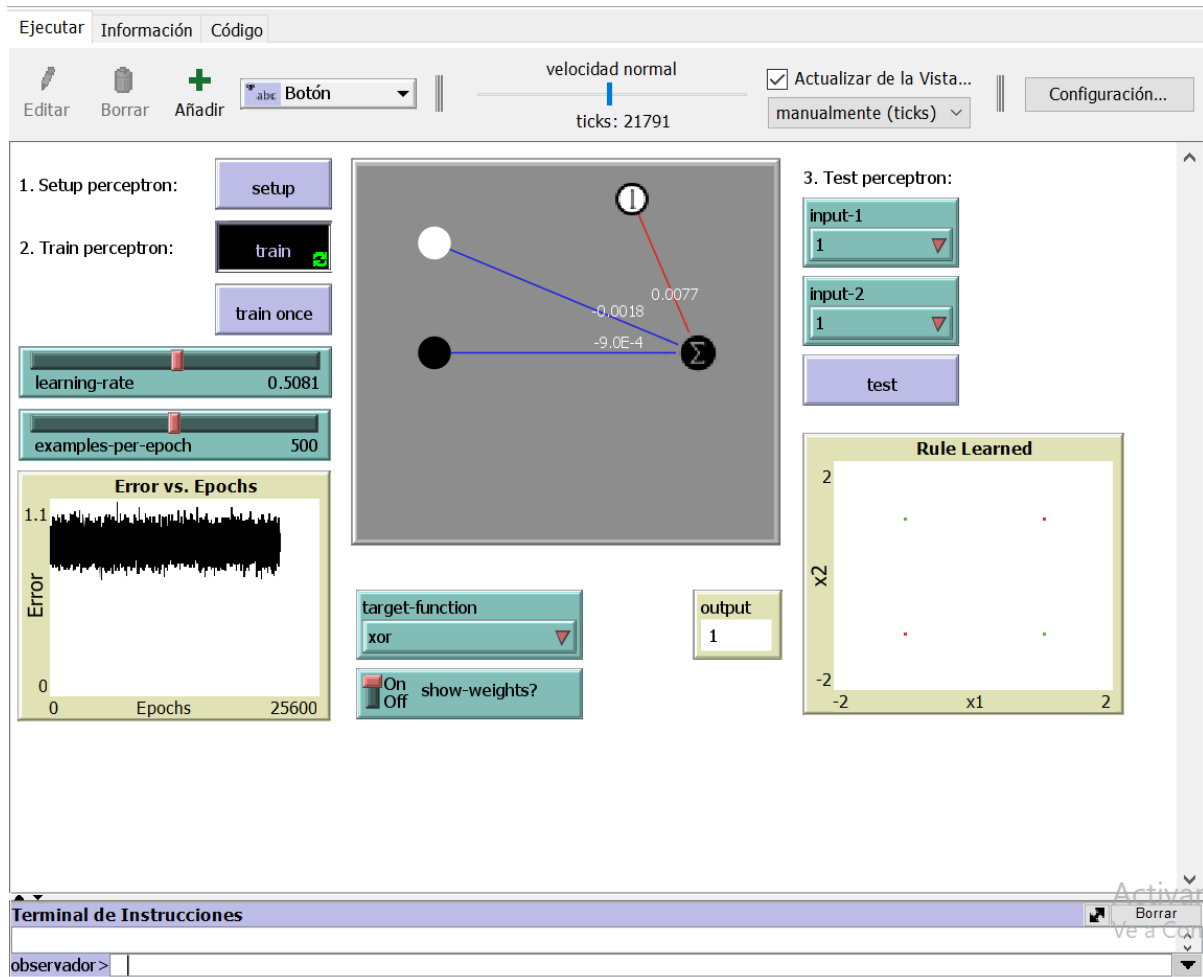
