tp1

August 14, 2025

1 Temas Tratados en el Trabajo Práctico 1

- Diferencia entre Inteligencia e Inteligencia Artificial.
- Concepto de omnisciencia, aprendizaje y autonomía.
- Definición de Agente y sus características. Clasificación de Agentes según su estructura.
- Identificación y categorización del Entorno de Trabajo en tabla REAS.
- Caracterización del Entorno de Trabajo.

2 Anotaciones

"Acordarse de la definición de agente"

3 Ejercicios Teóricos

1. Defina con sus propias palabras inteligencia natural, inteligencia artificial y agente.

La inteligencia natural es la propia generada por la naturaleza en los sere vivos, la cual esta dada por eventos que ocurren en el cerebro. asdas

La inteligencia artificial es aquella inteligencia que esta desarrollada completamente dentro de una computadora, y que funciona mediante un algoritmo generado por una persona

Un agente es alguna cosa que interactua con el entorno, a partir de percepciones que recibe de el

2. ¿Qué es un agente racional?

Un agente racional es aquel que elige la accion que supuestamente maximice su medida de rendimiento, basandose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado

3. ¿Un agente es siempre una computadora?

No todos los agentes son desarrollados por computadoras, como un agente es alguien que pecibe su entorno y actua sobre el, un agente si puede ser una computadora, pero tambien puede ser un ser vivo que recibe sentidos y actua.

4. Defina Omnisciencia, Aprendizaje y Autonomía.

Omnisciencia es el conocimiento completo y perfecto sobre el estado actual del entorno y sobre las consecuencias de cada una de las acciones posibles, esto es imposible en entornos reales debido a que existe incertidumbre

Aprendizaje es el proceso mediante el cual un agente mejora su comportamiento a partir del estudio de experiencias pasadas y predicciones sobre el futuro

Autonomia es la capacidad de un agente de operar y tomar decisiones sin intervencion humana directa, basandose en sus propias percepciones y conocimientos adquiridos

5. Defina cada tipo de agente en función de su **estructura** y dé un ejemplo de cada categoría.

Agente Reactivo Simple: funcion con reglas, a una determinada condicion le sigue una determinada accion. No almacena informacion del pasado, sino que responde directamente a las percepciones actuales. Por ejemplo un robot aspiradora basico que gira cuando detecta un obstaculo

Agente Reactivo basado en Modelos: mantiene un estado interno que representa informacion sobre el mundo no visible en el momento, usa ese estado y un modelo de como evoluciona el entorno para decidir. Por ejemplo un termostato que predice temperatura usando el modelo fisico, para decidir si prender o apagar

Agente basado en Objetivos: ademas del estado, tiene una descripcion de objetivos y elige las acciones que mas lo acercen a los objetivos. Por ejmplo un GPS que planifica la ruta mas corta

Agente basado en Utilidad: usa en funcon de utilidad que mide que tan bueno es un estado para el agente, elige acciones que maximizan la utilidad esperada. Por ejemplo un auto autonomo que elige la maniobra mas segura y rapida, considerando todos los aspectos externos

Agente que aprende: integra componentes de aprendizaje para mejorar su rendimiento con la experiencia. Por ejemplo un sistema de recomendacion que mejora la sugerencias de productos a un usuario, dependiendo de las compras que haga

6. Para los siguientes entornos de trabajo indique sus **propiedades**:

   a. Una partida de ajedrez. Contexto: una partida de dos jugadores donde cada uno busca re

   b. Un partido de baloncesto. Contexto: un partido de dos equipos de jugadores; el punto c

   c. El juego Pacman. Contexto: un jugador jugando al juego de pacman; el punto de vista es

   d. El truco. Contexto: una partida de dos jugadores donde la estrategia de cada jugador c

   e. Las damas. Contexto: juego de tablero de dos jugadores, donde el objetivo es capturar

   f. El juego tres en raya. Contexto: juego simple de tablero de dos jugadores que buscan c

   g. Un jugador de Pokémon Go. Contexto: jugador que recorre el mundo real buscando captura

   h. Un robot explorador autónomo de Marte. Contexto: robot que se desplaza por la superfic

7. Elabore una tabla REAS para los siguientes entornos de trabajo:

   a. Crucigrama. (Considero un juego de mesa)

b. Taxi circulando.

c. Robot clasificador de piezas. (Considero un brazo robotico)

Agente: {[Jugadores], [Conductor], [Robot clasificador de piezas]}
Medidas de Rendimiento: {[Variabilidad de combinaciones de letras], [Seguridad, Comodidad
Entorno: {[Tablero, Demas letras posicionadas], [Taxi, Calles, Peatones, Clientes, Rutas]
Actuadores: {[Las manos], [Pedales, Palanca de cambios, Volante, Luces], [Pinzas, motores
Sensores: {[Sentido de la vista], [Sentidos humanos, Sensores del auto, gps], [De Velocid

4 Ejercicios Prácticos

8. La Hormiga de Langton es un agente capaz de modificar el estado de la casilla en la que se encuentra para colorearla o bien de blanco o de negro. Al comenzar, la ubicación de la hormiga es una casilla aleatoria y mira hacia una de las cuatro casillas adyacentes. Si...

