSense[0])
                g.setColor(Color.BLUE);
        g.drawString("Taste  = ", 40, 230);
      if (Liberu.Robot == 1 || Liberu.Robot == 2)
            g.drawString(Integer.toString(Taste[0]),
                        150, 230);
      aux = 0;
      if (Liberu.Robot == 0)
```

```
            aux = Brain.NanalogTaste;
        if (Liberu.Robot == 1)
            aux = Brain.NanalogTasteR;
        if (Liberu.Robot == 2)
            aux = Brain.NanalogTasteN;
        if (execute)
            for (n=0; n < aux; n++)
                if ((int)mind.past.time[Math.abs(
                            Emotions.Good)][Brain.Taste].
                            getPerception(n).recPower >
                Brain.MinSense) {
                  if (Liberu.Robot == 0)
                      switch (n+1) {
                      // value 2          Elegans
                      case World.EARTH: g.drawString(
                                  "Earth", 250, 230);
                          break;
                      case World.WATER: g.drawString(
                                  "Water", 250, 230);
                          break;
                      case World.FOOD: g.drawString(
                                  "Home", 250, 230);
                          break;
                      default: g.drawString("Error",
                                  250, 230);
                          break;
                      }
                  if (Liberu.Robot == 1 ||
                            Liberu.Robot == 2)
                      g.drawString(Integer.toString(
                            (int)mind.past.time[Math.abs(
                            Emotions.Good)][Brain.Taste].
                            getPerception(n).recPower),
                                  250, 230);
                }
        g.setColor(colorBefore);
        // VISION
        if (Liberu.Already)
            if (Remember.NewSense[1])
                g.setColor(Color.BLUE);
        g.drawString("Vision  = ", 40, 265);
        if (Liberu.Robot == 0)
            Vision[1] = value[5];
        if (Liberu.Robot == 0)
            aux = Brain.NanalogVision;
        if (Liberu.Robot == 1)
            aux = Brain.NanalogVisionR;
        if (Liberu.Robot == 2)
            aux = Brain.NanalogVisionN;
        if (Vision[1] == World.WALL)
            // value 5          Elegans
            g.drawString("Wall", 150, 265);
                                        // Elegans
        else
                                                    // Elegans
            for (n=0; n < aux; n++) {
                g.drawString(Integer.toString(Vision[n]),
                        150+n*35, 265);
                if (execute)
                    g.drawString(Integer.toString(
                        (int)mind.past.time[Math.abs(
```

```
                                Emotions.Good)][Brain.Vision].
                        getPerception(n).recPower),
                                250+n*35, 265);
            }
        g.setColor(colorBefore);
        // TACT
        if (Liberu.Already)
            if (Remember.NewSense[2])
                g.setColor(Color.BLUE);
        g.drawString("Tact      = ", 40, 300);
        if (Liberu.Robot == 1 || Liberu.Robot == 2)
            value[1] = Mind.Tact[0];
        if (value[1] == Brain.TACT)
            g.drawString("Obstacle", 150, 300);
        else
            g.drawString("        ", 150, 300);
        if (Liberu.Already && execute)
            if (mind.past.time[Math.abs(Emotions.Good)]
                        [Brain.Tact].getPerception(0).
                        recPower > Mind.Power/2)
                  g.drawString("Obstacle", 250, 300);
            else
                g.drawString("        ", 250, 300);
        g.setColor(colorBefore);
        // SOUND
        if (Liberu.Already)
            if (Remember.NewSense[3])
                g.setColor(Color.BLUE);
        g.drawString("Sound  = ", 40, 335);
        if (Liberu.Robot == 0)
            aux = Brain.NanalogSound;
        if (Liberu.Robot == 1)
            aux = Brain.NanalogSoundR;
        if (Liberu.Robot == 2)
            aux = Brain.NanalogSoundN;
        for (n=0; n < aux; n++) {
            if (Liberu.Robot == 1 || Liberu.Robot == 2)
                  value[36+n] = Sound[n];
            g.drawString(Integer.toString(value[36+n]),
                        150+n*30, 335);
        }
        temp = 0;
        if (execute) {
            temp = 0;
            for (n=0; n < aux; n++)
                  temp += (int)mind.past.time[Math.abs(
                                Emotions.Good)][Brain.Sound].
                  getPerception(n).recPower;
            temp = (int)(temp / aux);
            g.drawString(Integer.toString(temp), 350, 335);
        }
        g.setColor(colorBefore);
        // OLFACT
        if (Liberu.Already)
            if (Remember.NewSense[4])
                g.setColor(Color.BLUE);
        g.drawString("Smell  = ", 40, 370);
        if (Liberu.Robot == 0)
            aux = Brain.NanalogSmell;
        if (Liberu.Robot == 1)
            aux = Brain.NanalogSmellR;
```

```
if (Liberu.Robot == 2)
    aux = Brain.NanalogSmellN;
if (Liberu.Robot == 1 || Liberu.Robot == 2)
    for (n=0; n < aux; n++) {
          value[n] = Smell[n];
        g.drawString(Integer.toString(value[n]),
                150+n*35, 370);
        if (execute)
            g.drawString(Integer.toString(
                (int)mind.past.time[Math.abs(
                      Emotions.Good)][Brain.Smell].
                getPerception(n).recPower),
                      250+n*35, 370);
    }
if (Liberu.Robot == 0)
    g.drawString(Integer.toString(value[0]),
                150, 370);
if (execute && Liberu.Robot == 0)
    for (n=0; n < aux; n++) {
        if ((int)mind.past.time[Math.abs(
                Emotions.Good)][Brain.Smell].
                getPerception(n).recPower >
        Brain.MinSense)
            g.drawString(Integer.toString(n+1),
                      250, 370);
    }
g.setColor(colorBefore);

if (Liberu.Already && execute)
    for (i=0; i<mind.past.time[Math.abs(
                Emotions.Good)][Brain.ProSelf].
                perception.size(); i++)
          if (mind.past.time[Math.abs(Emotions.Good)]
                      [Brain.ProSelf].getPerception(i).
                      recPower > Brain.MinSense)
                switch (i) {
          case Brain.DRINK:
                g.drawString("Drink", 250, 405);
                break;
          case Brain.EAT:
                g.drawString("Eat", 250, 405);
                break;
          case Brain.WALK:
                g.drawString("Walk", 250, 405);
                break;
          case Brain.TURN_LEFT:
                g.drawString("Left", 250, 405);
                break;
          case Brain.TURN_RIGHT:
                g.drawString("Right", 250, 405);
                break;
          default:
          }
g.drawString("Energy = ", 450, 90);
if (Brain.Energy) {
    g.setColor(Color.BLUE);
    g.drawString("Need", 560, 90);
    g.setColor(colorBefore);
}
else
    g.drawString("No need", 560, 90);
```

```
g.drawString("Happy = ", 450, 125);
if (QF >= 0) {
    g.setColor(Color.GREEN);
    g.drawString(Integer.toString(QF), 560, 125);
    g.setColor(colorBefore);
}
else {
    g.setColor(Color.RED);
    g.drawString(Integer.toString(QF), 560, 125);
    g.setColor(colorBefore);
}

g.drawString("Saudade= ", 450, 160);
if (Brain.Saudade) {
    g.setColor(Color.BLUE);
    g.drawString("Yes", 560, 160);
    g.setColor(colorBefore);
}
else
    g.drawString("No", 560, 160);

g.drawString("Anxiety= ", 450, 195);
if (Brain.Ansiedade) {
    g.setColor(Color.ORANGE);
    g.drawString("Yes", 560, 195);
    g.setColor(colorBefore);
}
else
    g.drawString("No", 560, 195);

g.drawString("Homeostase = ", 450, 55);
Emotions.Normal = 0;
for (n=0; n < mind.past.nNeuronsTime; n++)
    Emotions.Normal += mind.past.time[n]
                [Brain.ProSelf].finalQuality;
if (mind.past.time[0][0].step <
          mind.past.nNeuronsTime &&
          mind.past.time[0][0].step != 0)
    Emotions.Normal = (int)(Emotions.Normal /
                mind.past.time[0][0].step);
else
    Emotions.Normal = (int)(Emotions.Normal /
                mind.past.nNeuronsTime);
if ( Emotions.Normal>= 0) {
    g.setColor(Color.GREEN);
    g.drawString(Integer.toString(Emotions.Normal),
                560, 55);
    g.setColor(colorBefore);
}
else {
    g.setColor(Color.RED);
    g.drawString(Integer.toString(Emotions.Normal),
                560, 55);
    g.setColor(colorBefore);
}

g.drawString("Action  = ", 40, 445);
if (Ink == 1)
    g.setColor(Color.RED);
if (Ink == 3)
```

```
            g.setColor(Color.BLUE);
        if (Ink == 4)
            g.setColor(Color.ORANGE);
        if (Ink == 2 || Emotions.IdAct1 != Cerebellum.ESCape)
            g.setColor(Color.GREEN);
        if (Mind.Action[Brain.DRINK].active)
            g.drawString("Drink", 150, 445);
        if (Mind.Action[Brain.EAT].active)
            g.drawString("Eat", 240, 445);
        if (Mind.Action[Brain.WALK].active)
            g.drawString("Walk", 330, 445);
        if (Mind.Action[Brain.TURN_LEFT].active)
            g.drawString("Left", 420, 445);
        if (Mind.Action[Brain.TURN_RIGHT].active)
            g.drawString("Right", 510, 445);

        g.setColor(colorBefore);
        // ##############################################
        g.drawString("Emotion= ", 40, 480);
        if (Emotions.ActualPain) {
            g.setColor(Color.RED);
            g.drawString("Pain", 150, 480);
            g.setColor(colorBefore);
        }
        if (Emotions.EmotionNew) {
            g.setColor(Color.GREEN);
            g.drawString("New", 230, 480);
            g.setColor(colorBefore);
        }

        g.drawString("Lovely =", 310, 480); // Beautiful
        g.drawString(Integer.toString(Lovely), 400, 480);

        if (Liberu.Already)
            if (Emotions.BodyAct[Brain.Happy].active) {
                  g.setColor(Color.GREEN);
                g.drawString("Happy", 480, 480);
                  g.setColor(colorBefore);
            }

        if (Liberu.Already)
            if (Emotions.BodyAct[Brain.Sad].active) {
                  g.setColor(Color.RED);
                g.drawString("Sad", 560, 480);
                  g.setColor(colorBefore);
            }

        if (Liberu.Already)
            if (Emotions.BodyAct[Brain.Afraid].active) {
                  g.setColor(Color.ORANGE);
                g.drawString("Afraid", 640, 480);
                  g.setColor(colorBefore);
            }

        if (Liberu.Already)
            if (Emotions.rir) {
                  g.setColor(Color.GREEN);
                g.drawString("Laugh", 480, 505);
                  g.setColor(colorBefore);
            }
```

```
        if (Liberu.Already)
            if (Emotions.choro) {
                  g.setColor(Color.RED);
                g.drawString("Cry", 560, 505);
                  g.setColor(colorBefore);
            }

        if (Liberu.Already)
            if (Act.speak) {
                  g.setColor(Color.ORANGE);
                g.drawString("Speak", 640, 505);
                if (Cortex.nLastSpeak <=
                        Emotions.NSpeakRandom)
                  g.setColor(Color.RED);
                else
                  g.setColor(Color.RED);
                g.drawString("(" + Integer.toString(
                        Emotions.NSpeakRandom) + ")",
                        640, 535);
                  g.setColor(colorBefore);
            }

        if (execute) {
            g.drawString("Ta", 320 - 250, 515 + 0);
            g.drawString("Vi", 320 - 250, 515 + 35);
            g.drawString("Tc", 320 - 250, 515 + 70);
            g.drawString("So", 320 - 250, 515 + 105);
            g.drawString("Sm", 320 - 250, 515 + 140);
            g.drawString("PS", 320 - 250, 515 + 175);
            g.drawLine(350 - 250, 515 + 0, 350 - 250, 515 +
                        0 + 35 * Brain.SENSES); // eixo Y
            if (mind.past.nNeuronsTime > MaxDisplay)
                  m = MaxDisplay;
            else
                  m = mind.past.nNeuronsTime;
            g.drawLine(350 - 250, 515 + 0 + 35 *
                        Brain.SENSES, 350 + m - 250, 515 +
                        0 + 35 * Brain.SENSES); // eixo X
            n = 0;
            while (n < MaxDisplay) {
                if (n % 10 == 0) g.drawLine(350 + n+1 -
                        250, 515 + 0 + 35 * Brain.SENSES,
                              350 + n+1 - 250, 515 + 0 + 35 *
                              Brain.SENSES + 3);
                // escala 10 em 10 no eixo X
                n += 1;
            }

            g.drawString("Fired neurons, by Past...", 350 -
                        250, 515 + 0 + 35 * Brain.SENSES + 30);

                  if (mind.past.time[0][0].pointer >
                  MaxDisplay) {
                        fimJanela = mind.past.time[0][0].pointer;
                        pointerJanela = MaxDisplay;
                        inicioJanela = mind.past.time[0][0].
                                    pointer - MaxDisplay;
                  }
                  else {
                        fimJanela = MaxDisplay;
                        pointerJanela = mind.past.time[0][0].
```

