**National University of the Altiplano**
**Faculty of Statistical and Computer Engineering**
**Instructor:** Fred Torres Cruz
**Author:** Eva Ruth Mamani Jose

**Assigned Work - No. 009**

# Project Report: Predator-Prey Model

## 1. Introduction

This project develops a computational simulation based on the classic ecological problem of the predator-prey model. In this simulation, two types of agents - sheep (prey) and wolves (predators) - interact in a two-dimensional environment, allowing the study of population dynamics and emergent behaviors.

The objective is to observe the evolution of both populations over time, visualizing their spatial distribution and the change in the number of agents throughout the simulation.

## 2. Application Description

The application is developed using **Mesa**, a Python framework for creating agent-based simulations. The simulation contains:

- **Agents:**
  - **Sheep:** Prey that move and can reproduce.
  - **Wolves:** Predators that hunt sheep to survive.
- **Environment:** A two-dimensional grid where agents are located and move.
- **Key mechanisms:**
  - Agents have a life cycle and are updated at each simulation step.
  - Wolves prey on sheep, affecting the population.
  - The number of sheep and wolves is recorded at each step for analysis.

## 3. Code Structure

### 3.1. File model.py

```
from mesa import Model
from mesa.space import MultiGrid
from mesa.time import RandomActivation
from mesa.datacollection import DataCollector
from agents import Sheep, Wolf


class PredatorPrey(Model):
    def __init__(self, width=20, height=20, initial_sheep=50,
        initial_wolves=10):
        self.num_agents = initial_sheep + initial_wolves
        self.grid = MultiGrid(width, height, True)
        self.schedule = RandomActivation(self)
        self.running = True
        self.current_id = 0  # Line added to avoid next_id() error

        self.datacollector = DataCollector(
            model_reporters={
                "Sheep": lambda m: self.count_type(Sheep),
                "Wolves": lambda m: self.count_type(Wolf)
            }
        )

        # Create sheep
        for _ in range(initial_sheep):
            sheep = Sheep(self.next_id(), self)
            self.schedule.add(sheep)
            x, y = self.grid.find_empty()
            self.grid.place_agent(sheep, (x, y))

        # Create wolves
        for _ in range(initial_wolves):
            wolf = Wolf(self.next_id(), self)
            self.schedule.add(wolf)
            x, y = self.grid.find_empty()
            self.grid.place_agent(wolf, (x, y))

        self.datacollector.collect(self)

    def step(self):
        self.schedule.step()
        self.datacollector.collect(self)

    def count_type(self, agent_type):
        return sum(isinstance(a, agent_type) for a in self.schedule.agents
            )
```

## 3.2. File agents.py

```
from mesa import Agent
import random
```

```python


class Sheep(Agent):
    def step(self):
        self.move()

    def move(self):
        possible_steps = self.model.grid.get_neighborhood(self.pos, moore=
           True, include_center=False)
        new_position = random.choice(possible_steps)
        self.model.grid.move_agent(self, new_position)


class Wolf(Agent):
    def step(self):
        self.move()
        self.eat()

    def move(self):
        possible_steps = self.model.grid.get_neighborhood(self.pos, moore=
           True, include_center=False)
        new_position = random.choice(possible_steps)
        self.model.grid.move_agent(self, new_position)

    def eat(self):
        cellmates = self.model.grid.get_cell_list_contents([self.pos])
        sheep = [agent for agent in cellmates if isinstance(agent, Sheep)]
        if sheep:
            victim = random.choice(sheep)
            self.model.grid.remove_agent(victim)
            self.model.schedule.remove(victim)
```

## 3.3. File visualization.py

```python
from mesa.visualization.modules import CanvasGrid, ChartModule
from mesa.visualization.ModularVisualization import ModularServer
from model import PredatorPrey
from agents import Sheep, Wolf

def agent_portrayal(agent):
    if isinstance(agent, Sheep):
        return {
            "Shape": "resources/sheep.png",  # Sheep image
            "scale": 0.9,
            "Layer": 0
        }
    elif isinstance(agent, Wolf):
        return {
            "Shape": "resources/wolf.png",  # Wolf image
            "scale": 0.9,
            "Layer": 1
        }
```

```python

grid = CanvasGrid(agent_portrayal, 20, 20, 500, 500)

chart = ChartModule([
    {"Label": "Sheep", "Color": "blue"},
    {"Label": "Wolves", "Color": "red"}
])

server = ModularServer(
    PredatorPrey,
    [grid, chart],
    "Predator-Prey Model",
    {"width": 20, "height": 20, "initial_sheep": 50, "initial_wolves": 10}
)
```

## 3.4. Complementary files

### 3.4.1. requirements.txt

```
mesa
```

### 3.4.2. server.py

```python
from visualization import server

server.launch()
```

## 4. Application Functionality

1. At startup, a defined number of sheep and wolves are placed at random positions within the grid.
2. Each simulation step represents a cycle where all agents act:
   - Sheep move and can reproduce.
   - Wolves move, hunt sheep to feed, and can reproduce if they survive.
3. The visualization shows:
   - Agents on the grid: sheep with white images and wolves with red images.
   - A chart that dynamically updates the number of sheep and wolves as the simulation progresses.
4. The model allows observation of ecological phenomena such as population fluctuations, extinction, or coexistence.

## 5. Utility and Applications

- **Education:** Facilitates understanding of ecological concepts and population dynamics.
- **Research:** Can be adapted to explore different scenarios and parameters in complex systems studies.
- **Simulation:** Provides a platform to experiment with species interactions in a controlled environment.

## 6. Conclusions

The simulation created with Mesa demonstrates how simple interactions between agents can generate complex and dynamic patterns. The combination of graphical visualization and data analysis facilitates result interpretation and learning.

## 7. Git Repository

The complete source code of the project is available at:
`https://github.com/evaruth270/lobo_oveja.git`