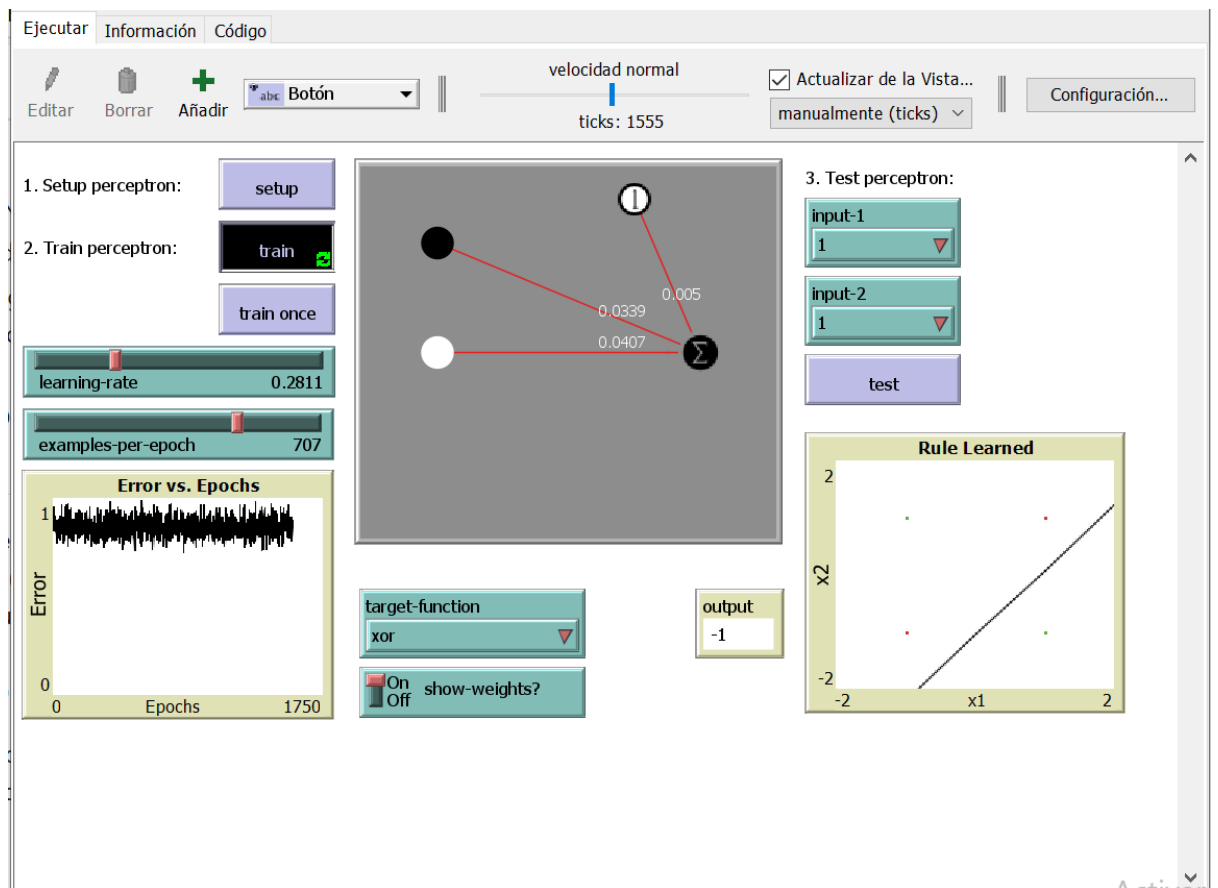


