

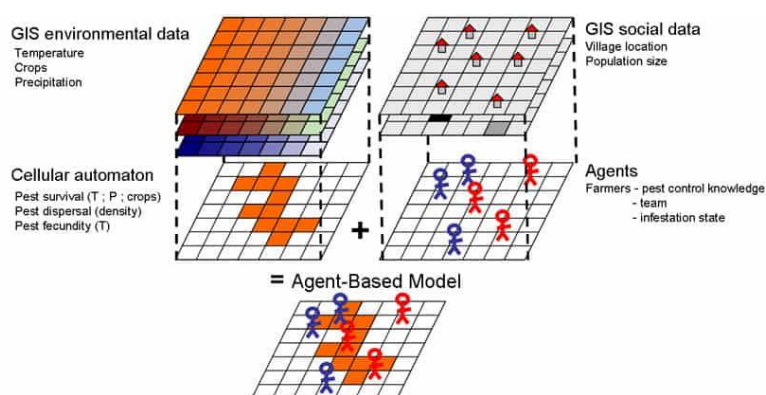


**Centro Universitario de Ciencias Exactas e
Ingenierías**
Universidad de Guadalajara



Práctica 9: Modelado basado en agentes

Aprendizaje Máquina



Alumno: Samuel David Pérez Brambila

Código: 222966286

Profesora: Karla Ávila Cárdenas

Sección: D01

Fecha de Entrega: 17 de Noviembre de 2024

Introducción

El Modelado Basado en Agentes (ABM) es “una técnica de modelado computacional que se utiliza para simular sistemas complejos mediante la creación de "agentes" individuales que interactúan entre sí y con su entorno. Estos agentes pueden representar entidades físicas o abstractas, como personas, animales, células, empresas o incluso ideas, dependiendo del sistema que se esté modelando.” (Silva, 2024)

En un modelo basado en agentes, cada agente sigue un conjunto de reglas simples que dictan su comportamiento y su interacción con otros agentes y el entorno. Estas reglas pueden ser tan simples como moverse aleatoriamente en un espacio o tan complejas como tomar decisiones basadas en el estado interno del agente y la información del entorno. Lo fascinante del ABM es que, a pesar de que cada agente sigue reglas simples, las interacciones entre los agentes pueden dar lugar a patrones complejos y comportamientos emergentes en el sistema en su conjunto. Estos patrones pueden ser difíciles de predecir mediante el análisis tradicional, lo que hace que el ABM sea una herramienta poderosa para estudiar sistemas complejos y adaptativos.

El modelado basado en agentes se ha utilizado en una amplia gama de disciplinas, incluidas la ecología, la economía, la sociología, la ciencia política y la informática, entre otras. Por ejemplo, en ecología, se puede utilizar para simular la propagación de enfermedades en una población animal, mientras que en economía, se puede utilizar para modelar el comportamiento de los consumidores en un mercado. Cabe aclarar que el ABM tiene un potencial significativo para abordar problemas contemporáneos como el cambio climático y las dinámicas urbanas. Por ejemplo, puede ser utilizado para simular cómo diferentes políticas económicas afectan el comportamiento del consumidor o cómo las interacciones sociales influyen en la propagación de información durante una crisis sanitaria.

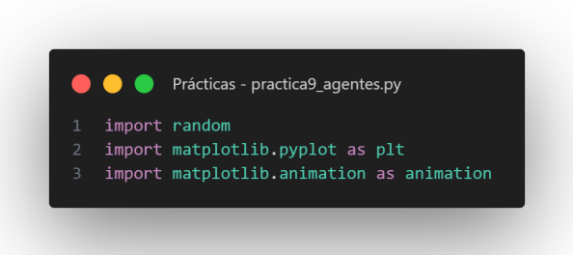
Silva (2024) destaca que una de las principales ventajas del ABM es su capacidad para modelar la complejidad y la heterogeneidad de los sistemas del mundo real, lo que lo hace ideal para estudiar fenómenos que no se pueden abordar fácilmente con otros enfoques de modelado. Sin embargo, el ABM también presenta desafíos, como la calibración y validación de modelos, la interpretación de resultados y la representación adecuada de la incertidumbre en los modelos.