```
                                pointer;
                        inicioJanela = 0;
                }
            // pointer
                g.drawLine(350 + pointerJanela+1 - 250, 515 +
                            29, 350 + pointerJanela+1 - 250, 515 +
                            0 + 35 * Brain.SENSES - 1);

                fDefault = g.getFont();
                f = new Font(g.getFont().getFontName(),
                            g.getFont().getStyle(),
                            16);
                g.setFont(f);
                g.drawString(Integer.toString(inicioJanela),
                            350 - 250, 515 + 0 + 35 *
                            Brain.SENSES + 15);
                g.drawString(Integer.toString(fimJanela),
                            650 - 250, 515 + 0 + 35 *
                            Brain.SENSES + 15);

                g.setFont(fDefault);
                // Neuron fires
            m = 0;
            while (m <= fimJanela && inicioJanela + m <
                        mind.past.nNeuronsTime) {
                for (i=0; i < Brain.SENSES; i++)
                        if (mind.past.time[inicioJanela+m][i].
                                    fire)
                            g.drawLine(350 + m+1 - 250, 515 +
                                        0 + 35 * i - 1, 350 + m+1 -
                                        250, 515 + 0 + 35 * i - 1);
                m += 1;
            }
                // gráfico da rede "learn"
            m = 0;
            while (m <= fimJanela && inicioJanela + m <
                        mind.past.nNeuronsTime) {
                for (i=0; i < mind.past.time
                            [inicioJanela+m][0].learn.size();
                            i++) {
                        bom = mind.past.time
                                    [inicioJanela+m][0].
                                    getLearn(i).emitter;
                  if (bom >= inicioJanela && bom <=
                            fimJanela)
                            g.drawLine(350 + m+1 - 250, 515 +
                                        210 - bom - inicioJanela,
                                        350 + m+1 - 250, 515 +
                                        210 - bom - inicioJanela);
                }
                m += 1;
            }
            // recent
            g.setColor(Color.RED);
            rct = Math.abs(Cortex.exec);
            if (rct >= inicioJanela && rct <= fimJanela) {
                g.drawString("+", 350 + rct -
                        inicioJanela - 3 - 250, 515 + 0 +
                        35 * Brain.SENSES + 4);
                g.setFont(f);
                  g.drawString(Integer.toString(rct), 350 +
```

```
                                rct - inicioJanela - 250, 515 +
                                0 + 35 * Brain.SENSES - 10);
                        g.setFont(fDefault);
                    }
                    else {
                          if (rct < inicioJanela) {
                            g.setFont(f);
                                g.drawString(Integer.toString(rct),
                                            350 - 250,
                                            515 + 0 + 35 * Brain.SENSES -
                                            10);
                            g.setFont(fDefault);
                          }
                    }
                          // neurónios que dispararam no Act
                    g.drawString("Fired neurons, by Act...", 350 -
                                250, 515 + 30 + 35 *
                                Brain.SENSES + 30);
                    n = 0;
                    while (n < mind.past.nNeuronsAct && n <
                                MaxDisplay) {
                          if (mind.past.actNode[n].fire)
                            g.drawString("*", 350 + n+1 - 250,
                                      515 + 30 + 35 *
                                      (Brain.SENSES-1) - 3);
                          n += 1;
                    }
                    g.setColor(colorBefore);
                } // fim do if
            }
        }
```

31 “Liberu”

```
/*
 * Liberu.java
 *
 * Created on 30 of August 2005, 2:05
 */
/**
 * web site: www.vitaliberu.pt
 * email: sysliberal@gmail.com
 * @author Paulo Roque Silva
 */
import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;

import javax.swing.*;

import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import gnu.io.SerialPortEvent;
import icommand.nxt.Motor;

public class Liberu extends JFrame {

    public static final long serialVersionUID = 2023L;
    // Versão do Liberu
    static boolean  Already; // Pronto a mostrar
    static int          Robot;
    static Liberu liberu;
    static Interface InterFace; // Interligação com mentor
    Mind            mind; // Mente a executar
    boolean             executed, // Se em execução
                        stopped, // Se parou
                        newUniverse, // Se novo universo
                        terminate, // Se pode new ou open
                        saved; // Se córtex guardado
      SerialPort      serialPort;
      public static String portaCOM = "COM3";
      public static BufferedReader input;
      public static String inputLine;
      public static OutputStream output;
   /** Milliseconds to block while waiting for port open */
      private static final int TIME_OUT = 2000;
      /** Default bits per second for COM port. */
      private static final int DATA_RATE = 9600;

    JTextField      text;           // Texto lido
    JPanel          panel;          // Window
      String                  directory = "", // Diretoria defeito
                              file = ""; // Ficheiro início
```

```
    /** Creates new form Liberu */
    public Liberu() {
        int screenWidth;
        int screenHeight;

        // Get Screen dimensions
        Toolkit kit = Toolkit.getDefaultToolkit();
        Dimension screenSize = kit.getScreenSize();
        screenWidth = screenSize.width;
        screenHeight = screenSize.height;
        // Center frame in Screen
        setLocation( screenWidth / 8, screenHeight / 8 );
        // Set frame icon
        Image img = kit.getImage("Brain.gif");
        setIconImage(img);

        Already      = false;
        Robot        = 0;
        executed     = false;
            stopped      = true;
            newUniverse = true;
            terminate    = true;
          text          = new JTextField(6);
          text.setText("local");
        InterFace    = new Interface();
        mind             = new Mind(false, text);
          mind.display(false, false);
        initComponents();
        if (screenWidth > 2000)
            setSize( 2*768, 2*600);
        else
            setSize( 768, 600);
        saved        = true;

          panel         = new JPanel();
            panel.add(text);
        getContentPane().add(panel, BorderLayout.SOUTH);
    }

      protected void processWindowEvent(WindowEvent e) {

            if (e.getID() == WindowEvent.WINDOW_CLOSING) {
              if (!saved) {
                  if (JOptionPane.showConfirmDialog(null,
      "The Robot's Cortex not saved! Do you want to exit?",
      "Alert", JOptionPane.OK_CANCEL_OPTION) ==
      JOptionPane.OK_OPTION)
                        exitLiberu();
              }
              else
                  exitLiberu();
            }
      }

      public static String inputLine() {

            return inputLine;
      }

    public synchronized void close() {
```

```
        if (serialPort != null) {
//            serialPort.removeEventListener();
            serialPort.close();
      }
    }

    public synchronized void serialEvent(
            SerialPortEvent oEvent) {

        if (oEvent.getEventType() == SerialPortEvent.
                DATA_AVAILABLE) {
            try {
                String inputLine=input.readLine();
                //  ##############################
                System.out.println(inputLine); // RETIRAR
            } catch (Exception e) {
                System.err.println(e.toString());
            }
        }
    }

    private void initComm() {

      // TODO code application logic here
        System.setProperty("gnu.io.rxtx.SerialPorts",
                portaCOM);

        try {
            CommPortIdentifier portId = CommPortIdentifier.
                        getPortIdentifier(portaCOM);
            if (portId == null) {
                    System.out.println(
                              "Could not find COM port.");
                    return;
            }

            serialPort = (SerialPort) portId.open(
                        this.getClass().getName(), TIME_OUT);

            // set port parameters
            serialPort.setSerialPortParams(DATA_RATE,
                        SerialPort.DATABITS_8,
                        SerialPort.STOPBITS_1,
                        SerialPort.PARITY_NONE);

            // open the streams
            input = new BufferedReader(
                        new InputStreamReader(serialPort.
                                    getInputStream()));
            output = serialPort.getOutputStream();

            // add event listeners
//                serialPort.addEventListener(this);
//                serialPort.notifyOnDataAvailable(true);
        } catch (Exception e) {
                System.err.println(e.toString());
        }
      System.out.println("Started");
    }

    private void initComponents() {
```

```
        jPanel1      = new JPanel();
        JPanel         panel; // Window
        jButton0     = new JButton();
        jButton1     = new JButton();
        jButton2     = new JButton();
        jButton3     = new JButton();
        jButton4     = new JButton();
        jButton5     = new JButton();
        jButton6     = new JButton();
        jMenuBar1    = new JMenuBar();
        jMenu1       = new JMenu();
        jMenu3       = new JMenu();
        jMenuItem1   = new JMenuItem();
        jMenuItem2   = new JMenuItem();
        jMenuItem3   = new JMenuItem();
        jMenuItem31  = new JMenuItem();
        jSeparator1  = new JSeparator();
        jMenuItem4   = new JMenuItem();
        jButton2Act  = new JButton();
        jButton3Act  = new JButton();
        jButton4Act  = new JButton();
        jButtonDAct  = new JButton();
        jButtonHAct  = new JButton();
        jLabel1 = new JLabel("Liberu:");
        jLabel2 = new JLabel("Sr. Mouse:");
        jLabel3 = new JLabel("Thumb:");

        setDefaultCloseOperation(
                  WindowConstants.EXIT_ON_CLOSE);
        setTitle(
                  "VitaLiberu 2024 - Vita-Mouse "
                  + " - "     + directory + file);
        jPanel1.add(jLabel1);
        jButton0.setText("Simulation");
        jButton0.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                typeAction(evt);
            }
        });
        jPanel1.add(jButton0);

        jPanel1.add(jLabel1);
        jButton1.setText("Play");
        jButton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                playAction(evt);
            }
        });
        jPanel1.add(jButton1);

        jButton2.setText("Warn");
        jButton2.setBackground(Color.GREEN);
        jButton2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                warnAction(evt);
            }
        });
        jPanel1.add(jButton2);

        jButton3.setText("Pause");
        jButton3.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent evt) {
                pauseAction(evt);
            }
        });
        jPanel1.add(jButton3);

        jButton4.setText("Continue");
        jButton4.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                continueAction(evt);
            }
        });
        jPanel1.add(jButton4);

        jButton5.setText("Stop");
        jButton5.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                stopAction(evt);
            }
        });
        jPanel1.add(jButton5);

        jPanel1.add(jLabel3);

        jButton6.setText("Right Up");
        jButton6.setBackground(Color.GREEN);
        jButton6.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                thumbAction(evt);
            }
        });
        jPanel1.add(jButton6);

        getContentPane().add(jPanel1, BorderLayout.NORTH);

        jMenu1.setText("Cortex");
        jMenuItem1.setText("New Cortex");
        jMenuItem1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                newAction(evt);
            }
        });
        jMenu1.add(jMenuItem1);

        jMenuItem2.setText("Open Cortex...");
        jMenuItem2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                openAction(evt);
            }
        });
        jMenu1.add(jMenuItem2);

        jMenuItem3.setText("Save Cortex...");
        jMenuItem3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                saveAction(evt);
            }
        });
        jMenu1.add(jMenuItem3);

        jMenu1.add(jSeparator1);
```

```
jMenuItem4.setText("Exit Liberu");
jMenuItem4.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        exitAction(evt);
    }
});
jMenu1.add(jMenuItem4);
jMenuBar1.add(jMenu1);

jMenu3.setText("Help");
jMenuItem31.setText("Version...");
jMenuItem31.addActionListener(
          new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        versionAction(evt);
    }
});
jMenu3.add(jMenuItem31);
jMenuBar1.add(jMenu3);

setJMenuBar(jMenuBar1);
getContentPane().add(Liberu.InterFace,
          BorderLayout.CENTER);
  panel        = new JPanel(new GridLayout(16, 1));


  getContentPane().add(panel, BorderLayout.EAST);

panel.add(jLabel2);

jButton2Act.setText("Walk");
jButton2Act.addActionListener(
          new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        walkAction(evt);
    }
});
panel.add(jButton2Act);


jButton3Act.setText("Left");
jButton3Act.addActionListener(
          new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        leftAction(evt);
    }
});
panel.add(jButton3Act);

jButton4Act.setText("Right");
jButton4Act.addActionListener(
          new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        rightAction(evt);
    }
});
panel.add(jButton4Act);

jButtonDAct.setText("Energy");
jButtonDAct.setBackground(Color.CYAN);
```

```
        jButtonDAct.addActionListener(
                  new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                energyAction(evt);
            }
        });
        panel.add(jButtonDAct);

        jButtonHAct.setText("Home");
        jButtonHAct.setBackground(Color.GREEN);
        jButtonHAct.addActionListener(
                  new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                homeAction(evt);
            }
        });
        panel.add(jButtonHAct);

        pack();
    }

    private void newAction(ActionEvent evt) {
      int i;

      if (stopped && terminate) {
            mind = new Mind(false, text);
            mind.display(false, false);
            newUniverse = true;
            for (i=0; i<Interface.Vision.length; i++)
                  Interface.Vision[i] = 0;
            for (i=0; i<Interface.Sound.length; i++)
                  Interface.Sound[i] = 0;
            for (i=0; i<Interface.Smell.length; i++)
                  Interface.Smell[i] = 0;
            Interface.Lovely = 0;
            Interface.QF = 0;
            JOptionPane.showMessageDialog(null,
                        "New Liberu!", "Information",
                JOptionPane.INFORMATION_MESSAGE);
        }
    }

    private void warnAction(ActionEvent evt) {
      int i; // index

        if (executed)
            Cortex.Warn = !Cortex.Warn;
      if (Cortex.Warn) {
            jButton2.setBackground(Color.RED);
      }
      else {
            jButton2.setBackground(Color.GREEN);
                  Emotions.WARNING = false;
                  Interface.Warned = false;
              for (i=0; i < Brain.SENSES; i++)
                  Interface.WarnSense[i] = false;
      }
    }

    private void thumbAction(ActionEvent evt) {
```

```
        if (executed)
            Act.Thumb = !Act.Thumb;
      if (Act.Thumb) {
            jButton6.setBackground(Color.RED);
            jButton6.setText("Left  Up");
      }
      else {
            jButton6.setBackground(Color.GREEN);
            jButton6.setText("Right Up");
      }
    }

    private void pauseAction(ActionEvent evt) {

        if (executed) {
            mind.pause();
            stopped = true;
            if (Robot == 1)
                  try {
                     output.write((byte)113);
                     output.write('\n');
                  }
                  catch (Exception e) {
                     System.err.println(e.toString());
                  }
            if (Robot == 2) {
                  Motor.A.stop();
                  Motor.B.stop();
            }
        }
    }

    private void continueAction(ActionEvent evt) {

      if (executed) {
            stopped = false;
            mind.continueRun();
        }
    }

    private void exitLiberu() {

      if (executed) {
            if (Robot == 1)
                  try {
                     output.write((byte)113);
                     output.write('\n');
                  }
                  catch (Exception e) {
                     System.err.println(e.toString());
                  }
            if (Robot == 2) {
                  Motor.A.stop();
                        Motor.B.stop();
            }
            mind.brain.endLiberu(true);
              Neuron.end = true;
              mind.setToEnd(); // Põe neurónios em fim
              mind.interrupt();
        }
        System.out.println("END Liberu...bye bye");
```