Segundo Parcial

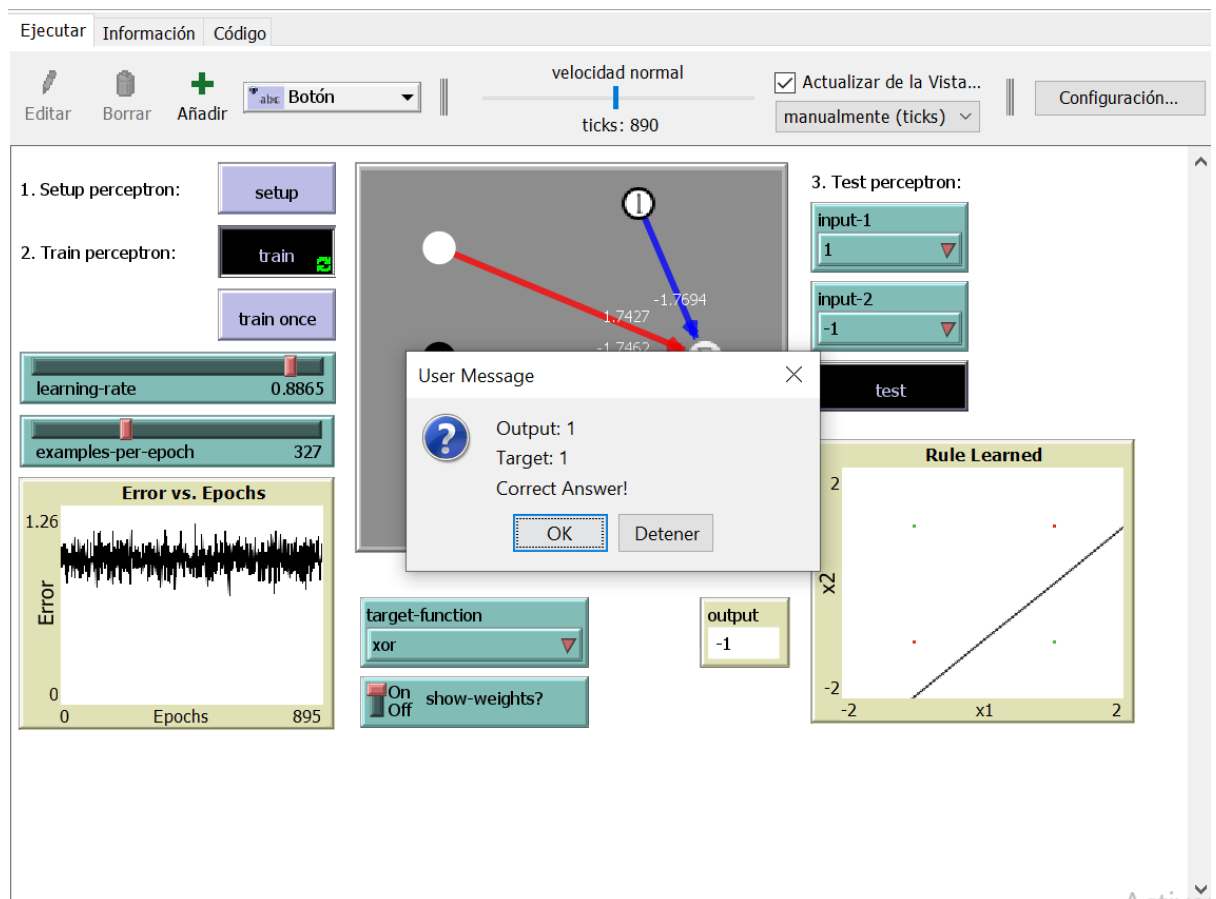
1. Modelamiento de un Perceptrón en NetLogo



(Entrenamiento Perceptrón)



(Entrenamiento Perceptrón)



(Resultados al poner en el input 1, -1)

Explicación Perceptron:

El perceptrón es una red neuronal de una capa que clasifica datos de entrada según un conjunto de pesos y un umbral, esto funciona ajustando los pesos de sus conexiones para minimizar el error en sus predicciones, logrando finalmente clasificar entradas binarias. En esta implementación en NetLogo, el perceptrón se modela con nodos de entrada, un nodo de sesgo, y un nodo de salida que representa el perceptrón. Los nodos se conectan mediante enlaces ponderados, donde sus pesos se ajustan durante el proceso de entrenamiento.

Estructura del Código:

