Universidad Nacional de General Sarmiento

# Instructivo de uso para simulación por modelo basado en agentes de evacuación de una sala de cine ante incendio. (MBA).

Licenciatura en Sistemas

Práctica Profesional Supervisada


**Docentes**

Enrique Fernandez

**Alumnos**

Sebastian Pintos spintos100@gmail.com

Luis Curto luisenriquecurto@gmail.com

# Índice

# 1. Introducción

Esta simulación de una evacuación es un trabajo que explora el uso de herramientas combinadas para obtener datos de un Modelo Basado en Agentes (MBA) realizado con NetLogo. La propuesta de este informe es servir de instructivo para su uso y profundización de lo realizado. Se desea explicar cómo está resuelta la propuesta en sus diferentes lenguajes de programación para que aquellos que deseen puedan implementar nuevas soluciones a partir de lo realizado. A grandes rasgos dividimos la explicación en la simulación realizada sobre NetLogo, la aplicación en JAVA que nos permite ejecutar las diferentes experiencias planteadas y obtener sus datos por medio de una base de datos en Postgres, la aplicación en Python que accede a estos datos y nos da información en gráficas de lo acontecido en las experiencias.

# 2. Descripción de la simulación y su código.

NetLogo es un lenguaje de programación y una plataforma para el diseño de modelos basados en agentes. Tiene licencia GLP por lo que es ideal para ámbitos académicos. En esta sección lo que haremos es desmenuzar el código realizado para la simulación de modo que quien quiera hacer modificaciones tenga facilitado el aprendizaje.

### *a. Definición de agentes y sus variables.*

```
breed
[
  people person
]
```

Una manera fácil de definir a los agentes es que son seres que pueden seguir instrucciones y a los cuales se les puede establecer variables. Se dividen en tortugas, parcelas, enlaces y el observador. La definición de tortugas es por medio de la palabra clave “breed” que nos permite crear familias de tortugas.

```
globals
[
  scape
  death
  final-ticks
  collisions
]
people-own
[
  health
  direction
  target
]
patches-own [
  flame
  is-fire?
]
```

En la definición de variables existe la división entre las que son globales y atañen a la simulación y aquellas que son propias de cada tipo de agente.

OBJ

### *b. Procedimientos.*

Existen dos procedimientos que pueden ser llamados desde la consola de instrucciones que pueden ser "setup" y "go", los cuales se utilizan para configurar las condiciones de inicio y correr la simulación. Estos pueden a su vez llamar a otros procedimientos que nos permitirán dar instrucciones a los agentes o modificar su estado.

```
to go
   ask people [
    ;; if a door is in front of person, scape
    if any? doors with [distance myself < 1.5]
    [
      set scape (scape + 1)
      die ]
    ;; Procedures to move, receive damage from fire and die if necessary
    walk
    eat-flame
    maybe-die

  ]
  ;; Save number of ticks when all population has scaped or died. Helpful to stop run in Java.
  if (death + scape) = population and final-ticks = 0 [set final-ticks ticks]

  ;; Spread flame
  diffuse flame 0.8

  ask patches [ spread-flame ]
  tick
end
```

Arriba vemos cómo definimos el procedimiento “go” que nos permite dar instrucciones por medio de la palabra reservada “ask” a los agentes definidos. Los procedimientos definidos para las tortugas son “walk”, “eat-flame” y “maybe-die”. En el procedimiento “setup” más abajo lo que veremos es que se definen las condiciones iniciales. Y creamos agentes puertas definiendo las variables dentro de este procedimiento, además de la población de agentes o tortugas “person”. Esto se puede hacer de otra manera pero como decidimos tener diferentes experimentos en diferentes archivos, para nosotros fue la manera más fácil de modificar las condiciones del experimento. Aquí vemos que el lenguaje que es base de NetLogo, Logo, nos da la flexibilidad necesaria para modificar sin problemas la estructura del código y adaptarlo a las necesidades.




```
to setup
  clear-all
  setup-center
  set-default-shape doors "door"
  create-ordered-doors 1
    [
    set xcor -20
    set ycor 8
    set size 10
  ]
  create-doors 1
    [
    set xcor 20
    set ycor 8
    set size 10
  ]
  create-ordered-doors 1
    [
    set xcor 20
    set ycor -8
    set size 10
  ]
  create-ordered-doors 1
    [
    set xcor -20
    set ycor -8
    set size 10
  ]
  ;; Create people
  create-people population
  [
    set shape "person"
    set health 1
    setxy random-xcor random-ycor

    ;; Change position of person until color is black (no scenario or seats)
    while [pcolor != black] [
      setxy random-xcor random-ycor
    ]
    ;; Make a person face a door
    set target one-of doors
    face target
  ]
```

```
  ask patches [
    set flame 0
    set is-fire? false
  ]
  create-fire
  ask patches [ spread-flame ]
  reset-ticks
end
```

