Documento Síntese IASA - TP1

# Inteligência Artificial

A inteligência artificial (Artificial Intelligence – AI) é a simulação de ações inteligentes demonstradas por máquinas ou agentes inteligentes, e o seu respetivo estudo. Frequentemente Inteligência Artificial é conhecida como a capacidade de uma máquina resolver inteligentemente - ou simulando inteligência, problemas e aprender com essas soluções. A AI, opera sobre uma perspetiva antropomórfica, ou seja, tem a inteligência humana como termo de comparação. A sua aplicação pode-se dividir em duas perspetivas, a analítica e sintética. A primeira, foca-se em ciências empíricas e a segunda em princípios e teorias, unindo-se ambos na exploração de modelos sintéticos - Michael Haenlein definem a inteligência artificial como “uma capacidade do sistema para interpretar corretamente dados externos, aprender a partir desses dados e utilizar essas aprendizagens para atingir objetivos e tarefas específicos através de adaptação flexível”.

O que é a inteligência ao certo? Podemos dizer que a inteligência é um ciclo de perceção, deliberação e ação diante de um problema. Pode ser também definida como a capacidade de aprender, e reagir com as informações obtidas anteriormente.

Comparando “inteligência” de um computador a de um ser humano, podemos dizer que estas diferem dependendo do problema que é exposto. Podemos dizer que o ser humano tem um melhor nível de desempenho que um computador à medida que a complexidade do problema aumenta.

Apos a sua conceção nos anos 50, a inteligência artificial ganhou relevância nos tempos modernos devido ao aumento exponencial do poder computacional dos dispositivos. Atualmente a inteligência artificial é aplicada em diversos ramos desde a exploração espacial onde a IA, realiza o planeamento e escalonamento de rotas a centenas de milhões de quilómetros de distância da Terra, controlo autónomo de veículos, planeamento logístico, etc…

## Principais paradigmas da IA:

- **Simbólico**: A IA trabalha sobre símbolos que representa o mundo natural, símbolos tais como números, letras e bits.

Um sistema de símbolos físico tem os meios suficientes e necessários para a atividade inteligente, em geral. A computação simbólica tornou-se num dos suportes principais de IA. Segundo este paradigma, a inteligência é a soma de processos e estruturas simbólicas, sendo estas processadas por computadores que produzem informações de saída.

- **Conexionista**: A IA tenta simular a inteligência humana, como tal, este paradigma tem como principal inspiração os neurónios do cérebro humano. Implementa as chamadas redes neurais. Os métodos de uso e aprendizagem destas redes, são diversos tais como carros autónomos ou o reconhecimento de face e voz.

- **Evolucionista**: Este paradigma utiliza mecanismos que se baseiam na evolução natural, estes algoritmos chamados algoritmos genéticos têm como principal objetivo encontrar soluções aproximadas para problemas de otimização e busca. Este paradigma é bastante utilizado para encontrar o melhor equilíbrio num novo produto que irá ser lançado pela indústria, tendo por fim um resultado o mais próximo possível do ideal.

## Teste de Turing

O teste de Turing é um teste desenvolvido por Alan Turing. Este tem como objetivo determinar se um computador é inteligente.

O teste consiste numa conversa estabelecida entre um avaliador, um humano, e uma máquina/computador. O computador diz-se inteligente se o avaliador não for capaz de distinguir as respostas dadas pelo humano, das fornecidas pelo computador.

Este teste não é perfeito pelo que desde a sua criação, em 1950 ele tornou-se como uma das principais influencias no campo da IA e respetivos teste, e também, altamente criticado.

## Limites da Inteligência Artificial

A inteligência artificial apesar de extremamente poderosa, possui limitações, sendo a sua maior limitação a diferença inerente que existe entre a inteligência natural e a inteligência artificial.

Uma máquina capaz de simular a inteligência é extremamente cara e de difícil manutenção.

## Vantagens da Inteligência Artificial

Apesar do custo associado a IA tem diversas vantagens tais como a redução de erros humano, inexistência de cansaço e capacidade processar um elevado volume de dados.

