

Agente Limpeza em Labirinto

[Descrição](#) | [Objetivo](#) | [Estrutura do Projeto](#) | [Requisitos](#) | [Como usar](#) | [Configuração](#) | [Licença](#) |

Info

Professor: Dr. Aldo Henrique Dias Mendes

Participantes: Lucas silva, Luan medrado

Disciplina: Inteligencia artificial

Semestre: 2024/2

Descrição:

Este projeto implementa um sistema de agente autônomo que navega e limpa um labirinto gerado aleatoriamente. O agente percorre o labirinto identificando e removendo sujeiras representadas por células específicas do ambiente.

Objetivos

O agente localizar a sujeita A(agua) e P(po) e limpa dando prioridade para Agua

Estrutura do Projeto

O projeto está organizado da seguinte forma:

- Main.java: Classe principal que executa o programa, inicializa o labirinto e controla o fluxo de execução.
- Labirinto.java: Classe que representa o ambiente do labirinto, com a geração aleatória de células contendo sujeira e obstáculos
- AgenteLabirinto.java: Classe que implementa o agente que navega pelo labirinto, tomando decisões sobre o movimento e a limpeza.
- PosicaoXY.java: Classe que define a posição do agente e outros objetos dentro do labirinto.

► PEAS (Performance, Environment, Actuators, Sensors)

Performance (Desempenho):

- O desempenho do agente é avaliado com base em sua capacidade de limpar todas as células do labirinto que contenham sujeira. Um agente bem-sucedido será aquele que limpar todas as células com sujeira ("A" para água e "P" para poeira) no menor número de movimentos possível e sem deixar sujeiras para trás.

– Environment (Ambiente):

- O ambiente é o labirinto, que é gerado aleatoriamente com um número fixo de células (por exemplo, 4x4). Cada célula do labirinto pode conter:
 - Água ("A"), representando sujeira líquida.
 - Poeira ("P"), representando sujeira sólida.
 - Células vazias ("L") que já foram limpas pelo agente.
- O agente deve navegar neste ambiente, evitando obstáculos e encontrando as células sujas.

– • Actuators (Atuadores):

- O agente usa os seguintes atuadores:
 - **Movimentação:** O agente se move para as células adjacentes nas quatro direções possíveis (cima, baixo, esquerda, direita).
 - **Limpeza:** Quando o agente chega em uma célula suja, ele usa o atuador de limpeza para remover a sujeira, transformando a célula em limpa ("L").

• Sensors (Sensores):

- O agente possui sensores que permitem:
 - **Percepção de Sujeira:** O agente pode detectar se a célula atual ou uma célula adjacente contém sujeira, seja ela água ("A") ou poeira ("P").
 - **Percepção de Obstáculos:** O agente pode detectar obstáculos, como paredes ou células já limpas, para evitar mover-se para locais inválidos.



1. Clone o repositório:
 - o `Bash git clone https://github.com/Lucca7r/`
2. Instale as dependências:
 - o `none`

⚙️ Exemplo de Execução

- Ao iniciar, o labirinto será exibido com as células representando água ("A") e poeira ("P"). O agente se moverá e limpará as células, mostrando visualmente seu progresso até completar a tarefa de limpeza.

© Licença:

Este repositório está licenciado sob a licença MIT.

PARTES DE CADA ALUNO:

LUAN RESPONSÁVEL PELO LABIRINTO

LUCAS RESPONSÁVEL PELO AGENTE BUSCA SUJEIRA

CODIGO FONTE:

-MAIN

```
--MAIN

import agente.AgenteLabirinto;
import ambiente.Labirinto;
import geral.PosicaoXY;

public class Main{
```

```

    public static void main(String[] args) throws
InterruptedException {

        Labirinto labirinto = new Labirinto(4);
        labirinto.exibirLabirinto();

        AgenteLabirinto agente = new AgenteLabirinto(labirinto);
        agente.setPosicao(new PosicaoXY(2,2));
        labirinto.setAgente(agente);

        while(agente.isAindaLimpando()) {
            agente.zerarPilha();
            agente.movimentar();
            labirinto.exibirLabirinto();
            Thread.sleep(1500);
        }
        return;
    }
}

```

--LABIRINTO

```

package ambiente;

import agente.AgenteLabirinto;
import geral.PosicaoXY;
import java.util.Random;

public class Labirinto {

    private int tamanhoLabirinto;
    private String[][] labirinto;
    private AgenteLabirinto agente;
    private Random random;

    public Labirinto(int tamanhoLabirinto) {
        this.tamanhoLabirinto = tamanhoLabirinto;
        this.random = new Random();
        this.construirNovoLabirinto();
    }
}