Un último ejemplo de los procedimientos es el que utilizamos para el movimiento de las personas en la sala, este ejemplo es para mostrar cómo dentro de los mismos existe también una relación entre los diferentes tipos de agentes, ya que el procedimiento que modifica el comportamiento de las tortugas está asociado a la observación de los estados de las parcelas.

```
to walk
  ;; If a "candidate" is found in front of a person there is a collision and must stop until it
  let candidate one-of people-at 1 0
  if candidate != nobody [
    set collisions (collisions + 1)
    stop
  ]

  ;; Face closest door
  set target min-one-of doors [ distance myself ]
  face target
  ;; If cannot move because scenary or seats, rotate direction
  while [can-move? 1 and [pcolor] of patch-ahead 1 = blue] [
     rt (90 + random 90)
    ]
  rt random 20
  fd 1
end
```

***c. Procedimiento de dibujo del espacio por medio de la modificación de parcelas***.

La manera de dibujar el espacio para la simulación fue optar por modificar las parcelas por medio de sus coordenadas. Si bien este es un método un poco rústico la resolución del espacio fue meramente decorativa y no tenía otra función que frenar el libre desplazamiento de los agentes y así representar una sala de cine. No estaba puesto el foco en el detalle de una sala real si no solo de manera esquemática para poder avanzar con

funciones más interesantes y la obtención de datos. Debajo dejamos un ejemplo de cómo fue establecida la sala.

```
ask patches
[
  ;;seats
   if pxcor >= ( - halfedge) and pycor = ( 12 ) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) an
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = ( 10 ) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) an
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = ( 8 ) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = ( 6 ) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = ( 4 ) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = ( 2 ) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = ( 0 ) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = (- 2) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = (- 4) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = (- 6) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = (- 8) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) and
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = (- 10) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) an
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = (- 12) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) an
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = (- 14) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) an
    [ set pcolor blue ]
   if pxcor >= ( - halfedge) and pycor = (- 16) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge) an
    [ set pcolor blue ]

  ;; scenario
  if pxcor >= ( - halfedge) and pycor = (20) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge)
    [ set pcolor orange ]
   if pxcor >= ( - halfedge) and pycor = (19) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge)
    [ set pcolor orange ]
   if pxcor >= ( - halfedge) and pycor = (18) and pxcor <= (0 + halfedge) and pxcor >= ( - halfedge ) and pxcor <= ( 0 + halfedge)
    [ set pcolor orange ]
]
```

# 3. Herramienta realizada en JAVA para la realización de múltiples experimentos y la persistencia de los datos obtenidos por medio de base de datos Postgres.

Para la ejecución de diferentes experimentos y la persistencia de los resultados en una base de datos utilizamos la librería de NetLogo desarrollada para JAVA. NetLogo corre en la JVM así es que podemos referenciar los experimentos y obtener las variables globales resultantes de cada ejecución, así cómo la cantidad de ticks totales de la ejecución y usarlos como parámetro de tiempo. Importamos la librería para Postgres para la persistencia y

debemos tener el cuidado obvio de tener la base de datos accesible en el puerto definido. Para este proyecto creamos diferentes experimentos de manera que se puedan ejecutar secuencialmente. Cada experimento corre unas 200 veces pero esto es fácilmente modificable en la cantidad de ciclos “for” que se ejecuten. Se debe advertir al correr la experiencia que el tiempo de ejecución es largo. Las optimizaciones de esto consumen demasiados recursos por lo que decidimos reducirlos y hacer que cada experimento se ejecute secuencialmente.

```
import db.PostgreSQL;
import domain.Experiment;
import org.nlogo.headless.HeadlessWorkspace;

import java.io.IOException;
import java.sql.SQLException;

public class Main {
    public static void main(String[] argv) throws InterruptedException, IOException {
        int[] flameRates = {2, 5, 8, 11};
        int[] fires = {5, 10, 15, 20};
        PostgreSQL postgres = new PostgreSQL(5432, "postgres", "admin", "postgres");
        try {
            postgres.createTables();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        HeadlessWorkspace workspace1 = HeadlessWorkspace.newInstance();
        HeadlessWorkspace workspace2 = HeadlessWorkspace.newInstance();
        HeadlessWorkspace workspace3 = HeadlessWorkspace.newInstance();

        workspace1.open("src/main/resources/models/2_doors.nlogo");
        experiment1(postgres, workspace1, fires, new int[]{5});
        experiment1(postgres, workspace1, new int[]{10}, flameRates);

        workspace2.open("src/main/resources/models/4_doors_no_collisions.nlogo");
        experiment2(postgres, workspace2, fires, new int[]{5});
        experiment2(postgres, workspace2, new int[]{10}, flameRates);

        workspace3.open("src/main/resources/models/4_doors.nlogo");
        experiment3(postgres, workspace3, fires, new int[]{5});
        experiment3(postgres, workspace3, new int[]{10}, flameRates);

        workspace1.dispose();
        workspace2.dispose();
        workspace3.dispose();
    }
```