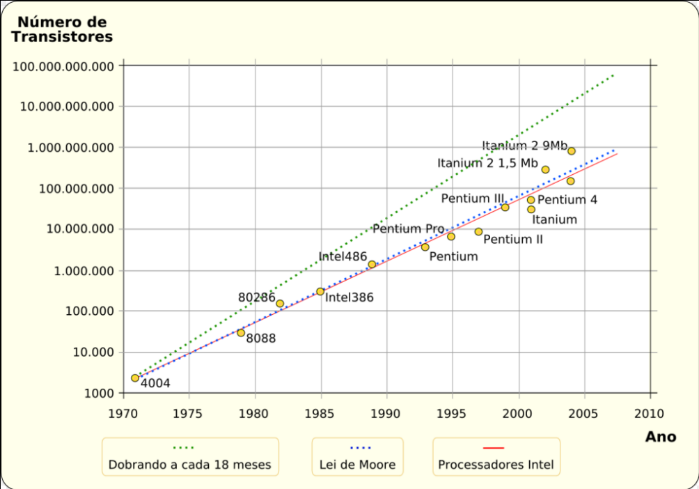
# Engenharia de software

Engenharia de Software, é uma área da informática que aborda o desenvolvimento do Software com uma atividade de engenharia, operação ou mudança/manutenção de Software.

A engenharia de software pretende reduzir a complexidade, aumentar a facilidade de mudança, e reduzir/eliminar a dependência critica.

## Lei de Moore:

A lei de Moore tenta prever a evolução do Hardware presente nos computadores, nomeadamente dos transístores num CPU. Esta lei indica que o número de transístores num CPU terá um aumento de 100% a cada 18 meses, mantendo um custo constante.



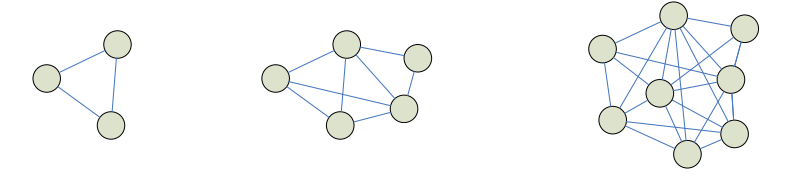
O cumprimento desta lei levou a uma rápida mudança tanto no hardware disponível tal como no software corrido nestas máquinas.

## Complexidade:

Complexidade indica o grau de dificuldade de previsão das propriedades de um sistema dadas as propriedades das partes individuais. Está diretamente relacionado com a informação que é necessária para caracterizar o sistema. A complexidade de um sistema é diretamente proporcional com a quantidade de informação necessária para a descrição. Reflete-se bem no esforço necessário para a geração de uma organização dentro do sistema.

Um sistema com 2 vezes mais partes é muito mais que duas vezes mais complexo.

Um sistema complexo, dificulta a sua manipulação, inclusive a correcao de erros



## Complexidade e Entropia

Entropia mede o grau de dispersão relativa que existe num sistema fechado num dado instante de tempo, diferentes campos têm diferentes definições de entropia.

- **Sistema**: Agrega partes relacionadas entre si, com uma função especifica. Pode ser constituído por subsistemas

- **Estado** (Macro): Configuração global resultante das partes e relações entre partes de um sistema.

- **Microestado**: Configuracao especifica das partes de um sistema

- **Multiplicidade de um estado**: Número de microestados, admissíveis num estado

## **Arquitetura de Software**

### Métricas:

**- Coesão:** Até que ponto um modulo realiza uma única função

Um nível de coesão elevado origina que caso surja a necessidade de se proceder a alterações ao subsistema, o número de módulos afetados é o menor possível. **Um módulo com um nível de coesão baixo é mais complexo, logo mais difícil de conceber e de testar.**

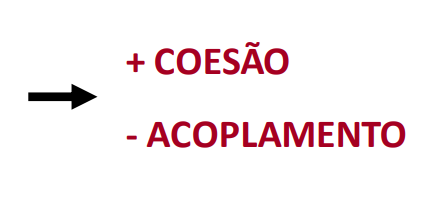
**- Acoplamento:** Grau de dependência entre diversos sistemas e subsistemas