- ... la casilla sobre la que está es blanca, cambia el color del cuadrado, gira noventa grados a la derecha y avanza un cuadrado.
- ... la casilla sobre la que está es negra, cambia el color del cuadrado, gira noventa grados a la izquierda y avanza un cuadrado.

Caracterice el agente con su tabla REAS y las propiedades del entorno para después programarlo en Python:

¿Observa que se repite algún patrón? De ser así, ¿a partir de qué iteración?

Agente: Hormiga Rendimiento: Entornos: Rejilla bidimensional de celdas Actuadores: Girar noventa grados a la derecha. Avanzar. Girar noventa grados a la izquierda Sensores: Sensor de color (para detectar el color). Sensor de orientacion (para saber para donde mira)

[1]:
```
import turtle
import sys
import time
import math
from win32api import GetSystemMetrics
from easygui import multenterbox, msgbox

def langtonAnt():

    #Configuración el multenterbox para pedir datos al usuario
    msg = "Por favor, introduzca los datos" + "\nNota 1: Tenga en cuenta que la␣
 ↪posición x, y = (0, 0) es el centro de la ventana, pero su <<n>> será tomado␣
 ↪de forma global." + "\nNota 2: Tenga en cuenta que La hormiga sigue un␣
 ↪camino aparentemente azaroso hasta los 10.000 pasos."
    title = "La hormiga de Langton"
    fieldNames = ["Número de movimientos:", "Posición inicial x:", "Posición␣
 ↪inicial y:", "Tamaño <<n>> de la grilla:"]
    fieldValues = multenterbox(msg, title, fieldNames)
```

```
    if fieldValues is None:
        sys.exit(0)

    #Se comprueba que el usuario ha llenado todos los campos
    while 1:
        errorMsg = ""
        for i, name in enumerate(fieldNames):
            if fieldValues[i].strip() == "":
                errorMsg += "{} Es un campo requerido para iniciar.\n\n".
↪format(name)
        if errorMsg == "":
            break #Todos los campos llenos
        fieldValues = multenterbox(errorMsg, title, fieldNames, fieldValues)
        if fieldValues is None:
            break

    #Configuración la ventana
    widht = GetSystemMetrics(0)
    height = GetSystemMetrics(1)
    wn = turtle.Screen()
    wn.title("La Hormiga de Langton")
    wn.bgcolor("white")
    wn.screensize(widht, height)

    #Se llama la función para dibujar el borde de la grilla
    border((int(fieldValues[3])+10))

    #Configuración del texto contador de movimientos
    text = turtle.Turtle()
    text.speed(0)
    text.color("red")
    text.penup()
    text.hideturtle()
    text.goto(0, 300)

    #Dicionario con las cordenadas y su respectivo color
    maps = {}

    #Configuración de la hormiga
    ant =  turtle.Turtle()
    ant.shape("square")
    ant.shapesize(0.5)
    ant.penup()
    ant.goto(int(fieldValues[1]), int(fieldValues[2]))
    ant.speed(0)
    pos = coordinate(ant)
```

```
    #Variable para contar los movimientos
    cont = 0

    #Variable para verificar que la hormiga no se salga de los limites
    hit = False

    #Movimiento de la hormiga
    while cont <= int(fieldValues[0])-1 and hit == False:
        step = 10
        if pos not in maps or maps[pos] == "white":
            ant.fillcolor("black")
            ant.stamp()
            invert(maps, ant, "black")
            ant.right(90)
            ant.forward(step)
            pos = coordinate(ant)
            cont += 1
        elif maps[pos] == "black":
            ant.fillcolor("white")
            ant.stamp()
            invert(maps, ant, "white")
            ant.left(90)
            ant.forward(step)
            pos = coordinate(ant)
            cont += 1

        #Comprueba que la hormiga esta dentro de la grilla
        if round(math.fabs(ant.xcor())) > (int(fieldValues[3]) / 2) - 1:
            hit = True
            msgbox(msg="La hormiga no puede segir avanzando porque chocó con␣
↪los bordes de la grilla que usted introdujo."
            "\nSi desea que la hormiga complete todos los movimientos, se␣
↪recomienda introducir una grilla mas grande.", title="Fuera de límites",␣
↪ok_button="Aceptar")
        if round(math.fabs(ant.ycor())) > (int(fieldValues[3]) / 2) - 1:
            hit = True
            msgbox(msg="La hormiga no puede segir avanzando porque chocó con␣