```
        System.exit(0);
    }

    private void exitAction(ActionEvent evt) {

        if (!saved) {
            if (JOptionPane.showConfirmDialog(null,
                  "The Robot's Cortex not, yet, saved! " +
                  "Do you want to exit?",
                  "Alert", JOptionPane.OK_CANCEL_OPTION) ==
                  JOptionPane.OK_OPTION)
                  exitLiberu();
        }
        else
            exitLiberu();
    }

    private void saveAction(ActionEvent evt) {
            File        fout;
            FileDialog  fd;

      if (stopped) {
            fd = new FileDialog(this, "File to Save?",
                        FileDialog.SAVE);
            fd.setVisible(true);
            if (fd.getDirectory() != null &&
                        fd.getFile() != null) {
                  directory = fd.getDirectory();
                  file = fd.getFile();
            }
            fout = new File(directory + file);
              try {
                  // Guarda cÓrtex
                  ObjectOutputStream out =
                              new ObjectOutputStream(
                                    new FileOutputStream(fout));
                  out.writeObject(mind.brain.past);
                  out.close();
                  saved = true;
                  setTitle(
                        "VitaLiberu 2024 - Vita-Mouse "
                              + " - "     + directory + file);
              }
              catch (Exception e) {
                  e.printStackTrace();
              }
        }
    }

    private void openAction(ActionEvent evt) {
            File        fin;
            FileDialog  fd;

      if (stopped && terminate) {
            fd = new FileDialog(this, "File to Load?",
                        FileDialog.LOAD);
            fd.setVisible(true);
            if (fd.getDirectory() != null &&
                        fd.getFile() != null) {
                  directory = fd.getDirectory();
                  file = fd.getFile();
```

```
            }
            fin = new File(directory + file);
            if (fin.exists()) {
              try {
                  // Lê córtex
                  ObjectInputStream in =
                              new ObjectInputStream(
                                    new FileInputStream(fin));
                 mind = new Mind(true, text);
                 mind.brain.past =
                              (Cortex)in.readObject();
                   Neuron.end = false;
                   newUniverse = true;
                   mind.display(true, false);
                 in.close();
                 setTitle(
                       "VitaLiberu 2024 - Vita-Mouse "
                              + " - "     + directory + file);
              }
                catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }

    private void playAction(ActionEvent evt) {

      if (!executed && newUniverse) {
            executed = true;
            terminate = false;
            stopped = false;
            if (Robot == 1)
                  liberu.initComm();  // Formica
            mind.step(false);
            mind.setDaemon(true);
            mind.start();
            mind.display(true, false);
            newUniverse = false;
            saved = false;
        }
    }

    private void stopAction(ActionEvent evt) {

      if (executed) {
            if (JOptionPane.showConfirmDialog(null,
                    "The Robot is going to Stop...",
                    "Alert",
                    JOptionPane.OK_CANCEL_OPTION) ==
                    JOptionPane.OK_OPTION) {
                 mind.setToEnd(); // P►e neur►nios para fim
                 mind.brain.endLiberu(true);
                 mind.interrupt();
                executed = false;
                stopped = true;
                terminate = true;
                mind.line.close();
            }
        }
    }
```

```
    private void versionAction(ActionEvent evt) {

        JOptionPane.showMessageDialog(null,
                  "Liberu, by Paulo Roque Silva... " +
            "2004-2024 www.VitaLiberu.pt"
                  + " - Formica A - " + Brain.NEURONS +
                  " neurons", "Information\n",
                JOptionPane.INFORMATION_MESSAGE);
    }

    private void walkAction(ActionEvent evt) {

      if (executed) {
            mind.brain.idAct1(Brain.WALK);
      }
    }

    private void leftAction(ActionEvent evt) {

      if (executed) {
            mind.brain.idAct1(Brain.TURN_LEFT);
      }
    }

    private void rightAction(ActionEvent evt) {

      if (executed) {
            mind.brain.idAct1(Brain.TURN_RIGHT);
      }
    }

    private void energyAction(ActionEvent evt) {

      if (executed)
            Brain.Energy = !Brain.Energy;
      if (executed && Brain.Energy) {
             jButtonDAct.setBackground(Color.RED);
      }
      if (executed && !Brain.Energy) {
             jButtonDAct.setBackground(Color.CYAN);
      }
    }

    private void homeAction(ActionEvent evt) {

      if (executed)
            Brain.BackHome = !Brain.BackHome;
      if (executed && Brain.BackHome) {
             jButtonHAct.setBackground(Color.RED);
      }
      if (executed && !Brain.BackHome) {
             jButtonHAct.setBackground(Color.GREEN);
      }
    }

    private void typeAction(ActionEvent evt) {

      if (!executed && stopped) {
            Robot += 1;
            if (Robot > 2)
```

```
                    Robot = 0;
        }
        if (!executed && Robot == 1) {
              jButton0.setText("Mouse");
        }
        if (!executed && Robot == 0) {
              jButton0.setText("Simulation");
        }
        if (!executed && Robot == 2) {
              jButton0.setText("NXT");
        }
      }

      /**
       * @param args the command line arguments
       */
      public static void main(String args[])
              throws Exception {

        EventQueue.invokeLater(new Runnable() {
              public void run() {
                    liberu = new Liberu();
                    liberu.setVisible(true);
                    if (args.length > 0)
                    Liberu.portaCOM = (String)args[0];
              }
          });
      }
      private JButton    jButton0;
      private JButton    jButton1;
      private JButton    jButton2;
      private JButton    jButton3;
      private JButton    jButton4;
      private JButton    jButton5;
      private JButton    jButton6;
      private JMenu      jMenu1;
      private JMenu      jMenu3;
      private JMenuBar   jMenuBar1;
      private JMenuItem  jMenuItem1;
      private JMenuItem  jMenuItem2;
      private JMenuItem  jMenuItem3;
      private JMenuItem  jMenuItem4;
      private JMenuItem  jMenuItem31;
      private JPanel     jPanel1;
      private JSeparator jSeparator1;
      private JButton    jButton2Act;
      private JButton    jButton3Act;
      private JButton    jButton4Act;
      public static JButton    jButtonDAct;
      public static JButton    jButtonHAct;
      private JLabel         jLabel1;
      private JLabel         jLabel2;
      private JLabel         jLabel3;
}
```

32 “Mind”

```
/*
 * Mind.java
 *
 * Created on 1 of February 2006, 2:06
 */

/**
 *
 * @author Paulo Roque Silva
 */

/**
 *
      TouchSensor(SensorPort.S1);
      SoundSensor(SensorPort.S2);
      LightSensor(SensorPort.S3);
      UltrasonicSensor(SensorPort.S4);

      Motor.A  Left leg
      Motor.B  Right leg
 */

import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.BrokenBarrierException;
import icommand.nxt.comm.NXTCommand;
import icommand.nxt.LightSensor;
import icommand.nxt.SensorPort;
import icommand.nxt.TouchSensor;
import icommand.nxt.UltrasonicSensor;
import icommand.nxt.SoundSensor;
import icommand.nxt.Motor;
import javax.swing.*;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;
import javax.sound.sampled.Port;
import javax.sound.sampled.TargetDataLine;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioFormat.Encoding;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.Clip;
import javax.sound.sampled.AudioInputStream;
import java.io.*;

public class Mind extends Thread {
      static final int TimeSlize = 0;
                                // Tempo (ms) intervalo entre passos
    final int       StartLocal  = 61;
                        // Local de começo do C. Elegans
      static final int PROFOUNDITY = 6; // Profundidade Sleep
    Brain           brain; // Todo o cérebro e córtex
// CONSTANTES DO CORPO
    private final int FeelWall20 = 20;
                        // Se inferior em centímetros sente
                        // obstáculo
    static final int Power = 100; // Amplitude
    static final int BANDS = 7; // Bandas de som
    final int           SAMPLE_RATE = 16000; // Amostragem som
```

```
    static final int TimeWait  = 64; // Tempo (ms) espera
 // VARIÁVEIS DO CORPO
    public static Signal []Action; // Ação do robô
    public static int Tact[], // Sentido Tacto do robô
                      NStep; // Número passo do robô
      int                   bottom, // NXT
                            ligth, // NXT
                            ultraSonic, // NXT
                            sound; // NXT
      TouchSensor       bottomN; // NXT
      LightSensor       ligthN; // NXT
      UltrasonicSensor ultraSonicN; // NXT
      SoundSensor     soundN; // NXT
      boolean               feelPain; // Se o robô sente Dor
    int                 senteTaste[];
    int             senteVision[];
    int             senteTact[];
    int                 senteSound[];
    int                 senteSmell[];
    int             senteProSelf[]; // Propriocetivo
    int                   tasteBefore[]; // Sensação anterior
    int                   smellBefore[];
    int                   visionBefore[];
    int                   pleasurePS[], // Nível de prazer
                        pleasureTa[],
                        pleasureTc[],
                        pleasureSm[],
                        pleasureSo[],
                        pleasureVi[];
    int                   quantity[]; // Nível de prazer
      int                 m; // indexe
      int                 profoundity; // indexe
    public static int Bom; // Para agir
    public static float veloc; // Velocidade, 1- Normal

    CyclicBarrier   stepBarrierAct; // Barreiras
    CyclicBarrier   stepBarrierDecision;
    CyclicBarrier   stepBarrierDecision2;
    CyclicBarrier   stepBarrierPast;
    CyclicBarrier   stepBarrierCerebellum;
    CyclicBarrier   stepBarrierSleep;

// VARIÁVEIS DA SIMULAÇÃO
    int                   i; // Índexe
    boolean         pause; // Se para parar
      public static boolean OpenData; // Se dados antigos

    static int      local; // Actual local
      int                 orient, // Orientação do C. Elegans
                          wallOrLocal, // Local ou Parede
                          tact, // Parede ou não
                          nutrition, // Comida, água ou Terra
                          vision, // Visão
                          nStep; // Passo movimentado pelo robô
    World           habitat; // Mundo do C. Elegans
    TargetDataLine  line; // Microfone
    AudioFormat         format; // Formato áudio
    int                   numBytesRead;
    int                   numBytesReadTotal;
```

```
    byte[]          data;
    DataLine.Info   info,
                    info2,
                        info3,
                        info4,
                        info5,
                        info6;
    AudioInputStream stream,
                        stream2,
                        stream3,
                        stream4,
                              stream5;
    public static Clip clip,
                        clip2,
                        clip3,
                        clip4, // Rir
                        clip5; // Chorar
    /** Creates a new instance of Mind */
    public Mind(boolean od, JTextField txt) {

        System.out.println(
            "VitaLiberu...by Paulo Roque Silva 2004-2024");
        brain        = new Brain(txt);
        Action      = new Signal[Brain.MOVES];
        for (int i=0; i < Action.length; i++)
            Action[i] = new Signal(i, 0, 0);
        if (Liberu.Robot == 0) {
            senteTaste  = new int[Brain.NanalogTaste];
            pleasureTa  = new int[Brain.NanalogTaste];
            tasteBefore = new int[Brain.NanalogTaste];
            senteVision = new int[Brain.NanalogVision];
            pleasureVi  = new int[Brain.NanalogVision];
            senteTact   = new int[Brain.NanalogTact];
            pleasureTc  = new int[Brain.NanalogTact];
            senteSound  = new int[Brain.NanalogSound];
            pleasureSo  = new int[Brain.NanalogSound];
            senteSmell  = new int[Brain.NanalogSmell];
            pleasureSm  = new int[Brain.NanalogSmell];
            Tact        = new int[Brain.NanalogTact];
            visionBefore= new int[Brain.NanalogVision];
            smellBefore= new int[Brain.NanalogSmell];
            }
        if (Liberu.Robot == 1) {
            senteTaste  = new int[Brain.NanalogTasteR];
            pleasureTa  = new int[Brain.NanalogTasteR];
            tasteBefore = new int[Brain.NanalogTasteR];
            senteVision = new int[Brain.NanalogVisionR];
            pleasureVi  = new int[Brain.NanalogVisionR];
            senteTact   = new int[Brain.NanalogTactR];
            pleasureTc  = new int[Brain.NanalogTactR];
            senteSound  = new int[Brain.NanalogSoundR];
            pleasureSo  = new int[Brain.NanalogSoundR];
            senteSmell  = new int[Brain.NanalogSmellR];
            pleasureSm  = new int[Brain.NanalogSmellR];
            Tact        = new int[Brain.NanalogTactR];
            visionBefore= new int[Brain.NanalogVisionR];
            smellBefore= new int[Brain.NanalogSmellR];
        }
        if (Liberu.Robot == 2) {
            senteTaste  = new int[Brain.NanalogTasteN];
            pleasureTa  = new int[Brain.NanalogTasteN];
```