Es así, que en la siguiente práctica se implementará un algoritmo de modelado basado en agentes relacionado al flujo de dinero de un grupo de personas, el algoritmo original se encuentra en Netlogo, por lo cual se requiere realizar la adaptación a Python. Para desarrollar este modelo, emplearemos librerías adicionales como Numpy, Matplotlib, entre otras, que facilitarán la carga, procesamiento y visualización de los resultados.

Contenido de la Actividad

El presente código implementa una simulación económica basada en agentes que modela el flujo de dinero en un grupo de personas (agentes) que interactúan entre sí y con el banco. En esta simulación, los agentes pueden intercambiar dinero, ahorrar, o solicitar préstamos, lo que permite observar cómo los estados económicos (ricos, clase media, pobres) se desarrollan a lo largo del tiempo. El banco administra las reservas y la disponibilidad de fondos para préstamos. Se utiliza `matplotlib.animation` para visualizar la simulación en tiempo real, mostrando el movimiento y el cambio de clase económica de los agentes a lo largo de un máximo de 200 pasos.

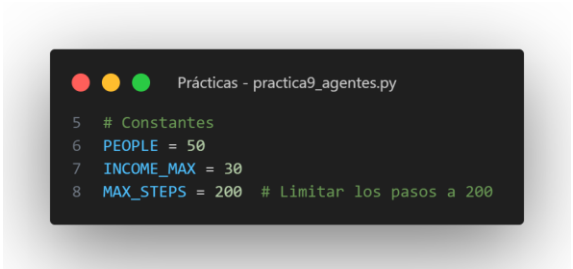
Importación de bibliotecas para tratamiento de datos



```
Prácticas - practica9_agentes.py
1 import random
2 import matplotlib.pyplot as plt
3 import matplotlib.animation as animation
```

- random: proporciona funciones para generar números aleatorios, usados aquí para simular la variabilidad de ingresos y de comportamiento de los agentes.
- matplotlib.pyplot: permite crear gráficos y visualizaciones.
- matplotlib.animation: proporciona herramientas para crear animaciones, que se usan para mostrar el progreso de la simulación en tiempo real.

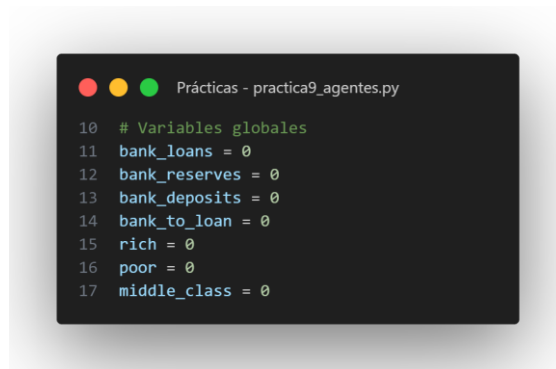
Definición de constantes y variables globales



```
Prácticas - practica9_agentes.py
5 # Constantes
6 PEOPLE = 50
7 INCOME_MAX = 30
8 MAX_STEPS = 200 # Limitar los pasos a 200
```

- PEOPLE: el número total de agentes en la simulación.
- INCOME_MAX: el nivel máximo de ingresos que define quién es "rico" en el sistema.

- MAX_STEPS: el número máximo de pasos de la simulación, definido en 200.

