Introducción al modelado basado en agentes

Una aproximación desde Netlogo

COLECCIÓN INVESTIGACIONES

INTRODUCCIÓN AL MODELADO BASADO EN AGENTES

UNA APROXIMACIÓN DESDE NETLOGO

Antonio Aguilera Ontiveros
Marta Posada Calvo

EL COLEGIO
DE SAN LUIS

006.3
A283i
Aguilera Ontiveros, Antonio
Introducción al modelado basado en agentes. Una aproximación desde Netlogo / Antonio Aguilera Ontiveros, Marta Posada Calvo. – 1ª edición. – San Luis Potosí, San Luis Potosí : El Colegio de San Luis, A. C., 2018

176 páginas ; 23 cm. – (Colección Investigaciones)
Incluye bibliografía (páginas 171-175)
ISBN: 978-607-8500-41-3

1.- Ciencias sociales – Investigación – Metodología 2. Sociología – Investigación – Metodología 3.- Inteligencia artificial – Aspectos sociales I.- Posada Calvo, Marta II.- t. II.- s.

Primera edición: 2017

Diseño de la portada: Natalia Rojas Nieto

ISBN COLSAN: 978-607-8500-41-3

Impreso y hecho en México

ÍNDICE

CAPÍTULO I
INTRODUCCIÓN

1.1 Haciendo un poco de historia

El modelado basado en agentes es una metodología relativamente novedosa en el campo de las ciencias sociales, en concreto, en la sociología computacional (Squazzoni, 2012; Cioffi-Revilla, 2014), la economía computacional basada en agentes (Hamill y Gilbert, 2016) y la teoría computacional de las organizaciones (Prietula *et al*., 1998; Carley y Prietula, 2014).

Hace ya veinte años de la publicación del libro *Growing artificial societies*, de Joshua M. Esptein y Robert Axtell (1996), trabajo icónico que abrió las puertas a toda una nueva generación de modeladores de fenómenos sociales y mostró el camino que se ha de seguir en la construcción de "mundos artificiales". En relativamente pocos años, los modelos basados en agentes han comenzado a hacerse populares entre los investigadores sociales, economistas y estudiosos de las organizaciones, y se ha disparado de forma exponencial la generación de nuevos modelos de sociedades basadas en el concepto de agente.

A pesar del tiempo transcurrido, todavía sigue sin existir un acuerdo sobre qué lenguaje computacional debe usarse para la construcción de modelos basados en agentes. Existen comunidades basadas en lenguajes de modelización especializados, tales como Swarm, Repast, Netlogo, que han sido desarrollados para facilitar la definición-construcción del agente y su entorno. Sin embargo, nada impide usar lenguajes de uso general como C++, Java o Phyton. Es más, ya existe Spade (Smart Python Multi-Agent Development Environment) que es una *suite* de Phyton con bibliotecas especiales para el desarrollo de modelos multiagente. En Java está el lenguaje Netlogo, el cual será el lenguaje que usaremos a lo largo de este libro para desarrollar los mundos artificiales.

Netlogo ha sido desarrollado por el grupo de investigación Connected Learning and Computer-Based Modeling (CCL) de la Northwestern University, dirigido por el Prof. Uri Wilensky. Nos hemos decantado por Netlogo porque es un lenguaje muy sencillo, fácil de aprender y utilizar ya que contiene una interfaz gráfica que representa el "mundo de los agentes", una interfaz de "línea de comandos" en donde se pueden introducir comandos Netlogo y, además, una interfaz gráfica de "laboratorio" *in silico*, en donde es posible insertar botones de inicio, paro, contadores y salidas gráficas. Por estas razones consideramos que el *ambiente* Netlogo es ideal para todos aquellos interesados en incursionar en el mundo de los modelos basados en agentes. En los próximos capítulos explicaremos el ambiente Netlogo y el lector comprenderá mejor el porqué de nuestra decisión.

1.2 La idea general del libro

En este libro explicaremos en qué consiste modelar basándose en agentes y, por medio de tres sencillos ejemplos, introduciremos al lector en el modelado basado en agentes en las ciencias sociales. En este capítulo contaremos algo de la historia del modelado basado en agentes, explicaremos sus orígenes y expondremos sus aplicaciones más recientes, haciendo alusión a los trabajos de investigadores en el campo de la sociología computacional, la economía computacional y la teoría computacional de las organizaciones.

El libro intenta ser un manual práctico para aquellos investigadores, ya sean sociólogos, economistas o estudiosos de las organizaciones que quieran utilizar el modelado basado en agentes en sus investigaciones. Para ello, ofreceremos al lector un marco modélico basado en uno de los estándares más difundidos para describir modelos basados en agentes, esto es el protocolo ODD (Grimm *et al.,* 2006; 2010). Utilizaremos dicho protocolo estándar para describir los ejemplos que se incluyen en este libro.

Además, se pretende que el libro sea un manual sencillo y práctico de introducción a Netlogo, el lenguaje computacional que nos servirá para la construcción de los modelos computacionales. No se pretende que el libro sea un manual exhaustivo de Netlogo; para ello remitimos

al lector al libro de Uri Wilensky y William Rand (2015) y a los ejemplos de la biblioteca de modelos de Netlogo. No obstante, esperamos que sea lo adecuadamente completo como para generar una base cognitiva de Netlogo tan extensa como para que el lector pueda explorar por su cuenta el lenguaje y generar sus propios modelos.

En la segunda parte de este libro se incluyen tres aplicaciones en las ciencias sociales, a saber: el modelo de segregación de Schelling como ejemplo paradigmático del uso de modelos basados en agentes en la sociología computacional (véase el capítulo V), un modelo de intercambio en un mercado como ejemplo del uso de modelos basados en agentes en la economía computacional basada en agentes (capítulo VI) y un modelo de rotación involuntaria de personal como ejemplo del uso de modelos basados en agentes en la teoría computacional de las organizaciones (capítulo VII).

1.3 La idea del modelado basado en agentes en la sociología computacional

La construcción de modelos basados en agentes para la investigación en ciencias sociales, y en concreto, en la sociología, tiene sus raíces en los trabajos de los primeros científicos sociales que usaron matemáticas y desarrollaron las bases de lo que hoy es la sociología matemática (Squazzoni, 2012). Investigadores como James S. Coleman, Thomas J. Fararo, Herbert A. Simon, Thomas C. Schelling y Mark S. Granovetter son algunos de quienes contribuyeron con sus trabajos a lo que ahora podemos llamar la *sociología analítica* (Manzo, 2010; Squazzoni, 2012). De la sociología analítica se desprende la idea de una aproximación *generativa* a la investigación sociológica (Epstein, 1999; 2006; 2013). La pregunta básica en la sociología generativa es ¿cómo pueden las interacciones de agentes locales descentralizados, autónomos y heterogéneos generar regularidades específicas? (Epstein, 1999: 41). Es aquí donde el modelado basado en agentes tiene sentido, tanto metodológico como teórico (Epstein, 1999; 2006; 2013; Squazzoni, 2012).

La *sociología computacional basada en agentes* es el estudio de patrones sociales por medio de modelos computacionales de interacción

social entre agentes heterogéneos incrustados en estructuras sociales dadas, por ejemplo, redes sociales, vecindades espaciales, estructuras institucionales, etcétera (Squazzoni, 2012).

Se puede establecer el principio de la sociología computacional con los trabajos seminales de James S. Coleman, quien lideró uno de los centros de investigación más activos en el campo de la investigación computacional en sociología en Estados Unidos. Dicho centro estaba establecido en la Johns Hopkins University. Coleman publicó algunas de las más interesantes contribuciones en el campo de la simulación y la construcción de modelos en sociología tendientes a investigar la interacción entre agentes (véase, por ejemplo, Coleman 1962; 1964).

De James S. Coleman podemos pasar a las contribuciones de Thomas J. Fararo, quien en su texto "The Nature of Mathematical Sociology: A Non-Technical Essay" (1969), defiende y expone las bondades y retos de la modelización de fenómenos sociales mediante modelos matemáticos. La idea detrás de la formalización de modelos matemáticos de fenómenos sociales es que éstos pueden hacer a la sociología más científica (Coleman, 1964; Fararo, 1969).

Otro investigador de gran renombre que contribuyó al desarrollo de la actual sociología computacional es el premio Nobel en Economía Herbert A. Simon. Una de las principales ideas de Simon es que no existe un isomorfismo entre la complejidad que los sistemas sociales muestran en el nivel macro y la complejidad que éstos muestran en el nivel micro. En muchos casos, la complejidad en el nivel macro no es más que el resultado de la interacción entre procesos simples a nivel micro. Además, la simulación computacional es básica para simplificar y modelar sistemas sociales complejos desde un punto de vista micro/macro (Simon, 1969).

Thomas C. Schelling, otro laureado Nobel en Economía, contribuyó al desarrollo de la sociología computacional gracias a sus observaciones y teorías simples sobre el comportamiento humano. Si bien Schelling tiene varios modelos sobre el comportamiento tanto individual como colectivo (Schelling, 1978); su legado más replicado y estudiado es el de la segregación espacial (Schelling, 1969; 1971). No ahondaremos en este modelo ya que en el capítulo V de este libro lo expondremos y exploraremos como ejemplo paradigmático del uso de modelos basados en agentes en la sociología computacional.

1.4 La economía computacional basada en agentes

Por economía computacional entendemos una rama de la Economía que usa de forma extensiva modelos basados en algoritmos computacionales para resolver problemas de la índole económica. Parte de un conjunto de modelos que presentan y explican un problema económico, a continuación se representa el problema mediante un modelo matemático, el cual posteriormente se transforma en un modelo computacional en el marco de un software específico (Kendrick *et al.*, 2006). Dicho software puede ser GAMS, Mathematica, Matlab, el paquete Solver de Excel (Kendrick *et al.*, 2006), o bien, Netlogo si la aproximación es con agentes (Hamill y Gilbert, 2016).

En el campo de la economía computacional existe una asociación especializada, la Society for Computational Economics, fundada en 1995 y tiene su revista oficial editada por Springer. Dicha revista es *Computational Economics*, en la cual se presentan investigaciones multidisciplinarias que utilizan la computación para comprender y resolver problemas complejos de todas las ramas de la economía. Los temas de la "Economía Computacional" incluyen métodos computacionales en la econometría como el filtrado, los enfoques bayesianos y no paramétricos, los procesos de Markov y la simulación de Monte Carlo; métodos basados en agentes, aprendizaje automático, algoritmos evolutivos, modelado de redes neurales; aspectos computacionales de sistemas dinámicos, optimización, control óptimo, juegos, modelado de equilibrio; desarrollos de *hardware* y *software*, lenguajes de modelado, interfaces, procesamiento simbólico, procesamiento distribuido y paralelo.

Al igual que la sociología computacional basada en agentes, la economía computacional basada en agentes tiene sus orígenes en la formalización matemática de modelos de comportamiento económico. La economía ha sido la ciencia social que más extensamente ha usado las matemáticas como un lenguaje científico para darle rigor a sus teorías y poder predecir comportamientos económicos. No es nuestra finalidad hacer un recorrido por la historia de la economía matemática, remitimos para ello a los trabajos de William Petty de 1690 o de Quesnay de 1767, o bien los de Ricardo de 1821 (citados en Hamill y Gilbert, 2016). Hoy en día, la matemática es ubicua en la economía y no se puede concebir un

economista que no sea capaz de entender el lenguaje matemático. Jacob Marschak (citado en Escobar, 2001: 1) decía: "El hecho de que una teoría internamente coherente y determinada se formule o no en términos matemáticos no cambia su esencia lógica; pero es más fácil verificar su coherencia y su determinación si se enuncia en términos matemáticos".

El diseño de mercados es una de las áreas de investigación de la economía computacional basada. Dotando a los agentes artificiales de diferentes capacidades de racionalidad y de aprendizaje, es posible analizar si las causas de las regularidades observadas en la economía experimental se deben al protocolo de intercambio del mercado o a las capacidades cognitivas de sus participantes. En el capítulo VI del libro exponemos un modelo de intercambio en un mercado como ejemplo del uso de modelos basados en agentes en la economía computacional basada en agentes.

1.5 La teoría computacional de las organizaciones

La teoría computacional de las organizaciones (o COT, por sus siglas en inglés) se define como el estudio de las organizaciones en tanto entidades computacionales (Prietula *et al.*, 1998). La COT considera que la organización es un constructo cultural, más precisamente, una construcción social de utilidad para la resolución de problemas que es instituida mediante la acción cotidiana de los actores que participan en ella, sin importar jerarquías, cargos o funciones, reconociendo que las reglas, normas, prácticas y ritos de la organización instituyen comportamientos bien definidos.

Esta simultaneidad de lo instituido y lo instituyente (Etkin y Schvarstein, 1989) ha fraguado un discurso que establece la mutua dependencia entre el individuo y la organización, de manera que el individuo reproduce, mediante sus actos, el conjunto de relaciones sociales existentes a priori haciendo evidente la existencia previa de la estructura organizacional, de sus habilidades y constreñimientos, así como de los recursos organizacionales.

La COT considera a las organizaciones como sistemas computacionales, toda vez que las organizaciones pueden ser entendidas como sistemas

complejos de procesamiento de información, realización de tareas y toma de decisiones. En este sentido, una organización está conformada por múltiples agentes distribuidos que tienen propiedades organizacionales, tales como la necesidad de actuar colectivamente, desempeñar tareas individuales y luchar por el poder. Estos agentes tienen tareas asignadas, usan tecnologías y recursos del sistema, y gracias a su conocimiento, habilidades y capacidades de comunicación resuelven problemas individuales y colectivos (Prietula *et al*., 1998; Lomi y Larsen, 2001; Ashworth y Carley, 2006; Harrison *et al*., 2007; Carley y Prietula, 2014).

La formalización computacional, en similitud con la formalización matemática, es indispensable para la COT. Harrison *et al.* (2007: 1232) definen un modelo formal como "una formulación precisa de las relaciones entre variables, incluyendo la formulación de los procesos a través de los cuales los valores de las variables cambian con el tiempo, basados en razonamiento teórico. El formalismo puede especificar relaciones matemáticas en forma de ecuación o conjuntos explícitos de reglas de la forma *cuando ocurre X, entonces ejecuta Y*".

Los orígenes de la COT pueden ser rastreados en el famoso y multicitado, pero defectuoso, trabajo del modelo del bote de basura (Garbage Can Model) planteado por Cohen *et al.* (1972). Masuch y LaPotin (1989) realizan una revisión crítica del modelo del bote de basura, en la cual se muestran sus inconsistencias. Dicho modelo ha sido de utilidad en cuanto a la metáfora de lo que es una anarquía organizada, pero este concepto no tiene nada que ver con el modelo computacional que los autores plantean en su artículo (Masuch y LaPotin, 1989). La contribución mayor de Masuch y LaPotin no es la crítica al modelo de Cohen, March y Olsen, sino la construcción de un nuevo modelo computacional que utiliza actores con inteligencia artificial; en específico, los actores están capacitados con el procedimiento de búsqueda desarrollado por Simon y Kadane (1975) que incluye el concepto de racionalidad limitada propuesta por Simon (1955).

Sin duda, la teoría computacional de las organizaciones sustentada en sistemas basados en agentes ha avanzado mucho desde el trabajo de Masuch y LaPotin (1989), ejemplo de ello son los trabajos de Carley y Gasser (1999); Davis *et al.* (2007); Harrison *et al.* (2007); Prietula (2011). Sin embargo, hay que reconocer que todavía el modelado basado

en agentes no ha tenido el suficiente impacto en la construcción, tanto teórica como empírica, de la teoría de las organizaciones. Miller (2015) plantea esto a debate y propone usar la perspectiva del realismo crítico para clarificar las ontologías, evaluar los resultados del modelo, validando los modelos mediante la triangulación e identificando los límites del modelado basado en agentes. Recomendamos ampliamente al lector el trabajo de Miller (2015).

En el capítulo VII del libro exponemos un modelo de rotación involuntaria de personal que trabaja en equipo, como ejemplo del uso de los sistemas basados en agentes en la COT.

PARTE I.
MODELADO BASADO EN AGENTES EN LAS CIENCIAS SOCIALES

CAPÍTULO II
MODELADO BASADO EN AGENTES

2.1 ¿Qué son los sistemas basados en agentes?

Un agente es una entidad computacional (esto es un código de computadora) con inteligencia artificial, capaz de tener interfaces que le permitan interaccionar con su medio ambiente y con otros agentes. Nuestro "código inteligente" es capaz de buscar y concretar objetivos, usar su memoria y capacidades cognitivas para discernir, tomar decisiones, "sobrevivir" y "prosperar" (Epstein y Axtell, 1996; Huhns y Singh, 1998), por lo menos en teoría.

La realidad es que existen muchos tipos de agentes computacionales, algunos más inteligentes que otros (o menos tontos, depende del punto de vista). Las razones de ser de los agentes son muy variadas. Algunos se programan para que generen minería de datos en internet, otros para buscar las mejores opciones en el pago de hoteles o billetes de avión, otros se generan como modelos simples del ser humano y son usados para indagar su comportamiento tanto individual como colectivo (Huhns y Singh, 1998; Lindemann *et al.*, 2004; Poteete *et al.*, 2012; Railsback y Grimm, 2012; Wilensky y Rand, 2015). En este libro nos interesa el modelado de agentes, con diferentes niveles de cognición, orientados a metas para su uso en la investigación social.

Una taxonomía de agentes requiere identificar las características clave de los agentes (véase la Tabla 2.1), de los sistemas multiagente en los que éstos participan (Tabla 2.2), del marco de trabajo (o *frameworks*) en que los agentes se desenvuelven (Tabla 2.3), de los roles que ellos juegan y del ambiente que habitan (Tabla 2.4) (Huhns y Singh, 1998: 2-3).

Tabla 2.1

Características intrínsecas y extrínsicas de un agente

Característica intrínseca	**Rango de valores**
Esperanza de vida	Transitoria a larga duración
Nivel de cognición	Reactivo a deliberativo
Construcción	Declarativo a procedural
Movilidad	Estacionario a itinerante
Adaptabilidad	Fija a enseñable a autodidacta
Modelamiento	De ambientes, de ellos mismos, de otros agentes
Característica extrínseca	**Rango de valores**
Localidad	De local a remoto
Autonomía social	Independiente a controlada
Sociabilidad	Autista, consciente, responsable, jugador en equipo
Amigabilidad	Cooperativo, competitivo, antagonista
Interacciones	Logísticas: directas o vía facilitador o mediador. Estilo/Cualidad/Naturaleza: con agentes/mundo/ambos. Nivel semántico: Comunicaciones declarativas o procedurales.

Fuente: Huhns y Singh, (1998: 2).

Tabla 2.2

Características de los sistemas multiagente

Característica	**Rango de valores**
Unicidad	Homogéneo a heterogéneo
Granularidad	Grano fino a grano grueso.
Estructura de control	Jerarquía a democracia
Autonomía de interfaz	Comunicación: vocabulario específico, lenguaje, protocolo, Intelecto: metas específicas, creencias, ontologías. Habilidades: procesos específicos, comportamientos.
Autonomía de ejecución	Independiente a controlada.

Fuente: Huhns y Singh, (1998: 2).

Tabla 2.3

Características del marco de trabajo

Característica	Rango de valores
Autonomía de diseño	Plataforma/lenguaje/arquitectura interna/protocolo de interacción.
Infraestructura de comunicación	Memoria compartida o basada en mensajes. Conectado o desconectado. Punto a punto, multidifusión, emisión Empujar o jalar Sincrónico o asincrónico
Servicio de directorio	Páginas blancas, páginas amarillas
Protocolo de mensajes	KQML HTTP y HTML OLE, CORBA, DSQM
Mediación de servicios	¿Basada en la ontología? ¿Transaccional?
Servicios de seguridad	Marcas de tiempo/autentificación
Servicios de envío	Facturación/moneda
Operaciones de soporte	Archivado/redundancia/restauración/contabilidad

Fuente: Huhns y Singh, (1998: 2).

Tabla 2.4

Características del ambiente

Característica	Definición
Conocible	¿En qué medida es el entorno conocido al agente?
Previsible	¿En qué medida puede el ambiente ser predecible por el agente?
Controlable	¿En qué medida puede el agente modificar el medio ambiente?
Histórico	¿Los estados futuros dependen de toda la historia o solo del estado actual?
Teleológico	¿Existen partes con propósito?
Tiempo real	¿Puede el medio ambiente cambiar mientras los agentes están deliberando?

Fuente: Huhns y Singh, (1998: 2).

2.2 Inteligencia artificial distribuida

Los agentes computacionales han sido un tópico activo de exploración e investigación durante las primeras seis décadas de la inteligencia artificial (IA). En sus fases tempranas, la inteligencia artificial investigó a los agentes de forma explícita, aunque usando modelos muy simples. El optimismo de construir sistemas realmente inteligentes, con las teorías y la tecnología de los años cincuenta y sesenta, resultó en una desesperanza. En los años setenta y ochenta los estudios de inteligencia artificial se concentraron en una estrecha gama de aplicaciones. Éstas fueron representación del conocimiento, técnicas de búsqueda, sistemas expertos, aprendizaje de máquinas, interpretación del lenguaje natural.

En los últimos años de la década de los setenta, una pequeña comunidad de investigadores en inteligencia artificial se conformó bajo el nombre de inteligencia artificial distribuida (IAD) (Huhns y Singh, 1998: 4). La inteligencia artificial distribuida puede definirse como el campo del conocimiento que estudia e intenta construir conjuntos de entidades autónomas e inteligentes, las cuales cooperan entre sí para desarrollar un trabajo o tarea específica y que además son capaces de comunicarse entre sí por medio de mecanismos basados en el envío y recepción de mensajes (Drogoul y Ferber, 1994; Ferber, 1994; Ferber 1999).

La inteligencia artificial distribuida tomó prestados muchos conceptos de la psicología, tales como las creencias y las intenciones (*beliefs and intentions*) y desarrolló todo un esquema teórico metodológico robusto que permite el diseño de grandes sistemas distribuidos. El préstamo no se limitó a la psicología; varias abstracciones, conceptos y constructos fueron tomados de la sociología, la teoría de las organizaciones, la economía y la filosofía del lenguaje y la lingüística (Huhns y Singh, 1998: 4).

2.3 Agentes por todos lados

El término *agente* comenzó a ser usado en los ochenta y noventa por las comunidades computacionales tradicionales, tales como las comunidades de bases de datos, sistemas operativos y redes. Un agente fue entonces entendido como un *proxy* (apoderado) para la computación

de un sitio (*site*), esto es, una transacción, un proceso o un ruteador de red, que podía sondear el estado de un sistema subyacente. Los agentes fueron entonces entendidos como una interfaz para un sistema arbitrario. Por ejemplo, el protocolo de red Simple Network Management Protocol (SNMP) tiene definidos ciertos tipos de agentes bajo los cuales la aplicación de la gestión de red puede ser diseñada. Esta aplicación puede en principio ejecutarse en un ambiente multivendedor tan grande como los agentes implementados por los diferentes vendedores, que producen la misma funcionabilidad mientras "esconden" los detalles internos del propietario de los componentes, tales como los ruteadores de red (Huhns y Singh, 1998: 5).

Con la expansión de internet y de la *web* en los noventa, fuimos testigos de la emergencia de agentes de *software* diseñados para los ambientes de información abierta. Dichos agentes desarrollan tareas al servicio de un usuario o sirven de nodos-*broker* o fuentes de información dentro del sistema global de información.

Hoy en día los agentes de *software* han abandonado los sistemas operativos tradicionales y se han mudado a los ambientes creados para teléfonos celulares inteligentes y tabletas. Son parte de lo que hoy llamamos *big data technologies*. Extraen información de dónde nos localizamos, dónde y qué compramos, qué gustos tenemos, y ayudan a las empresas a saber más sobre sus posibles clientes.

En el área científica de las ciencias sociales, la inteligencia artificial distribuida se ha extendido en el desarrollo de modelos de acción colectiva en donde hay dilemas sociales. Un dilema social es una situación en la cual, se actúe como se actúe, siempre tendremos una solución no satisfactoria para alguna de las partes involucradas. Es en esta área del conocimiento sociológico donde los modelos basados en agentes han contribuido a expandir el conocimiento con respecto a las situaciones en donde se comparten recursos y emerge sin esperarlo el comportamiento cooperativo (Poteete *et al*., 2012).

2.4 Modelos basados en agentes versus otras formas de modelar

Dado que la ecuación es la forma más común de modelo científico, al lector puede surgirle la siguiente pregunta: ¿Qué es lo que hace a los modelos basados en agentes distintos del modelado basado en ecuaciones?

Una distinción entre los modelos basados en agentes con respecto a los modelos basados en ecuaciones es que los primeros pueden modelar una población heterogénea, mientras que los modelos basados en ecuaciones típicamente asumen poblaciones homogéneas. Además, cuando se modelan individuos, las interacciones y resultados son típicamente discretos y no continuos. Los modelos continuos no siempre mapean bien las situaciones del mundo real. Por ejemplo, los modelos basados en ecuaciones de dinámicas poblacionales tratan dichas poblaciones como si fueran cantidades continuas, cuando en realidad dichas poblaciones están compuestas de individuos discretos. Cuando simulamos dinámicas poblacionales es muy importante conocer si tenemos una población sostenible. Por ejemplo, en los modelos basados en ecuaciones como el de Lotcka-Volterra (modelo depredador-presa), una población de depredadores (por ejemplo, lobos) se extingue si dicha población es menor que dos. Sin embargo, en los modelos basados en ecuaciones, un millonésimo de lobo tiene sentido y contribuye a la población. Obviamente, el modelo basado en ecuaciones deja al instante de ser realista.

Otra ventaja del modelado basado en agentes sobre el modelado basado en ecuaciones es que aquel no requiere conocimiento del fenómeno agregado. No es necesario conocer qué patrones globales resultan del comportamiento individual. Cuando se modela una variable de salida en el modelado basado en ecuaciones, se necesita tener una buena comprensión del comportamiento agregado y entonces probar las hipótesis contra las salidas agregadas del modelo. Por ejemplo, en el modelo depredador-presa, para construir el modelo basado en ecuaciones, se necesita tener una comprensión de la relación entre la población de lobos y la población de ovejas. Para codificar este conocimiento agregado tal como en el clásico modelo de ecuaciones de Lotka-Volterra, se debe tener conocimiento de ecuaciones diferenciales. En contraste, un modelo basado en agentes permite escribir reglas simples para entidades simples, requiriéndose sólo

conocimiento de sentido común con respecto a los comportamientos de los individuos lobos y ovejas y pese a la sencillez, se puede todavía observar el comportamiento agregado al momento de correr el modelo. Entonces, aun cuando no se tengan hipótesis de cómo interactuarán las variables agregadas, es posible construir un modelo y generar resultados.

Ya que los modelos basados en agentes describen individuos y no agregados, la relación entre un modelo basado en agentes y el mundo real es más fácilmente igualada. Por tanto, es mucho más sencillo explicar qué es lo que el modelo está haciendo a personas que no tienen entrenamiento previo en el paradigma de modelado basado en agentes. Esto es conveniente porque significa que no se requiere de un entrenamiento especial para entender un modelo basado en agentes. Todos los interesados en el proceso de modelado pueden entender el modelo. Más aún si se usa un lenguaje especializado para el modelado basado en agentes, tal como Netlogo la sintaxis será más entendible para los interesados sin ningún conocimiento de cómo construir el modelo. Esto ayuda tanto a la verificabilidad del modelo como a su replicabilidad.

Finalmente, los resultados generados por un modelo basado en agentes son más detallados que aquellos generados por un modelado basado en ecuaciones. El modelado basado en agentes puede proveer de resultados tanto a un nivel de detalle individual como a un nivel de detalle agregado al mismo tiempo. Dado que un modelo basado en agentes opera modelando a cada individuo y sus decisiones, es posible examinar la historia y vida de cada individuo en el modelo. O bien es posible examinar a los individuos agregados como un colectivo. La aproximación de "abajo hacia arriba" (*bottom-up*) de un modelo basado en agentes es frecuentemente contrastada con la aproximación "de arriba hacia abajo" (*top-down*) de muchos modelos basados en ecuaciones, los cuales nos hablan sólo de cómo se comporta el sistema como un agregado y no nos dice nada con respecto a los individuos.

2.5 Aleatoriedad versus determinismo

Un rasgo característico del modelado basado en agentes, y de los modelos computacionales en general, es que es muy fácil incorporar la aleatoriedad

dentro del modelo. Muchos modelos basados en ecuaciones y otras formas de modelar requieren que cada decisión en el modelo se realice de forma determinística. En los modelos basados en agentes, en lugar de ser deterministas, las decisiones pueden basarse en una probabilidad.

Por ejemplo, si tenemos un modelo basado en agentes en donde una de las variables de estado, la posición, se actualiza con el movimiento del agente, dicho movimiento puede ser determinístico o bien obedecer a una función aleatoria. La variable de estado es entonces una variable de tipo aleatoria. La aleatoriedad también se puede introducir al inicializar un modelo, ya que cada agente es único y está colocado en una posición al azar al inicio del programa.

Otro ejemplo de variable de estado aleatoria es el color del agente. El color puede ser aleatorio obedeciendo a una función de densidad, en donde se escoge que cierto porcentaje de la población de agentes sea de un color, y el resto de otro color.

Un ejemplo más de variable de estado aleatoria es el mismo agente. Los mundos artificiales no siempre se rellenan completamente de agentes. Puede haber un mundo en donde se creen al azar sólo unos cuantos agentes. Digamos de 2 a 15. La variable de estado "identificador de agente" o "ID" es la variable aleatoria y puede tomar como mínimo 2 valores y 15 como máximo.

Por último, con frecuencia existen ocasiones donde no se conoce de manera suficiente cómo funciona un sistema complejo, de tal forma que seamos capaces de construir completamente un modelo determinista. En muchos de estos casos, el único tipo de modelo que podemos construir es un modelo con algunos elementos de aleatoriedad. Los modelos basados en agentes, así como otras formas de modelado que permitan incorporar características aleatorias, son esenciales para estudiar este tipo de sistemas.

2.6 ¿Cuándo es un modelo basado en agentes más adecuado?

Los modelos basados en agentes pueden ser usados para modelar cualquier fenómeno natural (por ejemplo, construir un modelo de agentes

a escala atómica, tal como un modelo de *spines* al estilo de Ising). Sin embargo, hay algunos contextos para los cuales el costo de construir un modelo basado en agentes excede los beneficios, aunque también existen ocasiones en las cuales los beneficios exceden con creces los costos. A pesar de que a veces es difícil discernir la diferencia entre los dos escenarios, existen pocas guías generales que ayuden a identificar situaciones donde el modelado basado en agentes es particularmente valioso. Además, éstas se ofrecen como pautas y no como prescripciones o reglas acerca de cómo usar el modelado basado en agentes. Lo más frecuente es juzgar la ideoneidad basándose en situaciones particulares (Wilensky y Rand, 2015: 35).

En general, los modelos basados en agentes son útiles cuando los agentes no son homogéneos. Por ejemplo, modelar las transacciones y eventos en una bolsa de valores requiere un alto grado de detalle al examinar los comportamientos a nivel individual ya que diferentes agentes tienen diferentes umbrales de riesgo y, por tanto, no tomarán la misma decisión dadas las mismas condiciones del mercado. Por tanto, cuando los agentes son heterogéneos y la heterogeneidad de los agentes afecta el comportamiento del sistema, los modelos basados en agentes son útiles porque permiten que se haga un seguimiento individual y puntual a cada agente. Esta aproximación es mucho más poderosa que técnicas tales como el modelado de sistemas dinámicos propuesto por Forrester (1968) (Sterman, 2000). El modelado de sistemas dinámicos requiere la creación de valores separados para cada grupo de agentes con diferentes propiedades, y entonces el espacio de las propiedades es muy grande, lo que dificulta la construcción, rastreo e integridad del modelo. Por otro lado, un modelo basado en agentes requiere sólo la especificación de las propiedades de los agentes, lo cual provee de una descripción más concisa de un sistema complejo.

Asimismo, los modelos basados en agentes son útiles cuando las interacciones entre los agentes son complejas, bien por su naturaleza o porque involucran agentes heterogéneos. Por ejemplo, se pueden especificar unas pocas reglas simples para describir una rica gama de interacciones entre agentes de distinto tipo. Además, en los modelos basados en agentes, los agentes individuales pueden preservar su historia de interacciones, de forma que ellos pueden cambiar sus comportamientos e incluso sus estrategias basados en los eventos pasados (Wilensky y

Rand, 2015: 36). Por ejemplo, en un modelo basado en agentes de la evolución de la cooperación, es posible que los agentes puedan aprender y, por tanto, modificar su comportamiento como resultado de la interacción continua con un particular grupo de agentes. Quizá aprendan a desconfiar de un grupo o bien aprendan a comportarse más favorablemente hacia el grupo (Poteete *et al*., 2012).

De la misma manera que los modelos basados en agentes son útiles cuando la interacción entre los agentes es compleja, también son útiles cuando la interacción entre los agentes y su ambiente es compleja. Modelar interacciones agente-medio es tan general como modelar cualquier interacción agente-agente. El ambiente, en un modelo basado en agentes está frecuentemente formado por agentes estacionarios.

Por ejemplo, en un modelo basado en agentes de un entorno ecológico marino de pesca, un pescador puede reconocer una localización particular como un lugar en donde él ha pescado anteriormente y decidir no volver a pescar en dicho lugar de nuevo. Esta interacción agente-medio permite que información geográfica y dependiente del lugar sea incluida en el modelo, por lo que se obtiene una riqueza que no tendría un modelo geográficamente independiente. En los modelos de pesca, puede ser conocido que la población promedio de peces es constante sobre el tiempo para una gran área, pero esto podría significar que la población promedio de peces sea constante en todas las subáreas o que el balance entre diferentes subáreas, unas con otras, resulte en una gran población de peces en unos lugares mientras que otros tengan una población muy pequeña. Entonces, la descripción rica en especificidades del ambiente considerando la variable geográfica implica que el modelado basado en agentes genere información más detallada (Wilensky y Rand, 2015: 36).

