TALLER	**AGENTES INTELIGENTES**

Profesor: Jaime Alberto Guzmán Luna

Contenido del taller:

1. BDI
2. FIPA – ACL

Construcción de un agente Inteligente BDI, de un robot de cervezas, este se encargará de brindarle cervezas a un agente (Persona), obtener dinero en caso de que se quede sin dinero yendo al banco, y abasteciéndose de cervezas cuando este se quede sin suficientes disponibles en la nevera.

Para el desarrollo de esta práctica se adjunta un archivo "DomesticRobot.nlogo", suba este archivo a NetLogo, este archivo contiene la interfaz de la simulación ya construida y las librerías propuestas por Ilias Sakellariou, para el desarrollo de agentes inteligentes BDI y el intercambio de mensajes utilizando el lenguaje FIPA, ambas librerías simulan el funcionamiento, y tienen algunas limitaciones, pero sirven para ilustrar y realizar agentes complejos, en el lenguaje que dispone el software NetLogo. Para entender su funcionamiento a fondo se recomienda leer los manuales allí descritos.

Primero vamos a definir nuestros agentes, agruparlos y declarar sus atributos correspondientes, usando el procedimiento breed [] asignaremos a un grupo de agentes o un tipo, a los agentes como individuo.
Luego usando el comando turtles-own, añadiremos ciertos atributos a nuestros agentes, en este caso queremos asignar al agente robot que transportara la cerveza los atributos de cerveza (para conocer la cantidad de cervezas que contiene el agente), dinero (la cantidad de dinero que dispone el robot para la compra de cervezas), cervezas-totales-entregadas (para conocer el total de las cervezas transportadas por el robot). Para el agente propietario (la persona o agente que pedirá cerveza y las consumira) tenemos el atributo de cervezas (para conocer la cantidad de cervezas que contiene el agente). Los atributos intentions, beliefs, incoming-queue que se añaden en ambos agentes son necesarios para el uso de las librerias BDI y FIPA.

```
;;; Un modelo con comunicación explicita entre agentes

breed [propietarios propietario]
breed [robots robot]
breed [supermercados supermercado]
breed [bancos banco]

robots-own [cerveza dinero intentions beliefs incoming-queue cervezas-totales-
entregadas]
propietarios-own [cerveza intentions beliefs incoming-queue]
```

La función setup-bancos será usada para crear nuestro agente Banco encargado de brindarle dinero al robot en caso de quedarse sin fondos suficientes para comprar cervezas.

```
to setup-bancos
  create-bancos 1
    [set shape "triangle"
      set color red
     set size 3
     setxy -10 10
     set label "Banco"
    ]
end
```

La función setup-robot será usada para crear nuestro agente Robot encargado de moverse en la simulación y brindar las cervezas.

```
to setup-robot
  create-robots 1 [
    set shape "cylinder"
    set size 1
    setxy  5 0
    set color green
    ask patch-here [set pcolor grey]
    set cerveza 0
    set cervezas-totales-entregadas 0
    set dinero robot-dinero
    set intentions []
    set incoming-queue []
    add-intention "esperando-instrucciones" "false"
  ]
end
```

La función setup- supermercado será usada para crear nuestro agente supermercado que es el encargado de proporcionar al robot cervezas a cierto costo.

```
to setup-supermercado
  create-supermercados 1 [
    set shape "house"
    set size 3
    setxy  15 15
    set color cyan
    set label "Supermercado"
  ]
end
```

La función setup-propietario será usada para crear nuestro agente propietario, encargado de “tomarse las cervezas” y pedirle más cervezas a nuestro robot.

```
to setup-propietario
  create-propietarios 1 [
    set shape "person"
    set color red
    set size 2
    setxy  0 0
    set cerveza 0
    set intentions []
    set incoming-queue []
    add-intention "quiero-unas-cervezas" "false"
  ]
end
```

La función run-simulation será usada para correr la simulación y llamar a las funciones que evalúan las intentions en cada tick

```
to run-simulation
  ask robots [execute-intentions]
  ask propietarios [execute-intentions]
  tick
end
```

Crearemos una nueva función para dejar el programa en un estado inicial, en esta llamaremos a las funciones de **ca** propia de NetLogo para limpiar la consola. También llamaremos las funciones **setup-robot**, **setup-propietario**, **setup-supermercado**, **setup-bancos** con las cuales asignamos un valor inicial a nuestros agentes y a nuestra simulación, y la función **reset-ticks** propia de NetLogo con la cual reinicia a 0 ticks que pasan en la simulación.

```
to setup
  ;;clear all
  ca

  setup-robot
  setup-propietario
  setup-supermercado
  setup-bancos
  reset-ticks
end
```

A continuación, definiremos las acciones de nuestro agente propietario, usando las librerías BDI y FIPA, crearemos mensajes para comunicarnos entre los agentes, en este caso el agente propietario y al agente robot, con esta función procesaremos el mensaje tipo inform del agente junto al mensaje enviado y crearemos una request al robot dependiendo de los resultados obtenidos.

```
to procesar-mensaje
  let msg get-message
  if get-performative msg = "inform"
```

```
    [
     if get-content msg = "no-hay-cerveza" [add-intention "pedir-cerveza" "true"]
     if get-content msg = "tomar-cerveza"
        [
          add-intention "esperando-mi-bebida" "true"
          send add-content "trae-algunas" create-reply "request" msg
          ]
    ]
end
```

Definiremos también las intention, iniciales de nuestro agente propietario, las cuales son comenzar con la necesidad de una cerveza, comunicárselo al robot y esperar a que el robot llegue con una cerveza.

```
;;; Intenciones iniciales de un propietario .
to quiero-unas-cervezas
  send add-receiver mi-robot add-content "¿cerveza?" create-message "query"
  add-intention "procesar-mensaje" "true"
  add-intention "esperar-cerveza" "llego-mensaje"
end
```