Para la ejecución de cada experimento debemos definir el tipo de datos que obtenemos y definir las variables afectadas de cada uno. Cada uno agregá datos a las tablas creadas anteriormente.

```
private static void experiment1(PostgreSQL postgres, HeadlessWorkspace workspace, int[] fires, int[] flameRates) {
    try {
        for (int n : fires) {
            for (int k : flameRates) {
                for (int i = 0; i < 200; i++) {
                    workspace.command("set population 500");
                    workspace.command("set fire " + n);
                    workspace.command("set flame-rate " + k);
                    workspace.command("setup");
                    for (int j = 0; j < 400; j++) {
                        workspace.command("go");
                        if ((Double) workspace.report("final-ticks") > 0) break;
                    }
                    Double scape = (Double) workspace.report("scape");
                    Double death = (Double) workspace.report("death");
                    Double ticks = (Double) workspace.report("final-ticks");
                    Double firePits = (Double) workspace.report("fire");
                    Double fireStrength = (Double) workspace.report("flame-rate");
                    Double collisions = (Double) workspace.report("collisions");

                    Experiment e = new Experiment("2_doors", firePits.intValue(), fireStrength.intValue(), scape.intValue(), death.intValue(),
                            collisions.intValue(), ticks.intValue());
                    postgres.saveExperiment(e);
                }
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

# 4. Obtención de gráficas para análisis de datos por medio de aplicación en Python.

El código realizado en Python se hizo para poder utilizar las herramientas que nos provee Matplotlib para la generación de gráficas. Debemos hacerlo utilizando las herramientas estadísticas proporcionadas por esta librería y Numpy. Estos archivos se dividieron en dos, un main y uno aparte para las funciones que nos dará cada una de las gráficas. Se pueden obtener diversa cantidad de maneras de mostrar la información pero entre las que nos son propuestas elegimos las de cajas y bigotes más gráficas de barras dependiendo si lo que queremos ver es el comportamiento de resultante de cada experimento o una comparación entre los resultados de cada uno. Una vez más lo realizar es accesible y se debe configurar de manera correcta el puerto, nombre, clave y usuario de la base de datos de la que se van a obtener los resultados de las experiencias. A continuación se muestra la sencillez del código Main para Python que tiene la función principal de realizar la conexión con la base de datos.

```
import psycopg2
import psycopg2.extras
from functions import *
import os

def get_connection():
    try:
        return psycopg2.connect("dbname=postgres user=postgres password=admin")
    except:
        return False


conn = get_connection()

if conn:
    print("Connection to the PostgreSQL established successfully.")
else:
    print("Connection to the PostgreSQL encountered and error.")

curr = conn.cursor()

if not os.path.isdir('plots'):
    os.makedirs('plots')
os.chdir('plots')

boxplot(curr)
boxplotByStrength(curr)
deathsByStrength(curr)
deathsByPits(curr)
compareExperiments(curr)
compareExperimentsByStrength(curr)
timeUntilScape(curr)
```

Después de realizada la conexión se llama a cada una de las funciones que se dejan aparte en otro “script” de manera de que estén claramente diferenciadas las funciones que son necesarias para cada gráfica obtenida. A continuación se da ejemplo de esto.

```
private static void experiment1(PostgreSQL postgres, HeadlessWorkspace workspace, int[] fires, int[] flameRates) {
    try {
        for (int n : fires) {
            for (int k : flameRates) {
                for (int i = 0; i < 200; i++) {
                    workspace.command("set population 500");
                    workspace.command("set fire " + n);
                    workspace.command("set flame-rate " + k);
                    workspace.command("setup");
                    for (int j = 0; j < 400; j++) {
                        workspace.command("go");
                        if ((Double) workspace.report("final-ticks") > 0) break;
                    }
                    Double scape = (Double) workspace.report("scape");
                    Double death = (Double) workspace.report("death");
                    Double ticks = (Double) workspace.report("final-ticks");
                    Double firePits = (Double) workspace.report("fire");
                    Double fireStrength = (Double) workspace.report("flame-rate");
                    Double collisions = (Double) workspace.report("collisions");

                    Experiment e = new Experiment("2_doors", firePits.intValue(), fireStrength.intValue(), scape.intValue(), death.intValue(),
                            collisions.intValue(), ticks.intValue());
                    postgres.saveExperiment(e);
                }
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

# 5. Repositorio

El repositorio es de Github y está accesible, se recomienda siempre configurar adecuadamente la base de datos antes de ejecutar las experiencias.

https://github.com/SebastianPintos/ABM-FireEscape

# 6. Bibliografía.


- Beauchemin, Catherine & Liao, Laura. (2012). Tutorial on agent-based models in NetLogo.
- Almeida, João & Kokkinogenis, Zafeiris & Rossetti, Rosaldo. (2012). NetLogo Implementation of an Evacuation Scenario. CoRR.
- Kasereka, Selain & Kasoro, Nathanael & Kyamakya, Kyandoghere & Goufo, Emile-Franc & Chokki, Paterne & Yengo, Maurice. (2018). Agent-Based Modelling and Simulation for evacuation of people from a building in case of fire. Procedia Computer Science. 130. 10-17. 10.1016/j.procs.2018.04.006.
- An Introduction to Agent-Based Modeling Modeling Natural, Social, and Engineered Complex Systems with NetLogo - Uri Wilensky and William Rand - The MIT Press Cambridge, Massachusetts London, England