CAPÍTULO III

DESCRIPCIÓN DEL MODELO
PROTOCOLO ODD

3.1 Describiendo y formulando modelos basados en agentes

Describir el modelo es una de las partes más difíciles del arte del modelado basado en agentes. Desde los inicios de la simulación basada en agentes y de los sistemas multiagente ha sido patente la necesidad de contar con un estándar aceptado universalmente como protocolo para la creación de mundos artificiales, es decir, su formulación tanto escrita, como formal matemática y su posterior programación. En un principio, al no existir lenguajes especializados en la programación de sistemas basados en agentes, los científicos entusiastas de este enfoque recurrían a lenguajes estándar como C++, lo que permitía que se pudieran usar la noción de objetos y se podía recurrir al protocolo de diseño UML para especificar lo que hacía el programa.

La formulación de un modelo basado en agentes significa procesar el modelo desde su parte heurística en la cual se ha pensado acerca del problema que se está modelando (los datos, las ideas y las hipótesis) hasta la primera representación formal y rigurosa del modelo (Railsback y Grimm, 2012: 35).

Sin embargo, la descripción del modelo queda en un lugar incierto entre la retórica y el formalismo matemático necesario en algunos modelos, pero infértil en otros. El modelo debe ser explicado lo mejor posible con fines de replicabilidad, pero ¿qué es lo verdaderamente relevante del modelo que hay que explicar? ¿Cómo garantizar la completa explicación-descripción que permitiera la replicabilidad del modelo? ¿Qué características de un modelo basado en agentes necesitan describirse en su formulación y cómo podemos describir dichas características de forma concisa y al mismo tiempo clara y completa?

Cuando se usan modelos basados en ecuaciones, sabemos exactamente cómo responder las preguntas anteriores: anotamos las ecuaciones y los valores de sus parámetros. Pero, cuando usamos un modelo basado en agentes, estas cuestiones son difíciles porque los modelos pueden ser complejos y porque no contamos con una notación tradicional, tal y como la hay cuando modelamos con ecuaciones diferenciales.

Para enfrentar este reto, Grimm y un conjunto de colegas suyos y de otras áreas que trabajaban con sistemas basados en agentes desarrollaron un protocolo para describir sistemas basados en agentes, el cual combinaba dos elementos: 1) una estructura general para la descripción de sistemas basados en agentes, de tal forma que la descripción del modelo fuera independiente de su estructura específica; y 2) el uso del lenguaje de las matemáticas para separar claramente las consideraciones verbales de las ecuaciones, las reglas y los programas que constituyen el modelo (Grimm *et al*., 2006).

3.2 El protocolo ODD: ¿qué es? y ¿por qué usarlo?

Grimm, Railsback y 28 investigadores más acordaron utilizar un protocolo estándar para describir modelos basados en agentes y realimentar las ideas y conceptos. El resultado de este esfuerzo colectivo es el trabajo publicado en la revista *Ecological Modelling*, titulado "A standard protocol for describing individual-based and agent-based models" (Grimm *et al*., 2006). En dicho artículo, sus autores proponen un protocolo estándar al que denominan ODD (Overview, Design concepts, and Details) (Descripción general, los conceptos de diseño y los detalles). Estos tres bloques principales se subdividen en siete elementos (Tabla 3.1): 1) propósito, 2) variables de estado y escalas, 3) descripción y planificación de procesos, 4) conceptos de diseño, 5) inicialización, 6) entradas del modelo y 7) submodelos.

Grimm *et al*. (2010) y Railsback y Grimm (2012) son la base para la descripción del protocolo ODD. Esto garantiza la actualidad de los conceptos e ideas y evita las ambigüedades que existían en el protocolo propuesto en Grimm *et al.* (2006).

Tabla 3.1

Descripción general del protocolo ODD

	Elementos del protocolo ODD
Descripción general (Overview)	1. Propósito
	2. Entidades, variables de estado y escalas
	3. Descripción general de los procesos y programación
Conceptos de diseño (Design Concepts)	4. Conceptos de diseño • Principios básicos • Emergencia • Objetivos • Aprendizaje • Predicción • Detección • Interacción • Aleatoriedad • Colectivos • Observación
Detalles (Details)	5. Inicialización
	6. Datos de entrada
	7. Submodelos

Fuente: Adaptado de Railsback y Grimm (2012: 37).

Grimm *et al.* (2010) proporcionan evidencias procedentes de la base de datos Web of Science de que, desde su publicación en 2006, el protocolo ODD se había usado en más de cincuenta publicaciones científicas que involucraban modelos basados en agentes. Podemos afirmar que el protocolo ODD está cada vez más extendido entre la comunidad de científicos que nos dedicamos a generar modelos basados en agentes. Sus virtudes son muchas y por ello hemos elegido proponer en este libro este protocolo como una de las herramientas básicas para todos aquellos que quieran generar modelos basados en agentes.

3.3 Protocolo ODD: descripción general

La descripción general se divide a su vez en tres conceptos: 1) propósito; 2) entidades, variables de estado y escalas; 3) descripción general de los

procesos y programación. A continuación explicaremos cada uno de estos elementos.

3.3.1 Propósito

Pregunta: ¿Cuál es el propósito del modelo?

Explicación: Todo modelo tiene que partir de una pregunta clara, problema o hipótesis. Por tanto, ODD comienza con un breve resumen del objetivo global por el que se desarrolló el modelo. Aquí no se describe nada acerca de cómo funciona el modelo; sólo se dice qué se va a utilizar para desarrollarlo. Se recomienda ampliamente que este párrafo se escriba con independencia de cualquier presentación en la introducción del artículo, ya que el protocolo ODD debe ser completo y comprensible por sí mismo y no sólo en relación con toda la publicación (como también es el caso de las figuras, tablas y sus leyendas).

3.3.2 Entidades, variables de estado y escalas

Preguntas: ¿Qué tipos de entidades se encuentran en el modelo? ¿Mediante qué variables de estado o atributos se caracterizan estas entidades? ¿Cuál es la resolución y extensión temporal y espacial del modelo?

Explicación: Una entidad es un objeto distinto o separado o el actor que se comporta como una unidad y puede interactuar con otras entidades o ser afectados por factores ambientales externos. Su estado actual se caracteriza por sus variables de estado o atributos. Una variable de estado o atributo es una variable que distingue a una entidad de otras entidades del mismo tipo o categoría, o traza cómo la entidad cambia con el tiempo.

Las entidades de un modelo basado en agentes están caracterizadas por un conjunto, o vector de atributos, que puede contener tanto variables numéricas como las estrategias referentes al comportamiento (Grimm

et al., 2010: 2763). Muchos modelos basados en agentes incluyen los siguientes tipos de entidades (2010: 2764): agentes/individuos, unidades espaciales, ambiente y colectivos.

Agentes / individuos. Un modelo puede tener diferentes tipos de agentes; por ejemplo, lobos y ovejas, e incluso los diferentes subtipos dentro del mismo tipo, por ejemplo, diferentes tipos de plantas o diferentes etapas de la vida de un animal. Ejemplos de tipos de agentes incluyen los siguientes: organismos, los seres humanos o instituciones. Ejemplos de variables de estado son el número de identidad (es decir, incluso si todas las demás variables de estado sean las mismas, el agente todavía mantendría una identidad única), la edad, el sexo, la ubicación espacial (que puede ser la celda que ocupa en una cuadrícula que sirve de espacio), el tamaño, el peso, las reservas de energía, las señales de condición física, tipo de uso del suelo, la opinión política, el tipo de célula, parámetros que describen la especie (por ejemplo, tasa de crecimiento y la edad máxima), la memoria (por ejemplo, una lista de amigos o la calidad de los sitios visitados los veinte pasos anteriores de tiempo), las estrategias de comportamiento, etc.

Unidades espaciales (por ejemplo, celdas de un cuadriculado). Ejemplos de variables de estado espaciales son la ubicación espacial, una lista de agentes en la celda, los descriptores de las condiciones ambientales (altitud, cobertura vegetal, tipo de suelo, etc.). En algunos modelos basados en agentes se utilizan las celdas de una cuadrícula para representar a los agentes: el estado y el comportamiento de los árboles o de las empresas se puede modelar como características de la celda. Una celda puede desempeñar múltiples papeles. Por ejemplo, una celda de cuadrícula puede ser una entidad con sus propias variables (por ejemplo, pueden definirse variables propias como el contenido de humedad del suelo, la concentración de nutrientes del suelo, etc. para una celda representando el suelo), pero también puede funcionar dicha celda como una ubicación, y, por lo tanto, un atributo de un organismo.

Ambiente. Mientras que las unidades espaciales a menudo representan condiciones ambientales que varían en el espacio, esta entidad se refiere al conjunto ambiental o a las fuerzas que impulsan el comportamiento y

la dinámica de todos los agentes o celdas de la cuadrícula. Ejemplos de variables ambientales son la temperatura, la precipitación, el precio y la demanda de mercado, regulaciones fiscales, etc.

Colectivos. Grupos de agentes pueden tener sus propios comportamientos, de manera que puede tener sentido distinguirlos como entidades; por ejemplo, grupos sociales de los animales o los hogares de los agentes humanos. Un colectivo se caracteriza generalmente por la lista de sus agentes y las acciones específicas que sólo son realizadas por el colectivo, no por sus entidades constitutivas.

En la descripción de las escalas y extensiones espaciales y temporales (la cantidad de espacio y de tiempo representado en una simulación) es importante especificar qué unidades del modelo representan la realidad. Por ejemplo: "Un paso de tiempo representa 1 año y las simulaciones tienen una duración de 100 años". Una celda de la cuadrícula representa 1 ha y el modelo comprende 1000 × 1000 ha; es decir, 10 000 kilómetros cuadrados.

3.3.3 Descripción general de los procesos y programación

Preguntas: ¿Qué entidad hace qué y en qué orden? ¿Cuándo se actualizan las variables de estado? ¿Cómo está el tiempo modelado, como pasos discretos o como un continuo sobre el que ambos, eventos discretos y procesos continuos, pueden ocurrir? A excepción de una programación muy simple, se debe utilizar pseudo-código para describir el programa en todos los detalles, de manera que el modelo se puede volver a aplicar desde dicho código. Idealmente, el pseudo-código corresponde totalmente al código real utilizado en el lenguaje de programación que se haya usado para programar el modelo.

Explicación: Deben enumerarse los nombres autoexplicativos de los procesos del modelo, por ejemplo, "actualizar-hábitat", "moverse", "crecer", "comprar", "actualizar-parcelas", etc. Estos nombres son los submodelos que se describirán más tarde en el último elemento de ODD. Los

procesos se llevan a cabo ya sea por una de las entidades del modelo (por ejemplo: "moverse") o por un controlador de nivel superior que puede "actualizar-parcelas" o escribir archivos de salida. Para manejar este tipo de procesos de nivel superior, plataformas de *software* especializadas como Swarm (Minar *et al.*, 1996) y NetLogo (Wilensky, 1999) incluyen el concepto del "modelo" u "observador", esto es, un objeto controlador que realiza este tipo de procesos.

Además del nombre de los procesos, debe especificarse tanto el orden en el que los diferentes procesos se ejecutan como el orden en que un proceso es realizado por un grupo de agentes. Por ejemplo, la alimentación puede ser un proceso ejecutado por todos los agentes tipo animal en un modelo, pero también hay que especificar el orden en el que se alimentan los animales individuales; es decir, ya sea que se alimentan en orden aleatorio, fijo o especificado por una ordenación por tamaños. Las diferencias en tal orden pueden tener un efecto muy grande en los resultados del modelo (Grimm *et al.*, 2010: 2764).

La cuestión de cuándo se actualizan las variables implica especificar si a una variable de estado se le asigna inmediatamente un nuevo valor tan pronto como se calcula el valor por un proceso (actualización asíncrona), o si el nuevo valor se almacena hasta que todos los agentes hayan ejecutado el proceso, y luego todos se actualizan a la vez (actualización sincrónica). La mayoría de los modelos basados en agentes representan el tiempo simplemente mediante el uso de pasos de tiempo, asumiendo que el tiempo corre hacia adelante en pasos discretos.

3.4 Protocolo ODD: conceptos de diseño

El elemento "conceptos de diseño" del protocolo ODD no describe el modelo *per se*; es decir, no es necesario para replicar el modelo. Sin embargo, estos conceptos de diseño tienden a ser característicos de los modelos basados en agentes, aunque ciertamente no de forma exclusiva. También pueden ser cruciales al momento de interpretar los resultados de un modelo, los cuales no se describen bien a través de técnicas tradicionales, tales como las ecuaciones y los diagramas de flujo. Por ello,

en el protocolo ODD se incluye una especie de "lista de comprobación" para asegurar que las decisiones importantes del diseño han sido hechas de manera consciente y que los lectores son conscientes de estas decisiones (Grimm *et al.*, 2010). Hay once conceptos de diseño, discutidos ampliamente por Grimm *et al.* (2010) y Railsback y Grimm, (2012). A continuación explicaremos cada uno de estos elementos.

3.4.1 Principios básicos

¿En qué conceptos generales, teorías, hipótesis, o enfoques de modelado se fundamenta en el diseño del modelo? Explicar la relación entre estos principios básicos, la complejidad del modelo y el propósito del estudio. ¿Cómo fueron tomados en cuenta todos estos conceptos? ¿Se utilizan a nivel de submodelos (por ejemplo, decisiones con respecto al uso del suelo) o es su ámbito de aplicación el nivel del sistema (por ejemplo, la teoría de la segregación residencial)? ¿El modelo proporciona información acerca de los principios básicos en sí mismos, es decir, su ámbito de aplicación, su utilidad en escenarios del mundo real, y su validación o modificación? ¿Utiliza el modelo aspectos teóricos nuevos o previamente desarrollados dentro de su marco teórico-metodológico de competencia?

3.4.2 Emergencia

¿Qué resultados clave del modelo son modelados como emergentes a partir de los rasgos adaptativos o mediante el comportamiento de los individuos? Es decir, ¿qué resultados del modelo son esperados para variar de forma compleja y quizá de forma impredecible cuando cambian las características particulares de individuos o de su ambiente? ¿Hay otros resultados que están más estrechamente impuestos por las reglas del modelo y por tanto, depende menos del comportamiento de los individuos?

3.4.3 Adaptación

¿Qué rasgos de adaptación tienen los individuos? ¿Qué reglas tienen para la toma de decisiones o el cambio de comportamiento en respuesta a los cambios en sí mismos o de su entorno? ¿Qué rasgos explícitamente tratan de aumentar cierto grado de éxito individual con respecto a los objetivos (por ejemplo, "ir a la celda que proporciona la tasa de crecimiento más rápida", donde el crecimiento se supone que es un indicador de éxito)?

3.4.4 Objetivos

Si los rasgos de adaptación actúan de forma explícita para aumentar cierto grado de éxito del individuo a satisfacer algún objetivo, ¿qué es exactamente ese objetivo y cómo se mide? Cuando los individuos toman decisiones mediante la clasificación de las alternativas, ¿qué criterios utilizan? Algunos sinónimos de "objetivos" son "utilidad", para la recompensa económica en los modelos sociales o simplemente "criterios de éxito".

3.4.5 Aprendizaje

Muchos individuos o agentes, pero también las organizaciones e instituciones, cambian sus características de adaptación con el tiempo como consecuencia de su experiencia. ¿Si es así, cómo lo hacen?

3.4.6 Predicción

La predicción es fundamental para el éxito en la toma de decisiones. Si los rasgos de adaptación de un agente o procedimientos de aprendizaje se basan en la estimación de las consecuencias futuras de las decisiones, ¿cómo los agentes predicen las condiciones futuras (ya sean ambientales o internas) que experimentarán? En su caso, ¿qué modelos internos utiliza el agente para estimar las condiciones futuras o consecuencias de

sus decisiones? ¿Qué predicciones tácitas u ocultas están implicadas en estos supuestos del modelo interno?

3.4.7 Detección

¿Qué variables de estado internas o ambientales son asumidas por los individuos para detectar y considerar en sus decisiones? ¿Qué variables de estado de un individuo pueden ser percibidas por otros individuos o entidades (por ejemplo, las señales que otro individuo envía intencionalmente o no)? ¿A qué nivel se realiza la detección? La detección, a menudo, se supone que es local, pero puede ocurrir a través de redes o incluso a nivel global. Por ejemplo, un recolector en una ubicación puede detectar los niveles de recursos de todos las demás ubicaciones a las que se podría mover. Si los agentes se detectan el uno al otro a través de las redes sociales, ¿es la estructura de la red impuesta o es emergente? ¿Los mecanismos por los cuales los agentes obtienen información son modelados explícitamente, o se supone simplemente que los individuos conocen estas variables?

3.4.8 Interacción

¿Qué tipos de interacciones entre agentes se supone que existen? ¿Existen interacciones directas en las que los individuos se enfrentan y afectan unos a otros, o hay interacciones indirectas; por ejemplo, a través de la competencia por un recurso mediador? Si las interacciones implican la comunicación entre agentes, ¿cómo está representada dicha comunicación?

3.4.9 Aleatoriedad

¿Qué procesos son modelados partiendo del supuesto que son al azar o parcialmente azarosos? ¿Se usa la aleatoriedad, por ejemplo, para reproducir la variabilidad en los procesos para los que no es importante

modelar las causas reales de la variabilidad? ¿Son los procesos aleatorios usados para inicializar el modelo?

3.4.10 Colectivos

¿Existen colectivos representados en el modelo? Si existen dichos colectivos, ¿cómo son representados dichos colectivos? ¿Una propiedad emergente de los individuos (por ejemplo, una bandada de pájaros que se ensambla como resultado de los comportamientos individuales) es un colectivo particular?, o ¿el colectivo es simplemente una definición establecida por el modelador (por ejemplo, el conjunto de individuos con ciertas propiedades), que se define como una especie separada de la entidad con sus propias variables de estado y rasgos?

3.4.11 Observación

¿Qué salidas o resultados del modelo son necesarios para observar la dinámica interna y su comportamiento a nivel de sistema? ¿Qué datos son recolectados del modelo basado en agentes para probarlo, entenderlo y analizarlo? ¿Cómo y cuándo son recolectados dichos datos?

3.5 Protocolo ODD: detalles

La parte "detalles" del protocolo ODD se compone de tres elementos: 1) inicialización; 2) datos de entrada; y 3) submodelos. A continuación explicaremos cada uno de estos elementos.

3.5.1 Inicialización

Preguntas: ¿Cuáles son los valores iniciales de sus variables de estado de cada entidad? ¿Son los valores iniciales elegidos arbitrariamente o con base a los datos? ¿La inicialización es siempre la misma o varía entre simulaciones?

Explicación: Se debe describir cómo establecemos las condiciones iniciales del modelo al principio de la simulación porque los resultados del modelo, frecuentemente, dependen de dichas condiciones iniciales. Ejemplos de condiciones iniciales son el número de agentes creados y los valores dados a sus variables de estado (localización, tamaño, color, etc.). Conocer los datos de inicialización es una condición necesaria para la correcta replicabilidad del modelo. Si no se tienen los datos correctos con los que se "corrió" el modelo, no es posible replicarlo. A veces, el propósito de un modelo es analizar las consecuencias de su estado inicial, y otras veces modeladores se esfuerzan por reducir al mínimo el efecto de las condiciones iniciales sobre los resultados.

3.5.2 Datos de entrada

Preguntas: ¿El modelo usa insumos de fuentes externas tales como archivos de datos u otros modelos para representar procesos que cambian a través del tiempo?

Explicación: En el modelado de sistemas reales, las dinámicas son a menudo impulsadas en parte por una serie temporal de las variables ambientales, a veces llamada forzamientos externos; por ejemplo, la precipitación anual en sabanas semiáridas puede afectar a la variable de humedad del suelo, la cual es una variable de estado de celdas de la cuadrícula y, por tanto, afecta la dispersión y crecimiento de los árboles. A menudo tiene sentido utilizar series de tiempo observadas de variables del ambiente, de modo que sus cualidades estadísticas (media, la variabilidad, autocorrelación temporal, etc.) sean realistas.

3.5.3 Submodelos

Preguntas: ¿Cuáles son, en detalle, los submodelos que representan los procesos enumerados en la "visión general y programación de procesos"? ¿Cuáles son los parámetros del modelo, sus dimensiones y valores de referencia? ¿Cómo fueron diseñados o elegidos los submodelos, y cómo fueron parametrizados y puestos a prueba?

Explicación: Los submodelos se presentan en detalle y de forma completa ya que representan procesos que realiza el modelo. La descripción factual de los submodelos, esto es, las ecuaciones y algoritmos, debe ser lo primero en describirse y deben separarse claramente de la información adicional. Debe explicarse el porqué de la formulación del submodelo. Si la parametrización no se discute fuera de la descripción del protocolo ODD, esta puede ser incluida aquí. Las definiciones de los parámetros, unidades y los valores utilizados (en su caso) deben ser presentados en tablas.

Cualquier descripción de un modelo basado en agentes y sus submodelos es *ad hoc* y carece de credibilidad si no hay una justificación de por qué y cómo las formulaciones del modelo fueron elegidas o cómo se han diseñado y probado nuevas formulaciones. Debido a que el modelado basado en agentes es nuevo y carece de una tradición fundamentada en la teoría y los métodos establecidos, esperamos que las descripciones ODD incluyan niveles apropiados de explicación y justificación de las decisiones de diseño que ilustran, aunque esto no debe interferir con el objetivo principal de dar una descripción del modelo concisa y fácil de leer. La justificación puede ser muy breve en los apartados de la visión general y los conceptos del diseño, pero la descripción completa de submodelos es probable que incluya referencias a la literatura relevante, así como la aplicación independiente, pruebas, calibrado y análisis de los submodelos.

CAPÍTULO IV
PROGRAMACIÓN DEL MODELO NETLOGO

4.1 Comenzando a trabajar con Netlogo

El objetivo de este capítulo es introducir al lector al lenguaje Netlogo. Para ello, primero explicaremos sus partes constitutivas y luego, mediante un ejemplo sencillo, iremos explicando la programación en Netlogo y la construcción de la interfaz gráfica. En concreto, nos centramos en la creación de agentes, su interacción con su entorno, los procesos que les permiten interactuar con el entorno y los cambios que producen en el entorno.

Le recordamos al lector que este libro no es un manual de Netlogo, sino una introducción práctica. Tal vez queden muchas dudas respecto a ciertos comandos, pero no es nuestra intención ser exhaustivos en el lenguaje. Al final del capítulo incluimos una lista de elementos de Netlogo. La intención de esta lista es reunir los elementos que creemos esenciales. Muchos de dichos elementos están explicados en este mismo capítulo, pero otros están explicados en los tutoriales de Netlogo de ayuda en línea en inglés.

4.2 Un viaje rápido por Netlogo

Antes de nada, el lector debe ir a la página de Netlogo (https://ccl.northwestern.edu/netlogo/) para descargar la última versión de Netlogo e instalarla en su sistema operativo. Aquí damos la dirección electrónica, pero el lector podrá *googlear* e ir rápidamente a ella.

En este libro trabajamos con la versión 5.3.1 en español para Windows y usaremos la interfaz en español. Gracias a que Netlogo es un lenguaje interpretado en Java©, no hay problema con respecto al sistema operativo con el que se cuente. Netlogo funcionará de la misma

manera para usuarios de Windows©, Mac OS X y las distintas distribuciones de Linux siempre y cuando se tenga instalado el intérprete de Java. Conviene tener siempre la versión más actualizada de Java.

Una vez que el lector se encuentre en la página de Netlogo, debe ir a "Download" y descargar la versión adecuada para su sistema operativo. Una vez que el lector haya instalado Netlogo en su computadora, ábralo.

La interfaz gráfica es una ventana ordinaria de Windows o Mac que incluye un menú habitual de Windows (Figura 4.1). Haga clic en el menú de "Ayuda" de la interfaz gráfica. Inmediatamente aparecerá un cuadro de diálogo con los siguientes comandos: a) Buscar en el diccionario (F1); b) Guía del usuario Netlogo; c) Diccionario Netlogo; d) Comunidad de usuarios de Netlogo; e) "Introduction to agent-based modeling"; f) Donar a Netlogo; g) "About Netlogo 5.3.1". No obstante, ninguna de estas opciones se ha traducido al español.

En la interfaz gráfica, haga clic en el menú "Archivo". El comando "Archivo" nos lleva a un menú con las siguientes opciones: nuevo, abrir, etc. (Figura 4.2). La opción más interesante es la "Biblioteca de modelos" (Ctrl + M). Esta opción despliega una vasta colección de modelos prefabricados y ordenados por categorías. El lector puede explorar por su cuenta dichos modelos y tratar de entenderlos. No es la intención de este libro ahondar en todos ellos.

Figura 4.1
Despliegue de la ventana de Netlogo y la pestaña "Ayuda"

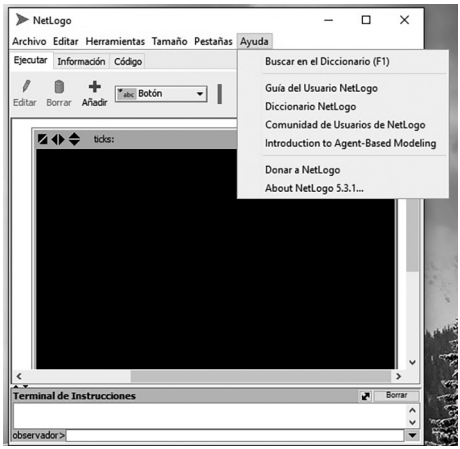
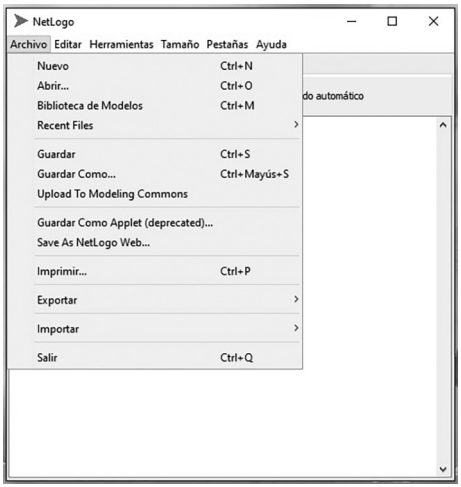


Figura 4.2
Despliegue de la pestaña "Archivo" y su menú de comandos



Volvamos a la Figura 4.1 y analicémosla. Está dividida en tres pestañas: "Ejecutar", "Información" y "Código". Dichas pestañas están relacionadas con el hecho de que Netlogo no es solamente un lenguaje de programación basado en agentes, sino también es un "ambiente" de programación y un "sistema de programación". A continuación se describe cada pestaña de forma breve:

1. La pestaña "Ejecutar" (Figura 4.1) es una interfaz gráfica donde se diseña el "laboratorio" y donde existe de forma explícita el "mundo" de los agentes. Nótese que en la parte inferior de la ventana está la "terminal de instrucciones". En ésta se puede escribir el código ejecutable de forma directa. También aparecen resultados del programa o de la simulación. Intente, por ejemplo, introducir en esta parte el código 1 + 1 (respete los espacios), pulse "Enter" y verá que Netlogo le despliega el resultado "2".
2. La pestaña "Información" (Figura 4.3) es una interfaz gráfica diseñada explícitamente para "explicar" el modelo. Aunque a continuación se describen las partes que, por defecto, tiene esta pestaña, recomendamos al lector seguir el protocolo estándar ODD para

explicar el modelo. Las partes en las que, por defecto, se divide esta pestaña son a) "What is it?" (¿qué es esto?) y es dónde el autor del modelo explica someramente de qué se trata el modelo; b) "How it works?" (¿cómo funciona?): se explica cómo funciona el modelo y se establecen las reglas que deben seguir los agentes y que darán como resultado el comportamiento global del modelo; c) "How to use it?" (¿cómo usarlo?): se proporcionan las instrucciones de cómo usar el modelo; d) "Things to notice" (cosas que resaltar): se sugieren los aspectos a los que prestar atención al correr el modelo. Por ejemplo, el movimiento de una variable durante la ejecución del modelo puede llevar a que emerjan comportamientos globales diferentes; e) "Things to try" (cosas que probar): es una invitación a "jugar" con el modelo. El modelador sugiere al usuario que pruebe moviendo botones, parámetros, variables y que observe el comportamiento del modelo; f) "Extending the model" (extendiendo el modelo): es una invitación a que el usuario modifique el modelo. Se sugiere mover el código fuente de forma directa; g) "Netlogo features" (características Netlogo): en esta parte se remarcan las características especiales que se han usado de Netlogo; h) "Related models" (modelos relacionados): se citan los modelos parecidos o relacionados; i) "Credits and references" (créditos y referencias): se establecen los créditos y las referencias necesarias con respecto a la autoría del modelo.

3. La pestaña "Código" (Figura 4.4) es donde se escribe el código Netlogo, el cual luego será interpretado en la ventana "Ejecutar". La ventana código tiene una palomita que se pone en verde cuando el código está bien escrito. Se hace clic sobre la palomita y se realiza la compilación (verificación de la correcta sintaxis del lenguaje). Si el código cumple con la sintaxis adecuada, la palomita se tornará verde; y si no, aparecerá un mensaje de error.

4.3 Programación con Netlogo paso a paso

A continuación desarrollaremos un ejemplo muy sencillo de código Netlogo. Usaremos un modelo desarrollado en Railsback y Grimm (2012)

llamado “¡A la caza de champiñones!” (MushroomHunt). En este modelo dos cazadores se mueven por el mundo negro buscando champiñones (que son parcelas en color rojo). Si una tortuga encuentra un champiñón, éste cambia de color de rojo a amarillo.

Abrimos Netlogo y aparecerá la ventana del sistema. A continuación damos clic en la ventana en el menú “Archivo” y damos clic en “Nuevo”. Esto creará un archivo nuevo. Nótese que también pudimos usar la secuencia de control de teclado “Ctrl + N”. *Salvaremos* (archivaremos) nuestro nuevo archivo con el nombre “Champiñones.nlogo” por medio del comando “Salvar como” (Figura 4.5).

Figura 4.3.
Ventana relativa a la información del modelo

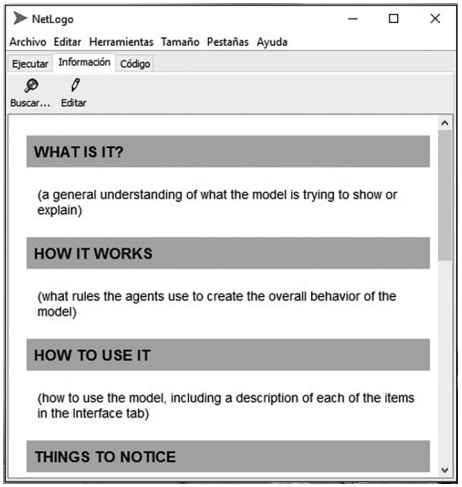


Figura 4.4

Ventana donde se escribe el código del modelo

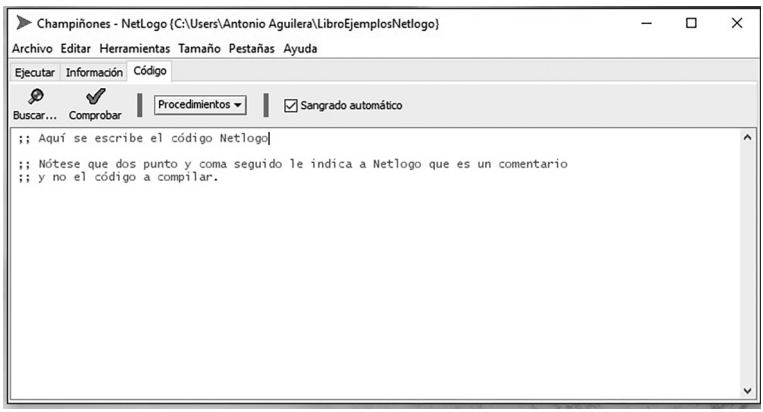
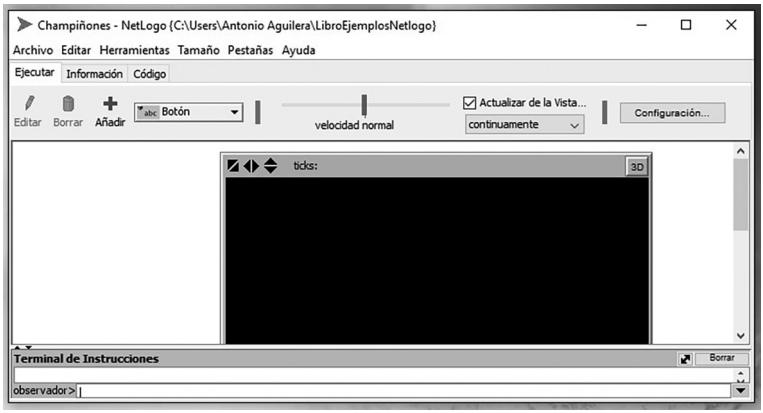


Figura 4.5

El archivo "Champiñones" en la ventana "Ejecutar"



4.3.1 Pestaña información: Descripción del modelo "a la caza de champiñones"

En el Capítulo III resaltamos la relevancia de utilizar un protocolo estandarizado para la descripción de los modelos en las ciencias sociales

y, en concreto, recomendamos el uso del protocolo ODD. Para hacer la descripción del modelo "¡A la caza de champiñones!" siguiendo el protocolo ODD, haga clic en la pestaña información de la interfaz gráfica (Figura 4.3). Pulsando el botón "Editar" se habilita la edición del texto, y en la ventana escribimos lo siguiente. Pulsar de nuevo "Editar" para salir del modo edición.

ODD PROTOCOL "¡A la caza de champiñones!" (MushroomHunt)
OVERVIEW (VISIÓN DE CONJUNTO)
Propósito
Mostrar de forma sencilla cómo trabaja Netlogo. Para ello, se modelan dos cazadores que se mueven por el mundo (de color negro) buscando champiñones (que son celdas en color rojo) mientras dejan un rastro en color amarillo de su recorrido por el mundo. Si un cazador encuentra un champiñón, éste cambia de color de rojo a amarillo.

Entidades, estados, variables y escalas
Existen dos agentes: cazadores (tortugas) y champiñones (parcelas).
Existe una variable global llamada *num-cluster* que determina el número de cúmulos de champiñones.
La escala espacial es la estándar dada por *default* en Netlogo. Esto es, un universo de tipo toroidal de 33×33.
Los observables del modelo son:

a) El número de ticks necesarios para "comer" todos los champiñones.
b) La tasa de champiñones remanente después de *x* número de *ticks*.
c) La probabilidad de encontrar un champiñón después de *x* número de *ticks*.

