

Modelación basada en Agentes

Alfonso Meléndez

Contents

	5
Prologo	7
Introducción	9
Sistemas Complejos	9
¿Para quién es este libro?	10
 I Conceptos Básicos	 13
1 ¿Qué es un modelo?	15
1.1 Introducción	15
1.2 Un primer ejemplo	16
1.3 El ciclo de modelaje	20
2 Modelacion Basada en Agentes	25
2.1 ¿Qué es Moba?	25
2.2 Un nuevo enfoque	26
 II Contexto Histórico	 29
3 Ocho Modelos Clásicos	31
3.1 Un descubrimiento importante	31
3.2 El Mundo de las hormigas	33
3.3 Modelo del Fuego	34
3.4 Bar El Farol	36
3.5 Tortugas y Ranas	37
3.6 El Moho	39
3.7 Trancones de Tráfico	41
3.8 Ecología basada en agentes	44
4 Algo de Historia	47

4.1	Autómatas celulares y modelado basado en agentes	50
4.2	Algoritmos genéticos, John Holland y sistemas adaptativos complejos	53
4.3	Seymour Papert, Logo y la tortuga	56
4.4	Paralelismo de datos	58
4.5	Gráficos por computadora, sistemas de partículas y boids	59
4.6	Conclusión	61
III	Un Modelo Básico	63
5	Buscadores de Hongos	65
5.1	Construyendo el Modelo	65
5.2	Análizando el Modelo	88
6	Construyendo Modelos Basados en Agentes	105
6.1	¿Cómo modelar un MOBA?	105
6.2	El paradigma del agente	106
6.3	Otros tipos de modelado	108
IV	Un Modelo Incremental	109
7	El Modelo Predador Presa	111
7.1	Descripción del Modelo	111
7.2	Diseño del Modelo	117
7.3	Cinco modelos incrementales	124
8	Analizando Modelos Basados en Agentes	151
8.1	Modelos e investigación	151
8.2	Herramientas básicas	152
8.3	Un Ejemplo de Análisis	153
V	Un Modelo de Infección	159
9	El Modelo SI de Infección	163
9.1	Construcción del Modelo	163
9.2	Ánálisis del Modelo	171
10	Verificación, Validación Replicación	185
10.1	Corrección de un modelo	185
10.2	Verificación	186
10.3	Comunicación	187
10.4	Descripción de Modelos Conceptuales	188
10.5	Verification Testing	189
10.6	Más allá de la verificación	191
10.7	Ánálisis de sensibilidad y robustez	194

CONTENTS	5
10.8 Beneficios de la verificación	197
10.9 Validación	198
10.10 Replicación	200
VI Un Modelo Económico	203
11 Modelo de Inversión	205
11.1 Descripción del Modelo	205
11.2 Construcción del Modelo	208
11.3 Análisis del Modelo	219
VII Proyectos Propuestos	233
12 Proyectos Básicos	235
12.1 La Hormiga Atomica	235
12.2 Partículas	236
12.3 El modelo de paridad	237
12.4 El Modelo del Chisme (Gossip Model)	238
13 Proyectos Intermedios	241
13.1 Majority model	241
13.2 Patron Misterioso	243
13.3 Heroes y Cobardes	245
13.4 Termitas	247
14 Proyectos Avanzados	249
14.1 Simple economy	249
14.2 Fuego en Linea	250
14.3 Un modelo básico de comercio (Toy Trader)	252
14.4 Un Eco-Sistema	254
A Redes	257
A.1 Introducción	257
A.2 Conceptos Básicos de Redes.	257
A.3 Redes aleatorias	260
A.4 Redes de mundo pequeño	263
A.5 Redes sin escala	266
A.6 El Modelo SIR	268
B Tutorial NetLogo	279
B.1 ¿Qué es NetLogo?	279
B.2 El mundo NetLogo	281
B.3 Programación	283
Programación en general	283

C Simulación de Experimentos con NetLogo	293
C.1 Simulacion Biologica	293
C.2 Experimentos de Simulación (Analizador de Comportamiento) . .	295
C.3 Analizando los resultados del experimento en Rstudio	301
D Bibliografía	309

We shape our tools and then our tools shape us.

— Marshall McLuhan

What I cannot create, I do not understand.

— Richard Feynman

Portada

Diagrama Interactivo de riqueza promedio de inversores (z) dados el número de estos (x) y el tamaño del espacio de inversión (y). (Mayor información sección 11.3.5 del libro)

```
## TypeError: Attempting to change the setter of an unconfigurable property.  
## TypeError: Attempting to change the setter of an unconfigurable property.
```


Prologo

“Algunos miran las cosas como son y preguntan por qué. Sueño con cosas que nunca fueron y pregunto ¿por qué no?”

— John F. Kennedy

“Creo que el próximo siglo será el siglo de la complejidad.”

— Stephen Hawking

Cualquier nuevo enfoque de la ciencia tiene que pasar por su infancia: como un niño aprendiendo a caminar, los primeros pasos de un nuevo enfoque son exploratorios, inseguros, y sin una idea clara de hacia dónde podrían llevar , sin embargo estos pasos son observados con gran interés y entusiasmo. El modelado basado en agentes (MOBA) ha estado en su infancia durante las últimas dos décadas, pero el rápido interés creciente por estos modelos ciertamente es alentado por los enormes aumentos en la potencia informática que ahora hacen práctico simular grandes cantidades de individuos (agentes) interactuando en diferentes ambientes (geográficos, redes, virtuales), pero también ha sido impulsado por otro tipo de poder que ha crecido rápidamente en los últimos años: el deseo de comprender los fenómenos complejos naturales y cómo emergen de la variabilidad, la adaptabilidad y la organización de elementos individuales. Los pioneros de los MOBA afirmaron que un cambio de enfoque de los “agregados” a los “individuos” conduciría a nuevas ideas fundamentales, de hecho numerosos modelos basados en agentes o en individuos (MOBAs) han demostrado la importancia de modelar las características individuales y sus interacciones para entender las dinámicas de los sistemas naturales y sociales. El modelado basado en agentes ha cambiado y a la vez complementado nuestra comprensión de los sistemas, pero un cambio de enfoque de agregados a individuos no conduce **automáticamente** a una mejor y más general teoría o a estrategias más efectivas para resolver problemas aplicados.Las limitaciones de los modelos analíticos (ecuaciones diferenciales) que consideran agregados homogeneos y no individuos o agentes heterogeneos tienen un precio, los modelos basados en agentes, MOBAs ,son más sofisticados que los modelos analíticos y esto los hace más difíciles de desarrollar, comprender y comunicar, muchos proyectos basados en MOBAS han sido desafiados por problemas metodológicos y computacionales, creo firmemente que estos problemas están siendo superados mediante la adaptación de nuevas técnicas para analizar

modelos y en un futuro estos modelos jugar un papel central en el análisis de fenómenos complejos dando nuevas luces a estos y complementando y ampliando los modelos tradicionales.

Introducción

Sistemas Complejos

Cuando necesitamos estudiar un sistema real compuesto de elementos interconectados , donde cada uno de estos tiene su propia dinámica, a menudo es imposible prever el surgimiento de una dinámica global para el sistema. En este caso, lo que está en juego es un sistema complejo, cualquier modificación ,incluso si es marginal, en términos de uno o varios de sus elementos constitutivos puede conducir a un cambio dramático en la operación general del sistema. Es claro que estos fenómenos pueden ser observados y entendidos solo a través de la construcción de un modelo y a pesar de que en ciertos casos particulares el modelo puede resolverse analíticamente, como es el caso del modelo depredador-presa (Lotka-Volterra), la simulación por computador es indispensable en casi todos los demás casos. Un amplio e interdisciplinar campo como los “sistemas complejos” (complex systems) es cada vez más importante para entender nuestro mundo y este libro es una introducción a una de las disciplinas que han surgido de la investigación en sistemas complejos. La modelación basado en agentes (MOBA), es una nueva forma de hacer ciencia, organiza nuestro pensamiento por analogía con el mundo alrededor de nosotros y es una forma elegante e intuitiva de visualizar y representar un fenómeno complejo, partiendo de lo individual y heterogeneo (personas en una ciudad, individuos en un mercado, hormigas en el desierto) y no de los agregados respectivos (poblaciones, mercados, colonias) y observando y analizando interacciones entre individuos para poder entender comportamientos emergentes (migraciones,oferta-demanda,adaptabilidad) en lugar de construyendo ecuaciones y resolviéndolas matemáticamente . Esta manera de abordar los fenómenos (bottom-up) se ha adoptado en una amplia gama de disciplinas como ciencias sociales, ecología y finanzas y permite abordar preguntas como:

- ¿Cómo interactúan y compiten múltiples especies para formar un ecosistema estable?
- ¿Cómo afectan las instituciones políticas las decisiones individuales, especialmente cuando esas personas tienen la capacidad de manipular las instituciones políticas?

El conocimiento de herramientas para el modelamiento de sistemas complejos se convertirá en una necesidad hacia el futuro, y el modelado basado en agentes (MOBA) es una de las maneras más importantes de modelar este tipo de sistemas.

¿Para quién es este libro?

Este texto puede servir como texto principal para un curso universitario interdisciplinario, un curso sobre sistemas complejos o una clase de informática con énfasis en el modelado basado en agentes. Puede ser utilizado como texto complementario en una amplia gama de clases de pregrado, incluido cualquier clase donde el modelado basado en agentes se puede aplicar a áreas como las ciencias naturales: (física, química y biología) , ciencias sociales (como psicología,sociología y lingüística) y clases de ingeniería como ciencia de materiales, ingeniería industrial e ingeniería civil. El campo de la modelación basada en agentes (MOBA) es aplicable a muchos dominios y se puede usar en una amplia variedad de contextos.

El requisito previo para comenzar a entender, construir y analizar un modelo basado en agentes es muy bajo, los modelos basados en agentes, como veremos, son muy intuitivos y sencillos de entender y modelar, pero al mismo tiempo hay pocos límites para lo que se puede lograr una vez se construye el modelo. Los MOBAs son una herramienta que además de ayudar a comprender sistemas complejos, le permite construir y usar modelos para investigar **sus propias preguntas..** El principio rector que tendremos a lo largo del libro es:

“Piso bajo, techo alto” (low floor , high ceiling)

El material para este libro de texto surgió del Curso de Pregrado que he dictado durante varios años en el Deaprtamento de Ingeniería Biomédica de la Escuela Colombiana de Ingeniería Julio Garavito, este curso no asume ningún conocimiento matemático más allá del álgebra básica y conceptos básicos de estadística y ningún conocimiento previo de programación de computadores, de hecho es una ventaja si no se ha programado todavía en ningun lenguaje, las herramientas computacionales que usaremos extensivamente en el curso son dos:

1. Para la construcción de modelos: Netlogo:

Existen muchos lenguajes de modelado basados en agentes, pero NetLogo sigue siendo el más ampliamente utilizado. De los otros actualmente en uso, Swarm, desarrollado en Santa Fe Institute, Repast, desarrollado en Argonne National Laboratory, y MASON, desarrollado en George Mason University, son mucho más avanzados y requieren de una empinada curva de aprendizaje. La mayoría de los kits de herramientas ABM (incluido NetLogo) son de código abierto y están disponibles de forma gratuita. AnyLogic es un paquete comercial que también ha tenido éxito. NetLogo es código abierto y se encuentra disponible de forma gratuita, ningún otro lenguaje de MOBA existente está cerca del “piso

bajo” . Como tal es un lenguaje ideal para aprender a construir MOBAs y se usa ampliamente en las aulas de todo el mundo, también es usado por un gran número de científicos y profesionales y es empleado regularmente en investigación de vanguardia. Aprender esta nueva manera de enfocar los problemas complejos y de hacer ciencia con ellos, lo hará un mejor investigador en su área de interés.

2. Para el análisis de MOBAs: R y RStudio:

Generando datos de los modelos construidos en NetLogo y usando “Ciencia de Datos” sobre estos, podremos validar hipótesis, realizar análisis de sensibilidad, calibrar modelos, para ello usaremos poderosas herramientas de visualización y análisis encontradas en el ambiente R.(por ejemplo librerías como tidyverse, ggplot y plot-ly)

Trabajar en este libro requerirá entender e implementar modelos basados en agentes en el computador, esto puede ser desconocido para algunos lectores. Aunque muchas personas creen que la programación de computadores es difícil de aprender, hay que decir que la investigación de educadores construcciónistas y décadas de experiencia han demostrado que prácticamente todos los estudiantes pueden aprender a programar en NetLogo y usar R para análisis estadísticos básicos. Esperamos que no se intimide con los modelos que se deben construir y que hacen parte de este libro de texto, es seguro que si usted puede lograrlo, tomándose el tiempo para ello, esto le brindará grandes dividendos.

Part I

Conceptos Básicos

Chapter 1

¿Qué es un modelo?

1.1 Introducción

Un modelo es una representación intencional de algún sistema real Starfield et al. 1990. Construimos y usamos modelos para resolver problemas o responder preguntas sobre un sistema o una clase de sistemas. En ciencia, generalmente se quiere entender cómo funcionan las cosas, explicar patrones que surgen de lo que se observa o predecir el comportamiento de un sistema en respuesta a algún cambio. Los sistemas reales a menudo son demasiado complejos o evolucionan muy lentamente para ser analizados mediante experimentos, por ejemplo, sería extremadamente difícil y lento entender cómo crecen las ciudades y como cambia el uso de la tierra en un país solo con experimentos, por lo tanto, intentamos formular una representación simplificada del sistema utilizando normalmente:

- ecuaciones matemáticas. (diferenciales por lo general)
- un programa de computadora que luego podemos manipular y experimentar (simulador).

Hay muchas formas de representar un sistema real (una ciudad o un paisaje por ejemplo) en una forma simplificada, pero:

- ¿Cómo podemos saber qué aspectos del sistema real incluir en el modelo y qué ignorar?

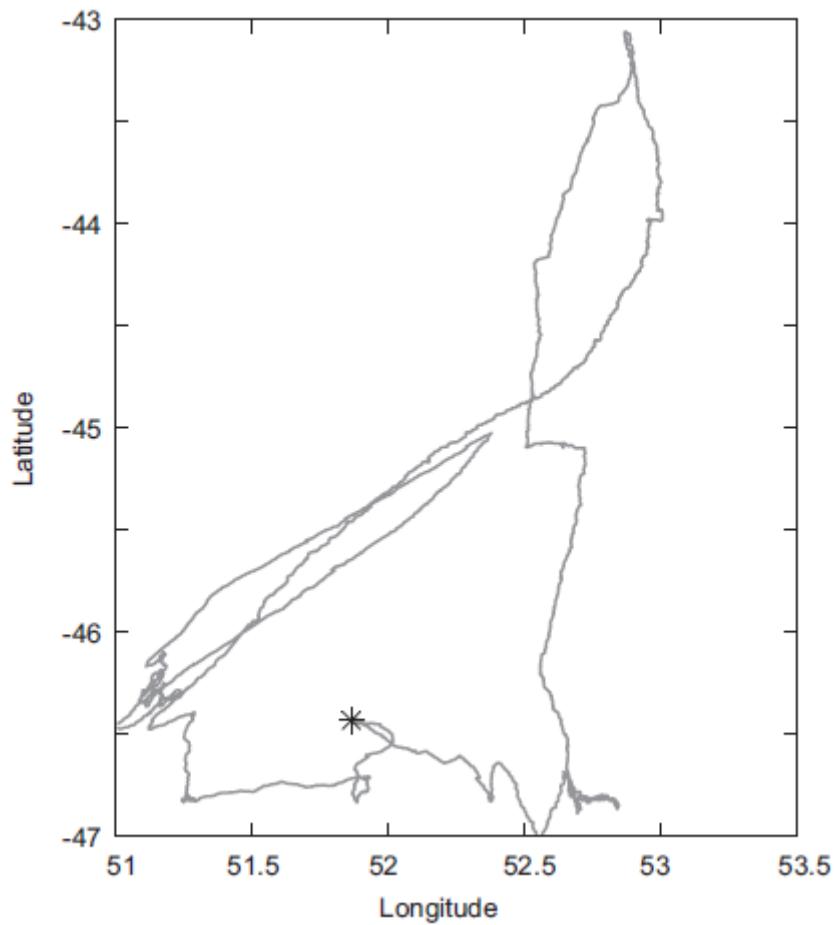
Para responder a esta pregunta, el **propósito** o intención del modelo es decisivo. La pregunta que queremos responder con el modelo es el filtro que nos permite excluir todos aquellos aspectos del sistema real considerados irrelevantes o no muy importantes para este propósito, estos son ignorados en el modelo o representados solo de una manera muy simplificada.

1.2 Un primer ejemplo

Consideremos un ejemplo simple, pero no trivial: ¿Alguna vez buscó hongos en un bosque? ¿Se preguntó cuál sería la mejor estrategia de búsqueda? Los hongos son muy difíciles de ver, si se va caminando por un bosque y a menudo se pisan antes de verlos, si es un experto en hongos sabría cómo reconocer un buen hábitat de hongos, pero supongamos que usted es un neófito.



Se puede pensar en varias estrategias intuitivas de búsqueda , como buscar en un área determinada haciendo barridos amplios pero, al encontrar un hongo, pasar a una escala más reducida de barrido porque se sabe que los hongos ocurren en racimos. Pero, ¿qué significa “grande”, “pequeño”, “barridos”? y ¿cuánto tiempo debería transcurrir para terminar los barridos más pequeñosy volver a los más amplios? Muchas especies animales enfrentan problemas similares, por lo que es probable que la evolución los haya equipado con buenas estrategias de búsqueda adaptativa, en general la búsqueda de recursos alimenticios por parte de una especie es un problema vital y es un tema grande de estudio en ecología ForAging. (Es probable que lo mismo sea cierto para las organizaciones humanas en búsqueda de ganancias o de paz con sus vecinos.) El albatros, por ejemplo, se comporta de cierta manera como un buscador de hongos cuando busca alimento para sus crías : alterna largas distancias más o menos lineales con movimientos a pequeña escala:



Una característica común del buscador de hongos y el albatros es que su radio de detección es limitado, solo pueden detectar lo que buscan cuando están cerca , además los elementos buscados no se distribuyen totalmente al azar, sino en

grupos, por lo que el comportamiento de búsqueda debe ser **adaptativo**: debe cambiar una vez que se encuentra lo buscado.

- ¿Por qué querríamos desarrollar un modelo de este problema?

Porque incluso para este simple problema es difícil desarrollar modelos mentales cuantitativos, intuitivamente encontramos una estrategia de búsqueda que funciona bastante bien, pero luego vemos a otros que usan estrategias diferentes y encuentran más hongos en este caso : ¿Son más afortunados o son mejores sus estrategias? Necesitamos un propósito claramente formulado antes de poder formular un modelo. Imagine que alguien simplemente dice: “Por favor, modele la captura de hongos en el bosque”, esta es una pregunta muy vaga, ¿En qué hay que concentrarse?

- En diferentes especies de hongos
- En diferentes tipos de bosques
- En identificación de buenos y malos hábitats.
- En efectos de un incendio en poblaciones de hongos, etc.

Sin embargo, si el propósito es:

“¿Qué estrategia de búsqueda maximiza el número de hongos encontrados en un tiempo específico?”

sabemos que, por ejemplo:

- Podemos ignorar los árboles y la vegetación; solo tenemos que tener en cuenta que los hongos están distribuidos en grupos. Además, podemos ignorar cualquier otra heterogeneidad en el bosque, como topografía o tipo de suelo, que pueden afectar un poco la búsqueda, pero no lo suficiente como para afectar la respuesta general a nuestra pregunta.
- Será suficiente con representar al cazador de hongos de una manera muy simplificada: como un punto (o una flecha) en movimiento que tiene un cierto radio de detección y registra cuántos hongos ha encontrado al igual que tiene en cuenta cuánto tiempo ha pasado desde que encontró el último hongo (para cambiar de tipo de búsqueda)

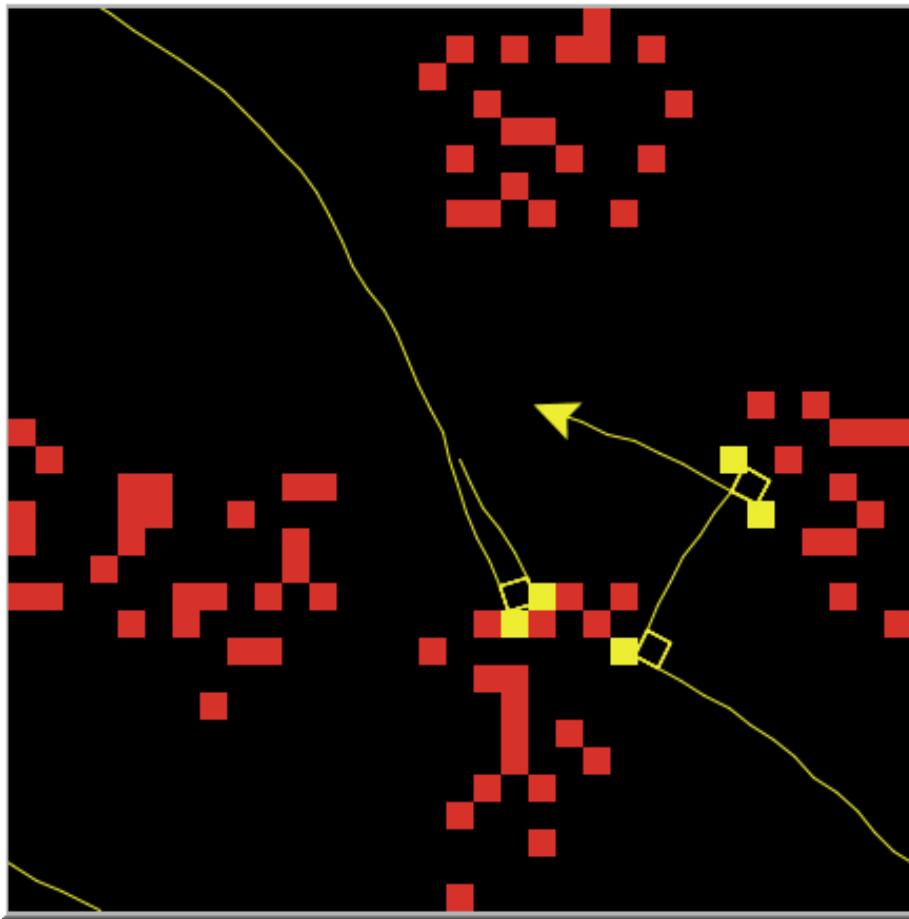
Con este propósito de maximizar los hongos encontrados en un periodo de tiempo, podemos formular un modelo muy sencillo que incluya dos tipos de agentes:

- clusters de hongos distribuidos en el bosque. (agentes estáticos)
- Un “agente” buscador que los busca. (agentes dinámicos)

Si el agente dinámico (o individuo) encuentra un hongo, su búsqueda cambia a una manera de búsqueda más local, pero si el tiempo transcurrido desde que encontró el último elemento supera un umbral (este podría ser un parámetro del modelo), el agente cambia su estrategia a movimientos más amplios para aumentar sus posibilidades de detectar otro cluster de hongos.

(Nota: Si suponemos que la capacidad de detección no cambia con la velocidad de movimiento, incluso podemos ignorar la velocidad con la que se mueve el

agente)



La anterior figura muestra un ejemplo de ejecución de un posible modelo construido en NetLogo, una pregunta muy importante, y a veces angustiante, es:

- ¿cómo podemos saber que factores o aspectos son importantes con respecto a la pregunta abordada con un modelo?

La respuesta clara y contundente es: ¡no podemos!, esta es exactamente la razón por la cual tenemos que formular, implementar y luego analizar un modelo, porque entonces y solo entonces podremos explorar rigurosamente las consecuencias de nuestros supuestos simplificadores y observar que factores son o no son relevantes.

Nuestra primera formulación del modelo debe basarse en nuestra comprensión preliminar de cómo el sistema funciona, cuáles son los elementos y procesos importantes, etc. Estas ideas preliminares pueden basarse en:

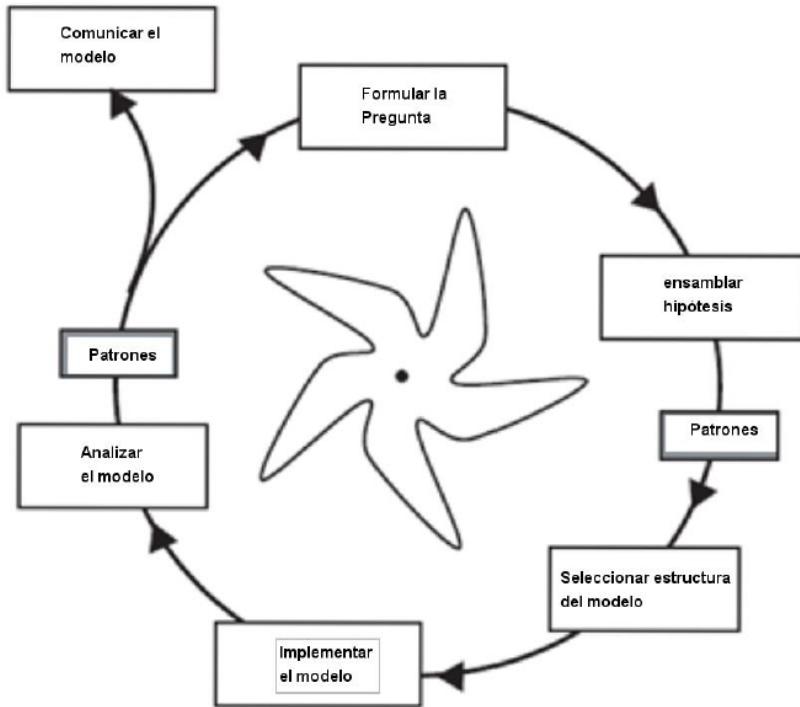
- el conocimiento empírico del comportamiento del sistema.

- en modelos anteriores que abordan preguntas similares
- en teorías
- o simplemente en nuestra imaginación (como en nuestro caso).

Sin embargo, si no tenemos idea de cómo funciona un sistema, no podremos formular un modelo! Por ejemplo, aunque los científicos son felices al modelar casi todo, hasta ahora parece no haber un modelo explícito de la conciencia humana, simplemente porque hasta el momento nadie tiene mucha idea de qué es realmente la conciencia y cómo se origina. Debido a que los supuestos en la primera versión de un modelo son experimentales, tenemos que probar si son apropiados y útiles. Para esto, necesitamos **criterios** para determinar si el modelo puede ser considerado una buena representación del sistema real. Estos criterios se basan en patrones o regularidades que nos permiten identificar y caracterizar el sistema real en primer lugar. Los modelos de mercado, por ejemplo, deberían producir los tipos de volatilidad en los precios y tendencias que vemos en mercados reales. A menudo encontramos que la primera versión de un modelo es demasiado simple, carece de procesos y estructuras importantes, o simplemente es inconsistente. Entonces es el momento de revisar nuestro propósito y nuestras suposiciones iniciales

1.3 El ciclo de modelaje

Cuando pensamos en construir un modelo como el de buscador de hongos (o el del albatros), realizamos casi siempre una serie de tareas. El modelado científico significa realizar estas tareas de forma sistemática y utilizar los modelos basados en agentes, MOBAS, construidos (llamados con frecuencia **simuladores**) para validar las consecuencias de los supuestos simplificadores que componen nuestros modelos. Ser científico siempre significa repetir las tareas de modelado varias veces, porque nuestros primeros modelos siempre se pueden mejorar de alguna manera: son demasiado simples o demasiado complejos, o nos hicieron darnos cuenta de que estábamos haciendo las preguntas equivocadas. Por lo tanto, es útil ver el modelado como una **iteración** a través del llamado “ciclo de modelado”:



Iterar no significa que siempre recorremos por el ciclo completo; más bien, a menudo pasamos por bucles más pequeños, el ciclo de modelado consiste en las siguientes tareas:

1. Formular la pregunta.

Necesitamos comenzar con una pregunta de investigación muy clara porque entonces la pregunta sirve como la brújula principal y el filtro para diseñar un modelo. A menudo, formulando una pregunta clara y productiva es en sí misma una tarea importante porque una pregunta clara requiere un enfoque claro. Para sistemas complejos, enfocarse puede ser difícil. Muy a menudo, incluso nuestras preguntas son solo experimentales y luego podríamos necesitar reformular la pregunta, tal vez porque resultó no ser lo suficientemente clara, o demasiado simple o demasiado compleja. La pregunta en nuestro modelo de búsqueda de hongos es:

- ¿Qué estrategia de búsqueda maximiza la tasa de encontrar hongos si estos se distribuyen en grupos?

2. Ensamblar hipótesis.

El modelado basado en agentes DeAngelis es intuitivo y análogo a la realidad, en el sentido de que no estamos tratando de agregar agentes y sus funcionalidades en algunas variables abstractas y agregadas como población, biomasa, riqueza

general, etc... En cambio, representamos los agentes individuales directamente y definimos a través de sencillas reglas su comportamiento, luego los colocamos en un entorno (mundo) y “corremos” el modelo para observarlo y ver qué podemos aprender de él. Tenemos que obligarnos a simplificar tanto como podamos, o incluso más. El ciclo de modelado debe comenzar con el modelo más simple posible, porque queremos desarrollar una comprensión gradual, un error muy común (y a veces difícil de evitar) es “arrojar” demasiado a la primera versión del modelo, generalmente argumentando que todos estos factores son claves y no pueden ser ignorados. Entonces, la respuesta del modelador experto debía ser:

- “Sí, puede que tengas razón, pero centrémonos en el número mínimo de factores primero, coloca todos los otros elementos que crees que podrías necesitar en el modelo en tu “lista de deseos” y podemos verificar su importancia más adelante”.

La razón de este consejo es esta: nuestra comprensión inicial de un sistema no es suficiente para decidir si las cosas son más o menos importantes para un modelo. **Es el propósito del modelo el que nos enseña lo que es importante.** Es aconsejable tener un modelo implementado lo antes posible, incluso si es ridículamente simple, cuanto más fácil sea implementarlo y analizarlo mejor. La fase productiva real en un proyecto MOBA comienza cuando ponemos en marcha el modelo y lo comenzamos a observar y analizar. Para el modelo de Búsqueda de Hongos asumimos que el proceso esencial es cambiar entre búsqueda relativamente recta a gran escala y búsqueda a pequeña escala, dependiendo de cuánto tiempo haya pasado desde la última vez que el buscador encontró un hongo

3. Elegir escalas, entidades, variables de estado, procesos y parámetros.

Una vez que elegimos algunos supuestos simplificadores e hipótesis para representar nuestro sistema, es hora de sentarse y pensar en nuestro modelo en detalle. Así producimos una formulación escrita del modelo. Producir y actualizar esta formulación es esencial para todo el proceso de modelado, incluida la entrega a nuestros “clientes” (nuestro comité de tesis, revisores de revistas, investigación patrocinadores, etc.). Este paso, para el modelo de Buscador de Hongos, incluye especificar:

- Cómo se representa el espacio donde se mueven los buscadores (como cuadrículas cuadradas con un tamaño igual).
- Qué tipos de objetos hay en el modelo (un buscador y los elementos que busca).
- Las variables de estado o las características del buscador (el tiempo que ha buscado y la cantidad de hongos que ha encontrado, y el tiempo desde la última vez que encontró un hongo) y..
- Cómo se busca (comportamiento del modelo).

(Nota: Existe un protótipo de formulación de un MOBA muy usado llamado el Protocolo ODD (Odd Protocol)

4. Implementar el modelo.

Esta es la parte más técnica del ciclo de modelado, donde convertimos nuestra descripción verbal del modelo en un objeto “animado”. ¿Por qué “animado”? Porque, en cierto modo, el modelo tiene su propia dinámica independiente (o “vida”), impulsada por la lógica interna del modelo y entonces podemos explorar, de forma rigurosa, las consecuencias de nuestros supuestos y ver si nuestro modelo inicial es útil. Esta tarea a menudo es la más desalentadora, porque generalmente no se tiene entrenamiento en cómo construir modelos en computador (en este caso de modelos basados en agentes). La plataforma que usaremos (NetLogo), nos permitirá implementar MOBAs de un manera natural, sencilla y eficiente a la vez.

5. Analizar, probar y revisar el modelo.

Los modeladores novatos podrían pensar que diseñar e implementar un modelo en la computadora es lo que requiere más trabajo pero no, la tarea una vez implementado el modelo es analizarlo y aprender de él es la que requiere más tiempo y es la más exigente. Con NetLogo, aprenderá a implementar rápidamente sus propios MOBAs, para hacer ciencia para analizarlos usaremos R y RStudio. Gran parte de este libro estará dedicado a esta tarea:

- ¿Cómo podemos aprender de nuestros modelos?

Para responder a la pregunta en el caso de la búsqueda de hongos, podríamos analizar el modelo construido, probándolo con una variedad de algoritmos de búsqueda y valores de parámetros para ver cuál produce la tasa más alta de encontrar hongos

Chapter 2

Modelacion Basada en Agentes

2.1 ¿Qué es Moba?

