Actividad 5 - El sistema Predador-Presa

Exactas Programa

Invierno 2019

Los modelos de depredación y competencia forman parte de la batería de herramientas clásicas del ecólogo. Vito Volterra en Italia y Alfred Lotka en Estados Unidos fueron los precursores en este tema y crearon los modelos que, con diversas modificaciones y mejoras, seguimos usando hoy en día. El modelo de Volterra para depredación comienza suponiendo la existencia de dos poblaciones de animales, una de las cuales (el depredador) se alimenta de la otra (la presa). Se supone que las dos poblaciones están formadas por individuos idénticos, mezclados en el espacio.

La gran mayoría de los modelos estudiados tienen en cuenta principalmente la dinámica temporal de estos sistemas. Esta actividad tiene como objetivo implementar una versión diferente de estos modelos. Intentaremos analizar tanto el aspecto temporal como espacial del modelo. La idea es recrear un mundo con depredadores (los llamaremos Leones y los identificaremos con la letra `L`) y presas (las llamaremos Antílopes y los identificaremos con la letra `A`).

El mundo en el cual los leones y los antílopes van a interactuar será bidimensional (exactamente como un tablero). Para saber dónde está cada individuo, cada posición del tablero será identificada por sus coordenadas (fila y columna).

El mundo que utilizaremos será un valle (rectangular) rodeado de montañas. Esto lo representaremos en un tablero en cuyos bordes pondremos la letra `M` para representarlas. De tal manera, si queremos que los animales se muevan en un medio ambiente de tamaño $j \times k$, con j filas y k columnas, agregaremos dos filas y dos columnas en la construcción del tablero para incluir el borde.

Vamos a seguir utilizando `NumPy` ya que es un módulo que le otorga soporte muy eficiente para operaciones con listas y matrices a `Python` utilizándose habitualmente en aplicaciones de simulación numérica.

Vamos a comenzar probando un poco el funcionamiento de `NumPy`:

1. Importe el paquete `NumPy` y renómbrelo con el siguiente comando:

```
import numpy as np
```

   Recuerde que esta línea además de importar, permite referirse al módulo `NumPy` de una manera más resumida (como si fuera un sobrenombre). Es algo que se hace habitualmente en el mundo `Python`.

2. Defina un *tablero* conteniendo espacios por medio de un `array` (empecemos probando con un medio ambiente de tamaño $j = 5$ y $k = 7$, para eso armemos entonces un tablero con 7×9 posiciones) y diciéndole que tenga forma de matriz:

```
t = np.repeat(" ", 7*9).reshape(7, 9)

# podemos verlo con:
print(t)
# podemos acceder a un elemento individual por su fila y columna:
print("La posicion (1,2) tiene el elemento:",t[(1,2)])
# Como el tablero tiene dos dimensiones, podemos accederlas como:
print("Mi tablero tiene", t.shape[0], "filas y", t.shape[1], "columnas.")
```

   ¿Qué pasaría si al crear el tablero en lugar de `" "`, ponemos `"L"`, o `"Sanguche"`?

3. Asigne el valor `M` a los *bordes* de `tablero`.

4. Vamos a poblar el tablero. Ubique 4(cuatro) antílopes ("A"), uno en cada una de las siguientes posiciones: $(1,3)$, $(2,1)$, $(3,1)$ y $(3,3)$ del `tablero`. Luego, agregue 1(un) león ("L") en el posición $(1,2)$.

```
# definimos la coordenadas de nuestros personajes
fil = [1, 2, 3, 3, 1]
col = [3, 1, 1, 3, 2]
tipo = ["A", "A", "A", "A", "L"]
# y ahora los asignamos dentro del tablero

for i in range(len(tipo)):
    t[(fil[i], col[i])] = ___COMPLETAR___

# veamos como queda:
print(t)
```

5. Para que nuestro mundo cobre vida, realizaremos funciones que nos permitirán recrear su dinámica. Empezaremos con una función que permirtirá a nuestros personajes identificar las coordenadas de su entorno cercano.

   Implementar una función `mis_vecinos(coord_centro)` que tome como entrada las coordenadas de una posición del tablero (llamada `coord_centro`), y devuelva una lista con las coordenadas de sus posiciones vecinas, en el orden determinado por el siguiente esquema:

$$\begin{bmatrix} 1 & 2 & 3 \\ 8 & coord\_centro & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

   Este esquema representa recorrer los vecinos en el orden de las agujas del reloj.

6. Implementar una función `buscar_adyacente(tablero, coord_centro, objetivo)`. La función toma como parámetros un tablero (`tablero`), la coordenada del centro de búsqueda (`coord_centro`), qué buscar (`objetivo` que puede ser `L`, `A` o un espacio en blanco `" "`). La función devuelve una lista con la coordenada vecina donde se encuentra el objetivo buscado, y en caso de no existir devolverá una lista vacía.

