UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Implementação de um Agente Inteligente

Raimundo Osvaldo Vieira
Ponta Grossa, março de 2020

O conceito de agente inteligente é fundamental para a Inteligência Artificial, sendo utilizado para a definição de um conjunto de princípios que guiam o desenvolvimento de sistemas que possam ser denominados inteligentes (RUSSELL; NORVIG, 2013). Pode-se dizer que um agente é uma entidade que interage com o ambiente em que está inserido, adquirindo percepções desse ambiente por meio de sensores e agindo (ou atuando) nele por meio de atuadores. Este trabalho está fundamentado nesse conceito, apresentado por Russell e Norvig (2013), dado que se trata de uma definição clássica amplamente aceita mundialmente. A Figura 1 ilustra essa ideia e destaca como os agentes interagem com o ambiente.

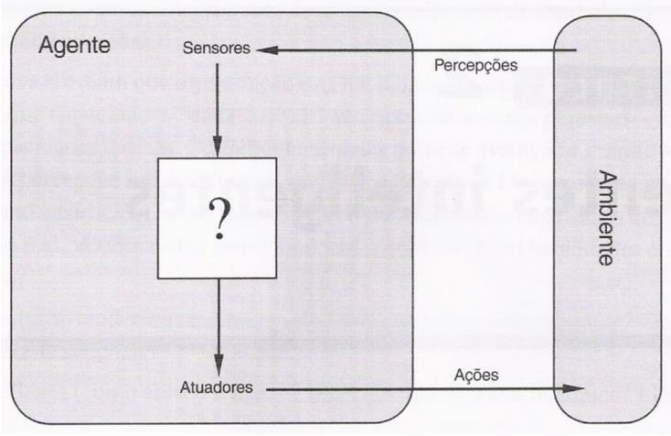

Figura 1 Interação de um agente inteligente com o ambiente
Fonte: (RUSSELL; NORVIG, 2013)

Os sensores captam as percepções do ambiente, que representam as entradas do agente em determinado momento. Uma questão fundamental é como o agente poderá selecionar as ações adequadas, dadas as percepções. Essa questão é ilustrada na Figura 1 pelo quadrado branco indicado pelo sinal de interrogação (?). Trata-se da função do agente, isto é, o mapeamento de uma percepção em uma ação específica. Tal função deve,

ainda, ser implementada em um programa do agente para que possa ser executada. Boa parte do esforço consiste na correta definição dessa função.

O projeto de um agente requer a descrição de um ambiente de tarefas, ou seja, a especificação do ambiente com que o agente interagirá, seus sensores e atuadores, e uma medida de desempenho, que define o quão corretamente o agente executou suas tarefas. Esses itens de descrição do ambiente de tarefas são comumente conhecidos como PEAS (*Performance, Environment, Actuators, Sensors*). Na descrição do ambiente de tarefas, é necessário, ainda, caracterizá-lo conforme as seguintes propriedades: 1) completamente observável ou parcialmente observável; 2) agente único ou multiagente; 3) determinístico ou estocástico; 4) episódico ou sequencial; 5) estático ou dinâmico; 6) discreto ou contínuo; 7) conhecido ou desconhecido.

Outro aspecto importante no projeto de uma agente é descrever seu comportamento interno. Neste caso, o trabalho consiste em desenvolver o programa do agente, isto é, implementar a função do agente. Conforme proposto por Russel e Norvig (2013), a estrutura básica de um programa agente consiste em cada agente receber a percepção atual por meio dos sensores e produzem uma ação para os atuadores. Agentes que necessitam de uma sequência de percepções devem armazenar as percepções. Além disso, é necessária a utilização de alguns componentes para que o programa possa produzir comportamento racional. Diante disso, pode-se categorizar os agentes como: 1) agentes reativos simples; 2) agentes reativos baseados em modelo; 3) agentes baseados em objetivos; 4) agentes baseados na utilidade. É possível, ainda, acrescentar a essa lista os agentes com inteligência.