**Uma redução** no acoplamento origina vantagens, tanto na simplicidade da arquitetura tal como na adaptabilidade. Alem de uma **maior facilidade de desenvolvimento, instalação manutenção e expansão.**

**- Simplicidade:** Nível de facilidade de compreensão da arquitetura do modelo

**- Adaptabilidade:** Facilidade com que alterações são implementadas na arquitetura

### Princípios:

**** **- Modularidade:**

**- Decomposição:** Permite sistematizar interações e lidar com a crescente complexidade de um sistema

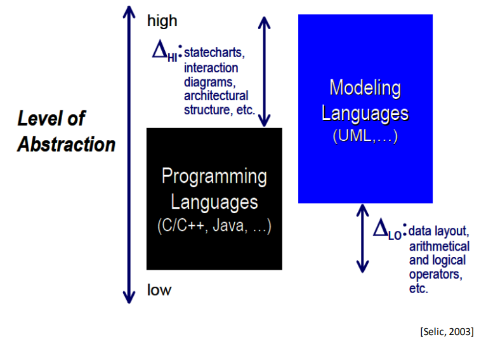
**- Encapsulamento**: Isola os detalhes internos das partes de um sistema em relação ao exterior.

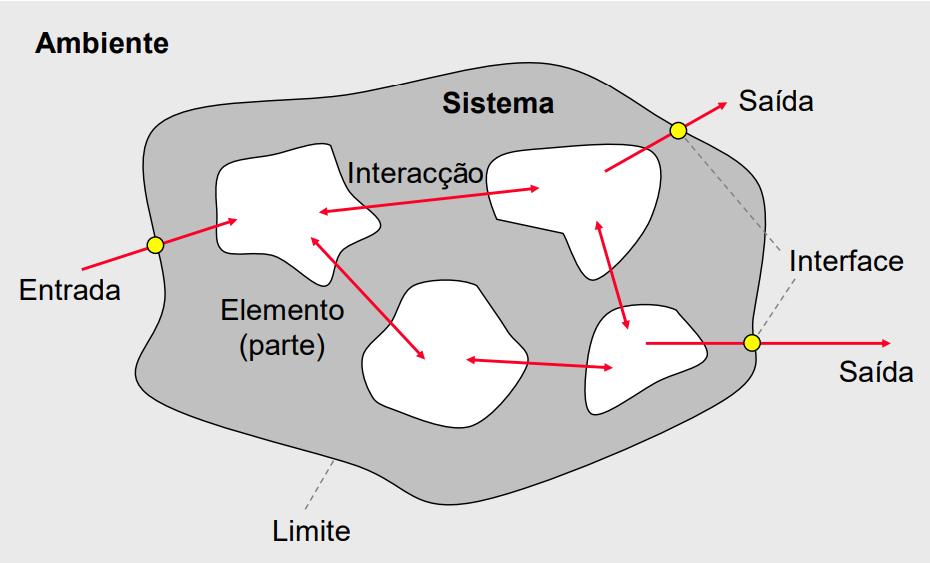
**Cria a noção de interface**. Estas interfaces são contratos funcionais que possibilitam a interação com o exterior.

