

Centro Universitário de Belo Horizonte – UniBH

Departamento de Ciências Exatas e Tecnologia

Ciência da Computação

Inteligência Artificial

Professor Felipe Leandro Andrade da Conceição

**Experimento 1 – Implementação de um Agente Dirigido por Tabela**

Rodrigo Reis

Igor Bueloni

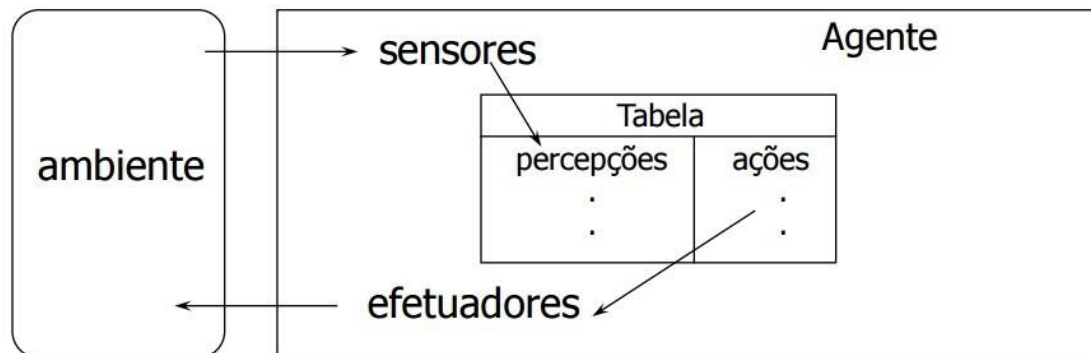
Wendell Ronald

Ivan Paulovich

Março – 2017

## I. Introdução

O trabalho proposto pelo professor Felipe Leandro Andrade da Conceição, consiste em implementar um Agente Dirigido por Tabela (um agente que mapeia um ambiente e consulta uma tabela para definir qual ação a ser realizada)



Esta implementação deve ser feita em três atividades: (A). Elaborar um agente dirigido por tabela sem limite de percepções; (B). Elaborar um agente dirigido por tabela utilizando o máximo de 6 (seis) percepções; (C) O agente elaborado pela atividade A deve enfrentar o agente elaborado pela atividade B.

## II. Desenvolvimento do Experimento

Para o desenvolvimento deste experimento, os agentes serão robôs e o ambiente será uma arena na qual estes se enfrentarão, buscando suas ações baseadas na percepção deste ambiente. Para isso foi utilizado o software *Robocode*<sup>1</sup> versão *Robot*, assim foram desenvolvidos dois robôs AURA (Apenas Um Robô Autônomo) e AURAL (Apenas Um Robô Autônomo Limitado); AURA é um Agente Dirigido por Tabela contendo sete percepções, enquanto AURAL é uma extensão de AURA porém este Agente Dirigido por Tabela contém apenas três percepções.

### II.I. Arquitetura

---

<sup>1</sup> *Robocode* é um jogo de competição entre robôs virtuais. Do qual utiliza-se da linguagem Java para implementar métodos e eventos das ações que o robô deverá tomar.

A arquitetura para estes agentes consiste em uma classe para cada implementação de robôs, uma enumeração para as percepções e uma classe para o mapeamento de PERCEPÇÃO/AÇÃO.

O enumerador de percepção contém sete percepções e serve apenas como chave para a classe de mapeamento PERCEPÇÃO/AÇÃO.

A classe de mapeamento PERCEPÇÃO/AÇÃO, funciona como um contrato entre a percepção e a ação, ou seja, define o que deve ser feito dada uma percepção.

A classe Agente (AURA/AURAL) contém uma lista (tabela) de mapeamento (Percepção/Ação); esta tabela é preenchida uma única vez no método construtor de cada agente, o diferencial é que no AURA a tabela é preenchida com todas as sete percepções definidas para o experimento, enquanto no AURAL é preenchida apenas com três.

Existe também um método que executa uma ação dada uma percepção como parâmetro, esta ação a ser executada é um método escrito na própria classe, além disso, como foi utilizado o software *Robocode* e ambos os agentes estendem a classe *AdvancedRobot*, estes possuem uma série de funções e eventos relacionados ao escopo robô, assim, os eventos de *AdvancedRobot* foram utilizados como os sensores do agente.

### II.I.I. Percepções

Foram mapeadas sete percepções/ações conforme a tabela abaixo:

Percepção	Ação
Varredura do Inimigo	Procura pelo inimigo no ambiente, utilizando o radar do robô.
Inimigo Detectado	Ataca o inimigo; Avança sobre o inimigo e através de sua posição mira o canhão para o ângulo do mesmo e efetua um disparo.

Atingido pelo Inimigo	Para, e retrocede 30 unidades; inicia a ação de “Varredura do Inimigo” novamente.
Colidir no Limite da Arena	Muda de direção virando 90º; inicia a ação de “Varredura do Inimigo” novamente.
Pouca Energia	Muda a cor do robô para amarelo.
Inimigo Derrotado	Executa uma comemoração.
Derrotado	Executa uma lamentação.