O propósito deste trabalho é apresentar uma solução para o problema proposto na disciplina de Inteligência Artificial do Programa de Pós-Graduação em Ciência da Computação da UTFPR, Câmpus Ponta Grossa. Tal problema consiste na simulação de um agente robô que seja capaz de navegar uma área desconhecida e determine a maior e menor temperatura detectadas. A seguir são apresentados detalhes da solução do problema, de acordo com os conceitos acima apresentados.

1. ESPECIFICAÇÃO DO PEAS E TIPO DE AGENTE IMPLEMENTADO

No planejamento da solução, definiu-se que o ambiente a ser explorado pelo agente consiste em um espaço em formato de um quadrado, delimitado por paredes, tendo em seu interior obstáculos dispostos aleatoriamente dentro da área delimitada pelas paredes. Os obstáculos são intransponíveis e sempre que o robô detectar um obstáculo

deverá desviar-se dele. Nos locais para livre circulação do agente robô estará disponível, a cada metro quadrado, um valor aleatório de temperatura, que poderá ser captado por um dos sensores do agente. Foram projetados dois sensores: um sensor de temperatura, capaz de ler a temperatura do local em que o agente robô está localizado, e um sensor de obstáculos que permite ao agente detectar se à sua frente existe uma parede ou um outro tipo de obstáculo. Os atuadores definidos para o agente consistem em um componente para mudar sua posição, avançando um metro à frente nas direções norte, sul, leste ou oeste, e outro componente que permite ao agente mudar sua direção. As mudanças de direção sempre ocorrem da seguinte forma: norte $\rightarrow$ sul; sul $\rightarrow$ oeste; oeste $\rightarrow$ leste; e leste $\rightarrow$ norte. A fim de avaliar a atividade do agente, definiu-se como medida de desempenho a taxa de cobertura do agente, que consiste na porcentagem do território do ambiente que o agente foi capaz de percorrer. Além disso, o agente deve ser capaz de informar qual a menor e a maior valor temperatura detectados durante o processo de varredura do espaço.

O agente implementado é do tipo reativo com estado, visto que ele responde às percepções atuais e guarda em memória os locais já percorridos.

A seguir são apresentados os detalhes de implementação.

2. IMPLEMENTAÇÃO

A implementação foi feita utilizando-se o paradigma da orientação a objetos e, para isso, foi utilizada a linguagem de programação Java. Foram definidos três pacotes: ambiente, agente e principal. O pacote principal contém a classe `Main` que corresponde à interface da aplicação, onde são parametrizadas as dimensões do ambiente e a quantidade de obstáculos, além de ser acionada a execução do agente. Na Figura 2 é apresentado o diagrama de classes e nas subseções a seguir serão detalhadas as implementações do ambiente e do agente robô.

