

**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**



**ISEL | DEETC**

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

**Licenciatura em Engenharia  
Informática e Multimédia**

## **Inteligência Artificial para Sistemas Autónomos**

**Semestre de Verão 19/20**

# **Síntese do 1º Trabalho Prático**

**Turma:** 41D

**Data:** 12 maio 2020

**Docente:** Luís Morgado

**Nome:** Luís Fonseca

**Número:** 45125

## Inteligência Artificial – o que é?

Inteligência artificial (ou alcunha para IA) é o campo que estuda a síntese e análise computacionais de maneira inteligente. Este conceito segue 3 principais paradigmas, que são:

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

## Simulador de um jogo com agente reativo

Para este trabalho, foi-nos pedido que fosse desenvolvido e implementado um sistema autónomo inteligente na forma de um jogo, recebendo vários comportamentos, dando a possibilidade ao utilizador, de escolher um dos diferentes comportamentos disponíveis.

## Arquitetura usada

A arquitetura implementada para este trabalho foi uma arquitetura reativa, que consiste em obter uma perceção e através do processamento da mesma, resulta uma ação. Em baixo, é possível de ver a figura que ilustra este modelo:



*Figura 1 - arquitetura usada para a implementação do jogo*

## Personagem

A personagem vai ser tratada em si, como sendo um agente reativo, que age segundo um princípio de **estímulo/ação**. Este agente efetua diversos comportamentos de acordo com uma máquina de estados (que irá ser explicada mais à frente).

No que diz respeito em programação, foram criadas as classes personagem, e os seus diversos comportamentos, que são: Patrulha, Defender, Inspeccionar e Patrulhar. Sendo que a personagem apresenta diversos comportamentos, é possível de ver o código realizado para os comportamentos:

```
    patrulha
        .transicao(EventoAmb.INIMIGO, defesa)
        .transicao(EventoAmb.RUIDO, inspeccao)
        .transicao(EventoAmb.SILENCIO, patrulha);

    inspeccao
        .transicao(EventoAmb.INIMIGO, defesa)
        .transicao(EventoAmb.RUIDO, inspeccao)
        .transicao(EventoAmb.SILENCIO, patrulha);

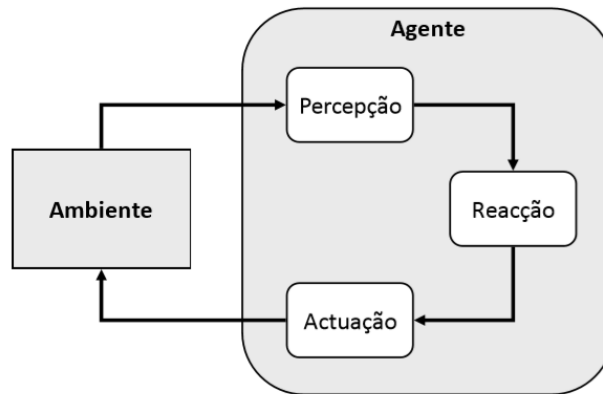
    defesa
        .transicao(EventoAmb.INIMIGO, combate)
        .transicao(EventoAmb.FUGA, inspeccao);

    combate
        .transicao(EventoAmb.INIMIGO, combate)
        .transicao(EventoAmb.FUGA, patrulha)
        .transicao(EventoAmb.VITORIA, patrulha)
        .transicao(EventoAmb.DERROTA, patrulha);
```

## Conceitos da arquitetura reativa:

- ➔ Percepção: contém 3 etapas: a primeira sendo a deteção, a segunda uma discriminação e por último uma codificação.
- ➔ Reação: permite associar um estímulo a uma resposta. Usando o mecanismo de seleção de ação, é escolhida a melhor reação para o nosso agente.
- ➔ Ação: descrição de uma atividade que deve ser realizada num determinado momento;

Na figura de baixo, é possível de ver os diferentes conceitos abordados acima, e como se relacionam na arquitetura reativa:



*Figura 2 - diagrama detalhado da arquitetura reativa*

Em programação Java, foram construídas as classes: Reacao, e Accao, segundo os conceitos abordados em cima. A realização das classes pode se encontrar em baixo:

```
public class Reacao implements Comportamento {

    private Estimulo estimulo;
    private Accao resposta;

    public Reacao(Estimulo estimulo, Accao resposta) {
        this.estimulo = estimulo;
        this.resposta = resposta;
    }

    @Override
    public Accao activar(Estimulo estimulo) {
        return (this.estimulo == estimulo)?this.resposta:null;
    }
}
```

```
public interface Accao {public void executar();}
```

Para efetuar as diversas trocas de ação e reação, foi feita outra classe de nome Comportamento e Comportamento Hierárquico, que ativa um estímulo, consoante uma ação. O código respetivo encontra-se em baixo:

```
public class ComportHierarq implements Comportamento{
    private Comportamento[] comportamentos;
    public ComportHierarq(Comportamento[] comportamentos) {
        this.comportamentos = comportamentos;
    }
    @Override
    public Accao activar(Estimulo estimulo) {
        for(Comportamento comportamento : comportamentos) {
            Accao accao = comportamento.activar(estimulo);
            if(accao != null)
                return accao;
        }
        return null;
    }
}
```

## Estado:

Representa uma situação de um problema. O estado atual representa o estado em que se encontra o agente no instante atual.