```

```

private void construirNovoLabirinto() {
    labirinto = new
String[this.tamanhoLabirinto][this.tamanhoLabirinto];
    for (int i = 0; i < this.tamanhoLabirinto; i++) {
        for (int j = 0; j < this.tamanhoLabirinto; j++) {
            if (random.nextBoolean()) {
                this.Labirinto[i][j] = "P";
            } else {
                this.Labirinto[i][j] = "A";
            }
        }
    }
}

public void exibirLabirinto() {
    atualizarPosicaoAgente();
    for (int i = 0; i < tamanhoLabirinto; i++) {
        for (int j = 0; j < tamanhoLabirinto; j++) {
            if (labirinto[i][j].equals("*A*")) {
                System.out.print("|" + labirinto[i][j] + "|");
            } else {
                System.out.print("| " + labirinto[i][j] + " |");
            }
        }
        System.out.println("");
    }
    System.out.println("");
}

private void atualizarPosicaoAgente() {
    if (this.agente != null) {
        PosicaoXY posAgente = this.agente.getPosicao();
        if
(!labirinto[posAgente.getPosX()][posAgente.getPosY()].equals("L")) {
            labirinto[posAgente.getPosX()][posAgente.getPosY()] =
"*A*";
        }
    }
}

public int getTamanhoLabirinto() {
    return this.tamanhoLabirinto;
}

public String retornarValorPosicaoLabirinto(PosicaoXY posicao) {
    return this.Labirinto[posicao.getPosX()][posicao.getPosY()];
}

public void setAgente(AgenteLabirinto agente) {

```

```

        this.agente = agente;
    }

    public void limpar() {
        if (this.agente != null) {
            PosicaoXY posicao = this.agente.getPosicao();
            labirinto[posicao.getPosX()][posicao.getPosY()] = "L";
        } else {
            throw new IllegalStateException("Agente não configurado no
labirinto.");
        }
    }
}

```

--AGENTELABIRINTO

```

package agente;

import ambiente.Labirinto;
import geral.PosicaoXY;

public class AgenteLabirinto {

    private Labirinto labirinto;
    private PosicaoXY posXY;
    private MovimentosAgenteLabirinto movimento;
    private int pilhaMovimentos;

    public AgenteLabirinto(Labirinto labirinto) {
        this.labirinto = labirinto;
        this.posXY = new PosicaoXY(0, 0);
        this.movimento = MovimentosAgenteLabirinto.CIMA;
        this.pilhaMovimentos = 0;
    }

    public void setPosicao(PosicaoXY posicao) {
        this.posXY = posicao;
    }

    public void movimentar() {

```

```

        boolean existeAgua = existeAguaNoLabirinto();

        PosicaoXY proximaAgua = buscarSujeiraProxima("A");

        if (existeAgua && proximaAgua != null) {
            this.posXY = proximaAgua;
            this.labirinto.Limpar();
            this.pilhaMovimentos++;
        }

        if (!existeAgua) {
            PosicaoXY proximoPo = buscarSujeiraProxima("P");
            if (proximoPo != null) {
                this.posXY = proximoPo;
                this.labirinto.Limpar();
                this.pilhaMovimentos++;
            }
        }
    }

    public PosicaoXY buscarSujeiraProxima(String tipoSujeira) {
        for (int i = 0; i < labirinto.getTamanhoLabirinto(); i++) {
            for (int j = 0; j < labirinto.getTamanhoLabirinto(); j++) {
                if (labirinto.retonarValorPosicaoLabirinto(new
PosicaoXY(i, j)).equals(tipoSujeira)) {
                    return new PosicaoXY(i, j);
                }
            }
        }

        return null;
    }

    public boolean existeAguaNoLabirinto() {
        for (int i = 0; i < labirinto.getTamanhoLabirinto(); i++) {
            for (int j = 0; j < labirinto.getTamanhoLabirinto(); j++) {
                if (labirinto.retonarValorPosicaoLabirinto(new
PosicaoXY(i, j)).equals("A")) {
                    return true;
                }
            }
        }

        return false;
    }

    public String verificarTipoSujeira(PosicaoXY posicao) {
        return this.labirinto.retonarValorPosicaoLabirinto(posicao);
    }

```

```

    }

    public boolean isPosicaoValida(int x, int y) {
        String valor = this.Labirinto.retonarValorPosicaoLabirinto(new
PosicaoXY(x, y));
        return !valor.equals("P");
    }

    public PosicaoXY retonarMovimento() {
        int retornoPosX = this.posXY.getPosX();
        int retornoPosY = this.posXY.getPosY();

        switch(movimento) {
            case CIMA:
                if (retornoPosX > 0 && isPosicaoValida(retornoPosX - 1,
retornoPosY)) {
                    retornoPosX -= 1;
                } else {
                    movimento = MovimentosAgenteLabirinto.BAIXO;
                }
                break;
            case BAIXO:
                if (retornoPosX < this.Labirinto.getTamanhoLabirinto() -
1 && isPosicaoValida(retornoPosX + 1, retornoPosY)) {
                    retornoPosX += 1;
                } else {
                    movimento = MovimentosAgenteLabirinto.ESQUERDA;
                }
                break;
            case ESQUERDA:
                if (retornoPosY > 0 && isPosicaoValida(retornoPosX,
retornoPosY - 1)) {
                    retornoPosY -= 1;
                } else {
                    movimento = MovimentosAgenteLabirinto.DIREITA;
                }
                break;
            case DIREITA:
                if (retornoPosY < this.Labirinto.getTamanhoLabirinto() -
1 && isPosicaoValida(retornoPosX, retornoPosY + 1)) {
                    retornoPosY += 1;
                } else {
                    movimento = MovimentosAgenteLabirinto.CIMA;
                }
                break;
        }
        return new PosicaoXY(retornoPosX, retornoPosY);
    }
}

```



```
public PosicaoXY getPosicao() {  
    return this.posXY;  
}  
  
public boolean isAindaLimpando() {  
    return pilhaMovimentos < 4;  
}  
  
public void zerarPilha() {  
    this.pilhaMovimentos = 0;  
}  
}
```