**- Abstracao: Reduz a complexidade.**

### Modelação:

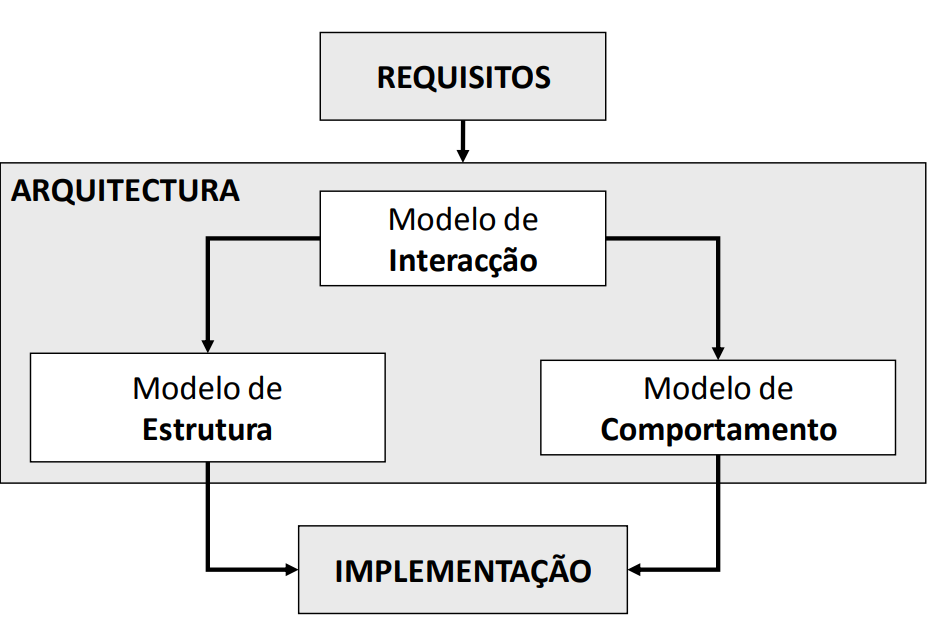
A criação de um modelo visa a simplificação da realidade.





### Processo de desenvolvimento de software

O processo de desenvolvimento é um conjunto de atividades que visa a criação e manutenção de software, legível, e que cumpra os objetivos do programa. **É considerado o principal mecanismo para a criação de software de qualidade**.



Tendo em conta os requisitos do programa, devemos criar um modelo de interação com o exterior. De seguida, tendo em conta este modelo devemos criar dois modelos, um que indique a estrutura do programa, e como estes interagem entre si, e outro modelo comportamental que indique os comportamentos esperados dadas as interações presentes ou passadas recebidas.

# Desenvolvimento de um jogo

Com vista a implementar os conceitos mencionados em cima, foi na proposta a criação de um simples jogo. Este jogo deveria ter um agente minimamente inteligente capaz de agir e reagir em consideração com o ambiente, e os estímulos recebidos até então.

A personagem, ou agente, irá interagir com um jogador humano, tendo como seu único objetivo bloquear o acesso a uma área reservada.

De forma a manter o nível de complexidade o mais baixo possível, seguimos uma arquitetura fornecida pelo docente da cadeira, com classes e métodos bem definidos.

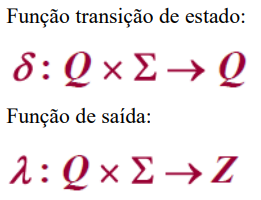
## Simbologia:

Como explicado em cima, existem funções de entrada e saída que regem o comportamento dos agentes. Este jogo não é exceção. Assim temos como função de entrada, um input do jogador humano, possível de serem escolhidos a partir de uma lista previamente comunicada. Cada entrada destas, é representada por um único caracter, que irá corresponder a uma ação que o jogador irá tomar.

Para definir o comportamento da personagem, tivemos em conta, o conjunto de símbolos (alfabeto) possíveis de existirem á entrada. Designado pelo símbolo ∑ e às saídas o alfabeto saída, designado pelo símbolo Z. Assim obtemos a função transformação lambda λ

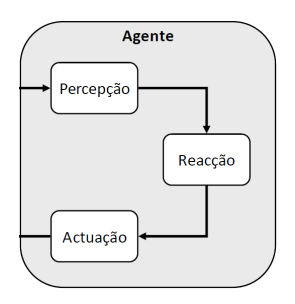


No entanto esta função transformação não é suficiente para a implementação de sistemas com memoria, pelo que é necessária a utilização de uma função que representa transições de estado. Sendo que cada estado tem associado à sua própria função λ.



Antes de explicar as classes que compõe o jogo iremos definir quais as peças fundamentais ao seu funcionamento.