Process overview and scheduling
Al principio de la simulación se inicia el modelo creando los agentes.
El tiempo transcurre de forma discreta por *ticks*. En cada *tick* se ejecuta el procedimiento de búsqueda ("Search").

DESIGN CONCEPTS
Basic principles
El modelo es una replicación del modelo sugerido en Railsback y Grimm (2012), el cual está inspirado en el modelo Sugar-Space, de Epstein y Axtell (1996).

Emergence
Debido a la sencillez del modelo, éste se limita a mostrar un mecanismo simple de búsqueda determinista. No hay estocacidad en el modelo en el sentido estricto de tener un mecanismo probabilístico que determinara la probabilidad de encontrar un champiñón. Sin embargo, la caminata del cazador (tortuga) sí es de tipo aleatorio.
Si se deja que el número de *ticks* tienda a números muy grandes, las tortugas comerán todos los champiñones.

Adaptation
El modelo muestra una adaptación muy simple a las condiciones dadas. Los cazadores sólo cambian su método de búsqueda si encuentran un champiñón. En ese caso, reducen el ángulo de avance hacia la derecha o hacia la izquierda, pasando de un ángulo de a lo más 90 grados a un ángulo más estrecho, a lo más 10 grados. Esto lo hace de forma aleatoria y se basa en que los champiñones están agrupados.
No existen mecanismos tales cuales de toma de decisiones.

Objetives
El objetivo implícito es que las tortugas "cacen" todos los champiñones. No existe un mecanismo explícito que mida un objetivo.

Learning
No existen mecanismos de aprendizaje.

Sensing
La dirección de la tortuga es usada como un mecanismo sensor. Al avanzar verifica si la parcela sobre la cual avanzó es un champiñón o una parcela vacía.

Interaction
Se produce interacción con el entorno. Cuando la tortuga encuentra un champiñón, la parcela cambia de color rojo a color amarillo.
Las tortugas cazan de forma independiente. No hay interacción entre ambas.

Stochasticity
Las direcciones de avance de las tortugas se definen de forma aleatoria.
Otro elemento estocástico es la creación al azar de los cúmulos de champiñones. Éstos se pueden crear en cualquier parte del espacio 5, y cada uno está formado por 20 champiñones (parcelas) que se encuentran dentro de un círculo de radio 5.

Collectives
No existen colectivos en el modelo. Los agentes (tortugas) son independientes.

Observation
La dinámica del modelo se observa de forma visual al ver la traza que dejan las tortugas al moverse en el espacio.
El cambio de color de los champiñones es otro observable del modelo.

DETAILS
Initialization
El modelo se inicializa con dos tortugas y la creación al azar de 4 cúmulos (*num-cluster*=4) con 20 champiñones cada cúmulo.

Input Data
No hay datos de entrada. Las variables globales son fijas.

Submodels
Submodelo búsqueda (*search*): define la dirección de avance del cazador. Existen dos formas de moverse, la primera está determinada por el tiempo que ha trascurrido sin que la tortuga encuentre champiñones. Si el tiempo está en un rango de cero a veinte, la tortuga tiene un rango de movimiento de 90 grados en el espacio. Si se excedió dicho tiempo, la tortuga restringe su movimiento a 10 grados.

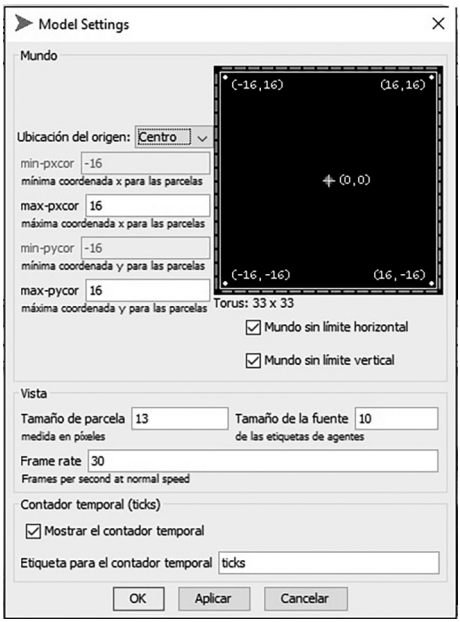
4.3.2 Pestaña código: La interfaz del modelo "a la caza de champiñones"

En la pestaña "Ejecutar", el ambiente del modelo ya está construido. Es la ventana en negro la cual mide 33×33 parcelas (*patches*). Una parcela (*patch*) es un agente en Netlogo que no tiene movilidad, pero tiene ubicación, color y hasta algunas características extras establecidas por el usuario.

Para ver la configuración que, por defecto, tiene el ambiente del modelo, damos clic en el botón "Configuración". Aparecerá la "Ventana de configuración" (Figura 4.6). En esta ventana ("Model settings") está el tamaño del mundo en el que viven nuestros agentes. Por defecto (en adelante, *default*, para utilizar la misma nomenclatura que Netlogo), el mundo mide 33×33 parcelas (*patches*). Dado que en Netlogo las parcelas son agentes, el mundo está formado por 1 089 agentes del tipo parcela. El mundo, aunque se ve cuadrado en la imagen, es en realidad toroidal, ya que Netlogo establece las condiciones de frontera cerrando el mundo sobre sí mismo. El origen del plano cartesiano que conforma nuestro cuadro es el centro, pero lo podemos ubicar donde más nos convenga. Nótese que nuestro "mundo artificial" no tiene límites ni verticales ni horizontales.

En la ventana de "Configuración ("Model settings") es posible cambiar el tamaño de los agentes parcelas. También podemos elegir el tamaño de la fuente de las etiquetas de los agentes. No obstante, en este caso, por sencillez, dejaremos la configuración ("Settings") tal y como está.

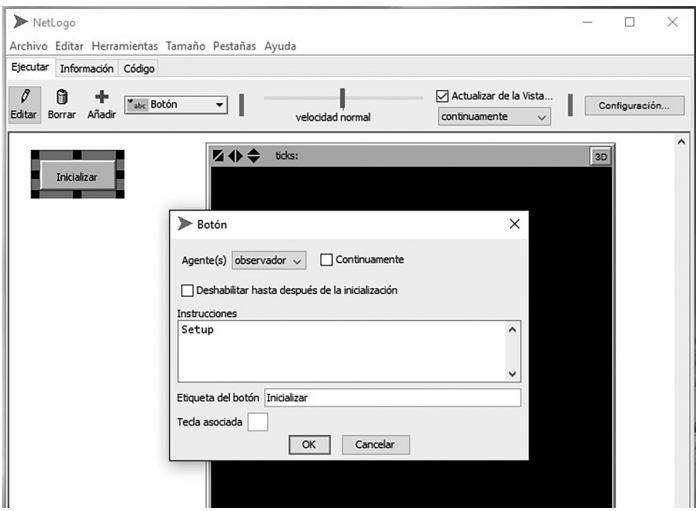
Figura 4.6
La ventana de "Configuración" y sus parámetros



Dado que nuestro objetivo es crear un "laboratorio" donde poder explorar los diferentes comportamientos de nuestros agentes, es necesario que editemos y construyamos dicho laboratorio. Veamos ahora cómo construir nuestro laboratorio en la ventana "Ejecutar" y cómo relacionaremos ciertos objetos de nuestro laboratorio con código Netlogo.

En la ventana "Ejecutar", daremos clic en "Botón" (Figura 4.7). Esto abrirá un cuadro de diálogo con un menú desplegable. Escogemos "Botón" y aparecerá el cursor en forma de cruz. Dándole clic en la pantalla, dentro del espacio vacío de la ventana "Ejecutar" aparecerá un "Botón", al cual nombraremos "Inicializar". Esto lo hacemos en el cuadro de diálogo que aparece automáticamente con el botón en la parte "Etiqueta del botón" (Figura 4.7). Además, en la parte "Instrucciones" escribiremos "Setup". Al dar OK, se creará en nuestra ventana ejecutar un botón con la palabra "Inicializar" en rojo, lo que indica que no está asociada todavía a ningún código.

Figura 4.7
Creación del botón "Inicializar"



4.3.3 Pestaña código: Inicializar el modelo "a la caza de champiñones"

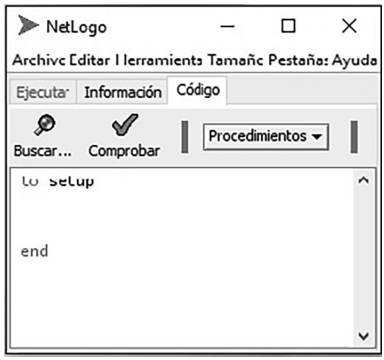
En la ventana de código asignamos código de programación al botón "Inicializar". Una ventaja de Netlogo es que asocia de manera automática los procedimientos o instrucciones a botones en la ventana "Ejecutar". Este simple código le indica a Netlogo que hay un nuevo procedimiento llamado "setup" (Figura 4.8).

```
to setup
end
```

El código se inicia en *to* y termina en *end*. Obviamente, nuestro código está vacío. Todavía no presenta instrucciones, pero ya se ligó de manera automática al botón "Inicializar". Esto lo hacemos al compilar nuestro código con el botón palomita, la cual, como vemos, está en verde, indicándonos que el código cumple las reglas de sintaxis y que además ya se ligó al botón "Inicializar".

Figura 4.8

Ventana de código con el código *SETUP*



Escribamos en el procedimiento el siguiente código para que todas las parcelas (*patches*) cambien su color negro original a rojo (Figura 4.9). El código Netlogo es:

```
to setup
      ask patches [ set pcolor red ]
end
```

Figura 4.9

Código que cambia el color de las parcelas a rojo

Si le damos palomita y compilamos este código, el resultado es que, al pulsar el botón "Inicializar", el mundo cambiará su color de negro a rojo. Recordemos que el *mundo* es el ambiente en el que "viven nuestros agentes". En este caso, Netlogo nos ofrece un mundo de inicio con agentes sin movimiento llamados "parcelas" (*patches*) de color negro, que es el color por *default*.

- El comando "Ask" indica sobre qué tipo de agente se realizará la acción. Nótese que después de "Ask" se escribe el tipo de agente y después se abren los símbolos de [] (*brackets*). Dentro de los *brackets* se escribe el código que ejecutarán los agentes indicados por "Ask".
- El comando "Pcolor" significa "patch_color", o sea, el color de la parcela. En este caso, nuestro comando indica que el color de la parcela es "red" (rojo).
- El comando "Set" indica que se realizará una acción o comando. En este caso, el comando "Set" establece que "pcolor" es "red".

Recordemos que el color rojo indica la presencia de champiñones. Para hacer la simulación interesante, necesitamos un mundo que no esté lleno de champiñones. Para ello, escribamos el siguiente código:

```
to setup
      ca
      ask n-of 4 patches
      [ask n-of 20 patches in-radius 5 [set pcolor red]]
end
```

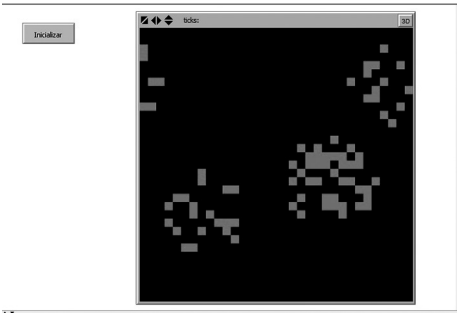
Nuestro nuevo código incorpora nuevos comandos, a saber: "Ca", "N-of" e "In-radius".

- El comando "Ca" significa "clear-all", es decir, limpia la pantalla de nuestro mundo y la coloca con los valores por *default* o predeterminados. Este comando es muy importante ya que si no lo tuviéramos la sesión pasada quedaría grabada y habría una superposición de pantallas.

- El comando “N-of” tiene como sintaxis: “n-of number agentes”. Mediante estos dos argumentos, el número y el tipo agentes, se indica que se selecione (al azar y sin repetición) a 4 agentes tipo parcela, que crearán un cúmulo de champiñoes alrededor.
- El comando “In-radius” tiene como sintaxis: “agentset in-radius number”. El comando “In-radius” informa de un *agentset* (conjunto de agentes), cuya distancia al agente que llama es menor o igual que un número. La distancia de una parcela se mide desde el centro de la parcela.

El código anterior primero limpia la pantalla y pone en modo predeterminado todo el mundo artificial (mediante el comando “ca”). A continuación, genera una lista de agentes tipo parcela (*patches*) de tamaño 4. Dicha lista la genera usando todo el “mundo”. Después, se generan listas de 20 parcelas que no estén separadas más de 5 parcelas entre sí y las colorea de rojo. El resultado lo podemos ver en la Figura 4.10.

Figura 4.10
Resultado del código. Se muestran cuatro cúmulos de parcelas rojas (aquí en gris) que representan los champiñones



Vamos a hacer que las 4 parcelas sean un parámetro del programa, al que llamaremos “num-cluster”. Para ello en la ventana “Código” definimos la variable como global con la instrucción: “global [num-cluster]”, asignamos a esa variable el valor 4 con la instrucción: “set num-clusters 4”, de forma que el nuevo código es:

```
globals
[
        num-clusters
]
to setup
ca
set num-clusters 4
ask n-of num-clusters patches
        [ask n-of 20 patches in-radius 5 [set pcolor red]]
end
```

El código anterior cambia el color del entorno para indicar dónde hay champiñones. Pero aún no se han creado los agentes que cazan los champiñones. Dado que los cazadores se mueven, serán agentes del tipo tortuga.

Llamar tortugas a los agentes móviles es una herencia del lenguaje Logo en el que se basa Netlogo. Logo es un lenguaje de programación de alto nivel, en parte funcional, en parte estructurado; de muy fácil aprendizaje, razón por la cual suele ser el lenguaje de programación preferido para trabajar con niños y jóvenes. Fue diseñado con fines didácticos por Danny Bobrow, Wally Feurzeig y Seymour Papert, los cuales se basaron en las características del lenguaje Lisp. Logo fue creado con la finalidad de usarlo para enseñar programación y puede usarse para enseñar la mayoría de los principales conceptos de la programación, ya que proporciona soporte para manejo de listas, archivos y entrada/salida.

Para crear las tortugas, Netlogo tiene comandos especiales. El siguiente código, que hay que incluir justo antes de "End", crea 2 tortugas de tamaño 2, pintadas de amarillo, que al moverse por el mundo dejarán rastro del camino recorrido.

```
crt 2
[
        set size 2
        set color yellow
]
```

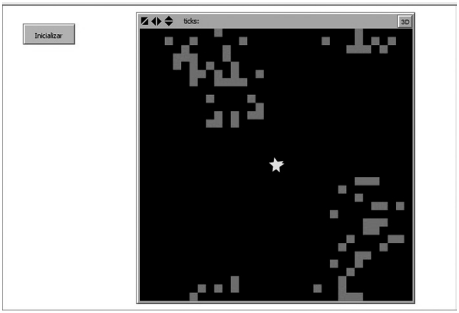
Nuestro nuevo código incorpora nuevos comandos, a saber: "Crt", "Pendown".

- El comando para la creación de las tortugas es “Crt”. Por *default*, las tortugas siempre se crean en el centro del mundo artificial y su forma predeterminada es la de flechas.
- El comando “Pen-down” establece que la “pluma” de la tortuga está bajada. Hay que recordar que Netlogo está basado en Logo, y en este lenguaje las tortugas tenían propiedades como la de tener una pluma con la cual podrían trazar su camino por el mundo. La idea es que las tortugas dejen un trazo de su trayectoria en el mundo cuando se muevan en él.

Resumiendo, el código original queda como sigue, y el resultado de este código se puede ver en la Figura 4.11. Recuerde siempre guardar los cambios y avances. Netlogo no guarda automáticamente los cambios. Para ello, en la interfaz gráfica, haga clic en el menú de “Archivo” en el comando “Guardar” o “Guardar como”.

Figura 4.11

Creación del mundo artificial. Los champiñones están en cúmulos rojos (aquí grises) y las tortugas están en amarillo (aquí blancos) en el centro




```
globals
[
      num-clusters
]
to setup
ca
set num-clusters 4
ask n-of num-clusters patches
[ask n-of 20 patches in-radius 5 [set pcolor red]]
```

```
crt 2
[
        set size 2
        set color yellow
        pen-down
]
end
```

4.3.4 Pestaña código: correr el modelo "a la caza de champiñones"

Tras crear los agentes y definir el mundo en el que interactúan, es el momento de definir las acciones que realizarán los agentes. En el modelo de caza de champiñones sólo hay una acción que realizar: "buscar champiñones", y que deberán realizar nuestras dos tortugas.

Para ello, incluimos un botón nuevo llamado "correr" en la pantalla "Ejecutar", al cual asociaremos con el procedimiento "go". En la ventana de código escribimos el siguiente código:

```
to go
        tick
        ask turtles [search]
end
```

El procedimiento "go" avanza el instante de tiempo (*tick*) en uno y actualiza tanto las parcelas como los gráficos que tenga el laboratorio. Luego, las tortugas invocan el procedimiento "search".

No obstante, si tratamos de compilar el código nos arroja el siguiente error: "Nothing names SEARCH has been defined". Para corregirlo, debemos indicar que "search" es un procedimiento, añadiendo después de "end" el siguiente código:

```
to search
end
```

El procedimiento “search” define la dirección de avance del cazador. Existen dos formas de moverse. La primera está determinada por el tiempo que ha trascurrido sin que la tortuga encuentre champiñones. Si el tiempo está en un rango de cero a veinte, la tortuga tiene un rango de movimiento de 90 grados en el espacio. Si se excedió de dicho tiempo, la tortuga restringe su movimiento a 10 grados.

Con base en lo anterior, es necesario definir en las tortugas una variable de estado para contar el tiempo que ha trascurrido sin que la tortuga encuentre champiñones, a la que llamaremos “time-since-last-found”. Para ello, en la ventana de código, después de “globals”, escribimos el siguiente código:

```
turtles-own
[
      time-since-last-found
]
```

Utilizamos “turtles-own” para indicar que “time-since-last-found” es una variable de las tortugas. Nótese que “time-since-last-found” no es un comando especial de Netlogo, sino un nombre arbitrario para indicar un contador de tiempo.

Antes de proceder, es necesario iniciar estas variables escribiendo en el procedimiento “inicializar” las siguientes instrucciones: “set time-since-last-found 999” y “reset-ticks”, y reemplazando el código previo:

```
crt 2
[
      set size 2
      set color yellow
      pen-down
      set time-since-last-found 999
]
reset-ticks
```

Nuestro nuevo código incorpora nuevos comandos, a saber: “Ca”, “N-of” e “In-radius”.

- El comando “Set” indica que la variable “time-since-last-found” se inicializa en 999.
- El comando “Reset-ticks” pone todos los *ticks* (instantes) en cero. Hay que recordar que Netlogo usa los *ticks* como su unidad primaria de contar el tiempo. Cada vez que pasa un *tick*, una serie de eventos es llevada a cabo. Sin embargo, nótese que en sí un *tick* (instante) no es una unidad de tiempo, aunque puede representar el tiempo, el cual puede expresarse en segundos, minutos, días o años. Así, el *tick* es sólo el avance del programa una vez.

En la ventana de código, después de “end” del procedimiento “to go”, escribimos el siguiente código correspondiente al procedimiento “to search”, que permite a las “tortugas” buscar los “champiñones”:

```
to search
        ifelse time-since-last-found <= 20
                [right (random 181) – 90]
                [right (random 21) – 10]
        fd 1
        ifelse pcolor = red
        [
                set time-since-last-found 0
                set pcolor yellow
        ]
        [
                set time-since-last-found time-since-last-found + 1
        ]
end
```

Nuestro nuevo código incorpora nuevos comandos, a saber: “Ifelse”, “Right” y “Fd”.

- El comando “Ifelse” es un comando condicional lógico de tipo “si sucede la condición, hacer; y si no sucede, hacer”. En este caso, la primera condición es con respecto a la variable “time-since-last-found”, y la segunda respecto al color de la parcela.
- El comando “Right” mueve la cabeza de la tortuga hacia la derecha cierto número de grados (o a la izquierda si los grados son negativos).

- El comando “Random” genera *n* números aleatorios entre 0 y n-1.
- El comando “Fd” (forward) establece el número de parcelas que una tortuga avanza en la dirección en la que está apuntando su cabeza.

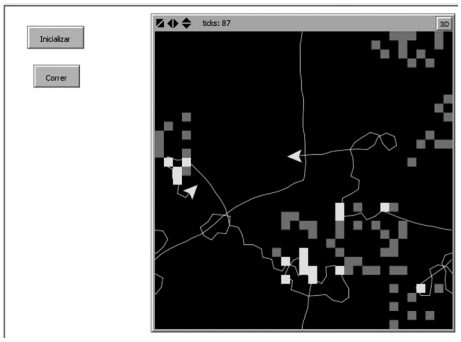
En este caso, si la condición es cierta (esto es, la variable “time-since-last-found” es menor o igual a 20), mueve la cabeza de la tortuga hacia la derecha un número aleatorio entre cero y 180, y le restamos 90. El resultado será el número de grados que la tortuga girará su cabeza. Restamos 90 para obtener tanto números positivos como negativos y que, de esta forma, la cabeza se mueva tanto a la derecha como a la izquierda. Nótese que sólo estamos poniendo una dirección y todavía no avanzamos. Si la condición es falsa (esto es, la variable “time-since-last-found” es mayor o igual a 20), entonces la tortuga gira su cabeza un número al azar entre cero y 20 y le resta 10. Ése será el número de grados que la tortuga girará su cabeza.

Después, el programa invoca el comando “Fd 1” (“Forward 1”), indicando que la tortuga avanzará un paso en la dirección en la que esté apuntando su cabeza. Así, por ejemplo, si la tortuga está apuntando hacia 45 grados a la derecha, la tortuga avanzará un paso en dicha dirección.

El siguiente comando condicional le permite a la tortuga verificar si existen champiñones en su posición. La condición es que la tortuga esté sobre una parcela (*patch*) de color rojo, esto es, que contenga un champiñón. En caso afirmativo, la tortuga pone su variable “time-since-last-found” a cero y la parcela se pinta de color amarillo, indicando con esto que el champiñón fue encontrado. En caso de que la parcela no contenga un champiñón, la variable de la tortuga, “time-since-last-found”, se actualiza con el valor anterior más uno.

En la Figura 4.12 se muestra el funcionamiento del programa. El programa fue primeramente inicializado y después se hizo clic sobre el botón “Correr”, lo que provoca que las tortugas se muevan en una dirección aleatoria, y cuando encontraba un “champiñón” (una parcela en rojo), cambiaba de color (parcela amarilla) para indicar que lo había encontrado. Nótese que las tortugas tienen forma de flecha y dejan un “rastro” de amarillo por donde han pasado.

Figura 4.12
Gráfica donde se muestra el funcionamiento del programa "Champiñones"



4.4 Mini-referencia a la programación Netlogo

En esta sección proveemos una pequeña lista de elementos importantes que destacar tanto en la interfaz como en el código. Recomendamos al lector que lea el tutorial y consulte regularmente el diccionario. Con la práctica, el lector se percatará de que Netlogo es muy sencillo e intuitivo.

4.4.1 Tabla de interfaz

- *Mundo*: Úsese para cambiar los ajustes o la visión
  - Localización del origen.
  - Tamaño del mundo.
  - Contador de instantes (*ticks*).
- *Monitores de agentes*. En la ventana del mundo, dé clic-derecho sobre una parcela o una tortuga para inspeccionar sus variables, seguir el comportamiento de la tortuga o la parcela.
- *El centro de comandos*. Sirve para introducir de forma directa comandos para las tortugas o las parcelas.
- *Botones*. Ejecutan procedimientos desde la interfaz. Los botones más comunes son:

    - Inicializar ("Setup").
    - Correr ("Go"). Este botón puede modificarse para que se realice para siempre ("forever") o bien de paso en paso.
    - Paso ("Step"): Un botón que ejecuta el procedimiento "to go" una vez.
- *Deslizadores* ("Sliders"). Botones que definen e inicializan variables globales.
- *Switches*. Botones que definen e inicializan una variable *booleana*.
- *Otros objetos de interfaz*. *Choosers*, *inputs*, monitores, *outputs*.
- *Gráficos*.

4.4.2 Tabla de procedimientos

- *Contextos*. Establece cuáles agentes pueden ejecutar cada pieza del código.
    - Observador, parcela, tortuga (raza (*breed*)).
    - Cada procedimiento tiene un contexto.
    - Cada comando "Ask" establece un nuevo contexto.
    - Los contextos son determinados por:
        - Una declaración "Ask".
        - Usando primitivas (código) o variables específicas a un contexto.
        - Usando la declaración "Global" si no hay "Ask".
    - Tipo de variables:
        - Numéricas.
        - *Booleanas* (verdadero/falso).
        - Color.
        - Agente.
        - Conjunto de agentes.
        - Lista.
    - Variables preconstruidas ("built-in") y conjunto de agentes.
        - Conjunto de agentes globales y variables:
            - Parcelas (conjunto de agentes).
            - Tortugas, razas (*breed*) (conjunto de agentes).
            - Ligas (conjunto de agentes).

        - *Ticks* (instantes) (número de veces que el comando *tick* es ejecutado.
    - Variables de parcela:
        - Pcolor.
        - Pxcor, pycor (de tipo entero).
        - Plabel (etiqueta de parcela).
    - Variables de tortuga:
        - Color.
        - Xcor, ycor (números reales, las coordenadas son continuas).
        - Label (etiqueta).
    - Comandos para definir variables:
        - Globals.
        - Patches-own (pertenecientes a las parcelas).
        - Turtles-own (pertenecientes a las tortugas).
        - Links-own (pertenecientes a las ligas).
    - El botón "Comprobar" ("Check"). Verifica la sintaxis del código.

4.5 Simulaciones con Netlogo

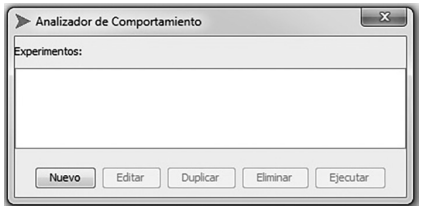
Una vez descrito y programado el modelo, es el momento de hacer un análisis estadístico de los resultados que se obtienen con el modelo.

4.5.1 El analizador de comportamiento

En la interfaz gráfica (Figura 4.1), haga clic en el menú "Herramientas". El comando "Herramientas" nos lleva a un menú con las siguientes opciones: monitor de variables, monitor de tortugas, etc. Aquí lo interesante es la opción "Analizador de comportamiento" (Ctrl + Mayúsculas + B). El "Analizador de comportamiento" es una potente herramienta de Netlogo que permite tanto diseñar como llevar a cabo experimentos. Al seleccionar la opción "Analizador de comportamiento" en "Herramientas" de la barra de menú, se despliega una ventana que nos

permite diseñar un nuevo experimento o ejecutar un experimento previamente diseñado (Figura 4.13).

Figura 4.13
Ventana del Analizador del Comportamiento



Para diseñar un experimento, pulsamos el botón "Nuevo", dado que la lista de experimentos inicialmente estará vacía. En la ventana que se despliega a continuación (Figura 4.14) está dividida en varias partes relevantes. En la primera parte incluye el nombre del experimento y los valores de las variables globales definidas en la pestaña "Ejecutar" (asociados a deslizadores, interruptores y seleccionadores) o en la pestaña de código, que se desean variar.

Por defecto, Netlogo asigna el nombre "Experiment" al nuevo experimento. Pero es posible hacer el cambio de nombre para que tenga sentido para nosotros.

Siguiendo con el análisis de la ventana de experimento, vemos un espacio donde podemos establecer cómo variarán nuestras variables. Estos valores pueden ser unos valores concretos o bien un intervalo. En este caso, debido a la sencillez del modelo ¡a la caza de champiñones!, no tenemos variables que mover. Así que esta parte quedaría en blanco.

["variable" 100 200] ;; para los valores concretos 100 y 200
["variable" [100 1 200]] ;; para los valores 100, 101, 102, 103, … hasta 200.
["variable" [100 10 200]] ;; para los valores 110, 120, 130, … hasta 200.

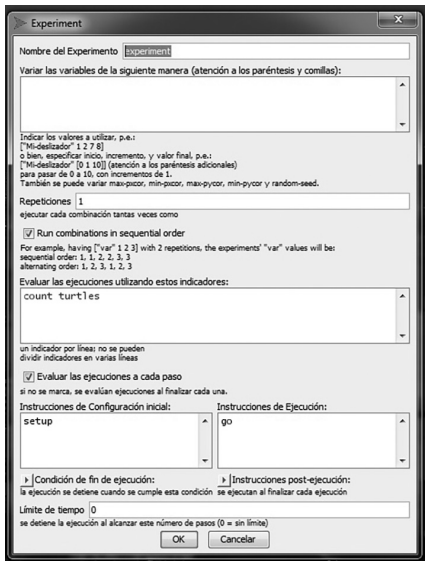
Asimismo, se indica el número de repeticiones y los resultados que queremos mostrar. Si la casilla "evaluar las ejecuciones en cada paso" está

activada, los resultados se registran en cada *tick*, o de lo contrario se registran al final de la simulación. Esto es una gran ventaja ya que de forma automática el modelo se ejecutará las veces que lo indiquemos y podremos generar un análisis estadístico con los datos obtenidos.

La opción de combinaciones en orden secuencial está activada por defecto ("Run combinations in sequential order"). En este caso, se realizan las repeticiones especificadas para una combinación de valores de las variables. Tras ello, se modifican la combinación de valores y se realizan las repeticiones especificadas y así, sucesivamente, hasta combinar todos los parámetros. Si la opción de combinaciones en orden secuencial está desactivada, se realizan las simulaciones para todas las combinaciones de parámetros, y el proceso se repite hasta realizar las repeticiones especificadas.

Lo siguiente es establecer los observables del modelo, es decir, qué vamos a medir. Por defecto, el indicador que se proporciona es el número de tortugas ("count turtles"). Obviamente, nosotros modificaremos esta parte cuando expliquemos cómo construir un experimento con nuestro modelo.

Figura 4.14
Ventana de Experimento



En la opción de evaluar las ejecuciones paso a paso, se especifica cómo se construirá la tabla de salida. Si marcamos la casilla, tendremos los resultados de los observables para cada *tick* de ejecución. Si lo desactivamos, sólo tendremos los valores de los observables en el tiempo final.

Después se encuentran las instrucciones de configuración inicial y de ejecución del modelo. Por *default* éstas son "Setup" y "Go". Es importante que nuestro código tenga dichas instrucciones para poderse ejecutar de forma correcta.

Lo siguiente es establecer las condiciones para que el programa se detenga. Aquí podríamos establecer, por ejemplo, la condición de cuando ya no haya champiñones que cazar. Las instrucciones posejecución son comandos que se ejecutan una vez que el programa se haya ejecutado. Por ejemplo, sacar la tasa del número de champiñones que quedaron sin comer después de 100 *ticks*.

Por último, está el número de *ticks* que queremos ejecute el programa. Esto se establece en el "Límite de tiempo". Si colocamos 0, no hay límite. Hay que tener cuidado al establecer sin límite de tener una condición que detenga el programa, ya que si no se entraría en un ciclo infinito y nunca pararía el programa. Se puede usar el número de *ticks* como una condición de detención del programa.

4.5.2 Experimentos con el modelo

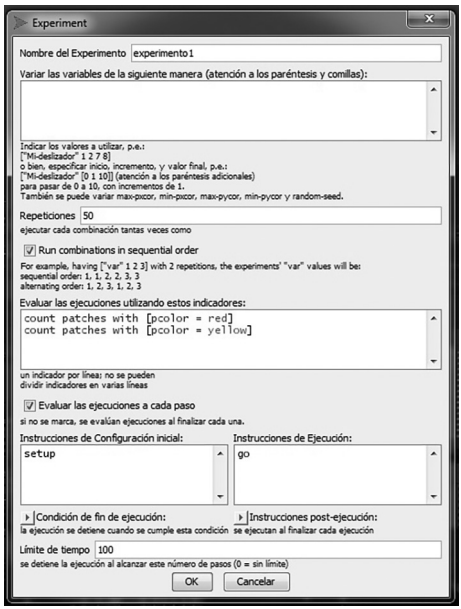
Para realizar el primer experimento del modelo, en la ventana de "Nombre del experimento" escribimos "Experimento1", luego pondremos el siguiente código en la parte de "Evaluar las ejecuciones utilizando estos indicadores" (Figura 4.15).

```
count patches with [pcolor = red]
count patches with [pcolor = yellow]
```

Este código nos indica que evaluaremos las ejecuciones contando el número de parcelas de cada color (rojo y amarillo). Cada *tick* quedará registrado y haremos que el programa funcione sólo por 100 *ticks*. Esto

lo haremos estableciendo el "Límite de tiempo" en 100. Ya que el modelo es extremadamente simple, no hay variables iniciales que mover. Nótese que en "Repeticiones" hemos establecido 50. Esto significa que tendremos 50 veces ejecutado el modelo por solamente 100 *ticks* cada ejecución (Figura 4.15).

Figura 4.15
Experimento 1



Al darle clic en "OK", el experimento queda guardado en nuestra ventana del analizador y aparece con el nombre "Experimento 1 (50 runs)" (Figura 4.16). Ahora damos clic en el botón "Ejecutar" (Figura 4.16), y aparece una ventana con las opciones de ejecución ("Run options") (Figura 4.17), que se despliega para configurar el número de simulaciones que se van a realizar en paralelo (por defecto, aparece el número de microprocesadores del ordenador) y la forma de guardar los datos.

Figura 4.16

Analizador del comportamiento con el experimento 1 cargado

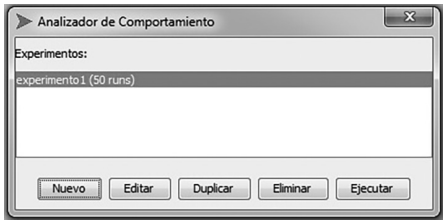
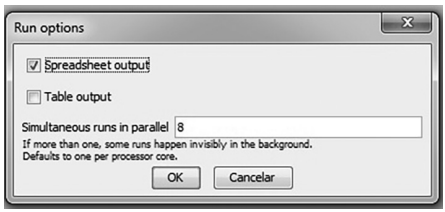


Figura 4.17

Opciones de ejecución



Si se activa la casilla "Spreadsheet output", los datos se disponen en filas (para tratarlos con una hoja de cálculo); mientras que si se activa la casilla "Table output", los datos se disponen en columnas (para tratarlos con una base de datos). Ambas casillas pueden seleccionarse a la vez. Si ocurre una interrupción de la simulación, sólo se guardan los datos de las simulaciones realizadas hasta que se produjo el error en la opción "Table output". Nótese que hemos marcado como documento de salida de los resultados de las 50 ejecuciones, una hoja de cálculo (*spreadsheet*) tipo Excel. Esto es de gran ayuda ya que podremos usar de forma inmediata los resultados del modelo y calcular sus estadísticas.