   Por ejemplo, si tenemos en la variable `t` al tablero:

"M"	"M"	"M"	"M"	"M"
"M"	" "	"L"	"A"	"M"
"M"	"A"	" "	" "	"M"
"M"	"A"	" "	"A"	"M"
"M"	"M"	"M"	"M"	"M"

Cuadro 1: Ejemplo de mundo

   y le preguntaramos:

   - `buscar_adyacente(t, (1, 1), "L")`, nos debe dar como respuesta: `[(1, 2)]`.
   - `buscar_adyacente(t, (1, 1), "A")`, nos debe dar como respuesta: `[(2, 1)]`.
   - `buscar_adyacente(t, (1, 1), " ")`, nos debe dar como respuesta: `[(2, 2)]`.
   - `buscar_adyacente(t, (2, 2), "A")`, nos debe dar como respuesta: `[(1, 3)]`.
   - `buscar_adyacente(t, (3, 3), "L")`, nos debe dar como respuesta: `[]`.

Ahora, vamos a implementar cada una de las fases involucradas en la dinámica descripta durante la clase. En adelante, cada vez que necesitemos recorrer todo el `tablero` *activo* (no se recorren los bordes), lo haremos por filas, empezando desde el extremo izquierdo superior `(1,1)`.

```
n_fila = t.shape[0]
n_col = t.shape[1]
for i in range(1, n_fila - 1):
    for j in range(1, n_col - 1):
        ___COMPLETAR___
```

7. `fase_mover(tablero)`: implementa la fase de movimiento del tablero recorriendo cada casillero *activo* del mismo. En cada paso, verifica si la posición (i, j) elegida está ocupada por un depredador o presa. En caso de estar ocupada, buscamos el primer vecino desocupado y allí desplazamos al individuo en cuestión.

8. `fase_alimentacion(tablero)`: implementa la fase alimentación recorriendo cada casillero *activo* del `tablero`. En cada paso, verifica si la posición `(i,j)` elegida está ocupada por un león y, en tal caso, busca al primer antílope disponible en la vecindad. Si lo encuentra, el león se mueve a la posición ocupada por el antílope, y lo devora (el antílope desaparece del tablero y aparece en la panza del león).

   Veamos el ejemplo inicial y cómo queda luego de una fase de alimentación:

"M"	"M"	"M"	"M"	"M"
"M"	" "	"L"	"A"	"M"
"M"	"A"	" "	" "	"M"
"M"	"A"	" "	"A"	"M"
"M"	"M"	"M"	"M"	"M"

Estado inicial

"M"	"M"	"M"	"M"	"M"
"M"	" "	" "	"L"	"M"
"M"	"A"	" "	" "	"M"
"M"	"A"	" "	"A"	"M"
"M"	"M"	"M"	"M"	"M"

Tablero luego de la fase de alimentación

9. `fase_reproduccion(tablero)`: todas las especies tiene su etapa de reproducción. Esta función tiene como entrada un tablero `tablero`, y ejecuta la fase de reproducción recorriendo cada casillero *activo* del mismo. En cada paso verifica si la posición `(i,j)` elegida está ocupada. En caso de estarlo, busca en la vecindad si hay otro individuo de su misma especie y, si lo encuentra, busca la primer posición vacía y hace aparecer en esa posición a un nuevo individuo de la misma especie (león o antílope, según corresponda).

10. `evolucionar(tablero)`: cada paso de evolución de nuestro mundo está formado por una fase de alimentación, una de reproducción y una de movimiento, en ese orden.

11. `evolucionar_en_el_tiempo(tablero, tiempo_limite)`: esta función realiza tantos ciclos de evolución como indica el parámetro `tiempo_limite`.

12. Contruya un mundo de 4×6 casilleros activos con tres antílopes en las posiciones $(2, 2)$, $(3, 3)$ y $(4, 3)$ y dos leones en los casilleros $(4, 5)$ y $(2, 5)$.

    Simule la evolución del sistema en un ciclo. Debería encontrarse en la siguiente situación:

"M"	"M"	"M"	"M"	"M"	"M"	"M"	"M"
"M"	"A"	"A"	" "	"L"	"A"	" "	"M"
"M"	"A"	"A"	" "	" "	" "	" "	"M"
"M"	" "	"A"	" "	"L"	" "	" "	"M"
"M"	" "	" "	" "	" "	" "	" "	"M"
"M"	"M"	"M"	"M"	"M"	"M"	"M"	"M"

Cuadro 2: Mundo simulado después de un ciclo.

   Indique la cantidad de antílopes presentes al finalizar los dos ciclos.

13. Estudie si los antílopes se extinguen durante los primeros 10 ciclos, y en tal caso, indique en qué ciclo se produce la extinción.