## Pacote Reação



Neste caso o agente reage a estímulos que perceciona do ambiente, gerando uma reação sobre a qual irá agir. Assim para permitir facilidade de manipulação do pacote, e clareza de código, foi criada a Biblioteca Reação. De notar que esta biblioteca não se encontra dentro do pacote de Jogo (explicado mais á frente) pois não é um comportamento unicamente presente neste, podendo ser utilizada e reaproveitada para outras classes.

### Pacote reação base

Por vista a aumentar a modularidade, foi decidido que seria implementado um pacote que serviria para interpretar estímulos, e fazer corresponder uma ação. Este pacote, corresponde a um sistema.

Estímulos:

Um estímulo neste caso, será um evento. Estes eventos serão esporádicos e momentâneos. Como tal pode surgir em qualquer altura. Na representação do estímulo foi criada a interface Estímulo, que declara os métodos a ser implementados.

Acão:

Ação será a atividade que a personagem irá realizar perante o estímulo obtido. A ação é representada na nossa aplicação pela interface Accao.

Reação:

Tendo em conta um estímulo e uma ação consequente, surge então a definição de Reação. Esta última classe corresponde diretamente á função lambda mencionada em cima. Possui o método reagir, que tendo em conta um estímulo, devolve uma ação caso a reação tenha sido criada para reagir a esse estímulo.

#### Comportamento:

O comportamento é a maneira que o agente tem de reagir a um estímulo. Assim esta classe fornece o contrato sobre a qual a classe reação age.

#### Comportamento Hierárquico:

Esta classe é uma implementação hierárquica da interface comportamento. É definida como um conjunto de comportamentos, ordenados pela sua hierarquia, onde a um estímulo irá corresponder a ação que surja primeiro na hierarquia definida.

#### Comportamento Máquina Estados:

A classe ComportMaqEst, surge como uma junção da interface Comportamento, com a classe MaquinaEstados, que irá ser explicada de seguida. Através desta é possível definir comportamentos distintos a ter em cada um dos estados. A classe possui um HashMap que irá corresponder ao estado atual da sua máquina de estados interna, ao conjunto de comportamentos que lhe estão associados.

## Pacote Máquina Estado

Esta biblioteca visa a implementação de uma máquina de estados finita.

Uma máquina de estados, é uma maneira de modelar um problema através de um conjunto de ações sequenciais ou condicionais. Simbolizando a representação das etapas necessárias a resolver um problema.

Um estado descreve um comportamento ou conjunto de ações a serem executadas. Cada estado é indivisível e as transições entre estes são expressões booleanas. Diferentes estados podem, no entanto, ter respostas diferentes a estímulos idênticos (exemplo da reação do agente autónomo á presença de um inimigo, avisando primeiro e só depois atacando, tendo em atenção que o estímulo é idêntico). **A máquina de estados, apenas se encontra num estado em cada instante.**

#### Estado

É a peca fundamental de uma máquina de estados, possui um nome, ações a tomar, e permite reagir a eventos.

Possui um atributo transições, que armazena a ação correspondente ao estímulo recebido, alterando ou não, o estado da máquina de estados, para o estado sucedente.