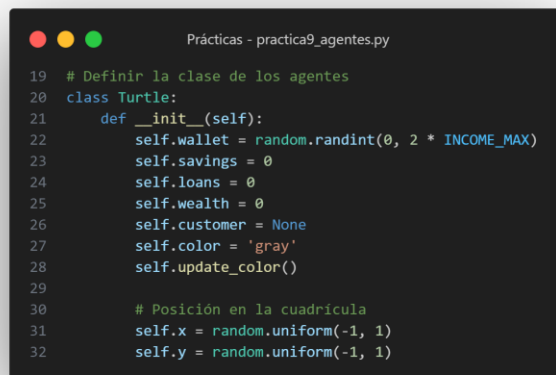


```
Prácticas - practica9_agentes.py

10 # Variables globales
11 bank_loans = 0
12 bank_reserves = 0
13 bank_deposits = 0
14 bank_to_loan = 0
15 rich = 0
16 poor = 0
17 middle_class = 0
```

Estas variables globales representan el estado económico del banco y el conteo de agentes en diferentes categorías socioeconómicas.

Definición de la clase Turtle (Agente)



```
Prácticas - practica9_agentes.py

19 # Definir la clase de los agentes
20 class Turtle:
21     def __init__(self):
22         self.wallet = random.randint(0, 2 * INCOME_MAX)
23         self.savings = 0
24         self.loans = 0
25         self.wealth = 0
26         self.customer = None
27         self.color = 'gray'
28         self.update_color()
29
30     # Posición en la cuadrícula
31     self.x = random.uniform(-1, 1)
32     self.y = random.uniform(-1, 1)
```

- wallet: representa el dinero disponible para el agente en efectivo.
- savings: ahorros del agente en el banco.
- loans: préstamos que el agente tiene pendientes.
- wealth: patrimonio del agente, calculado como savings - loans.
- customer: otro agente con quien el agente actual podría interactuar.
- color: color que representa el estado económico del agente (green para ricos, red para pobres, gray para clase media).
- x y y: coordenadas de posición del agente en la visualización.

```

Prácticas - practica9_agentes.py

34 def update_color(self):
35     if self.savings > INCOME_MAX:
36         self.color = 'green' # Verde para los ricos
37     elif self.loans > INCOME_MAX:
38         self.color = 'red' # Rojo para los pobres
39     else:
40         self.color = 'gray' # Gris para la clase media
41     self.wealth = self.savings - self.loans

```

Actualiza el color del agente según su categoría económica basada en savings y loans.

```

Prácticas - practica9_agentes.py

43 def balance_books(self):
44     global bank_to_loan
45     if self.wallet < 0:
46         if self.savings >= -self.wallet:
47             self.withdraw_from_savings(-self.wallet)
48         else:
49             self.withdraw_from_savings(self.savings)
50             temp_loan = bank_to_loan
51             if temp_loan >= -self.wallet:
52                 self.take_out_a_loan(-self.wallet)
53             else:
54                 self.take_out_a_loan(temp_loan)
55     else:
56         self.deposit_to_savings(self.wallet)
57
58     # Repagar préstamos si hay suficientes ahorros
59     if self.loans > 0 and self.savings > 0:
60         if self.savings >= self.loans:
61             self.repay_a_loan(self.loans)
62         else:
63             self.repay_a_loan(self.savings)
64

```

Administra el saldo del agente. Si el wallet es negativo, se intentará cubrir el déficit con ahorros o solicitando un préstamo al banco.

```

Prácticas - practica9_agentes.py

65 def do_business(self, other):
66     if ((self.savings > 0 or self.wallet > 0 or bank_to_loan > 0) and
67         random.random() < 0.5): # 50% de probabilidad de comercio
68         amount = 5 if random.random() < 0.5 else 2
69         self.wallet -= amount
70         other.wallet += amount

