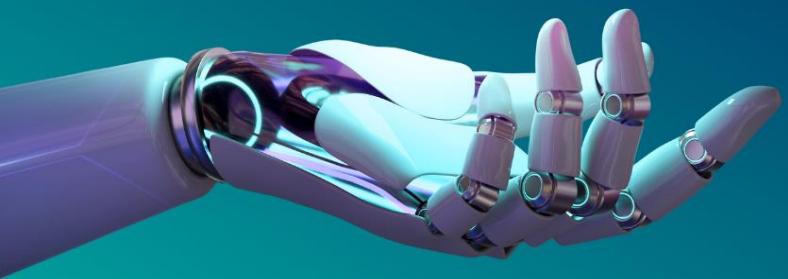
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

UNIVERSIDAD DE GUADALAJARA

PRÁCTICA 1-3

AGENTE REACTIVO SIMPLE



Alumnos:

Carbajal Armenta Yessenia Poala

Leon Estrada Rafael

Medina Bolaños Danna Paola

Códigos:

220286482

220286121

220286938

Maestro:

Oliva Navarro Diego Alberto

Materia:

Inteligencia Artificial

Sección:

D05

Guadalajara, Jal. a 10 de marzo del 2023

Practica 1- 3 Agente Reactivo Simple

Instrucciones

Desarrollar una versión modificada del entorno de la aspiradora, donde se penalice con un punto cada movimiento.

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 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. Esta fue la sección que se modificó dentro de la función de **actuar**, si está sucia va a sumar dos puntos al puntaje, en cambio, si solo se mueve penalizará un punto del puntaje por movimiento.

```
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 += 2 # Si limpia, suma dos por ser tarea más grande
        else:
            self.mover()
            self.puntaje -= 1 # Si se mueve, penaliza un punto

    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

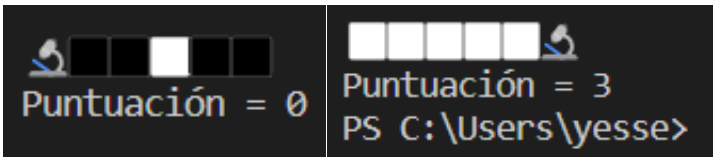
celdas estén limpias. En cada iteración del bucle, la aspiradora actúa y se actualiza el estado del entorno y la puntuación de la aspiradora en la consola.

```
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

En este ejemplo de ejecución podemos observar que tenemos 4 casillas sucias, por lo que en el puntaje se sumará un total de 8 puntos, pero para lograr esto, tiene que hacer 5 movimientos, por lo tanto, se penalizarán 5 puntos (1 por movimiento), por lo que la puntuación final quedará de 3 como se observa a continuación.



Preguntas

Este es un código que implementa un agente reactivo simple para el entorno de la aspiradora. El agente:

a. ***¿Puede un agente reactivo ser racional bajo las condiciones del problema? Justifica la respuesta.***

   Para determinar si un agente reactivo puede ser racional, es necesario considerar si el agente es capaz de tomar las mejores decisiones posibles en

función de la información disponible en cada momento y de los objetivos que se quieren alcanzar.

En general, los agentes reactivos son capaces de tomar decisiones rápidas y efectivas en situaciones en las que las opciones son limitadas y el conocimiento disponible es limitado. Sin embargo, la racionalidad del agente depende de la calidad de su diseño y de su capacidad para evaluar correctamente la información relevante en cada momento.

b. ***¿Qué sucedería si se tuviera un agente reactivo capaz de almacenar al menos un estado previo del entorno?***

Si se tiene un agente reactivo capaz de almacenar al menos un estado previo del entorno, este agente tendría la capacidad de recordar información relevante de estados anteriores y utilizarla para tomar decisiones más informadas en el futuro.

Con esta capacidad, el agente reactivo podría ser capaz de mejorar su rendimiento en ciertas situaciones, especialmente en aquellas donde las acciones que se toman en el presente tienen un impacto en el futuro. Al almacenar y recordar estados previos, el agente puede anticipar cómo las acciones actuales pueden afectar los futuros estados del entorno y tomar decisiones más racionales y efectivas.

Conclusión

Este es un código que implementa un agente reactivo simple para el entorno de la aspiradora. El agente es capaz de recibir la información inicial de las posiciones y suciedad en el ambiente y almacenar la puntuación del agente en base a su desempeño para cada configuración y la configuración media global.

En este caso, el agente penaliza cada movimiento con un punto y al aspirar suma 2 puntos al puntaje, lo que incentiva al agente a limpiar las celdas sucias en lugar de solo moverse por el entorno.

El código implementa una clase Entorno que representa el entorno de la aspiradora y una clase Aspiradora que representa al agente reactivo. La clase Aspiradora tiene un

método actuar() que decide si moverse o limpiar una celda y actualiza el puntaje en consecuencia.

El código también tiene una función limpiar_pantalla() para limpiar la pantalla entre cada acción del agente, lo que facilita la visualización del proceso de limpieza.