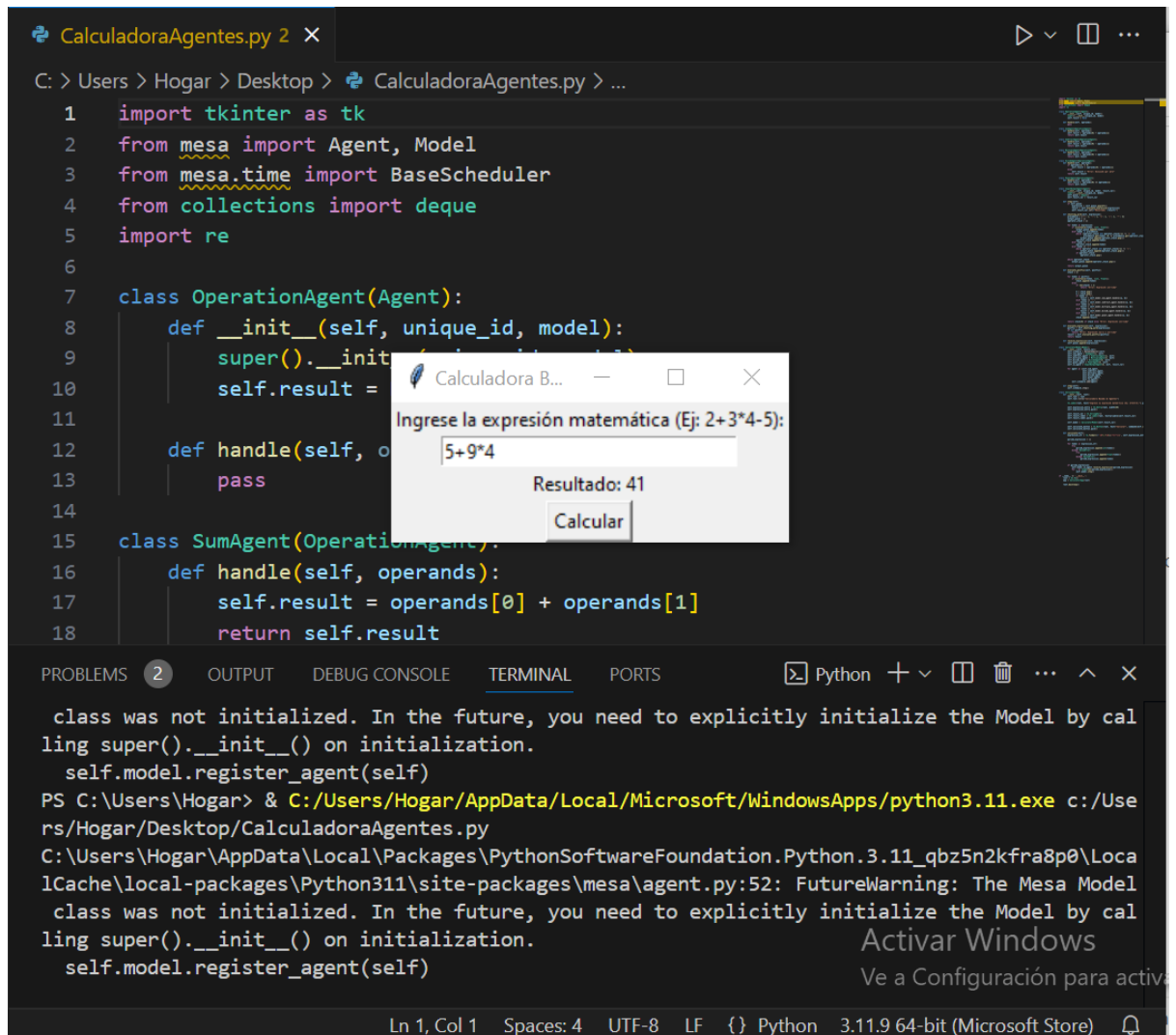
- Variables Globales: Se define un conjunto de variables para almacenar los elementos principales de la red:
 - epoch-error: Para medir el error en cada ciclo de entrenamiento.
 - perceptron: Nodo de salida que actúa como clasificador.
 - input-node-1 y input-node-2: Nodos de entrada.
 - Configuración Inicial (setup):

- Se crean y configuran los nodos de entrada, el nodo de salida, y el nodo de sesgo.
- Cada nodo se ubica en posiciones específicas y se establece su activación y umbral.
- Los nodos de entrada y sesgo se conectan al perceptrón mediante enlaces que tienen pesos iniciales aleatorios.
- Entrenamiento (train):
 - Se repite un número específico de veces en cada época.
 - Los nodos de entrada reciben activaciones aleatorias, luego el perceptrón calcula su activación utilizando la suma ponderada de las activaciones de los nodos de entrada y sesgo.
 - Se ajustan los pesos de los enlaces en función del error obtenido en la predicción.
- Cálculo de la Activación y Ajuste de Pesos:
 - compute-activation: Calcula la salida del perceptrón mediante una función de activación (signo) que compara la suma ponderada de entradas con el umbral.
 - update-weights: Ajusta los pesos según la diferencia entre la activación obtenida y la respuesta esperada (target-answer), utilizando una tasa de aprendizaje.
- Pruebas y Visualización:
 - test: Permite evaluar entradas específicas y comparar la salida del perceptrón con el resultado deseado.
 - Visualización: Se actualizan los colores de los nodos y los enlaces para reflejar el estado de activación y los pesos.