```

Simula una transacción de negocio entre el agente y otro. Existe un 50% de probabilidad de que realicen una transacción, en la cual el agente actual reduce su wallet y el otro agente incrementa el suyo en una cantidad aleatoria de 2 o 5.

```

Prácticas - practica9_agentes.py

72 def deposit_to_savings(self, amount):
73     self.wallet -= amount
74     self.savings += amount
75
76 def withdraw_from_savings(self, amount):
77     self.wallet += amount
78     self.savings -= amount
79
80 def repay_a_loan(self, amount):
81     global bank_to_loan
82     self.loans -= amount
83     self.wallet -= amount
84     bank_to_loan += amount
85
86 def take_out_a_loan(self, amount):
87     global bank_to_loan
88     self.loans += amount
89     self.wallet += amount
90     bank_to_loan -= amount

```

- deposit to savings: deposita una cantidad de wallet en savings.
- withdraw from savings: retira una cantidad de savings para cubrir déficits de wallet.
- repay a loan: reduce la deuda pendiente (loans) del agente y ajusta el wallet.
- take out a loan: solicita un préstamo al banco, incrementando wallet y actualizando bank_to_loan.

Inicialización de los agentes y configuración del banco

```

Prácticas - practica9_agentes.py

92 # Inicializar los turtles
93 turtles = [Turtle() for _ in range(PEOPLE)]

```

Crea una lista de agentes (turtles) usando la clase Turtle.

```

Prácticas - practica9_agentes.py

95 # Funciones de manejo bancario
96 def setup_bank():
97     global bank_loans, bank_reserves, bank_deposits, bank_to_loan
98     bank_loans = sum(t.loans for t in turtles)
99     bank_deposits = sum(t.savings for t in turtles)
100     bank_reserves = 0.1 * bank_deposits # Ejemplo de una reserva del 10%
101     bank_to_loan = bank_deposits - (bank_reserves + bank_loans)

```

Calcula y actualiza el estado financiero del banco según los ahorros, préstamos y reservas de los agentes.

Conteo de clases económicas

```
Prácticas - practica9_agentes.py

103 def poll_class():
104     global rich, poor, middle_class
105     rich = sum(1 for t in turtles if t.savings > INCOME_MAX)
106     poor = sum(1 for t in turtles if t.loans > INCOME_MAX)
107     middle_class = PEOPLE - (rich + poor)
```

Cuenta cuántos agentes están en cada categoría económica (rich, poor, middle_class) y actualiza los valores globales.

Configuración de la animación

```
Prácticas - practica9_agentes.py

109 # Animación
110 fig, ax = plt.subplots()
111 sc = ax.scatter([t.x for t in turtles], [t.y for t in turtles], c=[t.color for t in turtles])
```

Crea una figura (fig) y un área de dibujo (ax) para la animación, donde los puntos representan a los agentes en función de sus posiciones y colores.

```
Prácticas - practica9_agentes.py

113 # Configuración de título y leyenda(simbología)
114 ax.set_title('Simulación Económica y Bancaria')
115 rich_patch = plt.Line2D([0], [0], marker='o', color='w', label='Ricos', markerfacecolor='green', markersize=10)
116 middle_class_patch = plt.Line2D([0], [0], marker='o', color='w', label='Clase Media', markerfacecolor='gray', markersize=10)
117 poor_patch = plt.Line2D([0], [0], marker='o', color='w', label='Pobres', markerfacecolor='red', markersize=10)
118 ax.legend(handles=[rich_patch, middle_class_patch, poor_patch], loc='upper right')
```

Agrega título a la visualización. Configura una leyenda para representar visualmente cada clase económica (rich, middle_class, poor).

```
Prácticas - practica9_agentes.py

120 # Título dinámico
121 title = ax.set_title("Simulación Económica y Bancaria - Paso 0")
```

Se agrega un título dinámico que muestra el paso actual de la simulación.

```

Prácticas - practica9_agentes.py
123 # Texto para mostrar el conteo de clases
124 class_count_text = ax.text(0.05, 0.95, '', transform=ax.transAxes, fontsize=12, va='top', ha='left', bbox=dict(facecolor='white', alpha=0.7))

```

Añade un cuadro de texto para mostrar el conteo de cada clase económica (rich, middle_class, poor).