14. Ya tenemos toda la dinámica programada. Vamos a probarla con un tablero más grande y con distintas cantidades de leones y antílopes ubicados al azar.

    Hay muchas formas de realizar la asignación inicial al azar de casillas. Un ejemplo es el siguiente:

```
def mezclar_celdas(tablero):
    celdas = []

    for i in range (1 , ___COMPLETAR___) :
       for j in range (1 , ___COMPLETAR___) :
            celdas.append((i, j))

    # Ahora las mezclamos
    random.shuffle(celdas)

    return celdas
```

    Cada vez que llamemos a `mezclar_celdas` nos devolverá un orden distinto. Para completar el tablero, podemos poblarlo en el orden en que vienen, es decir, primero ubicamos los antílopes y luego los leones.

    Realizar la función `generar_tablero_azar(filas, columnas, n_antilopes, n_leones)`, que toma las cantidades expresadas y devuelve un tablero al azar siguiendo las especificaciones. Probar la función anterior generando un tablero de 10×10 con cinco leones y diez antílopes. Tomar en cuenta que las dimensiones del tablero deben ser las dimensiones *activas*, es decir, el tablero generado debe tener 10 filas y 10 columnas de valle, sin contar las montañas.

15. `cuantos_de_cada(tablero)`: devuelve una lista con dos elementos el primero corresponde a la cantidad total de antílopes que hay en el tablero y el segundo a la cantidad de leones. Por ejemplo: $[14, 3]$, 14 antílopes, y 3 leones.

16. `registrar_evolucion(tablero, tiempo_limite)`: es parecida a la función `evolucionar_en_el_tiempo` que implementamos en el item 11. Queremos, además de hacer evolucionar el tablero, registrar cada paso de dicha evolución. Usaremos la función `cuantos_de_cada` y en cada paso de tiempo registraremos la cantidad total de individuos de cada especie. De esta forma sabremos cuántos antílopes y leones hay vivos en cada paso de la evolución.

    Esta función debe armar una lista con los resultados de la función `cuantos_de_cada`. Con tres pasos, tendrán algo parecido a: $[[5, 1], [3, 1], [7, 1]]$.

17. Vamos a guardar el resultado de la función `registrar_evolucion(tablero, tiempo_limite)` en un archivo llamado `predpres.csv`. Editamos la función anterior y ahora en lugar de devolver una lista, únicamente guarda los resultados en un archivo de texto.

    En el siguiente ejemplo se muestra cómo guardar el contenido de la lista de la evolución de individuos que está almacenada en la variable `evolucion_especies`:

```
import csv

evolucion_especies = ___COMPLETAR___

with open("predpres.csv", "w", newline="") as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(["antilopes", "leones"])
    csv_writer.writerows(evolucion_especies)
```

18. Vamos a visualizar la evolución del paso anterior. Es muy importante que todas las funciones tengan los nombres que nosotros les dimos en esta guía. Nombre a su archivo: `predpresa.py`.

    Descomprimir el archivo `visualizacion.zip` en la misma carpeta que su proyecto.

    Crear un nuevo archivo `Python` con el siguiente contenido:

```
import visualizacion
import predpresa

visualizacion.simular(7, 5, 3, 2, 3)

visualizacion.simular(4, 3, 5, 2, 3)
```

    Ejecutar dicho archivo.

    Probar hacer otras simulaciones. La función `simular` toma 5 parámetros: La cantidad de filas y columnas (*activas*, en ambos casos) del tablero, la cantidad de antílopes y leones iniciales, y la cantidad de ciclos de la evolución.

    Requisito La visualización está implementada basada en el módulo `pygame` de `Python`. Si no está instalado, antes de correrla deberá instalarla. Para ello, en GNU+Linux abrir una terminal o en Windows la aplicacion `Anaconda Prompt`; y ahí escribir `pip install pygame` y apretar [Enter]. La descarga e instalación dura menos de dos minutos.

19. **Optativo+** `buscar_adyacente_aleatoria(tablero, coord_centro, objetivo)`: modificar la función para que devuelva de manera aleatoria una posición posible que cumpla con el criterio de búsqueda. Actualizar las funciones de las distintas etapas para que utilice esta nueva función.

20. **Optativo ++** Para hacer la representación de nuestro mundo más interesante y realista, vamos a suponer que es un toroide. Esto significa que al desplazarse más allá del la última columna *activa* de la derecha, el individuo aparecería en la primera columna *activa* de la izquierda. Lo mismo ocurre entre la última y la primer fila del tablero. Deberá modificar la generación del tablero para que las casillas de borde marcadas con `M` no estén más y adecuar la función `mis_vecinos` para que tome en cuenta la nueva geometría del mundo.

    Pista: El resto de dividir por la cantidad de filas o columnas daría una forma de *dar la vuelta* al mundo, para pensarlo.