Damos clic en "OK" y se abrirá una ventana en donde podremos escoger la carpeta donde queremos que se guarde la hoja de cálculo. Una vez escogida la carpeta y el nombre, se ejecutarán las 50 ejecuciones. En nuestro caso, hemos escogido el nombre "Champiñones Experimento 1 –spreadsheet". Una vez que se nombra el fichero, se despliega una ventana que nos permite controlar la simulación para que vaya más rápido (desactivando las opciones "Update view" y "Update plots and

monitors") o para abortarla (pulsando "Abort"), con el botón derecho se deactiva la opción

En la Tabla 4.1 se muestra una parte del contenido de la hoja de cálculo de Excel: al observar los resultados de las 50 repeticiones del experimento observamos que 100 *ticks* nos son suficientes para que las dos tortugas "cacen" todos los champiñones.

Al inicio de la simulación, se crean en promedio 77.96 champiñones. Al sacar la estadística de champiñones cazados (parcelas amarillas), encontramos que en 100 *ticks* se cazan 14.08 en promedio. Calculamos la tasa de efectividad de la caza de champiñones dividiendo el número promedio de champiñones cazados entre el número promedio de champiñones totales: (14.08/77.96)*100 = 18.06 %. Esto significa que hay una eficiencia del 18.06 % por parte de las tortugas en cazar champiñones bajo la condición de 100 *ticks*. ¿Qué ocurre si aumentamos el número de *ticks*? Por intuición, la efectividad debe crecer.

Tabla 4.1

Visión parcial de la hoja de cálculo

"Champiñones Experimento 1 –spreadsheet

BehaviorSpace results (NetLogo 6.0) Champinones.nlogo experimento1 03/05/2017 13:31:10:719 -0600				
min-pxcor	**max-pxcor**	**min-pycor**	**max-pycor**	
-16	16	-16	16	
[run number]	1	1	2	2
[reporter]	count patches with [pcolor = red]	count patches with [pcolor = yellow]	count patches with [pcolor = red]	count patches with [pcolor = yellow]
[final]	63	16	61	18
[min]	63	0	61	0
[max]	79	16	79	18
[mean]	70.7227723	8.27722772	67.009901	11.990099
[steps]	100	100	100	100
[all run data]	count patches with [pcolor = red]	count patches with [pcolor = yellow]	count patches with [pcolor = red]	count patches with [pcolor = yellow]

Tabla 4.1

Visión parcial de la hoja de cálculo

"Champiñones Experimento 1 –spreadsheet (continuación)

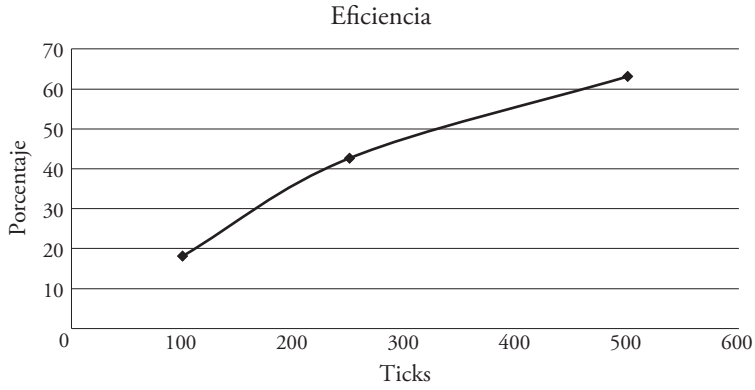
BehaviorSpace results (NetLogo 6.0) Champinones.nlogo experimento1 03/05/2017 13:31:10:719 -0600				
min-pxcor	**max-pxcor**	**min-pycor**	**max-pycor**	
	79	0	79	0
	79	0	77	2
	79	0	77	2
	79	0	77	2
	79	0	76	3
	79	0	75	4
	79	0	75	4
	79	0	75	4
	79	0	74	5
	79	0	74	5
	79	0	74	5
	78	1	74	5
	78	1	74	5
	78	1	74	5

Si hacemos un segundo experimento con 250 *ticks*, obtenemos que el promedio de champiñones creados es de 77.94, lo cual es consistente con nuestro conjunto de experimentos anteriores. Los champiñones "cazados" son en promedio 33.22 para las 50 repeticiones. Por tanto, el porcentaje de efectividad de las tortugas es 42.62 % para 250 *ticks*. ¿Existe una tendencia lineal a subir la efectividad cuando permitimos a las tortugas buscar con más tiempo?

Si hacemos un tercer experimento con 500 *ticks*, obtenemos que el promedio de champiñones creados es 77.8, lo cual sigue siendo consistente con nuestros dos conjuntos de experimentos anteriores. Los champiñones cazados en promedio son 49.1. La eficiencia para el escenario de 500 *ticks* es de 63.11 %.

Aunque hemos realizado muy pocos experimentos, se puede elaborar un gráfico que nos muestre el comportamiento del modelo en cuanto a eficiencia (Figura 4.18).

Figura 4.18
Gráfica del comportamiento de la eficiencia vs. *TICKS*



Cerramos este capítulo dejando al lector la tarea de seguir realizando experimentos con el modelo, incitándolo a pensar qué condición sería la correcta para detener el programa una vez que las tortugas hayan acabado de cazar todos los champiñones creados en el inicio.

PARTE II.
APLICACIONES

CAPÍTULO V
SOCIOLOGÍA COMPUTACIONAL
EL MODELO DE SCHELLING

5.1 El modelo de Schelling

El objetivo de este capítulo es presentar uno de los modelos más icónicos de la sociología matemática. Nos referimos al modelo de segregación propuesto por Thomas Schelling (1969; 1971), así como presentar su versión en Netlogo como ejemplo de un modelo sencillo de agentes.

Las ideas de Schelling se basan en una observación profunda acerca del comportamiento colectivo humano. Schelling observó que el comportamiento organizado de las personas no se plantea solamente en la forma de acciones individuales tendientes a minimizar esfuerzos o maximizar el confort. Los constreñimientos hechos por la acción de otros individuos que persiguen sus propias metas y objetivos son esenciales en este fenómeno. Schelling llamó a esto un comportamiento contingente, esto es, un comportamiento que depende de lo que los otros han hecho. Otra observación importante es que la coordinación entre las decisiones humanas aparece sin un conocimiento profundo de las decisiones de los otros. Usando monedas de un centavo (*pennies*) y monedas de diez centavos (*dimes*) sobre un tablero de ajedrez (sesenta y cuatro cuadros divididos en ocho renglones y ocho columnas), y moviendo las monedas de acuerdo con reglas específicas, Schelling explicó cómo aparecen espontáneamente patrones de segregación residencial, tales como *guetos*, aun si la población no es segregacionista. Schelling interpretó el tablero de ajedrez como una ciudad, con cada celda representando una casa o un lote vacío. En este juego de simulación, las monedas son los agentes que representa cada una a uno de los dos grupos en que divide su sociedad artificial.

La ausencia de una noción global de segregación en las preferencias del agente es una característica de gran relevancia del modelo. Los agentes tienen sólo preferencias locales sobre sus vecindades. La vecindad de un agente que ocupa cualquier sitio en el tablero de ajedrez consiste en los cuadros adyacentes a la localización. Las reglas de evolución del modelo dependen de lo que Schelling denomina la "felicidad" de los agentes. Dicha felicidad determina el cambio de ubicación del agente. La felicidad de un agente depende del número de agentes del mismo tipo localizados en su vecindad. Si un agente es "infeliz", éste tratará de moverse a otro lugar en el tablero de ajedrez o es posible que abandone el tablero para siempre. Schelling encontró diferentes configuraciones y reglas que permiten encontrar equilibrios y ciclos en su modelo.

El modelo de Schelling puede ser formalizado como sigue. En un espacio celular de N×N sitios, para cada celda puede asociársele uno de los siguientes estados tomados del conjunto $S=\{w,b,e\}$, donde w representa un agente blanco, b un agente negro y e un espacio vacío. Cada agente trata de escoger una ubicación siguiendo una regla que depende de la proporción de agentes de otros tipos (colores) en sus vecindades. Dado un umbral φ, el agente cambiará su ubicación si el porcentaje de individuos similares en la vecindad es menor que φ, esto es, para una configuración $\{P_i^{(t)}\}_{i\in\Lambda}$ del tablero de ajedrez en el tiempo t,

$$\text{Si } \left(P_i^{(t)} = b \;\; y \;\; \frac{n_i^{(t)}(b)}{n_i^{(t)}(b)+n_i^{(t)}(w)} < \varphi\right) \qquad (5.1)$$

O bien

$$\text{Si } \left(P_i^{(t)} = w \;\; y \;\; \frac{n_i^{(t)}(w)}{n_i^{(t)}(b)+n_i^{(t)}(w)} < \varphi\right) \qquad (5.2)$$

Entonces:

$$P_i^{(t+1)} = e \quad y \;\; P_j^{(t+1)} = P_i^{(t)} \qquad (5.3)$$

Si no, entonces:

$$P_i^{(t+1)} = P_i^{(t)}, \qquad (5.4)$$

donde $n_i^{(t)}(b)$ es el número de agentes negros en la vecindad de la localidad i en el tiempo t y $n_i^{(t)}(w)$ es el número de agentes blancos en la vecindad de la localidad i en el tiempo t. Por supuesto, la nueva localización j, la cual es escogida de forma aleatoria, debe estar vacía. Conforme el número de celdas vacías decrece, se incrementa el tiempo necesario para encontrar nuevas localizaciones.

5.2 El modelo Netlogo de Schelling

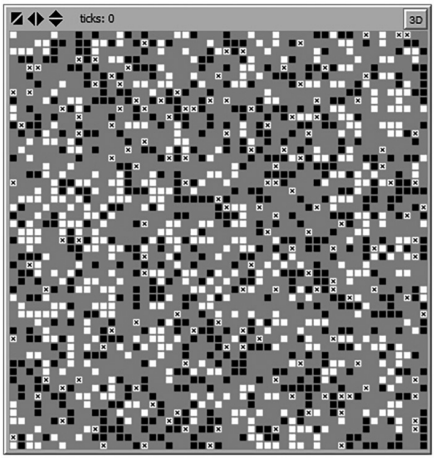
En el menú "Archivo/Biblioteca de modelos" de Netlogo está disponible una carpeta llamada "Social science". En dicha carpeta se encuentra un modelo llamado "Segregation", que está inspirado en el modelo de Schelling (1969). El modelo que usaremos fue programado por Uri Wilensky en 1999.

Se modela el comportamiento de los dos tipos de agentes en un espacio urbano. Los agentes blancos y los agentes negros se llevan bien entre sí. Pero cada agente quiere asegurarse de que vive cerca de algunos agentes parecidos a "sí mismo". Es decir, cada agente blanco quiere vivir cerca de al menos algunos agentes de su mismo color, y lo mismo pasa con los agentes negros. La simulación muestra cómo estas preferencias individuales se propagan para dar lugar a patrones a gran escala.

Originalmente, el modelo tiene las parcelas vacías de color negro, mientras que los agentes son de color rojo y verde. Para fines de claridad en el libro en blanco y negro, hemos realizado unas pequeñas modificaciones al código original. Las parcelas son pintadas de origen en gris, y los agentes son de color blanco y negro, según sea el caso (Figura 5.1). Para hacer que las parcelas del mundo sean de color gris, usamos el siguiente código al inicio del procedimiento "Setup":

```
ask patches [ set pcolor gray]
```

Figura 5.1
Pantalla de inicialización del programa "Segregation"



Los agentes pueden ser blancos o negros. Las partes grises son parcelas desocupadas.

5.2.1 La interfaz del modelo Netlogo de Schelling

En la pestaña "Ejecutar" se encuentra la interfaz del programa Segregation.nlogo desarrollado por Wilensky (Figura 5.2). Ésta incluye:

- Un "deslizador" para establecer la densidad ("density") de agentes en el mundo.
- Un "deslizador" para establecer el porcentaje de similaridad deseado, esto es, el número de agentes "de mi mismo color" que deben estar en mi vecindad para mantenerme "feliz" ("%-similar-wanted").
- Un botón que permite elegir entre dos tipos de visualizaciones del mundo. Usa la variable de tipo "string" llamada "visualization". Si se escoge la variable "old", las tortugas toman la forma de flecha por *default*. Si se escoge la variable "square-x", la representación refleja el estado de felicidad de la tortuga. Si la variable es verdadera (esto es, una tortuga feliz), la tortuga toma la forma de un cuadrado. Si es negativa (esto es, una tortuga infeliz), toma la forma de un cuadrado con una *x* en medio.

El funcionamiento del programa se controla con los siguientes botones:

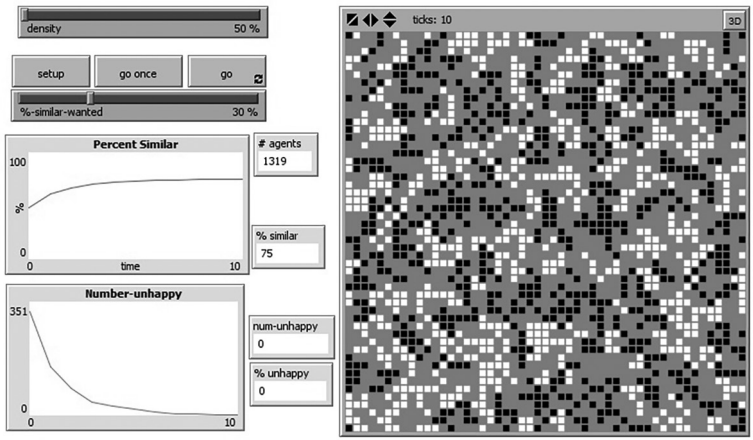
- El botón "Setup" inicializa el mundo con las condiciones iniciales establecidas por medio de los deslizadores estableciendo la ubicación espacial inicial de las tortugas blancas, las tortugas negras y los espacios vacíos en gris, así como el estado de felicidad inicial de las tortugas.
- El botón "Go once" permite avanzar la simulación con sólo un *tick*. Mientras que el botón "Go" realiza la simulación hasta terminarla.

Además, el modelo incluye las siguientes salidas:

- Una gráfica llamada "percent-similar" que monitoriza el porcentaje de similaridad conforme transcurre el tiempo (*ticks*).
- Una gráfica llamada "number-unhappy" que monitoriza el número de tortugas no felices conforme transcurre el tiempo (*ticks*).

Pulsando sobre de las gráficas con el botón derecho del ratón, se ingresa a la ventana que controla el gráfico donde se indica el rango de valores de los ejes X y Y, así como la instrucción para realizar las gráficas:

Figura 5.2
Estado final del modelo de Schelling



Densidad inicial del 50 % y porcentaje de similaridad requerido del 30 %.

```
plot percent-similar]
```

Asociado a dichas ventanas están:

- Un monitor del número de tortugas en el mundo.
- Un monitor del porcentaje de similaridad.
- Un monitor del número de tortugas infelices en el mundo.
- Un monitor del porcentaje de tortugas infelices.

Se puede observar, para las condiciones iniciales de densidad inicial del 50 % y porcentaje de similaridad requerido del 30 %, cómo el porcentaje de similaridad fue creciente hasta estabilizarse en 75 %. Existen 1 319 tortugas, de las cuales ninguna es infeliz y, por tanto, el porcentaje de infelicidad es cero. Nótese que se crearon cúmulos de agentes blancos y agentes negros con zonas grises vacías. Esto se logró en sólo 10 *ticks*. El sistema espacial ya es estable, por lo que ya no evoluciona más.

5.2.2 La definición de variables

En la pestaña "Código" se encuentra el código de programación. Al inicio del programa se inician las variables globales:

```
globals [
percent-similar      ;; Sobre el promedio, ¿qué porcentaje de agentes en la ;; vecindad
                        de la tortuga son del mismo color que ella?
percent-unhappy      ;; ¿Qué porcentaje de las tortugas son infelices?
]
```

Luego se definen las variables de las tortugas:

```
turtles-own [
happy?               ;; para cada tortuga, indica si al menos un porcentaje %-similar-
                        wanted;; de las tortugas en la vecindad son del mismo color que
                        la tortuga
```

```
similar-nearby      ;; cuántas parcelas en la vecindad tienen una tortuga con mi color
other-nearby        ;; cuántas tortugas en la vecindad tienen otro color
total-nearby        ;; Es la suma de las dos variables previas.
]
```

5.2.3 El código de programación para inicializar el modelo

Luego escribimos el código del procedimiento "Setup" (inicializar). Éste es:

```
to setup
clear-all
ask patches
[ set pcolor gray]                  ;; colorea las parcelas de color gris
ask patches                         ;; Crea tortugas sobre parcelas al azar
[if random 100 < density [          ;; establece la densidad de ocupación
sprout 1 [
set color one-of [white black]
]
]
]

update-turtles
update-globals
reset-ticks
end
```

Al principio del código aparece el comando Netlogo "clear-all", el cual borra las tortugas de ejecuciones anteriores y pone en *default* las parcelas para dejar un "mundo" libre donde poder construir. Este comando es muy útil para la realización de varios experimentos.

Los siguientes comandos, "ask patches [set pcolor gray]", los insertamos nosotros, como ya habíamos mencionado antes, con el fin de que el color de las parcelas fuera gris para su mejor visibilidad en una impresión en blanco y negro.

Luego sigue un conjunto de comandos que crean tortugas, las cuales se encuentran localizadas al azar en las parcelas.

```
ask patches [
        if random 100 < density [
        sprout 1 [
        set color one-of [white black]
]
```

Se comienza declarando que los comandos se realizarán sobre las parcelas, esto lo hace el comando "ask patches". Nótese que este comando actúa sobre todas las parcelas al mismo tiempo. Luego hay un proceso condicional en el que se genera un número aleatorio entre cero y cien si dicho número es menor que la densidad, la cual está dada por un botón deslizador; entonces se crea una tortuga en la parcela correspondiente y establece el color de ésta, ya sea blanca o negra. El comando "one-of" toma de manera aleatoria un objeto de una lista de objetos; en este caso, los objetos son comandos de color para los agentes. Nótese que hemos cambiado los colores originales del programa (verde y rojo) y los hemos sustituido por negro y blanco, esto con fines de que sea legible el mundo en el libro en blanco y negro.

Luego el programa llama a dos procedimientos y un comando. El comando "reset-ticks" pone en ceros todos los *ticks*, mientras que "update-turtles" y "update-globals" son procedimientos que cuentan el número de tortugas del mismo color en la vecindad y deciden qué tipo de forma ("shape") tendrán en el mundo.

El procedimiento "update-turtle" es como sigue:

```
to update-turtles
ask turtles [
        ;; en las siguientes dos líneas usaremos "neighbors" para probar las 8 parcelas
        ;; que rodean la parcela actual
        set similar-nearby count (turtles-on neighbors) with [ color = [ color ] of myself ]
        set other-nearby count (turtles-on neighbors) with [ color != [ color ] of myself ]
        set total-nearby similar-nearby + other-nearby
        set happy? similar-nearby >= (%-similar-wanted * total-nearby / 100)
        ;; Se añade el comando "visualization" aquí
        if visualization = "old" [ set shape "default" ]
        if visualization = "square-x" [
        ifelse happy? [ set shape "square" ] [ set shape "square-x" ]
        ]
]
end
```

Nótese que la variable "similar-nearby" toma el valor del número de tortugas en la vecindad con color similar al de la tortuga central. Así, si en la vecindad de tamaño ocho existen seis tortugas con el mismo color que la tortuga en el centro, la variable "similiar-nearby" tomará el valor de seis. La variable "other-nearby" toma el valor de las tortugas en la vecindad con color distinto al de la tortuga en la celda central. La variable "total-nearby" es igual a la suma de las variables "similar-nearby" y "other-nearby".

La variable "happy?" es de tipo *booleano* y es verdadera si "similar-nearby" es mayor o igual al porcentaje de similares requeridos, la cual es una variable que se inicializa desde el botón deslizador "%-similar-wanted" (Figura 5.2).

El procedimiento "update-globals" se describe a continuación.

```
to update-globals
      let similar-neighbors sum [ similar-nearby ] of turtles
      let total-neighbors sum [ total-nearby ] of turtles

      set percent-similar (similar-neighbors / total-neighbors) * 100
      set percent-unhappy (count turtles with [ not happy? ]) / (count turtles) * 100
end
```

El procedimiento "update-globals" inicializa las variables globales "percent-similar" y "percent-unhappy", creando dos variables locales "similar-neighbors", la cual es la suma de la tortugas similares entre sí y "total-neighbors", la cual es la suma del total de las tortugas en el mundo. La variable "percent-similar" es el porcentaje de tortugas similares en el mundo. Mientras que la variable "percent-unhappy" cuenta el número de tortugas con la variable *booleana* "happy?" negativa, y la divide entre el número total de tortugas en el mundo.

5.2.4 El código de programación para correr el modelo

A continuación, revisaremos el código que permite que se dé la dinámica anterior. El botón "Go once" invoca el procedimiento "go", el cual se muestra a continuación:

```
to go
        if all? turtles [ happy? ] [ stop ]
        move-unhappy-turtles
        update-turtles
        update-globals
        tick
end
```

El procedimiento "go" comienza preguntando si todas las tortugas están felices, si es así, el programa se detiene. Esto fue lo que ocurrió en el *tick 10* de la corrida anterior. Por el contrario, si existen tortugas no felices, el programa invoca el procedimiento "move-unhappy-turtles" y después actualiza las variables de las tortugas y las globales, y avanza un *tick*.

El procedimiento "move-unhappy-turtles" se muestra a continuación, junto con el procedimiento asociado "find-new-spot":

```
to move-unhappy-turtles
        ask turtles with [ not happy? ]
        [ find-new-spot ]
end
```

```
to find-new-spot
        ;; mueve a la tortuga hasta encontrar un lugar vacío.
        rt random-float 360
        fd random-float 10
        if any? other turtles-here [ find-new-spot ]    ;; Mantiene el procedimiento
        funcionando
        ;; hasta encontrar una parcela vacía
        move-to patch-here    ;; se mueve al centro de la parcela
end
```

El procedimiento "move-unhappy-turtles", que mueve a las tortugas infelices, les pregunta a todas las tortugas con la variable *booleana* "happy?" como falsa e invoca el procedimiento "find-new-spot" (encontrar un nuevo lugar). Este procedimiento rota la cabeza de la tortuga infeliz aleatoriamente entre cero y 360 grados. Luego avanza un número aleatorio

entre cero y 10. En dicho lugar pregunta si está previamente ocupado por una tortuga; si es así, vuelve a moverse. Si el lugar está desocupado, la tortuga se mueve a dicho lugar.

5.3 Descripción del modelo de Schelling según el protocolo ODD

Terminamos este capítulo con una descripción del modelo siguiendo el protocolo ODD. Dado que el modelo ya se ha explicado a detalle a lo largo del capítulo, la descripción se hace de forma breve.

5.3.1 Propósito

El objetivo de este modelo es explorar y analizar cómo estas preferencias individuales se propagan para dar lugar a patrones a gran escala de segregación.

5.3.2 Entidades, variables de estado y escala

Como variables globales del modelo, se definen las siguientes:

- "Density" representa la cantidad de tortugas en el mundo. Varía entre 50 y 100. Se inicia en la interfaz.
- "%similar-wanted" representa el porcentaje de las tortugas del mismo color que se requiere para que una tortuga sea feliz. Varía entre 0 y 100. Se inicializa en la interfaz.
- "Percent-similar" representa el porcentaje de agentes en la vecindad de la tortuga que son del mismo color que ella. Varía entre 0 y 100.
- "Percent-unhappy" representa el porcentaje de las tortugas infelices. Varía entre 0 y 100.
- Cada agente tiene las siguientes variables de estado:
- "Happy?" indica si al menos un porcentaje "%-similar-wanted" de las tortugas en la vecindad son del mismo color que la tortuga. Toma el valor "true" (verdadero) o "false" (falso).

- "Similar-nearby" indica cuántas parcelas en la vecindad tienen una tortuga del mismo color.
- "Other-nearby" indica cuántas tortugas en la vecindad tienen otro color.
- "Total-nearby" es la suma de las dos variables previas.

El entorno está formado por un *grid* de 50 de ancho por 50 de alto. La simulación termina cuando todas las tortugas están "happy".

5.3.3 Visión general y orden de los procesos

El tiempo transcurre de forma discreta, por pasos o etapas, y la actualización de las variables se produce de forma asíncrona.

En cada uno de estos pasos, los siguientes eventos tienen lugar en el siguiente orden. Cada tortuga, en un orden aleatorio, ejecuta la siguiente secuencia de acciones:

1. "Update-turtles": Evaluar su estado "happy?" en función del número de vecinos de su mismo color.
2. "Move-unhappy-turtles": Si "happy?" es "false", se mueve si hay un sitio vacío.
3. "Update-globals": Se evalúan los resultados agregados

5.3.4 Conceptos de diseño

- Principios básicos: Puede aplicarse en modelos de dinámica urbana.
- Emergencia: Emergen patrones a gran escala como consecuencia de las preferencias de similaridad de los individuos.
- Adaptación: Si los agentes no están satisfechos porque el porcentaje de vecinos de su mismo color ("similar-nearby" dividido entre "total-nearby") es inferior al deseado ("%similar-wanted"), las tortugas buscan nuevos lugares en los que situarse.
- Objetivos: El objetivo de las tortugas es rodearse de un porcentaje determinado de vecinos de su mismo color ("%similar-wanted").

- Aprendizaje: No hay ningún tipo de aprendizaje en el modelo.
- Predicción: No hay predicción en el modelo.
- Detección: Internamente las tortugas toman la decisión de moverse si no están felices. Externamente, perciben la felicidad en función del color de sus vecinos.
- Interacción: La única manera en que interactúan los agentes es, indirectamente, mediante su posición en el *grid*.
- Aleatoriedad: En la inicialización, como se verá más adelante.
- Colectivos: No hay organización de los agentes de más alto nivel que las tortugas.
- Observación: Se recogen y se muestran datos del número de tortugas infelices y del porcentaje de similaridad.

5.3.5 Inicialización

La posición de las tortugas en el *grid* se hace de forma aleatoria y el porcentaje de las tortugas del mismo color que se requiere para que una tortuga sea feliz se inicializa mediante la interfaz.

5.3.6 Datos de entrada

No hay datos de entrada externos al modelo.

5.3.7 Submodelos

La programación de estos procedimientos (o en su defecto, el pseudocódigo) no se ha incluido ya que ésta se detalla más adelante:

"Update-turtles": Evalúa el estado "happy?" de las tortugas en función del número de vecinos de su mismo color.
"Move-unhappy-turtles": Mueve la tortuga si es infeliz y si hay un sitio vacío.
"Update-globals": Se evalúan los resultados agregados

5.4 Experimentos con el modelo: el analizador de comportamiento

Una vez descrito y programado el modelo, es el momento de hacer un análisis de los resultados que se obtienen. Hemos visto en la sección anterior que para unas condiciones iniciales de densidad inicial del 50 % y un porcentaje de similaridad requerido del 30 % se ha obtenido que en 10 instantes (*ticks*) correspondientes a 10 ejecuciones del procedimiento "go" se alcanza un estado estacionario estable. ¿Qué ocurre si modificamos la densidad inicial de agentes? ¿Convergerá a un estado estable? ¿En cuántos *ticks* lo hará? ¿Qué ocurre si cambiamos el porcentaje de similarirad requerida? Éste es el tipo de preguntas que podemos plantearnos y con las cuales experimentar con el modelo.

En la Tabla 5.1 se muestra una serie de experimentos que proponemos realizar con el modelo con el fin de puntualizar algunos aspectos relevantes de la dinámica del modelo y motivar a que el lector replique el modelo por sí mismo. Dichos experimentos permitirán familiarizar al lector con las herramientas que Netlogo dispone para este fin.

Tabla 5.1
Posibles experimentos con el modelo de Schelling

Experimento	1	2	3	4	5	6
Density	50 %	70 %	70 %	90 %	50 %	50 %
%-similar-wanted	50 %	30 %	50 %	30 %	70 %	90 %
Repeticiones	varias	varias	50	50	50	50

5.4.1 Experimento 1

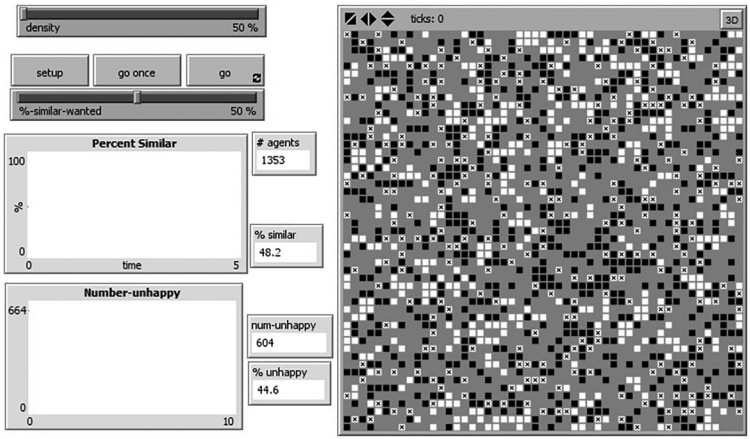
Inicializamos las variables "density" como 50 % y "%-similar-wanted" como 50 %. Al hacer clic en "Setup", se creará un mundo artificial nuevo (Figura 5.3). Se crearon 1 353 tortugas, con un porcentaje de similaridad del 48.2 %, existiendo en un principio 604 tortugas en estado de infelicidad, lo que se traduce como un porcentaje de infelicidad del 44.6 %.

Después de pulsar 18 veces el botón "Go once" (18 *ticks* del programa) obtenemos los resultados que aparecen en la Figura 5.4.

- El modelo convergió a un estado estable y presenta patrones de cúmulos ordenados tanto de tortugas blancas como de tortugas negras y espacios vacíos.
- El porcentaje de similaridad es de 88.5 % y no existen tortugas infelices; por ende, el porcentaje de infelicidad es de cero.

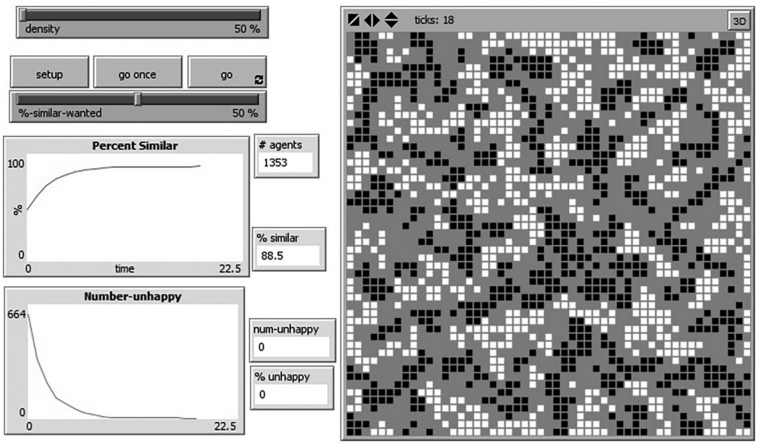
Nótese que se pueden realizar nuevas ejecuciones del programa con los mismos valores iniciales de las variables "density" y "%-similar-wanted", y el programa variará en el número de tortugas que se crean, así como en el número inicial de tortugas infelices y los porcentajes de similaridad y de infelicidad. Los patrones espaciales logrados después de correr el modelo serán semejantes a los de la Figura 5.4. Los "tiempos de convergencia" (el número de *ticks* necesarios para que el modelo llegue a un estado estable) varían entre 18 y 13. El porcentaje de similaridad siempre está en un rango de entre 88.3 y 90.3.

Figura 5.3
Experimento 1



(Densidad= 50 %, similaridad deseada = 50 %): estado inicial.

Figura 5.4
Experimento 1



(Densidad= 50 %, similaridad deseada = 50 %): estado final después de 18 instantes.

5.4.2 Experimento 2

Realizamos otro experimento donde las condiciones iniciales tienen una densidad igual al 70 % y porcentaje de similaridad deseada del 30 % (Figura 5.5). Se han creado 1 816 tortugas; el porcentaje de similaridad es del 50.2 % y existen 355 tortugas infelices.

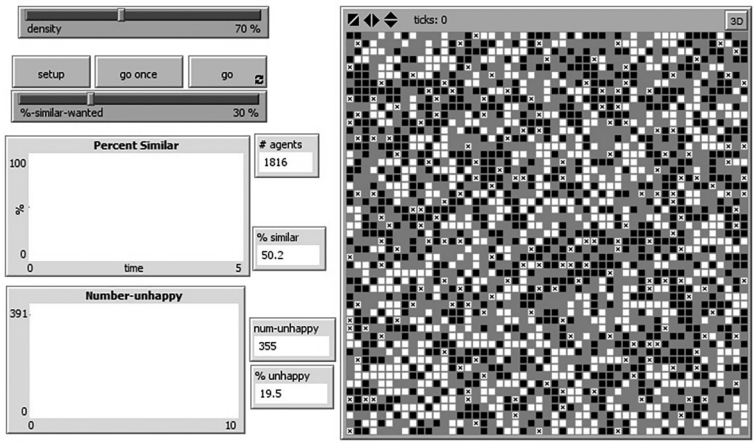
Al experimentar con este nuevo escenario, observamos que sólo se requieren 9 *ticks* para llegar al estado estable (es decir, no hay tortugas infelices). El porcentaje de similaridad es de 72.9 % (Figura 5.6).

Realizamos 50 experimentos con las condiciones iniciales de densidad igual al 70 % y porcentaje de similaridad deseada del 30 %. Dado que pulsar 50 veces el botón "Go" puede resultar tedioso, recomendamos el uso del "analizador de comportamiento" (Figura 5.7).