```
            tasteBefore = new int[Brain.NanalogTasteN];
            senteVision = new int[Brain.NanalogVisionN];
            pleasureVi  = new int[Brain.NanalogVisionN];
            senteTact   = new int[Brain.NanalogTactN];
            pleasureTc  = new int[Brain.NanalogTactN];
            senteSound  = new int[Brain.NanalogSoundN];
            pleasureSo  = new int[Brain.NanalogSoundN];
            senteSmell  = new int[Brain.NanalogSmellN];
            pleasureSm  = new int[Brain.NanalogSmellN];
            Tact        = new int[Brain.NanalogTactN];
            visionBefore= new int[Brain.NanalogVisionN];
            smellBefore= new int[Brain.NanalogSmellN];
        }
        senteProSelf= new int[Brain.NanalogProSelf];
        pleasurePS  = new int[Brain.NanalogProSelf];
        quantity    = new int[1];
        quantity[0] = 0;
        NStep       = 0;
        pause           = true;
        tasteBefore[0] = 0;
      for (i=0; i<visionBefore.length; i++)
            visionBefore[i] = 0;
      for (i=0; i<smellBefore.length; i++)
            smellBefore[i] = 0;
        OpenData    = od;
        veloc           = 1;
        habitat     = new World();
        local = StartLocal;
        orient = World.NORTH;
        tact        = Brain.NO_TACT;
        nutrition   = habitat.element(local);
        vision      = habitat.vision(local, orient);
        display(true, false);
    }

    public void step(boolean p) {

      pause = p;
    }

    public static int getBom() {

        return Math.abs(Bom);
    }

    public static void setBom(int b) {

        Bom = b;
    }

    public void display(boolean ex, boolean w) {

        Liberu.InterFace.displayRobot(brain, ex, w);
        Liberu.InterFace.saveValue(0, local);
        Liberu.InterFace.saveValue(1, tact);
        Liberu.InterFace.saveValue(2, nutrition);
        Liberu.InterFace.saveValue(4, orient);
        Liberu.InterFace.saveValue(5, vision);

        for (i=0; i<Brain.ACTS; i++)
            if (Action[i].active)
```

```
                Liberu.InterFace.saveValue(8+i, 1);
            else
                Liberu.InterFace.saveValue(8+i, 0);
        Liberu.InterFace.repaint();
    }

    private void ouve() {
      int n, m, // Índexes
            derivada, // Diferença no som
            readBefore,
            bandBefore,
            totalHits;

      numBytesReadTotal = 0;
      line.flush();
      readBefore = 0;
      System.out.println("Ouvindo...");
      while (numBytesReadTotal < SAMPLE_RATE * 2) {
            // Tempo ouvindo
            numBytesRead = line.read(data, 0, data.length);
            numBytesReadTotal += numBytesRead;
            for (n=0; n < numBytesRead; n++) {
                  derivada = Math.abs(data[n] - readBefore);
                  bandBefore = 0;
                  for (m=0; m < BANDS; m++) {
                        if (derivada >= bandBefore &&
                                    derivada < (int)(32000 /
                                                Math.pow(2, BANDS-1-m)/
                                                Brain.MaxInSense)) {
                              senteSound[m] += 1;
                              bandBefore = (int)(32000 /
                                          Math.pow(2, BANDS-1-m)/
                                          Brain.MaxInSense);
                        }
                  }
                  readBefore = data[n];
            }
      }
            System.out.println("Surdo...");
            totalHits = 0;
            for (m=0; m < BANDS; m++)
                  totalHits += senteSound[m];
            if (totalHits == 0) totalHits = 1;
            for (m=0; m < BANDS; m++)
                  senteSound[m] = (int)(senteSound[m] *
                              Brain.MaxInSense / totalHits);
            for (m=0; m < BANDS; m++)
              Liberu.InterFace.saveValue(36+m,
                        senteSound[m]);
    }

    private void actNXT(float veloc) {

            if (Action[Brain.WALK].active ) {
                  Motor.A.setSpeed((int)(veloc *
                              (int)Action[Brain.WALK].actPower));
                  // 900 = 100% power
                  Motor.B.setSpeed((int)(veloc *
                              (int)Action[Brain.WALK].actPower));
                  // Graus por segundo
                  Motor.A.forward();
```

```
                Motor.B.forward();
            }
            if ( Action[Brain.TURN_RIGHT].active) {
                Motor.A.setSpeed((int)(veloc *
                            (int)Action[Brain.TURN_RIGHT].
                            actPower));
                // 900 = 100% power
                Motor.B.setSpeed((int)(veloc *
                            (int)Action[Brain.TURN_RIGHT].
                            actPower));
                // Graus por segundo
                Motor.A.backward();
                Motor.B.forward();
            }
            if (Action[Brain.TURN_LEFT].active) {
                Motor.A.setSpeed((int)(veloc *
                            (int)Action[Brain.TURN_LEFT].
                            actPower));
                // 900 = 100% power
                Motor.B.setSpeed((int)(veloc *
                            (int)Action[Brain.TURN_LEFT].
                            actPower));
                // Graus por segundo
                Motor.A.forward();
                Motor.B.backward();
            }
            if (Action[Brain.EAT].active) {
                brain.decision.emotionSpeak();;
            }
    }

    private void actRobot(float veloc) {

            if (Action[Brain.TURN_LEFT].active) {
            try {
                Liberu.output.write((byte)112);
                Liberu.output.write('\n');
            }
            catch (Exception e) {
                System.err.println(e.toString());
            }
            try {
                Liberu.output.write((byte)116);
                Liberu.output.write('\n');
            }
            catch (Exception e) {
                System.err.println(e.toString());
            }
            }
            if (Action[Brain.TURN_RIGHT].active) {
            try {
                Liberu.output.write((byte)111);
                Liberu.output.write('\n');
            }
            catch (Exception e) {
                System.err.println(e.toString());
            }
            try {
                Liberu.output.write((byte)115);
                Liberu.output.write('\n');
            }
```

```
            catch (Exception e) {
                System.err.println(e.toString());
            }
            }
            if (Action[Brain.WALK].active) {
            try {
                Liberu.output.write((byte)111);
                Liberu.output.write('\n');
            }
            catch (Exception e) {
                System.err.println(e.toString());
            }
            try {
                Liberu.output.write((byte)116);
                Liberu.output.write('\n');
            }
            catch (Exception e) {
                System.err.println(e.toString());
            }
            }
    }

    private void updateSensesNXT() { // Comandos NXT
      int   m; // Índexe

        // TACT
      for (i=0; i<pleasureTc.length; i++)
            pleasureTc[i] = 0;
      Tact[0] = Brain.NO_TACT;
            feelPain = false;
            if (Brain.Energy)
                  pleasureTc[0] = Power;
            if (bottomN.isPressed()) {
                  feelPain = true;
                  Tact[0] = Brain.TACT;
                  pleasureTc[0] = -Power;
                  senteTact[0] = Power; // Sente tato
                  if (brain.past.time[0][0].step >
                  Emotions.ActRandom)
                        brain.decision.emotionPAIN();
            }
            else
                  senteTact[0] = (int)Power/2;
                        // Não sente tato

        // TASTE
      for (i=0; i<pleasureTa.length; i++)
            pleasureTa[i] = 0;
        senteTaste[0] = ligthN.getLightPercent();
        Interface.Taste[0] = senteTaste[0];
        if (NStep > 1) {
            if (Brain.BackHome)
                pleasureTa[0] = tasteBefore[0] -
                senteTaste[0];
            else
                pleasureTa[0] = senteTaste[0] -
                tasteBefore[0];
        }
        else
            pleasureTa[0] = 0;
        brain.feelTaste(senteTaste, pleasureTa);
```

```
        tasteBefore[0] = senteTaste[0];

      for (m=0; m < senteTaste.length; m++)
            senteTaste[m] = 0;

        // SOUND
      for (i=0; i<pleasureSo.length; i++)
            pleasureSo[i] = 0;
            System.out.println("Lisen..................:)");
            // REMOVER
            for (m=0; m < BANDS; m++) {
                  senteSound[m] = soundN.getdB();
                  Interface.Sound[m] = senteSound[m];
            }
            for (m=0; m < BANDS; m++) {
                  if (m != 0)
                        pleasureSo[m] = senteSound[m] -
                        senteSound[m-1];
                  else
                        pleasureSo[0] = 0;
            }
            brain.feelSound(senteSound, pleasureSo); // Sente

      for (m=0; m < senteSound.length; m++)
            senteSound[m] = 0;

            // SMELL
      for (i=0; i<pleasureSm.length; i++)
            pleasureSm[i] = 0;
        if (NStep > 1)
            for (m=0; m < senteSmell.length; m++)
                pleasureSm[m] = senteSmell[m] -
                smellBefore[m]; // Melhor se brilhante
        else
            pleasureVi[0] = 0;
            brain.feelSmell(senteSmell, pleasureSm); // Sente

      for (m=0; m < senteSmell.length; m++) {
            smellBefore[m] = senteSmell[m];
            senteSmell[m] = 0;
      }

        // VISION
      for (i=0; i<pleasureVi.length; i++)
            pleasureVi[i] = 0;
            senteVision[0] = ultraSonicN.getDistance();
            for (m=0; m < senteVision.length; m++)
                  Interface.Vision[m] = senteVision[m];
        if (NStep > 1)
            for (m=0; m < senteVision.length; m++)
                pleasureVi[m] = visionBefore[m] -
                senteVision[m]; // Melhor se pr?ximo
        else
            pleasureVi[0] = 0;
      if (senteVision[0] < FeelWall20 ) {
            feelPain = true;
            Tact[0] = Brain.TACT;
            senteTact[0] = Power; // Sente tato
            pleasureTc[0] = -Power;
            if (brain.past.time[0][0].step >
            Emotions.ActRandom)
```

```
                brain.decision.emotionPAIN();
        }
          brain.feelVision(senteVision, pleasureVi);

        for (m=0; m < senteVision.length; m++) {
              visionBefore[m] = senteVision[m];
              senteVision[m] = 0;
        }

          brain.feelTact(senteTact, pleasureTc);

        for (m=0; m < senteTact.length; m++)
              senteTact[m] = 0;
      }

      private void updateSensesRobot() { // Comandos Formica
        int   m, space;
        String s;

        for (m=0; m < senteVision.length; m++)
              senteVision[m] = 0;
        for (m=0; m < senteTact.length; m++)
              senteTact[m] = 0;
        for (m=0; m < senteSmell.length; m++)
              senteSmell[m] = 0;
        for (m=0; m < senteTaste.length; m++)
              senteTaste[m] = 0;
        for (m=0; m < senteSound.length; m++)
              senteSound[m] = 0;
        for (m=0; m < senteProSelf.length; m++)
              senteProSelf[m] = 0;
          // TACT
        s="";
        for (i=0; i<pleasureTc.length; i++)
              pleasureTc[i] = 0;
        Tact[0] = Brain.NO_TACT;
              feelPain = false;
              try {
              Liberu.output.write((byte)99);
              Liberu.output.write('\n');
              while (!Liberu.input.ready())
                  sleep(1);
              s = Liberu.input.readLine();
              }
        catch (Exception e) {
              System.err.println(e.toString() + "Tact");
        }
              if (Brain.Energy)
                    pleasureTc[0] = Power;
              if (s.length()>0) {
              if (s.contentEquals("1")) {
                    feelPain = true;
                    Tact[0] = Brain.TACT;
                    pleasureTc[0] = -Power;
                    senteTact[0] = Power; // Sente tato
                    if (brain.past.time[0][0].step >
                    Emotions.ActRandom)
                          brain.decision.emotionPAIN();
              }
              else
                    senteTact[0] = (int)Power/2;
```

```
            // Não Sente tato
            }

        // TASTE
      for (i=0; i<pleasureTa.length; i++)
            pleasureTa[i] = 0;
      s = "";
            try {
                  Liberu.output.write((byte)100);
            Liberu.output.write('\n');
            while (!Liberu.input.ready())
                sleep(1);
            s = Liberu.input.readLine();
            }
      catch (Exception e) {
            System.err.println(e.toString() + "Taste");
      }
            if (s.length()>0)
              senteTaste[0] = Integer.parseInt(s);
        Interface.Taste[0] = senteTaste[0];
        if (NStep > 1) {
            if (Brain.BackHome)
                pleasureTa[0] = tasteBefore[0] -
                senteTaste[0];
            else
                pleasureTa[0] = senteTaste[0] -
                tasteBefore[0];
        }
        else
            pleasureTa[0] = 0;
        tasteBefore[0] = senteTaste[0];

        // SOUND
      for (i=0; i<pleasureSo.length; i++)
            pleasureSo[i] = 0;
            System.out.println("Lisen.................:)");
            // REMOVER
            s = "";
            try {
            Liberu.output.write((byte)103);
            Liberu.output.write('\n');
            while (!Liberu.input.ready())
                sleep(1);
            s = Liberu.input.readLine();
            }
      catch (Exception e) {
            System.err.println(e.toString() + "Sound");
      }
            space=0;
            for (m=0; m < BANDS; m++) {
                  if (s.length()>0) {
                        space = s.indexOf(' ');
                        senteSound[m] =
                                    Integer.parseInt(s.substring(0,
                                                space));
                        s = s.substring(space+1, s.length());
                  }
                  Interface.Sound[m] = senteSound[m];
            }
            for (m=0; m < BANDS; m++) {
                  if (m != 0)
```

```
                        pleasureSo[m] = senteSound[m] -
                        senteSound[m-1];
                else
                        pleasureSo[0] = 0;
        }

        // SMELL
for (i=0; i<pleasureSm.length; i++)
        pleasureSm[i] = 0;
        s = "";
        try {
        Liberu.output.write((byte)98);
        Liberu.output.write('\n');
        while (!Liberu.input.ready())
            sleep(1);
        s = Liberu.input.readLine();
        }
catch (Exception e) {
        System.err.println(e.toString() + "Smell");
}
        space=0;
        for (m=0; m < senteSmell.length; m++) {
                if (s.length()>0) {
                        space = s.indexOf(' ');
                        senteSmell[m] = Integer.parseInt(
                                    s.substring(0, space));
                        s = s.substring(space+1, s.length());
                }
                Interface.Smell[m] = senteSmell[m];
        }
  if (NStep > 1)
        for (m=0; m < senteSmell.length; m++)
            pleasureSm[m] = senteSmell[m] -
            smellBefore[m]; // Melhor se brilhante
  else
        pleasureVi[0] = 0;

for (m=0; m < senteSmell.length; m++) {
        smellBefore[m] = senteSmell[m];
}

  // VISION
for (i=0; i<pleasureVi.length; i++)
        pleasureVi[i] = 0;
        s = "";
        try {
        Liberu.output.write((byte)101);
        Liberu.output.write('\n');
        while (!Liberu.input.ready())
            sleep(1);
        s = Liberu.input.readLine();
        }
catch (Exception e) {
        System.err.println(e.toString() + "VisionD");
}
        if (s.length()>0)
                senteVision[0] = Integer.parseInt(s);
        s = "";
        try {
        Liberu.output.write((byte)102);
        Liberu.output.write('\n');
```