Función update para la animación

```

Prácticas - practica9_agentes.py
126 def update(frame):
127     if frame >= MAX_STEPS:
128         ani.event_source.stop() # Detener la animación después de 100 pasos
129         return
130
131     for t in turtles:
132         other = random.choice(turtles)
133         if other != t:
134             t.do_business(other)
135             t.balance_books()
136             t.update_color()
137
138     setup_bank()
139     poll_class()
140
141     # Actualizar dispersión de colores y posiciones
142     sc.set_offsets([[t.x, t.y] for t in turtles])
143     sc.set_color([t.color for t in turtles])
144
145     # Actualizar el título con el paso actual
146     title.set_text(f"Simulación Económica y Bancaria - Paso {frame+1}")
147
148     # Actualizar el conteo de clases en el texto
149     class_count_text.set_text(f"Ricos: {rich}\nClase Media: {middle_class}\nPobres: {poor}")

```

Define los cambios en cada cuadro de animación:

- Los agentes realizan transacciones (do_business).
- Cada agente ajusta su balance y color (balance_books y update_color).
- El estado financiero del banco y la clase económica de cada agente se actualizan (setup_bank, poll_class).
- Las clases, colores y el título de la animación se actualizan.

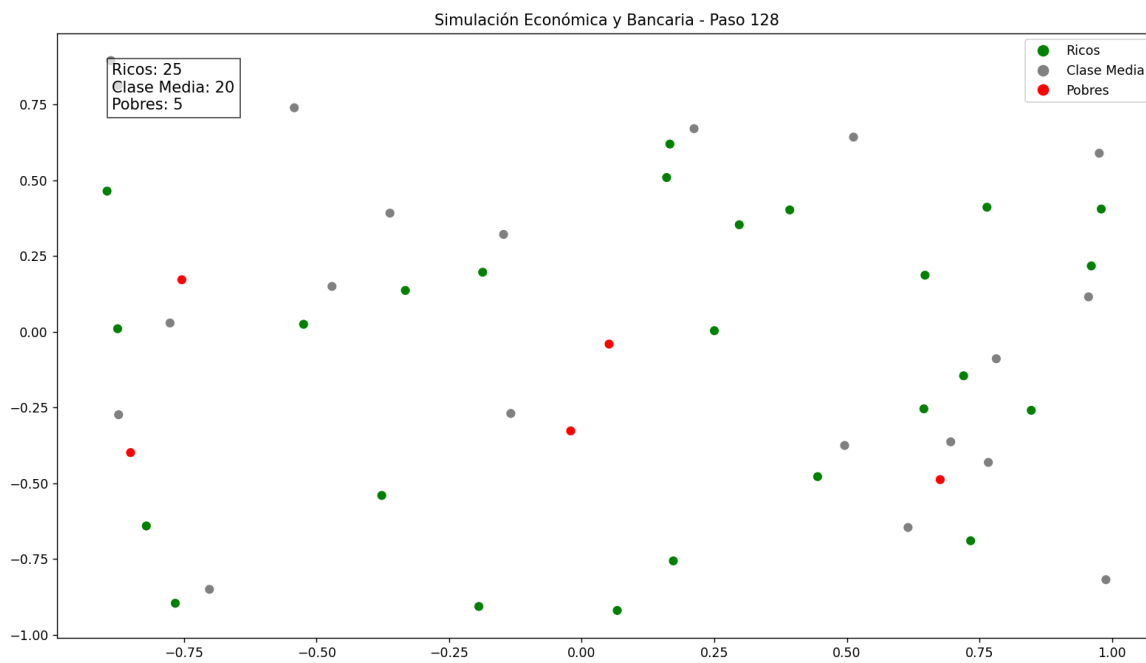
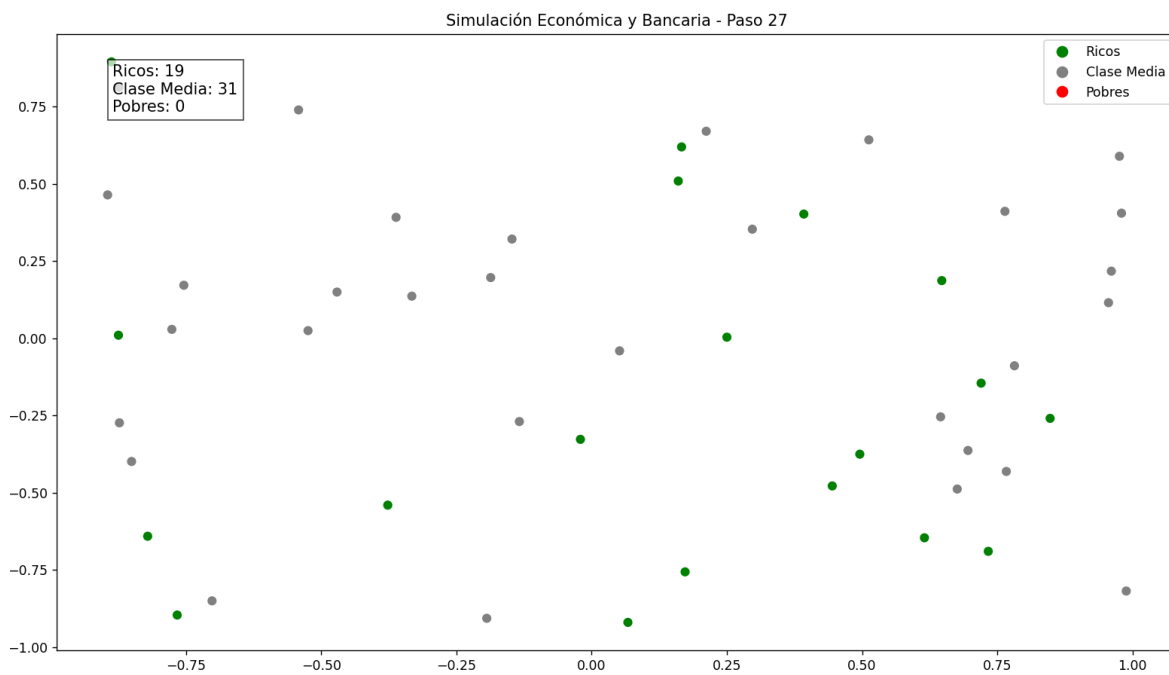
```

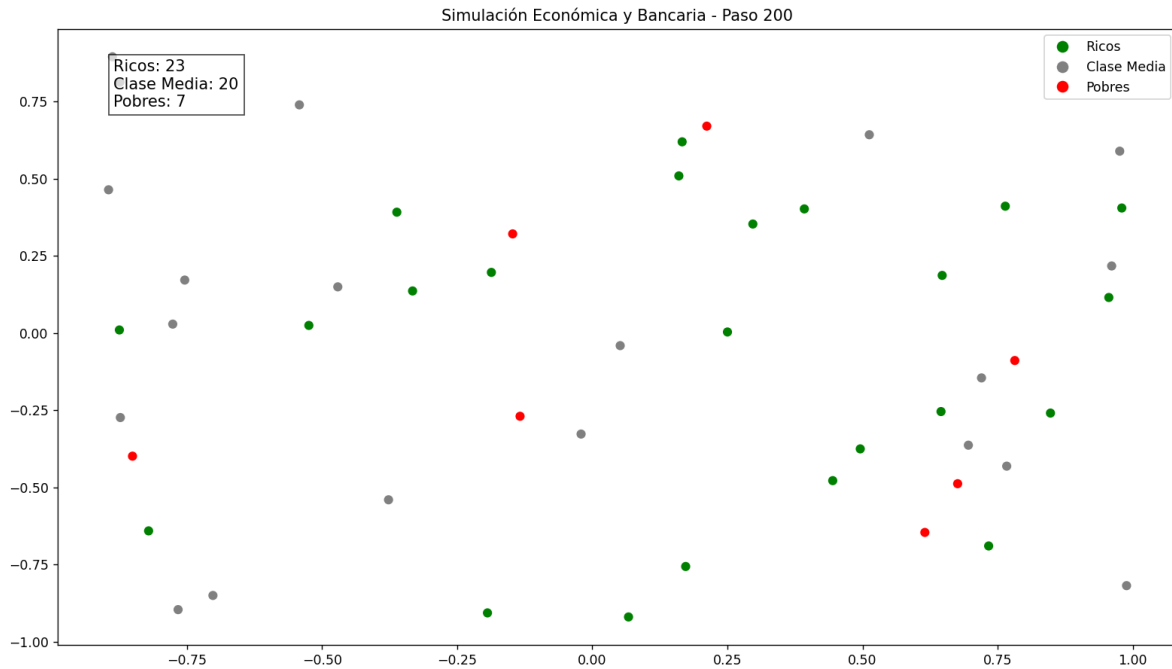
Prácticas - practica9_agentes.py
151 ani = animation.FuncAnimation(fig, update, frames=MAX_STEPS + 1, interval=200, blit=False)
152 plt.show()