Al realizar más repeticiones con las mismas condiciones iniciales en las variables "density" y "%-similar-wanted" observamos que el sistema presenta una tendencia a "estabilizarse pronto". Encontramos que es necesario un mínimo de 7 *ticks* para estabilizar el entorno y un máximo de 18 *ticks*, con una media de 11.8 y una desviación estándar de 3.0928.

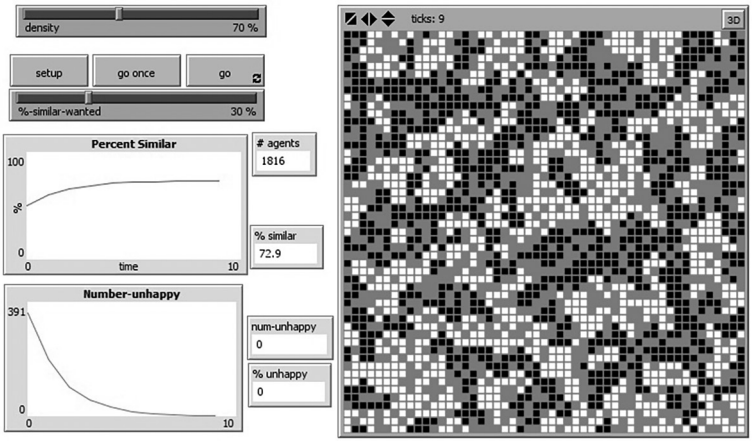
Para el porcentaje de similaridad, encontramos un mínimo de 70.2 % y un máximo de 74.3 % con una media de 72.52 y una desviación estándar del 1.3083. Los patrones espaciales encontrados presentan claros cúmulos de agentes blancos y negros, salpicados de zonas con parcelas vacías.

Figura 5.5
Experimento 2



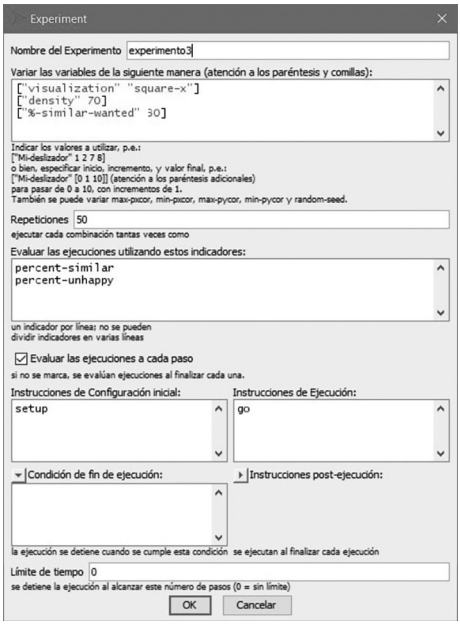
(Densidad= 70 %, similaridad deseada = 30 %): estado inicial.

Figura 5.6
Experimento 2



(Densidad= 70 %, similaridad deseada = 30 %): estado final después de 9 instantes.

Figura 5.7
Experimento 2

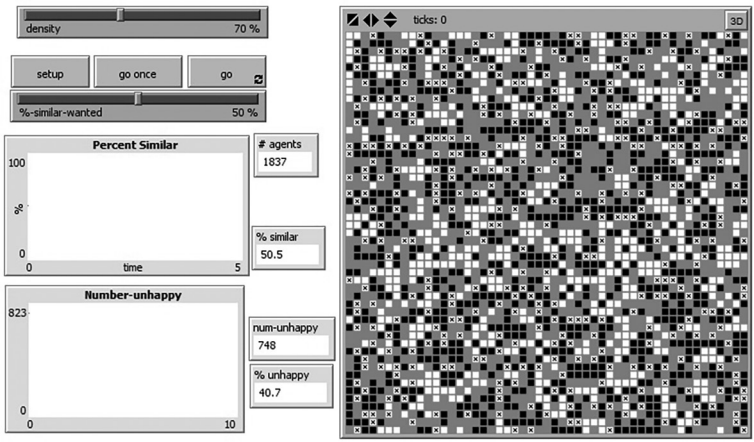


Ventana del analizador de comportamiento.

5.4.3 Experimento 3

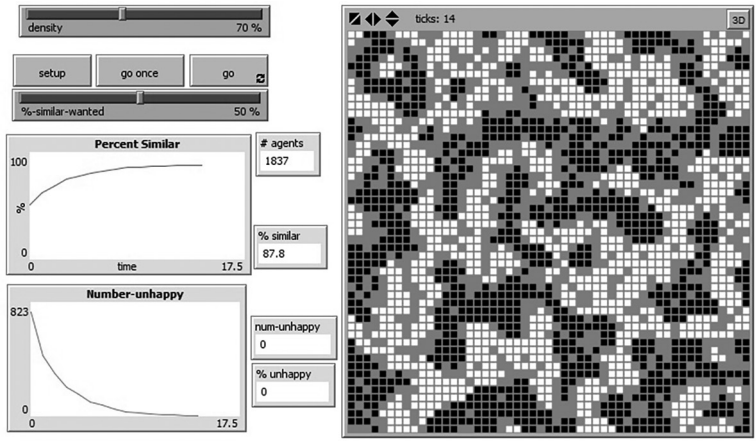
Recurrimos al analizador de comportamiento para realizar 50 repeticiones del experimento con unas condiciones iniciales de densidad igual al 70 % y porcentaje de similaridad deseada del 50 % (Figura 5.8). Encontramos un número mínimo de tortugas de 1 770 y un máximo de 1 862, con una media de 1 815.73 y una desviación estándar del 25.75. El porcentaje de similaridad medio alcanzado fue de 87.68 con una desviación estándar del 0.4161 y el número medio de *ticks* para estabilizar el entorno fue de 18.26 con una desviación estándar del 5.14 (Figura 5.9).

Figura 5.8
Experimento 3



(Densidad=70 %, similaridad deseada = 50 %): estado inicial.

Figura 5.9
Experimento 3



(Densidad=70 %, similaridad deseada = 50 %): estado final después de 14 instantes.

5.4.4 Experimento 4

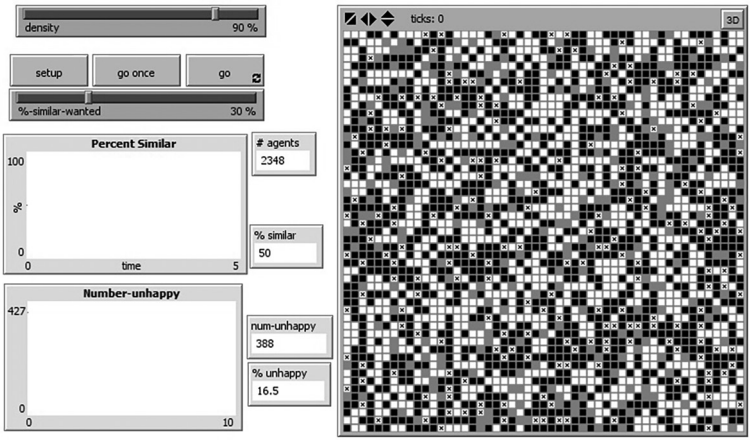
Recurrimos al analizador de comportamiento para realizar 50 repeticiones con las condiciones iniciales de densidad igual al 90 % y porcentaje de similaridad deseada del 30 % (Figura 5.10). Para estas condiciones, encontramos que se crean en promedio 2 337.2 tortugas con una desviación estándar de 15.3977. El porcentaje de similaridad alcanzado en el estado estable es de 74.04 % en promedio con una desviación estándar de 0.8991 y el número de *ticks* (instantes) necesarios para alcanzar el estado estable es de 14.53 en promedio, con una desviación estándar del 2.5317. Espacialmente, se pueden observar aglomeraciones de tortugas blancas y negras divididas por pequeñas porciones de espacio vacío (Figura 5.11).

5.4.5 Experimento 5

Recurrimos al analizador de comportamiento para realizar 50 repeticiones con las condiciones iniciales de densidad del 50 % y porcentaje de similaridad deseada del 70 % para ver qué ocurre cuando las tortugas son más segregativas (Figura 5.12). Esto se traduce en que el 70 % de los vecinos de cada tortuga sean de su mismo color para que la tortuga sea feliz.

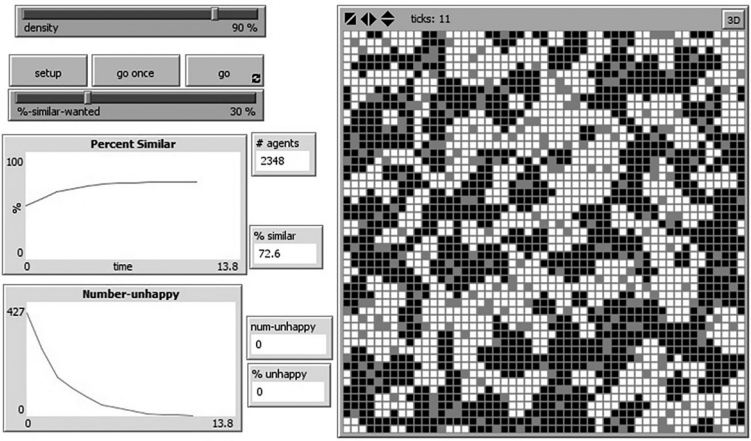
Observamos que el sistema tarda más tiempo en llegar al estado estable. En promedio tarda 51.46 *ticks*, con una desviación estándar del 8.82. La densidad del 50 % provoca que se creen en promedio 1 304.13 tortugas con una desviación estándar del 30.37. La similaridad alcanzada por el sistema es muy alta, siendo en promedio del 99.72 %, con una desviación estándar del 0.1207. Espacialmente, el comportamiento del modelo es muy característico, con una clara diferenciación entre las tortugas blancas y las tortugas negras. Los espacios vacíos sirven de barrera entre las dos especies de tortugas. Podemos decir que se han formado "guetos" (Figura 5.13).

Figura 5.10
Experimento 4



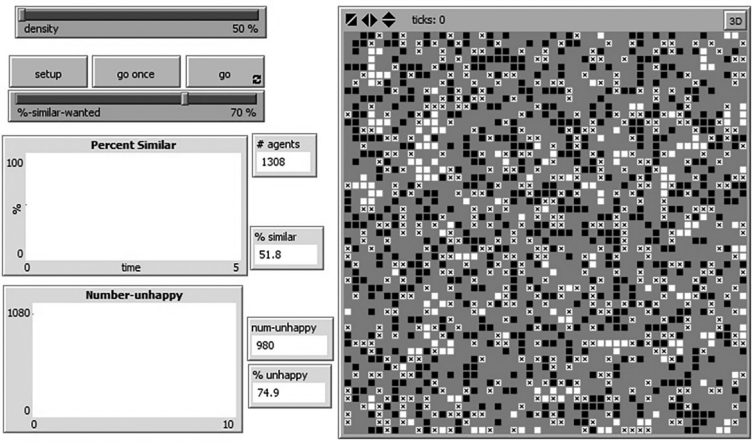
(Densidad=90 %, similaridad deseada = 30 %): estado inicial.

Figura 5.11
Experimento 4



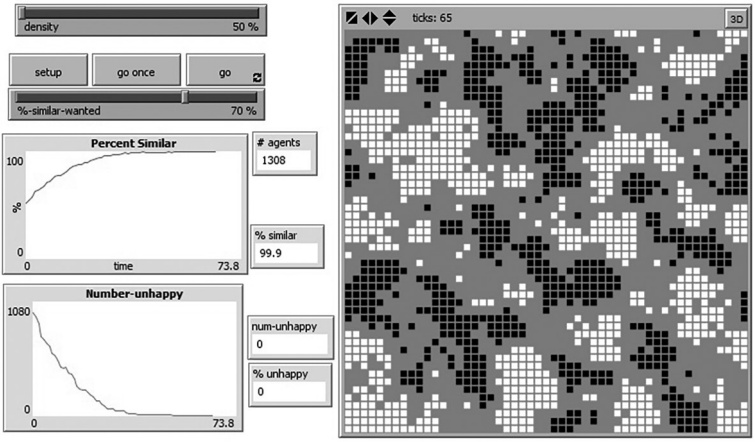
(Densidad=90 %, similaridad deseada = 30 %): estado final después de 11 instantes.

Figura 5.12
Experimento 5



(Densidad=50 %, similaridad deseada = 70 %): estado inicial.

Figura 5.13
Experimento 5



(Densidad=50 %, similaridad deseada = 70 %): estado final después de 65 instantes. Nótese la formación de "guetos" o vecindades totalmente aisladas.

5.4.6 Experimento 6

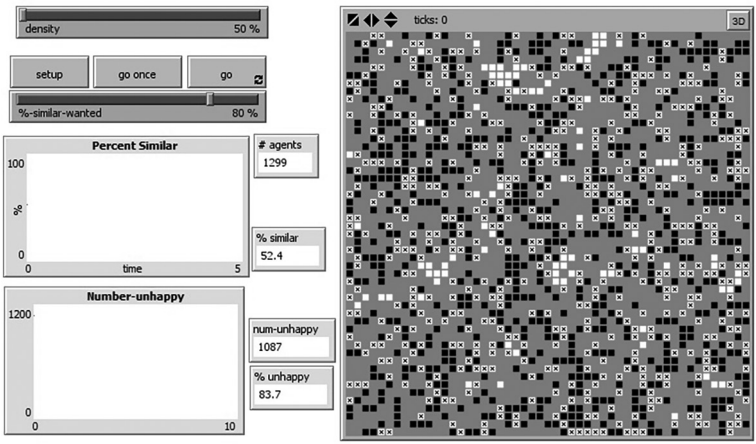
Recurrimos al analizador de comportamiento para realizar 50 repeticiones con las condiciones iniciales de densidad del 50 % y porcentaje de similaridad deseada del 80 % para ver qué ocurre cuando elevamos aún más la tendencia a la segregación por parte de las tortugas (Figura 5.14).

Obtuvimos que el número promedio de tortugas iniciales es de 1 309.73, con una desviación estándar del 34.82. Al evolucionar el modelo, éste se estabiliza con un valor promedio del porcentaje de similaridad del 99.99 %, con una desviación estándar del 0.0258. El tiempo que tarda en llegar al estado estable es en promedio de 468.06 *ticks*, con una desviación estándar del 131.23. Espacialmente, el estado estable se caracteriza por presentar unos cuantos cúmulos grandes y tortugas aisladas rodeadas de espacio vacío. Nótese que, como en los demás ejemplos, no hay tortugas infelices (Figura 5.15).

Un hallazgo es que, al aumentar el porcentaje de similaridad deseada ("%-similar-wanted") a 81 %, el sistema no converge a un estado estable. El sistema mantiene un porcentaje de similaridad de alrededor del 50 %, con un porcentaje de infelicidad del 85 %. Esto indica la existencia de una transición de fase gobernada, en parte, por el porcentaje de similaridad deseada. Cuando el porcentaje de similaridad deseada es mayor a 81 %, el modelo no logra llegar a un estado estable. Los agentes cambian de lugar incesantemente tratando de encontrar vecindades que les permitan ser felices. Sin embargo, esto no se logra debido a que es muy alto el nivel de satisfacción que tiene que cumplir la vecindad para que un agente se sienta feliz. El modelo nunca encuentra un estado estable. Nótese que el experimento tiene una densidad igual al 50 %. Es decir, existe mucho espacio vacío hacia donde se pueden mover las tortugas buscando vecindades que las hagan felices.

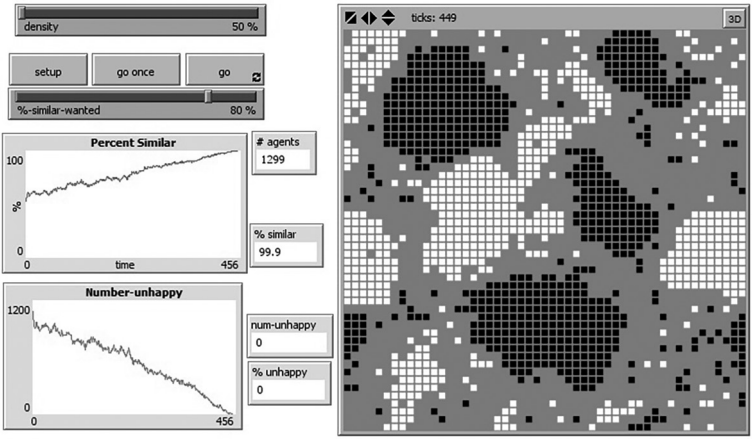
Por tanto, otro factor que incide en el comportamiento del modelo es el número de espacios vacíos que existen. A menor número de espacios vacíos (es decir, mayor densidad), menor tolerancia hacia porcentajes altos de similaridad deseada.

Figura 5.14
Experimento 6



(Densidad=50 %, similaridad deseada = 80 %): estado inicial.

Figura 5.15
Experimento 6



(Densidad=50 %, similaridad deseada = 80 %): estado después de 449 instantes. Nótese la formación de grandes cúmulos.

5.5 Conclusiones

El de Schelling es un modelo de agentes sencillo que presenta comportamientos emergentes a nivel macro de sumo interés. Los agentes tienen unas reglas de comportamiento sencillas, en las cuales no existe en sí una regla segregativa. A pesar de que podemos considerar el porcentaje de similaridad deseada como la variable de segregación, observamos que se presentan patrones segregativos para valores bajos de ésta. Asimismo observamos que el sistema tiende a un estado estable, es decir, un estado en donde todas las tortugas son felices.

Otra observación interesante es que, pasando ciertos umbrales con respecto a la densidad y al porcentaje de similaridad deseada, el modelo se vuelve inestable y nunca alcanza un estado estable.

En nuestros experimentos encontramos que para una densidad ("density") del 50 % y un porcentaje de similaridad deseada ("%-similar-wanted") del 81 %, el modelo se vuelve inestable. Otro par de variables que encontramos que conducen a comportamientos inestables son la densidad del 70 % y el porcentaje de similaridad deseada del 76 %.

No es nuestro interés hacer un estudio exhaustivo de la transición de fase. Pero sí es necesario hacer notar que el modelo, en su sencillez, permite estudiar el fenómeno de la segregación. Ésta tiene lugar en condiciones en donde cognitivamente la tortuga no tiene un pensamiento segregativo, es decir, para valores de "%-similar-wanted" (porcentaje de similaridad deseada) bajos. Es hasta que la tortuga cognitivamente se vuelve "muy segregativa", o sea, se piden porcentajes de similaridad deseada muy altos, cuando el modelo nos deja ver un movimiento perpetuo de las tortugas, con muy alto porcentaje de infelicidad. Esto es interesante ya que se esperaría que valores muy altos de la variable "%-similar-wanted" llevaran a una segregación máxima. Por el contrario, dichos valores hacen que el modelo se vuelva inestable y exista un movimiento eterno de posiciones dentro de la "ciudad", buscando satisfacer una condición de felicidad muy difícil de alcanzar de manera colectiva.

Por último, queremos subrayar que el objetivo del capítulo era presentar el modelo de Schelling y su versión en Netlogo como ejemplo

de un modelo sencillo de agentes. Hemos buscado explicarlo lo más a detalle posible, y hemos realizado algunos experimentos con el afán de motivar al lector a que experimente con dicho modelo, lo modifique y proponga cambios que sean de su interés particular.

CAPÍTULO VI

ECONOMÍA COMPUTACIONAL BASADA EN AGENTES

UN MODELO DE INTERCAMBIO EN UN MERCADO

6.1 Introducción al funcionamiento de los mercados

El objetivo de este capítulo es profundizar en la programación en Netlogo. En concreto, nos centramos en la definición de clases de tortugas, el manejo de listas y las salidas gráficas. El modelo que se presenta en este capítulo es una versión simplificada del modelo de intercambio en un mercado de Posada y López (2008).

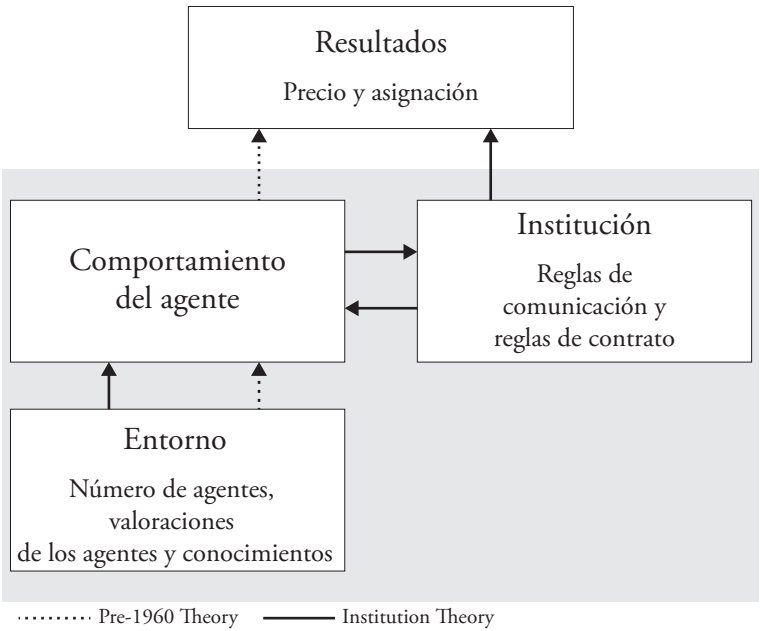
Pero antes de entrar en detalle sobre la programación en Netlogo, haremos una introducción a los conceptos clave de intercambio en los mercados.

El premio nobel de economía de 2002, Vernon Smith, establece que en el análisis de los mercados hay que tener en cuenta tres elementos básicos (Figura 6.1): el entorno (esto es, las condiciones de la oferta y la demanda, que incluyen las cantidades que se pueden negociar, los precios de reserva y los costes que motivan el intercambio), la institución (esto es, el lenguaje de comunicación entre los agentes, las reglas que gobiernan el intercambio de información y la forma en la que se cierra el contrato) y el comportamiento de los agentes que participan en el mercado (Smith, 1982).

En función de las condiciones de la oferta y la demanda de un mercado, la teoría económica determina cuáles serán el precio y la cantidad de equilibrio competitivo en el que todos los agentes maximizan su utilidad (Figura 6.2). Sin embargo, la teoría económica no captura cómo se alcanza el precio de equilibrio o el proceso de decisión de los agentes. Para llenar este vacío, se desarrolla la teoría de subastas y mercados.

Figura 6.1

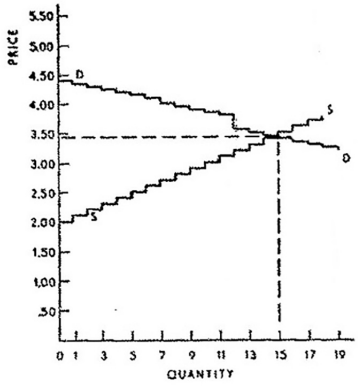
Elementos de un mercado



Fuente: Smith (1982).

Figura 6.2

Oferta y demanda del mercado



Fuente: Smith (1962).

Una de las instituciones más analizadas ha sido la subasta doble continua (en inglés, *continuous double auction*, CDA) debido a la complejidad de su dinámica y a que es una de las instituciones dominantes en el mundo real: mercado de la energía, derivados, activos negociables, etc.

El término *subasta doble* se utiliza en mercados donde tanto compradores como vendedores hacen ofertas de compra y de venta. No obstante, la subasta doble, así definida, engloba varias instituciones. Las reglas de intercambio de dichas instituciones difieren en cuándo se pueden hacer los anuncios de compra/venta (cada agente en el momento que lo desee o todos a la vez en un determinado momento); quién escucha los anuncios de precios (todos los agentes o sólo algunos de ellos); cuándo se pueden realizar las transacciones (en cualquier momento o todas a la vez al final del periodo) y en la presencia activa o pasiva de un subastador que hace que las transacciones sean todas al mismo precio o a precios diferentes. Estas diferencias en las reglas de intercambio afectan de forma importante al comportamiento estratégico de los agentes del mercado.

En particular, en la CDA los compradores anuncian sus ofertas de compra (en inglés, *bid*) y los vendedores anuncian sus ofertas de venta (*offer*) en cualquier momento. Todos los agentes escuchan simultáneamente los anuncios. Las ofertas se eliminan tan pronto como existe una oferta más favorable o tiene lugar un intercambio. La transacción o intercambio ocurre cuando un comprador acepta una oferta realizada por algún vendedor o un vendedor acepta una licitación realizada por algún comprador.

Ante la imposibilidad de analizar el comportamiento de los agentes en la CDA desde un enfoque teórico debido a la complejidad de su dinámica, la CDA se ha analizado fundamentalmente desde un enfoque experimental y un enfoque computacional.

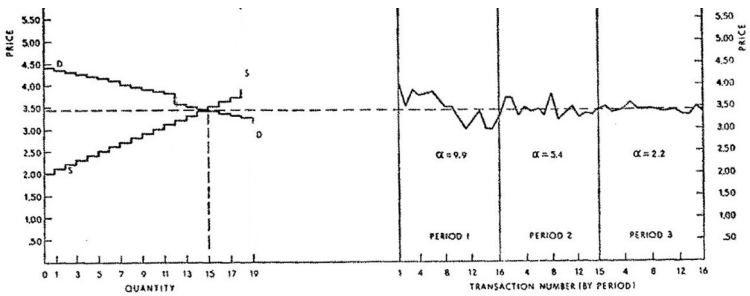
Mediante la economía experimental, se observa el comportamiento que emerge cuando los humanos participan en un mercado con unas condiciones del entorno dadas y unas reglas de intercambio específicas. Teniendo en cuenta que los agentes que participan en mercados estructurados bajo una CDA conocen solamente la valoración de sus unidades y no tienen información suficiente para determinar la oferta y la demanda de mercado, cada agente trata de hacer supuestos acerca de las pujas que realizarán los demás agentes (fijándose en la historia de las

pujas y en los precios a los que finalmente se han realizado las transacciones). Por tanto, los participantes en estos mercados se enfrentan a las siguientes decisiones: ¿cuánto pujar?, ¿cuándo pujar? y ¿cuándo aceptar una puja?

Smith (1962) observa en un mercado CDA una alta convergencia al precio de equilibrio competitivo teórico, que aumenta al incrementar los periodos de simulación (Figura 6.3), y una eficiencia cercana al 100 %.

Figura 6.3

Convergencia de precios



Fuente: Smith (1962).

Smith (1962) propone las siguientes medidas para cuantificar los resultados:

- La velocidad de convergencia al precio competitivo (α) mide la desviación típica del precio (*pj*) de cada una de las *k* transacciones realizadas en un periodo respecto al precio de equilibrio teórico (*po*) expresado en porcentaje sobre el precio de equilibrio teórico (*po*).

$$\alpha = 100 \frac{\sqrt{\dfrac{\sum_{j=1}^{k} (p_j - p_o)^2}{(k-1)}}}{p_o}$$

- La eficiencia (E) (en inglés, *allocative eficiency*) se define como los beneficios reales ganados por todos los agentes, expresados en porcentaje sobre el máximo de posibles beneficios que se podrían

ganar. Los beneficios reales ganados se calculan como el valor absoluto de la diferencia entre el precio de reserva del comprador-*i* o del vendedor-*i* (*PRi*) y el precio (*pj*) al que realizó la transacción-*j*. Los máximos beneficios posibles se calculan como el valor absoluto de la diferencia entre el precio de reserva del comprador-*i* / vendedor-*i* (*PRi*) y el precio de equilibrio competitivo teórico del mercado (*po*).

$$E = 100 \frac{\sum_{i=1}^{n+m} \sum_{j=1}^{k} \left| PR_i - p_j \right|}{\sum_{i=1}^{n+m} \left| PR_i - p_o \right|}$$

Donde *n* es el número de compradores, *m* es el número de vendedores y *k* es el número de transacciones que se han realizado en el mercado.

Intuitivamente, parece obvio que la inteligencia es necesaria en los agentes que participan en una subasta doble continua. Es, en esta cuestión, donde el enfoque computacional cobra relevancia ya que éste permite definir el comportamiento de los individuos que participan en el mercado.

Gode y Sunder (1993), definiendo unos agentes que no buscan beneficios ni mucho menos maximizan, sino que hacen sus ofertas de compra o de venta de forma aleatoria, demuestran que se puede conseguir una eficiencia en el mercado (*allocative efficiency*) de casi el 100 % con las reglas de mercado adecuadas.

Sin embargo, para que exista convergencia al precio de equilibrio competitivo establecido en la teoría económica, Cliff y Bruten (1997) demuestran que es necesario un mínimo de inteligencia. Cliff y Bruten (1997) proponen un agente artificial que hace sus ofertas de compra o de venta en función de un margen de beneficios que varía durante el proceso de subasta, mediante un aprendizaje adaptativo, basándose en la actividad previa de los otros agentes del mercado. Posada y López (2010) analizan la convergencia y la eficiencia de los mercados con diferentes tipos de agentes.

6.2 Descripción del modelo de mercado mediante el protocolo ODD

6.2.1 Propósito

Analizar la eficiencia del mercado y la convergencia al precio de equilibrio competitivo cuando interactúan agentes cero inteligentes en un mercado organizado mediante una subasta doble continua.

6.2.2 Entidades, variables de estado y escalas

El modelo incluye dos tipos de agentes: vendedores y comparadores.

Las variables globales definidas en la interfaz del modelo y relacionadas con el entorno son:

- "Numero-compradores" y "Numero-vendedores": representan el número de compradores y el número de vendedores, respectivamente. Su valor se asigna desde la interfaz.
- "Oferta-min" y "oferta-max" (en unidades monetarias): representan el rango de valores entre los que se van a generar de forma aleatoria los costes de los vendedores. Su valor se asigna desde la interfaz.
- "Demanda-min" y "oferta-max" (en unidades monetarias): representan el rango de valores entre los que se van a generar de forma aleatoria los precios de reserva de los compradores. Su valor se asigna desde la interfaz.
- "Periodos" (en unidades discretas de tiempo): representan el tiempo que está abierto el mercado.

Las variables globales definidas en la programación del modelo y relacionadas con el entorno son:

- "Oferta": es una lista, ordenada de menor a mayor, de los costes de los vendedores.

- "Demanda": es una lista, ordenada de mayor a menor, de los precios de reserva de los compradores.
- "Cantidad-equilibrio-teorico" es el resultado de la intersección entre la oferta y la demanda.
- "Precio-min": es la intersección entre la oferta y la cantidad de equilibrio teórica; "precio-max": es la intersección entre la demanda y la cantidad de equilibrio teórica. Cuando el "precio-min" no coincide con el "precio-max", el "precio-equilibrio-teorico" (en unidades monetarias) es el valor medio de ambos.
- "Excedente total" (en unidades monetarias): es la suma de las diferencias entre el precio de equilibrio y el precio de reserva de los compradores intramarginales (esto es, aquellos cuyos precios de reserva son mayores que el precio de equilibrio) y de las diferencias entre el precio de equilibrio y el coste de los vendedores intramarginales (esto es, aquellos cuyos costes son menores que el precio de equilibrio).

Las variables globales definidas en la programación del modelo y relacionadas con la institución son:

- "Best-bid" (en unidades monetarias): es la mejor oferta de compra en el mercado (esto es, la oferta de compra más alta). La "best-bid" se inicializa con el valor mínimo de la demanda.
- "Best-offer" (en unidades monetarias): es la mejor oferta de venta en el mercado (esto es, la oferta de venta más baja). La "best-offer" se inicializa con el valor máximo de la oferta.
- "Traders-list": es una lista que incluye los agentes que tendrían beneficios al participar en el mercado. En el caso de los compradores, aquellos cuyos precios de reserva sean mayores que la "best-bid"; y en el caso de los vendedores, aquellos cuyos costes sean menores que la "best-offer". Inicialmente, todos los compradores y vendedores pueden participar.
- "Precio" (en unidades monetarias): es el precio al que se realiza un intercambio. Dado que el precio es público, se ha definido como una variable global. Cuando un comprador acepta una oferta de venta subida, la transacción se realiza al precio de la oferta de venta.

Cuando un vendedor acepta una oferta de compra subida, la transacción se realiza al precio de la oferta de compra.

- "Precio-list": es una lista que incluye los precios de las transacciones realizadas en un periodo.
- "Intercambio": es una variable *booleana* que controla si ha habido intercambio o no. En caso de que haya intercambio, las "best-bid" y las "best-offer" se reinician.

Las variables globales definidas en la programación del modelo y relacionadas con los resultados son:

- "Eficiencia": es el porcentaje de la suma de los beneficios obtenidos por compradores y vendedores respecto al excedente total.
- "Velocidad-convergencia": es el porcentaje respecto al precio de equilibrio competitivo de la desviación típica de los precios de los intercambios (en relación con el precio de equilibrio competitivo).
- "Separador": es una variable que permite representar gráficamente los precios de los intercambios asociados a cada periodo de la simulación.

Las variables de estado de los compradores son:

- "Cantidad-pendiente-negociar": es la cantidad asignada para negociar. Su valor disminuye cuando hay una transacción. Inicialmente, cada agente dispone de 1 unidad. Cuando el agente no dispone de unidades para intercambiar, no puede participar en el mercado.
- "P-reserva" (en unidades monetarias): es el precio de reserva o precio máximo al que cada comprador está dispuesto a comprar.
- "Bid" (en unidades monetarias): es la oferta de compra que piensa un comprador. Éste puede o no subirla al mercado.
- "Beneficio" (en unidades monetarias): es la diferencia entre el precio de reserva y el precio del intercambio.

Las variables de estado de los vendedores son:

- "Cantidad-pendiente-negociar": es la cantidad asignada para negociar. Su valor disminuye cuando hay una transacción. Inicialmente, cada agente dispone de 1 unidad. Cuando el agente no dispone de unidades para intercambiar, no puede participar en el mercado.
- "Coste" (en unidades monetarias): es el coste marginal o precio mínimo al que cada vendedor está dispuesto a vender.
- "Offer" (en unidades monetarias): es la oferta de venta que un comprador. Éste puede o no subirla al mercado.
- "Beneficio" (en unidades monetarias): es la diferencia entre el precio del intercambio y el coste para los vendedores.

Aunque en Netlogo las tortugas (en este caso, compradores y vendedores) deben situarse en el *grid*, su posición en él no es relevante en este modelo, dado que la posición espacial de los agentes no influye en un mercado centralizado como la subasta doble continua. Por el contrario, en un mercado descentralizado, la posición de los agentes en el *grid* sería muy relevante.