Em java, foi criada a classe Estado, que funciona segundo a definição acima definida. Esta classe vai ser tratada como uma classe genérica, que instancia a classe EV, o seu funcionamento consiste em receber tudo, seja desde ao ambiente, até ao comportamento da personagem. O código realizado para esta classe foi:

```
public class Estado<EV>{
    private Map<EV, Estado<EV>> transicoes;
    private String nome;
    public Estado(String nome) {
        this.nome = nome;
        transicoes = new HashMap<EV, Estado<EV>>();
    }
    public String getNome() {
        return this.nome;
    }
    public Estado<EV> transicao(EV evento, Estado<EV> estado) {
        transicoes.put(evento, estado);
        return this
    }

    public Estado<EV> processar(EV evento) {
        return transicoes.get(evento);
    }
    public String toString() {
        return getNome();
    }
}
```

## Máquina de Estado:

Esta máquina de estados contém 7 tipos de estímulos e 7 tipos de ações. Em cada um dos estados, o agente encontra-se à espera de um estímulo.

Em java foi criada a classe MaquinaEstados, que executa a definição acima. O código respetivo pode ser visto em baixo:

```
public class MaquinaEstados<EV>{
    private Estado<EV> estado;
    public MaquinaEstados(Estado<EV> estado) {
        this.estado = estado;
    }
    public Estado<EV> getEstado() {
        return this.estado;
    }
    public void processar(EV evento) {
        Estado<EV> novoEstado = estado.processar(evento);
        if(novoEstado != null)
            estado = novoEstado;
    }
}
```

## Ambiente:

Representa a base sobre o qual o jogo assenta e é controlado pelo utilizador indicando o evento que quer que aconteça.

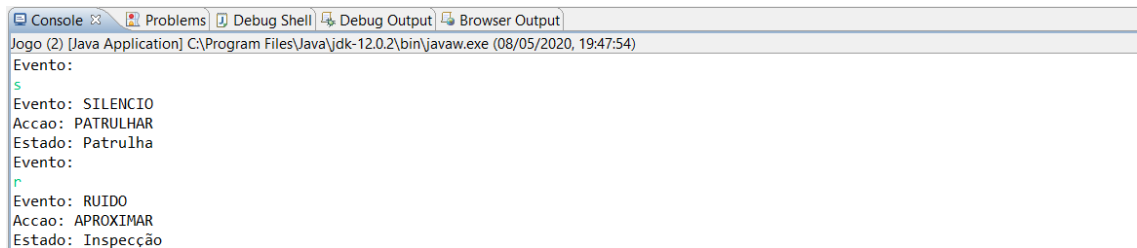
É através da classe Ambiente que é possível fazer esta apresentação ao utilizador, o código essencial para ver as diferentes ações encontra-se em baixo:

```
private EventoAmb gerarEvento() {
    System.out.println("Evento: ");
    String input = sc.next();
    switch(input) {
        case "s":
            return EventoAmb.SILENCIO;
        case "r":
            return EventoAmb.RUIDO;
        case "i":
            return EventoAmb.INIMIGO;
        case "f":
            return EventoAmb.FUGA;
        case "v":
            return EventoAmb.VITORIA;
        case "d":
            return EventoAmb.DERROTA;
        case "t":
            return EventoAmb.TERMINAR;
        case "e":
            System.out.println("O jogo terminou!");
            System.exit(0);
        default:
            System.out.println("Não introduziu um caracter válido.");
            return null;
    }
}
```

Para comprovar o correto funcionamento, foi criada a classe Jogo. O que esta classe faz é começar o jogo, fornecendo um agente, apresentando o respetivo evento à sua escolha. O código desta classe encontra-se em baixo:

```
private static Ambiente ambiente = new Ambiente();
private static Personagem personagem = new Personagem(ambiente);
private static void executar() {
    do {
        personagem.executar();
        ambiente.evolver();
    }while(ambiente.getEvento() != EventoAmb.TERMINAR);
}
public static void main(String[] args) {
    executar();
}
```

De seguida, é executado na consola, em modo texto, todo o trabalho realizado (apenas será apresentado alguns comandos, sendo que o resto das ações funciona corretamente):



```
Console
Jogo (2) [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\javaw.exe (08/05/2020, 19:47:54)
Evento:
S
Evento: SILENCIO
Accao: PATRULHAR
Estado: Patrulha
Evento:
R
Evento: RUÍDO
Accao: APROXIMAR
Estado: Inspeção
```

*Figura 3 - output previsto do primeiro trabalho prático*

## Bibliografia:

- ➔ Slides fornecidos pelo docente Luís Morgado.

## Diagramas de classes:

Na pasta “Mod” é possível encontrar os diagramas criados para este trabalho prático.