## II.II. Código-Fonte

Foi criado um repositório para o experimento no GitHub no seguinte URL: <https://github.com/rodrigossor/AgenteDirigidoTabela.Experimento>, permitindo que todo código-fonte possa ser baixado e estudado.

Percepcao.java

```
package Unibh;

public enum Percepcao {
    VarreduraDoInimigo,
    InimigoDetectado,
    AtingidoPeloInimigo,
    ColidirLimiteArena,
    PoucaEnergia,
    InimigoDerrotado,
    Derrotado
}
```

Mapeamento.java

```
package Unibh;

public class Mapeamento {

    private final Percepcao percepcao;
    private final String acao;
```

```
public Percepcao getPercepcao() {
    return this.percepcao;
}

public String getAcao() {
    return this.acao;
}

public Mapeamento(Percepcao percepcao, String acao) {
    this.percepcao = percepcao;
    this.acao = acao;
}
}
```

### Aura.java

```
package Unibh;

import java.awt.Color;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import robocode.AdvancedRobot;
import robocode.DeathEvent;
import robocode.HitByBulletEvent;
import robocode.HitWallEvent;
import robocode.RobotStatus;
import robocode.ScannedRobotEvent;
import robocode.StatusEvent;
import robocode.WinEvent;

/**
 * AURA - Apenas Um Robô Autônomo
 *
 * @version 1.0.0
 * @author Rodrigo Reis, Igor Bueloni, Wendell Ronald, Ivan Paulovich
 */
public class Aura extends AdvancedRobot {

    protected List<Mapeamento> tabela;
    protected int fator;

    public Aura() throws NoSuchMethodException {
        tabela = new ArrayList();
        tabela.add(new Mapeamento(Percepcao.VarreduraDoInimigo, "Procurar"));
    }
}
```

```

        tabela.add(new Mapeamento(Percepcao.InimigoDetectado, "Atacar"));
        tabela.add(new Mapeamento(Percepcao.AtingidoPeloInimigo, "Fugir"));
        tabela.add(new Mapeamento(Percepcao.ColidirLimiteArena, "MudarDirecao"));
        tabela.add(new Mapeamento(Percepcao.PoucaEnergia, "Amarelar"));
        tabela.add(new Mapeamento(Percepcao.InimigoDerrotado, "ExecutarDancaVitoria"));
        tabela.add(new Mapeamento(Percepcao.Derrotado, "Chorar"));
        fator = 255;
    }

    public void ExecutarAcao(Percepcao percepcao, Object parametro) {
        try {
            Optional<Mapeamento> mapeamento = tabela.stream().filter(m -> m.getPercepcao() ==
            percepcao).findFirst();
            if (parametro != null) {
                Method acao = this.getClass().getMethod(mapeamento.get().getAcao(),
            parametro.getClass());
                acao.invoke(this, parametro);
            } else {
                Method acao = this.getClass().getMethod(mapeamento.get().getAcao());
                acao.invoke(this);
            }
        } catch (IllegalAccessException | IllegalArgumentException | NoSuchMethodException |
            SecurityException | InvocationTargetException ex) {
            Logar("Percepcao -> Acao nao mapeada. [%s]", ex.getMessage());
        }
    }

    public void Escurecer() {
        fator -= 20;
        if (fator < 0) {
            fator = 255;
        }
        setColors(new Color(fator, 0, 0), new Color(fator, 0, 0), new Color(fator, 0, 0));
    }

    public void Logar(String format, Object... args) {
        System.out.println(String.format(format, args));
    }

    public void Procurar() {
        Logar(">> VarreduraDoInimigo");
        setAdjustRadarForGunTurn(true);
        setAdjustRadarForRobotTurn(true);
        setAdjustGunForRobotTurn(true);
        setTurnRadarRightRadians(Double.POSITIVE_INFINITY);
    }

    public void Atacar(ScannedRobotEvent e) {
        Logar(">> InimigoDetectado");
    }

```

```
double angulo = getHeading() - getGunHeading() + e.getBearing();
setTurnGunRight(angulo);
setTurnRight(angulo);
setAhead(30);
fire(1);
}

public void Fugir() {
    Logar(">> AtingidoPeloInimigo");
    stop();
    setBack(300);
    Escurecer();
    Procurar();
}

public void MudarDirecao() {
    Logar(">> ColidirLimiteArena");
    stop();
    setTurnRight(90);
    Procurar();
}

public void Amarelar(RobotStatus e) {
    if (e.getEnergy() < 20) {
        Logar(">> PoucaEnergia");
        setColors(Color.yellow, Color.yellow, Color.yellow);
    }
}

public void ExecutarDancaVitoria() {
    Logar(">> InimigoDerrotado");
    for (int i = 0; i < 50; i++) {
        stop();
        setTurnRight(30);
        setTurnLeft(30);
    }
}

public void Chorar() {
    Logar(">> Derrotado");
    Logar("(T_T) -> https://lidianta.files.wordpress.com/2014/09/engole1.png");
}

@Override
public void run() {
    setColors(Color.red, Color.red, Color.red);
    ExecutarAcao(Percepcao.VarreduraDoInimigo, null);
}
```

```
@Override
public void onScannedRobot(ScannedRobotEvent e) {
    ExecutarAcao(Percepcao.InimigoDetectado, e);
}

@Override
public void onHitByBullet(HitByBulletEvent e) {
    ExecutarAcao(Percepcao.AtingidoPeloInimigo, null);
}

@Override
public void onHitWall(HitWallEvent e) {
    ExecutarAcao(Percepcao.ColidirLimiteArena, null);
}

@Override
public void onStatus(StatusEvent e) {
    ExecutarAcao(Percepcao.PoucaEnergia, e.getStatus());
}

@Override
public void onWin(WinEvent e) {
    ExecutarAcao(Percepcao.InimigoDerrotado, null);
}

@Override
public void onDeath(DeathEvent e) {
    ExecutarAcao(Percepcao.Derrotado, null);
}
}
```