↪los bordes de la grilla que usted introdujo."
            "\nSi desea que la hormiga complete todos los movimientos, se␣
↪recomienda introducir una grilla mas grande.", title="Fuera de límites",␣
↪ok_button="Aceptar")

        text.clear()
        text.write("Movimientos: {}".format(cont), align="center",␣
↪font=("Courier", 24, "normal"))

    while True:
```

```
        wn.update()

#Devuelve la cordenada de la hormiga en una tupla
def coordinate(ant):
    return (round(ant.xcor()), round(ant.ycor()))

#Invierte el color de la celda de la grilla en que está la hormiga
def invert(graph, ant, color):
    graph[coordinate(ant)] = color

#Dibuja el borde de la grilla
def border(n):
    border = turtle.Turtle()
    border.hideturtle()
    border.penup()
    border.goto(n/2, 0)
    border.pendown()
    border.left(90)
    border.forward(n/2)
    border.left(90)
    border.forward(n)
    border.left(90)
    border.forward(n)
    border.left(90)
    border.forward(n)
    border.left(90)
    border.forward(n/2)

#Se ejecuta el código
langtonAnt()
```

```
---------------------------------------------------------------------------
TclError                                  Traceback (most recent call last)
Cell In[3], line 130
    127     border.forward(n/2)
    129 #Se ejecuta el código
--> 130 langtonAnt()

Cell In[3], line 80, in langtonAnt()
     78 elif maps[pos] == "black":
     79     ant.fillcolor("white")
---> 80     ant.stamp()
     81     invert(maps, ant, "white")
     82     ant.left(90)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.13_3.13.177
 ↪0_x64__qbz5n2kfra8p0\Lib\turtle.py:3095, in RawTurtle.stamp(self)
```

```
   3093 tshape = shape._data
   3094 if ttype == "polygon":
-> 3095     stitem = screen._createpoly()
   3096     if self._resizemode == "noresize": w = 1
   3097     elif self._resizemode == "auto": w = self._pensize

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.13_3.13.177
 ↪0_x64__qbz5n2kfra8p0\Lib\turtle.py:490, in TurtleScreenBase._createpoly(self)
    487 def _createpoly(self):
    488     """Create an invisible polygon item on canvas self.cv)
    489     """
--> 490     return␣
 ↪self.cv.create_polygon((0, 0, 0, 0, 0, 0), fill=  , outline=  )

File <string>:1, in create_polygon(self, *args, **kw)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.13_3.13.177
 ↪0_x64__qbz5n2kfra8p0\Lib\tkinter\__init__.py:3002, in Canvas.
 ↪create_polygon(self, *args, **kw)
   3000 def create_polygon(self, *args, **kw):
   3001     """Create polygon with coordinates x1,y1,…,xn,yn."""
-> 3002     return self._create(          , args, kw)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.13_3.13.177
 ↪0_x64__qbz5n2kfra8p0\Lib\tkinter\__init__.py:2976, in Canvas._create(self,␣
 ↪itemType, args, kw)
   2974 else:
   2975     cnf = {}
-> 2976 return self.tk.getint(self.tk.call(
   2977     self._w,          , itemType,
   2978     *(args + self._options(cnf, kw))))

TclError: invalid command name ".!canvas"
```

9. El Juego de la Vida de Conway consiste en un tablero donde cada casilla representa una célula, de manera que a cada célula le rodean 8 vecinas. Las células tienen dos estados: están *vivas* o *muertas*. En cada iteración, el estado de todas las células se tiene en cuenta para calcular el estado siguiente en simultáneo de acuerdo a las siguientes acciones:

- Nacer: Si una célula muerta tiene exactamente 3 células vecinas vivas, dicha célula pasa a estar viva.
- Morir: Una célula viva puede morir sobrepoblación cuando tiene más de tres vecinos alrededor o por aislamiento si tiene solo un vecino o ninguno.
- Vivir: una célula se mantiene viva si tiene 2 o 3 vecinos a su alrededor.

  Caracterice el agente con su tabla REAS y las propiedades del entorno para después programarlo en Python:

REAS (por célula-agente)

Rendimiento: Persistencia de patrones locales (p. ej., “seguir vivo si hay 2–3 vecinas” o “nacer si hay 3”). (Opcional) Objetivos emergentes: maximizar supervivencia local o estabilidad del patrón. Para lo que fue creado el código cumple con las expectativas

Entorno: Las 8 celdas vecinas inmediatas (vecindad de Moore) y su estado actual en cada tick.