6.2.3 Proceso general y secuencia

Al principio de la simulación, se inicializa el modelo en el orden indicado:

1. Se crean los compradores y vendedores y los inicializa ("setup traders").
2. Se inicializa el entorno ("setup-entorno").
3. Se inicializa el mercado ("setup-mercado").

El tiempo trascurre de forma discreta por periodos. En cada periodo, las siguientes acciones son ejecutadas cíclicamente hasta el final de la simulación en el orden indicado.

1. Se realizan las pujas y los intercambios en el mercado ("pujar-CDA").
2. Se calculan los resultados: eficiencia y convergencia ("resultados-mercado").
3. Se reinician los agentes ("setup-traders").
4. Se reinicia el mercado ("setup-mercado").

6.2.4 Conceptos de diseño

- "Principios básicos": Este modelo es una versión simplificada del modelo de mercado de Posada y López (2010), donde se analiza la convergencia y la eficiencia de los mercados con diferentes tipos de aprendizaje en los agentes.
- "Emergencia": Como consecuencia de la interacción entre agentes, emerge el precio de intercambio.
- "Adaptación": Los agentes adaptan sus ofertas de compra y de venta en función de los precios de intercambio del mercado.
- "Objetivos": En la vida real, compradores y vendedores persiguen obtener el máximo beneficio: los compradores comprando lo más barato posible (y siempre por debajo de su precio de reserva); mientras que los vendedores, vendiendo lo más caro posible (y siempre por encima de su coste). En nuestro modelo, al ser "zero-inteligentes" (ZI), compradores y vendedores persiguen sólo obtener beneficios positivos.
- "Aprendizaje": Los agentes ZI no aprenden.
- "Predicción": No hay predicción en el modelo.
- "Detección": En función de las pujas subidas al mercado, los agentes piensan sus pujas y aceptan las pujas hechas por otros agentes.
- "Interacción": Los compradores y vendedores interactúan mediante la institución del mercado.
- "Aleatoriedad": En la inicialización de los precios de reserva y de los costes, y en la interacción entre compradores y vendedores, como se verá más adelante.
- "Colectivos": No hay organización de los agentes de más alto nivel que las tortugas.

- "Observación": Se observa la eficiencia y la velocidad de convergencia en el mercado como consecuencia del intercambio entre compradores y vendedores.

6.2.5 Inicialización

Variables	Rango de valores
Periodos	[1-15] a través de la interfaz
Numero-vendedores	[1-50] a través de la interfaz
Numero-compradores	[1-50] a través de la interfaz
Oferta-min, oferta-max, demanda-min, demanda-max	[1-100] a través de la interfaz
Precio de reserva	Aleatoria entre [demanda-min, demanda-max]
Coste	Aleatoria entre [oferta-min, oferta-max]
Unidades a negociar	1

El resto de variables se inicializan, por defecto, con el valor 0.

6.2.6 Datos de entrada

Se asume que el entorno permanece constante durante la simulación, por lo que este modelo no utiliza ningún fichero con datos de entrada en función del tiempo.

6.2.7 Submodelos

En esta sección sólo incluimos la descripción del procedimiento. La programación de estos procedimientos (o en su defecto, el pseudocódigo) no se ha incluido, ya que ésta se detalla más adelante.

"Setup traders": Asigna una unidad para negociar y pone sus beneficios a cero.
"Setup-entorno": Construye la oferta ("crear-oferta") y la demanda ("crear-demanda"), las representa gráficamente ("representar-entorno") y calcula el precio de equilibrio, la cantidad de equilibrio y los excedentes totales.
"Setup-mercado": Inicializa las "best-bid" y la "best-offer", y vacía la lista de precios.
"Pujar-CDA": Un agente (comprador o vendedor), elegido al azar entre los que pueden participar en el mercado mejorando la puja actual, sube su puja al mercado ("subir-offer-ZI", "subir-bid-ZI"). Los agentes que pueden participar en el mercado aceptando la puja subida, piensan en una puja ("pensar-offer-ZI", "pensar-bid-ZI"). Un agente, elegido al azar entre aquellos que estarían dispuestos a aceptar la puja subida al mercado (esto es, si la oferta de venta es menor que "best-bid" o la oferta de compra en mayor que la "best-offer"), acepta la puja y la transacción se realiza al precio de la puja subida al mercado. Los agentes involucrados en la transacción calculan los beneficios obtenidos y actualizan las unidades pendientes de negociar ("update-comprador", "update-vendedor").

La programación de este procedimiento es clave y debe cumplir dos características: reproducir la dinámica de la CDA y ser independiente del tipo de comportamiento de los agentes.

- En relación con la primera, uno puede pensar que el siguiente procedimiento es más sencillo de programar: seleccionar al azar un comprador y un vendedor, y realizar la transacción si la oferta de compra es mayor que la oferta de venta a un precio intermedio entre ambas. Sin embargo, esto no se corresponde con una CDA, sino con una *clearing-house*.
- En relación con la segunda, el procedimiento debe permitir sustituir los procedimientos relacionados las pujas de los agentes ZI ("pensar-offer-ZI", "subir-offer-ZI", "pensar-bid-ZI", "subir-bid-ZI") por otros tipos de comportamientos más inteligentes sin más que llamar al procedimiento correspondiente.

"Resultados del mercado": Calcula la eficiencia del mercado y la convergencia de los precios y las representa gráficamente ("representar-resultados").

6.3 El modelo Netlogo de intercambio en un mercado

En el mercado de subasta doble continua participan dos tipos de agentes (compradores y vendedores). Cada agente puede negociar una unidad. Cada comparador recibe el precio que estaría dispuesto a pagar por dicha unidad (que llamamos precio de reserva). Cada vendedor recibe el coste de fabricar esa unidad (que llamamos coste marginal). Los precios de reserva y los costes marginales se generan de forma aleatoria entre un valor mínimo y un valor máximo, y la agregación de los mismos conforman la demanda y la oferta del mercado, respectivamente. La intersección de la oferta y la demanda nos da la cantidad y el precio de equilibrio competitivo del mercado, de acuerdo con la teoría económica.

Los agentes del mercado realizan y aceptan pujar con el objetivo de obtener el máximo beneficio, de forma que los compradores quieren comprar lo más barato posible y siempre por debajo de su precio de reserva, mientras que los vendedores quieren vender lo más caro posible y siempre por encima de su coste.

Consideramos que los agentes que participan en el mercado son "zero-inteligentes" (ZI). Esto significa que hacen sus pujas de forma aleatoria con la única condición de obtener beneficios.

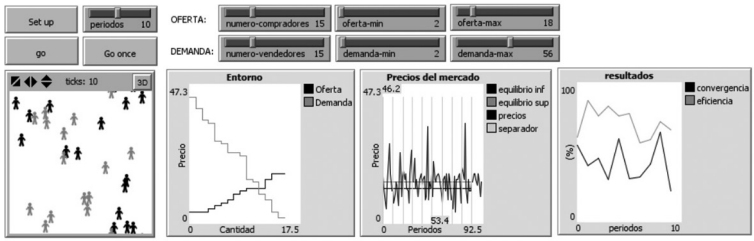
6.3.1 La interfaz

La programación del modelo de intercambio del mercado se encuentra disponible es www.eii.uva.es/posada/manualdenetlogo-capVI.html. En la pestaña "Ejecutar" se encuentra la interfaz del programa (Figura 6.4). Ésta incluye:

- Tres deslizadores para establecer/definir la demanda: número de compradores y los valores mínimos ("demanda-min") y máximos

("demanda-max") entre los que se generan de forma aleatoria los precios de reserva.
- Tres deslizadores para establecer/definir la oferta: número de vendedores y los valores mínimos ("oferta-min") y máximos ("oferta-max") entre los que se generan de forma aleatoria los costes.
- Un deslizador para establecer los periodos de la simulación (periodos).

Figura 6.4
Interfaz del modelo "Intercambio en un mercado"



El funcionamiento del programa se controla con los siguientes botones:

- El botón "Set up" inicializa un nuevo mundo y representa las gráficas de oferta y demanda con las condiciones iniciales establecidas a través de los botones deslizantes. Para fines de claridad, la oferta y compradores son de color negro, mientras que la demanda y los compradores son de color gris. Aunque situamos a compradores y vendedores en parcelas blancas, su posición en el mundo no es relevante para su participación en un mercado centralizado como es la subasta doble continua.
- Existe un botón nombrado "Go once" que permite ir avanzando la simulación de *tick* en *tick*. Mientras que el botón "Go" avanza todos los periodos de la simulación. Se representan gráficamente los precios de los intercambios de cada periodo y los resultados de los porcentajes de convergencia de precios y eficiencia del mercado.

En este modelo, las instrucciones para dibujar se programan desde la pestaña de código, por lo que las instrucciones de actualización de trazos estarán vacías en los tres gráficos.

6.3.2 La definición de variables

Los distintos tipos de variables se definen en la pestaña código. Al inicio del programa se definen los dos tipos de tortugas de la siguiente manera:

```
breed [ vendedores vendedor ]
breed [ compradores comprador ]
```

A continuación se definen las variables globales:

```
globals
[
        oferta
        demanda
        precio-min
        precio-max
        precio-equilibrio-teorico
        cantidad-equilibrio-teorico
        excedente-total

        best-bid
        best-offer
        list-traders
        precio
        precio-list
        intercambio

        alfa
        eficiencia
        separador
]
```

A continuación se definen las variables de cada tipo de tortuga:

```
links-own [ ]
```

```
vendedores-own
[
        cantidad-pendiente-negociar
        coste
        offer
        beneficios
]
```

```
compradores-own
[
        cantidad-pendiente-negociar
        p-reserva
        bid
        beneficios
]
```

Tras la definición de todas las variables, escribimos el código de programación asociado a los botones "Setup" y "Go".

6.3.3 El código de programación para inicializar el modelo

El código de programación del procedimiento "setup" (inicializar) es:

```
to setup
        clear-all
;; random-seed 681279294
        ask patches [ set pcolor white]    ;; colorea las parcelas de color gris
        set-default-shape turtles "person"
        create-compradores numero-compradores    ;; crea compradores
        [
                setxy random-xcor random-ycor
                set color gray
```

```
            set p-reserva (random (valor-maximo - valor-minimo) + valor-minimo)
            setup-trader
        ]
        create-vendedores numero-vendedores    ;; crea vendedores
        [
            setxy random-xcor random-ycor
            set color black
            set coste (random (valor-maximo - valor-minimo) + valor-minimo)
            setup-trader
        ]
        setup-entorno    ;; inicializa el entorno
        setup-mercado    ;; inicializa el mercado
        reset-ticks
end
```

Al principio del código aparece el comando Netlogo "Clear-all", el cual borra las tortugas de ejecuciones anteriores y pone en *default* las parcelas. Este comando es muy útil para la realización de varios experimentos.

La instrucción ";random-seed 681279294" se ha utilizado durante la programación del modelo para generar siempre el mismo entorno y depurar los errores de la programación.

Insertamos los siguientes comandos: "ask patches [set pcolor white]", para que el color de las parcelas sea blanco con fines de visibilidad en el libro en blanco y negro.

Mediante "set-default-shape turtles "person"" indicamos que las tortugas se representarán como personas.

Luego sigue un conjunto de comandos relacionados con la creación de los compradores:

```
create-compradores numero-compradores    ;; crea compradores
[
        setxy random-xcor random-ycor
        set color gray
        setup-trader
        set p-reserva (random (valor-maximo - valor-minimo) + valor-minimo)
]
```

Se comienza indicando el número de compradores que habrá en el mercado: "create-compradores numero-compradores". Utilizando el corchete, se indica que las siguientes instrucciones les afectan a todos ellos. Se indica que se encuentran localizadas al azar en las parcelas mediante "setxy random-xcor random-ycor", que son de color gris mediante "set color gray" y haciendo la subrutina "setup-trader" se indica que sus beneficios son 0 y que la cantidad disponible para negociar es 1. El procedimiento "setup-trader" es como sigue:

```
to setup-trader
        set cantidad-para-negociar 1
        set beneficios 0
end
```

Por último, se indica que su precio de reserva es un número aleatorio del intervalo [valor máximo, valor mínimo]. La instrucción "random x" genera *x* números aleatorios entre 0 y x. Como en nuestro caso, para indicar que el precio de reserva ("p-reserva") es un número aleatorio entre un valor mínimo y un valor máximo, utilizamos los siguientes comandos: "set p-reserva (random (valor-maximo - valor-minimo) + valor-minimo)".

Luego sigue un conjunto de comandos relacionados con la creación de los vendedores:

```
create-vendedores numero-vendedores    ;; crea vendedores
[
        setxy random-xcor random-ycor
        set color black
        setup-trader
        set coste (random (valor-maximo - valor-minimo) + valor-minimo)
]
```

Se comienza indicando el número de compradores que habrá en el mercado: "create-compradores numero-compradores". Utilizando el corchete, se indica que las siguientes instrucciones les afectan a todos ellos. Se indica que se encuentran localizadas al azar en las parcelas mediante

"setxy random-xcor random-ycor", que son de color negro mediante "set color black" y haciendo la subrutina "setup-trader" se indica que sus beneficios son 0 y que la cantidad disponible para negociar es 1.

Por último, indicamos que su coste es un número aleatorio del intervalo [valor máximo, valor mínimo] mediante los siguientes comandos: "set coste (random (valor-maximo - valor-minimo) + valor-minimo)".

Luego, el programa llama los procedimientos "setup-mercado", "setup-entorno" y al comando "reset-ticks" que pone en cero todos los *ticks*.

El procedimiento "setup-mercado" establece como la mejor oferta de compra ("best-bid") el valor mínimo de demanda (esto es, el más barato posible) y la mejor oferta de venta ("best-offer") el valor máximo de la oferta (esto es, el más caro posible) como sigue:

```
to setup-mercado
        set best-bid demanda-min
        set best-offer oferta-max
end
```

El procedimiento "setup-entorno" crea la oferta y la demanda, y calcula precio y cantidad de equilibrio competitivo, así como los excedentes totales como sigue:

```
to setup-entorno
        crear-demanda
        crear-oferta
        representar-graficas
        set cantidad-equilibrio-teorica interseccion oferta demanda
        set precio- min valor cantidad-equilibrio-teorica oferta
        set precio-max valor cantidad-equilibrio-teorica demanda
        set precio-equilibrio-teorico (precio-min + precio-max) / 2
let i 0    ;; excedente teorico
        while [i < cantidad-equilibrio-teorica]
                [set excedente-total excedente-total + item i demanda - item i oferta
                set i i + 1
                ]
end
```

El procedimiento “setup-entorno” llama en primer lugar al procedimiento “crear-oferta”, que crea una lista de los costes asignados a los vendedores, ordenada de menor a mayor. Para ellos se crea una lista temporal inicialmente vacía mediante los comandos “let lista []”. Dado que el coste es una variable de los vendedores, hay que preguntarles por su valor, y utilizando el comando “sentence” se indica que añada dicho coste a la lista: “ask vendedores [set lista sentence coste lista]”. El comando “sort” ordena la lista de menor a mayor, y mediante la instrucción “set oferta sort lista” se indica que esa lista ordenada es la oferta.

```
to crear-oferta
        let lista []
        ask vendedores [ set lista sentence coste lista]
        set oferta sort lista    ;; ordena la lista de menor a mayor
end
```

A continuación, el procedimiento “setup-entorno” llama al procedimiento “crear-demanda”, que crea una lista de los precios de reserva asignados a los compradores, ordenada de mayor a menor. Para ellos se crea una lista temporal inicialmente vacía mediante los comandos “let lista []”. Dado que el precio de reserva es una variable de los compradores, hay que preguntarles por su valor, y utilizando el comando “sentence” se indica que añada dicho precio de reserva a la lista: “ask compradores [set lista sentence p-reserva lista]”. El comando “sort-by >” ordena la lista de mayor a menor, y mediante la instrucción “set demanda sort-by > lista” se indica que esa lista ordenada es la demanda.

```
to crear-demanda
        let lista []
        ask compradores [set lista sentence p-reserva lista]
        set demanda sort-by > lista    ;; ordena la lista de mayor a menor
end
```

A continuación, el procedimiento “setup-entorno” llama al procedimiento “representar graficas”, que representa gráficamente la oferta y la demanda.

```
to representar-graficas
        set-current-plot “Entorno”
        clear-plot
        set-current-plot-pen “Oferta “
        representar-grafica oferta
        set-current-plot-pen “Demanda “
        representar-grafica demanda
end
```

En el procedimiento “representar graficas” en primer lugar se indica el nombre del gráfico donde se van a representar las gráficas mediante “set-current-plot “Entorno””. Tras borrar todo con el comando “clear-plot”, se indica que se va a dibujar la oferta teórica mediante las siguientes instrucciones:

```
set-current-plot-pen “Oferta”
representar-grafica oferta “Oferta”
```

y que va a dibujar la demanda teórica mediante las siguientes instrucciones:

```
set-current-plot-pen “Demanda “
representar-grafica demanda “Demanda “
```

En ambos casos, el procedimiento “representar-graficas” llama al procedimiento “representar-grafica” indicando si se trata de la gráfica de la demanda o de la gráfica de la oferta.

```
to representar-grafica [grafica name-plot-pen]
        let precio-grafica 0
        let cantidad-grafica 0
        plot-pen-down
        foreach grafica
        [
                set precio-grafica ?
                plotxy cantidad-grafica precio-grafica
```

```
            set cantidad-grafica cantidad-grafica + 1
            plotxy cantidad-grafica precio-grafica
        ]
        if name-plot-pen = “Oferta”[plotxy cantidad-grafica max (list oferta-max
        demanda-max) ]
        if name-plot-pen = “Demanda”[plotxy cantidad-grafica 0]
        plot-pen-up
end
```

El procedimiento “representar-grafica [grafica name-plot-pen]” inicializa a cero dos variables temporales: precio-grafica y cantidad-grafica, mediante las instrucciones “let precio-grafica 0” y “let cantidad-grafica 0”, respectivamente. Mediante el comando “plot-pen-down” se indica que comenzamos a dibujar. Utilizando el comando “foreach” recorremos la lista de la oferta o de la demanda, de forma que vamos trazando las líneas horizontales y verticales de la siguiente manera:

```
set precio-grafica ?
        plotxy cantidad-grafica precio-grafica
        set cantidad-grafica cantidad-grafica + 1
        plotxy cantidad-grafica precio-grafica
```

Antes de levantar el bolígrafo (“plot-pen-up”), las siguientes instrucciones dibujan al final de la demanda una línea vertical hacia abajo y al final de la oferta una línea vertical hacia arriba.

```
        if name-plot-pen = “Oferta”[plotxy cantidad-grafica max (list oferta-max
        demanda-max) ]
        if name-plot-pen = “Demanda”[plotxy cantidad-grafica 0]
```

A continuación, el procedimiento “representar-graficas” calcula el precio y la cantidad de equilibrio mediante los siguientes comandos:

```
set cantidad-equilibrio-teorico interseccion oferta demanda
set precio-min valor cantidad-equilibrio-teorico oferta
set precio-max valor cantidad-equilibrio-teorico demanda
set precio-equilibrio-teorico (precio-min + precio-max) / 2
```

En primer lugar, se calcula la cantidad de equilibrio entre la oferta y la demanda, llamando al procedimiento “interseccion”.

```
to-report interseccion [ oferta-cuantia demanda-cuantia ]
        let i 0
        while [ ( not empty? oferta-cuantia ) and ( not empty? demanda-cuantia )
        and ( ( first oferta-cuantia ) <= ( first demanda-cuantia ) ) ]
        [
                set oferta-cuantia but-first oferta-cuantia
                set demanda-cuantia but-first demanda-cuantia
                set i i + 1
        ]
        report i
end
```

En el procedimiento “interseccion”, en primer lugar, se inicializa a cero una variable temporal para contar el número de intercambios mediante “let numero 0”. A continuación, mientras las listas de oferta y demanda no estén vacías, se coge el primer elemento de la oferta y el primero de la demanda; y mientras el elemento de la oferta sea menor que el elemento de la demanda, se produce el intercambio,

```
while [ ( not empty? oferta-cuantia ) and ( not empty? demanda-cuantia )
and ( ( first oferta-cuantia ) <= ( first demanda-cuantia ) ) ]
```

y elimina el primer elemento tanto de la oferta como de la demanda utilizando “set oferta-cuantia but-first oferta-cuantia” y “set demanda-cuantia but-first demanda-cuantia” respectivamente, y se incrementa el contador de intercambios “set i i + 1”. El número de intercambios se monitoriza por pantalla mediante la instrucción “report i”.

Para esa cantidad de equilibrio, el precio que los compradores están dispuestos a pagar (precio-demanda) puede no coincidir con el precio al que están dispuestos a vender los vendedores (precio-oferta), de forma que el precio de equilibrio es el valor medio entre ambos precios:

```
set precio-equilibrio-teorico (precio-min + precio-max) / 2
```

Para calcular el precio de equilibrio es necesario haber calculado previamente dichos precios (que son variables locales), mediante las siguientes instrucciones:

```
let precio-min valor cantidad-equilibrio-teorico oferta
let precio-max valor cantidad-equilibrio-teorico demanda
```

En ambos casos se llama al procedimiento "valor", que es como sigue y que se obtiene de la lista de la oferta o de la demanda (según corresponda), el ítem que se corresponde con la cantidad de equilibrio - 1, dado que en Netlogo el primer elemento de una lista es el ítem 0:

```
to-report valor [ q-equilibrio grafica ]
        ifelse q-equilibrio > 0
                [ report item ( q-equilibrio - 1 ) grafica ]
                [ report "Ninguno" ]
end
```

Finalmente, el procedimiento "representar-graficas" calcula el excedente teórico mediante los siguientes comandos:

```
let i 0 ;excedente teorico
        while [i < cantidad-equilibrio-teorica]
                [set excedente-total excedente-total + item i demanda - item i oferta
                set i i + 1
        ]
end
```

6.3.4 El código de programación para correr el modelo

El botón "Go" llama al procedimiento "go-once":

```
to go
        while [ticks < periodos]
```

```
        [go-once]
end
```

El código del procedimiento “go-once” es:

```
to go-once
    ifelse ticks = periodos
    [stop]
    [pujar-CDA
    pujar-CDA
    resultados-mercado
    ask turtles [setup-trader]
    setup-mercado
    tick]
end
```

El procedimiento “go-once” llama en primer lugar al procedimiento “pujar-CDA”; lo hace dos veces con el objeto de garantizar que haya tiempo de sobra para realizar los intercambios. Después se calculan los resultados del mercado (eficiencia y convergencia) mediante el procedimiento “resultados-mercado”; se reinician los agentes mediante el procedimiento “setup-trader” (esto es, se pone el beneficio a 0 y la cantidad disponible para negociar a 1); y finalmente se reinicia el mercado (esto es, se vacía la lista de precios, se establece como la mejor oferta de compra (“best-bid”) el valor mínimo de demanda y la mejor oferta de venta (“best-offer”) el valor máximo de la oferta).

```
to pujar-CDA
set intercambio false
set traders-list []
ask compradores with [ (cantidad-pendiente-negociar > 0) and (p-reserva >= best-bid)]
[set traders-list sentence traders-list who]
ask vendedores with [ (cantidad-pendiente-negociar > 0) and (coste <= best-offer)] [set
traders-list sentence traders-list who]
let i 0
while [i < length traders-list]
[ask turtle item (random length traders-list) traders-list
    [if breed = compradores
        [
```

```
            subir-bid-ZI
            set best-bid bid    ;; se actualiza la bid del mercado
            ask vendedores with [ cantidad-pendiente-negociar > 0] [pensar-offer-ZI]
            ;; piensa la puja pero no la sube
            let trader one-of vendedores with [(cantidad-pendiente-negociar > 0) and
            (offer <= best-bid)]    ;; algún vendedor acepta?
            if trader != nobody
                [ask trader
                    [set intercambio true    ;; hay intercambio
                    set precio best-bid    ;; precio=puja subida al mercado (bid)
                    set precio-list sentence precio-list precio
                    update-vendedor
                    set best-bid demanda-min    ;; se reinicia la puja a mejorar (bid)
                ]]
            if intercambio [update-comprador]
            ]
        if breed = vendedores
            [
                subir-offer-ZI
                set best-offer offer ;sube puja al mercado (offer)
                ask compradores with [ cantidad-pendiente-negociar > 0] [pensar-bid-ZI]
                ;piensa la puja pero no la sube
                let trader one-of compradores with [ (cantidad-pendiente-negociar) > 0
                and (bid >= best-offer)]    ;; algún comprador acepta?
            if trader != nobody
                [ask trader
                    [set intercambio true    ;; hay intercambio
                    set precio best-offer    ;; precio=puja subida al mercado (offer)
                    set precio-list sentence precio-list precio
                    update-comprador
                    set best-offer oferta-max    ;; se reinicia la puja a mejorar (bid)
                ]]
            if intercambio [update-vendedor]
            ]
            set intercambio false
            ]
        set traders-list []
        ask compradores with [ (cantidad-pendiente-negociar > 0)
        and (p-reserva >= best-bid)] [set traders-list sentence traders-list who]
        ask vendedores with [ (cantidad-pendiente-negociar > 0)
        and (coste <= best-offer)] [set traders-list sentence traders-list who]
        set i i + 1
        ]
End
```

En el procedimiento "pujar-CDA", inicialmente se crea una lista con los compradores y vendedores que pueden optar por negociar, es decir, aquellos que tienen una cantidad pendiente de negociar mayor que 0 y que pueden superar las mejores pujas.

```
set list-traders []
ask compradores with [ (cantidad-pendiente-negociar > 0) and (p-reserva >= best-bid)]
[set list-traders sentence list-traders who]
ask vendedores with [ (cantidad-pendiente-negociar > 0) and (coste <= best-offer)] [set
list-traders sentence list-traders who]
```

La instrucción "sentence" se utiliza para añadir comparadores o vendedores (identificándolos mediante la instrucción "who") a la lista. Dado que la variable "p-reserva" ha sido definida sólo para los compradores y el "coste" sólo para los vendedores, debemos preguntarles de forma independiente. Es importante utilizar una lista en lugar de la instrucción "ask turtles with [(cantidad-pendiente-negociar > 0)", para evitar preguntar en un periodo a un participante que haya realizado previamente un intercambio en ese mismo periodo.

Se elige al azar un agente de esa lista. Si ese agente es un comprador, hace una oferta de compra ("subir-bid-ZI"), la sube al mercado y por tanto, actualiza la mejor oferta de compra ("best-bid"). Todos los vendedores piensan una oferta de venta ("pensar-offer-ZI"):

```
ask turtle item (random length list-traders) list-traders

[if breed = compradores
      [
      set best-bid bid    ;; se actualiza la bid del mercado
      ask vendedores with [ cantidad-pendiente-negociar > 0] [pensar-offer-ZI]
      ;piensa la puja pero no la sube

De entre los vendedores que estarían dispuestos a hacer una oferta de compra
(offer) más barata que la oferta de compra hecha por el comprador, se selecciona
aleatoriamente a uno.

let trader one-of vendedores with [(cantidad-pendiente-negociar > 0)
and (offer <= best-bid)]
```

Si hay alguno ("if trader != nobody"), se realiza el intercambio pero al precio de la oferta de compra subida al mercado ("set precio best-bid"). Se añade el precio en una lista de precios, se actualizan los datos del vendedor mediante el procedimiento "update-vendedor" (esto es, se calculan los beneficios y se disminuye la cantidad pendiente de negociar) y se reinicia la oferta de compra con el valor mínimo de la demanda (esto es, el más barato posible).

```
ask trader
        [set intercambio true    ;; hay intercambio
        set precio best-bid    ;; precio=puja subida al mercado (bid)
        set precio-list sentence precio-list precio
        update-vendedor
        set best-bid demanda-min    ;; se reinicia la puja a mejorar (bid)
        ]]
```

Finalmente, si se ha realizado el intercambio, el comprador que subió la oferta de compra al mercado también actualiza sus datos (esto es, se calculan los beneficios y se disminuye la cantidad pendiente de negociar).

```
if intercambio [update-comprador]
```

Los procedimientos para actualizar los datos de los compradores y vendedores que realizan el intercambio son "update-comprador" y "update-vendedor":

```
to update-comprador
        set cantidad-pendiente-negociar cantidad-pendiente-negociar - 1
        set beneficios beneficios + p-reserva - precio
end
```

```
to update-vendedor
        set cantidad-pendiente-negociar cantidad-pendiente-negociar - 1
        set beneficios beneficios + precio - coste
end
```

Si el agente de la lista elegido al azar es un vendedor, se procede de forma análoga. Se indica que el intercambio es "false" y se recalcula la lista de los agentes que pueden realizar el siguiente intercambio. Dicha lista no incluye a los agentes que ya han negociado su unidad, o bien a aquellos cuyos costes o precios de reserva no les permite hacer una oferta con la que obtengan beneficio. Finalmente, se actualiza el contador.

Cabe destacar que este procedimiento es común para cualquier tipo de comportamiento de los agentes. En el caso de que los agentes sean ZI ("zero-inteligentes"), se llama a los procedimientos "pensar-bid-ZI" (los compradores hacen una oferta de compra (*bid*) de forma aleatoria por debajo de su precio de reserva), "subir-bid-ZI" (los compradores hacen una oferta de compra de forma aleatoria por debajo de su precio de reserva y que mejore la oferta de compra subida al mercado), "pensar-offer-ZI" (los vendedores hacen una oferta de venta (*offer*) de forma aleatoria por encima de su coste) y "subir-offer-ZI" (los vendedores hacen una oferta de venta de forma aleatoria por encima de su coste y que mejore la oferta de venta subida al mercado).

```
to pensar-bid-ZI
      set bid (p-reserva - random (p-reserva - demanda-min))    ;; piensa puja entre
      [valor-minimo-demanda, p-reserva]
end
```

```
to subir-bid-ZI
      set bid (p-reserva - random (p-reserva - best-bid) )    ;; hace puja entre [best-bid,
      p-reserva]
end
```

```
to pensar-offer-ZI
      set offer (coste + random (oferta-max - coste) )    ;; piensa puja entre [coste,
      valor-maximo-oferta]
end
```

```
to subir-offer-ZI
      set offer (coste + random (best-offer - coste))    ;; hace puja entre [coste, best-offer]
end
```

Tras los intercambios realizados, se calculan la convergencia en precios y la eficiencia del mercado de acuerdo con Smith (1962), mediante el procedimiento “resultados-mercado”.

```
to resultados-mercado
;;------eficiencia
        set eficiencia round (100 * sum [ beneficios ] of turtles / excedente-total)
;;------convergencia
        set coeficiente 0
        foreach precio-list [set coeficiente coeficiente + ( ? - precio-equilibrio-teorico ) ^ 2]
        if ( length precio-list >= 2 ) [set alfa 100 * sqrt ( coeficiente / ( length precio-list - 1 ) ) /
        precio-equilibrio-teorico]
        representar-resultados
end
```

Finalmente, representamos los resultados mediante el procedimiento "representar resultados". En la gráfica “precios del mercado” incluimos los intercambios realizados en cada periodo. Dado que en cada periodo hay más de un intercambio, se recorre la lista de precios utilizando la instrucción “foreach”, indicando que dibuje cada elemento de la lista [plot ?]:

```
set-current-plot-pen “precios”
foreach precio-list [plot ?]
```

Además, es necesario calcular el número de intercambios del periodo para establecer el separador de periodos de simulación. Calculados el punto inferior y superior, bajamos el lápiz para unir los valores inferior y superior, pero lo levantamos para no realizar un trazo entre el nivel superior y el siguiente. Recomendamos al lector poner un punto y coma en “;plot-pen-up” para que dicha instrucción se considere un comentario y ver cuáles son sus consecuencias gráficas.

```
to graficas-resultados

        set-current-plot “precios del mercado”
        set-current-plot-pen “precios”
                foreach precio-list [plot ?]
```

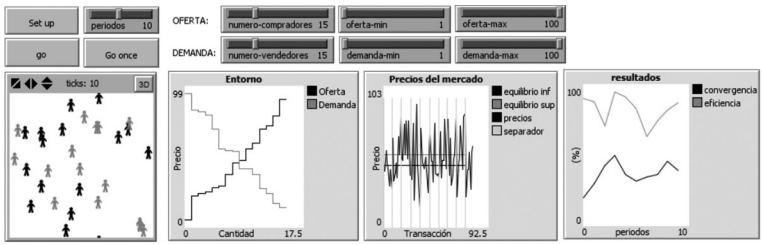
```
        set-current-plot-pen “equilibrio inf”
                plotxy negociacion precio-min
        set-current-plot-pen “equilibrio sup”
                plotxy negociacion precio-max
        set-current-plot-pen “separador”
                plot-pen-up
                plotxy negociacion 0
                plot-pen-down
                plotxy negociacion max (list max [p-reserva] of compradores max [coste] of
                vendedores)
        set-current-plot “resultados”
        set-current-plot-pen “eficiencia”
                plot eficiencia
        set-current-plot-pen “convergencia”
                plot alfa
        set negociacion negociacion + length precio-list
end
```

6.4 Experimentos con el modelo

En esta sección solamente puntualizamos algunos aspectos relevantes de la dinámica del modelo con el fin de motivar al lector a que replique el modelo por sí mismo y que lo amplíe con los comportamientos inteligentes considerados en Posada y López (2010).

Recomendamos utilizar como experimento de referencia un mercado equilibrado donde el número de compradores y el número de vendedores sea el mismo, y la elasticidad de la oferta y de la demanda en el punto de equilibrio sean iguales (Figura 6.5).