Resultados Obtenidos

Durante el entrenamiento, el perceptrón ajusta sus pesos hasta que su error disminuye en cada época. Al final del entrenamiento el perceptrón puede clasificar correctamente las entradas para las funciones lógicas seleccionadas, como OR, AND, y XOR. Las gráficas generadas muestran el progreso en la reducción del error y la línea de clasificación aprendida, visualizando cómo el perceptrón se adapta a los patrones de entrada, esta implementación en NetLogo demuestra cómo un perceptrón aprende a clasificar entradas binarias ajustando sus pesos en función de los errores acumulados durante el entrenamiento.

2. Implementación de una Calculadora Basada en el Paradigma de Agentes



```
CalculadoraAgentes.py 2 x
C: > Users > Hogar > Desktop > CalculadoraAgentes.py > ...

1  import tkinter as tk
2  from mesa import Agent, Model
3  from mesa.time import BaseScheduler
4  from collections import deque
5  import re
6
7  class OperationAgent(Agent):
8      def __init__(self, unique_id, model):
9          super().__init__(unique_id, model)
10         self.result = 0
11
12         def handle(self, operands):
13             pass
14
15     class SumAgent(OperationAgent):
16         def handle(self, operands):
17             self.result = operands[0] + operands[1]
18         return self.result
```

Calculadora B...

Ingrese la expresión matemática (Ej: 2+3*4-5):

5+9*4

Resultado: 41

Calcular

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - - - - -

class was not initialized. In the future, you need to explicitly initialize the Model by calling super().__init__() on initialization.

self.model.register_agent(self)

PS C:\Users\Hogar> & C:/Users/Hogar/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Hogar/Desktop/CalculadoraAgentes.py

C:\Users\Hogar\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\mesa\agent.py:52: FutureWarning: The Mesa Model class was not initialized. In the future, you need to explicitly initialize the Model by calling super().__init__() on initialization.

self.model.register_agent(self)

Activar Windows

Ve a Configuración para activar Windows

Ln 1, Col 1 Spaces: 4 UTF-8 LF {} Python 3.11.9 64-bit (Microsoft Store)

(Se observa en la interfaz una expresión realizada en la calculadora)

The screenshot shows a Python IDE with a file named 'CalculadoraAgentes.py'. The code defines two classes: 'OperationAgent' and 'SumAgent'. 'OperationAgent' has an 'handle' method that takes operands and performs a calculation. 'SumAgent' inherits from 'OperationAgent' and implements the 'handle' method to sum two operands. A small window titled 'Calculadora B...' is overlaid on the code, showing the input '6/4+8*2+7-5' and the result 'Resultado: 21.0'. The terminal at the bottom shows a warning message: 'FutureWarning: The Mesa Model class was not initialized. In the future, you need to explicitly initialize the Model by calling super().__init__() on initialization.'

```
1 import tkinter as tk
2 from mesa import Agent, Model
3 from mesa.time import BaseScheduler
4 from collections import deque
5 import re
6
7 class OperationAgent(Agent):
8     def __init__(self, unique_id, model):
9         super().__init__(unique_id, model)
10        self.result = 0
11
12    def handle(self, operands):
13        pass
14
15 class SumAgent(OperationAgent):
16     def handle(self, operands):
17         self.result = operands[0] + operands[1]
18         return self.result
```

Calculadora B...
Ingrese la expresión matemática (Ej: 2+3*4-5):
6/4+8*2+7-5
Resultado: 21.0
Calcular

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - □ □ ... ^ ×

class was not initialized. In the future, you need to explicitly initialize the Model by calling super().__init__() on initialization.
self.model.register_agent(self)
PS C:\Users\Hogar> & C:/Users/Hogar/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Hogar/Desktop/CalculadoraAgentes.py
C:\Users\Hogar\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\mesa\agent.py:52: FutureWarning: The Mesa Model class was not initialized. In the future, you need to explicitly initialize the Model by calling super().__init__() on initialization.
self.model.register_agent(self)

(Se observa en la interfaz una expresión realizada en la calculadora)

Informe sobre el Funcionamiento

- Estructura General

El código está organizado en varias clases que representan diferentes componentes de la calculadora. Utiliza el patrón de diseño basado en agentes, donde cada operación matemática se maneja a través de un agente específico.