Históricamente, la complejidad de los modelos científicos ha estado limitada por la trazabilidad matemática, cuando el cálculo diferencial era el único enfoque que teníamos para modelar, teníamos que mantener modelos lo suficientemente simples para “resolverlos” matemáticamente y, por desgracia, a menudo estábamos limitados a modelar problemas bastante simples. Con la simulación por computador, se eliminan estas limitaciones y entonces podemos abordar problemas que requieren modelos que están menos simplificados e incluyen más características de los sistemas reales. Los MOBAs están menos simplificados de una manera específica e importante: representan los componentes individuales de un sistema y sus comportamientos. En lugar de describir un sistema solo con variables que representan el estado de todo el sistema, modelamos sus agentes individuales. Los MOBA son, por lo tanto, modelos en los que los individuos o agentes se describen como únicos y autónomos, entidades que generalmente interactúan entre sí y con su entorno local. Los agentes pueden ser organismos, seres humanos, empresas, instituciones o cualquier otra entidad que persiga cierta objetivo. Los agentes generalmente son **diferentes** entre sí en características como tamaño, ubicación, consumo de recursos etc... Interactuar localmente significa que los agentes usualmente no interactúan con todos los demás agentes sino solo con sus vecinos en el espacio geográfico o en algún otro tipo de “espacio” como una red. Ser autónomo implica que los agentes actúan independientemente el uno del otro y persiguen sus propios objetivos. Los organismos se esfuerzan por sobrevivir y reproducirse, los comerciantes, en el mercado de valores, intentan ganar dinero, las empresas tienen objetivos como: cumplir metas de ganancias ó permanecer en el negocio, las autoridades gubernamentales quieren hacer cumplir las leyes y

brindar bienestar público. Por lo tanto, los agentes utilizan un comportamiento **adaptativo**: ajustan su comportamiento a sus estados actuales y a los de otros agentes y a su entorno. El uso de MOBAs nos permite abordar problemas relacionados con lo que se denomina **emergencia**: la dinámica de un sistema que surge de cómo los componentes individuales del sistema interactúan y responden entre sí y su entorno. Por lo tanto, con MOBAs podemos estudiar preguntas sobre cómo el comportamiento de un sistema que surge de, y está vinculado a, las características y comportamientos de sus componentes individuales. ¿Qué tipo de preguntas son estas? Aquí hay unos ejemplos:

- ¿Cómo podemos manejar los bosques tropicales de manera sostenible, manteniendo ambos usos económicos? y niveles de biodiversidad críticos para las propiedades de estabilidad de los bosques (Huth et al. 2004)?
- ¿Qué causa la dinámica compleja y aparentemente impredecible de un mercado de valores?
- Las fluctuaciones del mercado causadas por el comportamiento dinámico de los comerciantes, la variación en el valor de las acciones, o ¿simplemente las reglas comerciales del mercado (LeBaron 2001, Duffy 2006)?
- ¿Cómo se regula el desarrollo del tejido humano por las señales del genoma y el entorno extracelular y por comportamientos celulares como la migración, proliferación, diferenciación y muerte de las células? ¿Cómo resultan las enfermedades de anomalías en este sistema (Peirce et al. 2004)?
- ¿Cómo responden las poblaciones de aves playeras a la pérdida de las marismas en las que se alimentan y cómo pueden mitigarse estos efectos de la mejor manera? (Natillas Goss et al. 2006)?
- ¿Qué impulsa los cambios en el uso del suelo durante la expansión urbana y cómo se ven afectados por el entorno físico y las políticas de gestión (Brown et al. 2004, Parker et al. 2003)?

2.2 Un nuevo enfoque

Los MOBAs son útiles para problemas que involucran fenómenos emergentes, los modelos emergentes son modelos “multinivel”:

lo que sucede al sistema por lo que hacen sus individuos y lo que les sucede a los individuos por lo que hace el sistema

Tradicionalmente algunos científicos han estudiado sistemas, modelándolos utilizando enfoques tales como ecuaciones diferenciales que representan cómo cambia todo el sistema. Otros científicos estudian solo lo que llamamos agentes: cómo cambian las plantas y los animales, las personas, las organizaciones, etc. y cómo se adaptan a las condiciones externas. Los MOBAs son diferentes porque están relacionados con dos (y a veces más) niveles y sus interacciones, entonces

debemos enfocarnos en los agentes y, al mismo tiempo, observar y comprender el comportamiento del sistema construido por ellos. Los MOBAs también son a menudo diferentes de los modelos tradicionales al ser “no simplificados”, por ejemplo al representar cómo los individuos y las variables ambientales los afectan, como varían según el espacio, el tiempo u otras dimensiones. Los MOBAS a menudo incluyen procesos que sabemos que son importantes pero que son demasiado complejos para incluirlo en modelos más simples, la capacidad de los MOBAs para abordar problemas complejos de varios niveles tiene un costo, por supuesto, el modelado tradicional requiere habilidades matemáticas, especialmente cálculo diferencial y estadística, pero para usar el modelado y la simulación basada en agentes necesitamos habilidades adicionales. Este libro le ayudará a:

- Desarrollar Un nuevo “lenguaje” para pensar y describir modelos.

Esto porque no podemos definir Los MOBAs de la manera concisa y precisa que usan las ecuaciones diferenciales o la estadística, se necesitan un conjunto nuevo de conceptos (por ejemplo, emergencia, comportamiento adaptativo, interacción, detección) que describen los elementos importantes de los MOBAs

- Las habilidades de software para implementar modelos

Construir modelos de computador es muy importante, para luego simular los modelos y analizarlos.

- Estrategias para diseñar y analizar modelos.

Casi no hay límite de que tan complejo puede ser un modelo de simulación por computadora, pero si un modelo es demasiado complejo, se convierte rápidamente demasiado difícil de parametrizar, validar o analizar. Necesitamos una forma de determinar qué entidades, variables y procesos deben y no deben estar en un modelo, y necesitamos métodos para **analizar un modelo, después de su construcción**, para aprender sobre el sistema real.

Los MOBAs completamente desarrollados suponen que los agentes son diferentes entre sí; que interactúan con solo algunos, no con todos los demás agentes, que cambian con el tiempo, que pueden tener diferentes “Ciclos de vida” o etapas por las que pasan, posiblemente incluyendo nacimiento y muerte, que toman decisiones adaptativas autónomas para perseguir sus objetivos etc... Sin embargo, como con cualquier hipótesis de modelado, asumir que estas características individuales son importantes es experimental, podría resultar que para muchas preguntas no necesitemos explícitamente todas, o incluso ninguna de estas características. Y, de hecho, los modelos basados en agentes completamente desarrollados son bastante raros. En ecología, por ejemplo, muchos MOBAs útiles incluyen solo algunas características individuales, y algunas interacciones locales. Por lo tanto, aunque los MOBAs se definen usando el supuesto de que los agentes están representados de alguna manera, todavía tenemos que tomar muchas decisiones sobre qué tipos de agentes representar y en qué detalle, debido a que la mayoría de los supuestos del modelo son experimentales, necesitamos probar nuestro modelo: debemos implementar el modelo y analizar sus supuestos.

Para los sistemas complejos, que se estudian en la ciencia, solo pensar no es suficiente para deducir rigurosamente las consecuencias de nuestras suposiciones simplificadoras: tenemos que dejar que el modelo implementado en el computador nos muestre lo que sucede. Por lo tanto tenemos que iterar a través del ciclo de modelado. El modelado basado en agentes no es un enfoque completamente nuevo, ofrece muchas nuevas maneras de ver problemas viejos y nos permite estudiar de una manera original muchos problemas nuevos. De hecho, el uso de Los MOBAs es mucho más emocionante ahora que el enfoque ha madurado: los peores errores han sido cometidos y corregidos, y los MOBAS ya no se consideran raros y sospechosos, tenemos herramientas adecuadas para construir MOBAs, y las personas que deciden adoptar este enfoque pueden aprovechar de lo que los pioneros han aprendido y las herramientas que construyeron, y llegar más rápidamente a trabajar problemas interesantes. Ver Historia de los MOBAs

Hemos proporcionado las ideas fundamentales e importantes sobre MOBAs. Cada vez que se sienta frustrado con un modelo propio o de otra persona, al abordar preguntas generales como:

- ¿Qué hace exactamente este modelo? ¿Es un buen modelo o no? *¿Debo agregar este o aquel proceso a mi modelo? ¿Mi modelo está ya terminado?

Podría ser útil revisar las ideas fundamentales, que en resumen son:

- Un modelo es una simplificación intencionada de un sistema para resolver un problema particular (o una categoría de problemas).
- Usamos MOBAs cuando creemos que es importante que un modelo incluya a los individuos del sistema y lo que hacen
- El modelado es un ciclo iterativo.

Part II

Contexto Histórico

Chapter 3

Ocho Modelos Clásicos

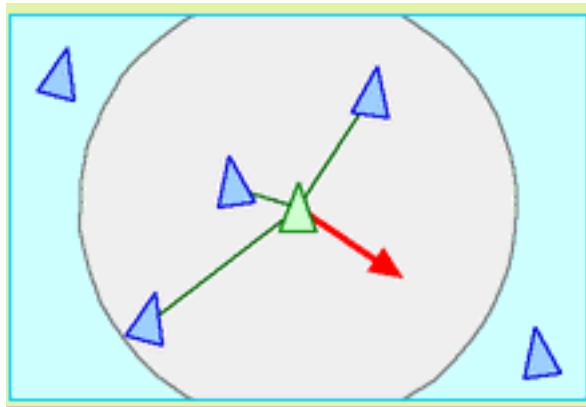
3.1 Un descubrimiento importante

En la década de 1980, y después de décadas de investigación intensiva, los biólogos todavía no tenían idea de cómo las aves logran la coordinación requerida para volar en bandadas formando patrones en forma de v. Un día, un científico de la computación observaba una bandada agruparse y la respuesta le vino a la mente. Las aves no son coordinadas por un líder o por una comunicación de otro tipo, ellas se autoorganizan. En otras palabras, las aves individuales obedecen las mismas reglas locales y seguir estas reglas llevan a esos fascinantes patrones de vuelo que vemos en la naturaleza. El éxito de Craig Reynolds (1987) consistió en probar con cautela su hipótesis construyendo un modelo basado en agentes (ABM):

Este modelo usa un grupo de objetos para simular a los pájaros y estos vuelan en un ambiente simulado. Cada triángulo representa un solo pájaro (que Reynolds llama boid). El modelo usa tres reglas simples de comportamiento que todos sus boids siguen individualmente:

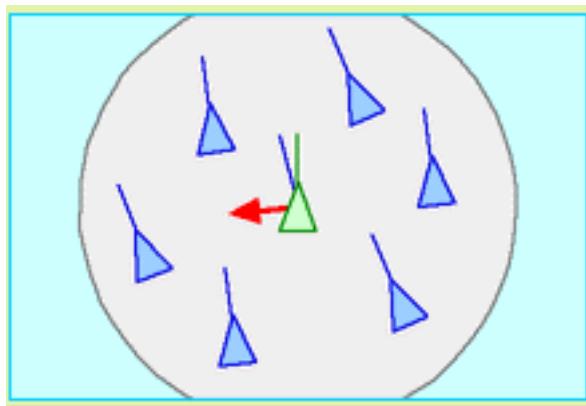
- evitar colisiones con compañeros cercanos

Separación



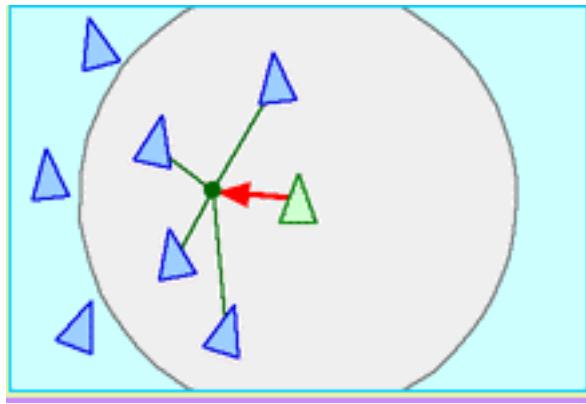
- intentar coincidir en velocidad con los compañeros de bandada cercanos

Alineación



- intentar permanecer cerca de los compañeros cercanos.

Cohesión:



El resultado fue majestuoso y muy parecido al comportamiento de vuelo que vemos en el mundo real. Hoy en día, los ABM se utilizan en muchas disciplinas de investigación para descubrir resultados emergentes de individuos que interactúan de acuerdo con reglas individuales de comportamiento.

3.2 El Mundo de las hormigas

Una pequeña hormiga camina en el suelo arenoso de Arizona buscando comida. La comida es para alimentar a las crías que están siendo atendidas por otras hormigas en la colonia. Diferentes hormigas tienen diferentes tareas en la colonia y la hormiga que estamos siguiendo se encarga de abandonar el nido todos los días en busca de comida. El aspecto sorprendente de las colonias de hormigas es que existe una estructura organizacional que no está controlada por la reina o por un pequeño grupo de hormigas burocráticas, tampoco hay un plan o una lista de tareas que las hormigas estén siguiendo. No, la complejidad de la colonia de hormigas surge de las interacciones locales entre las hormigas. Las hormigas siguen un rastro de feromonas, que les indica que otras hormigas encontrarán comida y dejan caer las feromonas en su camino de regreso al nido. El uso de las feromonas son una forma indirecta en que las hormigas se comunican con las demás (“sigue mi rastro y entonces puedes encontrar comida”). El rastro de feromona se evapora a cierta velocidad, por lo tanto, sólo son de uso limitado. Si otras hormigas no siguen este camino el sendero desaparecerá. Pero cuando el rastro se ve reforzado por feromonas de otras hormigas, puede surgir una carretera de hormigas. En tal carretera, vemos un carril de hormigas sin comida siguiendo la señal de la feromona, y en el otro carril hormigas que traen comida al nido. Existe una enorme diversidad de especies de hormigas, todas las cuales tienen alguna variación en su organización social. Algunas especies de hormigas producen hormigas con diferentes características físicas, que se distinguen por su papel en la colonia: trabajadores, recolectores, soldados, etc. En otras especies de hormigas, todas las hormigas de la colonia son físicamente similares y pueden cambiar de roles cuando sea necesario. Por ejemplo, cuando un oso hormiguero mata a las hormigas que buscan comida, la colonia experimentará una reducción de los alimentos que se llevan a casa, lo que luego indica a otras hormigas que intercambien sus papeles y esto hace que el ambiente de las hormigas cambie, lo que posteriormente cambia el comportamiento de otras hormigas. Esta influencia indirecta de los agentes a través del cambio del medio ambiente se denomina estigmatización, otro ejemplo de estigmatización son los senderos digitales que desarrollamos cuando interactuamos con sitios web al comprar libros, alquilar dvds o escuchar archivos de música, dejamos información (“feromonas”) y recibimos recomendaciones de otros libros, DVD o música que a la gente “le gusta” ó que usted compró, alquiló o escuchó. Estas interacciones estigméticas pueden conducir a un refuerzo en las elecciones. Las películas populares que aparecen en YouTube tienden a atraer aún más espectadores, si queremos entender cómo ciertos libros, películas o canciones se vuelven tan

populares, debemos analizar las diversas formas en que los demás influyen y refuerzan las elecciones.

3.3 Modelo del Fuego



Muchos sistemas complejos tienden a exhibir un fenómeno conocido como “umbral crítico” (Stauffer y Aharony, 1994) o “punto de inflexión” (Gladwell, 2000). Esencialmente, el punto ocurre cuando un pequeño cambio en un parámetro resulta en un gran cambio en el resultado. Un modelo que claramente contiene un punto de inflexión es el modelo de un bosque que arde. Es un modelo fácil de entender y exhibe un comportamiento interesante. Además de ser interesante por derecho propio, el modelo de propagación de incendios forestales es muy relevante para otros fenómenos naturales como la propagación de una enfermedad, la filtración de petróleo en una roca o la difusión de información dentro de una población (Newman, Girvan y Farmer, 2002). Este modelo simple es altamente sensible a un parámetro. Al observar si un incendio arderá o no de un lado de un bosque a otro, el resultado depende principalmente del porcentaje del terreno cubierto por árboles. A medida que aumenta este parámetro, habrá poco o ningún efecto en el sistema durante mucho tiempo, pero de repente el fuego saltará por todos lados. Este es un punto crítico del sistema, saber que un sistema tiene puntos de inflexión puede ser útil por varias razones. Primero, si se sabe que un sistema exhibe un punto de inflexión, se sabe que continuar poniendo esfuerzo en el sistema, incluso si todavía no está viendo ningún resultado, aún puede dar frutos. En segundo lugar, si se sabe dónde está el punto de inflexión y

si sabe lo cerca que está de él, entonces puede determinar si vale la pena poner un esfuerzo adicional en el sistema. Si está lejos del punto de inflexión, puede que no valga la pena intentar cambiar el estado del sistema, mientras que si está cerca del punto de inflexión, puede tomar solo una pequeña cantidad de esfuerzo para hacer un gran cambio en el estado del sistema. El modelo de Fuego surgió de una serie de esfuerzos independientes para comprender el fenómeno de percolación. En la percolación, una sustancia (como el aceite) se mueve a través de otro material (como una roca), que tiene cierta porosidad. Broadbent y Hammersley (1957) plantearon este problema por primera vez, y desde entonces muchos matemáticos y físicos han trabajado en ello. Influenciados por modelos de autómatas celulares, introdujeron un modelo de percolación utilizando como ejemplo de un piedra porosa sumergida en un balde de agua. La pregunta en la que se centraron fue:

- ¿Cuál es la probabilidad de que el centro de la piedra se moje?

Un fuego que se mueve a través de un bosque puede considerarse como una especie de percolación donde el fuego es como el agua y el bosque es como la roca, con los lugares vacíos en el bosque análogos a la porosidad de la roca. Una pregunta similar a la de Broadbent y Hammersley es:

¿si se comienza con algunos árboles en llamas en un borde del bosque, ¿qué tan probable es que el fuego se propague al otro lado del bosque?

Muchos científicos crearon y estudiaron este modelo, en 1987, el físico danés y teórico de sistemas complejos Per Bak y sus colegas mostraron que la propagación del fuego dependía de un parámetro crítico, la densidad del bosque. Debido a que este parámetro surge naturalmente, puede surgir la complejidad del incendio espontáneamente y, por lo tanto, es un posible mecanismo para explicar la naturaleza del mundo y la complejidad que surge. Bak y sus colegas llamaron a este fenómeno “críticamente-autoorganizado” y lo demostró en una serie de contextos que incluyen como el más famoso, el surgimiento de avalanchas en montones de arena.

3.4 Bar El Farol



El Farol es un bar-restaurante en Canyon Road Santa Fe, en donde se suele tocar música irlandesa todos los jueves por la noche. En una época, el economista irlandés Brian Arthur, del Instituto Santa Fe, solía visitarlo dos veces a la semana para escuchar música que le traía recuerdos de su juventud. Pero a él no le gustaba hacerlo en medio de una multitud de personas y entonces el problema de Arthur cada jueves era decidir si la multitud en El Farol sería tan grande que el disfrute que recibiría de la música se vería compensada por la irritación de tener que escucharla en medio de apretujones y gritos. Arthur atacó la cuestión de asistir o no en términos analíticos y en el proceso llegó a conclusiones sorprendentes sobre la racionalidad en economía. Supongamos que hay 100 personas en la ciudad de Santa Fe, cada una de las cuales, como Arthur, quiere ir a El Farol a escuchar música irlandesa, pero ninguno de ellos quiere ir si el bar va a estar demasiado lleno, para ser específicos, supongamos que las 100 personas conocen la asistencia al bar para varias semanas anteriores. Por ejemplo, dicho registro podría ser 44, 78, 56, 15, 23, 67, 84, 34, 45, 76, 40, 56, 23 y 35 asistentes. Entonces, cada individuo emplea independientemente algún proceso para estimar cuántas personas aparecerán en el bar el próximo Jueves por la tarde. Los predictores típicos de este tipo pueden ser:

- el mismo número que la semana pasada (35);
- una imagen espejo alrededor de 50 de la asistencia de la semana pasada (65);
- un promedio redondeado de las últimas cuatro semanas de asistencia (39);
- lo mismo que hace dos semanas (23).

Supongamos que cada persona decide, de manera independiente, ir al bar si su predicción conduce a que que irán menos de 60 personas, de lo contrario, la persona se queda en casa. Para poder hacer esta predicción, cada persona tiene su propio conjunto individual de predictores y usa el más preciso actualmente para pronosticar la asistencia de la próxima semana a El Farol. Una vez que se ha tomado el pronóstico y la decisión de asistir a cada persona, las personas convergen en la barra, y la nueva cifra de asistencia se publica al día siguiente.

en el periódico. En este momento, **todos** actualizan sus predicciones para la siguiente semana. Este proceso crea lo que podría llamarse Una “ecología” de predictores. El problema que enfrenta cada persona es pronosticar la asistencia con la mayor precisión como sea posible, sabiendo que la asistencia real estará determinada por los pronósticos que otros hacen, esto lleva inmediatamente a un tipo de “yo creo que tu crees que ellos creen...”, supongamos que alguien se convence de que asistirán 87 personas. Si esto es cierto la persona asume que los demás son igual de inteligentes, entonces es natural suponer que lo harán también y entonces ve que 87 es un buen pronóstico. ¡¡Pero entonces todos se quedan en casa, negando la exactitud de este pronóstico!! , la lógica deductiva falla. Por lo tanto, desde un punto de vista científico, el problema se reduce a cómo crear una teoría para saber cómo las personas deciden si ir a El Farol o no el jueves por la noche. Arthur no tardó mucho en descubrir que parece ser muy difícil un proceso de decisión en términos matemáticos convencionales. Entonces decidió modelar en su computadora el problema , para poder estudiar cómo las personas actuarían en esta situación. Lo que él quería hacer era entender cómo razonan los humanos cuando las herramientas de lógica deductiva no son suficientes. Los experimentos de Arthur demostraron que si los predictores no son demasiado simplistas, entonces el cantidad de personas que asistirán fluctúa alrededor de un nivel promedio de 60.

3.5 Tortugas y Ranas



Érase una vez, en una tierra lejana, un estanque habitado por tortugas y ranas. Era un estanque muy feliz: las tortugas convivían con las ranas y las ranas con las tortugas. Todos se llevaban muy bien, por muchos años las tortugas y las ranas

dividieron el estanque casi como si fuera un tablero de ajedrez, en un nenúfar vivía una rana, en el siguiente nenúfar una tortuga, en el siguiente otra rana, luego otra tortuga, y así sucesivamente. Había una buena simetría en ello. Cada tortuga tenía ocho vecinos, cuatro tortugas y cuatro ranas. Del mismo modo, cada rana tenía cuatro ranas vecinas, vecinos y cuatro tortugas vecinas. Una noche oscura, una terrible tormenta golpeó el estanque, las fuertes lluvias cayeron sobre el estanque, el viento azotaba volteando nenúfares, Las tortugas y las ranas fueron arrojadas por todo el estanque. Varias de los criaturas murieron al estrellarse contra las rocas. A la mañana siguiente las lluvias habían cesado y el sol se asomó, a través de las nubes. Las tortugas y las ranas inspeccionaron el daño, Los nenúfares estaban esparcidos por todo el estanque, pero afortunadamente estaban en gran parte intactos. Las tortugas y las ranas pasaron algún tiempo reorganizando los nenúfares en una ordenada serie, como lo habían estado antes. Entonces las criaturas se dispusieron a encontrar nuevos lugares para vivir, cada uno en busca de un nenúfar al que pudieran llamar su hogar.

*¿Cómo eligieron entre todos los nenúfares?

Los nenúfares eran casi idénticos, así que eso no fue un factor, las criaturas eran bastante tolerantes, a las ranas no les importaba vivir al lado de las tortugas, y a las tortugas no les importaba vivir al lado de las ranas. Pero cada tortuga quería asegurarse de tener al menos algunas otras tortugas cercanas. Y de manera similar, cada rana quería asegúrese de al menos otras ranas cerca, todos habían estado contentos con el acuerdo anterior, pero no estaban muy seguros de cómo volver a crearlo. No había nadie a cargo para decirles a dónde ir. Entonces las tortugas comenzaron a gatear (y las ranas a saltar) esperando encontrar nenúfares donde estarían contentos. Cada tortuga esperaba encontrar un nenúfar donde al menos el 30 por ciento de sus vecinos fueran tortugas. Si menos del 30 por ciento de sus vecinos eran tortugas, buscaría un nenúfar vacío cerca y se movería allí, saltando, con la esperanza de encontrar más vecinos tortuga. Si 30 por ciento o más de los vecinos eran tortugas, se asentaría. Pero, por supuesto, si el vecindario había cambiado, llevando el porcentaje de tortuga por debajo del 30 por ciento, tendría que comenzar a moverse de nuevo. (Cada rana siguió una estrategia análoga, esperando encontrar un nenúfar donde al menos el 30 por ciento de sus vecinos eran ranas). Después de un tiempo, todas las criaturas encontraron nenúfares donde estaban contentas. Cada criatura tenía al menos el 30 por ciento de “su propio tipo” como vecino. **Pero el estanque parecía bastante diferente al de antes.** Muchas de los tortugas no tenían ranas como vecinas, y muchas de las ranas no tenían tortugas como vecinas. “Ranita” era una de las ranas sin vecinos tortugas. Ella estaba muy preocupada y confundida. Le preguntó a su amigo “Ganso” que volaba por los aires si el estanque, visto desde arriba, se veía muy diferente al de antes de la tormenta.

- “Seguro que sí”, dijo “Ganso”. “Solía mirar el estanque mientras volaba por encima. Había tal mezcla uniforme de ranas y tortugas. Ahora el estanque parece tan segregado. las ranas viven en algunos grupos, y las tortugas viven en otros grupos, hice un pequeño cálculo mientras volaba sobre el

estanque, conté todos los vecinos para todas las tortugas, y descubrí que más de 70% de los vecinos de las tortugas son otras tortugas. Y fue lo mismo para ustedes ranas, más del 70% de tus vecinos son otras ranas. yo pensé que las ranas debieron haber tenido una terrible pelea con las tortugas después de la tormenta.”

- “No lo entiendo”, suspiró “Ranita” . “No tuvimos una pelea. Solo queríamos tener algunos vecinos como nosotros.No queríamos separarnos. ¿Que podría haber ocurrido?”

Si solo “Ranita” hubiera tenido acceso a NetLogo, podría haber obtenido una mejor comprensión de lo que sucedió. A continuación se muestra una simulación que recrea la historia de las tortugas y las ranas.

3.6 El Moho



El moho no es la criatura más glamorosa, pero seguramente una de las más extraños e intrigantes. Mientras la comida sea abundante, las células de moho existen independientemente como pequeñas amebas, se mueven alrededor, se alimentan de bacterias del medio ambiente y se reproducen simplemente dividiéndose en dos. Pero cuando la comida escasea, el comportamiento del moho cambia dramáticamente, las células del moho dejan de reproducirse y se mueven una hacia las otras, formando grupos formados por decenas de miles de células. En este punto, las células del moho comienzan a actuar como un todo unificado, en lugar de comportarse como muchas criaturas unicelulares, actúan como una sola criatura multicelular, cambian de forma y comienzan a gatear, buscando un

ambiente más favorable. Cuando encuentran un lugar a su gusto, se convierten en un tallo que sostiene una masa de esporas. Estas esporas finalmente se desprenden y se extienden por todo el nuevo entorno, comenzando un nuevo ciclo. El proceso a través del cual las células de moho se agregan en una sola criatura multicelular ha sido objeto de debate científico. Hasta 1980, más o menos, la mayoría de los biólogos creían que unas células especializadas “marca-pasos” coordinaban la agregación. Pero los científicos ahora ven la agregación de moho como un proceso muy descentralizado. Según las teorías actuales, las células del moho son homogéneas: ninguna se distingue por ninguna característica especial o comportamiento. La agrupación de células de moho **no surge** de los comandos de un líder pero a través de interacciones locales entre miles de células idénticas. De hecho, el proceso de agregación de moho es ahora visto como uno de los ejemplos clásicos de comportamiento autoorganizado.

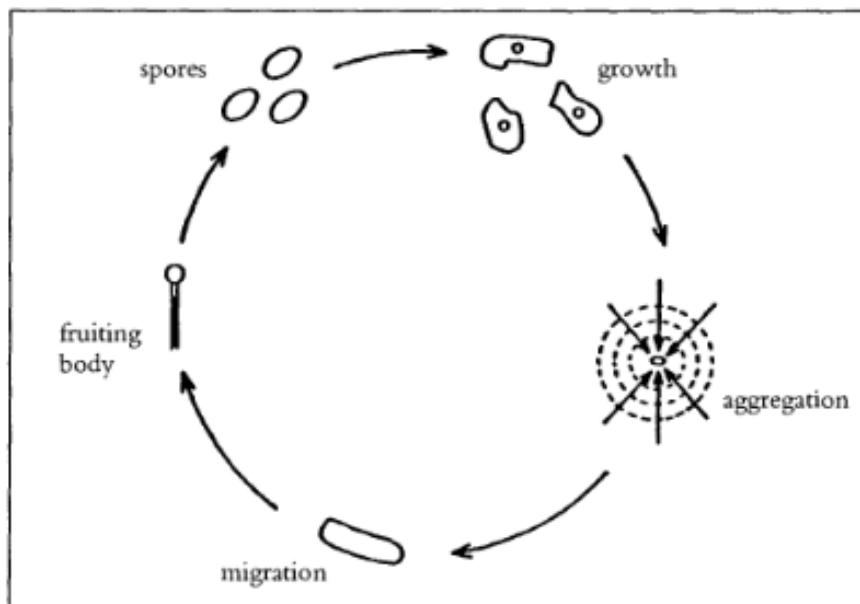


Figure 3.1

Life cycle of slime mold, reproduced from Prigogine and Stengers 1984

- ¿Cómo se agregan las células del moho ?

El mecanismo tiene que ver con un químico llamado “AMP cíclico” o AMPc (Goldbeter y Segal 1977). Cuando las células del moho pasan a su fase de agregación, producen y emiten AMPc en el medio ambiente, también se sienten atraídos por el mismo químico. A medida que las células se mueven, siguen el gradiente del AMPc. Es decir, prueban a su alrededor y se mueven en la dirección donde la concentración de AMPc es más alta, e este proceso es muy local. Cada

célula puede detectar AMPcs solo en su vecindad inmediata, no se puede saber cuánto AMPc puede existir a unos centímetros de distancia. Las células del moho producen el AMPc en pulsos periódicos. Como resultado, el moho tiende a unirse en ondas concéntricas, pero esta periodicidad no parece esencial para el proceso de agregación. De hecho, Prigogine y Stengers 1984) describen cómo las larvas de ciertos escarabajos (*Dendroctonus-micans*) se agregan en grupos utilizando un mecanismo similar al utilizado por el moho.

3.7 Trancones de Tráfico

(Adaptación del libro Resnick , M. (1994b). Turtles, Termites, and Traffic Jams)



Cuando Ari y Fadhl comenzaron a trabajar con NetLogo, estaban a la vez tomando una clase de educación vial. Cada uno había cumplido 16 años un poco tiempo antes, y estaban entusiasmados por obtener su licencia de conducción. Gran parte de su conversación se centró en los automóviles. Entonces cuando le di a Ari y Fadhl una colección de artículos para leer, no es sorprendente que un Artículo científico titulado “Flujo de tráfico vehicular” (Herman y Gardels 1963) capturó su atención. El flujo de tráfico es un dominio amplio para estudiar el comportamiento colectivo. Las interacciones entre automóviles en un flujo de tráfico pueden conducir a sorprendentes fenómenos. Considere un camino largo sin calles transversales o intersecciones.

- ¿Qué pasa si agregamos algunos semáforos a lo largo del camino?

Las luces de tráfico parecerían no tener ningún propósito constructivo. Sería natural suponer que los semáforos reducirían el rendimiento general del tráfico (número de automóviles por unidad de tiempo). Pero en algunas situaciones, los semáforos realmente mejoran el rendimiento general del tráfico. En la ciudad de Nueva York la Autoridad Portuaria, por ejemplo, descubrió que podría aumentar el tráfico a través del rendimiento en el “Túnel de Holanda” en un 6 por ciento al

detener deliberadamente coches antes de entrar en el túnel (Herman y Gardels 1963). Los estudios tradicionales del flujo de tráfico se basan en tecnología analítica sofisticada, técnicas como la teoría de colas. Pero muchos fenómenos de tráfico se pueden explorar con simples programas de simulación. Ari y Fadhl decidieron crear una carretera de un solo carril. (Más tarde, experimentaron con múltiples carriles) Ari sugirió agregar un radar de la policía en algún lugar a lo largo del camino, para parar coches que superan el límite de velocidad. Pero también quería que cada auto tuviera su propio detector de radar, por lo que los automóviles deberían reducir la velocidad cuando se acercaran al radar. Después de una discusión, Ari y Fadhl decidieron que cada agente (automóvil) debía seguir tres reglas básicas:

- Si hay un automóvil cerca de usted, reduzca la velocidad.
- Si no hay ningún automóvil cerca de usted, acelere (a menos que ya esté moviéndose al límite de velocidad).
- Si se detecta una trampa de radar, reduzca la velocidad.