Actuadores: Cambiar su propio estado en el próximo tick.

Sensores: Contar vecinas vivas (lectura de estados inmediatos).

Propiedades del entorno (desde la perspectiva de cada célula)

Observabilidad: Parcial-local (solo percibe vecinas inmediatas; globalmente, el sistema es observable si miramos toda la grilla).

Determinismo: Determinista (su próxima acción/estado depende únicamente del recuento local actual).

Episódico vs. Secuencial: Secuencial (la historia importa a través del estado previo).

Estático vs. Dinámico: Dinámico por ticks (las vecinas pueden cambiar en cada paso).

Discreto vs. Continuo: Discreto.

Individual vs. Multiagente: Multiagente (muchas células actuando en paralelo con reglas locales).

[ ]:
```python
import numpy as np
from scipy.signal import convolve2d

class GameOfLife:
    """
    Implementación del Juego de la Vida de Conway.
    Estados: True = viva, False = muerta.
    boundary: "fixed" (bordes muertos) o "toroidal" (mundo envolvente).
    """
    KERNEL = np.array([[1,1,1],
                       [1,0,1],
                       [1,1,1]], dtype=int)

    def __init__(self, board: np.ndarray, boundary: str = "fixed"):
        if board.dtype != np.bool_:
            board = board.astype(bool)
        self.board = board.copy()
        if boundary not in ("fixed", "toroidal"):
            raise ValueError('boundary debe ser "fixed" o "toroidal"')
        self.boundary = boundary

    @classmethod
    def from_random(cls, rows: int, cols: int, p_alive: float = 0.2, boundary:␣
 ↪str = "fixed", seed: int | None = None):
        rng = np.random.default_rng(seed)
```

```
        board = rng.random((rows, cols)) < p_alive
        return cls(board, boundary)

    def _neighbor_count(self) -> np.ndarray:
        if self.boundary == "fixed":
            # bordes como muertos (pad implícito con zeros en la convolución␣
↪'same' y 'fillvalue=0')
            return convolve2d(self.board.astype(int), self.KERNEL, mode="same",␣
↪boundary="fill", fillvalue=0)
        else:
            # toroidal: usar wrap para simular mundo envolvente
            # Truco: convolve2d con 'wrap' maneja bordes enrollados
            return convolve2d(self.board.astype(int), self.KERNEL, mode="same",␣
↪boundary="wrap")

    def step(self) -> np.ndarray:
        """Calcula y aplica una iteración; devuelve el nuevo tablero."""
        neighbors = self._neighbor_count()
        birth = (~self.board) & (neighbors == 3)
        survive = self.board & ((neighbors == 2) | (neighbors == 3))
        self.board = birth | survive
        return self.board

    def run(self, steps: int) -> np.ndarray:
        """Avanza varias iteraciones; retorna el tablero final."""
        for _ in range(steps):
            self.step()
        return self.board

    def to_ascii(self, alive_char: str = "■", dead_char: str = " "):
        """Devuelve una string con el tablero en ASCII/Unicode."""
        # Convertimos booleanos a caracteres
        rows = ["".join(alive_char if c else dead_char for c in row) for row in␣
↪self.board]
        return "\n".join(rows)

# -------------------------
# Ejemplos de uso:
# -------------------------
if __name__ == "__main__":
    # 1) Semilla aleatoria
    gol = GameOfLife.from_random(rows=20, cols=40, p_alive=0.25,␣
↪boundary="toroidal", seed=42)
    print("Estado inicial:")
    print(gol.to_ascii())

    # 10 iteraciones
```

```
    gol.run(10)
    print("\nEstado tras 10 iteraciones:")
    print(gol.to_ascii())

    # 2) Semilla con patrón clásico: "glider"
    glider = np.zeros((15, 15), dtype=bool)
    # Coordenadas del planeador
    coords = [(1,2), (2,3), (3,1), (3,2), (3,3)]
    for r, c in coords:
        glider[r, c] = True

    gol2 = GameOfLife(glider, boundary="toroidal")
    print("\nGlider - estado inicial:")
    print(gol2.to_ascii())

    for i in range(4):
        gol2.step()
        print(f"\nGlider - paso {i+1}:")
        print(gol2.to_ascii())
```

5 Bibliografía

Russell, S. & Norvig, P. (2004) *Inteligencia Artificial: Un Enfoque Moderno.* Pearson Educación S.A. (2a Ed.) Madrid, España

Poole, D. & Mackworth, A. (2023) *Artificial Intelligence: Foundations of Computational Agents.* Cambridge University Press (3a Ed.) Vancouver, Canada