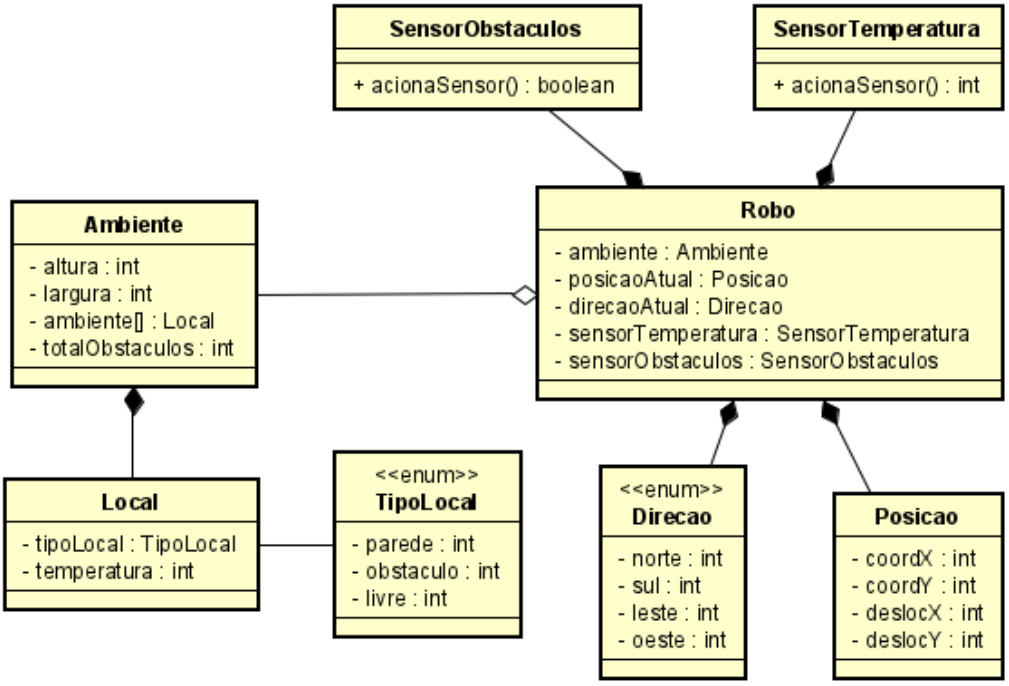

Figura 2 Diagrama de Classes - implementação do ambiente e agente robô
Fonte: o autor

2.1. O ambiente

A classe `Ambiente` descreve o ambiente com o qual o agente interage. Ele possui as dimensões `altura` e `largura` que, para efeito deste trabalho, terão sempre os mesmos valores, visto que se considera o ambiente como uma matriz quadrada. Além disso, definiu-se para o ambiente o `totalDeObstaculos` e uma matriz do tipo `Local`, que representa a superfície na qual o robô se deslocará. Parte do código da classe é mostrado a seguir e outros detalhes são apresentados mais à frente.

```
public class Ambiente {

	private int altura, largura;
	private final Local[][] ambiente;
	private int totalDeObstaculos;

	public Ambiente(int altura, int largura, int totalDeObstaculos) {
		this.altura = altura;
		this.largura = largura;
		this.totalDeObstaculos = totalDeObstaculos;
		this.ambiente = new Local[altura][largura];
		this.montaAmbiente(totalDeObstaculos);
	}
}
```

A classe `Local`, define que um local possui um tipo, definido pela enumeração `TipoLocal`, e um valor de temperatura. Detalha-se aqui que os tipos de locais definidos são: parede, obstáculo e livre. Apenas o tipo livre terá o valor de temperatura definido, conforme se pode ver no código do construtor da classe, cuja parte do código é mostrado a seguir:

```
public class Local {

	private TipoLocal tipoLocal;
	private int temperatura;

	public Local(TipoLocal tipo) {
		this.tipoLocal = tipo;

		if(tipoLocal == TipoLocal.livre) {
			Random random = new Random();
			this.temperatura = (random.nextInt(301))-100;
		}
	}
}
```

Os principais comportamentos definidos na classe Ambiente foram `montaAmbiente()` e `getEstadoAtual()`, cujas implementações são mostradas a seguir:

```
private void montaAmbiente(int totalDeObstaculos) {

   do {
      Random random = new Random();
      int x = random.nextInt(this.altura);
      int y = random.nextInt(this.largura);
      if (this.ambiente[x][y] == null && !isParede(x, y)) {
         this.ambiente[x][y] = new Local(TipoLocal.obstaculo);
            totalDeObstaculos--;
      }
   } while (totalDeObstaculos > 0);

   for (int x = 0; x < this.altura; x++) {
      for (int y = 0; y < this.largura; y++) {
         if (isParede(x, y)) {
            this.ambiente[x][y] = new Local(TipoLocal.parede);
         } else if (!isObstaculo(x, y)) {
            this.ambiente[x][y] = new Local(TipoLocal.livre);
         }
      }
   }
}
```

`montaAmbiente()` é um método privado da classe `Ambiente` que é invocado sempre que o construtor é executado. O propósito desse método é definir um formato inicial para o ambiente, inserindo em posições aleatórias tantos obstáculos quantos forem definidos por parâmetro no construtor e, em seguida, preenchendo as demais posições com locais do tipo livre e paredes. Observe que as paredes definem as bordas do ambiente e, por isso, são alocadas nas posições correspondentes às primeiras linhas e colunas da matriz de locais.