Figura 6.5
Escenario 0- escenario de referencia

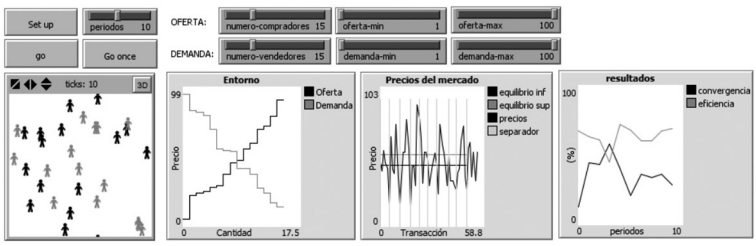


Obtenemos las siguientes conclusiones:

- En la gráfica de precios del mercado observamos que los precios de intercambio no siguen una pauta, sino que oscilan aleatoriamente alrededor del precio de equilibrio. No se observa convergencia hacia el precio de equilibrio competitivo, como ocurre en los experimentos con humanos. Esto se debe a que los agentes con un comportamiento ZI no aprenden al hacer sus pujas. Si queremos reproducir el comportamiento de los humanos con agentes artificiales, es necesario dotarlos de inteligencia.
- En la gráfica de resultados no se observa ninguna evolución en la eficiencia del mercado ni en la convergencia de los precios de un periodo al siguiente de la simulación. Esto se debe a que los agentes con un comportamiento ZI no aprenden al hacer sus pujas.
- En la gráfica de resultados se observa una eficiencia del mercado relativamente alta. Esto se debe a las reglas de intercambio. Si en la programación ponemos un punto y coma delante de una de las llamadas al procedimiento "puja-CDA", sería equivalente a reducir el tiempo en el que el mercado está abierto para hacer los intercambios (Escenario 1). En ese caso, la eficiencia del mercado disminuye porque no ha habido de realizar todos los intercambios posibles (Figura 6.6).

Figura 6.6

Escenario 1-reducción de opciones de intercambio

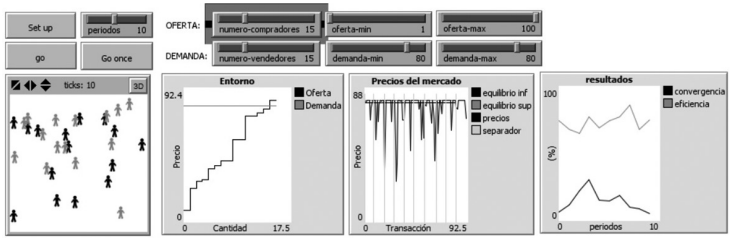


En el escenario 2 (Figura 6.7) se analiza un mercado con una oferta infinitamente elástica. Observamos que no se alcanza el precio de equilibrio de mercado, que, según establece la teoría económica, coincide con el precio de reserva de los compradores. Los precios de las transacciones

oscilan alrededor del centro de masas del excedente de los vendedores. Esto se debe al comportamiento ZI de los agentes.

Figura 6.7

Escenario 2- demanda elástica



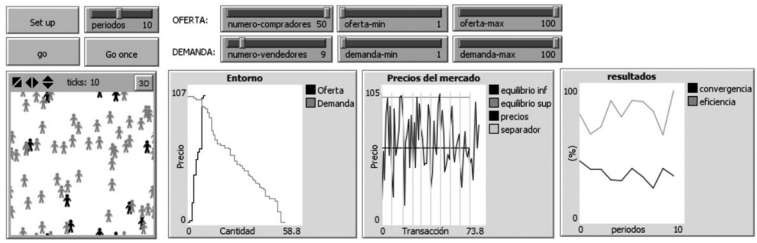
Si se analiza un mercado con una demanda infinitamente elástica, obtenemos unos resultados similares: el precio de equilibrio de mercado (que según la teoría económica coincide con el coste de los vendedores) no se alcanza. Esto es debido al comportamiento ZI de los agentes.

En el escenario 3 (Figura 6.8) se analiza un mercado con exceso de demanda ("número de compradores > número de vendedores"). Observamos que no se alcanza el precio de equilibrio de mercado (que según la teoría económica es el precio mínimo del intervalo de equilibrio). Esto se debe al comportamiento ZI de los agentes.

Si se analiza un mercado con exceso de oferta ("número de compradores < número de vendedores"), obtenemos unos resultados similares: el precio de equilibrio de mercado (que según la teoría económica es el precio máximo del intervalo de equilibrio) no se alcanza. Esto es debido al comportamiento ZI de los agentes.

Figura 6.8

Escenario 3- exceso de demanda



CAPÍTULO VII

TEORÍA COMPUTACIONAL DE LAS ORGANIZACIONES

UN MODELO DE ROTACIÓN INVOLUNTARIA DE PERSONAL QUE TRABAJA EN EQUIPO

7.1 Introducción a la rotación de personal que trabaja en equipo

El objetivo de este capítulo es profundizar en la programación en Netlogo. En concreto, nos centramos en la definición de agentes del tipo vínculo. En la biblioteca de modelos de Netlogo, en la carpeta Networks, podemos encontrar varios ejemplos relacionados. El modelo que se presenta en este capítulo es una versión simplificada de modelo de rotación de personal propuesto por Posada *et al.* (2017).

Pero antes de entrar en detalle sobre la programación en Netlogo, haremos una introducción a los conceptos clave de trabajo en equipo y rotación de personal.

Salas *et al*. (2005) resumen en 5 componentes (que llaman "big five") lo que los investigadores deben saber acerca del trabajo en equipo. Estos componentes son:

- Estilo de liderazgo: Habilidad para dirigir y coordinar las actividades de los miembros del equipo para conseguir los objetivos del proyecto.
- Monitorización del desempeño común: Habilidad para desarrollar un entendimiento común en el equipo y aplicar objetivos estratégicos para monitorizar el desempeño.
- Comportamiento de apoyo: Habilidad para anticipar las necesidades de los otros miembros del equipo.
- Adaptabilidad: Habilidad para ajustar las estrategias basadas en la información generada desde el entorno mediante el uso del comportamiento de apoyo y la reasignación de los recursos del equipo.

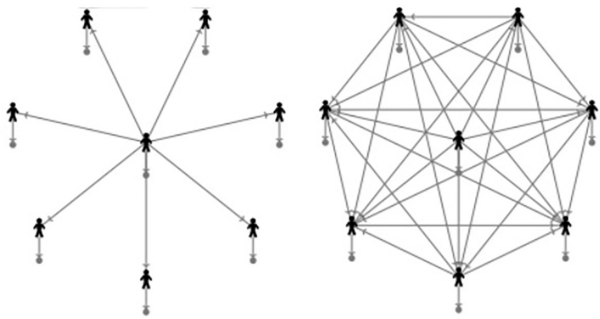
- Orientación del equipo: Propensión para tomar en cuenta otros comportamientos en las inteacciones del grupo y la creencia en la importancia del objetivo del equipo por encima de los objetivos individuales.

Nosotros nos centramos en el estilo de liderazgo porque es el que dicta las interacciones formales entre los miembros del equipo. Los individuos trabajando en equipo juegan uno de los siguientes roles: gestor del proyecto o miembro del equipo. Los dos tipos extremos de liderazgo de un proyecto son autoritario y democrático.

- En un equipo con un liderazgo autoritario los miembros del equipo no interactúan entre sí. Dado que cada miembro del equipo sólo interactúa con el gestor del proyecto, el número de interacciones es n-1, donde *n* es el tamaño del equipo. Este estilo de liderazgo suele asociarse con una configuración en estrella situando al gestor el proyecto en el centro (Figura 7.1, izquierda).
- En un equipo con un liderazgo democrático los miembros del equipo interactúan entre sí, además de interactuar con el gestor del proyecto (Figura 7.1, derecha). El número de interacciones es el resultado de las combinaciones sin repetición de *n* elementos tomados de 2 en 2, esto es n *(n-1)/2, donde *n* es el tamaño del equipo.

Figura 7.1

Estilos de liderazgo en equipos



Autoritario (izquierda) versus democrático (derecha).

Para modelar el comportamiento de los miembros del equipo respecto al esfuerzo que realizan, nos basamos en el trabajo de Dal Forno y Merlone (2002) que, con el fin de observar la emergencia de una cultura corporativa, diseñan un modelo en el que los individuos interactúan en un *grid* modificando la intensidad de su esfuerzo en función de su tipo de comportamiento y el del individuo con el que actúan. Por ejemplo, el individuo aumenta su esfuerzo si éste es la mitad del que hace su compañero. Por el contrario, si su compañero no se esfuerza en el cumplimiento de sus tareas no hay incentivo para que el individuo también lo haga.

7.2 Descripción del modelo de rotación de personal mediante el protocolo ODD

7.2.1 Propósito

Analizar la emergencia de una cultura corporativa ante el esfuerzo de los equipos de trabajo con diversos estilos de liderazgo (autoritario frente a democrático), diferentes tipos de comportamiento frente al esfuerzo y distintos umbrales de esfuerzo mínimo en la política de despidos de la empresa.

7.2.2 Entidades, variables de estado y escalas

El modelo incluye dos tipos de agentes: vínculos y tortugas, así como dos clases de tortugas: puestos de trabajo ("jobs") y trabajadores ("workers").

Las variables globales definidas en la interfaz del modelo son:

- "Years": es el número de periodos de la simulación. Su valor se asigna desde la interfaz.
- "Team-size": es el tamaño del equipo. Su valor se asigna desde la interfaz.

- "Team-leadership": es el estilo de liderazgo del gestor del equipo, el cual puede ser de dos tipos: autoritario ("star") o democrático ("whole"). Se configura desde la interfaz entre los definidos.
- "Population": es el tipo de comportamiento frente al esfuerzo que tendrán los empleados, el cual puede ser de siete tipos: "null", "high", "shriking", "averager", "imitator", "replicator" y "comparator". Se configura desde la interfaz entre los definidos.
- "New-employees": es el tipo de comportamiento frente al esfuerzo que tendrán los nuevos empleados, el cual puede ser de siete tipos: "null", "high", "shriking", "averager", "imitator", "replicator" y "comparator". Se configura desde la interfaz entre los definidos.
- "Firing-level": es la intensidad mínima de esfuerzo establecido por la empresa para no ser despedido. Su valor se asigna desde la interfaz.

Las variables globales definidas en la programación del modelo son:

- "Number_of_new_workers": es el número de nuevos trabajadores que entran en cada periodo de la simulación.
- "Index_of_first_new_worker": es el número que identifica al primer nuevo trabajador que se crea en cada periodo de la simulación.
- "List_team_who": es una lista que incluye los trabajadores del equipo en cada periodo. El tamaño de la lista es el tamaño del equipo ("team-size") y su primer elemento es el trabajador que desempeña el papel de gestor del proyecto.
- "List_team_empty_jobs": es una lista que incluye tanto los trabajadores del equipo que no han sido despedidos como los puestos de trabajo que han quedado vacantes.
- "List_team_new_workers": es una lista que incluye tanto los trabajadores del equipo que no han sido despedidos como los nuevos trabajadores que han ocupado los puestos de trabajo que quedaron vacantes.
- "Worker-aux" y "job-aux": son variables auxiliares utilizadas para almacenar un trabajador o un puesto de trabajo según corresponda.

Las variables de estado de los "jobs" son:

- "Job-project-manager": es una variable *booleana* que indica si el puesto de trabajo lleva asociado la gestión del proyecto ("true") o no ("false"). Es necesario definir esta variable asociada al puesto de trabajo y no al trabajador, porque si éste es despedido, otro debe realizar esa función.

Las variables de estado de los "workers" son:

- "Behaviour": es el tipo de comportamiento frente al esfuerzo del trabajador. Es un *string* y toma su valor de las variables globales "population" o "new-employees".
- "Effort": es la intensidad del esfuerzo realizado. Toma un valor entre 0 y 100, y cambia a lo largo de la simulación dependiendo de la intensidad del esfuerzo del resto de miembros del equipo con los que interactúa el agente y de su comportamiento frente al esfuerzo. Su valor condiciona que el trabajador sea o no despedido por la empresa.
- "Project-manager": es una variable *booleana* que indica el rol que juega el trabajador en el proyecto. Este rol se asocia al puesto de trabajo ("job") que ocupa el trabajador. Si es el gestor del proyecto, será "true"; y si es un miembro, será "false".

El tamaño del *grid* no es relevante dado que la posición espacial de los agentes no influye en las interacciones ya que éstas no se realizan por vecindad sino con base en los vínculos de los agentes.

7.2.3 Proceso general y secuencia

Al principio de la simulación, se inicializa el modelo en el orden indicado:

1. Se crean y se inicializan los puestos de trabajo ("setup-jobs").
2. Se crean y se inicializan los trabajadores ("setup-workers").

3. Se crean los vínculos entre los trabajadores y sus puestos de trabajo ("link-workers-TO-jobs").
4. Se crea el equipo de trabajo y se crean los vínculos entre trabajadores ("create-team").
5. Se inicializan las variables relacionadas con la creación de nuevos trabajadores.

El tiempo trascurre de forma discreta por periodos. En cada periodo, las siguientes acciones son ejecutadas cíclicamente hasta el final de la simulación en el orden indicado.

1. La empresa despide a los trabajadores cuyo esfuerzo es menor que el nivel mínimo exigido ("involuntary-turnover-decision").
2. La empresa calcula las necesidades de nuevos trabajadores ("staffing-requirements").
3. Se crean y se inicializan los nuevos trabajadores ("setup-workers").
4. La empresa asigna los nuevos trabajadores a los puestos de trabajo vacíos ("new-workers-on-empty-jobs").
5. Los trabajadores actualizan la intensidad del esfuerzo como consecuencia de las interacciones entre los miembros del equipo ("update-effort").

7.2.4 Conceptos de diseño

- Principios básicos: Este modelo es una versión simplificada del modelo de rotación de personal propuesto por Posada *et al.* (2017).
- Emergencia: Como consecuencia de la interacción entre los trabajadores, emerge una cultura organizacional en relación con el esfuerzo.
- Adaptación: Los trabajadores adaptan la intensidad del esfuerzo en función de su tipo de comportamiento y del esfuerzo realizado por sus compañeros de equipo de trabajo.
- Objetivos: El objetivo de la empresa es que emerja una cultura corporativa de alto esfuerzo. No obstante, la empresa no ha sido

modelada como agente en este modelo. Los trabajadores no tienen ningún objetivo definido explícita ni implícitamente.
- Aprendizaje: No aprenden.
- Predicción: No hay predicción en el modelo.
- Detección: Los trabajadores detectan la intensidad del esfuerzo realizado por sus compañeros de equipo.
- Interacción: Los trabajadores interactúan mediante los vínculos establecidos, los cuales están condicionados por el estilo de liderazgo del equipo.
- Aleatoriedad: En la inicialización de la intensidad del esfuerzo que realizan los trabajadores.
- Colectivos: No hay organización de los agentes de más alto nivel que las tortugas.
- Observación: Se observa la emergencia de una cultura corporativa en relación con el esfuerzo.

7.2.5 Inicialización

Variables	Rango de valores
"Years"	[1-15] a través de la interfaz
"Team-size"	[3-10] a través de la interfaz
"Team-leadership"	Star o democratic a través de la interfaz
"Population", "new employees"	Null, high, shriking, averager, replicator, comparator
"Firing-level"	[1-100] a través de la interfaz
"Effort"	Aleatoriamente entre [0, 100]

El resto de variables se inicializan, por defecto, con el valor 0.

7.2.6 Datos de entrada

No hay.

7.2.7 Submodelos

En esta sección sólo incluimos la descripción del procedimiento. La programación de estos procedimientos (o en su defecto, el pseudocódigo) no se ha incluido ya que se detalla más adelante.

"Setup jobs": Se crean y se inicializan los puestos de trabajo asignando o no a cada puesto la responsabilidad de gestionar el proyecto y situándolos en el mundo en forma de estrella, de forma que el puesto con responsabilidad se sitúa en el origen (centro del mundo). Se representan mediante un círculo.

"Setup workers": Se crean y se inicializan los trabajadores asignándoles un tipo de comportamiento frente al esfuerzo (a elegir entre los siguientes: "null", "high", "shriking", "averager", "imitator", "replicator" y "comparator") y la intensidad del mismo. Se representan mediante una persona y su color depende de la intesidad del esfuerzo utilizando una escala cromática que va del azul (bajo esfuerzo) al rojo (alto esfuerzo).

"Link workers TO Jobs": Se crean los vínculos entre los trabajadores y sus puestos de trabajo. De esta forma, los trabajadores heredan las características de responsabilidad y posición en el mundo. Se representan mediante una línea dirigida del trabajador al puesto de trabajo.

"Create team": Se crea el equipo de trabajo creando una lista llamada "list-team-who" y se crean los vínculos entre trabajadores (que se representa mediante una línea no dirigida) en función del estilo de liderazgo del equipo, de forma que este procedimiento llama a otros dos procedimientos: "team-star" y "team-whole".

- El procedimiento "team-star" crea un vínculo entre el gestor del proyecto y cada uno de los miembros del equipo. Este procedimiento se ejecuta siempre que hay un vínculo entre el gestor de proyecto y los miembros de su equipo.

- El procedimiento "team-whole" crea vínculos entre los miembros del equipo. Este procedimiento se ejecuta si el estilo de liderazgo es democrático.

"Involuntary-turnover-decision": La empresa despide a los trabajadores cuyo esfuerzo es menor que el nivel mínimo exigido.

"Staffing-requirements": La empresa calcula las necesidades de nuevos trabajadores tras despedir a los trabajadores que no llegan al esfuerzo mínimo.

"New-workers-on-empty-jobs": La empresa asigna a los nuevos trabajadores los puestos de trabajo vacíos estableciendo un vínculo entre ellos, actualiza la lista "list-team-who" con los nuevos trabajadores y relaciona a los nuevos y antiguos trabajadores en función del estilo de liderazgo a través de los procedimientos: "links-new-&-old-workers-star" y "links-new-&-old-workers-whole".

- El procedimiento "links-new-&-old-workers-star" indica si el nuevo trabajador es el gestor o un miembro del equipo, y establece un vínculo, según corresponda, entre el nuevo gestor y los miembros de su equipo, o entre el nuevo miembro del equipo y el gestor del mismo.
- El procedimiento "links-new-&-old-workers-whole" establece vínculos entre todos los miembros del equipo.

"Update-effort": Los trabajadores actualizan la intensidad del esfuerzo como consecuencia de las interacciones entre los miembros del equipo y, dado que estas interacciones están condicionadas por el estilo de liderazgo, este procedimiento llama a otros dos procedimientos: "update-effort –star" y "update-effort –whole".

- El procedimiento "update-effort –star" establece combinaciones entre el gestor del proyecto y cada miembro del equipo y ambos cambian su esfuerzo llamando al procedimiento "change-effort". El procedimiento "update-effort–star" se

ejecuta siempre dado que siempre cuando hay un vínculo entre el gestor de proyecto y los miembros del equipo.

- El procedimiento "update-effort –whole" establece combinaciones entre dos miembros del equipo y ambos cambian su esfuerzo llamando al procedimiento "change-effort". El procedimiento "update-effort –whole" se ejecuta si el estilo de liderazgo es "democrático".
- El procedimiento "change-effort" incluye las reglas para el cambio de la intensidad del esfuerzo en relación con el tipo de comportamiento de los trabajadores, a saber:
  - Tipo "null": realiza un bajo esfuerzo (entre 0 y 10) con independencia del esfuerzo que realice su compañero.
  - Tipo "high": realiza un bajo esfuerzo (entre 80 y 100) con independencia del esfuerzo que realice su compañero.
  - Tipo "shrinking": realiza la mitad del esfuerzo de su último compañero.
  - Tipo "replicator": realiza el mismo esfuerzo que su último compañero.
  - Tipo "effort comparator": aumenta un 10 % su esfuerzo si su esfuerzo fue menor que el de su último compañero y lo disminuye en la misma proporción si fue mayor.
  - Tipo "averager": realiza un esfuerzo medio entre su esfuerzo y el esfuerzo de su último compañero.

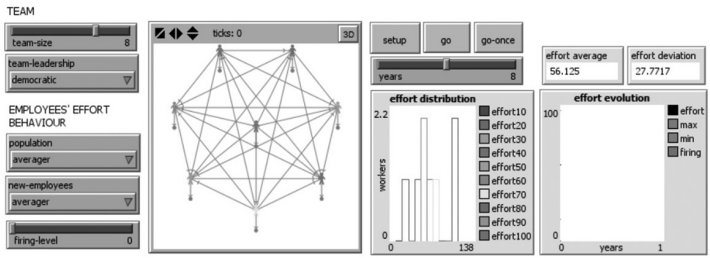
7.3 El modelo Netlogo de rotación involuntaria de personal

7.3.1 La interfaz

La programación de modelo de intercambio del mercado se encuentra disponible en www.eii.uva.es/posada/manualdenetlogo-capVII.html. En la pestaña "Ejecutar" se encuentra la interfaz del programa (Figura 7.2). Ésta incluye:

- Un deslizador ("years") para el número de periodos de la simulación.
- Un deslizador ("team-size") para establecer el tamaño del equipo.
- Una lista desplegable ("team-leadership") que incluye los dos tipos extremos de liderazgo de equipos: autoritario ("star") o democrático ("democratic").
- Una lista desplegable ("population") que incluye distintos tipos de comportamiento frente al esfuerzo: "null", "high", "averager", "shrinking", "comparator" y "replicator".
- Una lista desplegable ("new-employees") que incluye distintos tipos de comportamiento frente al esfuerzo de los nuevos trabajadores que entran en la empresa: "null", "high", "averager", "shrinking", "comparator" y "replicator".
- Un deslizador ("firing.level") para establecer el esfuerzo mínimo que la empresa exige a sus trabajadores para seguir en ella.

Figura 7.2
Interfaz del modelo "Intercambio en un mercado"



El funcionamiento del programa se controla con los siguientes botones:

- El botón "Set up" inicializa un nuevo mundo y representa el equipo de trabajo. Aunque situamos a empleados en parcelas (blancas), su posición en el mundo no es relevante, ya que la interacción no se produce por vecindad sino por los vínculos que existen entre los agentes.
- Existe un botón nombrado "Go once", que permite ir avanzando la simulación de periodo en periodo, Mientras que el botón "Go" avanza todos los periodos de la simulación. Se representan

gráficamente la evolución de los esfuerzos medio, mínimo y máximo del periodo, así como el porcentaje de rotación involuntaria de personal.

Además, el modelo incluye las siguientes salidas:

- Un monitor del valor medio de los esfuerzos realizados por los miembros del equipo de trabajo que se indica mediante la instrucción "precision mean [effort] of workers 4".
- Un monitor de la desviación típica de los esfuerzos realizados por los miembros del equipo de trabajo que se indica mediante la instrucción "precision deviation [effort] of workers 4".
- Un gráfica llamada "effort evolution" que monitoriza el valor medio del esfuerzo a lo largo de la simulación mediante la instrucción "plot precision mean [effort] of workers 4", el mínimo y el máximo esfuerzos realizados por los trabajadores mediante las instrucciones "plot precision max [effort] of workers 4" y "plot precision min [effort] of workers 4", respectivamente, así como el esfuerzo mínimo exigido por la empresa para no ser despedido, mediante la instrucción "plot firing-level".
- Una gráfica llamada "effort distribution" que monitoriza la distribución percentil de esfuerzos entre 0 y 100 mediante un histograma cuyos colores se asocian a los colores establecidos para cada rango decimal de esfuerzos según una escala cromática que va del azul (mínimo esfuerzo) al rojo (máximo esfuerzo). Para cada rango decimal, se utiliza la instrucción "plotxy", siendo la coordenada *x* el valor máximo del rango de esfuerzo y la coordenada *y* el número de trabajadores en el rango de esfuerzo. Por ejemplo, para un esfuerzo en el rango [0,10], la instrucción asociada al trazo es:

```
plotxy 10 count workers with [(effort >= 0) and (effort <= 10)]
```

7.3.2 La definición de variables

Los distintos tipos de variables se definen en la pestaña "Código". Al inicio del programa se definen los dos tipos de tortugas de la siguiente manera:

```
breed [workers worker]
breed [jobs job]
```

A continuación se definen las variables globales:

```
globals
[
        number_of_new_workers
        index_of_first_new_worker
        list_team_who
        list_worker&empty_job
        list_job&new_worker
        worker_aux
        job_aux
]
```

Luego se definen las variables de cada tipo de agente:

```
workers-own
[
        behaviour
        effort
        project_manager
]
```

```
jobs-own
[
        job_project_manager
]
```

Tras la definición de todas las variables, escribimos el código de programación asociado a los botones "Setup" y "Go".

7.3.3 El código de programación para inicializar el modelo

El código de programación del procedimiento "setup" (inicializar) es:

```
to setup
clear-all
;; random-seed 681279294
ask patches [set pcolor white]
setup-jobs                                  ;; create jobs (job 0, ....job n-1)
setup-workers team-size population          ;; create workers: worker n, ....worker m
link-workers-TO-jobs                        ;; assign jobs to workers
create-team                                 ;; create team of workers
set number_of_new_workers 0                 ;; Reset the tick counter to zero
set index_of_first_new_worker 2* team-size
reset-ticks
end
```

Al principio del código aparece el comando Netlogo "clear-all", el cual borra las tortugas de ejecuciones anteriores y pone en *default* las parcelas. Este comando es muy útil para la realización de varios experimentos.

La instrucción "; random-seed 681279294" se ha utilizado durante la programación del modelo para generar siempre el mismo entorno y depurar los errores de programación. Sin embargo, para correr simulaciones diferentes se pone un punto y coma delante de la instrucción.

Insertamos los siguientes comandos: "ask patches [set pcolor white]" para que el color de las parcelas sea blanco con fines de visibilidad en el libro en blanco y negro.

Luego se llama al procedimiento "setup jobs" que crea y distribuye en el mundo los puestos de trabajo necesarios para la ejecución del proyecto mediante las siguientes instrucciones:

```
to setup-jobs    ;; create jobs: job 0, job 2,...job n-1
create-jobs team-size
        [ set shape "circle"
```

```
        set color gray
        ]
;;--------------------star configuration--------------
ask job 0 [set job_project_manager 1]
let i 1
while [i < team-size]
        [ask job 0 [create-link-to job i]
        set i i + 1
        ]
layout-radial jobs links (job 0)
ask links [hide-link] ; make invisible links between jobs
end
```

Analizando con detalle el código de programación, observamos que, en primer lugar, se crean tantos puestos de trabajo ("jobs") como el tamaño de equipo ("team-size"). Además, se indica que sean círculos de color gris.

```
create-jobs team-size
        [ set shape "circle"
        set color gray
        ]
```

A continuación se indica que el primer puesto de trabajo que se genera corresponde al puesto de gestión del proyecto. Por defecto, se sitúa en el centro del mundo.

```
ask job 0 [set job_project_manager 1]
```

El puesto de trabajo del gestor del proyecto se une mediante un vínculo con el resto de puestos del equipo de la siguiente manera:

```
let i 1
while [i < team-size]
        [ask job 0 [create-link-to job i]
        set i i + 1
        ]
```

Tenga en cuenta que debe unir los puestos de trabajo y no los trabajadores que ocupan dichos puestos ya que se trata de simular las relaciones formales establecidas en la empresa. Si lo que se unen son los trabajadores, se estarían simulando las relaciones informales de la empresa.

A continuación, se indica que los puestos de trabajo se sitúan de forma radial alrededor del puesto de gestión del proyecto mediante la siguiente instrucción:

```
layout-radial jobs links (job 0)
```

Tras ello se llama al procedimiento "setup-workers", que indica el número de trabajadores que se crean y su comportamiento frente al esfuerzo. Mediante la instrucción "setup-workers team-size population" se llama al citado procedimiento y se indica que se generan tantos trabajadores como el tamaño del equipo ("team-size") y su comportamiento frente al esfuerzo ("population").

Dentro de la programación del procedimiento "setup-workers", mediante la instrucción "set-default-shape workers "person"" se indica que los trabajadores se representan como personas y se crean (definiendo su comportamiento frente al esfuerzo, la cuantía del esfuerzo y su color en función de dicha cuantía).

```
to setup-workers [number effort-behavior]
set-default-shape workers "person"
        create-workers number
        [ set behaviour effort-behavior
        define-effort
        define-color
        ]
end
```

El esfuerzo que realizan se define mediante el procedimiento "define-effort" en función de su comportamiento de la siguiente forma: si realiza un esfuerzo nulo se genera un número aleatorio entre 0 y 10 (mediante la instrucción "set effort random 11"), si realiza un esfuerzo alto se

genera un esfuerzo aleatorio entre 81 y 100 (mediante la instrucción "set effort random 20 + 81", por la que se genera un número aleatorio entre 0 y 19 al que se suma 81) y en el resto de casos, se genera un esfuerzo aleatorio entre 0 y 100 (mediante la instrucción "set effort random 101").

```
to define-effort
        [set effort random 10 ]
        [ifelse behaviour = "high effort"
                [set effort random 20 + 81]
                [set effort random 101]
        ]
end
```

El valor del esfuerzo está asociado a un color de la forma que se indica en el procedimiento "define color", que es como sigue: azul para esfuerzos bajos entre [0,10], azul-cielo para esfuerzos entre [11,20], azul-cian para esfuerzos entre [21,30], azul-turquesa para esfuerzos entre [31,40], verde-lima para esfuerzos entre [41,50], verde para esfuerzos entre [51,60], amarillo para esfuerzos entre [61,70], marrón para esfuerzos entre [71,80], naranja para esfuerzos entre [81,90] y roja para esfuerzos altos entre [91,100].

```
to define-color    ;;-----------------------------color depending on the effort
ifelse effort < 11
        [set color 105]    ;; blue
                [ifelse effort < 21
                    [set color 95]    ;; sky
                [ifelse effort < 31
                    [set color 85]    ;; cyan
                [ifelse effort < 41
                    [set color 75]    ;; turquoise
                [ifelse effort < 51
                    [set color 65]    ;; lime
                [ifelse effort < 61
                    [set color 55]    ;; green
                [ifelse effort < 71
                    [set color 45]    ;; yellow
                [ifelse effort < 81
                    [set color 35]    ;; brown
```

```
            [ifelse effort < 91
                [set color 25]    ;; orange
            [set color 15] ;red
        ]]]]]]]]
end
```

Después de crear los puestos de trabajo y los trabajadores, se asigna un trabajador a cada puesto de trabajo mediante el procedimiento "link-workers-to-jobs", de forma que los trabajadores heredan su posición en el mundo y las características de gestión del puesto.

```
to link-workers-TO-jobs
let i 0
let j team-size
while [i < team-size]
        [ask worker j [set project_manager [job_project_manager] of job i
                setxy ([xcor] of job i ) ( 5+ [ycor] of job i )
                create-link-to job i
        ]
        set i i + 1
        set j j + 1]
end
```

Finalmente, se crea el equipo con el procedimiento "create-team". En primer lugar, se crea una lista de trabajadores que inicialmente está vacía. Mediante la instrucción "sentence" ("ask workers with [project_manager = 0][set list_team_who sentence who list_team_who]") se añaden a la lista todos los trabajadores del equipo, con excepción del gestor del proyecto, el cual se añade como el primer elemento de la lista mediante la instrucción "fput" ("ask workers with [project_manager = 1] [set list_team_who fput who list_team_who]").

```
to create-team
        set list_of_team_who_list []
        ask workers with [project_manager = 0][set list_team_who sentence who
        list_team_who]
        ask workers with [project_manager = 1] [set list_team_who fput who
        list_team_who]
```

```
        team-star
        if team-leadership = "democratic" [team-whole];
end
```

A continuación se llama al procedimiento "team-star", que se ejecuta siempre ya que consideramos que siempre hay interacción entre el gestor del proyecto y los miembros del equipo. Si el estilo de liderazgo del equipo fomenta la interrelación entre los miembros del equipo, se llama además al procedimiento "team-whole".

En el procedimiento "team-star", el vínculo que se crea es dirigido del gestor de proyecto, que es el primero de la lista ("item 0 list_team_who"), a los miembros del equipo ("item j list_team_who").

```
to team-star
let i 0
While [i < team-size] ;go over the list of workers in teams
        [let j i + 1
        while [j < team-size]
                [ask worker item 0 list_team_who [create-link-to worker item j
                list_team_who]
                set j j + 1]
        set i i + 1]
end
```

En el procedimiento "team-whole" se establece un vínculo entre todos los miembros del equipo. Suponemos que cuando un trabajador se marcha, se crea inmediatamente un nuevo vínculo entre el nuevo trabajador y el resto de miembros del equipo.

```
to team-whole
let i 1
While [i < team-size] ;go over the list of workers in teams
        [let j i + 1
        while [j < team-size]
                [ask worker item i list_team_who [create-link-to worker item j
                list_team_who]
                set j j + 1]
        set i i + 1]
end
```

Finalmente se inicializan las variables necesarias para crear nuevos trabajadores de la siguiente forma:

```
set number_of_new_workers 0
set index_of_first_new_worker 2 * team-size
```

7.3.4 El código de programación para correr el modelo

El botón "Go" llama al procedimiento "go-once".

```
to go    ;;--------------------------
        while [ticks < years] [go-once]
end
```

El procedimiento "go-once" llama a una serie de procedimientos que detallaremos a continuación.

```
to go-once    ;;--------------------------step by step
ifelse ticks = years
[stop]
        [
        involuntary-turnover-decision
        staffing-requirements
        setup-workers number_of_new_workers new-employees ; to create new employees
        new-workers-on-empty-jobs
        update-effort
        tick
        ]
end
```

El procedimiento "involuntary-turnover-decision" sustituye en la lista del equipo de trabajo ("list-team-who") a aquellos trabajadores cuyo esfuerzo es inferior al mínimo exigido por la empresa ("ask workers with [effort < firing-level]") por los puestos de trabajos que ocupan y

finalmente se eliminan de la simulación (mediante la instrucción "die").

```
to involuntary-turnover-decision
      set list_team_empty_jobs list_team_who
      ask workers with [effort < firing-level]
            [set worker_aux who                          ;; who is going to die?
            ask out-link-neighbors [ if breed =
            jobs [set job_aux who]]
      let i 0
      while [i < length list_team_empty_jobs ]
            [if worker_aux = item i list_team_           ;; replace worker_aux by
            empty_jobs [set list_team_empty_jobs         job_aux
            replace-item i list_team_empty_jobs
            job_aux]
            set i i + 1]
      die]
end
```

El procedimiento "staffing-requirements" calcula cuántos nuevos empleados se necesitan crear mediante las siguientes instrucciones:

```
to staffing-requirements
      set index_of_first_new_worker (index_of_first_new_worker +
      number_of_new_workers)
      set number_of_new_workers team-size - count workers
end
```

Para ello, en primer lugar se calcula el número que identifica al primer nuevo empleado creado por Netlogo en cada interacción mediante la siguiente instrucción: "set index_of_first_new_worker (index_of_first_new_worker + number_of_new_workers)".

