Diagrama

Descripción generada automáticamente

**Practica 1- 2 Agente Reactivo Simple**

# Instrucciones

Implementar un agente reactivo simple para el entorno de la aspiradora. El agente debe trabajar con el simulador del ejercicio 1, debe recibir la información inicial de las posiciones y suciedad en el ambiente. También debe ser capaz de almacenar la puntuación del agente en base a su desempeño para cada configuración y la configuración media global.

# Código

Primero importamos las librerías necesarias.

from enum import Enum

from random import random, randint

import os

import time

from time import sleep

**TipoDeCelda** es una enumeración que define tres tipos de celdas posibles en el entorno: limpia, sucia y ocupada. Se utiliza para representar el estado de cada celda en el entorno.

class TipoDeCelda(Enum):

    Limpia = "⬜"

    Sucia = "⬛"

    Ocupada = "🔬"

**Entorno** es una clase que representa el entorno de la aspiradora. El constructor de la clase crea un entorno de ancho **ancho** y una proporción **proporcion\_sucia** de celdas sucias. La función **\_\_str\_\_** se utiliza para imprimir el estado actual del entorno en la consola. La función **esta\_dentro** comprueba si una posición está dentro del entorno. La función **esta\_sucia** comprueba si una celda en una posición determinada está sucia. La función **limpiar** limpia una celda en una posición determinada. La función **esta\_todo\_limpio** comprueba si todas las celdas del entorno están limpias.

class Entorno:

    def \_\_init\_\_(self, ancho: int, proporcion\_sucia: float = .5) -> None:

        self.espacio = []

        for i in range(ancho):

            if random() > proporcion\_sucia:

                self.espacio.append(TipoDeCelda.Limpia)

            else:

                self.espacio.append(TipoDeCelda.Sucia)

        self.espacioAux = self.espacio.copy()#espacioAux tendrá el espacio sin la aspiradora, es necesario para mostrar la aspiradora y que no intervenga en las condiciones

    def \_\_str\_\_(self):

        cadena = ""

        for celda in self.espacio:

            cadena += celda.value

        return cadena

    def esta\_dentro(self, x: int) -> bool:

        if x >= 0 and x < len(self.espacio):

            return True

        else:

            return False

    def esta\_sucia(self, x: int) -> bool:

        return self.espacioAux[x] == TipoDeCelda.Sucia

    def limpiar(self, x: int) -> None:

        if self.esta\_dentro(x):

            self.espacioAux[x] = TipoDeCelda.Limpia

    def esta\_todo\_limpio(self) -> bool:

        return TipoDeCelda.Sucia not in self.espacioAux

**Aspiradora** es una clase que representa la aspiradora en el entorno. El constructor de la clase toma un objeto **Entorno** y una posición **x** para inicializar la aspiradora en el entorno. La función **actuar** se utiliza para que la aspiradora limpie la celda actual si está sucia y además sume uno en la variable del puntaje o se mueva a la siguiente celda si está limpia. La función **mover** se utiliza para mover la aspiradora a la siguiente celda en la dirección actual.

class Aspiradora:

    class Direccion(Enum):

        Izquierda = -1

        Derecha = 1

    def \_\_init\_\_(self, ent: Entorno, x: int):

        self.entorno = ent

        self.entorno.espacio[x] = TipoDeCelda.Ocupada

        self.x = x

        self.puntaje = 0

        self.direccion = Aspiradora.Direccion.Izquierda

    def actuar(self):

        if self.entorno.esta\_sucia(self.x):

            self.entorno.limpiar(self.x)

            self.puntaje += 1

        else:

            self.mover()

    def mover(self):

        self.entorno.espacio[self.x] = TipoDeCelda.Limpia

        self.x += self.direccion.value

        if not self.entorno.esta\_dentro(self.x):

            if self.direccion == Aspiradora.Direccion.Derecha:

                self.direccion = Aspiradora.Direccion.Izquierda

            else:

                self.direccion = Aspiradora.Direccion.Derecha

            self.x += self.direccion.value \* 2

        self.entorno.espacio[self.x] = TipoDeCelda.Ocupada

**limpiar\_pantalla** es una función que se utiliza para borrar la consola y mostrar el estado actual del entorno y la puntuación de la aspiradora.

def limpiar\_pantalla():

    os.system('cls')

En la parte final del código, se crea un objeto **Entorno** y un objeto **Aspiradora**. El bucle **while** se utiliza para que la aspiradora siga limpiando el entorno hasta que todas las celdas estén limpias. En cada iteración del bucle, la aspiradora actúa y se actualiza el estado del entorno y la puntuación de la aspiradora en la consola.

tamanio  = 6

limpiar\_pantalla()

entorno = Entorno(tamanio)

Aspiradora = Aspiradora(entorno, 0)

print(entorno)

print(f'Puntuación = {Aspiradora.puntaje}')

sleep(0.7)

while not Aspiradora.entorno.esta\_todo\_limpio():

    Aspiradora.actuar()

    limpiar\_pantalla()

    print(entorno)

    print(f'Puntuación = {Aspiradora.puntaje}')

    sleep(1)

# Ejecución

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

# Conclusión

Este programa y práctica nos muestran cómo implementar un agente reactivo simple para el entorno de la aspiradora. El código está escrito en Python y utiliza la programación orientada a objetos para definir las clases Entorno y Aspiradora, que representan el ambiente en el que se ejecuta el agente y el agente en sí mismo, respectivamente. El agente es capaz de detectar la suciedad en el ambiente y de moverse para limpiarla. Además, el programa incluye una función para limpiar la pantalla, lo que hace que la visualización del entorno sea más clara y fácil de seguir.

Esta práctica es un buen ejemplo de cómo implementar un agente reactivo simple y puede ser útil para entender los conceptos básicos de la inteligencia artificial y la programación de agentes. También nos muestra cómo utilizar Python para implementar estos algoritmos de una manera clara y concisa. Sin embargo, es importante tener en cuenta que este agente reactivo simple tiene limitaciones y puede no ser capaz de manejar situaciones más complejas o entornos más grandes. En general, esta práctica es un buen punto de partida para aprender más sobre la inteligencia artificial y la programación de agentes.