Estas reglas se pueden implementar fácilmente y Ari y Fadhl esperaban que se formara un embotellamiento detrás de la trampa de radar, y de hecho lo hizo (figura 3.6). Después de unas pocas docenas de iteraciones, una línea de automóviles comenzó a formarse a la izquierda de la trampa de radar. Los automóviles se movieron lentamente a través de la trampa, luego se alejaron rápidamente tan pronto como lo pasaron. Ari explicó:

“El primer auto se ralentiza por la trampa del radar, luego la que está detrás se ralentiza, luego la que está detrás, y así se tiene un embotellamiento”. El único efecto inesperado fue la rápida aceleración de los autos cuando se movieron más allá del radar. La trampa del radar, en efecto, organizó los autos para una aceleración máxima. Cuando los autos disminuyeron la velocidad para la trampa del radar, formaron una cola con distancias aproximadamente iguales entre cada automóvil. Entonces cuando los autos se movieron más allá de la trampa del radar, no interferían entre sí. Los autos fueron “liberados” por la trampa del radar uno por uno, y aceleraron suavemente hasta que alcanzaron el límite de velocidad.



Ari y Fadhl notaron que un accidente al costado del camino tiene el mismo efecto que la trampa del radar, argumentaron que incluso una pequeña interrupción podría causar un atasco. Fadhl explicó:

“Cuando un automóvil en la carretera incluso toca el frenos, las luces de freno se encienden y la persona detrás de él no quiere golpearlo, así que se ralentiza un poco más, y la persona detrás de él un poco más, y la persona detrás de él más, y terminas teniendo un atasco de tráfico. Y el primer automóvil ni siquiera bajó

la velocidad”.

Le pregunté a Ari y Fadhl qué pasaría si solo algunos de los autos tuvieran detectores de radar. Ari predijo que solo algunos de los autos frenarían a la trampa del radar. Fadhl tenía una idea diferente:

“Los que tienen detectores de radar se ralentizarán, lo que hará que los otros disminuyan”. Ari encontró ese argumento convincente y rápidamente cambió su manera de pensar. Y, de hecho, Fadhl tenía razón. Se modificó la simulación para que solo el 25 por ciento de los automóviles tuvieran detectores de radar. El resultado, el flujo de tráfico se veía exactamente igual que cuando todos los autos tenían radar detectores

- ¿Qué pasaría si ninguno de los autos tuviera detectores de radar o, equivalentemente, si las trampas de radar fueron eliminados por completo? Sin trampa de radar, los autos estarían controlados por solo dos reglas simples:
 - si ve otro automóvil muy cerca, ve más despacio.
 - si no, acelera.

Las reglas no podrían ser mucho más simples. Fadhl predijo que el flujo de tráfico sería uniforme: los automóviles estarían espaciado uniformemente, viajando a una velocidad constante. Después de todo, sin la trampa del radar, ¿qué podría causar un atasco? Pero rápidamente se cambió de opinión y se predijo que se formaría un atasco de tráfico. Pero Ari y Fadhl reconocieron que tal situación sería difícil de establecer en el mundo real. Las distancias entre los autos tenían que estar iguales, y los autos tenían que comenzar exactamente al mismo tiempo, como un pelotón de soldados que comienzan a marchar al unísono. Se introdujo algo de aleatoriedad en las condiciones iniciales y los atascos regresaron. Mirando más de cerca, Ari y Fadhl notaron que los atascos no se quedaban en un lugar pero tendían a **moverse** con el tiempo. De hecho, los embotellamientos tienden a moverse **hacia atrás**, a pesar de que todos los autos dentro de ellos se movían hacia adelante. Fadhl lo describió:

“El atasco se mueve hacia atrás. Si ustedno pierdas de vista un auto, deja el atasco, pero el atasco en sí, donde se ven los autos amontonarse, retrocede”. El movimiento hacia atrás del atasco destaca una idea importante, las estructuras colectivas (como los atascos) a menudo se comportan de manera muy diferente a los elementos que los componen. Esta idea es cierta no solo para el tráfico sino para una gama mucho más amplia de fenómenos. El hecho de que Ari y Fadhl hallan desarrollado fuertes intuiciones sobre el flujo de tráfico mientras trabajaba en su proyecto StarLogo se debió, en gran parte, a sus profundos intereses y experiencias con autos.

3.8 Ecología basada en agentes

(Adaptación del libro Resnick , M. (1994b). Turtles, Termites, and Traffic Jams)



“Si no sabes a donde vas, podrías terminar en otro lugar”, pero existe un corolario, lo digo por experiencia: “Incluso si crees que sabes a dónde vas, probablemente terminarás algún en algún otro lugar.” Eso le sucedió a Benjamin, un estudiante , cuando se propuso crear un modelo que simulara la evolución por selección natural. En la colección de artículos que le había dado estaba un artículo de Scientific American (Dewdney 1989) sobre un programa de computadora llamado Evolución Simulada, Benjamin, que acababa de terminar su tercer año de secundaria, leyó el artículo y decidió que quería crear algo similar al programa comercial descrito en el artículo. Su objetivo era crear un conjunto de criaturas informáticas que interactuaran y evolucionaran. En el centro de la simulación de Benjamin había ovejas y comida. Su idea básica era simple: las ovejas que comen mucha comida se reproducen y las ovejas que no coman suficiente comida mueren. Finalmente, planeó agregar “genes” a sus ovejas, diferentes genes podrían proporcionar ovejas con diferentes niveles de “ajuste” o “aptitud” (quizás capacidades diferentes para encontrar comida). **Pero nunca llegó a los genes**, más bien, en el camino a la evolución, Benjamin se desvió en una exploración interesante de sistemas ecológicos, Benjamin comenzó haciendo que la comida creciera al azar a lo largo del Mundo (Durante cada paso de tiempo, cada parcela tenía una posibilidad aleatoria de cultivar algo de comida.) Luego creó algunas ovejas. las ovejas tenían capacidades sensoriales muy escasas. No podían “ver” o “oler” la comida a distancia. Podían sentir la comida solo cuando estuvieran “sobre” ella. Entonces las ovejas siguieron una estrategia muy simple:

- dada oveja arranca con 20 unidades de energía
- deambulan aleatoriamente, comiendo cualquier comida que se encuentre.

Benjamin le dio a cada oveja una variable de “energía”. Cada vez que una oveja daba un paso, su energía disminuyó un poco. Cada vez que comía algo de comida, su energía aumentaba Luego Benjamin agregó una regla más:

- si la energía bajó a cero, la oveja muere.

Benjamin simulo su modelo con 300 ovejas. Pero el medio ambiente no podra soportar tantas ovejas, no había suficiente comida. Así que algunas ovejas comenzaron a morir. La población de ovejas cayó rápidamente al principio, luego se fue nivelando a unas 150 ovejas. El sistema parecía alcanzar un estado estable con 150 ovejas: tanto el número de ovejas como la densidad de los alimentos permaneció más o menos constante. Entonces Benjamin probó el mismo programa con 1,000 ovejas. Sí había suficiente comida para 300 ovejas, ciertamente no habría suficiente para 1,000 ovejas. Así que Benjamin no se sorprendió cuando la población de ovejas comenzó a caer. Pero se sorprendió de que Después de un tiempo, solo quedaban 28 ovejas. Benjamin estaba perplejo:

- “Si se comenzó con más, ¿por qué deberíamos terminar con menos?”

Después de un tiempo, se dió cuenta de lo que había sucedido. Con tantas ovejas, la escasez es aún más crítica que antes. El resultado: inanición masiva. Notó que muchas de las ovejas morían casi al mismo tiempo, Supuso que estas ovejas casi no habían comido nada (Murieron cuando su suministro inicial de energía se agotó). Le sugerí un pequeño cambio, que cada oveja comenzara no con 20 unidades de energía, sino con una cantidad aleatoria entre 0 y 20 (Eso requería solo un pequeño cambio en el modelo: cambiar 20 a “aleatorio 20”) Aunque la población general de ovejas comenzaría con menos energía,

- ¿podrían sobrevivir más ovejas a la larga?

Benjamin comprendió de inmediato la idea. explicó: “Los que mueren rápido son los que tienen menos energía (inicial) y dejan más comida. Ellos no desperdician la comida comiéndola y simplemente muriendo”. Benjamin predijo que más ovejas sobrevivirían a la larga, y efectivamente 97 ovejas sobrevivieron (en comparación con las 28 anteriores). Benjamin entendió lo que había sucedido, pero aún encontraba el comportamiento un poco extraño:

“Las ovejas tienen menos energía inicial como grupo, y menos a veces es más”.

Luego Benjamin decidió agregar reproducción a su modelo. Su plan:

Cada vez que la energía de una oveja aumenta por encima de cierto umbral, la oveja debería “clonarse” y dividir su energía con su nuevo gemelo. Benjamin supuso que la regla para la clonación equilibraría de alguna manera la regla de muerte, y conduciría a algún tipo de equilibrio, pensó:

“Con suerte, se equilibrará de alguna manera. Quiero decir que lo hará, Pero no sé en qué número se equilibrará”. Después de pensarla un poco más, sugirió que

el suministro de alimentos podría caer primero, pero luego volvería a subir y se estabilizaría:

“La comida se irá para abajo, muchos de ellos morirán, luego la comida subirá y se equilibrará” Benjamin comenzó a ejecutar el programa. Como esperaba, la oferta de comida bajó y luego subió. Pero no se equilibró: cayó y arriba otra vez, y otra vez, y otra vez. Mientras tanto, la población de ovejas también oscilaba, pero en una fase diferente a la comida. Estas oscilaciones son característico de situaciones como depredadores (en este caso, ovejas) y presas (en este caso, comida).

Chapter 4

Algo de Historia

Muchos campos diferentes han aportado ideas y métodos a MOBA, nos centraremos en los antecedentes claves de las tecnologías MOBA de la informática y campos computacionales asociados. Además de los campos computacionales, notamos que también ha habido fuertes contribuciones de biología, física, ingeniería y ciencias sociales. En biología, estas contribuciones vinieron en gran parte de la ecología, el comportamiento individual de animales o plantas en un ecosistema, en lugar de trabajar con variables de nivel de población (DeAngelis y Mooij, 2005) trabajan con modelaje de lo individual, los modelos a menudo se ubican dentro de paisajes o entornos particulares, y los modeladores de MOBA se centran en cómo los paisajes pueden afectar en gran medida los resultados de un modelo (Grimm & Railsback, 2005). Gran parte de la metodología de MOBA se ha incorporado al trabajo en física, para describir el magnetismo y mostraron que modelos simples podrían producir transición de fase. Estos modelos también fueron precursores de la telefonía celular. el físico Per Bak creó el clásico modelo de las montañas de arena, que usó para ilustrar el concepto de autoorganización crítica, pasó a aplicar métodos basados en fenómenos complejos(como el modelo del Fuego y varios modelos bursátiles)



Per Bak En ingeniería, ingeniería de procesos y cibernética, entre otras áreas de investigación, se ha contribuido al desarrollo de MOBAS En ingeniería de procesos, el objetivo es diseñar un óptimo resultados dados comportamientos de bajo nivel, que pueden considerarse como un marco similar al de los modelos basados en agentes, pero con un objetivo diferente, algunas de las herramientas y métodos de ingeniería de procesos también son útiles en el contexto de MOBA, Esto lleva a las nociones de cambios de fase de un sistema, es decir, áreas de espacio de parámetros donde pequeños cambios pueden tener consecuencias desproporcionadas (Wiener, 1961). Los modelos tienen que lidiar con la estocasticidad, lo que llevó a la técnicas de simulación de Monte Carlo (Metropolis & Ulam, 1949), al igual que con los ingenieros de procesos y cibernéticos, los científicos sociales se dieron cuenta de que las complejidades de las organizaciones sociales no eran capturadas suficientemente por los modelos y herramientas disponibles como resultado, los científicos sociales comenzaron a usar modelos algorítmicos y computacionales para describir los fenómenos sociales, con un método para comparar datos empíricos con datos por predecir similares a los de científicos naturales (Lave y marzo de 1975). Los modelos de teoría de juegos de Nash (1950) fueron algunos de los primeros intentos de capturar el comportamiento humano en ecuaciones matemáticas. La mayoría de estos modelos iniciales de ciencias sociales utilizaron datos agregados promediados entre individuos. Por ejemplo, el famoso modelo del Club de Roma hizo suposiciones muy simples sobre cómo el mundo y la población aumentaría y cómo se consumirían los recursos (Meadows, 1972).



John Nash Estos esfuerzos dieron lugar con el tiempo al modelado dinámico de sistemas (SDM; Forrester, 1961), que propuso crear modelos utilizando existencias (cantidades de bienes, entidades u objetos en ubicaciones particulares) y flujos (las tasas de aumento o disminución de estas existencias).los modeladores hicieron contribuciones importantes a los sistemas complejos, pero los elementos granulares de estos modelos eran cantidades agregadas, un paso lógico era bajar un nivel, modelando los individuos que componen estos agregados. Por lo tanto, el modelado basado en agentes fue una progresión natural que permitió un examen más profundo del comportamiento de individuos heterogéneos. De hecho, en parte debido a esta necesidad de ciencia para comprender el comportamiento individual en un marco cada vez más rico, algunas de los primeros modelos basados en agentes surgieron en las ciencias sociales. El modelo de segregación de Schelling (discutido en el capítulo 3) es considerado por muchos como uno de los primeros modelos basados en agentes que se creó, a pesar de que se realizó manualmente usando un tablero de ajedrez y monedas (Schelling, 1971). El libro de Schelling *Micromotives and Macrobehavior* (1978) demostró cómo las acciones a nivel individual podrían dar lugar a sorprendentes patrones sociales. Su modelo de segregación mostró que la segregación de la vivienda ocurriría incluso si ningún individuo lo quiere, siempre y cuando las personas tengan preferencia y no pertenezcan a una minoría extrema en su barrio.



Thomas Schelling

En la última década, hemos sido testigos de un tremendo crecimiento en el campo de la teoría de redes y la incorporación de soporte para redes como un elemento central en el modelado basado en agentes. Este trabajo en redes (grafos) fue iniciado por el matemático Euler en el siglo XIX para resolver el problema de los siete puentes de Konigsberg (ver Newman, 2010). En la década de 1950 y principios de los 60, los matemáticos Erdosy Renyi caracterizaron las redes aleatorias (1960), y esto fue seguido por el modelo de red preferencial de Albert Barabasi(1999). Los trabajos de psicólogos como Stanley Milgram sugirieron que la longitud de camino promedio en humanos en las redes sociales son cortas (seis grados de separación) (1967), una idea que luego formalizaron Duncan Watts y Stevan Strogatz con las redes de mundos pequeños Small Worlds,1998). Este tipo de redes y sus métodos de análisis asociados se han convertido en elementos básicos del modelado basado en agentes. Una historia completa de las raíces de MOBA en otros campos y el papel de dichos campos en el origen de los MOBA está más allá del alcance de este capítulo, aquí, sin embargo, exploraremos algunos descubrimientos clave que dieron forma al desarrollo de MOBA, Presentamos aquí seis áreas importantes que describen varios de los principales antecedentes de las tecnologías asociadas a los MOBA

4.1 Autómatas celulares y modelado basado en agentes

John von Neumann contribuyó en gran medida a una gran cantidad de campos científicos, incluidos mecánica cuántica, teoría de juegos económicos y ciencias de la computación. De este matemático húngaro se dice que fue “el último de

los grandes matemáticos” poseía el “intelecto más centelleante de este siglo.” A finales de la década de 1940, von Neumann, inventó la arquitectura moderna de la computadora (Máquina de Von Neumann), se interesó en tratar de crear máquinas artificiales que podrían reproducirse de forma autónoma, influenciado por su trabajo en la primera computadora (ENIAC; von Neumann et al., 1987), creó una máquina autoreproducible que él llamó un autómata celular que usaba veintinueve estados diferentes (¡usando lápiz y papel cuadriculado!) que llamó una máquina universal (Burks, 1970). Aunque una máquina física aún no había sido construida, esta máquina fue prueba del concepto de que una máquina auto-reproductora podría ser construida, no solo construyó una máquina capaz de autorreplicarse, sino también una que podía evolucionar, ya que las instrucciones podrían modificarse y agregarse al comienzo de cada generación sucesiva para proporcionar más y más capacidades.



John Von Neumann Sin embargo, en 1970,

John Conway creó un celular mucho más simple un autómata que llamó el “Juego de la vida”, Debido a la falta de suficiente poder computacional, Conway utilizó un tablero físico para llevar a cabo sus experimentos y no una computadora. “Life” se hizo popular cuando Martin Gardner publicó el juego en su popular columna Scientific American (Gardner, 1970). El juego de la vida de Conway tiene tres reglas:

- (1) Si una celda tenía exactamente tres vecinos (de sus ocho vecinos inmediatos), pasaría de estar muerto a estar vivo (nacimiento).
- (2) Si tuviera dos o tres vecinos y estaba vivo, permanecería vivo (sin cambios)
- (3) Si tuviera alguna otra combinación de vecinos, iría al estado muerto (soledad o hacinamiento).

Cuando Conway alimentó el juego con un conjunto aleatorio de células vivas y muertas, el sistema comenzó a producir patrones hermosos e intrincados de objetos interesantes. Notablemente, aunque las regla de Conway requería solo dos estados, (vivos o muertos) y nueve entradas (la celda y sus ocho vecinos),

Conway probó que sus regla de “Vida” (Berlekamp et al., 1982) contenían el material necesario para la auto-reproducción, es decir, el sistema podría crear una entidad computacional que podría crearentidades computacionales adicionales. Sin embargo, el poder del Juego de la Vida no termina ahí. En 2009, Adam Goucher construyó una computadora / constructor universal dentro del juego de la vida de Conway y demostró que las reglas del juego de la vida eran suficientes calcular todos los posibles problemas (Hutton, 2010).



John Conway

Casi al mismo tiempo que algunos de los primeros trabajos de Conway, y después del trabajo de von Neumann fallecido en 1957, Arthur W. Burks, uno de los colaboradores de von Neumann, siguió examinando los autómatas celulares como parte del grupo Logic of Computers de la Universidad de Michigan. Eventualmente editó una colección de la mayoría de los documentos originales sobre autómatas celulares en un libro titulado Theory of Self-Reproducción (von Neumann y Burks, 1966). Burks también fue asesor de John H. Holland, del cual hablaremos más adelante. El trabajo de Burks preservó el estudio de los AC hasta que Stephen Wolfram lo revivió a principios de la década de 1980 (Wolfram, 1983). Wolfram hizo varias contribuciones notables al campo de los autómatas celulares, Por ejemplo, él realizó un estudio exhaustivo de todas las reglas unidimensionales para mundos bidimensionales, celdas que tienen solo vecinos izquierdo y derecho . Incluso esta simple clase de AC exhiben un comportamiento sorprendentemente complejo. Wolfram los dividió en cuatro clases:

- De estado final uniforme
- De estado final cíclico.
- De estado final aleatorio.
- De estado final complejo.

Wolfram demostró que estos cuatro tipos de patrones pueden generar muchos patrones encontrados en la naturaleza. Wolfram y Ed Fredkin del MIT desataron mucha controversia cuando hicieron la afirmación de que todo el universo puede modelarse utilizando AC o, aún más sorprendente, la posibilidad de que el universo mismo es un automata celular (Wolfram, 2002; Fredkin, 1990).



Stephen Wolfram

La relación histórica entre AC y MOBAs es algo desordenada. Muchos autores han escrito que las AC condujeron directamente al desarrollo de los MOBA. De hecho, las AC se pueden ver como MOBAs simples donde todos los agentes son estacionarios: en contraste con los AC, más generales Los MOBA también pueden incluir agentes que se mueven explícitamente. Su capacidad de tener agentes móviles, permite a los investigadores representar y modelar objetos en movimiento de forma más natural. Como resultado de esta similitud, se podría suponer que los MOBAs fueron una evolución natural de los autómatas celulares. Sin embargo, hay poca evidencia de que los desarrolladores de los primeros MOBAs; por ejemplo, Schelling y sus colegas comenzaron con una AC a partir de la cual hallan desarrollado un modelo basado en agentes, estos investigadores estaban al tanto del trabajo de AC, pero, a juzgar por entrevistas con varios de ellos, parece que desarrollaron la noción de modelado basado en agentes independiente del trabajo en curso de la teoría de AC. En muchos sentidos, las AC (y el Juego de la vida de Conway en particular) exhiben muchas de las propiedades de los modelos modernos basados en agentes. Cada celda se puede ver como un agente con una descripción simple del estado. Las acciones que toma para cambiar su estado en función de sus vecinos son similares a las acciones que los agentes toman en respuesta a las interacciones locales. Los AC de Von Neumann estuvieron entre los primeros intentos de crear modelos computacionales de un sistema biológico. Además, la escala de tiempo discreta de los ACes muy similar a la a los ticks de un MOBA, el modelo “Life” de Conway también exhibe muchos comportamientos que a menudo se ven en los Moba (**El resultado final del modelo es difícil de predecir sobre la base de las entradas iniciales, y son patrones emergentes que solo pueden describirse a un nivel superior**). Por último, el trabajo de Wolfram para clasificar las reglas de los AC en clases de comportamiento fue un ejemplo temprano de los intentos de la ciencia de la complejidad de entender nuestro mundo tomando grandes cantidades de fenómenos dispares y clasificándolos en grupos para ayudar a entender las similitudes entre sistemas que, a primera vista, parecen completamente no relacionados.

4.2 Algoritmos genéticos, John Holland y sistemas adaptativos complejos

En informática, los algoritmos se han diseñado tradicionalmente como artefactos de ingeniería. La inspiración para estos algoritmos provino de aplicaciones de

ingeniería como el ensamblaje de líneas de producción, construcción de puentes e incluso sistemas de alcantarillado. A finales de la década de 1960, un joven ingeniero eléctrico llamado John Holland se topó con el trabajo de Donald Hebb. En el libro de Hebb, Organización del comportamiento (1949), el psicólogo había escrito sus ideas sobre cómo las neuronas pueden considerarse algoritmos simples. Frank Rosenblatt (1962) fue influenciado mediante estas ideas para construir un modelo computacional de neuronas humanas, al que llamó perceptrón, Rosenblatt utilizó este modelo para resolver problemas informáticos convencionales de modos que él afirmó eran similares a como la mente humana resuelve los problemas. Esta técnica eventualmente resultaría desembocando en el campo moderno de las redes neuronales. En este naciente campo, Holland tuvo una idea poderosa:

se dio cuenta de que un perceptrón era esencialmente un modelo de cómo una neurona se adapta a sus entradas y salidas. Por lo tanto, el perceptrón no era solo un modelo estático de la mente, pero también, uno de **adaptación mental al mundo**. Holland comenzó a preguntarse si era posible hacer más universal este modelo de adaptación, específicamente, comenzó a pensar en la adaptación evolutiva y cómo podría una computadora modelar la evolución. Presagiando un principio general de sistemas complejos, Holland intentó generalizar su modelo teórico de un dominio computacional y lo aplicó ampliamente a otros dominios. La adaptación parecía (al menos en algún nivel) existir en, por ejemplo, tanto en las neuronas en el cerebro, como en las especies se adaptan a los cambios climáticos. Mediante la extracción de la esencia de la adaptación a un principio universal, Holland arrojó nueva luz sobre cómo entender muchos sistemas adaptativos, encontrando una perspectiva muy importante, y hoy en día central, de los sistemas complejos. En 1975, John Holland publicó su monografía *Adaptation in Natural and Artificial Systems*, Este trabajo fue la culminación de sus primeros años en la investigación de la adaptación. y resultó en la idea del **algoritmo genético (AG)**. En esencia, los AG crean una población de soluciones a un problema, y evalúan qué tan efectivas son las soluciones para resolver el problema, luego combinan y mutan las mejores soluciones para crear una nueva población. Esta nueva población vuelve a ser evaluada y el proceso es repetido. El AG fue un desarrollo único en informática ya que postuló el uso de modelos biológicos para resolver problemas computacionales, evolucionando poblaciones de soluciones a problemas de búsqueda.

Holland usó los AG para programar computadores para jugar. Sus pensamientos sobre las computadoras y el juego estuvieron muy influenciados por el trabajo de Samuel en el aprendizaje automático y el juego de damas (Samuel, 1959). Holland pensó que las computadoras podrían resolver problemas aún más complejas si los programas para jugar no solo pudieran cambiar sus acciones sino también sus estrategias a lo largo del tiempo, imagino la evolución de estrategias para jugar. Estas ideas condujeron naturalmente a la construcción de agentes adaptativos que pueden cambiar y adaptarse a sus alrededores. Holland comenzó su estudio de los agentes adaptativos con tanto ahínco a como anteriormente había estudiado la evolución, comenzó con datos del mundo real y luego desarrolló modelos informáticos de estos sistemas, desde el principio (alrededor de 1985) comenzó a

4.2. ALGORITMOS GENÉTICOS, JOHN HOLLAND Y SISTEMAS ADAPTATIVOS COMPLEJOS57

trabajar con el Instituto Santa Fe (SFI), un instituto de investigación dedicado al estudio de sistemas complejos. En una reunión en el Instituto Santa Fe en septiembre 1987, Holland presentó sus ideas de cómo la economía entera podría ser vista como un sistema adaptativo complejo compuesto por agentes adaptativos (Holland, 1995) y conoció a Brian Arthur, y influenciado por el trabajo de Holland desarrolló uno de los primeros modelos en economía que empleaban la modelación basada en agentes. Durante este tiempo, muchos investigadores del Instituto Santa Fe, como Holland, Arthur, Anderson, Arrow y Pines, se interesaron en modelar la economía como un complejo sistema evolutivo. Este esfuerzo de investigación resultaría en uno de los primeros modelos clásicos basados en agentes, el Mercado de Valores Artificiales del Instituto Santa Fe (Arthur et al,1997).



John Holland

Holland también influenció desde el principio a un grupo de sus compañeros de la Universidad de Michigan: Burks, Axelrod, Cohen y Hamilton. Todos estos investigadores (que juntos se llamaron ellos mismos, el grupo BACH) serían influyentes en el campo de los sistemas complejos, y algunos incluso incursionaron en el modelado basado en agentes. Axelrod creó el famoso dilema del prisionero (Prisoner Dilemma) (Axelrod, 1984) Holland comenzó a investigar una generalización del algoritmo genético, El **sistema clasificador**, basado en el principio de una jerarquía predeterminada, el sistema clasificador evolucionaría reglas o “clasificadores” que traducirían entradas en salidas para satisfacer un objetivo. El sistema clasificador contenía una población de reglas simples que funcionaban juntas para modelar fenómenos complejos. Esto es similar a la forma en que un MOBA contiene una población de agentes simples que trabajan juntos para modelar un fenómeno complejo. Por otra parte, el sistema clasificador fue un primer intento de crear agentes verdaderamente adaptativos, ya que contenía un algoritmo para que los agentes desarrollaran estrategias arbitrariamente complejas. En los años siguientes, Holland trabajó con dos estudiantes de posgrado, Melanie Mitchell, y Stephanie Forrest. Juntos, comenzaron a explorar el uso

del algoritmo genético en diversos escenarios relacionados con la vida artificial (Mitchell y Forrest, 1994). Mitchell demostró (1998) que los algoritmos genéticos podrían encontrar mejores soluciones para los problemas de autómatas celulares clásicos. Luego realizó una investigación básica y escribió un libro de texto introductorio sobre sistemas complejos (2009) que describe la relación entre complejidad y computación, evolución e inteligencia artificial, Forrest continuaría expandiéndose sobre estas ideas y creando sistemas inmunes artificiales (AIS; Hofmeyr y Forrest, 2000).

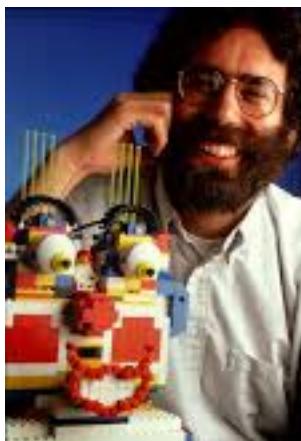


Melanie Mitchell

4.3 Seymour Papert, Logo y la tortuga

A fines de la década de 1960, Seymour Papert fue nombrado codirector del laboratorio Inteligencia Artificial del MIT junto con su colega Marvin Minsky. Papert había llegado al MIT después de terminar su doctorado en matemáticas en Cambridge y pasó varios años estudiando el pensamiento matemático de los niños con Piaget en el Instituto Jean Piaget en Ginebra, Suiza. Poco después de su llegada al MIT, comenzó a colaborar con científicos y crearon un lenguaje de programación, Logo, diseñado para ser utilizado por niños. Logo heredó gran parte de su forma y sintaxis del lenguaje Lisp, que se usó prominentemente en el trabajo de Inteligencia Artificial. Papert creía que dar acceso a los niños a la programación les permitiría convertirse en pensadores y darles acceso a ideas poderosas (Papert, 1980). Llamó a este enfoque educativo construcciónismo. Después de trabajar con Logo por un año, Papert inventó la “tortuga”, un objeto que estaba controlado por el lenguaje Logo. Algunas tortugas tempranas eran mecánicas (Walter, 1950) y estaban atadas a la computadora. pero pronto la tortuga se volvió virtual. Las tortugas tenían propiedades como la ubicación en la pantalla, una “cabeza” (dirección) a la que estaban mirando y una pluma para dejar huellas. Cada tortuga obedeció comandos, como “avanzar”, “girar a la

izquierda”, etc. Al emitir comandos a la tortuga, los niños podían dibujar figuras geométricas clásicas, así como repetitivas y recursivas (fractales) Posteriormente, el lenguaje LOGO se introdujo en las escuelas, con la participación de millones de niños en todo el mundo, mucho del éxito de LOGO se atribuyó a la tortuga. Papert describió a la tortuga como un “cuerpo sintonico” (1980): un usuario podría “proyectarse” en la tortuga y de esta manera descubrir qué comandos la tortuga debe efectuar, los usuarios podrían imaginar lo que harían con sus cuerpos para lograr el efecto deseado. Entonces, para dibujar un cuadrado, el usuario avanzaría, giraría a la derecha, avanzar la misma cantidad, y así sucesivamente. En miles de aulas LOGO, niño jugaron con la tortuga y aprendieron mucho sobre figuras geométricas y de paso sobre cómo programar computadoras de esta manera. La tortuga LOGO puede haber sido el primer agente computacional en el sentido en el que pensamos hoy a los agentes de un MOBA. Al igual que los agentes de NetLogo, la tortuga Logo tiene una ubicación y dirección, y su poder se basa en que los usuarios puedan proyectarse en él, que es imaginar ser una tortuga. NetLogo tomó gran parte de su sintaxis de Logo y la expandió a miles de tortugas. Las tortugas NetLogo suelen “dibujar” con sus cuerpos en lugar de con sus bolígrafos, y es la configuración de sus cuerpos lo que crea una visualización de NetLogo (Wilensky, 2001). NetLogo también tomó prestado el eslogan **“umbral bajo, techo alto”** de LOGO, lo que significa que debería ser lo suficientemente simple para que los principiantes sean capaz de trabajar con él de inmediato y, sin embargo, lo suficientemente potente como para que los expertos realicen sus investigaciones con él (Tisue y Wilensky, 2004). Además, NetLogo se basó en la sintaxis de LOGO hacer que su lectura sea lo más cercana posible al lenguaje natural. La influencia de Papert y Logo puede verse no solo en NetLogo, sino también en la forma que muchas plataformas de modelado basadas en agentes conceptualizan a los agentes como entidades con sus propias propiedades y acciones. La mayoría de las primeras plataformas de modelado basadas en agentes, como Swarm (Minar et al., 1996) y Repast (Collier, 2001), consideraron dada la necesidad de visualizar agentes en una pantalla 2D, de la misma manera que la tortuga Logo se proyectó en una pantalla 2D.



Mitch

Resnick, Seymour Papert y la tortuga

4.4 Paralelismo de datos

A mediados de la década de 1980, Danny Hillis completó su tesis doctoral en el MIT sobre arquitectura de una computadora paralela que él llamó una “máquina de conexión.” A diferencia de la arquitectura clásica de Von Neumann, arquitectura de la mayoría de las otras computadoras en ese momento, la máquina de conexión (o CM) no manejaba todos sus cálculos a través de una sola unidad central de procesamiento (Hillis, 1989). En lugar de eso hacía uso de miles de procesadores de bajo costo y baja capacidad conectados entre sí de tal manera que cada procesador pudiera comunicarse con cualquier otro, el CM empleó una arquitectura que se llamaba, “Instrucción única, datos múltiples”, generalmente abreviado SIMD, que daba una misma instrucción a cada uno de los miles de procesadores, cada uno de los cuales contenía elementos de datos. Hillis fundó una empresa, Thinking Machines, para fabricar y vender la computadora. Al principio, la máquina se percibía como difícil de programar, ya que los lenguajes de programación estándar no funcionaban eficientemente en la máquina. A finales de la década de 1980, se crearon lenguajes paralelos como StarLisp (una versión paralela de Lisp) y C-Star para programar el CM. El CM-2 tenía 65.536 procesadores, y para usarlos eficientemente era importante no tener algunos procesadores esperando a otros. La forma más fácil de hacer esto era utilizar métodos de “paralelismo de datos”, en los que los datos se distribuían de manera uniforme en los procesadores, cada uno de los cuales ejecutaba simultáneamente la misma instrucción en sus datos. El físico Richard Feynman, junto con Stephen Wolfram, desarrollaron una aplicación de mecánica de fluidos de en un autómata celular (Tucker y Robertson, 1988), Feynman había demostrado que casi todos los flujos de fluidos podían modelarse independientemente del tipo de partícula, usando una red hexagonal. Entonces Feynman y Wolfram eligieron partículas esféricas para calcular el cambio en cada hexágono de la red usando paralelismo de datos, l resultado fue una hermosa visualización de fluidos turbulentos, este y otros ejemplosparecidos han sido tomados por modeladores basados en agentes y pueden ejecutarse en hardware en serie utilizando métodos MOBA. El paradigma paralelo de datos SIMD en última instancia no tuvo éxito en el mercado de hardware, pero, sus métodos de dar la misma instrucción a múltiples procesadoresinfluyeron los lenguajes para MOBAs, que también daban instrucciones únicas a múltiples agentes distribuidos. En esencia, el modelo paralelo de datos fue adoptado por MOBA.