Além dos métodos de acesso e modificação adequados, foram definidos métodos auxiliares na classe, cujas assinaturas são listadas a seguir:

```
public boolean isEspacoLivre(int x, int y);

private boolean isParede(int x, int y);

private boolean isObstaculo(int x, int y);
```

O método `getEstadoAtual()` cria uma representação imprimível do estado atual do ambiente, indicando as paredes, obstáculos e a posição atual do robô.

```
public String getEstadoAtual(Robo robo) {

   StringBuilder ambiente = new StringBuilder();

   for (int x = 0; x < this.altura; x++) {
      for (int y = 0; y < this.largura; y++) {
         if (isEspacoLivre(x, y)) {
            if (x == robo.getPosicaoX() && y == robo.getPosicaoY()) {
               ambiente.append(" & ");
            } else {
               ambiente.append(this.ambiente[x][y].getTipoLocal());
            }
         } else if (isParede(x, y)) {
            ambiente.append(this.ambiente[x][y].getTipoLocal());
         } else {
            ambiente.append(this.ambiente[x][y].getTipoLocal());
         }
      }
      ambiente.append("\n");
   }
   return ambiente.toString();
}
```

Na subseção seguinte é detalhada a implementação do agente robô.

2.2. O agente robô

Para melhor entendimento da classe `Robo`, faz-se necessário descrever inicialmente as seguintes classes:

A classe `Posicao` é uma abstração do conceito de posição do agente no ambiente, sendo defina por duas coordenadas, `coordX` e `coordY`, e por uma medida de deslocamento na direção: `deslocX` e `deslocY`. As coordenadas são associadas aos índices da matriz de locais e as medidas de deslocamento indicam, de acordo com a direção atual do agente, a qual coordenada será adicionada ou diminuída uma unidade, a fim de que o agente se mova um metro para frente na sua direção atual (método `setDeslocamento()`). Essa classe também disponibiliza métodos que indicam os valores das coordenadas referentes à próxima posição. Esse método é importante para simular o movimento do robô e para o correto funcionamento do sensor de obstáculos. O código dessa classe é apresentado a seguir:

```
public class Posicao {

	private int coordX, coordY;
	private int deslocX, deslocY;

	public Posicao() {

	}

	public Posicao(int x, int y) {
		this.coordX = x;
		this.coordY = y;
	}

	public void setCoordenada(int x, int y) {
		this.coordX = x;
		this.coordY = y;
	}

	public void setDeslocamento(Direcao direcao) {
		switch (direcao) {
		case norte:
			this.deslocX = 0;
			this.deslocY = -1;
			break;
		case sul:
			this.deslocX = 0;
			this.deslocY = 1;
```

```
                break;
            case oeste:
                this.deslocX = -1;
                this.deslocY = 0;
                break;
            case leste:
                this.deslocX = 1;
                this.deslocY = 0;
                break;
            default:
                break;
            }
        }

        public int getCoordX() {
            return this.coordX;
        }

        public int getCoordY() {
            return this.coordY;
        }
        public int getDeslocX() {
            return this.deslocX;
        }

        public int getDeslocY() {
            return this.deslocY;
        }

        public int getProxCoordX() {
            return (this.coordX + this.deslocX);
        }

        public int getProxCoordY() {
            return (this.coordY + this.deslocY);
        }
}
```