```
            while (!Liberu.input.ready())
                sleep(1);
            s = Liberu.input.readLine();
            }
      catch (Exception e) {
            System.err.println(e.toString() + "VisionE");
      }
            if (s.length()>0)
                  senteVision[1] = Integer.parseInt(s);
            for (m=0; m < senteVision.length; m++)
                  Interface.Vision[m] = senteVision[m];
        if (NStep > 1)
            for (m=0; m < senteVision.length; m++)
                pleasureVi[m] = visionBefore[m] -
                senteVision[m]; // Melhor se pr?ximo
        else
            pleasureVi[0] = 0;
      if (senteVision[0] < FeelWall20 || senteVision[1] <
                  FeelWall20) {
            feelPain = true;
            Tact[0] = Brain.TACT;
            senteTact[0] = Power; // Sente tato
            pleasureTc[0] = -Power;
            if (brain.past.time[0][0].step >
            Emotions.ActRandom)
                  brain.decision.emotionPAIN();
      }

      for (m=0; m < senteVision.length; m++) {
            visionBefore[m] = senteVision[m];
      }

            if (Action[Brain.DRINK].active)
                  senteProSelf[Brain.DRINK] = Power;
            if (Action[Brain.EAT].active)
                  senteProSelf[Brain.EAT] = Power;
            if (Action[Brain.WALK].active) {
                  senteProSelf[Brain.WALK] = Power;
                  pleasurePS[Brain.WALK] = Power * 3;
            }
            if (Action[Brain.TURN_LEFT].active) {
                  senteProSelf[Brain.TURN_LEFT] = Power;
                  pleasurePS[Brain.TURN_LEFT] = Power * 1;
            }
            if (Action[Brain.TURN_RIGHT].active)
                  senteProSelf[Brain.TURN_RIGHT] = Power;
            if (Action[Brain.Motive].active)
                  senteProSelf[Brain.Motive] = Power;
    }

    private void updateSenses() { // Comandos C.Elegants
      int m, temp;

      for (m=0; m < senteVision.length; m++)
            senteVision[m] = 0;
      for (m=0; m < senteTact.length; m++)
            senteTact[m] = 0;
      for (m=0; m < senteSmell.length; m++)
            senteSmell[m] = 0;
      for (m=0; m < senteTaste.length; m++)
            senteTaste[m] = 0;
```

```
for (m=0; m < senteSound.length; m++)
      senteSound[m] = 0;
for (m=0; m < senteProSelf.length; m++)
      senteProSelf[m] = 0;
// OUVE
for (i=0; i<pleasureSo.length; i++)
      pleasureSo[i] = 0;
      System.out.println("Lisen.................:)");
      // REMOVER
ouve();
      for (m=0; m < BANDS; m++) {
            if (m != 0)
                  pleasureSo[m] = senteSound[m] -
                  senteSound[m-1];
            else
                  pleasureSo[0] = 0;
      }

if (Action[Brain.TURN_RIGHT].active &&
            Action[Brain.TURN_LEFT].active)
         ;
      else {
      if (Action[Brain.TURN_LEFT].active)
            orient = habitat.orientate(orient,
                        Brain.TURN_LEFT);
        if (Action[Brain.TURN_RIGHT].active)
            orient = habitat.orientate(orient,
                        Brain.TURN_RIGHT);
      }

  // TACT
for (i=0; i<pleasureTc.length; i++)
      pleasureTc[i] = 0;
Tact[0] = Brain.NO_TACT;
      feelPain = false;
  tact = Brain.NO_TACT;
  if (Action[Brain.WALK].active) {
      wallOrLocal = habitat.move(local, orient);
      if (wallOrLocal != World.WALL) {
          local = wallOrLocal; // Novo local
            senteTact[0] = (int)Power/2;
            // Não sente tato
      }
      else {
          tact = Brain.TACT; // Sente obstáculo
            senteTact[0] = Power; // Sente tato
            if (brain.past.time[0][0].step >
            Emotions.ActRandom) {
                  feelPain = true;
              Tact[0] = Brain.TACT;
                  pleasureTc[0] = -Power;
            senteTact[0] = Power; // Sente tato
                  if (brain.past.time[0][0].step >
                  Emotions.ActRandom)
                        brain.decision.emotionPAIN();
            }
      }
  }
      if (Brain.Energy)
            pleasureTc[0] += Power;
```

```
            nutrition = habitat.element(local);
        vision = habitat.vision(local, orient);

        temp = orient;

        // TASTE
      for (i=0; i<pleasureTa.length; i++)
            pleasureTa[i] = 0;

      senteTaste[nutrition-1] = Power;

        if (Action[Brain.EAT].active &&
                  nutrition != World.FOOD) {
                  feelPain = true;
                  if (brain.past.time[0][0].step >
                  Emotions.ActRandom)
                        brain.decision.emotionPAIN();
        }

        if (Action[Brain.DRINK].active &&
                  nutrition != World.WATER) {
                  feelPain = true;
                  if (brain.past.time[0][0].step >
                  Emotions.ActRandom)
                        brain.decision.emotionPAIN();
        }

      for (m=0; m < senteTaste.length; m++)
            if (NStep > 1) {
                  if (Brain.BackHome)
                        pleasureTa[m] = (senteTaste[m] -
                                    tasteBefore[m]) * 5;
                  else
                        pleasureTa[m] = (tasteBefore[m] -
                                    senteTaste[m] * 5);
            }
            else
                  pleasureTa[m] = 0;

            // SMELL
      for (i=0; i<pleasureSm.length; i++)
            pleasureSm[i] = 0;
      senteSmell[local-1] = Power;
        pleasureSm[0] = local * 5;

        // VISION
      for (i=0; i<pleasureVi.length; i++)
            pleasureVi[i] = 0;
        temp = habitat.orientate(orient, Brain.TURN_LEFT);
        senteVision[0] = habitat.vision(local, temp);
        temp = habitat.orientate(orient, Brain.TURN_RIGHT);
        senteVision[2] = habitat.vision(local, temp);
      senteVision[1] = vision;
            for (m=0; m < senteVision.length; m++)
                  Interface.Vision[m] = senteVision[m];
        if (NStep > 1)
            for (m=0; m < senteVision.length; m++)
                  pleasureVi[m] = (visionBefore[m] -
                        senteVision[m]) * 5;
        else
            pleasureVi[0] = 0;
```

```
        // PROPRIOCETIVO
            if (Action[Brain.DRINK].active)
                  senteProSelf[Brain.DRINK] = Power;
            if (Action[Brain.EAT].active)
                  senteProSelf[Brain.EAT] = Power;
            if (Action[Brain.WALK].active) {
                  senteProSelf[Brain.WALK] = Power;
                  pleasurePS[Brain.WALK] = Power * 3;
            }
            if (Action[Brain.TURN_LEFT].active) {
                  senteProSelf[Brain.TURN_LEFT] = Power;
                  pleasurePS[Brain.TURN_LEFT] = Power * 1;
            }
            if (Action[Brain.TURN_RIGHT].active)
                  senteProSelf[Brain.TURN_RIGHT] = Power;
            if (Action[Brain.Motive].active)
                  senteProSelf[Brain.Motive] = Power;
    }

    private void senseProSelfNXT() {

            // PROPRIOCETIVO
      for (i=0; i<pleasurePS.length; i++)
            pleasurePS[i] = 0;
        pleasurePS[0] = brain.proSelfNXT(senteProSelf,
                  ligthN, Power); // Atualiza propriocetivo
        brain.motivationNXT(senteProSelf, Power);
        // Motivação NXT
        if (feelPain)
            pleasurePS[0] += -Power;
        brain.feelProSelf(senteProSelf, pleasurePS);

      for (m=0; m < senteProSelf.length; m++)
            senteProSelf[m] = 0;
    }

    private void senseProSelfRobot() {

            // PROPRIOCETIVO
      for (i=0; i<pleasurePS.length; i++)
            pleasurePS[i] = 0;
        pleasurePS[0] = brain.proSelfRobot(senteProSelf,
                  Power); // Atualiza propriocetivo
        brain.motivationNXT(senteProSelf, Power);
        // Motivação
        if (feelPain)
            pleasurePS[0] += -Power;
        brain.feelProSelf(senteProSelf, pleasurePS);
    }

    private void senseProSelf() {

            // PROPRIOCETIVO
      for (i=0; i<pleasurePS.length; i++)
            pleasurePS[i] = 0;
        pleasurePS[0] = brain.proSelf(senteProSelf,
                  senteTaste, Power);
        // Atualiza propriocetivo
        brain.motivationNXT(senteProSelf, Power);
        // Motivação
```

```
        if (feelPain)
            pleasurePS[0] += -Power;
        brain.feelProSelf(senteProSelf, pleasurePS);
    }

    public void Sono() {

        // Sleep in action ********************************
            Brain.Ansiedade = false;
            Brain.Saudade = false;
        brain.clearBusSleepPreviews();
        brain.clearBusSleepPreviews2();
            Brain.Best = 0;
            setBom(Math.abs(Emotions.Good));
            Brain.BusSleepPreviews.add(new Signal(Bom,Bom,
                        brain.past.time[getBom()][Brain.ProSelf].
                        finalQuality));
      for (i=0; i<brain.past.time[getBom()][0].
                learn.size(); i++)
            Brain.BusSleepPreviews2.add(new Signal(
                        brain.past.time[getBom()][0].
                        getLearn(i).emitter,
                        brain.past.time[getBom()][0].
                        getLearn(i).emitter,
                        brain.past.time[getBom()]
                                    [Brain.ProSelf].finalQuality));
            profoundity = 0;
            while (profoundity < PROFOUNDITY) {
                  brain.pollSleepOut();
                  brain.pollSleepOut2();
              try {
                  stepBarrierSleep.await(); // Início Sleep
                  while (stepBarrierSleep.
                              getNumberWaiting() <
                              stepBarrierSleep.getParties() - 1)
                      sleep(TimeWait);
              }
              catch (InterruptedException ex) {
              }
              catch (BrokenBarrierException ex) {
            }
                  profoundity += 1;

              brain.decide(profoundity);

                  brain.transferOutIn();


                  System.out.println("SONO acting...");
                  // REMOVER
                  brain.decide2(profoundity);

                  brain.transferOutIn2();
            }

            Cortex.exec = Bom;
      System.out.print("Bom= "); //REMOVER
      System.out.println(Bom); //REMOVER
    }

    private void runStepBrain() {
```

```
int i;

  // COMEÇA O CICLO
  NStep = brain.addStep();
  System.out.print("Passo = "); // REMOVER
  System.out.println(NStep); // REMOVER
if (brain.past.time[0][0].step == 0)
      brain.past.time[0][0].standby2 = true;

// Começa o passo de vida do robô
  brain.clearBusesIn(true);
      brain.feelSound(senteSound, pleasureSo); // Sente
  brain.feelTaste(senteTaste, pleasureTa);
      brain.feelSmell(senteSmell, pleasureSm);
  brain.feelVision(senteVision, pleasureVi);
  brain.feelTact(senteTact, pleasureTc);
  if (Liberu.Robot == 0)
      senseProSelf();
  if (Liberu.Robot == 1)
      senseProSelfRobot();
  if (Liberu.Robot == 2)
      senseProSelfNXT();

for (m=0; m < senteVision.length; m++)
      visionBefore[m] = senteVision[m];
for (m=0; m < senteSmell.length; m++)
      smellBefore[m] = senteSmell[m];
for (m=0; m < senteTaste.length; m++)
      tasteBefore[m] = senteTaste[m];

  display(true, false); // Mostra ecrã

  // Fim do tempo real

  for (i=0; i<Action.length; i++)   { // reset actions
      Action[i].setActive(false);
      Action[i].setActPower(0);
      Action[i].setPleasure(0);
  }

  Remember.Lovely = 0;
  Interface.Lovely = 0;
Remember.RecentMost = 0;
      Remember.threshold = 0;
  for (i=0; i<Brain.SENSES; i++)
      Remember.NewSense[i] = true;
// Remember em ação ******************************
  try {
      stepBarrierPast.await(); // Sincroniza
      while (stepBarrierPast.getNumberWaiting() <
                  stepBarrierPast.getParties() - 1)
          sleep(TimeWait);
  }
  catch (InterruptedException ex) {
      return;
  }
  catch (BrokenBarrierException ex) {
      return;
      }
      System.out.println("Past Remember..."); // REMOVER
      Emotions.Good = Remember.RecentMost;
```

```
brain.clearBusSleepPreviews();
  // Emotions em a►►o ******************************
  try {
      stepBarrierDecision.await(); // Sincroniza
      while (stepBarrierDecision2.
                  getNumberWaiting() <
                  stepBarrierDecision2.getParties() - 1)
          sleep(TimeWait);
  }
  catch (InterruptedException ex) {
      return;
  }
  catch (BrokenBarrierException ex) {
      return;
      }
      System.out.println("Emotions acted..."); // REMOVER
System.out.print("recent= "); //REMOVER
System.out.println(Remember.RecentMost); //REMOVER
display(true, false); // shows in screen
if (brain.past.time[0][0].step < 2) {
      if (brain.past.time[0][0].step == 0)
            Cortex.lastExec = Remember.RecentMost;
      else
            if (!Emotions.ActualPain) {
                  brain.past.time[Math.abs(
                              Cortex.beforeLastExec)][0].
                  standby2 = true;
                  Cortex.beforeLastExec = Cortex.
                              lastExec;
            }
}
else
      if (!Emotions.ActualPain) {
            brain.past.time[Math.abs(Cortex.
                        beforeLastExec)][0].standby2 =
                        true;
      System.out.println(Cortex.beforeLastExec);
      //REMOVER
      }
if (!Emotions.ActualPain) {
      Cortex.beforeLastExec = Cortex.lastExec;
      Cortex.lastExec = Remember.RecentMost;
}
Cortex.exec = Remember.RecentMost;
if (Emotions.Good >= 0 && Emotions.IdAct1 ==
            Cerebellum.ESCape)
        // Sleep em ação ******************************
      Sono();
System.out.print("exec= "); //REMOVER
System.out.println(Cortex.exec); //REMOVER
      // Cerebellum em ação ****************************
  try {
      stepBarrierCerebellum.await(); // Sincroniza
      while (stepBarrierCerebellum.
                  getNumberWaiting() <
                  stepBarrierCerebellum.getParties() - 1)
          sleep(TimeWait);
  }
  catch (InterruptedException ex) {
      return;
  }
```