Danny Hillis

4.5 Gráficos por computadora, sistemas de partículas y boids

Los gráficos por computadora han avanzado notablemente con el resto de la informática, a medida que las computadoras se han vuelto cada vez más rápidas, las pantallas de las computadoras han mejorado para renderizar visualizaciones cada vez más realistas e imágenes basadas en computadora. Los gráficos por computadora tienen como objetivo tomar una imagen dentro de la cabeza del autor y traducirla en una imagen o representación en el computador, esto tiene mucha similitud con el modelado por computadora, el objetivo es tomar un modelo conceptual y traducirlo en un artefacto computacional. Así que no sorprende que los gráficos por computadora hayan influido en el modelado basado en agentes. Al principio de los gráficos por computadora, muchos desarrolladores usaron una gran combinación de planos y superficies para representar objetos. Esto resultó ser una buena primera aproximación: después de todo, la vasta mayoría de nuestro rango visual generalmente está ocupado por superficies, como paredes y techos, el cielo o un camino delante de nosotros. Sin embargo, las superficies no son una representación adecuada para fenómenos menos claramente definidos como el humo o las estrellas o la luz (Blinn, 1982). Para modelar estos fenómenos, los desarrolladores de gráficos por computadora recurrieron a representaciones puntuales. Los puntos podrían tener tamaño, posición y velocidad y se creará una representación más natural para estos fenómenos que las superficies. Además, los puntos eran más fáciles de trabajar y sus reglas de movimiento eran más fáciles de tratar que las superficies. El enfoque basado en puntos para gráficos por computadora se conoció como sistemas de partículas (Reeves, 1983). Tales sistemas tienen mucho en común con el modelado basado en agentes porque

se pueden simular fenómenos emergentes visualmente , como el humo que sale de una chimenea, modelando las partículas individuales de humo y escribiendo reglas simples de cómo interactúan para visualizar el patrón global resultante. Modelos basados en agentes igualmente usan reglas simples de interacción basada en agentes y luego te permite observar el fenómeno global. Inspirado por el uso de sistemas de partículas y el trabajo antes mencionado sobre tortugas y LOGO, Craig Reynolds fue más allá del concepto de partículas y los usó para describir el movimiento de las aves que vuelan en bandadas (Reynolds,1987),Reynolds llamó a estas configuraciones Flocados (Flocks) y a las criatura voladoras Boids y usó tres reglas simples para describir su comportamiento:

- Los Boids de separación no deben acercarse demasiado a ningún otro objeto en el entorno.
- Los Boids de alineación deben dirigirse hacia el mismo rumbo que sus compañeros locales
- Los Boids de Cohesión deberían moverse hacia el centro de su grupo local.

A pesar de la simplicidad de estas tres reglas, Reynolds pudo lograr un vuelo (flocado) realista, de hecho la interacción local de estas reglas no solo permitió a Reynolds desarrollar un modelo que agrupaba los Boids a nivel global, sino que cada grupompequeño de BOIds se comportaba como uno solo,por ejemplo, si se introduce un obstáculo en el vuelo de los Boids estos pueden dividirse fácilmente en dos bandadas, moverse alrededor del objeto y luego volver a ensamblarse. Todo esto se logró solo con las tres reglas y sin ninguna modificación adicional al modelo.El modelo “Boids” fue en muchos sentidos un modelo basado en agentes, aunque este término no estaba en uso en esa época, el hecho de que los grupos pudieran adaptarse a situaciones novedosas (por ejemplo, la introducción de un objeto extraño) sin interrumpir el patrón emergente es un ejemplo clásico de cómo un MOBA bien escrito es generalizable más allá de las condiciones para las cuales fue concebido originalmente. Como los MOBA no requieren una descripción global del sistema, no necesitan anticipar todos los posibles eventos que pueden suceder. Poco después de la presentación de Reynolds del modelo Boids en SIGGRAPH '87, Chris Langton organizó el primer taller sobre Vida Artificial, donde el modelo de Reynolds también fue presentado. La vida artificial se ha convertido en una comunidad que abarca muchos de estos diferentes métodos de computación en los que se utilizan computadoras para emular humanos y sistemas biológicos La Vida Artificial, por lo tanto, comenzó a usar sistemas como los Boids de Reynolds,Los autómatas celulares de von Neumann y los algoritmos genéticos de Holland. Aunque la vida artificial es distinta del modelado basado en agentes en el sentido que su objetivo es construir sistemas que reflejen objetos realistas dentro de una computadora, muchos de los métodos y técnicas de MOBA y de vida artificial son bastante similares.



Craig Reynolds

4.6 Conclusión

los MOBAs han recorrido un largo camino desde los días de Thomas Schelling (1971) lanzando monedas en un tablero de ajedrez. Estas seis viñetas han intentado arrojar algo de luz sobre las raíces de este campo. Sin embargo, el campo aún es bastante joven. Existe una considerable investigación que aún no se ha llevado a cabo sobre cómo implementar mejor MOBAs, qué herramientas son más útiles para soportarlos, y dónde aplicarlos mejor. Está claro que los modelos basados en agentes se han convertido en un metodología y conjunto de herramientas importantes para comprender sistemas complejos en las ciencias naturales, las ciencias sociales y la ingeniería.

Part III

Un Modelo Básico

Chapter 5

Buscadores de Hongos

Estamos listos para construir nuestro primer modelo, este es el modelo de búsqueda de hongos que discutimos en la sección 1.2 en este momento es conveniente volver a leer la sección 1.2 para recordar en que consiste.

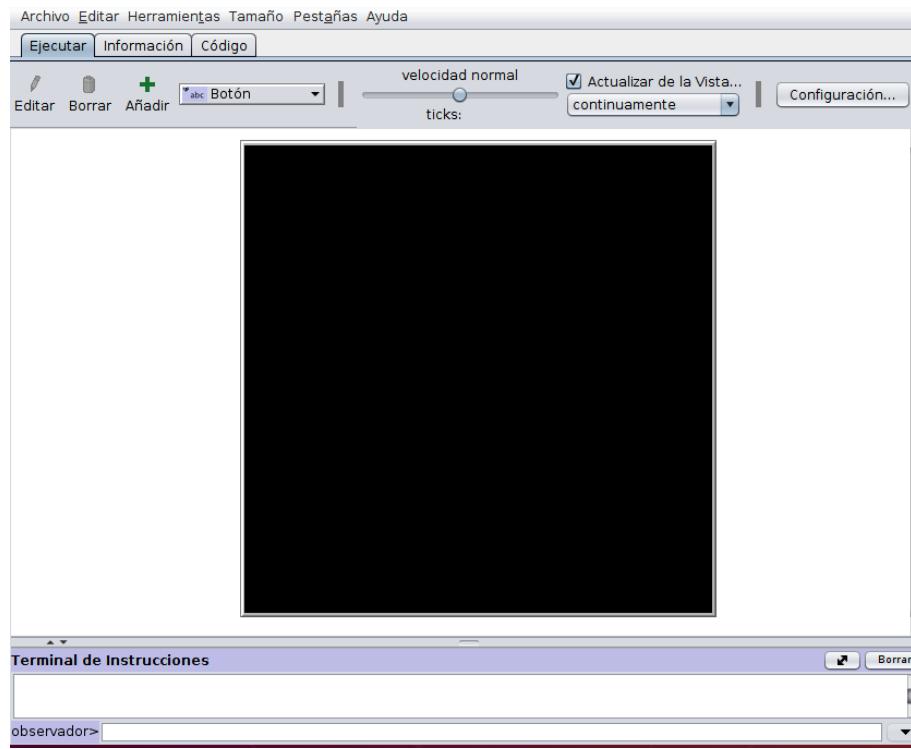
5.1 Construyendo el Modelo

5.1.1 Preliminares: Instalando y Abriendo NetLogo

El programa NetLogo (Versión 6.1) lo puede bajar del siguiente enlace:

enlace para bajar NetLogo

Usaremos la versión 6.1 Una vez lo instale, abra el programa en su computadora



debe aparecer lo siguiente:

- Crear, a través de Archivo / Nuevo en el menú de NetLogo, un nuevo archivo de NetLogo. Guárdelo a través de Archivo /Guardar como, en un directorio de su elección y con el nombre de archivo BusHongos.nlogo.
 - Haga clic en el botón Configuración, el botón Configuración abre el cuadro de diálogo de configuración del modelo, donde puede verificar la geometría del mundo. Usaremos la geometría predeterminada con:
 - el origen (parcela 0,0) en el centro del mundo.
 - la coordenada máxima en x (max-pxcor) y coordenada máxima en y(max-



pycor) en 16.

Esto forma un cuadrado de 33×33 parcelas. Haga clic en Aceptar para cerrar el cuadro de diálogo Configuración.

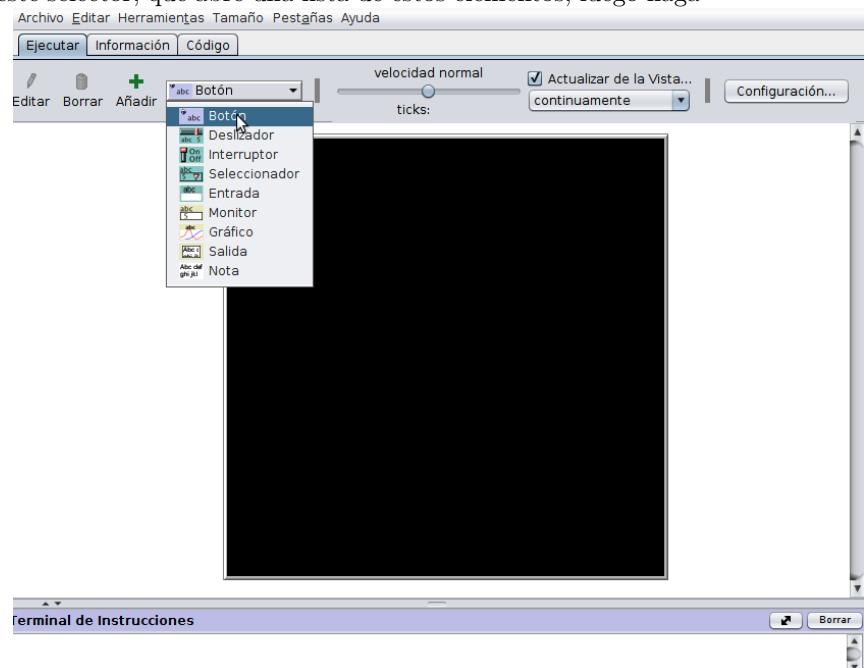
Para nuestro modelo de búsqueda de hongos, queremos crear un mundo con parcelas negras, con algunos grupos de parcelas rojas (que representan hongos) distribuidas al azar, también crearemos dos agentes (turtles) que serán nuestros buscadores y luego haremos que los buscadores encuentren las parcelas rojas. Por lo tanto, necesitamos :

- setup: inicializar el Mundo y los buscadores , y luego
- go: modelar las acciones que realizarán los buscadores para encontrar los hongos.

5.1.2 Procedimiento setup

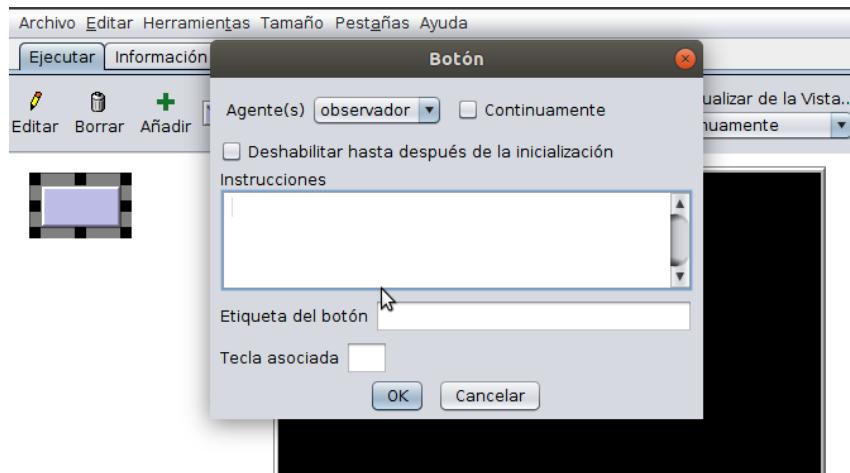
Seguiremos la convención en NetLogo de usar el nombre **setup** para el procedimiento que inicializa el mundo y los agentes (buscadores), y el nombre **go** para el procedimiento que realiza las acciones que realiza el modelo. A estas alturas ya debía saber que un procedimiento comienza y termina con las palabras clave **to** y **end** respectivamente.(al leer la guía de programación de NetLogo) Antes de construir los procedimientos de inicio (**setup**) y de acciones (**go**) vamos a crear dos botones en la interfaz de NetLogo (pestaña Interfaz) que se utilizarán para activar estos procedimientos, active la pestaña interfaz de NetLogo.

- En la pestaña Interfaz, hay un menú desplegable, generalmente etiquetado como “Botón”, que le permite seleccionar uno de los elementos de interfaz proporcionados por NetLogo (botón, deslizador, interruptor, etc.).
- Haga clic en este selector, que abre una lista de estos elementos, luego haga



clic en Botón.

- Coloque el cursor del mouse en la parte en blanco de la interfaz, a la izquierda de la ventana negra del mundo y haga clic. Esto pone un nuevo botón en la interfaz y abre una ventana que le permite colocar las características de este nuevo botón:

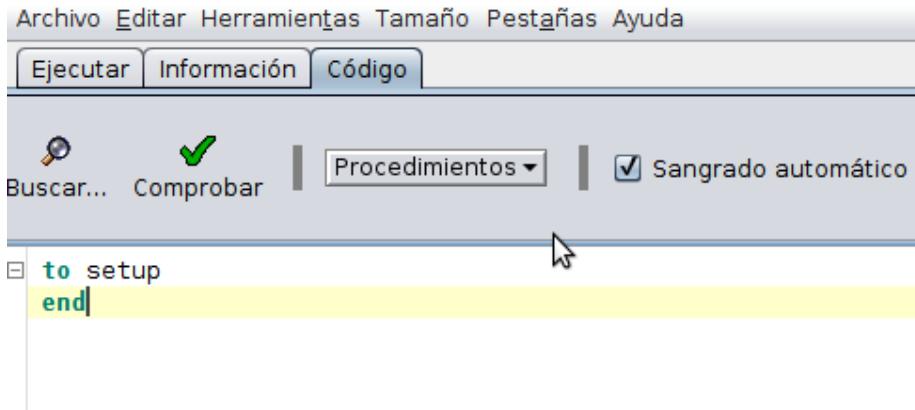


- En el campo Instrucciones de la ventana que apareció, escriba “setup” y haga clic en Aceptar, hemos creado un botón que ejecutará un procedimiento llamado setup, la etiqueta del botón está en rojo, lo que indica que:

no hay un comando o procedimiento correspondiente llamado de esa manera, esto lo arreglaremos en un minuto. A continuación, escribamos el procedimiento de setup

- Haga clic en la pestaña Código, donde encontrará un espacio en blanco para ingresar su código. Al inicio escriba:

```
to setup
end
```

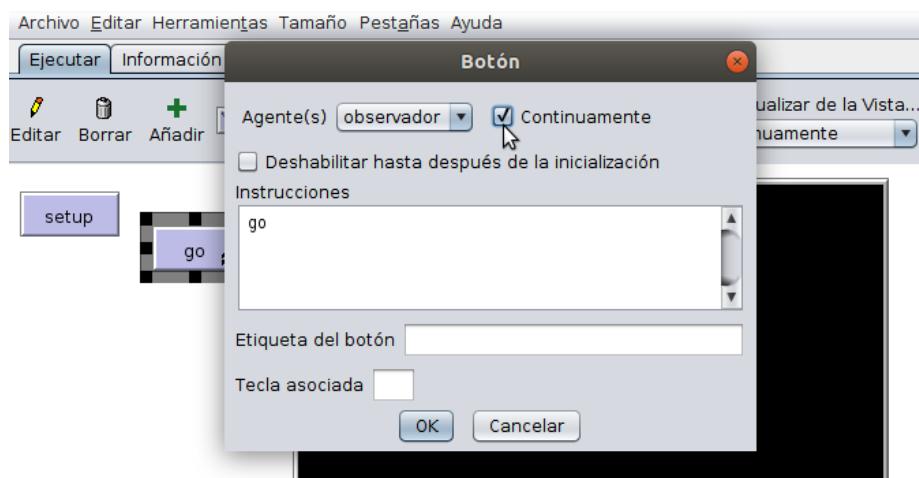


- Haga clic en el botón Comprobar (el que tiene un chulito). Este botón activa el verificador de sintaxis de NetLogo, que busca errores en su código como :

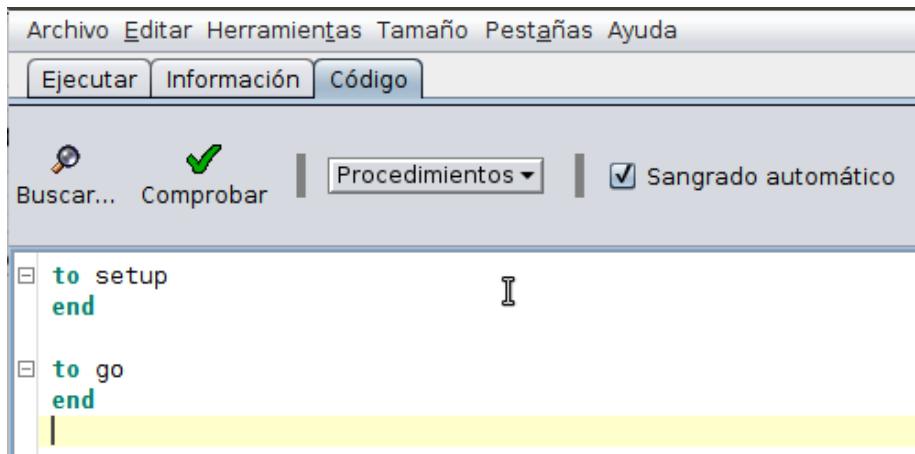
- ¿faltan paréntesis?
- ¿Los comandos tienen el número correcto de entradas?
- ¿el procedimiento no existe?, etc.

Es importante usar el verificador de sintaxis **con frecuencia**, después de cada declaración que escriba, de esta manera se encuentran errores de inmediato y se sabe exactamente donde están. Sin embargo, este verificador no encuentra todos los errores; más adelante veremos otras formas de encontrar errores. Ahora no debería haber ningún mensaje de error. Si vuelve a la pestaña Interfaz, la etiqueta del botón **setup** debe estar en negro, lo que indica que ahora el procedimiento ya está definido y se puede ejecutar (a pesar de que este procedimiento todavía está vacío y no hace nada.)

- Regrese a la pestaña Código y elimine la palabra *end* Haga clic en el botón Verificar nuevamente. Se obtendrá un mensaje de error que dice **End expected** o sea que se esperaba un end.
- Deshaga la eliminación de end usando CTRL-Z .
- Regrese a la pestaña Interfaz y cree un segundo botón, coloque en la ventana de instrucciones la palabra “go” y luego haga clic en la casilla que dice “Continuamente”:



- Haga clic en Aceptar. Este segundo botón, go, ahora tiene un par de flechas circulares que indican que es un botón que al oprimirlo se ejecutará continuamente (“Para siempre”) significa que cuando se hace clic en el botón, se ejecutará su procedimiento (go) una y otra vez hasta que se vuelva a hacer clic en el botón.
- De la misma manera que creamos el procedimiento de setup en la pestaña de código (con dos líneas que dicen setup y end), escriba el procedimiento go de la misma manera.

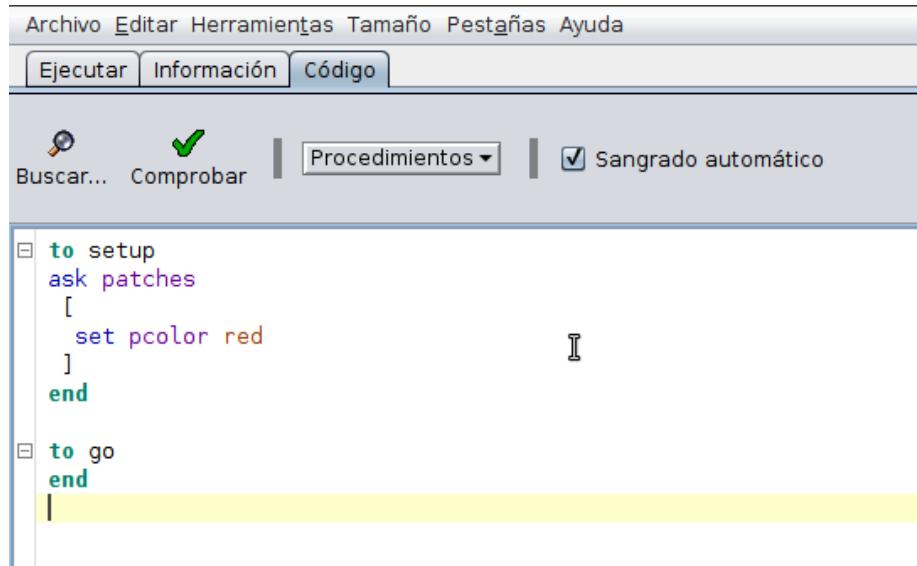


5.1.3 Construyendo las parcelas.

- Ahora, le diremos al procedimiento **setup** que cambie el color de las parcelas.
- Cambie el procedimiento de configuración a lo siguiente:

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

ask es el comando más importante y poderoso de NetLogo. Hace que los agentes seleccionados (aquí, todos las parcelas) efectuen las acciones especificadas dentro de los paréntesis cuadrados. Estos paréntesis cuadrados siempre definen un bloque de acciones que se ejecutan juntas. Las declaraciones que acabamos de escribir “preguntan” (ask) a todos las parcelas que cambien su color a rojo (pcolor = red). Las parcelas por defecto tienen su color (pcolor) configurado en negro (por eso el mundo se ve así al comienzo como un gran cuadrado negro), en nuestro nuevo procedimiento de setup usaremos el comando “set” para cambiar el color a rojo, este comando es un operador de asignación: asigna un valor (rojo) a una variable, en este caso pcolor. (En muchos lenguajes de programación, esta declaración se escribiría pcolor=rojo, que no funciona en NetLogo!)



- Pruebe el nuevo procedimiento de setup yendo a la pestaña Interfaz y haciendo clic en el botón. Todas las parcelas deben ponerse rojas. Sin embargo, queremos que solo unos pocos grupos de parcelas se vuelvan rojas (los hongos). Hagamos que NetLogo seleccione cuatro parcelas aleatorias, y luego pida a esas cuatro parcelas que vuelvan rojas a veinte parcelas cercanas. La modificación del procedimiento setup es la siguiente:

```

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

```

Los comandos n-of y in-radius son nuevos.

- Busque n-of y in-radius en el Diccionario NetLogo. Puede encontrarlos en el Diccionario de NetLogo, haciendo clic en la categoría Agentsset (lugar donde se encuentran todos los comandos que tienen que ver con grupos de agentes).

(Nota: un truco importante para ir directamente a la definición de cualquier comando es haciendo clic en la palabra y presionar F1.)

El Diccionario de NetLogo explica que n-of selecciona un subconjunto aleatorio de un grupo de agentes, in-radius selecciona todos los agentes (en este caso parcelas) dentro de un radio especificado (cinco). Por lo tanto, nuestro nuevo código de setup identifica cuatro parcelas aleatorias y les pide a cada una de ellas que identifique aleatoriamente veinte parcelas dentro de un radio de cinco,

y que les cambie el color a rojo.

- Vaya a la pestaña Interfaz y presione el botón de setup. Si todos las parcelas siguen en rojo ¡no pasa nada! ¿Hay algún error? En realidad no, porque solo le dijó al programa que pusiera algunas parcelas en rojo, **pero todos las parcelas ya estaban en rojo**, así que no se ve ningún cambio. Para evitar este tipo de problema, asegúrenos de que el mundo siempre sea restablecido a su estado inicial (o sea las parcelas en negro) Esto se hace con el comando clear-all que se debe agregar como primera línea al procedimiento setup:

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

Olvidar poner clear-all al comienzo del procedimiento de setup es un error muy común. Casi siempre necesitamos comenzar el proceso de setup borrando todo lo que quedaba desde la última vez que ejecutamos el modelo. Si se hace clic en el botón de setup ahora, verá un mundo en negro con algo de rojo. Si se hace clic en el botón de setup varias veces, verá que cuatro grupos aleatorios de manchas rojas se crean, de hecho a veces parece que se crean menos grupos porque los grupos se superponen. Otras veces parece que parte de un grupo se crea en el borde del mundo; para entender por qué, vaya a la sección de la Guía de interfaz de NetLogo en las Vistas y lea sobre “World Wrapping”, y vea la sección de la Guía de programación llamada “Topología”. Comprender la noción de “World Wrapping” de NetLogo que es muy importante!

5.1.4 Variables globales

- Desde el menú principal de NetLogo, presione Archivo / Guardar. ¡Guarde su trabajo a menudo! NetLogo no lo hará!! Ahora hagamos que el número de grupos de parcelas rojas sea un parámetro del modelo, para hacer esto necesitamos :
- crear una variable global numérica llamada num-clusters (número de grupos de hongos)
- darle un valor de 4 (usando el comando set)
- En la pestaña de Código, vaya a la parte superior de todo su código e inserte arriba del procedimiento setup lo siguiente:

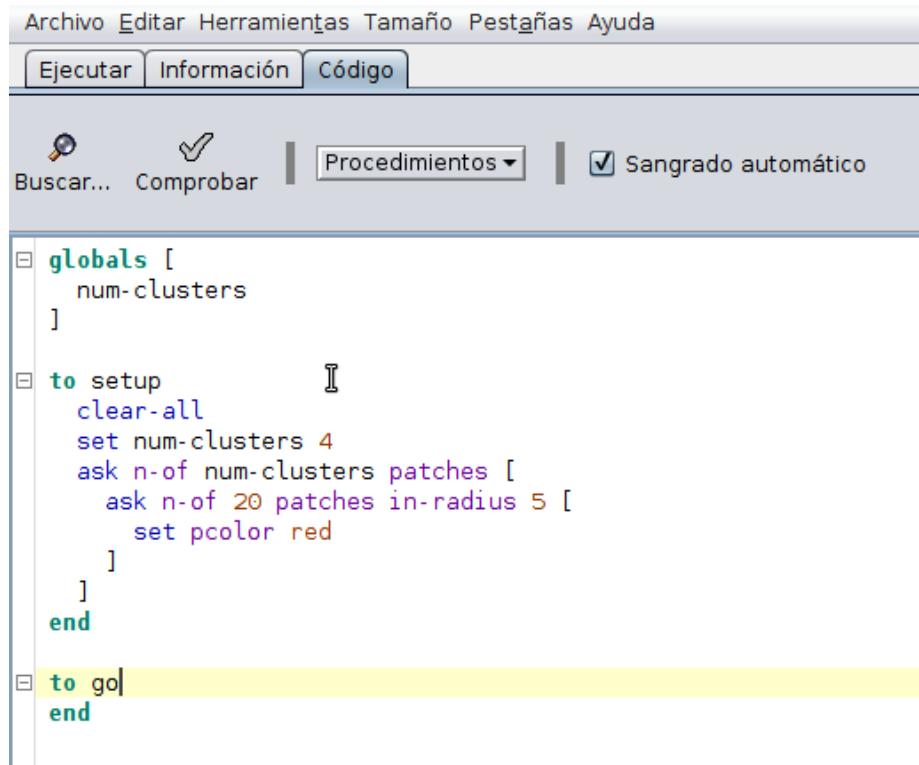
```
globals [
  num-clusters
]
```

- Coloque:

```
set num-clusters 4
```

luego de la instrucción clear-all en el setup.