Aural.java

```
package Unibh;

import java.awt.Color;
import java.util.ArrayList;
import robocode.ScannedRobotEvent;
import robocode.StatusEvent;

/**
 * AURAL - Apenas Um Robô Autônomo Limitado
 *
 * @version 1.0.0
 * @author Rodrigo Reis, Igor Bueloni, Wendell Ronald, Ivan Paulovich
 */
public class Aural extends Aura {
```



```

public Aural() throws NoSuchMethodException {
    tabela = new ArrayList();
    tabela.add(new Mapeamento(Percepcao.VarreduraDoInimigo, "Procurar"));
    tabela.add(new Mapeamento(Percepcao.InimigoDetectado, "Atacar"));
    tabela.add(new Mapeamento(Percepcao.PoucaEnergia, "Amarelar"));
    fator = 255;
}

@Override
public void Escurecer() {
    fator -= 20;
    if (fator < 0) {
        fator = 255;
    }
    setColors(new Color(0, fator, 0), new Color(0, fator, 0), new Color(0, fator, 0));
}

@Override
public void run() {
    setColors(Color.green, Color.green, Color.green);
    ExecutarAcao(Percepcao.VarreduraDoInimigo, null);
}

@Override
public void onScannedRobot(ScannedRobotEvent e) {
    ExecutarAcao(Percepcao.InimigoDetectado, e);
}

@Override
public void onStatus(StatusEvent e) {
    ExecutarAcao(Percepcao.PoucaEnergia, e.getStatus());
}
}

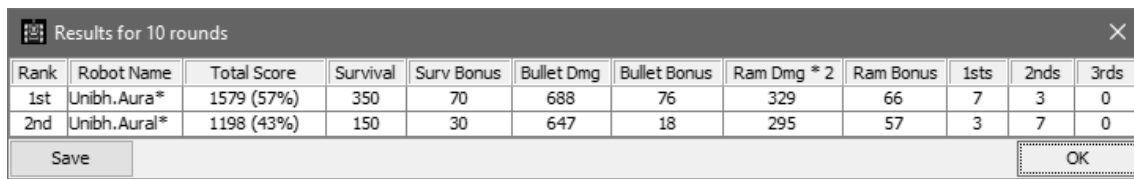
```

### III. Conclusão

A conclusão vista no experimento mostra claramente que quanto mais percepções existem na tabela, mais inteligente o agente se torna, conseguindo tomar decisões mais assertivas sobre o ambiente, o que neste experimento garantiu sua superioridade (Figura 1).

Entretanto, o maior número de ações na tabela torna o processamento das ações do agente mais demoradas, (mesmo que por milissegundos) assim como o consumo de memória, neste ambiente simulado do experimento, estes contratempos não são

facilmente percebidos e se tornam irrelevantes, porém em larga escala pode apresentar defeitos ou desperdícios.



Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	Unibh.Aura*	1579 (57%)	350	70	688	76	329	66	7	3	0
2nd	Unibh.Aural*	1198 (43%)	150	30	647	18	295	57	3	7	0

Figura 1 Resultado de 10 turnos de batalha entre AURA e AURAL. Vantagem de 42,85% para AURA, comprovando a conclusão do experimento.

## IV. Referências

GASPAR, Willian Rozin. TT3 TUTORIAL DE ROBOCODE. DCC - UDESC Joinville,

[http://d3m21rn3ib0riu.cloudfront.net/PAT/Upload/1910069/tutorialrobocode\\_20170317134838.pdf](http://d3m21rn3ib0riu.cloudfront.net/PAT/Upload/1910069/tutorialrobocode_20170317134838.pdf), 2012

ZAMBIASI, Saulo Popov. Robocode – Eventos. UFSC. <http://www.gsigma.ufsc.br/~popov/aulas/robocode/eventos.html>

ZAMBIASI, Saulo Popov. Robocode – Funções. UFSC. <http://www.gsigma.ufsc.br/~popov/aulas/robocode/funcoes.html>

ROBOCODE COMMUNITIES, <http://robowiki.net>