```
        catch (BrokenBarrierException ex) {
            return;
            }
            System.out.println("Cerebellum acted...");
            // REMOVER
        // Act em ação ***********************************
        try {
            stepBarrierAct.await(); // Sincroniza
            while (stepBarrierAct.getNumberWaiting() <
                        stepBarrierAct.getParties() - 1)
                sleep(TimeWait);
        }
        catch (InterruptedException ex) {
            return;
        }
        catch (BrokenBarrierException ex) {
            return;
            }
            System.out.println("Acting Nodes..."); // REMOVER
        // Emotions parte 2 em ação **********************
        try {
            stepBarrierDecision2.await(); // Sincroniza
            while (stepBarrierDecision.getNumberWaiting() <
                        stepBarrierDecision.getParties() - 1)
                sleep(TimeWait);
        }
        catch (InterruptedException ex) {
            return;
        }
        catch (BrokenBarrierException ex) {
            return;
            }
            System.out.println("Decision 2 acted...");
            // REMOVER

        // Começo do tempo real
            brain.act(Action); // Robô age
        display(true, Cortex.Warn); // Mostra ecrô
        if (Liberu.Robot == 1)
            actRobot(veloc);
        if (Liberu.Robot == 2)
            actNXT(veloc);
        if (Liberu.Robot == 0)
            updateSenses();
        if (Liberu.Robot == 1)
            updateSensesRobot();
        if (Liberu.Robot == 2)
            updateSensesNXT();
        veloc = 1;
        // Fim do passo de vida do robô
            Emotions.WARNING = false;
    }

    public void continueRun() {

        System.out.println("Continue...");
        pause = false;
    }

    public void pause() {
```

```
        System.out.println("Pause...");
        pause = true;
    }

    public void setToEnd() {

      stepBarrierAct.reset();
      stepBarrierDecision.reset();
      stepBarrierDecision2.reset();
      stepBarrierPast.reset();
      stepBarrierCerebellum.reset();
      stepBarrierSleep.reset();
    }

    public void run() {

        if (Liberu.Robot == 2) {
            NXTCommand.open();
            bottomN = new TouchSensor(SensorPort.S1);
            ligthN = new LightSensor(SensorPort.S3);
            ultraSonicN = new UltrasonicSensor(
                        SensorPort.S4);
            soundN = new SoundSensor(SensorPort.S2);

            Motor.A.setSpeed(150); // 900 = 100% power
            Motor.B.setSpeed(150); // Graus por segundo
        }
      if (!OpenData)
            brain.past.bornTime();

        if (Liberu.Robot != 2)
                  bottom = Brain.NO_TACT;
            ligth = 0;
        if (Liberu.Robot == 2)
            ligthN = new LightSensor(SensorPort.S3);
            ultraSonic = vision;
            sound = 0;

        if (Liberu.Robot == 2) {
            Motor.A.setSpeed(150); // 900 = 100% power
            Motor.B.setSpeed(150); // Graus por segundo
        }

            format = new AudioFormat(Encoding.PCM_SIGNED,
                        SAMPLE_RATE, 8, 1, 1, SAMPLE_RATE, true);
      info = new DataLine.Info(TargetDataLine.class,
                  format);
      if (AudioSystem.isLineSupported(
                  Port.Info.MICROPHONE)) {
            try {
                  line = (TargetDataLine)AudioSystem.
                              getLine(info);
            line.open(format);
            data = new byte[line.getBufferSize()/5];
            line.start();
                  System.out.print("microfone...");
                  System.out.println(line.getBufferSize());
            }
            catch (LineUnavailableException ex) {
                  System.out.println("Sem microfone...");
```

```
            }
        }
        else
                System.out.println("Sem microfone......");
        if (AudioSystem.isLineSupported(
                Port.Info.SPEAKER)) {
            try {
                stream = AudioSystem.getAudioInputStream(
                            new File("audiofile1nxt.wav"));
                stream2 = AudioSystem.getAudioInputStream(
                            new File("audiofile2nxt.wav"));
                stream3 = AudioSystem.getAudioInputStream(
                            new File("audiofile3nxt.wav"));
                stream4 = AudioSystem.getAudioInputStream(
                            new File("Rir.wav"));
                stream5 = AudioSystem.getAudioInputStream(
                            new File("Choro.wav"));
            }
            catch (IOException e) {
                e.printStackTrace();
            }
            catch (UnsupportedAudioFileException e) {
                e.printStackTrace();
            }
            AudioFormat format = stream.getFormat();
            info2 = new DataLine.Info(Clip.class,
                        stream.getFormat(),
                    ((int) stream.getFrameLength() *
                            format.getFrameSize()));
            info3 = new DataLine.Info(Clip.class,
                        stream2.getFormat(),
                    ((int) stream2.getFrameLength() *
                            format.getFrameSize()));
            info4 = new DataLine.Info(Clip.class,
                        stream3.getFormat(),
                    ((int) stream3.getFrameLength() *
                            format.getFrameSize()));
            info5 = new DataLine.Info(Clip.class,
                        stream4.getFormat(),
                    ((int) stream4.getFrameLength() *
                            format.getFrameSize()));
            info6 = new DataLine.Info(Clip.class,
                        stream5.getFormat(),
                    ((int) stream5.getFrameLength() *
                            format.getFrameSize()));
            try {
                clip = (Clip) AudioSystem.getLine(info2);
                clip.open(stream);
                clip2 = (Clip) AudioSystem.getLine(info3);
                clip2.open(stream2);
                clip3 = (Clip) AudioSystem.getLine(info4);
                clip3.open(stream3);
                clip4 = (Clip) AudioSystem.getLine(info5);
                clip4.open(stream4);
                clip5 = (Clip) AudioSystem.getLine(info6);
                clip5.open(stream5);
            }
            catch (LineUnavailableException ex) {
            }
            catch (IOException e) {
            }
```

```
            clip.start();
        }
        else
                    System.out.println("Sem Som......");

        stepBarrierPast  = brain.initiatePastTime();
          stepBarrierDecision   = brain.initiateDecision();
          stepBarrierDecision2  = brain.initiateDecision2();
          stepBarrierCerebellum = brain.initiateCerebellum();
          stepBarrierSleep      = brain.initiateSleep();
              stepBarrierAct   = brain.initiateAct();
              try {
              while(!interrupted() && !Neuron.end)
                  if (pause)
                      sleep(200); // Espera pelo Mentor (ms)
                  else {
                    brain.past.startTime();
                      runStepBrain();
                      if (Liberu.Robot == 0)
                          sleep(TimeSlize); // Espera ciclo
                      brain.past.addTime();
                  }
          }
          catch(InterruptedException e) {
          }
          if (Liberu.Robot == 2)
              NXTCommand.close();
          if (Liberu.Robot == 1)
              try {
                  Liberu.input.close();
                  Liberu.output.close();
            }
          catch(IOException e) {
          }
          line.close();
                                          // NXT
          System.out.println("End of Mind Thread...");
      }
}
```

33 “Neuron”

```
/*
 * Neuron.java
 *
 * Created on 4 of August 2004, 21:54
 */

/**
 *
 * @author Paulo Roque Silva
 */
import java.util.*;
import java.io.*;

public class Neuron implements Serializable {
    public static final long serialVersionUID = 2023L;
    // Versão dos dados
    private static final int Nperceptions = 10;
    // perceptions atribuidas por defeito
    static boolean                  end;
    // Se verdadeiro o programa do neurónio é para acabar
    int                             pointer, // atual passo
                                    step; // total de passos dados
    int                                   who, // Identificação própria
                                    quality, // Bom ou mau
                                    finalQuality, // quality total
                                    threshold;
    // Nivel de reação (em percentagem 0-100)
    boolean                         fire, // Neurónio disparou
                                    standby,
    // Para a aprendizagem por sobrevivência no Act
                                    standby2;
    // Para a aprendizagem Out-Wait-In do Sleep
    ArrayList<Signal>         perception;
    // Sinais a que o neurónio ouve ou age
    ArrayList<Signal>         learn; // Sinais para o Sleep

    /** Creates a new instance of Neuron */
    public Neuron(int neuronId, int th) {

            pointer = 0;
        step = 0;
        end = false;
        who = neuronId;
        threshold = th;
        quality = 0;
        finalQuality = 0;
        fire = false;
        standby = false; // Espera por disparo do aleatório
        standby2 = false; // Espera disparo do Mind
        perception = new ArrayList<Signal>(Nperceptions);
        learn = new ArrayList<Signal>(Nperceptions);
    }

    public void addStep() {

        step += 1;
    }
```

```
    public void addPointer() {

        pointer += 1;
    }

    public void setPointer(int i) {

        pointer = i;
    }

    public void addFinalQuality(int q) {

      finalQuality += q;
    }

    public void setFinalQuality(int q) {

      finalQuality = q;
    }

    public void addQuality(int q) {

      quality += q;
    }

    public Signal getPerception(int n) {

      if (n < perception.size())
            return (Signal)perception.get(n);
      else
            if (n == 0)
                  return new Signal(0, 0, 0);
            return new Signal(0, 0, 0);
    }

    // As perceções são de emissores diferentes
    private boolean newPerception(int e) {

        boolean isNew;
        int       i;

        isNew =true;
        for (i=0; i<perception.size(); i++)
            if (getPerception(i).emitter == e)
                isNew = false;
        return isNew;
    }

    // adiciona o signal com emissor <e>, a amplitude <pw>
    // e prazer <plsr>
    public void addPerception(int e, float pw, int plsr) {

        if (newPerception(e))
            perception.add(new Signal(e, pw, plsr));
    }

    public void clearSignals() {
      int i;

      for (i=0; i<perception.size(); i++)
```

```
            getPerception(i).setActive(false);
    }

    public void clearPerceptions() {

        perception.clear();
        perception = new ArrayList<Signal>(Nperceptions);
    }

    public Signal getLearn(int n) {

      return (Signal)learn.get(n);
    }

    public boolean newLearn(int e) {

        boolean isNew;
        int       i;

        isNew =true;
        for (i=0; i<learn.size(); i++)
            if (Math.abs(getLearn(i).emitter) == e)
                isNew = false;
        return isNew;
    }

    // Adiciona sinal com emissor <e>, a amplitude <pw>
    // e prazer <plsr>
    public void addLearn(int e, float pw, int plsr) {

        if (newLearn(e))
            learn.add(new Signal(e, pw, plsr));
    }

    public void clearLearn() {

        learn.clear();
        learn = new ArrayList<Signal>(Nperceptions);
    }

    public void removeLearn(int e) {

        learn.remove(e);
    }

    // Para os neurónios no Emotions
    public int attentionOfFire(int sense) {
        float     activeCredit,
                  betterCredit; //Melhor surpresa
      int         i, rtrn;
      float    x; // Diferença absoluta entre amplitudes

      activeCredit = 0;
      betterCredit = 0;
      rtrn = Cerebellum.ESCape;
        for (i=0; i<perception.size(); i++) {
            if (getPerception(i).active ||
                        getPerception(i).recPower >
                        Brain.MinSense) {
                x = Math.abs(getPerception(i).recPower -
                        getPerception(i).actPower);
```

```
                    if (x > Brain.MaxInSense)
                      x = Brain.MaxInSense;
                    activeCredit = (Brain.MaxInSense - x) *
                           100 / Brain.MaxInSense;
                    if (activeCredit < threshold) {
                      if (sense == Brain.Vision &&
                                 Liberu.Robot == 0)
                            switch (i) {  // Muda a a??o
                            case 1:
                                  if (activeCredit >
                                  betterCredit) {
                                        betterCredit =
                                                     activeCredit;
                                        rtrn = Brain.WALK;
                                  }
                                  break;
                            case 0:
                                  if (activeCredit >
                                  betterCredit) {
                                        betterCredit =
                                                     activeCredit;
                                        rtrn = Brain.TURN_LEFT;
                                  }
                                  break;
                            case 2:
                                  if (activeCredit >
                                  betterCredit) {
                                        betterCredit =
                                                     activeCredit;
                                        rtrn = Brain.TURN_RIGHT;
                                  }
                                  break;
                            default:
                                  System.out.print(
                                        "ERROR Act (actID)...");
                                  rtrn = Cerebellum.ESCape;
                                  break;
                            }
                      if (sense == Brain.Taste &&
                                 Liberu.Robot == 0)
                            switch (i) {  // Muda a a??o
                            case (World.EARTH - 1):
                            break;
                            case (World.WATER - 1):
                                  rtrn = Brain.DRINK;
                            break;
                            case (World.FOOD - 1):
                                  rtrn = Brain.EAT;
                            break;
                            default:
                                  System.out.print(
                                        "ERROR Act (actID)...");
                                  rtrn = Cerebellum.ESCape;
                                  break;
                            }
                    }
              }
        }
        return rtrn;  // = IdAct1
    }
```

```
    // Para os neurónios no Remember
    public float powerOfFire(int sense) {
        float     activeCredit,
                  nActives, // Número de sinais ativos
                  rtrn ;
      int         i; // index
      float   x; // Diferença absoluta entre amplitudes

        nActives = 0;
      activeCredit = 0;
      rtrn = 0;
        for (i=0; i<perception.size(); i++)
            if (getPerception(i).active ||
                        getPerception(i).recPower >
            Brain.MinSense) {
                  // Calcula activeCredit
                  nActives += 1;
                x = Math.abs(getPerception(i).recPower -
                        getPerception(i).actPower);
                if (x > Brain.MaxInSense)
                  x = Brain.MaxInSense;
                activeCredit += (Brain.MaxInSense - x) *
                        100 / Brain.MaxInSense;
            }
      fire = false;
      if (nActives == 0) nActives = 1;
      activeCredit = activeCredit / nActives;
        if (activeCredit >= threshold) {
            fire = true;
            rtrn = activeCredit;
        }
        return rtrn;
    }