- reemplace el 4 en la instrucción ask n-of 4 patches por num-clusters. Ahora, si queremos cambiar el número de grupos, sabemos exactamente dónde encontrar y cambiar el valor del parámetro sin modificar nada más en el modelo. (Más tarde, usaremos deslizadores para cambiar los parámetros desde la interfaz)

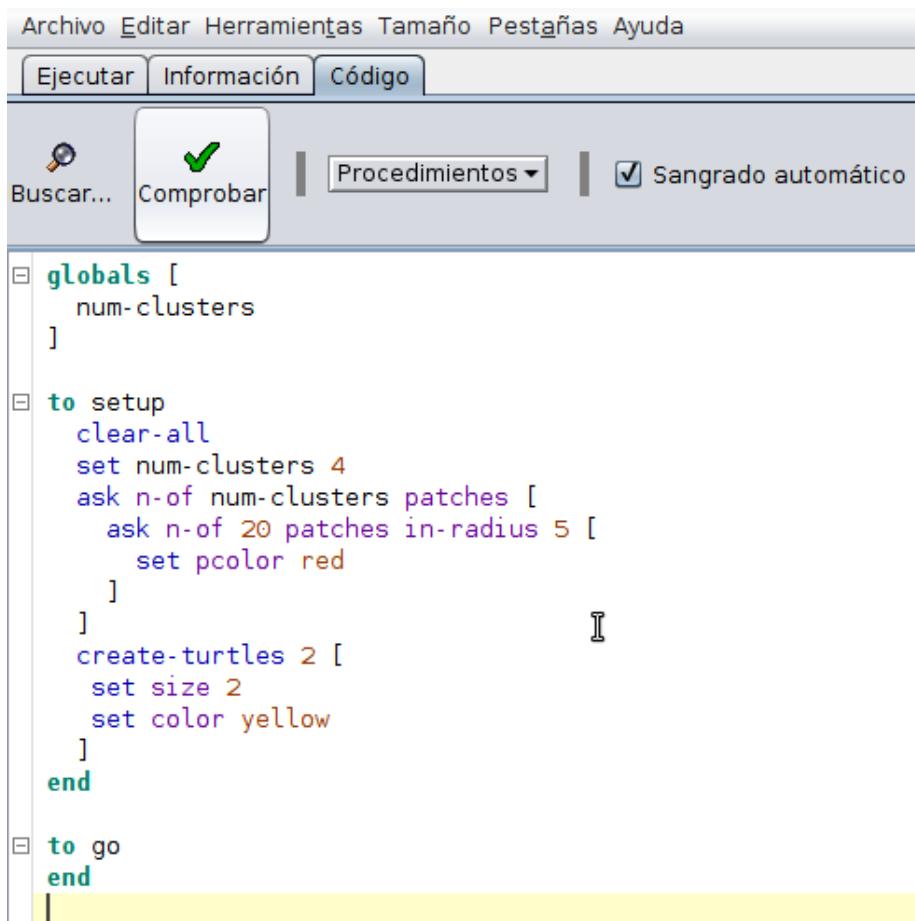


5.1.5 Creando los agentes (buscadores)

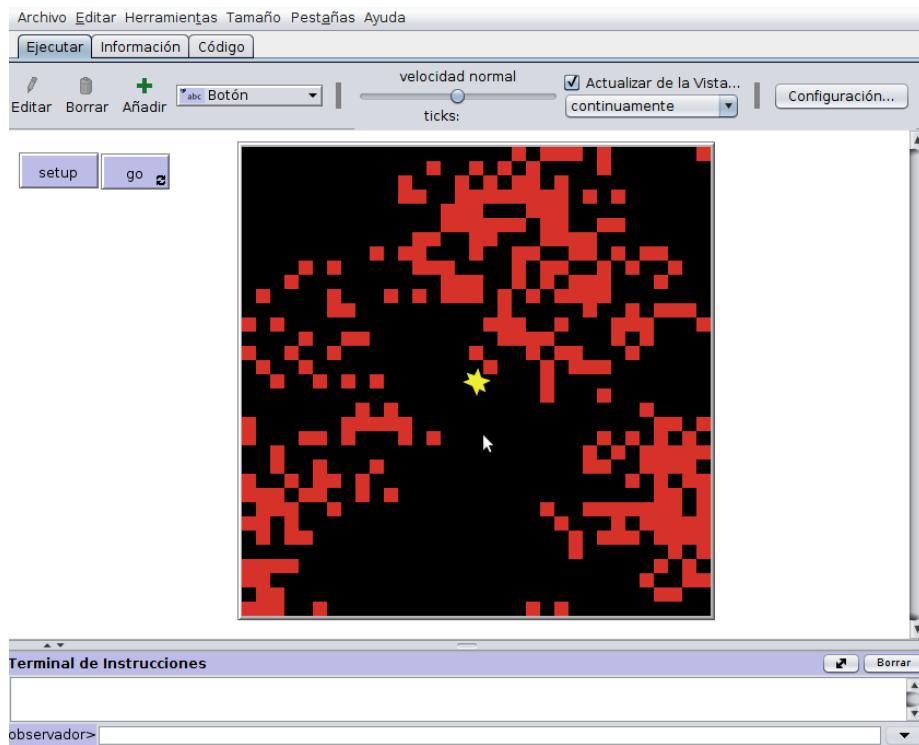
Ahora que hemos construido el “mundo” del modelo, necesitamos crear los agentes, en nuestro caso dos buscadores. Esto se hace con el comando `create-turtles`. La declaración `create-turtles 2 []` crea dos tortugas (agentes) y les hace ejecutar el código dentro de los corchetes. Agregue el siguiente código al final

del procedimiento de setup Haga clic en Comprobar para buscar errores, luego pruebelo usando el botón de setup en la interfaz:

```
create-turtles 2 [
  set size 2
  set color yellow
]
```



El tamaño y el color son variables predefinidas de los agentes (turtles) por esta razón podemos definir sus valores sin definirlas. Si se hace clic en el botón setup varias veces, verá que dos buscadores se colocan en el centro del mundo (por defecto) y su dirección varía al azar:



5.1.6 Comportamiento de los agentes (Procedimiento go)

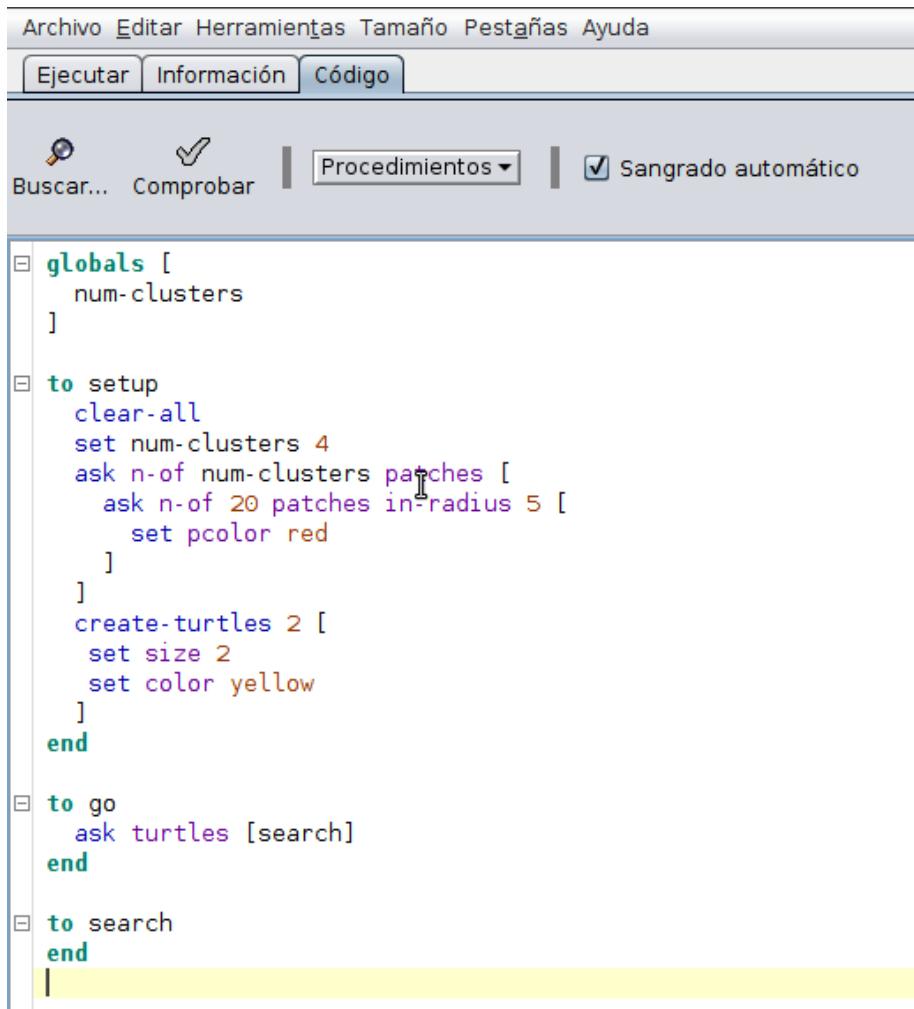
Ahora vamos a construir el procedimiento go. El procedimiento go define el “comportamiento” del modelo, o sea los procesos que se realizarán una y otra vez, y su orden. Para mantener el procedimiento go simple y fácil de entender, lo cual es muy importante porque es el corazón del programa: casi nunca programamos procesos o acciones reales dentro del go. En cambio, definimos procedimientos auxiliares y se programa cada acción. En nuestro modelo de búsqueda de hongos, solo tenemos una acción, la búsqueda de los agentes, por lo tanto el procedimiento go necesita incluir solo un procedimiento, que le dice a todas las tortugas (nuestros dos buscadores) que ejecuten un procedimiento de búsqueda:

```
ask turtles [search]
```

Para que esto funcione, por supuesto, necesitamos escribir el procedimiento llamado “search” para los buscadores.

- Agregue la linea, ask turtles [search] al procedimiento go. Luego cree un procedimiento vacío para que pueda verificar el código :

```
to search
end
```



The screenshot shows the NetLogo interface with the following details:

- Menu Bar:** Archivo, Editar, Herramientas, Tamaño, Pestañas, Ayuda.
- Toolbar:** Ejecutar, Información, Código. The "Código" button is highlighted.
- Search/Check Buttons:** Buscar... (magnifying glass), Comprobar (checkmark).
- Procedimientos:** A dropdown menu currently set to "Procedimientos".
- Automatic Logging:** A checked checkbox labeled "Sangrado automático".
- Code Editor:** Displays a partial NetLogo script with the following structure:
 - globals [** num-clusters **]**
 - to setup**
 - clear-all
 - set num-clusters 4
 - ask n-of num-clusters patches [
 - ask n-of 20 patches in-radius 5 [
 - set pcolor red
 - create-turtles 2 [
 - set size 2
 - set color yellow
 - end**
- to go**
 - ask turtles [search]
- to search**
 - end**

Ahora programemos como buscan los agentes (buscadores) Cada vez que se ejecuta la búsqueda, queremos que los cazadores giren, **un poco** si no han encontrado nada recientemente (por ejemplo, en las últimos veinte parcelas recorridas), o **bruscamente** para seguir buscando en la misma área si encontraron un hongo recientemente. Luego de girar los buscadores deben avanzar hacia una parcela vecina. * Agregue el siguiente código al procedimiento de búsqueda, cualquier parte que no entienda búsquela en el diccionario de NetLogo:

```
ifelse time-since-last-found <= 20
  [right (random 181) - 90]
  [right (random 21) - 10]
  forward 1
```

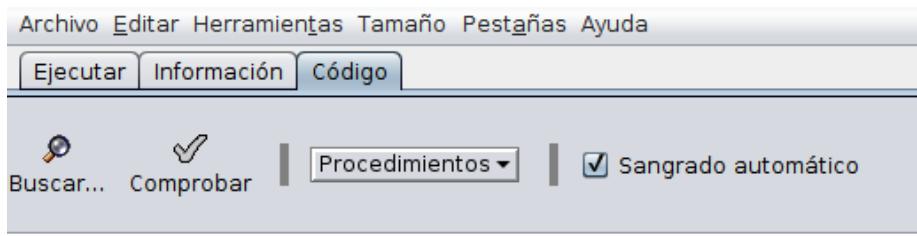
Puede deducir buscando en el Diccionario de NetLogo que la declaración right

(random 181 - 90) hace que los buscadores (turtles) giren un ángulo aleatorio entre -90 y +90 grados. Observe cómo usamos la instrucción **ifelse** para modelar decisiones. Si la condición booleana que sigue a ifelse es verdadera, entonces el código en el primer conjunto de corchetes se ejecuta; si es falso, se ejecuta el código en el segundo conjunto de corchetes. Una condición booleana es una declaración que es verdadera o falsa. Aquí, la condición es:

```
time-since-last-found <= 20
```

consiste en una comparación: es cierto si el valor de time-since-last-found es menor o igual a 20. Si presiona el botón Comprobar saldrá un error:

“Nothing named time-since-last-found has been defined”



The screenshot shows the NetLogo interface with the toolbar at the top. The 'Código' tab is selected. Below the toolbar, there are search and check buttons, a 'Procedimientos' dropdown, and an unchecked 'Sangrado automático' checkbox. A yellow status bar at the bottom displays the error message: 'Nothing named TIME-SINCE-LAST-FOUND has been defined.'

```

globals [
    num-clusters
]

to setup
    clear-all
    set num-clusters 4
    ask n-of num-clusters patches [
        ask n-of 20 patches in-radius 5 [
            set pcolor red
        ]
    ]
    create-turtles 2 [
        set size 2
        set color yellow
    ]
end

to go
    ask turtles [search]
end

to search
    ifelse time-since-last-found <= 20
        [right (random 181) - 90]
        [right (random 21) - 10]
    forward 1
end

```

5.1.7 Variables locales de agentes

No hemos definido esta variable, queremos que sea una variable que registre cuánto tiempo ha pasado desde que cada buscador encontró el último hongo. Eso significa tres cosas.

- Primero, Cada cazador debe tener su propio **valor único** para esta variable, por lo que debe ser una variable de tipo turtle (agente), justo después de

la declaración global al comienzo de su programa, agregue:

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

En segundo lugar, necesitamos establecer el valor inicial de esta variable cuando creamos los buscadores Estableceremos,asumiendo que los cazadores aún no han encontrado un hongo,un valor mayor que 20.

- En el procedimiento de setup, cambie la instrucción create-turtles a esto:

```
create-turtles 2 [
    set size 2
    set color yellow
    set time-since-last-found 999
]
```

Finalmente, los cazadores deben actualizar time-since-last-found cada vez que se mueven. Si encuentran un hongo, necesitan restablecer time-since-last-found a cero (¡y recoger el hongo!); de lo contrario, necesitan agregar uno a time-since-last-found.

- Agregue estas declaraciones al final del procedimiento search:

```
ifelse pcolor = red [
    set time-since-last-found 0
    set pcolor yellow
] [
    set time-since-last-found time-since-last-found + 1
]
```

```

Archivo Editar Herramientas Tamaño Pestañas Ayuda
Ejecutar Información Código
Buscar... Comprobar Procedimientos Sangrado automático
globals []
  num-clusters
]
turtles-own [
  time-since-last-found
]

to setup
  clear-all
  set num-clusters 4
  ask n-of num-clusters patches [
    ask n-of 20 patches in-radius 5 [
      set pcolor red
    ]
  ]
  create-turtles 2 [
    pen-down
    set size 2
    set color yellow
    set time-since-last-found 999
  ]
  reset-ticks
end

to go
  ask turtles [search]
  tick
end

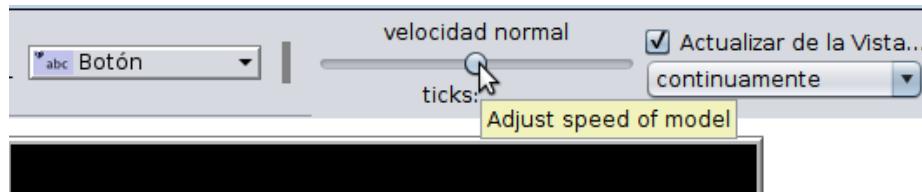
to search
  ifelse time-since-last-found <= 20
    [right (random 181) - 90]
    [right (random 21) - 10]
    forward 1
  ifelse pcolor = red [
    set time-since-last-found 0
    set pcolor yellow
  ] [
    set time-since-last-found time-since-last-found + 1
  ]
end

```

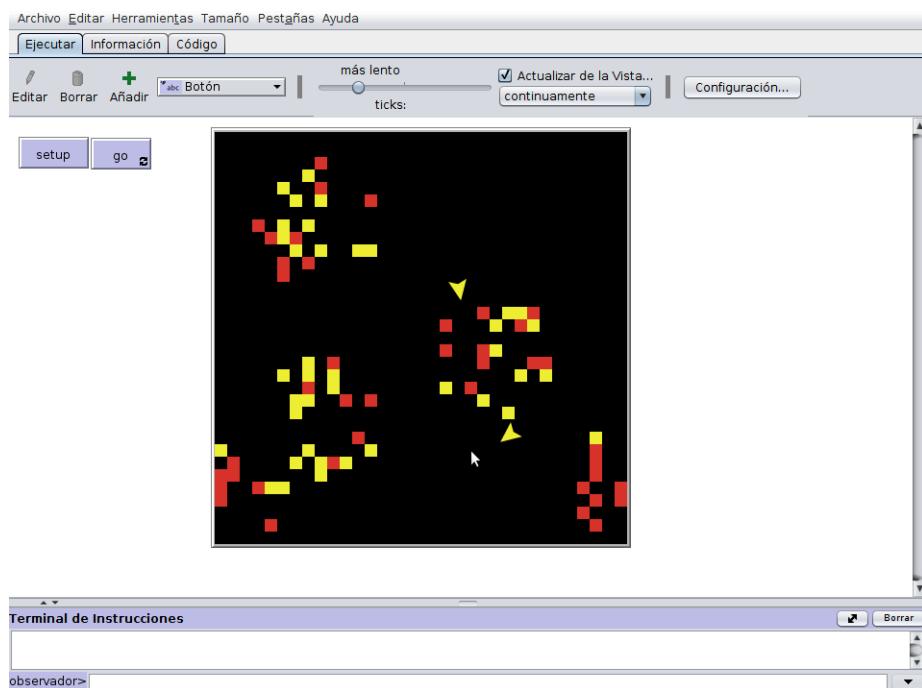
Tenga en cuenta que estas declaraciones funcionan porque NetLogo permite a los agentes leer y cambiar variables de la parcela en la que están (en este caso set pcolor yellow). También tenga en cuenta que para agregar un uno a time-since-last-found tuvimos que usar set para asignar su nuevo valor a su antiguo valor más uno. (Tenga en cuenta que en NetLogo **debe usar espacios**

alrededor de operadores aritméticos como “+”; de lo contrario, NetLogo cree que son parte del nombre de la variable).

Asegúrese de comprender que las coordenadas de la parcela son variables discretas: solo tienen valores enteros, en unidades de ancho de parcela. En contraste, las coordenadas de tortuga son variables continuas:pueden tomar cualquier valor dentro de una parcela, por ejemplo, 13.11 unidades. Por eso es importante distinguir entre instrucciones que se refieren a coordenadas de tortuga y aquellas que se refieren a parcelas. Por ejemplo la instrucción **move-to** mueve un agente (turtle) al centro de una parcela, pero **forward** puede colocar un agente en cualquier lugar de esta.Si ahora prueba el programa haciendo clic en el botón go, es posible que no pueda ver mucho porque los agentes buscadores se mueven demasiado rápido Si es así, ajuste el controlador de velocidad de ejecución en la interfaz.



(Haga clic en go por segunda vez para detener la ejecución). Voila! ¡Aquí tenemos nuestro primer programa completo de NetLogo! (¡Asegúrese de guardarlo!) Al correr el modelo , oprimiendo setup y luego go debe aparecer lo siguiente:



Sorprendentemente, a pesar de su simplicidad, este modelo contiene los elementos más importantes de cualquier modelo basado en agentes. Pero antes de mirar de manera más detallada nuestro modelo detengámonos un momento para echar un vistazo a tres herramientas muy importantes de NetLogo

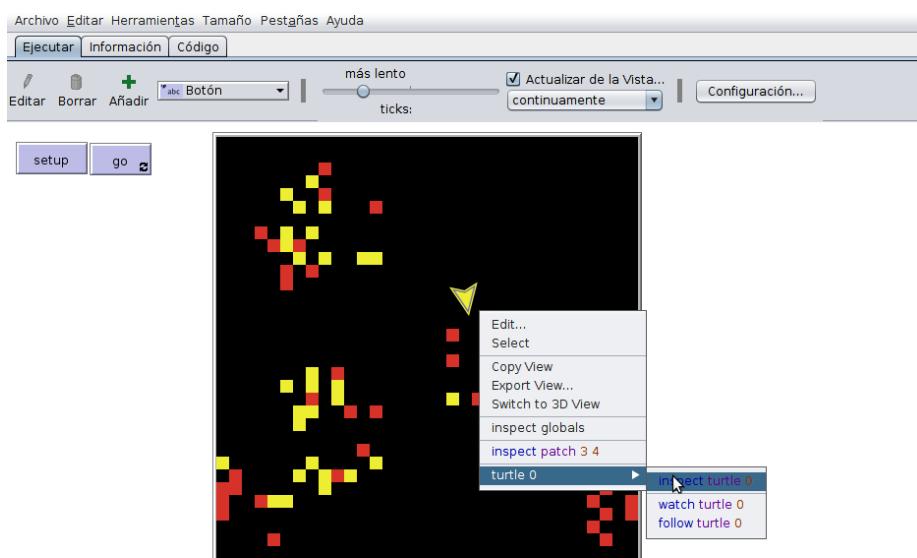
5.1.8 Tres Herramientas Importantes

Las herramientas son:

- Monitores de agentes
- el Centro de Comandos y
- los ticks (Manejo del tiempo)

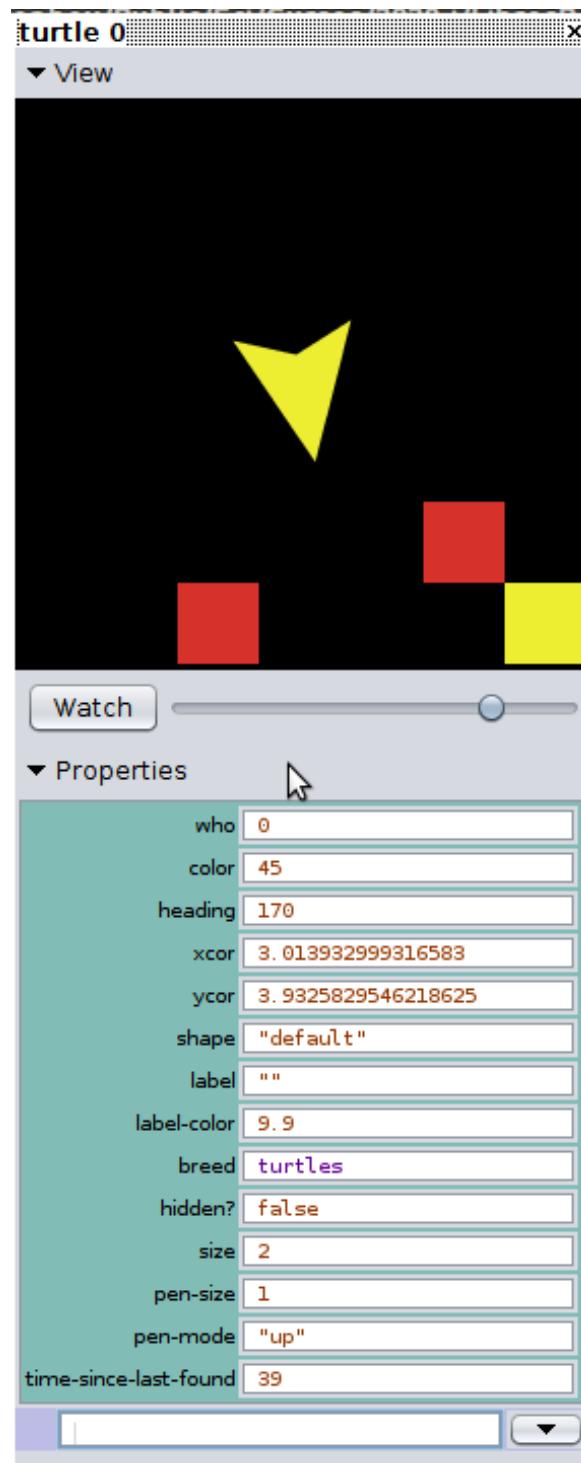
5.1.9 Monitor de Agentes

Los monitores de agentes son herramientas para ver y cambiar las variables de un agente (tortuga, parcela o enlace). * Mueva el cursor sobre uno de los buscadores y haga clic derecho. Aparece un panel en el que aparece al final “turtle 0”; mueva el cursor a turtle 0 y seleccione “inspect turtle 0”.



Se abre un Monitor de agente que incluye dos paneles: una vista ampliada de donde se encuentra el agente (turtle 0) y su entorno y una lista de todas las variables del agente:

su posición, color, dirección, etc.



Observe que aparece la variable que definimos a los agentes buscadores (time-since-last-found) (Cada panel se puede cerrar y se vuelve a abrir haciendo clic en el pequeño triángulo negro en su esquina superior izquierda). Sin cerrar esta ventana, puede reiniciar el modelo haciendo clic en go y observe cómo cambian las variables del agente, también puede cambiar sus valores siempre que lo deseé, simplemente ingresando un nuevo valor en el monitor.

- Detenga el programa y cambie el tamaño de la tortuga a 5, y su tiempo desde la última vez que se encontró a -99 (Sus cambios se activan cuando presiona la tecla Enter o mueve el cursor fuera del cuadro de diálogo donde se ingresan los valores).

Hay más cosas que puede hacer con los monitores de agentes, incluido dar comandos directamente al agente seleccionado escribiendo los mismos tipos de declaraciones de NetLogo que usaría en su código. Lea la sección Monitores de agentes de la Guía de interfaz del Manual del usuario.

5.1.10 Centro de Comandos

El Centro de Comandos aparece en la parte inferior de la pestaña Interfaz. Aquí se pueden colocar comandos para el observador o para todos los parcelas, tortugas(agentes) o enlaces. Lea la sección del Centro de comandos del Manual del usuario y pruebe algunos comandos, como decirle a las parcelas que cambien su color a azul, el Centro de Comandos no parece útil al principio, pero a medida que se convierta en un mejor modelador basado en agentes con NetLogo, encontrará estas herramientas extremadamente útiles. Como ejemplo divertido, use el Centro de comando para hacer lo siguiente:

- Despues de configurar el modelo y dejar que se ejecute por un segundo o dos,parelo. Haga clic en el texto “observador>” en el Centro de Comandos. De la lista que aparece seleccione “tortugas>” para que ahora pueda emitir un comando a todas las tortugas. En la ventana adyacente, ingrese “hatch 1 [right 160]”, que le dice a cada buscador que cree un segundo buscador, que luego gira 160 grados:



- Ejecute el programa y vea cómo los cuatro buscadores buscan parcelas rojas. Usted puede repetir el comando para crear aún más buscadores: simplemente coloque el cursor en la línea de comando y presione la flecha hacia arriba para recuperar el comando anterior, luego teclee enter
- Despues de crear más buscadores , haga que el Centro de Comando le diga cuántos buscadores hay Seleccione observador> para enviar comandos e ingrese “show count turtles”

5.1.11 Manejo del tiempo

Hasta ahora, nuestro modelo no maneja el tiempo explícitamente: no hacemos un seguimiento de cuántas veces se ejecuta el procedimiento, por lo que no podemos determinar cuánto tiempo ha transcurrido si por ejemplo, que cada vez que los buscadores de hongos se mueven representa un minuto. Para modelar el tiempo, hay que usar la instrucción **tick** cada vez que se ejecuta el procedimiento go. (Lea en la guía de programación la sección contador de ticks (tick counter)) y se dará cuenta que:

- Al final del procedimiento setup, se debe insertar una línea con la instrucción **reset-ticks**. Al comienzo del procedimiento go, inserte una línea con la instrucción **tick**

Si ahora hace clic en setup y luego en go va, puede observar el contador de ticks en la parte superior de la Vista; este muestra cuántas veces se ha ejecutado el procedimiento go. Finalmente, ¿no sería útil ver el camino de cada buscador?:

- En el procedimiento setup, agregue la instrucción pen-down, luego de la instrucción create-turtles.

UUUfff Listo!!!! El modelo completo de buscador de hongos es el siguiente:

```
globals [
    num-clusters
]

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

to setup
    clear-all
    set num-clusters 4
    ask n-of num-clusters patches [
        ask n-of 20 patches in-radius 5 [
            set pcolor red
        ]
    ]
    create-turtles 2 [
        pen-down
        set size 2
        set color yellow
        set time-since-last-found 999
    ]
    reset-ticks
end
```

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

to search
  ifelse time-since-last-found <= 20
    [right (random 181) - 90]
    [right (random 21) - 10]
    forward 1

  ifelse pcolor = red [
    set time-since-last-found 0
    set pcolor yellow
  ] [
    set time-since-last-found time-since-last-found + 1
  ]
end
```

Puede ensayarla en el siguiente applet:

```
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

5.2 Análizando el Modelo

5.2.1 Introducción

Antes de realizar análisis específicos, me gustaría hablar de la exploración de un modelo de manera general, cuando no se tiene una pregunta o meta concreta de exploración de un modelo, es útil simplemente sondear las profundidades de lo que es posible en el comportamiento de un modelo. Los modeladores comúnmente exploran el comportamiento del modelo a su manera, haciendo variar los parámetros del modelo, probando diferentes combinaciones, ejecutando el modelo varias veces y mirando que pasa. Los modeladores podrían describir esta actividad simplemente como “jugar” con el modelo o familiarizarse con las cosas . Este proceso informal es natural (y diría que necesario) como parte del la comprensión del modelo, a través de retoques, los modeladores están sondeando el sistema para aprender algo al respecto. Es un proceso iterativo de desarrollar pequeñas hipótesis (o quizás simplemente “corazonadas”) sobre el modelo y su comportamiento, y probándolos. Cada vez que alguien juega y experiemnta con el modelo de esta manera, está intentando responder a la pregunta

“*¿qué comportamiento del modelo producen estas configuraciones de parámetros?*”

5.2.2 Cambios el modelo para el análisis (deslizadores)

Modificaremos el modelo construido en el taller anterior con el objetivo de efectuar un análisis básico de nuestro modelo. El Modelo Netlogo lo puede bajar del siguiente enlace (Dropbox):

Modelo Buscador de Hongos

El modelo , a pesar de su simplicidad, se presta a efectuar diferentes análisis, nos concentraremos a analizar el modelo con base en la pregunta o propósito inicial que tuvimos al construirlo:

“*¿Qué estrategia de búsqueda maximiza el número de hongos encontrados en un tiempo específico?*”

Si conservamos fijo el mundo de nuestro modelo, su tamaño al igual que el número de hongos del bosque (80 hongos) y nos concentraremos en los dos agentes que buscan hongos, su estrategia de búsqueda está determinada por dos parámetros del modelo:

- El tiempo de espera para cambiar de modo de búsqueda
- Los ángulos de giro que efectuan los buscadores

Dejaremos “fijos” los ángulos y variaremos el tiempo de espera de los agentes para poder responder la siguiente pregunta:

“¿Cuál es el tiempo de espera que maximiza el número de hongos encontrados?”

(Nota: Si se mantienen “fijos” el número de hongos y los ángulos de giro)

Antes de realizar el análisis hagamos los siguientes cambios al modelo:

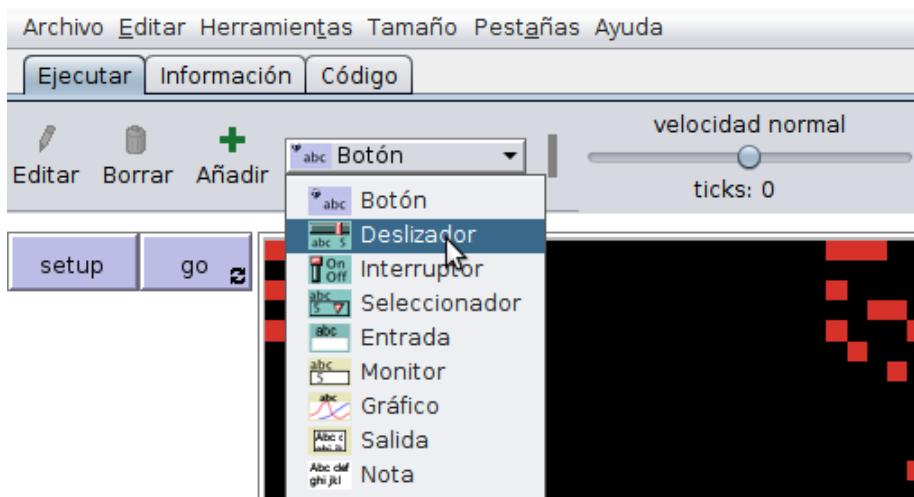
- Fijar el ángulo amplio de búsqueda en : 85 + (random 5)
- Fijar el ángulo corto de búsqueda en : 5 + (random 5)

Cambie las tres primeras líneas del procedimiento search por:

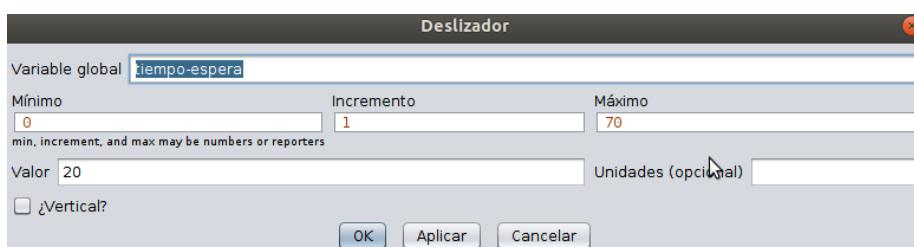
```
ifelse time-since-last-found <= tiempo-espera
  [right 85 + (random 5) ]
  [right 5 + (random 5)]
```

- crear un deslizador para variar el tiempo de espera.

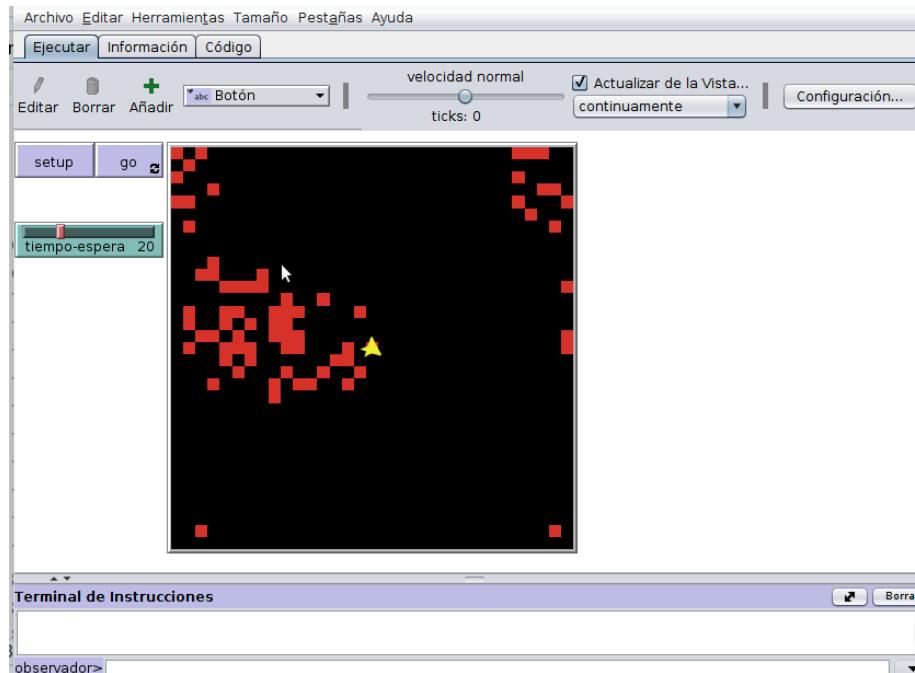
Para crear un deslizador, seleccione del menú de elementos de interfaz de NetLogo (Aparece con el nombre botón) la opción deslizador:



Haga clic debajo de los botones setup y go y aparece una ventana para llenar los datos del deslizador, llénela de la siguiente manera:



El Modelo se debe ver de la siguiente manera:



Listo!!!! Ya modificamos el modelo, consideremos un tiempo específico, por ejemplo 800 unidades de tiempo (ticks), entonces la pregunta traducida a nuestro modelo modificado sería la siguiente:

“¿Qué valor de tiempo-espera produce que los buscadores optimizan el número de hongos encontrados si tienen 800 unidades de tiempo para encontrar los hongos?”

5.2.3 Posibles Preguntas

- ¿Existirá un único valor para este tiempo?
- ¿Será un tiempo pequeño? (por ejemplo menor que 20)
- ¿Será un tiempo grande? (por ejemplo mayor que 70)
- ¿Qué número máximo de hongos se pueden encontrar en estos 800 ticks? (24, 45, 57, 66)
- ¿Podrán encontrar los buscadores más de 100 hongos en este periodo de tiempo?