La interfaz gráfica permite al usuario ingresar expresiones matemáticas y obtener resultados.

Clases Principales

- Agentes de Operación

- OperationAgent: Clase base para todos los agentes de operación. Define un método handle que debe ser implementado por las subclases.
- SumAgent: Realiza la suma de dos operandos.
- SubtractAgent: Realiza la resta de dos operandos.

- MultiplyAgent: Realiza la multiplicación de dos operandos.
- DivideAgent: Realiza la división de dos operandos y maneja la división por cero.
- PowerAgent: Realiza la potenciación (elevar un número a otro).

Cada uno de estos agentes implementa el método handle, que toma una lista de operandos y devuelve el resultado correspondiente.

- Agente de Entrada/Salida
 - InputOutputAgent: Este agente gestiona la entrada del usuario y la salida del resultado. Utiliza una cola (deque) para almacenar las expresiones ingresadas y un método step para procesarlas.
 - shunting_yard: Convierte una expresión infija en notación postfija (también conocida como notación polaca inversa), lo que facilita su evaluación respetando la precedencia de las operaciones.
 - evaluate_postfix: Evalúa la expresión en notación postfija utilizando una pila, llamando a los agentes correspondientes para realizar las operaciones.
- Modelo de Calculadora:
 - CalculatorModel: Coordina todos los agentes. Crea instancias de los agentes de operación y del agente de entrada/salida, añadiéndolos a un programador (BaseScheduler) que controla el flujo de ejecución.
- Interfaz Gráfica
 - CalculatorApp: Clase que define la interfaz gráfica utilizando Tkinter. Incluye:
 - Un campo de entrada para que el usuario ingrese expresiones matemáticas.
 - Un botón para calcular el resultado.
 - Una etiqueta para mostrar el resultado.

Cuando el usuario hace clic en el botón "Calcular", se llama al método calculate, que:

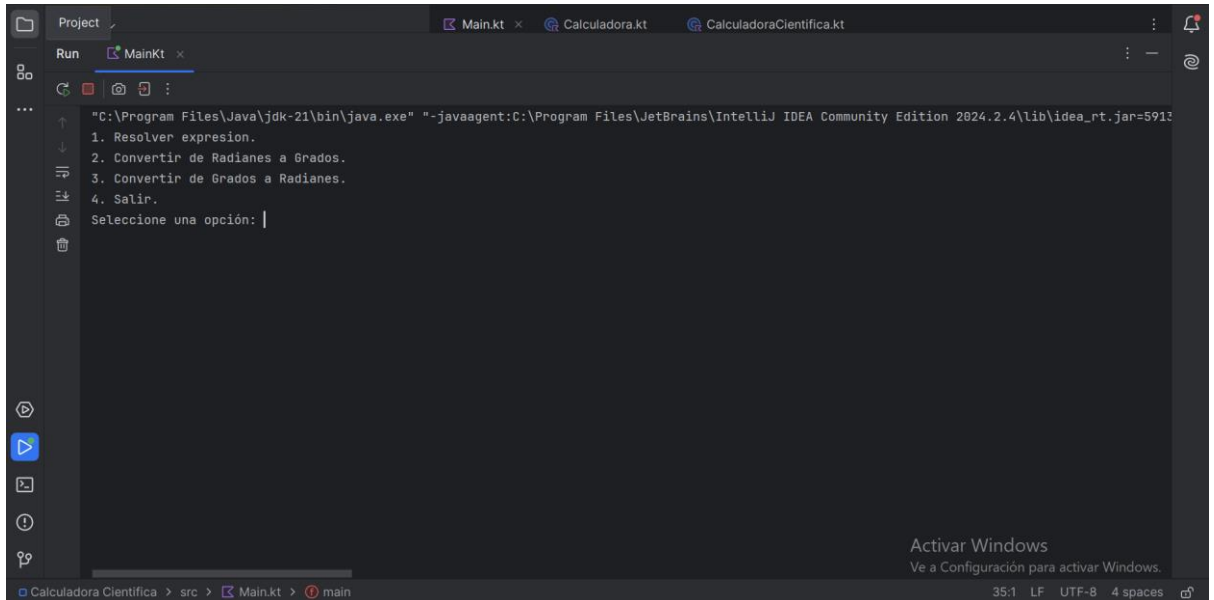
1. Extrae y tokeniza la expresión ingresada utilizando expresiones regulares para identificar números y operadores.
2. Envía la expresión al agente de entrada/salida.
3. Ejecuta el modelo para procesar la expresión.
4. Ejecución