A classe `SensorObstaculo` encapsula o conceito do sensor de obstáculos do agente. É definido apenas um método, `acionaSensor()`, que recebe a referência de um local do ambiente, retorna verdadeiro caso o local seja do tipo parede ou obstáculo e falso, caso contrário. Optou-se por acumular nesse sensor as funcionalidades de detectar obstáculos e paredes, dado que uma parede pode ser considerada um obstáculo. A seguir é apresentado o código da classe.

```
public class SensorObstaculos{

	public boolean acionaSensor(Local local) {
		if(local.getTipoLocal() == TipoLocal.obstaculo
				|| local.getTipoLocal() == TipoLocal.parede) {
			return true;
		}
		return false;
	}
}
```

A classe `SensorTemperatura` encapsula o conceito do sensor de temperatura do agente. Nela, é definido apenas um método, `acionaSensor()`, que recebe a referência de um local do ambiente e retorna o valor de temperatura daquele local, caso este seja do tipo livre. A seguir é apresentado o código da classe.

```
public class SensorTemperatura {

	public int acionaSensor(Local espaco) {
		return espaco.getTemperatura();
	}
}
```

A classe `Robo` descreve a abstração do agente robô desenvolvida como parte da solução do problema proposto. O agente robô guarda conhecimento de sua posição e direção atuais, além do histórico de locais visitados, de uma contagem dos locais não visitados e do registro da maior e menor temperatura detectada ao longo do trajeto que é percorrido. Foram definidos dois sensores para o agente robô: um sensor de obstáculos, definido na classe `SensorObstaculos`, e um sensor de temperatura, implementado na classe `SensorTemperatura`. A seguir é apresentado parte do código da classe, destacando a implementação do construtor.

```
public class Robo {

	private Ambiente ambiente;
	private Posicao posicaoAtual;
	private Direcao direcaoAtual;
	private SensorTemperatura sensorTemperatura;
	private SensorObstaculos sensorObstaculos;
	private HashSet<Posicao> locaisVisitados;
	private int maiorTemperatura, menorTemperatura;
	private int numLocaisNaoVisitados;
	private int contaTentativaDeMudarPosicao;
```

```
    public Robo(Ambiente ambiente) {
        this.ambiente = ambiente;
        this.posicaoAtual = new Posicao();
        this.sensorTemperatura = new SensorTemperatura();
        this.sensorObstaculos = new SensorObstaculos();
        this.setEstadoInicial();
        this.locaisVisitados = new HashSet<>();
        this.numLocaisNaoVisitados = (this.ambiente.getLargura() - 2)
                * (this.ambiente.getAltura() - 2)
                - this.ambiente.getTotalDeObstaculos();
    }
}
```

Destaca-se do código do construtor a chamada ao método privado `setEstadoInicial()`, que tem por propósito definir uma posição aleatória para o robô na superfície a ser explorada, bem como definir sua direção inicial, também de maneira aleatória, e acionar o sensor de temperatura para registrar a temperatura da posição inicial do robô. A implementação desse método é apresentada abaixo.

```
private void setEstadoInicial() {

   Random random = new Random();
   int x;
   int y;

   do {
      x = random.nextInt(this.ambiente.getLargura());
      y = random.nextInt(this.ambiente.getAltura());
   } while (!this.ambiente.isEspacoLivre(x, y));

   this.setPosicao(x, y);
   int direcaoInicial = random.nextInt(4);

   switch (direcaoInicial) {
      case 0:
         setDirecao(Direcao.norte);
         break;
      case 1:
         setDirecao(Direcao.sul);
         break;
      case 2:
         setDirecao(Direcao.leste);
         break;
      case 3:
         setDirecao(Direcao.oeste);
```

```
            break;
         default:
            break;
      }

      this.posicaoAtual.setDeslocamento(this.direcaoAtual);
      this.maiorTemperatura = this.menorTemperatura =
                this.sensorTemperatura.acionaSensor(this.getLocalAtual());
}
```