Nuestra pregunta inicial de búsqueda nos ha llevado a construir un modelo y a modificarlo con el fin de intentar responder a la pregunta, hemos decidido (de pronto arbitrariamente) que el parámetro clave del modelo es el tiempo de espera de los buscadores para pasar de una búsqueda local a una búsqueda más global

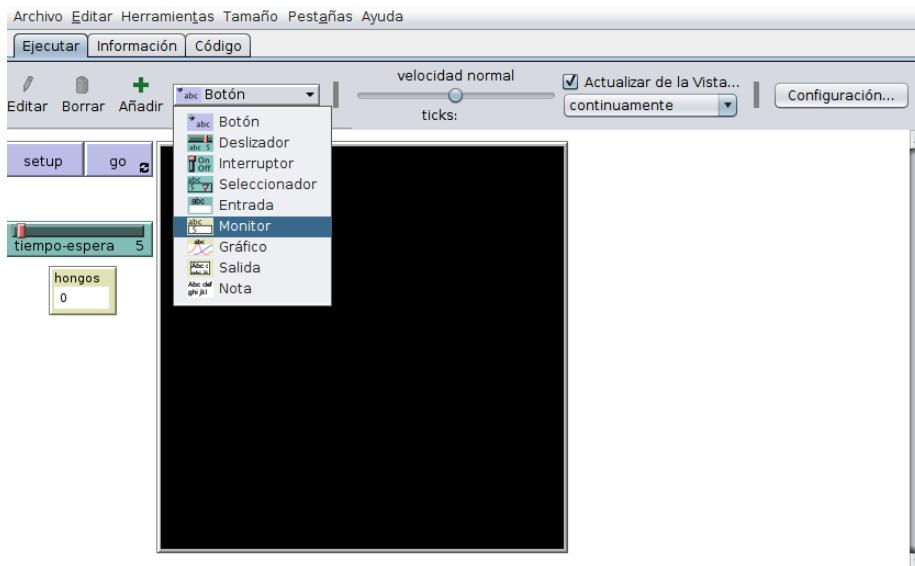
5.2.4 Reporteros

Es importante contabilizar en cada momento el número de hongos encontrados, así que vamos a definir una función que reporte este valor, escriba al final de la Pestaña de Código del modelo lo siguiente

```
to-report hongos-encontrados
  report patches with [pcolor = yellow]
end
```

el reporter hongos-encontrados entrega el conjunto de hongos encontrados (parcelas de color amarillo) Para ver este reportero en acción vamos a definir un monitor que nos permita visualizar en la interfaz de NetLogo el número de hongos-encontrados a medida que el modelo está corriendo.

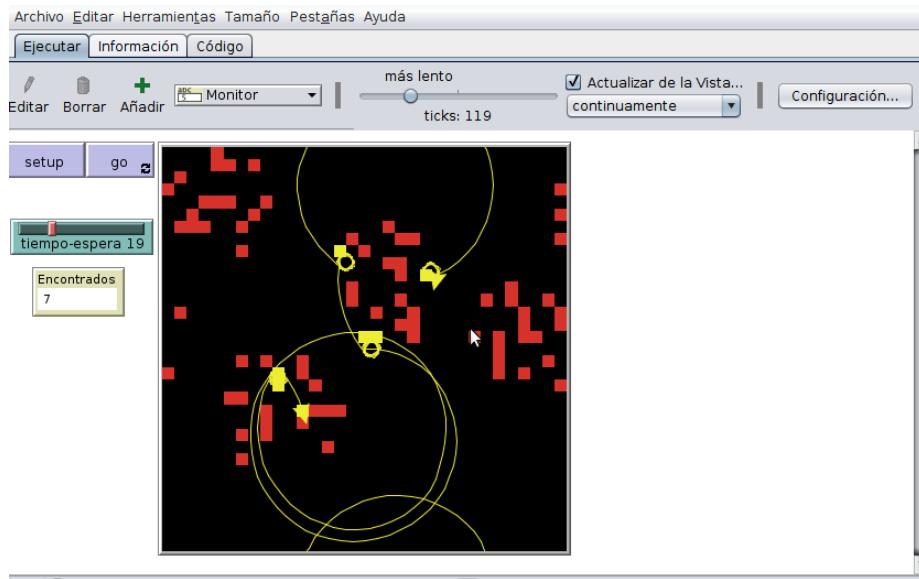
Vaya a la pestaña interfaz, del menú botón seleccione el elemento monitor:



Haga clic debajo del deslizador tiempo-espresa, aparece una ventana llenela de la siguiente manera:



El monitor usa una función importante **count** que cuenta el número de elementos de un conjunto de agentes (en este caso de parcelas amarillas) Corra el modelo (setup y luego go) y observe como el monitor muestra los hongos encontrados a medida que los buscadores los encuentran:



Al jugar y experimentar con el modelo podríamos descubrir valores de tiempo de espera que pueden conducir a optimizar los hongos encontrados, pero muy pronto nos damos cuenta que:

- Los valores posibles de tiempo-espera son muchos
- Recolectar información de hongos encontrados para diferentes valores puede

ser dispendioso y demorado.

Podríamos correr el modelo con diferentes valores de tiempo-espera y registrar los hongos encontrados luego de 800 ticks y luego llevar eso a una hoja electrónica para analizar la información, pero esto es muy dispendioso y hay una manera mucho mejor de hacerlo en NetLogo y es usar el **Analizador de Comportamiento** de NetLogo.

5.2.5 El Analizador de Comportamiento

La funcionalidad que ofrece el Analizor de Comportamiento (BehaviorSpace) es la de ejecutar muchas veces un modelo, modificando para cada ejecución sus parámetros y grabando los resultados que el usuario decide. Proporciona una forma de explorar secciones del espacio de posibles comportamientos y determinar, por ejemplo, qué combinaciones de configuraciones definen comportamientos interesantes o compararlas entre sí para ver la influencia de los parámetros en la dinámica BehaviorSpace ofrece así una forma ordenada y metódica de realizar la exploración del espacio de parámetros. Una exploración que, aunque la mayoría de las veces no podrá ser exhaustiva, si se diseña adecuadamente permitirá una aproximación más eficiente a la comprensión del modelo. Además, los datos almacenados en cada experimento se graban en formatos que permiten ser analizados posteriormente con herramientas específicas de análisis de datos, en nuestro caso usaremos R y Rstudio Para abrir el Analizador de Comportamiento, vaya al menu Herramientas y seleccione la opción Analizador de Comportamiento:

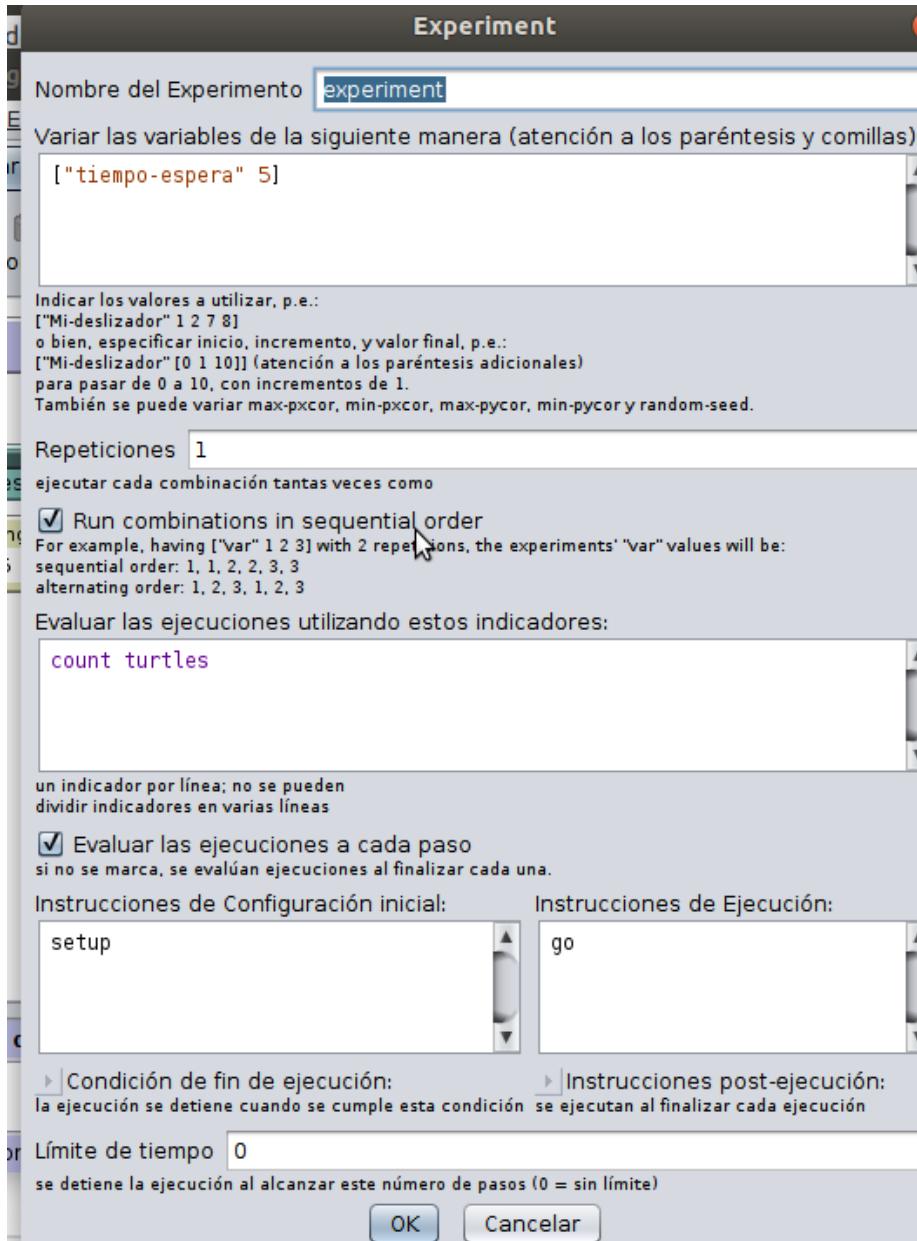


Aparece la siguiente ventana:

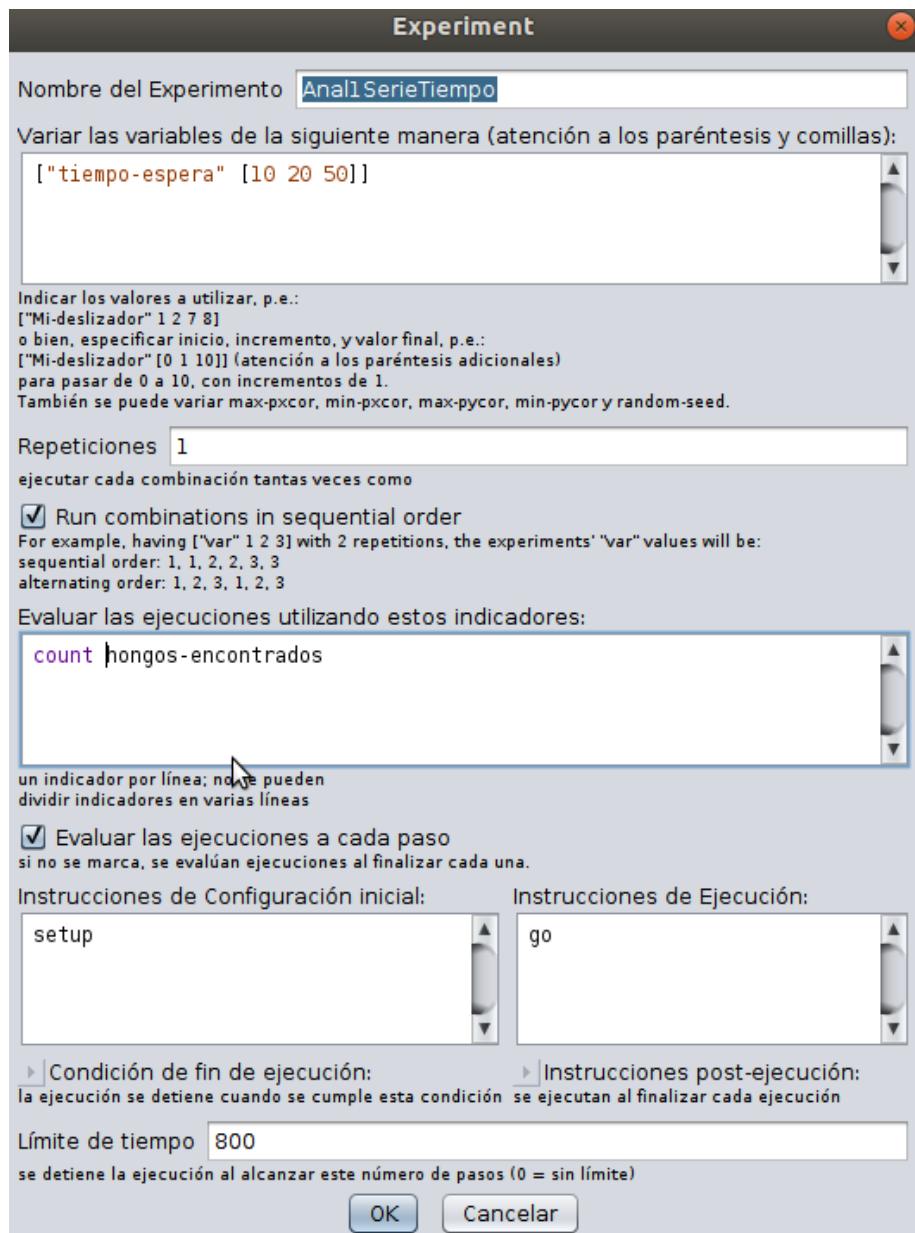


Haga Clic en Nuevo, al hacer clic aparece una ventana que nos permitirá definir lo que se denomina un experimento, que es simplemente una manera de decirlo

a NetLogo que datos y cuantos datos quiero generar del modelo:



Llenemos esta ventana de la siguiente manera:

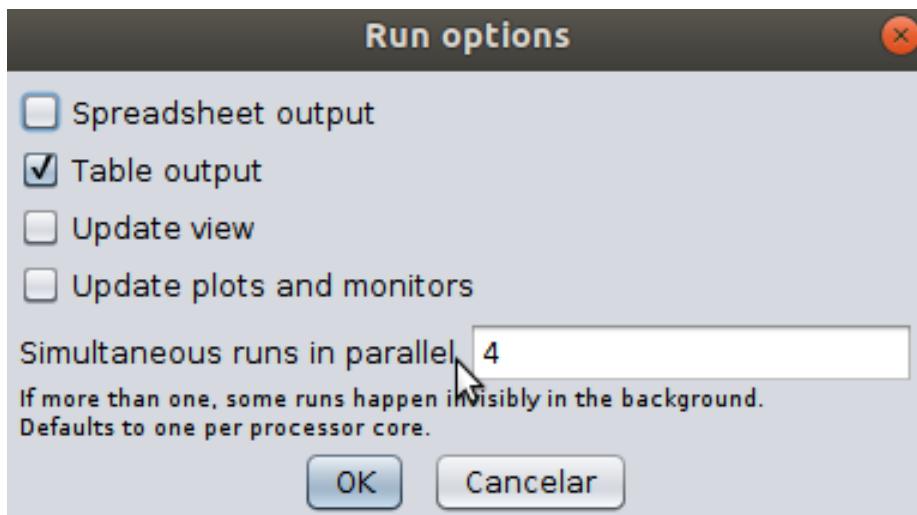


- La primera ventana nos indica que vamos a variar los valores de tiempo-espera entre 10 y 50 en incrementos de 20 ([10 20 50])

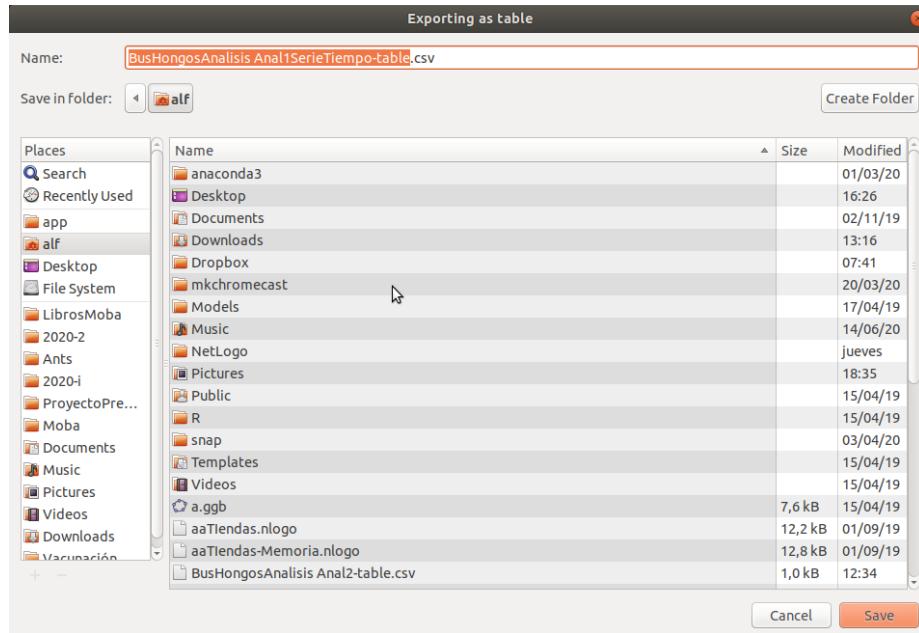
Oprima el botón Ok, aparece la siguiente ventana:



Oprima el botón Ejecutar, aparecen las opciones para generar los datos:



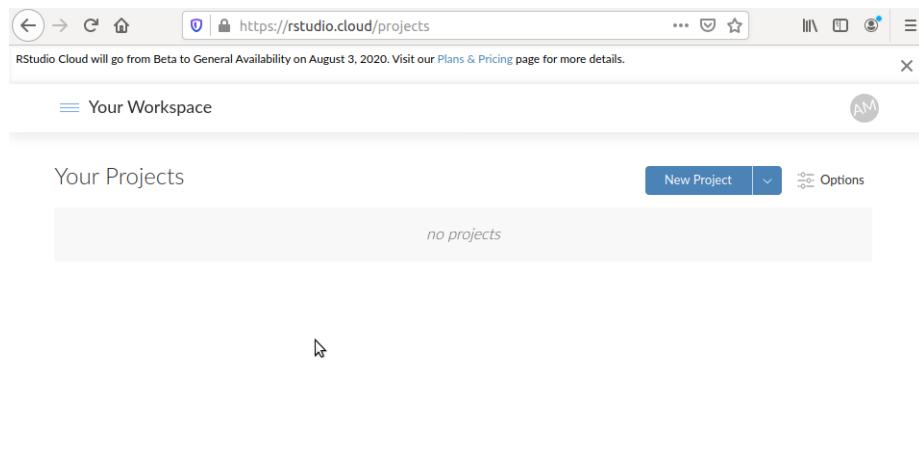
Seleccione la opción Table Output y oprima Ok, aparece un cuadro de diálogo para guardar los datos, guarde los datos en el mismo directorio donde se encuentra el modelo NetLogo, no le cambie el nombre al archivo exportado:



Listo!!! Ya guardamos el archivo de datos generado por nuestro modelo

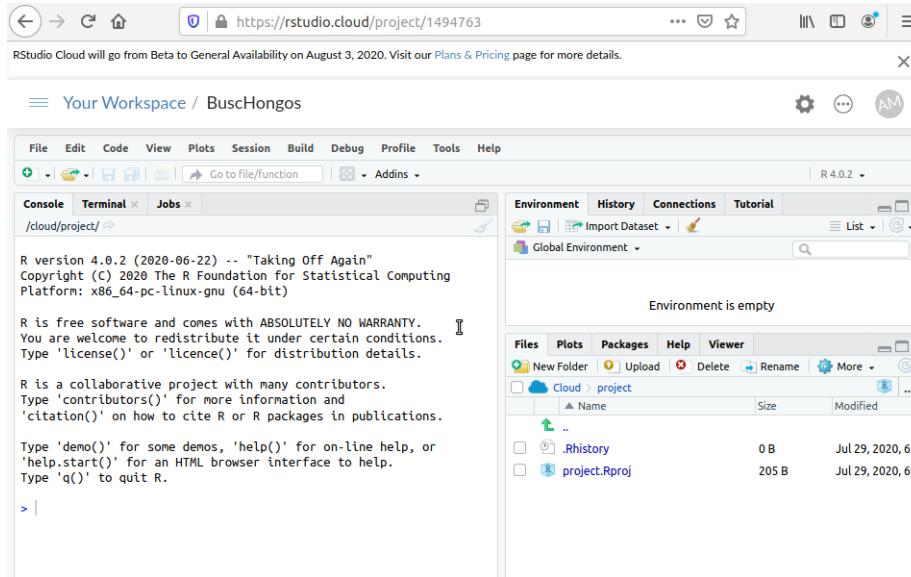
5.2.6 Analizando los datos en R

Ahora vamos a abrir Rstudio-Cloud para importar los datos y analizarlos Vaya a Rstudio cloud (si no esta registrado registrese y cree una cuenta), al entrar verá lo siguiente:

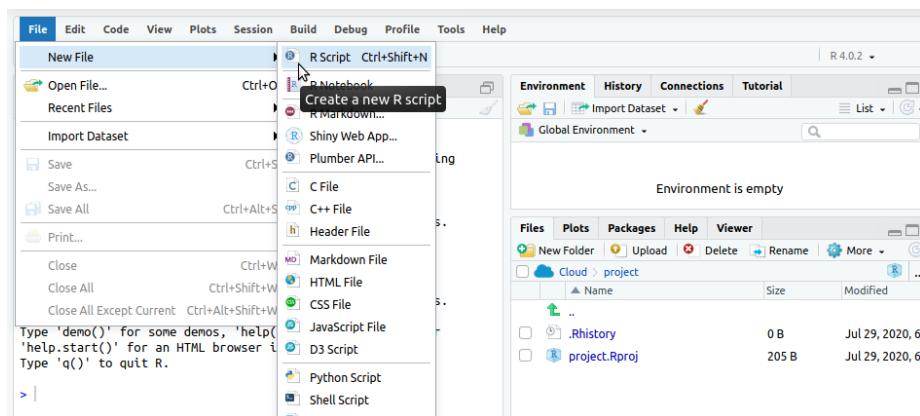


Haga clic en New Project y cuando aparezca la interfaz de Rstudio, coloque en

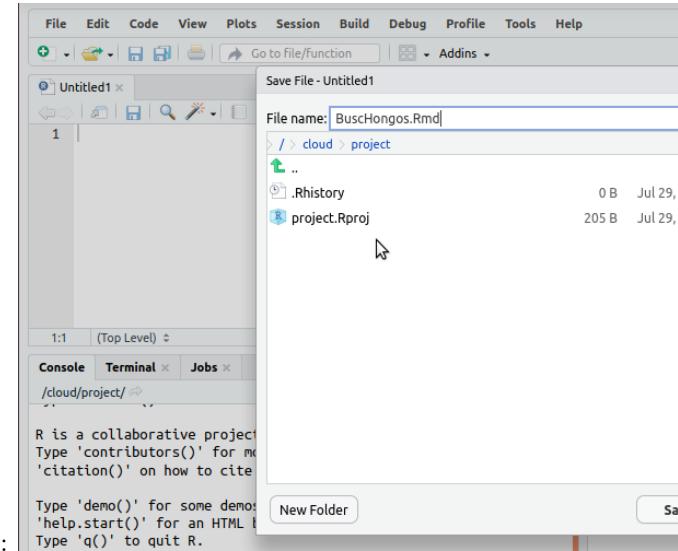
la parte superior el nombre al proyecto : BuscHongos:



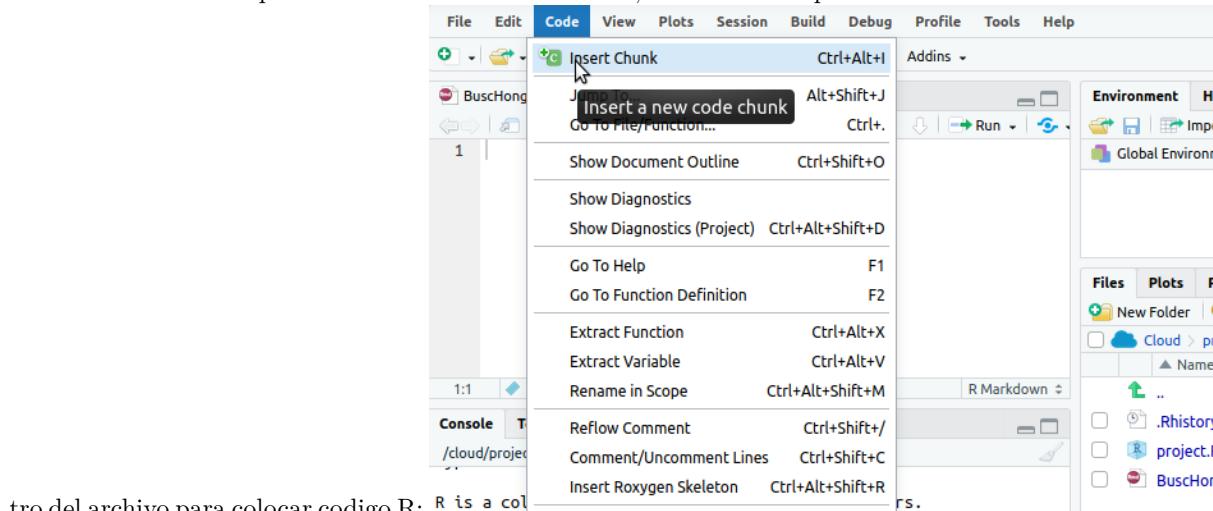
Cree un Nuevo Script (un Script de R es un archivo donde colocaremos los diferentes comandos para importar los datos y analizarlos):



Salve el Archivo con Nombre, BuscHongos.Rmd:

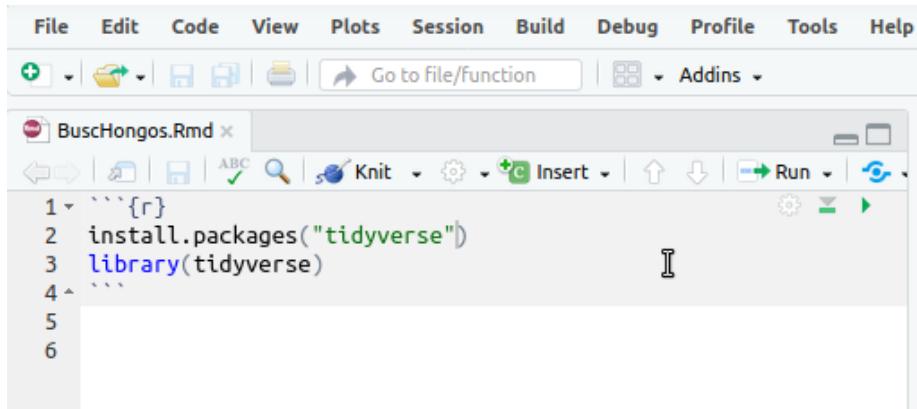


Vamos a importar la librería que usaremos para nuestro análisis (tidyverse), para ello seleccionemos la opción insert chunk del menu code, esto crea un espacio dentro del archivo para colocar código R:



Coloque las siguientes líneas dentro del chunk de código:

```
install.packages(tidyverse)
library(tidyverse)
```

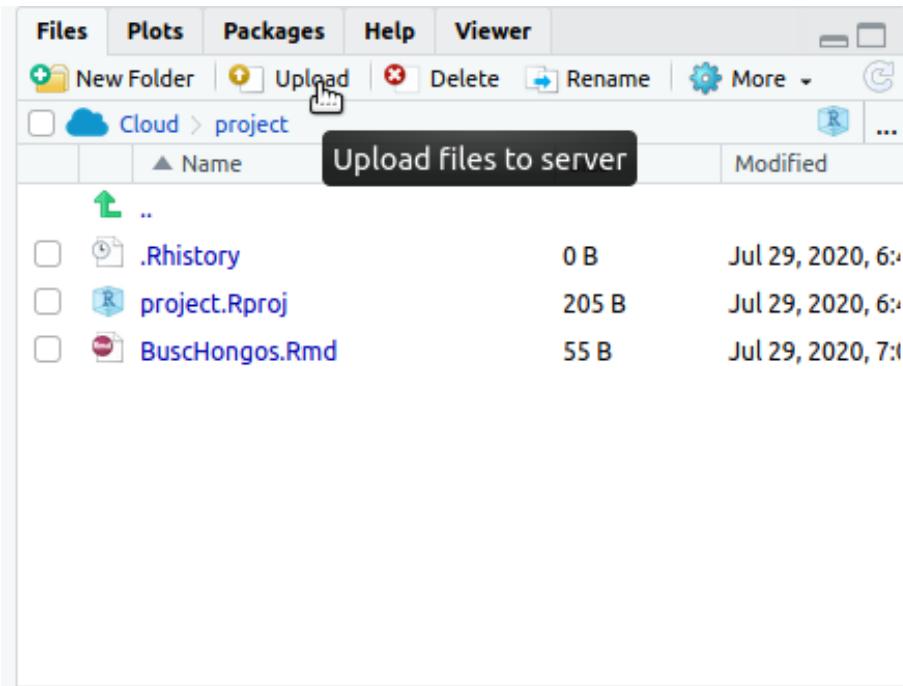


```

1 ````{r}
2 install.packages("tidyverse")
3 library(tidyverse)
4 ````
```

Sale el mensaje que el paquete tidyverse no está instalado haga clic en install y espere unos minutos que Rstudio instale los paquetes.

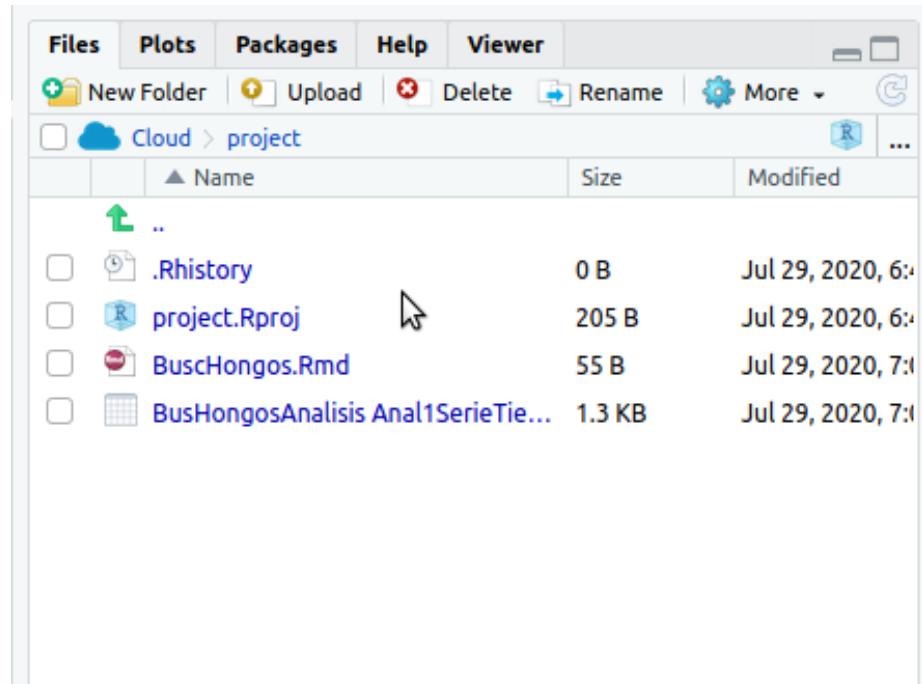
Finalmente hay que cargar el archivo de datos netLogo al proyecto Rstudio para ello haga clic en el botón Upload situado en la ventana inferior derecha de Rstudio:



The screenshot shows the RStudio interface with the 'Files' tab selected. In the top bar, the 'Upload' button is highlighted with a cursor. Below the toolbar, there's a list of files in the 'project' folder. The 'Upload files to server' button is also visible in the center of the screen.