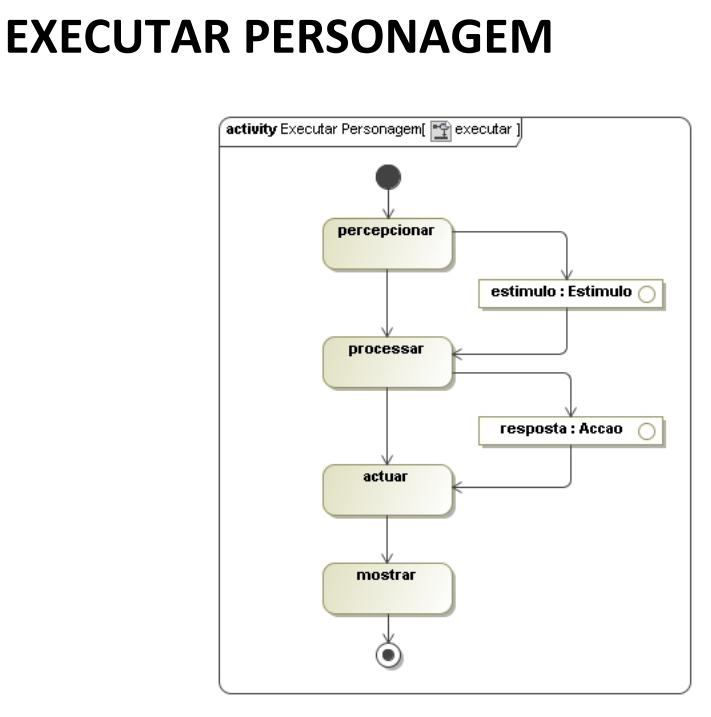
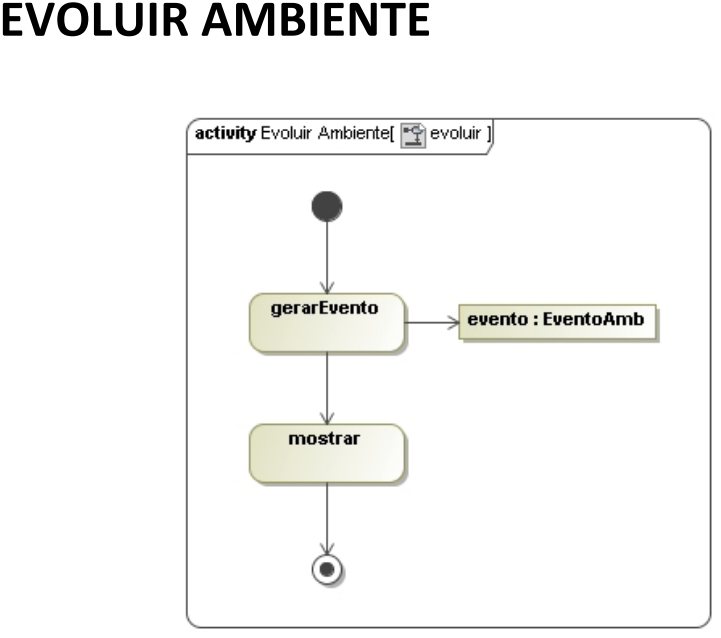
#### MaquinaEstados

É a classe responsável por armazenar todos os estados, tratar os eventos, passando-os ao respetivo estado atual, além de controlar o fluxo destes.

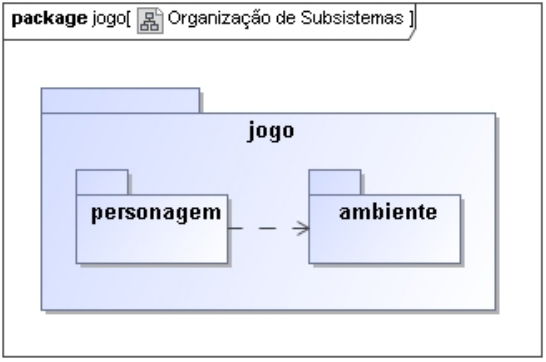
## Pacote Jogo

Adaptado dos slides disponíveis pelo docente da cadeira, o objetivo deste, é “a criação de um ambiente onde a personagem tem por objetivo impedir que inimigos entrem numa zona á sua guarda”.

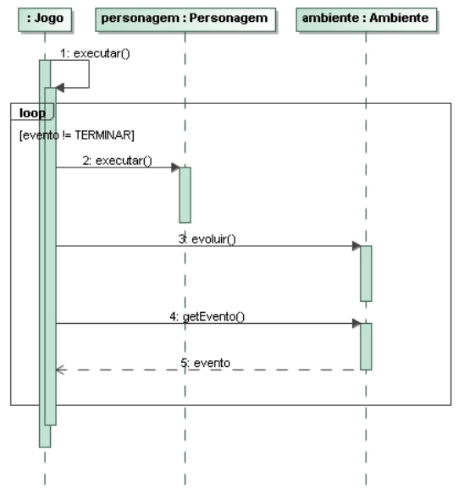
No enunciado a máquina de estados fica definida explicitamente textualmente, sendo posteriormente dividida em partes para facilitar a sua compreensão. Tendo em atenção aos diagramas elaborados pelo professor.