```

Inicia la animación con FuncAnimation, que ejecuta la función update por cada cuadro hasta alcanzar MAX_STEPS y luego la muestra.

Obteniendo así los siguientes resultados:





Igualmente se agrega el enlace a un video donde se muestra el progreso de manera dinámica: <https://youtu.be/HyRHcADZ0WU>

Conclusiones

Al finalizar la práctica de modelado basada en agentes en Python enfocado a la simulación de la situación económica de un grupo o sector de personas, se ha demostrado que este tipo de modelado computacional es una herramienta potente para entender las interacciones económicas complejas entre individuos y su relación con una entidad bancaria. A través de la creación de agentes con comportamientos básicos (realizar transacciones, ahorrar, o pedir préstamos), fue posible observar cómo estas simples reglas generan dinámicas emergentes en la economía del sistema.

Durante el desarrollo del modelo, se subrayó la importancia de ajustar parámetros clave, como el número máximo de pasos en la simulación y el nivel máximo de ingresos, ya que estos factores afectan significativamente el comportamiento global del sistema y la representación de cada clase económica (ricos, clase media y pobres). La inclusión de un banco que controla préstamos y ahorros permitió simular el flujo financiero y observar cómo los agentes se mueven entre diferentes clases económicas en función de su capacidad para ahorrar o endeudarse.

Las visualizaciones dinámicas facilitaron la interpretación de los resultados en tiempo real, proporcionando una representación clara de las interacciones económicas entre agentes y el cambio en sus estados financieros. Las actualizaciones del estado de cada agente permitieron observar cómo se agrupan o distribuyen los agentes en categorías económicas, lo que resulta útil para analizar las disparidades y los efectos de préstamos y ahorros en el sistema.

En resumen, esta práctica ha demostrado que el modelado basado en agentes es un enfoque robusto para estudiar interacciones económicas y patrones emergentes en un sistema complejo. La experiencia adquirida en el ajuste de parámetros y visualización de resultados ha sido fundamental para comprender mejor los principios de simulación y modelado basado en agentes. Herramientas como random, matplotlib, y FuncAnimation de Python han sido esenciales para el desarrollo y visualización del modelo, subrayando su utilidad en el análisis de simulaciones y comportamientos de sistemas complejos en tiempo real.

Referencias

- Silva, F. (2024, 9 de mayo). *Modelo Basado en Agentes (ABM)*. LinkedIn. Recuperado el 01 de Noviembre de 2024 de: <https://www.linkedin.com/pulse/modelo-basado-en-agentes-abm-fredy-silva-o--xfe3e/>