Name	Modified
.Rhistory	Jul 29, 2020, 6:15 PM
project.Rproj	Jul 29, 2020, 6:15 PM
BuschHongos.Rmd	Jul 29, 2020, 7:15 PM

Busque el Archivo que generamos en Netlogo en su computador (“BusHongosAnalisis Anal1SerieTiempo-table.csv”) y carguelo, debe aparecer en Rstudio el archivo



cargado:

Ahora coloquemos en el Archivo Script lo siguiente:

```

File Edit Code View Plots Session Build Debug Profile Tools Help
+ | - | ↻ | ABC | Knit | ⚙ | Insert | Run | ⚙ |
BuscHongos.Rmd × data ×
1 ``{r}
2 install.packages("tidyverse")
3 library(tidyverse)
4 ``
5
6 1. Importar los Datos de NetLogo
7
8 ``{r}
9 data <- read_csv("BusHongosAnalisis Anal1SerieTiempo-table.csv",skip=6)
10 ``
11
12
10:4 (Top Level) R Markdown
Console Terminal × Jobs ×

```

Oprima el botón Play situado en la parte derecha del chunk que lle os datos,

debe aparecer a la derecha el archivo de datos (data), con la información : 2403 obs. of 4 variables, si oprima la fila donde aparece data puede observar los datos generados por NetLogo:

	[run number]	tiempo-espera	[step]	count hongos-encontrados
1	1	10	0	0
2	3	50	0	0
3	3	50	1	1
4	2	30	0	0
5	1	10	1	0
6	3	50	2	1
7	1	10	2	0
8	1	10	3	1
9	1	10	4	1

Showing 1 to 10 of 2,403 entries, 4 total columns

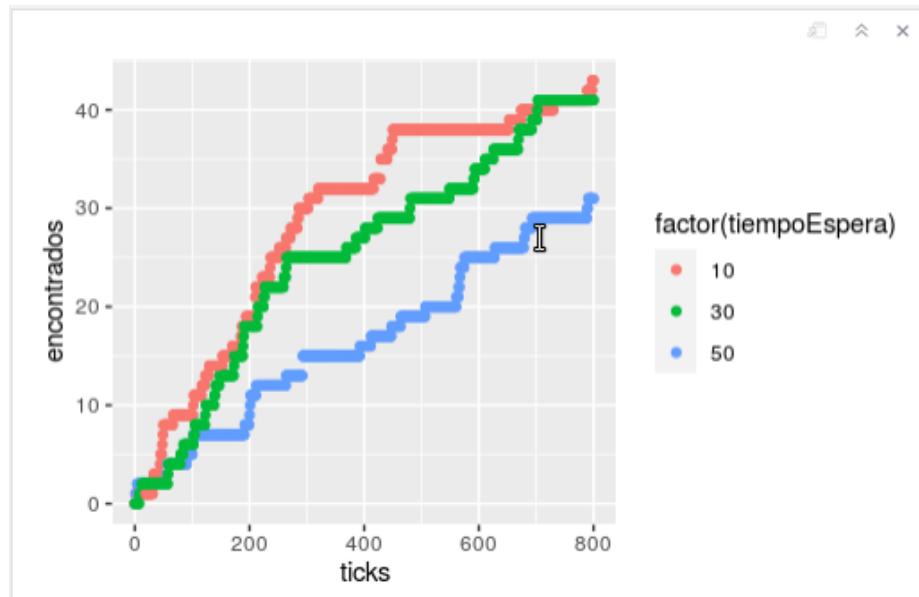
Renombremos las columnas con el comando:

```
colnames(data) <- c("run", "tiempoEspera", "ticks", "encontrados")
```

finalmente grafiquemos los datos:

```
ggplot(data) +
  geom_point(aes(ticks, encontrados,
                 group=tiempoEspera,
                 color=factor(tiempoEspera)))
```

Debe aparecer la siguiente gráfica:



Chapter 6

Construyendo Modelos Basados en Agentes

6.1 ¿Cómo modelar un MOBA?

Para modelar un sistema, la distancia entre el modelo y el sistema que se está modelando debe hacerse tan pequeño como sea posible. Este es un ejercicio desafiante por varias razones, se requiere muy buen conocimiento del sistema que se está modelando, adecuadas herramientas de modelado y se debe crear el modelo con estas herramientas, mientras que las restricciones inherentes al sistema a ser modelado deben ser tenidas en cuenta. Un modelo siempre busca abordar una pregunta de investigación. Esto es una condición previa requerida para la identificación de los elementos constituyentes del sistema objetivo. Para lograr esto, es necesario descubrir:

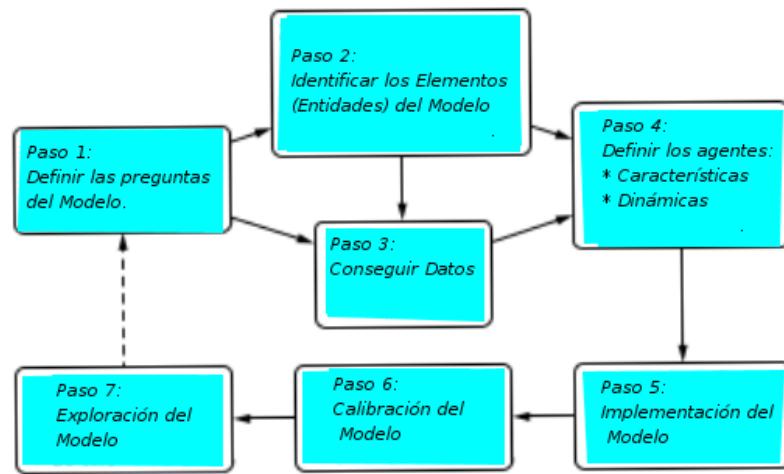
- los agentes del sistema.
- las interacciones entre ellos y su entorno-
- las entidades autónomas y sus comportamientos.
- el espacio también es de primordial importancia, y su topología y propiedades deben ser considerados

Las relaciones tienen un gran significado en el proceso, a menudo son la clave de la complejidad de los sistemas. Así, el modelo no necesita apuntar a cubrir todos los aspectos del sistema modelado. Más bien, puede concentrarse en sus particularidades. La validación del modelo es una etapa compleja en este proceso. ¿Cómo podemos estar seguros de que el modelo representa el sistema simulado?

Los resultados del modelo pueden usarse como base para esto, se pueden comparar con lo que sucede en el mundo real

El método paso a paso para construir un modelo basado en agentes es:

- 1) definición de las preguntas científicas que el modelo pretende resolver
- 2) identificación de los componentes del sistema objetivo;
- 3) recopilación de datos necesarios para construir el modelo;
- 4) definición de los agentes y el medio ambiente. Definición de las interacciones entre todos los elementos del modelo;
- 5) implementación del modelo;
- 6) calibración del modelo a través de simulaciones sucesivas;
- 7) exploración del modelo que responde las preguntas científicas, o redefinición de estas preguntas



Pasos del Modelado

6.2 El paradigma del agente

6.2.1 Conceptos básicos

Una de las definiciones más influyentes del concepto de agente, que es utilizado como punto de referencia por la comunidad de investigación , fue sugerido por Jacques Ferber en [FER 95]. De acuerdo con esta definición, el agente es una entidad física o virtual:

- que puede actuar en un entorno.
- que puede comunicarse directamente con otros agentes;
- que es impulsado por un conjunto de tendencias (en la forma de objetivos individuales o de satisfacción, tal vez incluso función de supervivencia, o de optimización).
- que posee sus propios recursos.
- que es capaz de percibir su entorno (en una manera limitada).

- que solo tiene una representación parcial de su entorno disponible (incluso puede no tener representación de la misma);
- que tiene habilidades y ofrece servicios.
- que posiblemente puede reproducirse;
- cuyo comportamiento tiende a satisfacer objetivos, teniendo en cuenta los recursos y habilidades a su disposición y en función de su percepción.

Esta definición establece las propiedades mínimas que una entidad debe tener para ser considerado un agente. Estas características pueden ser resumidas en cuatro palabras:

- autonomía: capacidad de evolucionar según su propio comportamiento sin intervención externa.
- reactividad: capacidad de reaccionar a eventos externos.
- proactividad: capacidad de tomar decisiones para lograr sus objetivos.
- sociabilidad: capacidad de interactuar con otros agentes.

En el mismo espíritu de la definición anterior, el concepto de un modelo basado en agentes considera los siguientes elementos:

- agentes: conjunto de entidades activas en el sistema que tienen sus comportamientos propios.
- entorno: medio en el que evolucionan los agentes. Su estructura depende del dominio de la aplicación. Sin embargo, a menudo se le otorga una métrica.
- interacciones: conjunto de idiomas y protocolos de intercambio entre los agentes. Estos a veces son de bajo nivel, originarios de la física, o de alto nivel, como actos de lenguaje.
- organización: conjunto de agrupaciones de agentes donde todos los agentes tienen un objetivo común

6.2.2 Interacciones

La riqueza de los MOBAs radica, en gran medida, en las interacciones que ocurren entre agentes. Estas interacciones se pueden expresar de muchos formas y se pueden resumir de la siguiente manera:

- los agentes pueden comunicarse entre sí;
- un agente proporciona un conjunto de servicios y los pone a disposición de todos los otros agentes en el sistema. *cada agente es responsable de limitar su accesibilidad a otros agentes.
- cada agente es responsable de definir sus relaciones, contratos,etc..., con otros agentes.

Por lo tanto, un agente directamente “sabe” (a través de su conjunto de conocimiento) todos los agentes con los que puede interactuar.

6.2.3 Tipos de agentes

Todos aquellos interesados en MOBA están de acuerdo en que, para fines pedagógicos, hay dos categorías principales de agentes :

- reactivos y cognitivos.

La primera categoría de agentes se basa en comportamientos simples que corresponden a una acción de estímulo respuesta. En contraste, la segunda categoría de agentes tiene genuinas facultades para reflexionar y adaptar su comportamiento, los agentes y los MOBAs para simular fenómenos complejos tanto como sea posible y a menudo se encuentran en la encrucijada entre la reacción y la cognición, o determinismo y estocasticidad y a menudo presentan una arquitectura que combina:

- reglas de comportamiento reactivo que se basan en estímulos recibidos o percibidos por el agente (eventos, mensajes, observaciones o leyes estocásticas), las reglas de comportamiento reactivo pueden aplicar algunas acciones o llamar a algunas funciones cognitivas de alto nivel.
- reglas de comportamiento cognitivo que utilizan algoritmos sofisticados y el conocimiento del agente, por ejemplo, un algoritmo de ruta más corto basado en un espacio estructurado en forma de grafo.

6.2.4 Paradigmas de organización de MOBAs

Como todos los sistemas distribuidos, hay dos tipos principales de control en los MOBAs:

- control centralizado: en el que un agente maestro gestiona el trabajo, organiza soluciones y media conflictos y..
- control distribuido, donde se dice que el sistema es evolutivo y donde cada agente tiene un plan de acción total o parcial.

En la práctica, no hay arquitecturas totalmente centralizados ó arquitecturas totalmente distribuidas las arquitecturas combinan ambos enfoques.

6.3 Otros tipos de modelado

Hay varios posibles enfoques para estructurar el modelo y la formalización del problema. Primero, podemos mencionar la matemática enfoques como ecuaciones diferenciales ordinarias (ODE) y parciales ecuaciones diferenciales (PDE), métodos estocásticos (como redes Bayesianas y cadenas de Markov).

Part IV

Un Modelo Incremental

Chapter 7

El Modelo Predador Presa

7.1 Descripción del Modelo

Introducción

Este taller tiene la intención de llevarlo desde el primer paso de diseñar una pregunta (hipótesis) de un área que se desea explorar, a través del diseño y la construcción del modelo y finalmente a refinar la pregunta revisar el modelo y analizar estadísticamente el modelo para responder a la pregunta (hipótesis). Esta secuencia se presenta en este documento en orden lineal, pero en realidad estos pasos se entremezclan entre sí y forman parte de una exploración iterativa de refinamiento del modelo y de la pregunta (hipótesis) motivadora inicial. Trabajaremos todo lo anterior dentro del contexto de un modelo particular, pero al mismo tiempo discutiremos cuestiones generales relacionadas con el diseño y construcción de modelos. Para facilitar el proceso, este taller está dividido en tres secciones principales:

- El diseño : que lo llevará a través del proceso de determinar qué elementos incluir en su modelo.
- La construcción : le mostrará cómo tomar un modelo conceptual y crear un objeto computacional
- El análisis: se ocupará de cómo ejecutar su modelo, crear algunos resultados y analizar esos resultados para proporcionar una respuesta a su pregunta inicial.

Pregunta Básica

A lo largo de este taller estaremos diseñando, construyendo y examinando un modelo simple de un sistema ecológico. La pregunta básica que vamos a abordar

es:

“¿Cómo los niveles de población de dos especies animales que comparten un hábitat cambian con el tiempo?”

Llamaremos a este modelo el modelo Predador-Presa. Aunque discutiremos este modelo en el contexto de dos especies biológicas, el modelo podría ser generalizado a otras situaciones como empresas que compiten por los consumidores, partidos políticos que compiten por los votos, o virus que evolucionan en un sistema informático. Más importante aún, los conceptos y componentes que vamos a desarrollar en este modelo son fundamentales y de hecho son utilizados en la mayoría de los modelos basados en agentes (ABM)

Diseñando el Modelo

Hay muchas maneras de diseñar un modelo basado en agentes. La que se elija depende de muchos factores incluyendo el tipo de fenómeno a ser modelado, el nivel de conocimiento del dominio de contenido, su comodidad con la codificación y su personal estilo de modelado. Consideramos dos grandes categorías de modelos:

- * Modelación basada en el fenómeno
- * Modelación exploratoria.

En la modelación basada en el fenómeno, se comienza con un fenómeno conocido que tiene un comportamiento característico. Algunos ejemplos podrían incluir patrones comunes de segregación de vivienda en las ciudades, galaxias en forma de espiral en el espacio, patrones de disposición de las hojas en las plantas, o niveles oscilantes de población en especies que interactúan. El objetivo de la modelación basada en el fenómeno es crear un modelo que capture de alguna manera el comportamiento. En ABM, esto se traduce a encontrar un conjunto de agentes, y reglas para esos agentes, que lo generan. Una vez constituido el modelo, se tiene un mecanismo para explorar el comportamiento. Variando los parámetros del modelo, se puede observar si emergen otros patrones o comportamientos encontrando esos nuevos patrones en un conjunto específico de valores de los parámetros del modelo o realizando experimentos con los datos.

La segunda categoría es la modelación exploratoria, en esta categoría la idea es crear un conjunto de agentes, definir su comportamiento y explorar los patrones o comportamientos que emergen. Pero para que tenga sentido este tipo de modelado, debemos encontrar similitudes de nuestro modelo con algunos fenómenos en el mundo. A continuación, refinamos nuestro modelo con destino a las similitudes percibidas con estos fenómenos y de esa manera llegamos a converger hacia un modelo explicativo del fenómeno.

Una distinción importante entre ambas metodologías es hasta qué punto especificamos una pregunta a ser contestada por un modelo. En un extremo del espectro

(Modelación basada en el fenómeno) , formulamos una investigación específica, pregunta (o conjunto de preguntas) como:

“¿Cómo una colonia de hormigas hace para buscar alimentación?”¿Cómo una bandada de estorninos hace para volar en forma de V? ”

En el otro extremo (Modelación exploratoria) simplemente comenzamos con una noción básica de querer modelar hormigas o comportamiento de aves, pero sin una pregunta clara a ser contestada. A medida que exploramos a través del modelo, gradualmente vamos refinando nuestra pregunta hacia una que pueda ser abordado por un modelo específico.

Sin embargo, una tercera dimensión tiene que ver con el grado en que el proceso de diseño del Modelo se combina con la codificación del modelo. En algunos casos, es aconsejable realizar todo el diseño del modelo conceptual antes de cualquier codificación del modelo. Esto es la metodología de arriba hacia abajo (top-down), el diseñador del modelo decide los tipos de agentes del modelo, el entorno en el que residen y sus reglas de interacción antes de escribir una sola línea de código. En otros casos, el diseño del modelo conceptual y la codificación del modelo co-evolucionan, influyendo cada uno en la evolución del otro. Esto es a menudo referido como diseño de abajo hacia arriba.(bottom-up) . En el diseño de abajo hacia arriba, se selecciona un dominio o fenómeno de interés con o sin especificar una pregunta formal, Utilizando este enfoque, se comenzaría a escribir el código relevante a ese dominio, construyendo el modelo conceptual de abajo para arriba, acumulando los mecanismos, las características y las entidades necesarias, y quizás formulando algunas preguntas formales de la investigación a lo largo del camino. Por ejemplo, en un diseño de abajo hacia arriba, puede comenzar con una pregunta sobre cómo evolucionaría un mercado económico, se podrían codificar comportamientos de algunos compradores y vendedores y, al hacerlo, darse cuenta de que se tendrían que agregar corredores como agentes en el modelo.

Estas dimensiones de diseño del modelo se pueden combinar . Usted podría comenzar con una pregunta de investigación muy específica y diseñar todos los agentes y reglas antes de codificar, o puede comenzar con algunos agentes, jugar con varias reglas para ellos y sólo obtener su pregunta de modelado cerca del final del proceso. En la práctica, los autores de modelos rara vez usan exclusivamente un estilo al construir sus modelos, pero usan una combinación de estilos, y a menudo cambian entre formas y estilos a medida que cambian sus necesidades e intereses de investigación. En los casos en que un investigador está colaborando con un programador que codifique el modelo, el estilo de diseño de arriba hacia abajo es normalmente el que se emplea, ya que separa los roles de los dos miembros del equipo. NetLogo fue diseñado para facilitar a los científicos la codificación de sus propios modelos. A menudo, a medida que los constructores de modelos se sienten más cómodos con la codificación, usan el código NetLogo como una herramienta para construir el modelo conceptual, presentaremos nuestro modelo de construcción utilizando una mezcla de enfoques, pero para mayor claridad de la exposición, haremos hincapié en el enfoque de arriba hacia abajo.

El proceso de diseño de arriba hacia abajo comienza eligiendo un fenómeno o situación que desea modelar o proponiendo una pregunta que quiera responder, y luego diseñar agentes y reglas de comportamiento que modelan los elementos de la situación. A continuación, se refina ese modelo conceptual y se revisar hasta que esté en un nivel de detalle lo suficientemente adecuado para que se puede comenzar a escribir el código para el modelo. A lo largo del proceso de diseño hay un principio muy importante que usaremos. Llamamos a este el principio de diseño de ABM:

Comienza simple y construye hacia la pregunta que deseas contestar

Hay dos componentes principales de este principio. La primera es comenzar con el conjunto más simple de agentes y reglas de comportamiento que pueden usarse para explorar el sistema que se desea modelar. Esta parte del principio se ilustra con una cita de Albert Einstein:

“El objetivo supremo de toda teoría es hacer que los elementos básicos irreductibles sean tan simples y pocos como sea posible sin tener que dejar de representar adecuadamente ningún dato de experiencia” (1933),

O en otra frase que se dice que también dijo:

“Todo debe ser lo más simple posible, pero no más simple.”

En el caso de ABM, esto significa hacer el modelo tan simple como sea posible dado que debe proporcionarle un escalón hacia su destino final. El estadístico George Box proporciona una cita que ilustra este punto:

“Todos los Modelos son incorrectos, pero algunos modelos son útiles”(1979).

Lo que quería decir Box era que todos los modelos son por necesidad incompletos porque simplifican aspectos del mundo. Sin embargo, algunos de ellos son útiles porque están diseñados para responder a preguntas particulares y las simplificaciones en el modelo no interfieren con la obtención de esa respuesta. Este principio básico de diseño ABM es útil de varias maneras. En primer lugar, nos obliga a examinar cada agente y cada regla y eliminar elementos si el progreso se puede hacer sin ella. No es raro que los modeladores principiantes construyan un modelo en el que ciertos componentes no tengan ningún efecto. Al comenzar simple y lentamente agregar elementos a su modelo, puede asegurarse de que estos componentes extraños nunca se desarrollan. Al examinar la necesidad de cada componente adicional con respecto a la pregunta que se está persiguiendo, se reduce la tentación de, parafraseando a Guillermo de Occam, “multiplicar las entidades innecesariamente”Al hacerlo, se reduce la posibilidad de introducir ambigüedades, redundancias e inconsistencias en el modelo. Otra virtud del principio de diseño es que, al mantener el modelo simple, lo hace más comprensible y más fácil de verificar. La verificación es el proceso de asegurar que un modelo computacional implemente fielmente su modelo conceptual objetivo. Un modelo conceptual más simple conduce a una implementación más sencilla del modelo, lo que facilita la verificación del modelo. En cada punto del proceso de desarrollo del modelo, el modelo debe ser capaz de proporcionarle algunas respuestas a su

pregunta de investigación. Esto no solo le ayuda a hacer un uso productivo de su modelo desde el principio, sino que también le permite empezar a cuestionar las suposiciones de modelo y examinar sus resultados desde el principio en el proceso de modelado. Esto puede evitar que usted vaya demasiado lejos por un camino improductivo. Menos componentes también significan menos combinaciones para “validar”.

Predador-Presa

Para aplicar nuestro principio al contexto del modelo Predador-Presa , debemos comenzar nuestro diseño reflexionando sobre dos especies animales que comparten un hábitat e identificando agentes y comportamientos simples para nuestro modelo. Comenzaremos identificando una pregunta que queremos explorar, Después de eso vamos a discutir que agentes debemos definir y cómo pueden interactuar. Luego nos trasladamos al medio ambiente y a sus características. Como parte de este proceso, necesitamos discutir lo que sucede con el modelo a lo largo del tiempo. Finalmente, discutimos qué medidas vamos a utilizar para responder a nuestra pregunta.

Comencemos reflexionando sobre dos especies animales que comparten un hábitat e identificando agentes y comportamientos simples para nuestro modelo. Comenzaremos identificando una pregunta que queremos explorar, Después de eso vamos a discutir que agentes debemos definir y cómo pueden interactuar. Luego nos trasladamos al medio ambiente y a sus características. Como parte de este proceso, necesitamos discutir lo que sucede con el modelo a lo largo del tiempo. Finalmente, discutimos qué medidas vamos a utilizar para responder a nuestra pregunta

** Encontrando las preguntas

La elección de una pregunta puede parecer una cuestión aparte del diseño del modelo. Después de todo, la progresión natural parece ser: primero escoja una pregunta y construya un modelo para responder a esa pregunta. A veces, ese puede ser el procedimiento que seguimos, pero en muchos casos **tendremos que refinarnas nuestras preguntas** cuando comencemos a pensar en ello de una manera basada en agentes. Nuestra pregunta original para el modelo Lobos-Ovejas puede ser:

“¿Cómo cambian los niveles de población de dos especies con el tiempo cuando coexisten en un hábitat compartido?”

Ahora vamos a evaluar si esta pregunta es una que es susceptible para usar ABM y en caso afirmativo tendremos que refining nuestra pregunta dentro del paradigma ABM. El modelado basado en agentes es particularmente útil para dar sentido a los sistemas que tienen un número de entidades que interactúan, y por lo tanto tienen resultados impredecibles. Hay ciertos problemas y preguntas que son más susceptibles a las soluciones ABM. Si nuestra pregunta principal

de interés viola nuestras directrices, puede ser una indicación de que debemos considerar un método de modelado diferente. Por ejemplo, podríamos estar interesados en examinar la dinámica de dos poblaciones muy grandes bajo el supuesto de que las especies son homogéneas y bien mezcladas (sin componentes espaciales o propiedades heterogéneas) y que el nivel de población de cada especie depende simplemente del nivel de población de las otras especies. Si este es el caso entonces podríamos haber usado un modelo basado en ecuaciones en lugar de un modelo basado en agentes ya que el trabajo de EBM (Modelos basados en ecuaciones) es bueno para grupos homogéneos grandes y existe un EBM clásico para esta situación conocida como as ecuaciones diferenciales de **Lotka-Volterra** (Lotka, 1925, Volterra, 1926) ABM será más útil para nosotros si pensamos en los agentes como **heterogéneos y con ubicaciones espaciales**. Un aspecto de los animales que es probable que sea relevante para nuestra pregunta es cómo hacen uso de sus recursos. Los animales hacen uso de los recursos al convertirlos en energía, por lo que queremos asegurarnos de que nuestros agentes tienen diferentes cantidades de energía y ocupan diferentes lugares en el mundo. Una tercera directriz es considerar si los resultados “agregados” dependen de las interacciones de los agentes y de la interacción de los agentes con su entorno. Por ejemplo, si una especie está consumiendo otra, entonces los resultados dependerán de la interacción del agente. Las interacciones predador-presa suelen establecerse en ambientes abundantes. Manteniendo el principio de diseño ABM en mente, comenzamos con el ambiente **más simple**:

Enriquecemos el ambiente un poco más allá de los depredadores y las presas e incluyendo los recursos del medio ambiente que consumen las especies de presa de nivel más bajo. Otra directriz es que el modelado basado en agentes es más útil para modelar procesos dependientes del tiempo. En el modelo Predador-Presa, nuestro interés central radica en examinar cómo los niveles de población cambian con el tiempo. Por lo tanto, podemos refinar nuestra pregunta para centrarnos en las condiciones que conducen a las dos especies a coexistir durante algún tiempo. De esta forma, nuestras pautas nos ayudan a evaluar si nuestra pregunta es adecuada para ABM y, en caso afirmativo, enfocar nuestra pregunta y nuestro modelo conceptual. Habiendo evaluado la relevancia de nuestra pregunta para ABM, estamos ahora en posición de decirlo más formalmente:

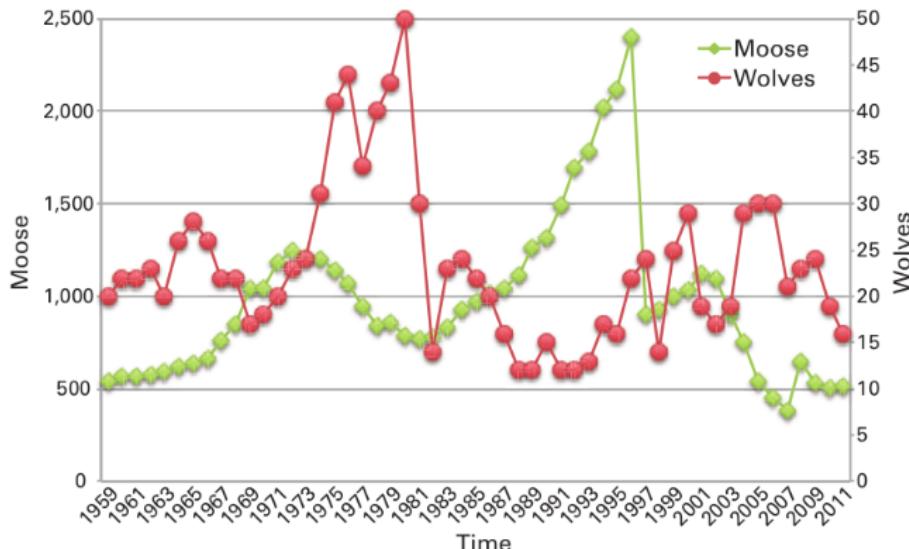
** “¿Podemos encontrar parámetros del modelo para dos especies tal que sostengan niveles de población positivos en un área geográfica limitada cuando una especie es un depredador del otro y la segunda especie consume recursos del medio ambiente?”**

Ahora, teniendo en cuenta esta pregunta, procederemos a diseñar el modelo conceptual.

7.2 Diseño del Modelo

Concretando el Modelo

Ahora que hemos identificado nuestra pregunta de investigación en detalle, puede ser útil considerar un contexto particular para esta pregunta de investigación. Anteriormente, discutimos los patrones de referencia como una fuente de modelos basados en fenómenos A veces ese patrón de referencia es la inspiración original para el modelo. Otras veces, como ahora, hemos refinado nuestra pregunta de investigación lo suficiente como para buscar un patrón de referencia que nos ayude a probar si nuestro modelo es una respuesta válida a la pregunta. En el caso de las relaciones depredador-presa, hay un caso famoso de cohabitación de pequeñas poblaciones de depredadores-presas en un área geográfica pequeña. Este es el caso de poblaciones fluctuantes de lobos y alces en Isle Royale, Michigan.



La figura muestra las poblaciones de lobos y alces en la Isla Royale desde 1959 hasta 2009. Esta gráfica puede servir como patrón de referencia para nuestro modelo. Nuestro modelo ya completado deberá ser capaz de generar un gráfico “similar” como posible comportamiento del Modelo.

Como podemos ver en los datos de la Isla Royale, las poblaciones de lobos y alces de la Isla Royale se han mantenido durante más de cincuenta años sin que ninguna de las especies se extinga. Las poblaciones también exhiben una oscilación, con el alce en un punto bajo cuando los lobos están en altos y viceversa. Estos datos pueden servir como patrón de referencia para nuestro modelado y nos permite refinar aún más nuestra pregunta de investigación:

“¿Podemos encontrar parámetros de modelo para dos especies que sostengan los niveles poblacionales positivos oscilantes en un área

geográfica limitada cuando una especie es un depredador del otro y la segunda especie consume recursos del medio ambiente?"

Para nuestro modelo en vez de modelar lobo y alce, modelaremos lobos y ovejas. El conjunto de datos de lobos y alces está bien establecido, pero nuestro objetivo en este capítulo no es coincidir con estos datos en particular, sino presentarles ejemplos clásicos del modelado predador-presa y tratar de reproducir el patrón sostenido oscilante de los niveles poblacionales.

Seleccionando los Agentes

Ahora que hemos identificado nuestra pregunta de investigación, podemos comenzar a diseñar los componentes que nos ayudarán a responderla. La primera pregunta que debemos hacernos es: ¿Cuáles son los agentes en el modelo? Al diseñar nuestros agentes, queremos elegir aquellos componentes de nuestro modelo que sean autónomos y que tengan propiedades, estados y comportamientos que puedan tener relación con nuestra pregunta. Pero debemos tener cuidado de evitar la sobrecarga del agente. Dependiendo de la perspectiva que se tome, casi cualquier componente del modelo podría ser considerado un agente. Sin embargo, un modelo que está diseñado con un exceso de clases de agentes puede llegar a ser rápidamente inmanejable. Al elegir los agentes en un modelo, es importante concentrarse en aquellas entidades autónomas que son más relevantes para nuestra pregunta de investigación.

Un problema relacionado es la “granularidad” del agente. Cada entidad está compuesta de múltiples entidades más pequeñas. ¿Cuál es el nivel adecuado de entidad para elegir? ¿Deberían nuestros agentes ser moléculas o átomos? ¿Órganos o células del cuerpo? . Si queremos modelar un campo de pasto, tal vez no queramos modelar cada hoja de pasto, sino que en lugar de eso, elegir “grumos” de pasto como nuestros agentes. Es importante que la granularidad de cada agente esté aproximadamente al mismo nivel.

Teniendo en cuenta lo anterior, comenzamos eligiendo tres tipos de agente. Modelamos los depredadores, que llamaremos lobos, las presas que llamaremos ovejas, y los recursos que las ovejas consumen, el pasto.

Nosotros podríamos tener muchos otros agentes en este modelo. Por ejemplo, podríamos modelar un cazador o los niveles de precipitación o de nutrición del suelo. Sin embargo, al elegir sólo lobos, ovejas y pasto, nos atenemos al principio de diseño ABM (keep it simple). Tenemos dos tipos de agentes móviles simples y un tipo de agente estacionario para modelar el ambiente, y éstos son el conjunto mínimo de agentes necesarios para responder a nuestra pregunta de qué parámetros permitirán que coexistan dos poblaciones en un área geográfica limitada.

Seleccionando Propiedades de los Agentes

Los agentes tienen propiedades que los distinguen de otros agentes. Es importante determinar estas propiedades con antelación para que podamos conceptualizar el agente y diseñar las interacciones entre ellos y con el medio ambiente. En el modelo Predador-Presa damos a las ovejas y a los lobos tres propiedades :

- (1) un nivel de energía, que define el nivel de energía del agente.
- (2) un lugar, que es un lugar en el área geográfica donde el agente está-
- (3) una dirección, que indica hacia donde está mirando y la dirección que el agente tomaría en caso de que se mueva.

La propiedad energética no describe meramente energía temporal (como si un animal está fresco o fatigado). Más bien, la “energía” incorpora alguna noción de la cantidad de “vitalidad” en una criatura, abstrayendo los detalles del metabolismo, el almacenamiento de calorías o el hambre y condensándolo todo en una sola medida. Podríamos agregar propiedades adicionales y algunas de ellas podrían ser útiles para futuras extensiones, por ejemplo, podríamos añadir una velocidad de movimiento y permitir que agentes se muevan a distintas velocidades, o una capacidad de ofensa / defensa que afecte la capacidad del individuo resistirse a la depredación. Sin embargo, estas propiedades adicionales no parecen necesarias para responder a nuestra pregunta más simple, y por lo tanto, resistimos a la tentación de incluirlas innecesariamente. Si las ovejas y los lobos tienen exactamente las mismas propiedades, ¿qué los hace diferentes unos de otros? Discutiremos esto más adelante, donde hablamos sobre el comportamiento que cada uno de estos dos tipos de agentes exhiben.

Características del ambiente y parcelas

Ahora que definimos los agentes móviles y sus comportamientos ,pensemos en el entorno en el que vivirán estos agentes móviles y cómo pueden interactuar con ese entorno. En el modelo Predador-Presa, el primer atributo ambiental obvio es la presencia o ausencia de pasto, ya que es lo que consumen las ovejas. Podríamos modelar muchos otros atributos tales como la elevación, el agua, los bosques y otras características que podrían afectar el movimiento de los animales o la depredación de las ovejas. Sin embargo, de acuerdo con nuestro principio de diseño, comenzamos con un entorno que consiste en un gran campo de pasto. Usamos los tipos de agente llamados parcelas (típicas de Modelos Basados en Agentes) para modelar el pasto.

Como se mencionó, no tendría sentido modelar cada hoja de pasto, por lo que el modelo podría dar a las parcelas una propiedad “cantidad de pasto” que tendrá un valor numérico. Esto es efectivamente utilizar las parcelas para modelar montones de pasto, que es nuestro tipo de agente estacionario. El valor numérico de esta propiedad debe ser proporcional al comportamiento alimenticio de las ovejas, ya que así lo utilizaremos en el modelo. Con el fin de evitar tratar

con las condiciones de frontera (como lobos que superan los límites del mundo modelado), el mundo se “envolverá” horizontal y verticalmente, por lo que un lobo saliendo del borde derecho del mundo aparecerá a la izquierda. Esta topología en forma de toro es a menudo muy conveniente para ABMs. También vale la pena señalar que en algunas ABMs el entorno también controla los procesos de nacimiento y muerte de los agentes. En este modelo el nacimiento y la muerte serán modelados endógenamente dentro de las acciones de los agentes, pero es posible tener nacimiento y muerte de agentes controlados por el ambiente . Esta es una forma menos “emergente” de modelar el ciclo de vida, pero a veces es una simplificación útil.