// Para os neurónios do Act
    public int powerToAct() {
        int       activeCredit,
                  totalActivos, // Total das percepções
                  credit, // Credito dado
                  i; // index

      activeCredit = 0;
        totalActivos = 0;
        for (i=0; i<perception.size(); i++) {
            if (getPerception(i).active) {
                activeCredit += getPerception(i).recPower;
                totalActivos += 1;
            }
        }
      fire = false;
      if (totalActivos == 0) totalActivos = 1;
       // Média percentual dos ativos
      credit = (int)(100 * activeCredit / totalActivos);
      System.out.println(credit); // RETIRAR
        if (credit >= threshold) {
            fire = true;
            standby = true;
        }
        return credit;
    }
```

```
}
```

34 “Remember”

```
/*
 * Remember.java
 *
 * Created on 3 of February 2011, 15:07
 */

/**
 *
 * @author Paulo Roque Silva
 */
import java.util.Iterator;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.ConcurrentLinkedQueue;

public class Remember implements Runnable {

      public static int            Lovely;
                                                  // Número de disparos
      public static boolean       []NewSense;
                                                  // Se novidade
    Neuron          []time; // Rede por sentiso
    CyclicBarrier   bar; // Barreira de passos
    int                        i; // Indexe
    public static int          RecentMost, // Lembrado recente
                                     threshold; // Limiar
    int                        activeCredit;

    /** Creates a new instance of Remember */
    public Remember(CyclicBarrier barr,
                                  Neuron []tneurons,
                             Emotions d) {

      bar          = barr;
        time         = tneurons;
        NewSense     = new boolean[Brain.SENSES];
        for (i=0; i<Brain.SENSES; i++)
            NewSense[i] = true;
        Lovely       = 0;
      RecentMost = 0;
            threshold = 0;
    }

    public void keepSense(int sense,
            ConcurrentLinkedQueue<Signal> busReal) {
      Iterator<Signal> listen; // Ouve o bus
      Signal                   signInput; // Guarda entrada

      time[sense].clearPerceptions();
      time[sense].quality = 0;
      listen = busReal.iterator();
      while (listen.hasNext()) { // Lê todo o bus
            signInput = listen.next();
            time[sense].addPerception(signInput.emitter,
                        signInput.actPower,
                        signInput.pleasure);
            time[sense].addQuality(signInput.pleasure);
      }
      time[Brain.ProSelf].addFinalQuality(
```

```
                time[sense].quality);
    }

  private int remember(int sense,
          ConcurrentLinkedQueue<Signal> busReal) {
      Iterator<Signal> listen; // Ouve o bus
      Signal                signInput; // Lê bus
      int                   i, active; // Indexe

      time[sense].clearSignals();
    i=0;
    listen = busReal.iterator(); // Lê o primeiro
          while (listen.hasNext()) { // Lê todo o bus
                signInput = listen.next();
          if (time[sense].perception.size() > i) {
                if (signInput.actPower > Brain.MinSense)
                      time[sense].getPerception(i).
                      setActive(true);
                time[sense].getPerception(i).
                setActPower(signInput.actPower);
          }
                i += 1;
          }
          active = (int)time[sense].powerOfFire(sense);
          if (time[sense].fire) {
                NewSense[sense] = false;
                Lovely += 1;
            Interface.Lovely = Lovely;
          }
          return active;
    }

  public void run() {

      while (!Neuron.end && !bar.isBroken()) {
          try {
                bar.await();
                }
          catch (InterruptedException ex) {
                return;
                }
          catch (BrokenBarrierException ex) {
                return;
                }
          if (time[0].pointer == time[0].who) {
                // Neurónio do presente
                time[Brain.ProSelf].finalQuality = 0;
                keepSense(Brain.Taste, Brain.BusInTaste);
                keepSense(Brain.Vision, Brain.BusInVision);
                keepSense(Brain.Tact, Brain.BusInTact);
                keepSense(Brain.Sound, Brain.BusInSound);
                keepSense(Brain.Smell, Brain.BusInSmell);
                keepSense(Brain.ProSelf,
                            Brain.BusInProSelf);
                System.out.println(
                            time[Brain.ProSelf].finalQuality);
                // REMOVER
                } // Fim do If
          if (time[0].pointer-1 == time[0].who &&
                      Emotions.ActualPain) {
                // Prévio neurónio se em Dor
```

```
                    RecentMost = time[0].who;
            }
            // Todos os neurónios do passado
            if ((time[0].pointer-3 != time[0].who &&
                  time[0].pointer-2 != time[0].who &&
                  time[0].pointer-1 != time[0].who &&
                  time[0].pointer   != time[0].who) &&
                        !Emotions.ActualPain) {
                  activeCredit = 0;
                  activeCredit += remember(Brain.Taste,
                              Brain.BusInTaste);
                  activeCredit += remember(Brain.Vision,
                              Brain.BusInVision);
                  activeCredit += remember(Brain.Tact,
                              Brain.BusInTact);
                  activeCredit += remember(Brain.Sound,
                              Brain.BusInSound);
                  activeCredit += remember(Brain.Smell,
                              Brain.BusInSmell);
                  activeCredit += remember(Brain.ProSelf,
                              Brain.BusInProSelf);
                if (activeCredit > threshold) {
                  RecentMost = time[0].who;
                  threshold = activeCredit;
                }
                else
                    if (activeCredit == threshold)
                            if (time[0].who >= RecentMost &&
                        (RecentMost > time[0].pointer ||
                                    time[0].who <=
                                    time[0].pointer))
                              RecentMost = time[0].who;
                        else
                                    if (RecentMost >
                        time[0].pointer && time[0].who <=
                        time[0].pointer)
                                          RecentMost = time[0].who;
            }
      } // Fim ciclo While
    }
}
```

35 “Signal”

```
/*
 * Signal.java
 *
 * Created on 25 of June 2007, 17:14
 */

/**
 *
 * @author Paulo Roque Silva
 */
import java.io.*;

public class Signal implements Serializable {
    public static final long    serialVersionUID = 2023L;
    // Versão dos dados
    int                                 emitter;
    // Identificação do nó que emitiu o sinal
    boolean                             active;
    // Se o sinal está ou não presente
    int                                         pleasure;
    // Positivo se prazer, negativo se desprazer
    float                               recPower,
    // Amplitude do sinal original
                                        actPower;
    // Atual ou real amplitude

    /** Creates a new instance of Signal */
    public Signal(int em, float pwr, int plsr) {

        emitter  = em;
        active   = false;
        pleasure = plsr; // Entrada do prazer
        recPower = pwr; // Nunca muda excepto os nós do Act
                            // pela adaptação de sobrevivência
        actPower = pwr; // A atualizar para um novo valor
    }

    public void setActive(boolean ac) {

        active = ac;
    }

    public void setRecPower(float pw) {

        recPower = pw;
    }

    public void setActPower(float pw) {

        actPower = pw;
    }

    public void addActPower(float pw) {

        actPower += pw;
    }

    public void setPleasure(int pl) {
```

```
        pleasure = pl;
    }
}
```

36 “Sleep”

```
/*
 * Sleep.java
 *
 * Created on 27 of July 2015, 15:57
 */
/**
 *
 * @author Paulo Roque Silva
 */
import java.util.Iterator;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;

public class Sleep implements Runnable {

      Signal                        signInput; // Guarda a entrada bus
    Neuron              [][]time; // Neurónio por sentido
    Iterator<Signal>    listen; // Lê o bus
    CyclicBarrier             sleepBarrier; // Barreira de passos
    int                       me; // O próprio neurónio

    public Sleep(CyclicBarrier barrSono,
            Neuron [][]tneurons, int who) {

      sleepBarrier = barrSono;
      time         = tneurons;
      me = who;
      }

    public void write(int w, float a, int q) {

            Brain.BusSleepOut.add(new Signal(w, a, q));
    }

    public void write2(int w, float a, int q) {

            Brain.BusSleepOut2.add(new Signal(w, a, q));
    }

    public void runSono() {
      int i;

      try {
            sleepBarrier.await();
        }
        catch(InterruptedException e) {
                e.printStackTrace();
            }
        catch (BrokenBarrierException ex) {
            return;
        }
      listen = Brain.BusSleepPreviews.iterator();
      if (listen.hasNext())
            while (listen.hasNext()) { // Lê entrada bus
                  signInput = listen.next();
                  if (!time[me][0].newLearn(
                              Math.abs(signInput.emitter)))
                        // "Ouvindo"
```

```
                        write(me, signInput.actPower,
                                    signInput.pleasure +
                                    time[me][Brain.ProSelf].
                                    finalQuality);
            }
      listen = Brain.BusSleepPreviews2.iterator();
      if (listen.hasNext())
            while (listen.hasNext()) { // L⌂ entrada bus
                  signInput = listen.next();
                  if (signInput.emitter == me) // "Ouvindo"
                  for (i=0; i<time[me][0].learn.size();
                              i++)
                              write2(time[me][0].getLearn(i).
                                          emitter, signInput.actPower,
                                          signInput.pleasure +
                                          time[me][Brain.ProSelf].
                                          finalQuality);
            }
      if (time[me][0].standby2) { // Se em espera
                  time[me][0].standby2 = false;
            time[me][0].addLearn(Math.abs(Cortex.lastExec),
                        0, time[Math.abs(Cortex.lastExec)]
                                    [Brain.ProSelf].finalQuality);
            System.out.print("Cortex.lastExec= "); //REMOVER
            System.out.println(Cortex.lastExec); //REMOVER
      }
    }

    public void run() {

            while(!sleepBarrier.isBroken() && !Neuron.end) {
            runSono();
      }
    }
}
```

37 “World”

```
/*
 * World.java
 *
 * Created on 5 of December 2006, 2:20
 */
/**
 *
 * @author Eng. Paulo Roque Silva
 */
public class World {

    final static int // Orientação do robô
      NORTH = 1,       // Move para Norte - NÃO ALTERAR
        EAST  = 2,       // Move para Este  - NÃO ALTERAR
        SOUTH = 3,       // Move para Norte - NÃO ALTERAR
        WEST  = 4;       // Move para Este  - NÃO ALTERAR
    public static final int // Elementos do Mundo
      EARTH = 1,
        WATER = 2,
        FOOD  = 3;
    // Elementos do mundo
    public static final int
      WALL  = 65;
    // elements[local] retorna comida, água e terra
    int          []elements =
      {EARTH, EARTH, EARTH, EARTH, EARTH, EARTH, WATER,
      EARTH,  EARTH, EARTH, EARTH, EARTH, EARTH, EARTH,
      EARTH,  EARTH, EARTH, EARTH, EARTH, EARTH, EARTH,
      EARTH,  EARTH, EARTH, EARTH, EARTH, EARTH, EARTH,
      EARTH,  EARTH, EARTH, EARTH, EARTH, EARTH, EARTH,
      EARTH,  EARTH, EARTH, EARTH, EARTH, EARTH, EARTH,
      EARTH,  EARTH, EARTH, EARTH, EARTH, EARTH, EARTH,
      EARTH,  EARTH, EARTH, EARTH, EARTH, EARTH, EARTH,
        EARTH, EARTH, EARTH, EARTH,  FOOD,  EARTH, EARTH,
        EARTH};
    // place[local][orientation] retorna o próximo local ou
    // WALL
    // local de 0 a 7; orientation de Norte para Oeste
    // no sentido dos ponteiros do relógio
    int          [][]place = {{WALL,    2,    9, WALL},
                              {WALL,    3, WALL,    1},
                              {WALL,    4, WALL,    2},
                              {WALL, WALL,   12,    3},
                              {WALL,    6,   13, WALL},
                              {WALL, WALL,   14,    5},
                              {WALL, WALL,   15, WALL},
                              {WALL, WALL,   16, WALL},
                              {   1, WALL,   17, WALL},
                              {WALL,   11,   18, WALL},
                              {WALL, WALL, WALL,   10},
                              {   4, WALL,   20, WALL},
                              {   5, WALL, WALL, WALL},
                              {   6,   15,   22, WALL},
                              {   7,   16, WALL,   14},
                              {   8, WALL, WALL,   15},
                              {   9, WALL,   25, WALL},
                              {  10, WALL,   26, WALL},
                              {WALL,   20,   27, WALL},
```

```
                                {  12,   21, WALL,   19},
                                {WALL,   22, WALL,   20},
                                {  14,   23,   30,   21},
                                {WALL,   24, WALL,   22},
                                {WALL, WALL, WALL,   23},
                                {  17,   26, WALL, WALL},
                                {  18, WALL, WALL,   25},
                                {  19, WALL,   35, WALL},
                                {WALL,   29, WALL, WALL},
                                {WALL,   30, WALL,   28},
                                {  22, WALL, WALL,   29},
                                {WALL,   32,   39, WALL},
                                {WALL, WALL, WALL,   31},
                                {WALL,   34, WALL, WALL},
                                {WALL,   35, WALL,   33},
                                {  27,   36,   43,   34},
                                {WALL,   37, WALL,   35},
                                {WALL,   38, WALL,   36},
                                {WALL,   39, WALL,   37},
                                {  31,   40, WALL,   38},
                                {WALL, WALL, WALL,   39},
                                {WALL,   42,   49, WALL},
                                {WALL,   43,   50,   41},
                                {  35,   44, WALL,   42},
                                {WALL,   45, WALL,   43},
                                {WALL,   46,   53,   44},
                                {WALL,   47, WALL,   45},
                                {WALL,   48, WALL,   46},
                                {WALL, WALL, WALL,   47},
                                {  41, WALL, WALL, WALL},
                                {  42, WALL,   58, WALL},
                                {WALL,   52, WALL, WALL},
                                {WALL, WALL,   60,   51},
                                {  45,   54,   61, WALL},
                                {WALL,   55, WALL,   53},
                                {WALL,   56, WALL,   54},
                                {WALL, WALL,   64,   55},
                                {WALL,   58, WALL, WALL},
                                {  50,   59, WALL,   57},
                                {WALL,   60, WALL,   58},
                                {  52, WALL, WALL,   59},
                                {  53, WALL, WALL, WALL},
                                {WALL,   63, WALL, WALL},
                                {WALL,   64, WALL,   62},
                                {  56, WALL, WALL,   63}};
int          [][]place2 = {{WALL,    2,    9, WALL},
          {WALL,    3, WALL,    1},
          {WALL,    4, WALL,    2},
          {WALL, WALL,   12,    3},
          {WALL,    6,   13, WALL},
          {WALL, WALL,   14,    5},
          {WALL, WALL,   15, WALL},
          {WALL, WALL,   16, WALL},
          {   1, WALL,   17, WALL},
          {WALL,   11,   18, WALL},
          {WALL, WALL, WALL,   10},
          {   4, WALL,   20, WALL},
          {   5, WALL, WALL, WALL},
          {   6,   15,   22, WALL},
          {   7,   16, WALL,   14},
          {   8, WALL, WALL,   15},
```