Al ejecutar el script, se inicializa una ventana Tkinter donde el usuario puede ingresar expresiones matemáticas. El flujo general es:

1. El usuario escribe una expresión en el campo de entrada.
2. Al presionar "Calcular", se tokeniza la expresión y se envía al InputOutputAgent.

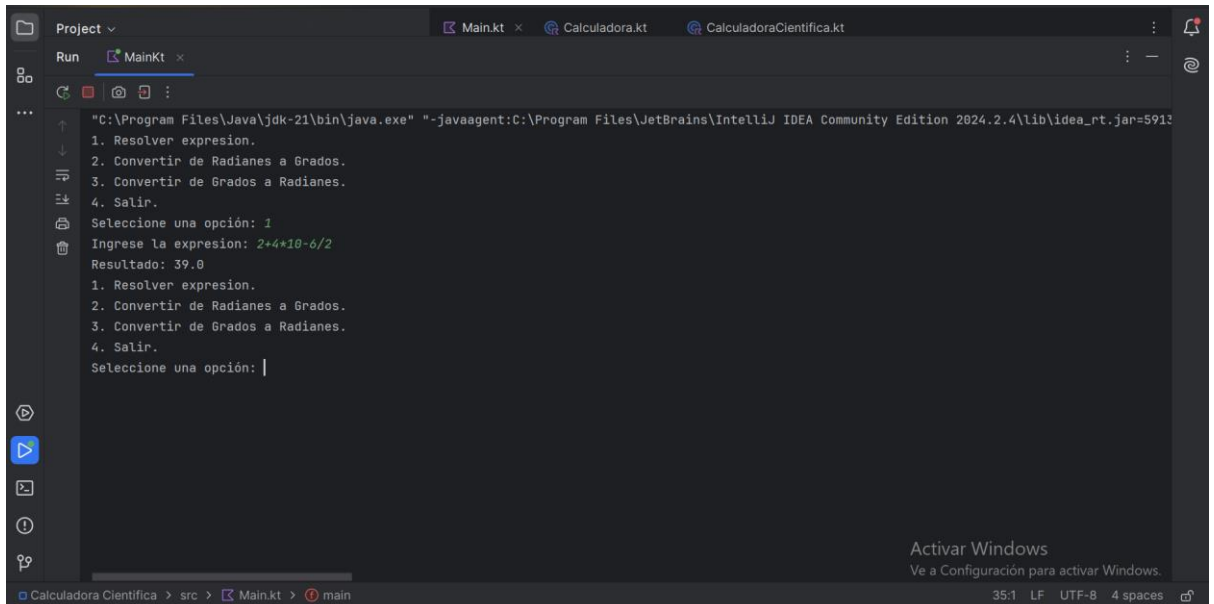
3. Este agente convierte la expresión a notación postfija y la evalúa utilizando los agentes correspondientes.
4. El resultado se muestra en la etiqueta designada en la interfaz gráfica.

3. Implementación de una Calculadora Científica usando el Paradigma de Objetos en Kotlin



```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.4\lib\idea_rt.jar=5912..."
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: |
```

(En esta imagen evidenciamos el inicio del programa que evidencia un menú de las funciones de la calculadora)



```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.4\lib\idea_rt.jar=5912..."
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: 1
Ingrese la expresion: 2+4*10-6/2
Resultado: 39.0
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: |
```