O método `atual()` corresponde à implementação dos atuadores do agente robô e seu funcionamento consistem em: 1) marcar o local como visitado, adicionando num set de locais; 2) checar a temperatura do local e verificar se esta temperatura é a mais alta ou mais baixa do que temperaturas anteriormente verificadas. Isto é feito por meio de um método privado `verificaTemperaturaLocal()`. Este método aciona o `sensorDeTemperatura` e faz a verificação e, se necessário, o registro do valor da temperatura local; 3) ajustar o registro de locais não visitados. Este dado é usado no cálculo da taxa de cobertura do agente; 4) fazer o robô se deslocar uma posição à frente, na sua direção atual, atualizando a nova posição. Quando este cenário não pode ser executado porque o robô tem à sua frente um obstáculo ou uma parede, entra em ação o atuado de mudança de direção, implementado no método `mudaDirecao()`, que consistem em alterar a direção do agente, a partir do conhecimento da posição atual. Caso, a nova direção também indique ao robô uma posição nas mesmas condições, o processo é repetido a partir de uma chamada recursiva da função. Caso as quatro direções na posição atual indiquem locais cujo tipo seja obstáculo ou parede ou locais já visitados, o agente finaliza seu trabalho de varredura do ambiente. A seguir são apresentados os códigos dos métodos `atua()`, `verificaTemperaturaLocal()` e `mudaDirecao()`.

```
public void atua() {

   if (this.contaTentativaDeMudarPosicao >= 4) {
      return;
   }

   if (this.sensorObstaculos.acionaSensor(this.getProximoLocal())
      || isLocalVisitado(this.posicaoAtual.getProxCoordX(),
        this.posicaoAtual.getProxCoordY())) {

      this.mudaDirecao();
      this.contaTentativaDeMudarPosicao++;
      this.atua();
```

```
    } else {
       this.locaisVisitados.add(new Posicao(this.getPosicaoX(),
                                                this.getPosicaoY()));
       this.verificaTemperaturaLocal();
       this.numLocaisNaoVisitados--;
       this.setPosicao(this.posicaoAtual.getProxCoordX(),
                                      this.posicaoAtual.getProxCoordY());
    }
}
```

```
private void verificaTemperaturaLocal() {
    int temperatura =
              this.sensorTemperatura.acionaSensor(this.getLocalAtual());
    if (temperatura >= this.maiorTemperatura) {
       this.maiorTemperatura = temperatura;
    }


    if (temperatura <= this.menorTemperatura) {
       this.menorTemperatura = temperatura;
    }
}
```

```
private void mudaDirecao() {

    switch (this.direcaoAtual) {
       case sul:
          this.direcaoAtual = Direcao.oeste;
          break;
       case leste:
          this.direcaoAtual = Direcao.norte;
          break;
       case norte:
          this.direcaoAtual = Direcao.sul;
          break;
       case oeste:
          this.direcaoAtual = Direcao.leste;
          break;
       }

      this.posicaoAtual.setDeslocamento(direcaoAtual);
}
```

O método `obterDesempenho()` calcula a taxa de cobertura do agente robô. Essa taxa é a porcentagem de locais visitados em relação ao todos os locais livre do ambiente.

Além disso, esse método retorna uma versão imprimível da medida de desempenho do agente, contendo sua taxa de cobertura e os valores da menor e da maior temperatura detectadas nos locais percorridos pelo agente. Destaca-se que o intervalo considerado de temperatura é de -200 a 200, diferente do que foi proposto no problema. Fez-se essa alteração nos requisitos pela maior facilidade em trabalhar com intervalo de extremidades simétricas e por se considerar um fator de menor relevância mediante o objetivo do trabalho. O código do método é mostrado abaixo.

```
public String obterDesempenho() {

   NumberFormat percent = NumberFormat.getPercentInstance();
   StringBuilder desempenho = new StringBuilder("Desempenho do Agente");

   double taxaDeCobertura = (double) this.locaisVisitados.size()
           / (this.locaisVisitados.size() + this.numLocaisNaoVisitados);

   return desempenho.append("Locais Visitados: " +
                this.locaisVisitados.size()).append("\n")
                .append("Locais não visitados: " +
                this.numLocaisNaoVisitados).append("\n")
                .append("Taxa de Cobertura: " +
                percent.format(taxaDeCobertura)).append("\n")
                .append("Menor Temperatura: " +
                this.menorTemperatura).append("\n")
                .append("Maior Tempertadura: " +
                this.maiorTemperatura)
                .toString();
}
```