Através da imagem anterior podemos verificar que serão necessárias duas instâncias de máquina de estado, alem de dois pacotes distintos para a realização do projeto. Pelo que a estrutura implementada foi a seguinte.



#### Jogo

Esta classe é o ponto de partida do nosso jogo, onde são definidos todos os elementos que irão interagir, nomeadamente a personagem, e o ambiente. Chama repetidamente, os métodos executar e evoluir do personagem e do ambiente, respetivamente. Implementa o seguinte flowchart.

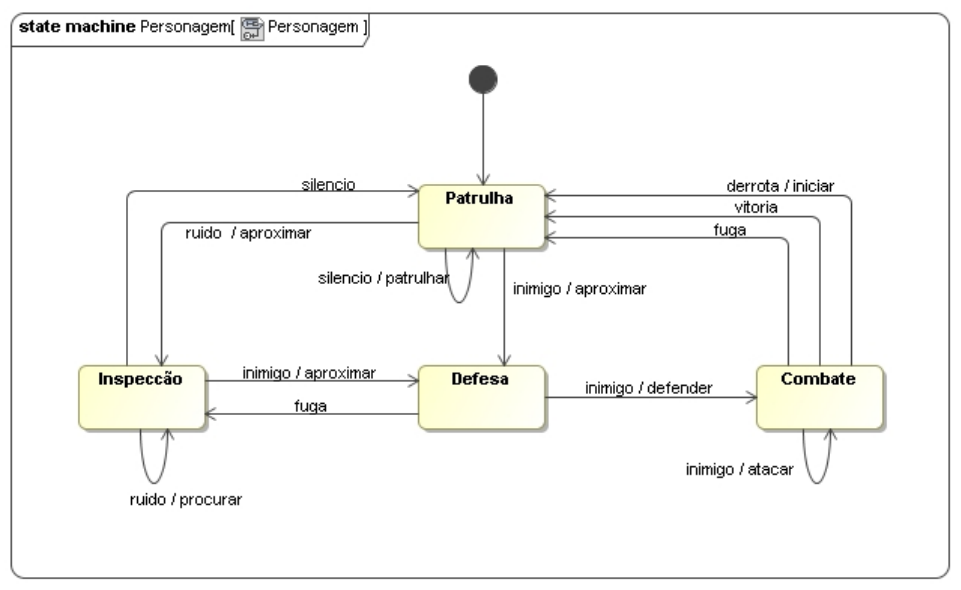
### Pacote Personagem

Neste pacote foram definidos todos os elementos que serão necessários ao funcionamento da personagem. A personagem será responsável por percecionar estímulos e agir em conformidade tendo em conta o estímulo presente e os estímulos passados.

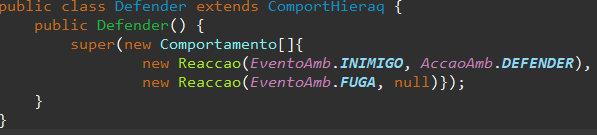
#### Personagem

Esta é a classe responsável por criar a personagem e comunicar com os sistemas externos. Possui métodos para avaliar o ambiente, processar o estimulo recebido e atuar em conformidade.

#### Pacote Comportamentos



Tendo em conta as ações a serem tomadas em cada um dos estados na máquina de estados em cima, o pacote Comportamentos trata de os implementar. Correspondendo a cada estado, um comportamento com as transições adequadas, ordenadas hierarquicamente. Tendo como exemplo a classe Defender podemos ver o resultado da escrita modular do programa até agora, sendo que apenas será necessário escrever poucas linhas de código para atingir o resultado pretendido.



#### Classe ComportPersonagem

A classe ComportPersonagem implementa ComportMaqEst, **cria a máquina de estados da Personagem**, alem disso define os comportamentos, servindo como ponto de comunicação do pacote com a personagem além de definir os estados que esta poderá assumir, e as ações na transição entre eles.