```
            {   9, WALL,   25, WALL},
            {  10, WALL,   26, WALL},
            {WALL,   20,   27, WALL},
            {  12,   21, WALL,   19},
            {WALL,   22, WALL,   20},
            {  14,   23,   30,   21},
            {WALL,   24, WALL,   22},
            {WALL, WALL, WALL,   23},
            {  17,   26, WALL, WALL},
            {  18, WALL, WALL,   25},
            {  19, WALL,   35, WALL},
            {WALL,   29, WALL, WALL},
            {WALL,   30, WALL,   28},
            {  22, WALL, WALL,   29},
            {WALL,   32,   39, WALL},
            {WALL, WALL, WALL,   31},
            {WALL,   34, WALL, WALL},
            {WALL,   35, WALL,   33},
            {  27,   36,   43,   34},
            {WALL,   37, WALL,   35},
            {WALL,   38, WALL,   36},
            {WALL,   39, WALL,   37},
            {  31,   40, WALL,   38},
            {WALL, WALL, WALL,   39},
            {WALL,   42,   49, WALL},
            {WALL,   43,   50,   41},
            {  35,   44, WALL,   42},
            {WALL,   45, WALL,   43},
            {WALL,   46,   53,   44},
            {WALL,   47, WALL,   45},
            {WALL,   48, WALL,   46},
            {WALL, WALL, WALL,   47},
            {  41, WALL, WALL, WALL},
            {  42, WALL,   58, WALL},
            {WALL,   52, WALL, WALL},
            {WALL, WALL,   60,   51},
            {  45,   54,   61, WALL},
            {WALL,   55, WALL,   53},
            {WALL,   56, WALL,   54},
            {WALL, WALL,   64,   55},
            {WALL,   58, WALL, WALL},
            {  50,   59, WALL,   57},
            {WALL,   60, WALL,   58},
            {  52, WALL, WALL,   59},
            {  53, WALL, WALL, WALL},
            {WALL,   63, WALL, WALL},
            {WALL,   64, WALL,   62},
            {  56, WALL, WALL,   63}};
    // orientation[orientationBefore][roboTurn] retorna a
    // nova orientação do robô
    // orientationBefore: de Norte para Oeste - no sentido
    // dos ponteiros do relógio
    // roboTurn: 0 virar esquerda e 1 virar direita
    int         [][]orientation = {{ WEST,  EAST},
                                   {NORTH, SOUTH},
                                   { EAST,  WEST},
                                   {SOUTH, NORTH}};

    /** Creates a new instance of World */
    public World() {
    }
```

```
    // O Liberu move em frente, de <localBefore> na
    // <orientation>, retorna o novo <local> ou WALL
    public int move(int localBefore, int orientation) {

        return place[localBefore-1][orientation-1];
    }

    // Para cada local retorna o elemento
    public int element(int local) {

        return elements[local-1];
    }

    // Para uma orientationBefore e roboTurn retorna a
    // nova orientation
    public int orientate(int orientationBefore, int roboTurn) {

        return orientation[orientationBefore - 1]
                                [roboTurn - Brain.TURN_LEFT];
    }

    // Para um local e orientation retorna o próximo local
    public int vision(int local, int orientation) {

        return move(local, orientation);
    }
}
```

38 Programa em C para Arquitetura Arduíno (Formica)

```
#include <SPI.h>

// Formica
// Created 3 Out 2015 18:50
// by Paulo Roque Silva
// sysliberal@gmail.com
// www.vitaliberu.pt
// This version uses the internal data queing so you can treat it like Serial (kinda)!

#include <math.h>

int s0=14,s1=15,s2=16,s3=17; // port definition of color sensor
int out=8;   // interrupt
int flag=0;  // initialization
int counter=0;
int countR=0,countG=0,countB=0;

  int buz  = 42;

  int led1  = 13;
  int led2  = 7;
  int led3  = 3;

  int motor1  = 11;
  int motor2  = 12;
  int motorF1 = 22;  // Active LOW
  int motorB1 = 23;
  int motorF2 = 24;
  int motorB2 = 25;
  int in;
  int inOld;
  int voz = 8;
```

```
  int pain = 4;
  int temp = A0;
  int distD = A1;
  int distE = A3;
  int sound = A2;
  int batt = A4;
  boolean button;
  int inButton;
  int i; // index
  int strobe = 6; // strobe pins on digital 4
  int res = 5; // reset pins on digital 5
  int audio[7]; // store band values in these arrays
  int band;
  int som;
  int ledvalueR = 0;
  int ledvalueG = 0;
  int ledvalueB = 0;

void setup() {
  // put your setup code here, to run once:
  analogReference(DEFAULT);
  Serial.begin(9600); // init BLE
//  Serial.println("Ola Liberu... Aqui Formica!");
  pinMode(buz, OUTPUT);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(motor1, OUTPUT);
  pinMode(motor2, OUTPUT);
  pinMode(motorF1, OUTPUT);
  pinMode(motorB1, OUTPUT);
  pinMode(motorF2, OUTPUT);
  pinMode(motorB2, OUTPUT);
  pinMode(voz, OUTPUT);
  pinMode(pain, INPUT);
```

```
  pinMode(temp, INPUT);
  pinMode(distD, INPUT);
  pinMode(distE, INPUT);
  pinMode(sound, INPUT);
  pinMode(batt, INPUT);
  in = 0;
  inOld = 0;
  pinMode(res, OUTPUT); // reset
  pinMode(strobe, OUTPUT); // strobe
  digitalWrite(res,LOW); // reset low
  digitalWrite(strobe,HIGH); //pin 5 is RESET on the shield

  // Colors
  pinMode(s0,OUTPUT);
  pinMode(s1,OUTPUT);
  pinMode(s2,OUTPUT);
  pinMode(s3,OUTPUT);
  pinMode(out,INPUT);
  TCS();
  digitalWrite(motorF1, LOW);
  digitalWrite(motorB1, HIGH);
  digitalWrite(motorF2, LOW);
  digitalWrite(motorB2, HIGH);

  analogWrite(led1, LOW);
  analogWrite(led2, LOW);
  analogWrite(led3, LOW);

  digitalWrite(buz,LOW);
}

void TCS()
{
  digitalWrite(s0,HIGH);
  digitalWrite(s1,LOW);
```

```
}

void ISR_INTO()
{
  counter++;
}
int Raverage = 0;
int Baverage = 0;
int Gaverage = 0;

void buzzer() {
  for (int i=0; i<2000; i++) {  // 3000 = 1segundo
    digitalWrite(buz,HIGH);
    delayMicroseconds(166);
    digitalWrite(buz,LOW);
    delayMicroseconds(166);
  }
}

int temC() {
  float raw = analogRead(temp);
  float percent = raw / 1023.0;
  float volts = percent * 5.0;
  return (int)100.0 * volts;
}

int distanceCm(int dist) {
  int rav = analogRead(dist);
  int volt = map(rav, 0, 1023, 0, 5000);
  int cm = (21.61 / (volt-0.1696)) * 1000;
  return cm; // cm
}

void readMSGEQ7()
// Function to read 7 band equalizers
```

```
{
  digitalWrite(res, HIGH);
  digitalWrite(res, LOW);
  for(band=0; band <7; band++)
  {
  digitalWrite(strobe,LOW); // strobe pin on the shield - kicks the IC up to the next band
  delayMicroseconds(30); //
  audio[band] = analogRead(sound); // store left band reading
  digitalWrite(strobe,HIGH);
  }
}

void Tcstrigger()  //   data acquisition
{
      flag++;
    if(flag==1){
      digitalWrite(s2,LOW);
      digitalWrite(s3,LOW);
      countR=counter;
      digitalWrite(s2,HIGH);
      digitalWrite(s3,HIGH);
    }
    else if(flag==2){
      countG=counter;
      digitalWrite(s2,LOW);
      digitalWrite(s3,HIGH);

    }
    else if(flag==3){
      countB=counter;
      digitalWrite(s2,LOW);
      digitalWrite(s3,LOW);
      flag=0;
    }
    counter=0;
```

```
}
void  procedata()        // data processing
{
     static int Rinput[5] ={0,0,0,0,0}
   ,Binput[5] ={0,0,0,0,0}
   ,Ginput[5] ={0,0,0,0,0};
   for(int i = 4;i > 0;i--){
     Rinput[i] = Rinput[i-1];
     Binput[i] = Binput[i-1];
     Ginput[i] = Ginput[i-1];
   }

   if(countR < 2500)
     Rinput[0] = countR;
   else
     Rinput[0] = Rinput[1];
   if(countB < 2500)
     Binput[0] = countB;
   else
     Binput[0] = Binput[1];

   if(countG < 2500)
     Ginput[0] = countG;
   else
    Ginput[0] = Ginput[1];

   Raverage = 0;
   Baverage = 0;
   Gaverage = 0;
   for(int i = 0;i <= 4;i++){
     Raverage += Rinput[i];
     Baverage += Binput[i];
     Gaverage += Ginput[i];
   }
   Raverage /= 5;
```

```
    Baverage /= 5;
    Gaverage /= 5;

}
void  ledup()     // data output
{
    digitalWrite(s2,LOW);
    digitalWrite(s3,LOW);
    ledvalueR = pulseIn(out, LOW);
    Serial.print(ledvalueR,DEC);
    digitalWrite(s2,HIGH);
    digitalWrite(s3,HIGH);
    ledvalueG = pulseIn(out, LOW);
    Serial.print(" ");
    Serial.print(ledvalueG,DEC);
    digitalWrite(s2,LOW);
    digitalWrite(s3,HIGH);
    ledvalueB = pulseIn(out, LOW);
    Serial.print(" ");
    Serial.print(ledvalueB,DEC);
    Serial.print(" ");
    Serial.println();
}

void loop() {
// put your main code here, to run repeatedly:
  in = ' ';
  if (Serial.available() > 0)
    in = Serial.read();
  if (in == 98) // key b
     ledup();
  if (in == 99) // key c
    Serial.println(digitalRead(pain), DEC);
  if (in == 100) // key d
    Serial.println(temC(), DEC);
```

```
  if (in == 101) // key e
    Serial.println(distanceCm(distD), DEC);
  if (in == 102) // key f
    Serial.println(distanceCm(distE), DEC);
  if (in == 103) { // key g
    readMSGEQ7();
    for (band = 0; band < 7; band++) {
      som = map(audio[band],0,1023,0,255);
      Serial.print(som, DEC);
      Serial.print(" ");
    }
    Serial.println();
  }
  if (in == 104) // key h
    analogWrite(led1, HIGH);
  if (in == 105) // key i
    analogWrite(led2, HIGH);
  if (in == 106) // key j
    analogWrite(led3, HIGH);
  if (in == 107) // key k
    analogWrite(led1, LOW);
  if (in == 108) // key l
    analogWrite(led2, LOW);
  if (in == 109) // key m
    analogWrite(led3, LOW);
  if (in == 110) // key n
    buzzer();
  if (in == 111) { // key o WALK AWALK
    digitalWrite(motorF1, LOW);
    digitalWrite(motorB1, HIGH);
    analogWrite(motor1, 75);   // max 50
  }
  if (in == 112) { // key p // BACK ABACK
    digitalWrite(motorF1, HIGH);
    digitalWrite(motorB1, LOW);
```

```
    analogWrite(motor1, 75);   // max 50
  }
  if (in == 113) { // key q
    analogWrite(motor1, LOW);
    analogWrite(motor2, LOW);
  }
  if (in == 115) { // key s LEFT BBACK
    digitalWrite(motorF2, HIGH);
    digitalWrite(motorB2, LOW); // motor2 Back
    analogWrite(motor2, 75);   // max 50
  }
  if (in == 116) { // key t RIGHT BWALK
    digitalWrite(motorF2, LOW); // motor2 Front
    digitalWrite(motorB2, HIGH);
    analogWrite(motor2, 75);   // max 50
  }
  if ((inOld == 111 || inOld == 112) && (in == 115 || in == 116)) {
    delay(1000);
    analogWrite(motor1, LOW);
    analogWrite(motor2, LOW);
  }
  inOld = in;

  delay(1); // ms
}
```

39 BIBLIOGRAFIA

1. Russell, S., Norving, P. Inteligência Artificial. Campus (2004).
2. Ernesto Costa, Anabela Simões. Inteligência Artificial Fundamentos e Aplicações. FCA (2004).
3. Coon, D. Introdução à psicologia: uma jornada. Thomson (2006).
4. Bear, M., Connors, B., Paradiso, M. Neurociencia, Explorando el ccerebro. Masson-Williams & Wilkins (1998).
5. Mackay, W. Neurofisiologia Sem Lágrimas. Fundação Caloustre Gulbenkian (1999).
6. Engelbrech, A. Computational Intelligence, An Introdution. Wiley (2007).
7. Haikonen, P. Robot Brains, Circuits and Systems for Conscious Machines. Wiley (2007).
8. Wilson, H. spikes decisions and actions, dynamical foundations of neuroscience. Oxford (1999).
9. Mass, W., Bishop, C. Pulsed Neural Networks. MIT Press (2001).
10. Nunes, B. Memória, Funcionamento, Perturbações e Treino. Lidel (2008).
11. Damásio, A. O erro de Descartes. Europa - America (1995).
12. Levy, S. Vida Artificial, em demanda de uma nova criação. Publicações Dom Quixote (1994).
13. Sternberg, R. Cognitive Psychology. Thomsons (2006).
14. José Delgado, Carlos Ribeiro. Arquitetura de Computadores. FCA (2007).
15. Katz, B. Neuroengineering the future, Virtual Minds and Creation of Immortality. Infinity Science press (2006).