Y a continuación, se calcula el número de nuevos empleados (como la diferencia entre el tamaño del equipo y los empleados que todavía siguen en la empresa) mediante la siguiente instrucción: "set number_of_new_workers team-size - count workers".

Tras calcular las necesidades, se llama al procedimiento "setup workers" descrita en la subsección anterior, pero en este caso los argumentos son el número de nuevos trabajadores ("number_of_new_workers") y

el tipo de comportamiento frente al esfuerzo de los nuevos trabajadores ("new-employees").

A continuación, mediante el procedimiento "new-workers-on-empty-jobs", se asignan los nuevos trabajadores a los puestos de trabajo vacíos. Esto se hace en tres pasos.

```
to new-workers-on-empty-jobs
let i 0
set worker_aux index_of_first_new_worker
set list_team_new_workers list_team_empty_jobs ;inizializate replacement
;;----------
while [i < team-size ]
      [ask job i
            [ let choice one-of (workers with [out-link-neighbor? myself])
                  if choice = nobody ;; if job is empty, create a new link with a new
                  worker
                  [create-link-from worker worker_aux ;create link job-worker
                  ask worker worker_aux
                  [
                        set project_manager [job_project_manager] of job i
                        setxy ([xcor] of job i ) (5 + [ycor] of job i )
                  ]
            ;;-----replace empty JOB by NEW WORKER
            let j 0
            while [j < length list_team_new_workers] ;go over the list of list
                  [if i = item j list_team_new_workers [set list_team_new_workers
                  replace-item j list_team_new_workers worker_aux] ;replace job i by
                  worker worker_aux
                        set j j + 1]
                        set list_team_who list_team_new_workers
                        set worker_aux worker_aux + 1
                  ]]
      set i i + 1]
      ;;-----NEW LINKS
            ifelse team-leadership = "ego-centered" [links-new-&-old-workers-star]
            [links-new-&-old-workers-whole]
end
```

El primer paso es crear un vínculo entre cada puesto de trabajo vacío y el nuevo trabajador para que él herede las características del puesto de trabajo (posición en el mundo y papel en el equipo). Para ello utilizamos la siguiente secuencia de instrucciones:

```
let i 0
set worker_aux index_of_first_new_worker
set list_team_new_workers list_team_empty_jobs ;inizializate replacement
;;----------
while [i < team-size ]
      [ask job i
            [ let choice one-of (workers with [out-link-neighbor? myself])
            if choice = nobody ;; if job is empty, create a new link with a new worker
                  [create-link-from worker worker_aux ;create link job-worker
                  ask worker worker_aux
                        [
                        set project_manager [job_project_manager] of job i
                        setxy ([xcor] of job i ) (5 + [ycor] of job i )
                        ]
```

El segundo paso es sustituir en la lista ("list_team_new_workers") los puestos de trabajo vacíos por el nuevo trabajador que los ocupa y, tras finalizar este proceso, la lista "list_team_who" se actualiza con la lista generada. Para ello utilizamos la siguiente secuencia de instrucciones:

```
            let j 0
                  while [j < length list_team_new_workers]    ;; go over the list of list
                        [if i = item j list_team_new_workers [set list_team_new_workers
                        replace-item j list_team_new_workers worker_aux]    ;; replace job i
                        by worker worker_aux
                        set j j + 1]
            set list_team_who list_team_new_workers
            set worker_aux worker_aux + 1
      ]]
set i i + 1]
```

El tercer paso consiste en establecer los vínculos entre los nuevos empleados y los antiguos empleados. Para ello se llama a los procedimientos "links-new-&-old-workers-star" y "links-new-&-old-workers-whole", según corresponda con el estilo de liderazgo, autoritario o democrático, respectivamente. Ambos procedimientos tienen una programación similar.

```
ifelse team-leadership = "ego-centered" [links-new-&-old-workers-star]
[links-new-&-old-workers-whole]
```

Si el estilo de liderazgo es autoritario, tienen lugar dos situaciones:

- Si el nuevo empleado es el gestor del proyecto (que es el primer elemento (ítem 0) de la lista "list_team_new_workers"), crea vínculos con todos los miembros del equipo (son los elementos situados a su derecha en la lista).

```
[ask worker k [create-link-to worker item r list_team_new_workers]
```

- Si el nuevo empleado es sólo un miembro del equipo, crea un vínculo con el gestor del proyecto:

```
[ask worker k [create-link-from worker item 0 list_team_new_workers]
```

```
to links-new-&-old-workers-star
let k index_of_first_new_worker
while [k <= worker_aux] ; for the new workers who are numerated from k to
worker-aux
      [let i 0
            while [i < length list_team_new_workers]
                  [if k = item i list_team_new_workers ;get the position of the new worker k
                        [if i = 0
                        [let r i + 1;-----get the other workers on the right of the list
            while [r < length list_team_new_workers] ;go over the team on the right
                  [ask worker k [create-link-to worker item r list_team_new_workers]
                        set r r + 1]]
                        if i > 0
                  [ask worker k [create-link-from worker item 0 list_team_new_workers]
                        ]
                  ]
      set i i + 1]
set k k + 1]
end
```

Si el estilo de liderazgo es democrático, tienen lugar las mismas dos situaciones anteriores:

- Si el nuevo empleado es el gestor del proyecto (que es el primer elemento (ítem 0) de la lista "list_team_new_workers"), crea vínculos con todos los miembros del equipo (son los elementos situados a su derecha en la lista).

```
ask worker k [create-link-to worker item r list_team_new_workers]
```

- Si el nuevo empleado es sólo un miembro del equipo, crea vínculos con todos los demás miembros del equipo. Para ello se determina la posición del trabajador en la lista y crea vínculos, primero con los trabajadores que se encuentran a la derecha de su posición en la lista; y segundo, el gestor del pro con los trabajadores que se encuentran a la derecha de su posición en la lista:

```
[ask worker k [create-link-from worker item 0 list_team_new_workers]
```

```
to links-new-&-old-workers-whole
let k index_of_first_new_worker
while [k <= worker_aux] ; for the new workers who are numerated from k to c
      [let i 0
            while [i < length list_team_new_workers] ;go over the list of list:
            list_team_new_workers
                  [if k = item i list_team_new_workers ;get the position of the new worker k
                        [let r i + 1
            while [r < length list_team_new_workers] ;go over the team on the right
                  [ask worker k [create-link-to worker item r list_team_new_workers]
                        set r r + 1]
                        if i > 0
                  [let l i - 1
            while [l >= 0]
                  [ask worker k [create-link-from worker item l list_team_new_workers]
                  set l l - 1]
                  ]
            ]
      set i i + 1]
set k k + 1]
end
```

El último procedimiento llamado por el "go-once" es "update-effort". Éste actualiza los esfuerzos que realizan los trabajadores como consecuencia de su interacción con los otros miembros del equipo con los que tienen vínculos, y llama a dos procedimientos más, que son: "update-effort-star" y "update-effort-whole".

```
to update-effort
      update-effort-star
      if team-leadership = "democratic" [update-effort-whole]
end
```

El procedimiento "update-effort-star" se ejecuta siempre ya que, independientemente del estilo de liderazgo del equipo, el gestor tiene vínculos con todos los miembros del equipo.

Si el estilo es democrático, los esfuerzos de los miembros del equipo cambian también por la interacción con sus compañeros de equipo y, en ese caso, se llama al procedimiento "update-effort-whole" mediante una sentencia "if":

```
if team-leadership = "democratic" [update-effort-whole]
```

```
to update-effort-star
let j 1
while [j < length list_team_who]
      [ask worker item 0 list_team_who
            [ask worker item j list_team_who [change-effort [effort] of worker item j
            list_team_who [effort] of worker item 0 list_team_who]
            change-effort [effort] of worker item 0 list_team_who [effort] of worker
            item j list_team_who]
      set j j + 1]
end
```

```
to update-effort-whole
let j 1
while [j < length list_team_who]
      [ask worker item j list_team_who
            [let r j + 1    ;;-----get the other workers on the right of the list (in order to
            create links to them)
            while [r < length list_team_who] ;go over the team on the right
                  [ask worker item r list_team_who
                        [change-effort [effort] of worker item r list_team_who [effort] of
                        worker item j list_team_who]
                  change-effort [effort] of worker item j list_team_who [effort] of worker
                  item r list_team_who
            set r r + 1]]
      set j j + 1]
end
```

En ambos casos se llama al procedimiento "change-effort", que actualiza la intensidad del esfuerzo en función del tipo de comportamiento, indicando en los argumentos en primer lugar el propio esfuerzo y en segundo lugar el esfuerzo del compañero. Se han programado los siguientes comportamientos:

- Tipo "shrinking": realizan la mitad del esfuerzo de su último compañero.
- Tipo "replicator": realizan el mismo esfuerzo que su último compañero
- Tipo "effort comparator": aumentan un 10 % su esfuerzo si su esfuerzo fue menor que el de su último compañero y lo disminuyen en la misma proporción si fue mayor.
- Tipo "averager": realizan un esfuerzo medio entre su esfuerzo y el esfuerzo de su último compañero.

Los comportamientos tipo "null" y "high" no se han incluido en este procedimiento porque los trabajadores tipo "null" realizan un bajo esfuerzo (entre 0 y 10) con independencia del esfuerzo que realice su compañero, y los trabajadores tipo "high" realizan un bajo esfuerzo (entre 0 y 10) independientemente del esfuerzo que realice su compañero.

Al final del procedimiento, se ha limitado al esfuerzo al 100 % y se llama al procedimiento "define color" explicado en la subsección anterior para cambiar el color del trabajadores en función de la intensidad del esfuerzo realizado.

```
to change-effort [own partner]
if behaviour = "shrinking" [set effort partner / 2]
if behaviour = "replicator" [set effort partner]
if behaviour = "effort comparator" [
ifelse own < partner [set effort own * 1.1][set effort own * 0.9]]
if behaviour = "averager" [set effort (partner + own) / 2] if effort > 100 [set effort 100]
define-color    ;;----change the color according to new effort value
end
```

7.4 Experimentos con el modelo

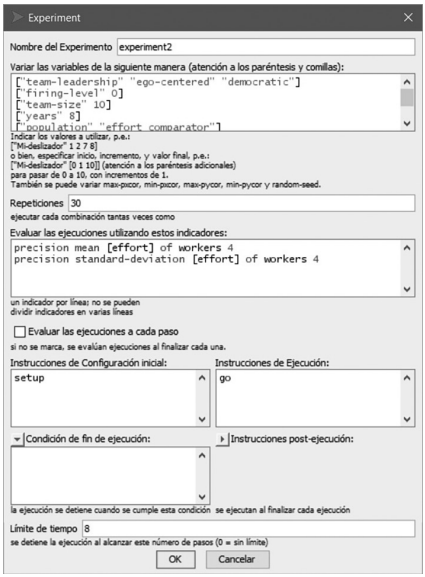
Para hacer un análisis de sensibilidad de los parámetros del modelo, sería necesario analizar 5 760 escenarios y repetir cada escenario como mínimo 30 veces para realizar inferencia estadística. Por ello, mostraremos solamente algunos resultados y animamos al lector a explorar el resto de posibilidades.

7.4.1 Escenarios básicos: poblaciones homogéneas sin rotación de personal

Hay 96 escenarios básicos que son el resultado de combinar 6 poblaciones homogéneas de tipo de comportamiento ("population=null", "high", "shrinking", "comparator", "averager", "replicator") por 2 estilos de liderazgo ("team-leadership= ego-centered", "democratic") por 8 tamaños de equipo ("team-size= 3", 4, 5, … hasta 10 trabajadores) por 1 umbral de esfuerzo mínimo requerido ("firing level =0").

En la Figura 7.3 se muestra la configuración del analizador de comportamiento. Dado que la condición de parada es el límite de tiempo, no olvide indicar el número de interacciones.

Figura 7.3



Condiciones de las simulaciones para población homogénea sin rotación involuntaria de trabajadores del tipo "effort comparator" en un equipo de trabajo de tamaño 10 y evaluando los dos estilos de liderazgo "ego.centered" y "democratic".

De la experimentación se han obtenido las siguientes conclusiones:

Población homogénea de Shrinking: emerge una cultura corporativa de bajo esfuerzo con independencia del tamaño y del estilo de liderazgo (Figura 7.4 y Figura 7.5). Se observa una sola barra.

Figura 7.4

Shrinking –star

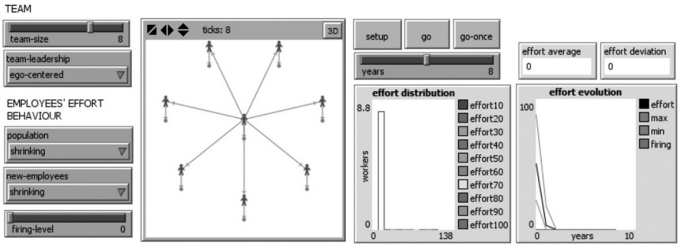
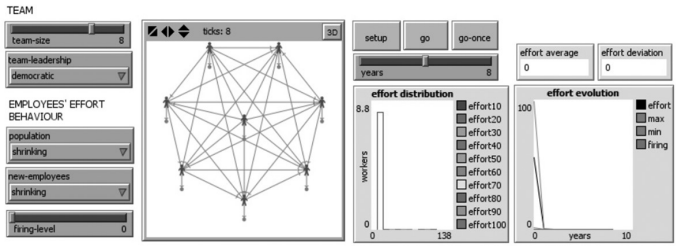


Figura 7.5

Shrinking –democratic



Población homogénea de averager: emerge una cultura corporativa alrededor del valor medio, pero no siempre es la misma. Comparando la Figura 7.5 con la 7.6, observamos que en ambos casos converge a un esfuerzo medio cercano a 50. El estilo democrático hace que la cultura emerja antes, y eso se refleja en una desviación típica menor.

Figura 7.6

Averager –star

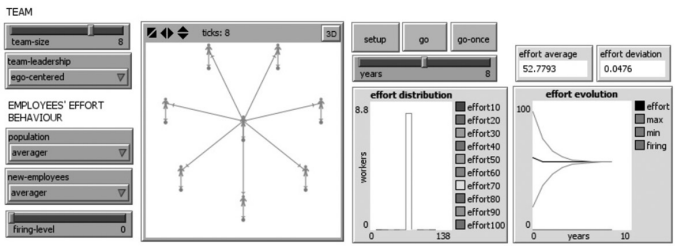
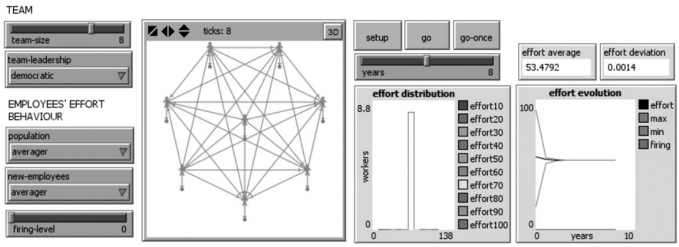


Figura 7.7

Averager –democratic



Población homogénea de replicator: Emerge una cultura corporativa, pero no es única (Figura 7.8 y Figura 7.9).

Figura 7.8

Replicator –star

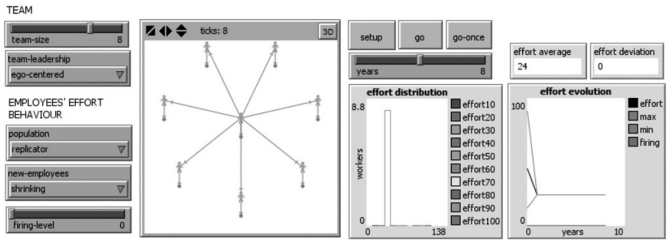
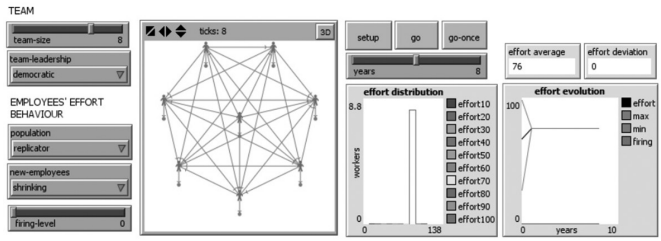


Figura 7.9

Replicator – democratic



Población homogénea de replicator: Observamos que no emerge una cultura corporativa dado que la gráfica "effort-distribution" presenta varios diagramas de barras en los dos estilos de liderazgo (Figura 7.10 y Figura 7.11).

Figura 7.10

Comparator –star

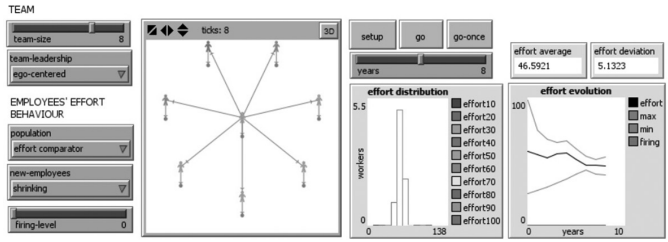
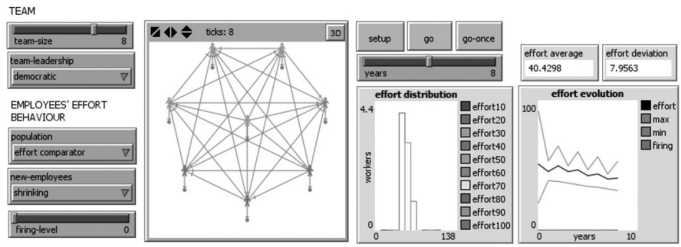


Figura 7.11

Comparator –democratic



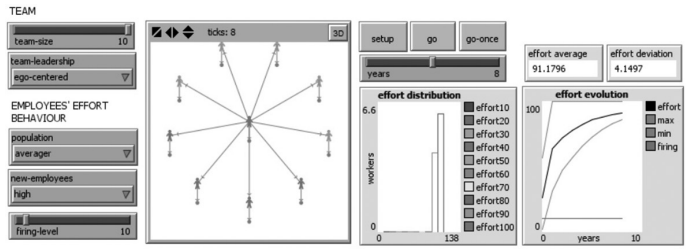
7.4.2 Poblaciones heterogéneas

Hay 5 664 escenarios como resultado de combinar los 36 tipos de escenarios heterogéneos (que a su vez son el resultado de combinar los 6 tipos de poblaciones iniciales con los 6 tipos de poblaciones para los nuevos empleados) por 2 estilos de liderazgo ("team-leadership= ego-centered", "democratic") por 8 tamaños de equipo ("team-size= 3", 4, 5, … hasta 10 trabajadores) por 10 umbrales de esfuerzo mínimo requerido ("firing level =0, 10, 20, 30, 40 … hasta 90).

La emergencia se reduce según aumenta la rotación del personal y la heterogeneidad (comparar la Figura 7.12 con la Figura 7.6).

Figura 7.12

Averager-high for ego-centered leadership y esfuerzo mínimo requerido del 10 %



BIBLIOGRAFÍA

Ashworth, M.J., y K.M. Carley (2006). "Can tools help unify organization theory? Perspectives on the state of computational modeling", *Computational and Mathematical Organization Theory*, vol. 13, núm. 1, pp. 89-111.

Carley, K.M. y L. Gasser (1999). "Computational organization theory", en G. Weiss (ed.), *Multiagent systems: A modern approach to distributed artificial intelligence*, Cambridge: MIT Press, pp. 299-330.

Carley, K.M. y M.J. Prietula (2014). *Computational organization theory*. Nueva York: Psychology Press.

Cioffi-Revilla, C. (2014). *Introduction to computational social science: Principles and applications*, Londres: Springer-Verlag.

Cliff, D. y J. Bruten (1997). *Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets*. Bristol: HP Laboratories.

Cohen, M.D., J.G. March y J.P. Olsen, (1972). "A garbage can model of organizational choice", *Administrative Science Quarterly*, vol. 17, núm. 1, pp. 1-25.

Coleman, J.S. (1964). "Mathematical models and computer simulation", en R.E.L. Faris (ed.), *Handbook of Modern Sociology*. Chicago: Rand McNally and Company, pp. 1027-1062.

Coleman, J.S. (1962). "Analysis of social structures and simulation of social processes with electronic computers", en H. Guetzkow (ed.), *Simulation in Social Science*. Englewood Cliffss: Prentice Hall, pp. 63-69.

Dal Forno, A. y U. Merlone (2002). "A multi-agent simulation platform for modeling perfectly rational and boundedrational agents in organizations", *Journal of Artificial Societies and Social Simulation*, vol. 5, núm. 2, pp. 1460-7425.

Davis, J.P., K.M. Eisenhardt y C.B. Bingham (2007). "Developing theory through simulation methods", *Academy of Management Review*, 32, pp. 480-499.

Drogoul, A. y J. Ferber (1993). "From Tom Thumb to the dockers: Some experiments with foraging robots", en J. Meyer, H.L. Roiblat y S. W. Wilson (eds.), *From Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. Cambridge: The MIT Press, pp. 451-459.

Epstein, J.M. (2013). *Agent-Zero: Toward neurocognitive foundations for generative social science*. Princeton: Princeton University Press.

Epstein, J.M. (2006). *Generative social science: Studies in agent-based computational modeling*. Princeton: Princeton University Press.

Epstein, J.M. (1999). "Agent-based computational models and generative social science", *Complexity*, vol. 4, núm. 5, pp. 41-60.

Epstein, J.M. y R. Axtell (1996). *Growing artificial societies: Social science from the bottom up*. Washington: Brookings Institution Press.

Escobar, D. (2001). *Economía matemática*. Bogotá: Alfaomega.

Etkin, J. y L. Schvarstein (1989). *Identidad de las organizaciones. Invariancia y cambio*. Buenos Aires: Paidós.

Fararo, T.J. (1969). "The nature of mathematical sociology: A non-technical essay", *Social Research*, vol. 36, núm. 1, pp. 75-92.

Ferber, J. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*. Harlow: Addison Wesley Logman.

García-Valdecasas, J.I. (2016). *Simulación basada en agentes. Introducción a NetLogo.* España: Centro de Investigaciones Sociológicas.

Gilbert, N. (2008). *Agent-Based Models*. Los Angeles: Sage Publications.

Gode, D. y S. Sunder (1993). "Allocative Efficiency of Market with Zero-Intelligent Traders: Market as a Partial Substitute for Individual Rationality", *Journal of Political Economy*, vol. 101, núm. 1, pp. 119-137.

Grimm, V., U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. Heinz, G. Huse, A. Huth, J.U. Jepsen, C. Jørgensen, W.M. Mooij, B. Müller, G. Pe'er, C. Piou, S.F. Railsback, A.M. Robbins, M.M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Souissi, R.A. Stillman, R. Vabø, U. Visser, y D.L. DeAngelis

(2006). "A standard protocol for describing individual-based and agent-based models", *Ecological Modelling*, vol. 198, pp. 115-126.

Grimm, V., U. Bergerb, D.L. DeAngelis, G. Polhill, J. Giskee y S.F. Railsbackf (2010). "The ODD protocol: A review and first update", *Ecological Modelling*, vol. 221, pp. 2760-2768.

Hamill, L. y N. Gilbert (2016). *Agent-based modelling in economics*. Chichester: John Wiley & Sons.

Harrison, J.R., Z. Lin, G.R. Carroll y K.M. Carley (2007). "Simulation in organizational and management research", *The Academy of Management Review*, vol. 32, núm. 4, pp. 1229-1245.

Huhns, M.N. y M.P. Singh (1998). *Readings in agents*. San Francisco: Morgan Kaufmann Publisher.

Kendrick, D., P.R. Mercado y H.M. Amman (2006). *Computational economics*. Princeton: Princeton University Press.

Lindemann, G., D. Moldt y M. Paolucci (eds.) (2004). *Regulated Agent-Based Social Systems*. Berlín: Springer-Verlag.

Lomi, A. y E.R. Larsen (eds.) (2001). *Dynamics of organizations: Computational modeling and organization theories*. Menlo Park: AAAI Press/MIT Press.

Manzo, G. (2010). "Analytical sociology and its critics", *European Journal of sociology*, vol. 51, núm. 1, pp. 129-170.

Masuch, M. y P. LaPotin (1989). "Beyond garbage cans: An AI model of organizational choice", *Administrative Science Quarterly*, vol. 34, núm. 1, pp. 38-67.

Miller, K.D. (2015). "Agent-based modeling and organization studies: A critical realist perspective", *Organization Studies*, vol. 36, núm. 2, pp. 175-196.

Minar, N., R. Burkhart, C. Langton y M. Askenazi (1996). http://www.santafe.edu/projects/swarm/overview/overview.htm

Posada, M. y A. Lopez-Paredes, (2008) "How to choose the bidding strategy in continuous double auctions: Imitation versus take-the-best heuristics". *Journal of Artificial Societies and Social Simulation* 11(1)6, disponible en http://jasss.soc.surrey.ac.uk/11/1/6.html

Posada, M., C. Martín-Sierra y E. Perez (2017) "Effort, Satisfaction and Outcomes in Organisations", *Journal of Artificial Societies*

and Social Simulation 20(2)9, disponible en http://jasss.soc.surrey.ac.uk/20/2/9.html

Poteete, A.R, M.A. Janssen y E. Ostrom (2012). *Trabajar juntos: acción colectiva, bienes comunes y múltiples métodos en la práctica*. México: UNAM.

Prietula, M. J. (2011). "Thoughts on complexity and computational models", en P. Allen, S. Maguire y B. McKelvey (eds.), *The Sage handbook of complexity and management*. Londres: SAGE Publications, pp. 93-110.

Prietula, M.J., K.M. Carley y L. Gasser (1998) (eds.). *Simulating organizations: Computational models of institutions and groups*. Cambridge: MIT Press.

Railsback, S.F. y V. Grimm (2012). *Agent-Based and Individual-Based Modeling*. Princeton: Princeton University Press.

Salas, E., D.E. Sims y C.S. Burke (2005). "Is there a 'Big Five' in teamwork?", *Small group research*, vol. 36, núm. 5, pp. 555-599.

Schelling, T.C. (1978). *Micromotives and macrobehavior*. Nueva York: W. W. Norton & Company.

Schelling, T.C. (1971). "Dynamics models of segregation", *Journal of Mathematical Sociology*, vol. 1, núm. 2, pp. 143-186.

Schelling, T.C. (1969). "Models of segregation", *The American Economic Review*, vol. 59, núm. 2, pp. 488-493.

Simon, H. (1969). *The Sciences of the Artificial*. Cambridge: The MIT Press.

Simon, H. (1955). "A behavioral model of rational choice", *Quartely Journal of Economics*, vol. 69, núm. 4, pp. 99-118.

Simon, H. y J.B. Kadane (1975). "Optimal problem-solving search: All of none solutions", *Artificial Intelligence*, núm. 6, pp. 235-247.

Squazzoni, F. (2012). *Agent-based computational sociology*. Chichester: John Wiley & Sons.

Smith, V.L. (1982). "Microeconomic systems as an experimental science", *The American Economic Review*, vol. 72, núm. 5, pp. 923-955.

Smith, V.L. (1962). "An Experimental Study of Competitive Market Behavior", *Journal of Political Economy*, vol. 70, núm. 2, pp. 111-137.

Sterman, J.D. (2000). *Business dynamics: Systems thinking and modeling for a complex world*. Boston: McGraw-Hill Higher Education.

Wilensky, U.. (1999). Netlogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. Disponible en http://ccl.northwestern.edu/netlogo

Wilensky, U. y W. Rand (2015). *An introduction to agent-based modeling: Modeling natural, social and engineered complex systems with Netlogo*. Cambridge: MIT Press.

Introducción al modelado basado en agentes. Una aproximación desde Netlogo
de Antonio Aguilera Ontiveros y Marta Posada Calvo
se terminó de imprimir el 28 de diciembre de 2017
en los talleres de Editorial y Distribuidora Académica Libertad Mexicana
S.A. de C.V., Misterios núm. 192, Col. Vallejo, Gustavo A. Madero,
C.P. 07870 Ciudad de México. tel. (55) 56132520.
La composición tipográfica la realizó Ernesto López Ruiz.
La edición estuvo al cuidado de la Unidad de Publicaciones
de El Colegio de San Luis y los autores.
El tiro consta de 250 ejemplares.

NOVEDADES

Márgenes del canon: la antología literaria en México e Hispanoamérica
Antonio Cajero Vázquez
(editor)

La estadística general de 1848. Demografía y espacios socio-económicos en la ciudad de San Luis Potosí,
Sergio Alejandro Cañedo Gamboa y Marco Antonio Vázquez Rocha
(coordinadores)

Medrar para sobrevivir. Individualidades presas en la fragua de la historia (siglos XVI-XIX)
Thomas Calvo y José Armando Hernández S.
(coordinadores)

La antropología de lo nefasto en comunidades indígenas
Danièle Dehouve

Problemática y gestión del agua en la cuenca semiárida y urbanizada del valle de San Luis Potosí
Francisco Peña y Germán Santacruz
(coordinadores)

Programa Constructivo. Su significado y su lugar
Mohandas Karamchand Gandhi

Comercio, alcabalas y negocios de familia en San Luis Potosí, México. Crecimiento económico y poder político, 1820-1846
Sergio Alejandro Cañedo Gamboa

El ir y venir de los norteños. Historia de la migración mexicana a Estados Unidos, siglos XIX-XXI
Fernando Saúl Alanís y Rafael Alarcón Acosta
(coordinadores)

Palabras de injuria y expresiones de disenso. El lenguaje licencioso en el mundo hispánico
Claudia Carranza y Rafael Castañeda
(coordinadores)

El espacio de la fiesta y lugares de la tradición. Tensiones y vínculos en torno a la fiesta patronal del barrio de San Miguelito en San Luis Potosí
David Madrigal

Antonio Aguilera Ontiveros
Marta Posada Calvo

Introducción al modelado basado en agentes. Una aproximación desde NetLogo

EL COLEGIO DE SAN LUIS

COLECCIÓN INVESTIGACIONES

El libro *Introducción al modelado basado en agentes. Una aproximación desde NetLogo* explica en qué consiste modelar basándose en agentes y, por medio de tres sencillos ejemplos, introducir al lector en el modelado basado en agentes en las ciencias sociales. Dichos ejemplos caen dentro del campo de la sociología computacional, la economía computacional y la teoría computacional de las organizaciones.

El libro intenta ser un manual práctico para aquellos investigadores, ya sean sociólogos, economistas o estudiosos de las organizaciones que quieran utilizar el modelado basado en agentes en sus investigaciones. Para ello, se ofrece al lector un marco modélico basado en uno de los estándares más difundidos para describir modelos basados en agentes, esto es, el protocolo ODD. Se usa dicho protocolo como estándar para describir los ejemplos que se incluyen en este libro.

Además, se pretende que el libro sea un manual sencillo y práctico de introducción a NetLogo, el lenguaje computacional que nos servirá para la construcción de los modelos computacionales.

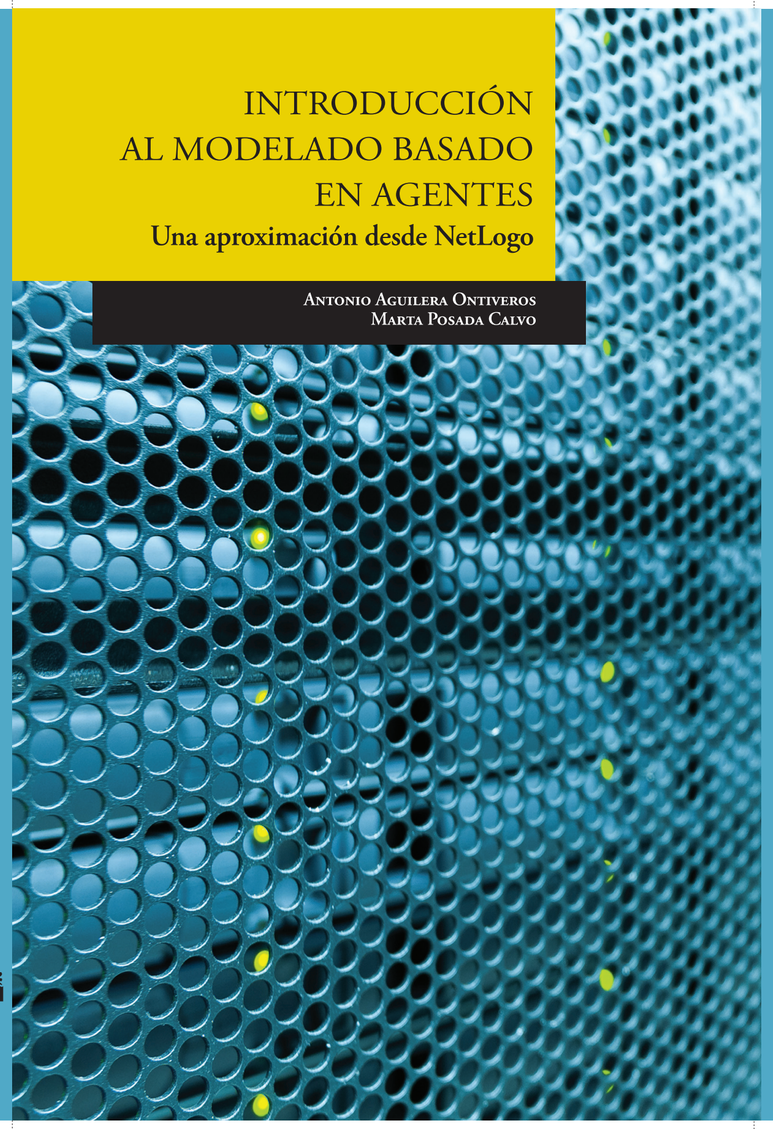
ISBN 978-607-8500-41-3

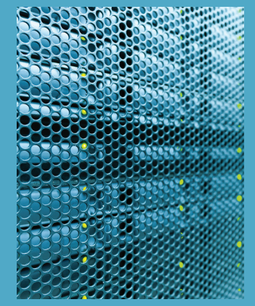
INTRODUCCIÓN AL MODELADO BASADO EN AGENTES
Antonio Aguilera Ontiveros. Marta Posada Calvo

INTRODUCCIÓN AL MODELADO BASADO EN AGENTES

Una aproximación desde NetLogo

Antonio Aguilera Ontiveros
Marta Posada Calvo





Portada: Sala de centro de datos de Internet de alta tecnología. Fotografía de Konstantin Yolshin. Imagen publicada bajo licencia de Shutterstock.