Luego definiremos el estado y las intentions que nuestro agente propietario que tendrá constantemente, que es comunicar su necesidad de una cerveza y consumirlas

```
;;; Esperando por las bebidades y tomando cervezas
to esperando-mi-bebida
  add-intention "beber-cerveza" "no-hay-cerveza"
  add-intention "esperar-cerveza" "tomar-cerveza"
end
```

Luego definiremos las funciones para comunicar los diferentes estados y acciones que tomara nuestra simulación como pedir cerveza que usando las librerías crea las intentions y las comunica al robot, otras son esperar y beber.

```
to pedir-cerveza
  add-intention "esperando-mi-bebida" "true"
  send add-receiver mi-robot add-content "¿cerveza?" create-message "request"
end
```

```
;;; Revisar si hay mensajes en el stack
to-report llego-mensaje
   report get-message-no-remove != "no_message"
end
```

```
to esperar-cerveza
  ;;; hacer nada
end
```

```
to beber-cerveza
  if cerveza > 0 [set cerveza cerveza - 1]
end
```

Otra necesidad importante es la forma en que nuestro agente propietario detecta la disponibilidad de cervezas y la respectiva comunicación con el robot.

```
;;; Sensores del propietario
to-report tomar-cerveza
  report cerveza > 0
end
```

```
to-report no-hay-cerveza
  report cerveza = 0
end
```

```
to-report mi-robot
  report [who] of one-of robots
end
```

Luego de modelar todo acerca de nuestro agente propietario, debemos crear una función que nos capture, procese y responda los mensajes para nuestro robot. Para esto procesamos el tipo de mensaje que fue enviado al robot, y la intention de esta acción, para luego decidir las nuevas órdenes de nuestro robot.

```
;;; Intenciones principales y respuesta a mensajes
to esperando-instrucciones
  let msg get-message
  if msg = "no_message" [stop]

  if get-performative msg = "request" and get-content msg = "¿cerveza?" [add-intention
"obtener-cerveza" "true"]
  if get-performative msg = "request" and get-content msg = "trae-algunas" [add-intention
"entregar-cerveza" "true"]
  if get-performative msg = "query" and get-content msg = "¿cerveza?"
    [ ifelse tomar-cerveza
      [send add-content "tomar-cerveza" create-reply "inform" msg]
      [send add-content "no-hay-cerveza" create-reply "inform" msg]
    ]
end
```

Agregamos la función para generar el plan de moverse del punto inicial al agente propietario y darle una cerveza

```
;;; Entrega la cerveza y el robot vuelve a la posicion original. Solo se aplica cuando el
robot tiene cerveza
to entregar-cerveza
   add-intention "moverse" "en-posicion"
   add-intention "poner-cerveza" "true"
```

```
   add-intention "mover-hacia-propietario" "en-propietario"
end
```

Agregamos el plan para ir al supermercado por cervezas, volver con el agente propietario o en caso de quedarse sin dinero ir al banco por dinero.

```
;;; EL plan para obtener cerveza del supermercado y llevarla al propietario.
to obtener-cerveza
   add-intention "entregar-cerveza" "true"
   add-intention "comprar-cerveza" "tomar-cerveza"
   add-intention "mover-hacia-supermercado" "en-supermercado"
    if (dinero < 10) [
     add-intention "obtener-dinero" "true"
     add-intention "mover-hacia-banco" "en-banco"
   ]
end
```

Definiremos las acciones para que nuestro robot se mueva en el espacio o agentes, y las acciones que el robot realiza para comprar cervezas, obtener dinero o entregar las cervezas.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Acciones del robot
to mover-hacia-supermercado
  face one-of supermercados
  fd 1
end
```

```
to mover-hacia-propietario
  face one-of propietarios
  fd 1
end
```

```
to mover-hacia-banco
  face one-of bancos
  fd 1
end
```

```
to moverse
  face one-of patches with [pcolor = grey]
  fd 1
end
```

```
to comprar-cerveza
  if dinero >= 10
    [set cerveza cerveza + 10
     set dinero  dinero - 10
      ]
end
```

```
to poner-cerveza
  ask one-of propietarios in-radius 1.5 [set cerveza cerveza + 2]
  set cerveza cerveza - 2
  set cervezas-totales-entregadas cervezas-totales-entregadas + 2
end
```

```
to obtener-dinero
  set dinero robot-dinero
end
```

Definiremos las funciones para identificar en el espacio si nos encontramos cerca del Banco, el supermercado, con el propietario o en el lugar de inicio.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Avisos del robot
to-report en-propietario
   report any? propietarios in-radius 1
end
```

```
to-report en-banco
   report any? bancos in-radius 1
end
```

```
to-report en-supermercado
   report any? supermercados in-radius 1
end
```

```
to-report en-posicion
  report pcolor = grey
end
```

Material de apoyo:
-Paper que ilustra la creación y las intenciones para contribuir con una librería para NetLogo que simule la arquitectura BDI y FIPA:
[https://users.uom.gr/~iliass/projects/NetLogo/Papers/Extending_NetLogo_SETN08_SVerlag_Camera_ready.pdf]
-Manual librería BDI:
[https://users.uom.gr/~iliass/projects/NetLogo/AgentsWithBeliefsAndIntentionsInNetLogo.pdf]
-Manual librería FIPA:
[https://users.uom.gr/~iliass/projects/NetLogo/FIPA_ACL_MessagesInNetLogo.pdf]
-Netlogo Dictionary 6.1.1, para revisar el funcionamiento de algunas funciones propias de NetLogo:
[https://ccl.northwestern.edu/netlogo/docs/dictionary.html#turtles-own]