Além desses métodos e dos métodos de acesso e modificação, foram implementados métodos auxiliares, cujas assinaturas são listadas a seguir.

```
public boolean isTrabalhando();

public void zeraTentativaDeMudarPosicao();

private boolean isLocalVisitado(int x, int y);

private Local getProximoLocal();

private Local getLocalAtual();
```

3. CONSIDERAÇÕES FINAIS

A execução do agente robô, após sua criação e definição do seu estado e do estado inicial do ambiente é basicamente controlada pelo seguinte trecho de código.

```
while (roboEmMarte.isTrabalhando()) {
   System.out.println(superficieDeMarte.getEstadoAtual(roboEmMarte));
   roboEmMarte.zeraTentativaDeMudarPosicao();
   roboEmMarte.atua();
   Thread.sleep(1500);
}
System.out.println(roboEmMarte.obterDesempenho());
```

O método `isTrabalhando()` verifica se todos os locais foram visitados ou o robô chegou a um local de onde não consegue sair, o que é descrito adiante. Enquanto o robô consegue se movimentar no ambiente, o programa exibe o estado atual do ambiente e do robô, indicando a posição em que o robô está e, em seguida invoca o método atua(), que é responsável por gerenciar todo o comportamento do robô.

Conforme já explicitado, o agente não passa pelo menos local mais de uma vez. Isto é uma fragilidade da implementação proposta, visto que tal condição impede que o agente percorra todo o espaço disponível em alguns cenários. Um cenário, neste caso, corresponde ao estado inicial do ambiente determinado pela disposição dos obstáculos e posição inicial do agente. Por exemplo, na Figura 3 tem-se um exemplo de um destes cenários, onde “X” simboliza uma parede, “O” representa um obstáculo e “&” indica o agente posicionado no ambiente.

```
X X X X X X
X   O     X
X         X
X O O     X
X O O &   X
X X X X X X
```

Figura 3 Estado inicial do ambiente e do agente
Fonte: o autor

Considerando que o agente tenha sua direção inicial definida como leste, temos a seguinte sequência de deslocamento do agente (Figura 4). Na situação ilustrada, a ação do agente cessa, pois o local a oeste contém um obstáculo e os demais locais possíveis já foram visitados pelo agente. Neste caso, a taxa de cobertura do agente não será total, pois alguns locais não serão visitados.

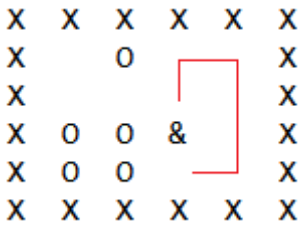

Figura 4 Estado final do ambiente e do agente
Fonte: o autor

Fez-se a opção por planejar a movimentação do agente desta forma para evitar que o agente caia em laços infinitos ou que demore muito tempo para percorrer todas os locais disponível. Os resultados permitem compreender o mecanismo do agente e, dependendo da configuração inicial do ambiente, podem ser alcançados bons resultados.

De modo geral, a solução apresentada, embora apresente as limitações exposta acima, contempla as etapas do processo de projeto de um agente inteligente e apresenta os elementos básicos de um agente reativo com estado, tendo contribuído para o processo de aprendizagem dos conceitos e aplicação desses conceitos numa atividade prática.

Referências

RUSSEL, Stuart; NORVIG, Peter. **Inteligência Artificial**. Rio de Janeiro: Elsevier, 2013.