(En esta imagen se seleccionó la función 1, y se realizó el cálculo de una expresión sencilla)


```
Project > Run MainKt x Calculadora.kt CalculadoraCientifica.kt
Run MainKt x
More tool windows
expresion: 2+4*10-6/2
Resultado: 39.0
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: 2
Ingrese el valor en radianes: 54
Resultado en grados: 3093.9720937064453
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: 3
Ingrese el valor en grados:
3093.9720
Resultado en radianes: 53.999998364514
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: |
Activar Windows
Ve a Configuración para activar Windows.
Calculadora Cientifica > src > MainKt > main
35:1 LF UTF-8 4 spaces
```

(Aquí evidenciamos la conversión de Grados a Radianes y de Radianes a grados obteniendo la conversión del mismo número)

```
Project > Run MainKt x Calculadora.kt CalculadoraCientifica.kt
Run MainKt x
Resultado: 1.7018070490682369
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: 2*sin(45)-tan(20)-(6+4)
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: 1
Ingrese la expresion: 2*sin(45)-tan(20)-(6+4)
Resultado: -10.535353895156506
1. Resolver expresion.
2. Convertir de Radianes a Grados.
3. Convertir de Grados a Radianes.
4. Salir.
Seleccione una opción: 4
Saliendo del programa.
Process finished with exit code 0
Activar Windows
Ve a Configuración para activar Windows.
Calculadora Cientifica > src > MainKt > main
35:1 LF UTF-8 4 spaces
```

(Se observa que se realiza un cálculo más complejo y la opción 4 que finaliza el programa)

Explicación código:

- Clase Calculadora:

La clase Calculadora es la base que define las operaciones básicas (suma, resta, multiplicación, división).

Tiene un método potencia sobrecargado que calcula potencias de números enteros y decimales. Para ello, el método usa “x.toDouble().pow(a).toInt()” para convertir el resultado en un entero en el caso de los números enteros.

- Clase CalculadoraCientifica:
Esta clase hereda de Calculadora y agrega funciones complejas como seno, coseno, tangente, raizCuadrada, raizCubica, logaritmos y conversiones entre grados y radianes.
Redefine el método potencia usando “override”, pero realiza un ajuste para manejar el caso especial de la raíz cúbica cuando x es $1/3$.
- Función main:
Se presenta un menú interactivo que permite al usuario elegir entre 4 opciones (resolver una expresión, convertir entre grados y radianes, o salir del programa).
Se utiliza un bucle while para continuar con el menú hasta que el usuario elija la opción “salir”.
- Funciones
 - Función tokenize:
Esta función convierte una expresión matemática en una lista de tokens usando expresiones. Detecta números, operadores básicos, y funciones (sin, cos, tan),
esto facilita la lectura de cada parte de la expresión por separado, logrando realizar el cálculo de expresiones complejas.
 - Función resolverExpresion:
Esta es la función principal para evaluar una expresión matemática. Esta implementa el algoritmo de Shunting Yard de Dijkstra para procesar expresiones con una precedencia definida en el mapa precedencia.

Shuntin yard: El algoritmo Shunting Yard convierte expresiones en notación infija a notación postfija para facilitar su evaluación, este utiliza pilas para gestionar operadores y la salida, aplicando precedencia para llevar un correcto orden de los operadores. Al procesar la expresión el resultado en postfija puede evaluarse en orden secuencial.
 - Función aplicarOperador:
Toma un operador y los operandos necesarios para aplicar la operación, ya sea una operación básica o compleja.
Según el operador se extraen los valores de la pila y se llama a la función.
- Principios de Programación Orientada a Objetos
 - Encapsulamiento:

Las funciones de Calculadora y CalculadoraCientifica son encapsuladas dentro de las clases, esto permite proteger el acceso a los detalles internos de cada operación. Por ello, los detalles de cada cálculo están ocultos al usuario, y solo se proporcionan funciones públicas para operar con la calculadora.

- Herencia

CalculadoraCientifica hereda de Calculadora, lo que le permite extender las funcionalidades de Calculadora añadiendo operaciones complejas, esto permite reutilizar métodos y agregar nuevos sin duplicar código, que ayuda a simplificar la estructura de la calculadora.

- Polimorfismo:

La función potencia es un ejemplo de polimorfismo, ya que, CalculadoraCientifica redefine usando “override” el método potencia de Calculadora para adaptarlo a sus propias necesidades como el cálculo de raíz cúbica.

También, el polimorfismo permite a la clase CalculadoraCientifica llamar a los métodos heredados de Calculadora para realizar operaciones básicas.