Comportamiento de los agentes

Además de definir las **propiedades** de los agentes es importante determinar qué tipo de **comportamiento** pueden exhibir los agentes. Estos comportamientos son necesarios para describir cómo los agentes interactúan entre sí y con el medio ambiente.

En el modelo Predador-Presa, las ovejas y los lobos comparten muchos comportamientos comunes, ambos tienen la capacidad de girar al azar, avanzar, reproducirse y morir. Sin embargo, ovejas y lobos **difieren** en que las ovejas tienen la capacidad de consumir pasto y lobos tienen la capacidad de consumir ovejas. Esto diferencia las dos especies (breeds) ó tipos de agente entre sí. Por supuesto, una vez más hay muchos otros comportamientos que podríamos prescribir para estos agentes. Por ejemplo, podríamos dar a las ovejas la capacidad de esconderse en los rebaños para defenderse contra ataques de lobo, o la capacidad de luchar. Los lobos podrían tener la capacidad de moverse a diferentes velocidades para perseguir ovejas. Lobos y ovejas también pueden tener comportamientos comunes como: dormir, digerir los alimentos y buscar refugio durante una tormenta. Sin embargo, nuevamente los comportamientos que hemos descrito (Mover, reproducir, comer y morir) son opciones razonables para un modelo simple que pueden abordar nuestra pregunta de investigación. Para los agentes herbáceos (pasto) , damos un simple comportamiento, la capacidad de crecer.

Manejo del Tiempo

Ahora que hemos establecido los componentes básicos del modelo,podemos diseñar el paso de tiempo en el modelo. Para ello necesitamos pensar en los comportamientos que serán exhibidos por los agentes de nuestro modelo y decidir cómo y en qué orden se deben realizar. En el mundo real, los animales se comportan concurrentemente y el tiempo parece continuo. Para construir nuestro ABM, simplificaremos esto dividiendo el tiempo en pasos discretos **ticks**, además dividimos cada paso en fases ordenadas y serializadas. Haciéndolo de esta manera, estamos haciendo un supuesto implícito de que al definir un orden

para realizar las acciones esto no afectará sustancialmente nuestros resultados. Esto es un supuesto de trabajo y puede que tenga que ser reexaminado más tarde.

En general, la determinación del orden en que los agentes muestran comportamientos puede ser complicado.

En el modelo hay cuatro comportamientos animales básicos:

- mover
- morir
- comer
- reproducir

y un comportamiento del pasto:

- crecer

Otra hipótesis de trabajo que podemos mirar, es decidir qué el orden en que los animales realizan estos comportamientos puede ser arbitrario. Cualquier orden para los comportamientos sería razonable y es mucho más fácil trabajar con un orden fijo de comportamientos. Elegimos ordenar los comportamientos como en la primera oración de este párrafo (mover, morir, comer, reproducir), podríamos chequear y asegurarnos si este orden tiene sentido.

El movimiento es el acto de girar y luego dar un paso adelante. Dado que la acción de movimiento cambia la ubicación de los agentes y por lo tanto cambia el entorno local de cada agente, tiene sentido moverse primero, el movimiento cuesta energía y así programaremos la muerte a continuación, porque debemos comprobar para ver si alguno de los agentes ha gastado tanta energía mientras se mueve que ya se queda sin energía. Después de esto, programaremos a los agentes para que intenten obtener nueva energía, si hay algo en su entorno local que puedan comer. Ya que ahora tienen nueva energía y los agentes pueden reproducirse (ya que esto también requiere energía). Por lo tanto, cada agente verifica si tienen suficiente energía para crear un nuevo agente. Finalmente, dado que el modelo ha hecho todo lo demás, los agentes de pasto crecen. Y el ciclo de la vida termina

Seleccionando los parámetros del modelo

Podríamos decidir escribir un conjunto de reglas completamente específicas para controlar el comportamiento de todos estos agentes y sus interacciones ambientales durante un paso de tiempo, pero tiene más sentido **crear algunos parámetros** que nos permitan controlar el modelo, para que podamos fácilmente examinar diferentes condiciones. Un paso siguiente es definir qué atributos del modelo podremos controlar a través de parámetros.

Existen varios parámetros posibles de interés en el modelo, por ejemplo, queremos ser capaces de controlar el número de ovejas y lobos iniciales. Esto nos permitirá

ver cómo los diferentes valores de los niveles de población inicial afectan los niveles finales de población. Otro factor a controlar es la cantidad de energía que cuesta a un agente para moverse. Usando este parámetro, podemos hacer que el paisaje sea más o menos difícil de recorrer, y así simular diferentes tipos de terreno. Relacionado con el costo de movimiento tenemos la energía que cada especie gana de lo que consume. Así, elegimos tener parámetros para controlar la energía que se obtiene del pasto.

Por último, dado que las ovejas consumen pasto, para mantener a la población en el tiempo queremos que el pasto vuelva a crecer. Así que necesitaremos un parámetro para la tasa de rebrote del pasto.

Hay muchos otros parámetros que podríamos haber incluido en este modelo. Por ejemplo, los parámetros que elegimos son homogéneos en todo el modelo. En otras palabras, una oveja ganará la misma energía del pasto que cualquier otra oveja. Sin embargo, podríamos hacer este modelo más heterogéneo al extraer la ganancia de energía para cada oveja de una **distribución estadística** y tener dos parámetros de modelo que controlan la media y la varianza. También podríamos agregar parámetros para controlar aspectos que actualmente estamos considerando como valores constantes. Por ejemplo, no creamos un parámetro para controlar la velocidad de los agentes. Tener la capacidad de modificar esas velocidades y (particularmente la relación entre las tasas de movimiento de lobos y ovejas) podría afectar **dramáticamente** al modelo. Sin embargo, guiado por nuestro principio de diseño ABM (KISS), esta complicación no parece necesaria en esta etapa del proceso de modelado. Permitir diferentes velocidades de movimiento para lobos y ovejas es una expansión en este modelo que se deja para que el estudiante avanzado.

Las mediciones

Siimplementamos todos los componentes anteriores, tendríamos un primer modelo de trabajo. Sin embargo, todavía no tendríamos nada para responder a nuestra pregunta. Para ello debemos decidir qué medidas recogeremos del modelo. Crear medidas puede ser muy simple a veces, pero a menudo algunas de los resultados más interesantes de un modelo no se perciben hasta después de que se hayan diseñado las mediciones del modelo, cuando se considera qué medidas incorporar en el modelo es entonces útil revisar la pregunta de investigación.

Es aconsejable incluir sólo las medidas más relevantes, porque un exceso de medidas puede ser abrumador y asfixiante y puede hacer lenta la ejecución del modelo. En nuestro modelo, las medidas más relevantes son:

la población de lobos y ovejas en el tiempo

ya que lo que nos interesa es qué conjuntos de parámetros nos permitirá mantener niveles positivos de ambas poblaciones a lo largo del tiempo. Podríamos construir medidas de muchos otros datos en este modelo, tales como la energía promedio de ovejas o lobos. Esto podría afectar a nuestra pregunta, ya que podría indicar

la probabilidad de que persistan las poblaciones actuales, pero no se dirige directamente a la pregunta y entonces esta medida no la incluiremos. A veces es útil incluir medidas como ésta para propósitos de depuración. Por ejemplo, si vemos que los niveles de energía de ovejas estaban aumentando a pesar de que no había crecimiento de pasto, entonces nos gustaría ver si hubo un error en la parte donde convertimos pasto en energía para la oveja.

Resumen

1. Pregunta Orientadora:

“¿Bajo qué condiciones dos especies pueden mantener niveles poblacionales positivos oscilantes en un área geográfica limitada cuando una especie es un depredador del otro y la segunda especie consume recursos del medio ambiente?”

2. Tipos de agentes Ovejas, lobos, pasto

3. Propiedades de los agentes:

- Energía, ubicación, dirección (lobos y ovejas), cantidad de pasto (pasto).

4. Comportamiento de los agentes:

- Mover, morir, reproducir (lobos y ovejas), comer ovejas (solo lobos), comer pasto (solo ovejas), Reproducir (pasto).

5. Parámetros

Número de ovejas, Número de lobos, Costo del movimiento, Ganancia de energía del pasto, Ganancia de energía de las ovejas, tasa de crecimiento del pasto

6. Tiempo

En cada tick :

1. Movimiento de ovejas y lobos

2. Ovejas y lobos mueren

3. Las ovejas y los lobos comen

4. Las ovejas y los lobos se reproducen

5. El pasto vuelve a crecer

6. Mediciones:

- población de ovejas versus tiempo
- población de lobos versus tiempo.

Al diseñar un modelo es bastante útil escribir notas para usted mismo, como lo hemos hecho en esta sección. Encontrará estas notas invaluables después de haber dejado el modelo por un tiempo, ya que usted podrá regresar y recordar

por qué tomó ciertas decisiones y opciones alternativas. Además, para explicar su modelo a otras personas, es muy útil tener dicha documentación sobre el modelo. Finalmente, se recomienda poner fechas a sus notas a medida que las crea para poder seguirle el paso a su modelo.

Por ejemplo, si está utilizando el proceso de diseño de arriba hacia abajo (top-down) que acabamos de discutir, entonces puede revisar el siguiente conjunto de preguntas y escribir las respuestas a ellas como guía provisional sobre cómo construir su modelo:

7.2.1 Siete Criterios de Diseño:

1. ¿De qué parte de tu fenómeno te gustaría construir un modelo? (Alcance / Pregunta)
2. ¿Cuáles son los principales tipos de agentes involucrados en este fenómeno? (Agentes)
3. ¿Qué propiedades tienen estos agentes (por tipo de agente)? (Propiedades)
4. ¿Qué acciones o comportamientos pueden tomar estos agentes (por tipo de agente)? ¿Cómo interactúan entre sí o con el medio ambiente? (Comportamientos)
5. ¿En qué tipo de entorno operan estos agentes? Hay agentes estacionarios? (Ambiente)
6. Si tuviera que definir el fenómeno como pasos de tiempo discretos(ticks), qué eventos ocurrirían en cada paso de tiempo y en qué orden?
7. ¿Qué esperas observar de este modelo? (Entradas y salidas)

7.3 Cinco modelos incrementales

7.3.1 Version Uno

Introducción

Concluido el proceso de diseño conceptual debemos ahora entrar en el proceso de construcción del modelo, en esta etapa continuaremos aplicando los principios de diseño de un MOBA. Aunque nuestro modelo es bastante simple, lo dividiremos en una serie de submodelos que implementaremos en cinco iteraciones. Los submodelos serán **ejecutables** y nos permitirán construir y completar el modelo paso por paso, verificando nuestro progreso en el camino y asegurándonos de que el modelo funciona como lo esperamos.

Muchas veces, en modelos basados en agentes (MOBAs), los resultados finales no son los que esperamos, esto puede suceder por un error la implementación del modelo pero a menudo no es nuestra implementación ni que nuestro diseño esté

mal, sino que surge de una propiedad fundamental de los sistemas complejos: el comportamiento **emergente** que es difícil de predecir. Al construir nuestro modelo de manera gradual, podremos observar comportamientos inusuales pronto y determinar más fácilmente su causa que si hubiéramos construido el modelo completo de una vez. Por lo tanto, el principio de diseño MOBAs (*KISS:Keep It Simple Stupid*) todavía se aplica a lo largo de toda la construcción del modelo.

Primera Versión

¿Cuál es el modelo más simple que podemos crear que muestre algún tipo de comportamiento?

Una simplificación que podemos hacer del modelo total es mirar solo **una especie** e ignorar la otra especie y el medio ambiente. Dadas estas simplificaciones, el modelo más simple tendría, por ejemplo, algunas ovejas deambulando por el paisaje. Para hacer esto, vamos a crear dos procedimientos, un procedimiento de **setup**, que crea las ovejas, y un Procedimiento **go** que las hace moverse en el paisaje.

Antes de ello, crearemos una raza de ovejas :

```
breed[ovejas oveja]
```

Esto define una clase de agentes móviles (en NetLogo, tortugas) llamada ovejas, Primero se da la forma plural “ovejas”, seguida de la forma singular “oveja”. Casi siempre se usará la forma plural (ovejas), pero es útil proporcionar la forma singular para poder referirse a una oveja en particular y además para que NetLogo pueda darle mensajes de error más significativos, entre otras cosas. Una última cosa a tener en cuenta es que, aunque estamos creando la raza “ovejas”, el conjunto de agentes turtles todavía existe. Todos los agentes móviles en un modelo NetLogo pertenecen a la clase “turtles” independiente de su raza. Entonces, si desea pedir a todos los tipos de agentes de un modelo que hagan algo, es decir, tanto OVEJAS como LOBOS, entonces puedes colocar “ask turtles”. Si solo quieres que hagan algo las ovejas, entonces coloca “ask ovejas”.

```
to setup
  clear-all
  ask patches [ ;; colorea de verde el mundo
    set pcolor green
  ]
  create-ovejas 100 [ ;; crea algunas ovejas
    setxy random-xcor random-ycor
    set color white
    set shape "sheep"
  ]
  reset-ticks
end
```

Descripción del setup

Este procedimiento setup terminará siendo el procedimiento más largo del modelo terminado, pero su descripción es bastante sencilla. Primero, se limpia el mundo. El mundo de nuestro modelo es la representación de todos los agentes, incluidos los agentes móviles (por ejemplo, tortugas, ovejas, lobos), así como los agentes estáticos (por ejemplo, parcelas, pasto), el comando **clear-all** relimpiá el mundo y lo prepara para una nueva ejecución.

Después de esto, se les pide a todas las parcelas que establezcan su color de parcela (**pcolor**) a verde para representar el pasto. Aunque nuestro modelo aún no incluye ninguna propiedad para el pasto ni ninguna regla para que las ovejas interactúen con él, cambiar el color ayuda a la visualización.

Finalmente, creamos cien ovejas, cuando creamos las ovejas les definimos algunas propiedades iniciales: * Les asignamos una coordenada x aleatoria y un coordenada y aleatoria para extenderlas por todo el mundo. * Establecemos su color en blanco y su forma a la forma de “oveja” (“sheep”) para que se vean un poco más como ovejas reales.

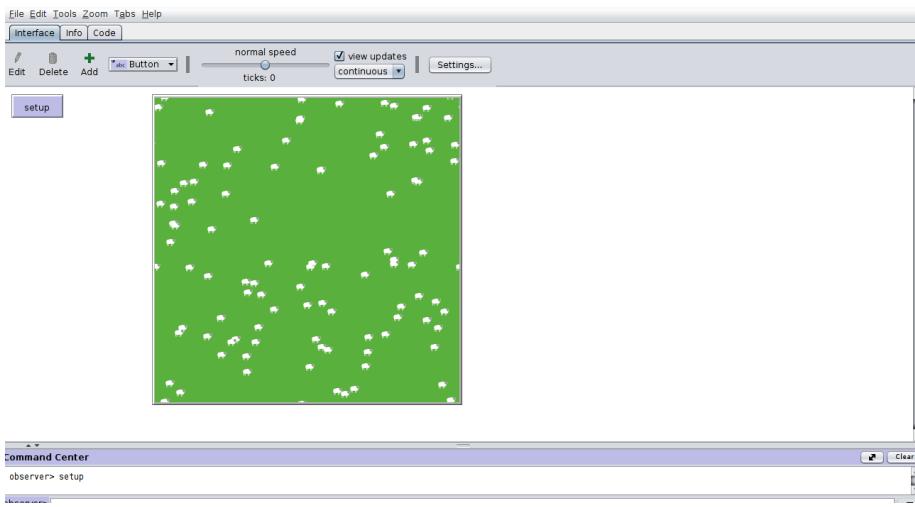
La linea final, (**reset-ticks**) , inicia el reloj en 0 para que el modelo esté listo para comenzar su ejecución.

Documentación

Documentar los procedimientos dentro de su modelo (use punto y coma para comentar el código) es muy útil. Cualquier texto escrito después de un punto y coma se ignora cuando el modelo se encuentra en ejecución, agregar texto de esta manera se llama “comentar” su código. Sin estos comentarios, no solo es bastante difícil para otra persona leer y entender el modelo, sino que también se volverá cada vez más difícil, a medida que pasa el tiempo, que ud mismo entienda su propio código. Un modelo sin comentarios (u otra documentación) no es muy útil, ya que será difícil que otros descubran el comportamiento del modelo.

Validando el setup

Para verificar que el procedimiento setup hace lo que esperamos, vamos a correr el modelo, para ello vaya a la pestaña interfaz, cree un botón llamado “setup” y oprima el botón debe observar lo siguiente:



7.3.1.1 Procedimiento go

Después de haber creado las ovejas, pasamos a escribir el procedimiento “go” que indicará a las ovejas como deben comportarse en el mundo. Mirando hacia atrás en nuestro documento de diseño, uno de los principales comportamientos de las ovejas es su movimiento, así que haremos simplemente que se muevan. Dividiremos el movimiento en dos partes giro y avance. Creemos los dos procedimientos (**wiggle**) y (**muevase**):

```
to go
  ask ovejas [
    wiggle ;; gire al azar en cierta dirección
    muevase  ;; luego muevase
  ]
  tick
end
```

Esto pide a todas las ovejas (**ask**) que realicen dos acciones:

- wiggle
- luego muevase

El orden en el que las ovejas realizan estos dos comandos, uno después del otro, es aleatorio, cada oveja completará ambas acciones antes de que la próxima oveja tome su turno. Después de que todas las ovejas hayan terminado, el comando (**tick**) incrementa el reloj del modelo, lo que indica que ha pasado una unidad de tiempo. Por supuesto, para que el código funcione, debemos definir **wiggle** y **muevase**.

```
;; procedimiento de ovejas, la oveja cambia de dirección
to wiggle    ;; gira derecha luego izquierda, pero en promedio apunta hacia adelante
```

```

right random 90
left random 90
end

;; procedimiento de ovejas, la oveja se mueve una unidad
to muevase
  forward 1
end

```

Estos dos procedimientos los documentaremos como “procedimientos de ovejas”, es decir que están escritos con el supuesto implícito de que el procedimiento que los llame solicitará al conjunto de agentes adecuado (en este caso las ovejas) que los realice.

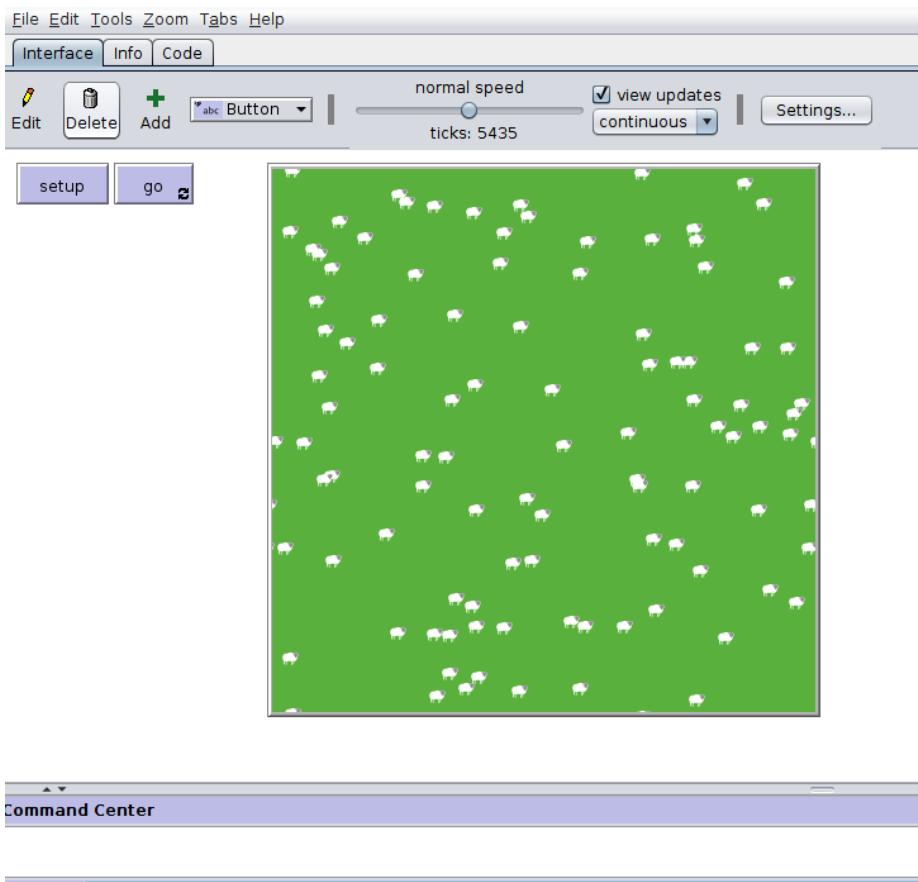
7.3.1.1.1 wiggle

El primer procedimiento simplemente gira a la derecha una cantidad aleatoria entre 0 y 90, y luego de vuelta a la izquierda una cantidad aleatoria entre 0 y 90. La idea detrás de este giro es hacer que las ovejas cambien la dirección en que se dirigen, sin sesgos de izquierda o derecha. Este tipo de giro aleatorio es muy común en los MOBAS, y el nombre adoptado en la comunidad MOBA es este.

7.3.1.1.2 muevase

El segundo procedimiento simplemente mueve la oveja hacia adelante una unidad (el ancho de una parcela)

La distancia a la que se mueve la oveja podría controlarse más tarde por un parámetro, pero por ahora lo mantendremos en una sola unidad constante, puede ejecutar este modelo ahora mismo. Cree un botón llamado go activela opción “continuamente” del botón y oprimalo. Debe observar a las ovejas moviéndose de manera continua, sin cansarse, dentro del paisaje verde.



Ovejas luego de 5435 ticks

7.3.2 Version dos

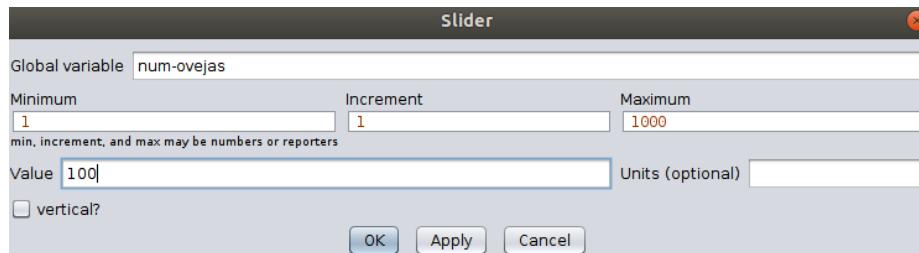
Introducción versión dos

Ahora que tenemos nuestras ovejas moviéndose, tenemos una primera verificación de que nuestro modelo funciona como pretendemos que funcione, es muy importante verificar los MOBAs tan frecuentemente como sea posible.

Parametrizando el Modelo (Deslizadores)

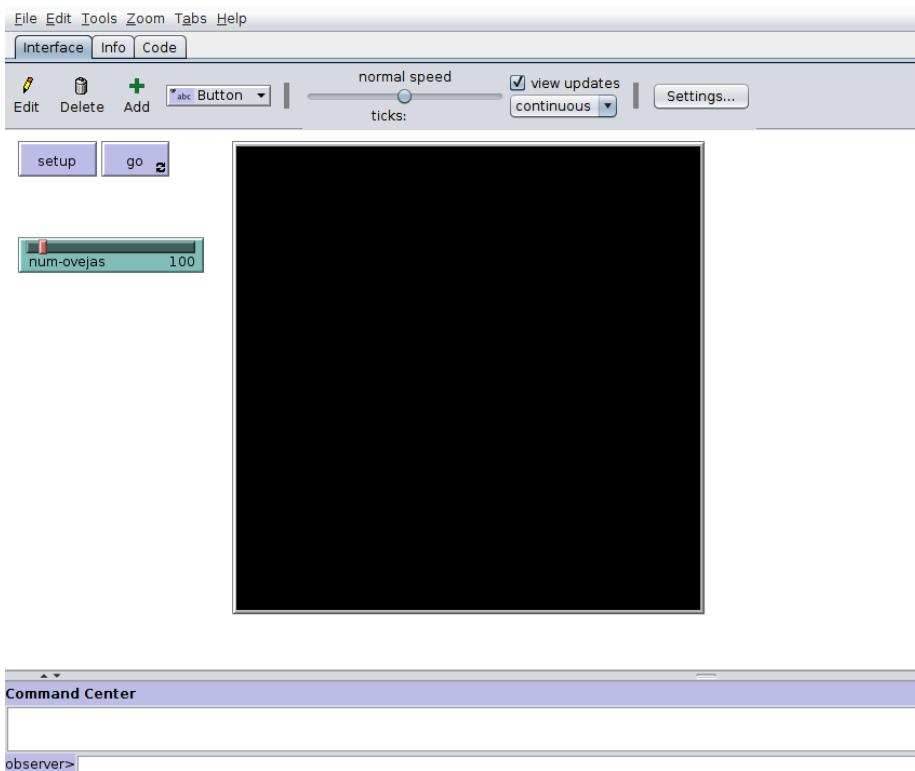
En la versión Uno el número de ovejas es 100. Sin embargo para poder responder nuestra pregunta de investigación deberíamos poder considerar diferentes poblaciones de ovejas, la mejor manera para variar el número de ovejas es creando

un deslizador (que llamaremos **num-ovejas**) Esto nos permitirá cambiar el número de ovejas en el modelo fácilmente desde la interfaz. Al crear el deslizador, necesitamos darle algunas propiedades, un valor mínimo , un valor máximo y un incremento, que es la cantidad que el deslizador cambiará cuando se haga clic en él. En este caso, podemos establecer el mínimo en 1 (ya que menos de una oveja no tiene sentido), el número máximo en 1,000 y el incremento a 1, ya que no tiene sentido tener, por ejemplo, 2.1 ovejas.



Ahora debemos cambiar el procedimiento **setup** para amarrar el deslizador al procedimiento, el cambio es el siguiente:

```
to setup
  clear-all
  ask patches [ ;; colorea el mundo de verde
    set pcolor green
  ]
  ;; crea ovejas y sus propiedades iniciales
  create-ovejas num-ovejas [ ; Nuevo número de ovejas controladas por deslizador
    setxy random-xcor random-ycor
    set color white
    set shape "sheep"
  ]
  reset-ticks
end
```



Extendiendo el Modelo

A continuación, debemos considerar cuál es la extensión más simple que podemos hacerle a nuestro modelo. Mmmmmmm!!! Ok, Tenemos a las ovejas moviéndose, pero !!! el movimiento no les cuesta nada !!!, En el mundo real, el movimiento requiere energía. por lo tanto, el siguiente paso es incluir un “costo” al movimiento. Recordemos que las ovejas tienen tres propiedades:

- dirección, ubicación y energía.

En la primera versión ya las dotamos de dirección y ubicación (de hecho estas propiedades son inherentes a cualquier agente de NetLogo), pero la propiedad de energía de un agente no es una propiedad predefinida de un agente (existen muchos MOBAs donde no es obligatorio definir la energía de un agente como una propiedad de estos). ¿Cómo podemos dotar de energía a cada oveja? muy sencillo, tenemos que definir una nueva propiedad (variable) para la energía, esto lo podemos hacer agregando la siguiente linea al modelo:

```
ovejas-own [ energía ] ;; las ovejas tienen una propiedad llamada energía
```

Con esta linea estamos declarando que las ovejas tienen una nueva propiedad (su energía) pero simplemente declarar que las ovejas tienen energía no es suficiente,

necesitamos inicializar dotar a cada oveja de una energía inicial y también hacer que cambie (de hecho que disminuya) cuando las ovejas se mueven.

El procedimiento setup modificado es el siguiente:

```
to setup
  clear-all
  ask patches [ ;; colorea el mundo de verde
    set pcolor green
  ]
  ;; crea ovejas y sus propiedades iniciales
  create-ovejas num-ovejas [
    setxy random-xcor random-ycor
    set color white
    set shape "sheep"
    set energía 100 ; Nuevo energía inicial para cada oveja
  ]
  reset-ticks
end
```

También necesitamos modificar (añadiendo una linea) el procedimiento muevase para asignarle un costo energético al movimiento:

```
to muevase
  forward 1
  set energía energía - 1 ; Nuevo reducir una unidad de energía en cada movimiento
end
```

El costo es de una unidad de energía, pero es posible que a medida que ampliamos el modelo, este costo se convierta en un parámetro del modelo. Agregar un costo para moverse no significa nada si no hay penalidad por gastar energía.

¿En qué penalidad se podría pensar?" (Mmmmm!!)

....En una drástica, queremos que las ovejas mueran si tienen muy poca energía, por lo tanto debemos verificar si las ovejas han gastado toda su energía, para ello usaremos un procedimiento llamado **verifique-si-hay-muertes**:

```
to go
  ask ovejas [
    wiggle ; gire al azar en cierta dirección
    muevase ;; luego muevase
    verifique-si-hay-muertes ; Nuevo elimina ovejas sin energía
  ]
  tick
end

to verifique-si-hay-muertes ; procedimiento oveja : mínima energía --> al papayo
  if energía < 0 [die]
```

```
end
```

Vayamos a la pestaña interfaz y oprimamos **setup** y luego **go** el modelo se ejecutará por un tiempo y en cierto instante todas las ovejas desaparecerán al mismo tiempo.(¿Por qué?) Lamentablemente, el modelo seguirá funcionando ad-eternum (visualmente esto es cierto usted porque el botón go sigue presionado y los ticks siguen aumentando). Sería muy bueno si nuestro modelo se detiene cuando ya no hay seres vivientes en el paisaje. implementarlo es muy fácil, se agrega una linea al comienzo del **go**: P

```
to go
  if not any? ovejas [stop]  ;; Nuevo si noa hay ovejas pare el modelo
  ask ovejas [
    wiggle ;; gire al azar en cierta dirección
    muevase ;; luego muevase
    verifique-si-hay-muertes
  ]
  tick
end
```

Ahora, si vuelve a ejecutar el modelo, cuando desaparezcan todas las ovejas, el modelo deja de funcionar.

Construyendo las Gráficas

Será muy útil saber cuántas ovejas hay en cada instante de tiempo (tick), por lo que podemos agregar una gráfica que muestre cuál es la población de ovejas en cualquier momento.Hay dos formas de manejar gráficas (plots):

1. El método basado en widgets, ya que hace uso de un widget gráfico.
2. El método programático o basado en código.

En ambos métodos, el código es escrito para actualizar las gráficas, pero en el método basado en código, el código se encuentra en la pestaña Código de NetLogo. En el método basado en widgets, el código se encuentra dentro del widget que hará el trazado de la gráfica.

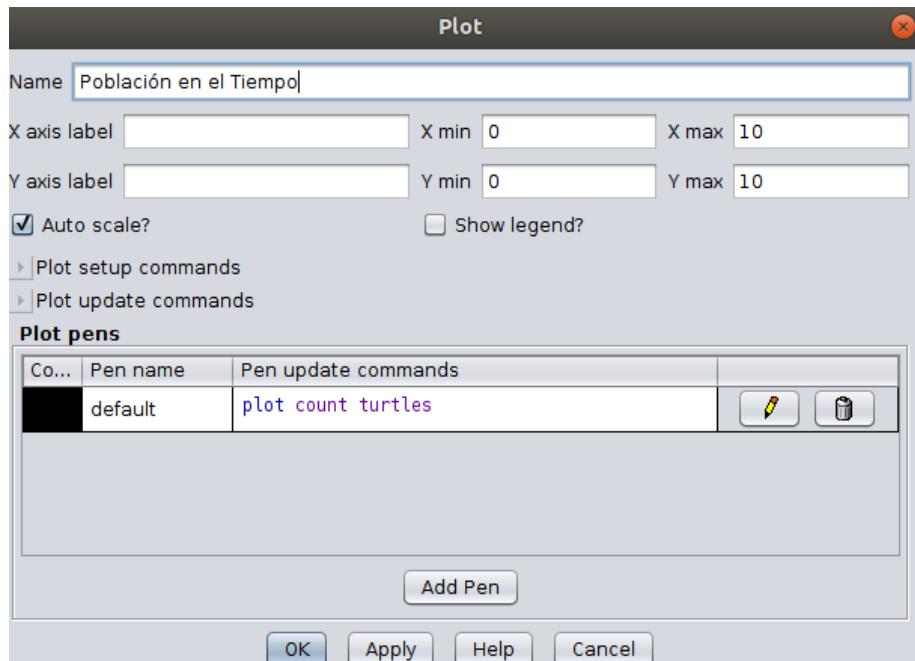
Los dos métodos son equivalentes, por lo que depende del modelador decidir qué método usar. Ambos tienen ventajas y desventajas. La ventaja de usar widgets es que no se necesita “saturar” la pestaña de Código con código adicional para trazar las gráficas, y que para trazados simples, es más rápido de configurar, sin embargo para gráficas complejas puede ser más difícil configurar un diagrama con el método de widgets. Además, si hay errores en la gráfica del widget, puede ser difícil darse cuenta, ya que no se mostrará como un error en la pestaña de Código. El método de trazado de la pestaña Código tiene las ventajas y desventajas inversas.

Nuestra recomendación general es usar widgets para gráficos relativamente simples y la pestaña de código para gráficos complejos.

Usaremos para nuestro modelo, el método basado en la pestaña de Código. Todo lo que haremos aquí también puede ser realizado usando el método de widgets:

- primero se crea un diagrama (Widget) en la interfaz y se establecen sus propiedades

En este caso colocaremos solo el título al Widget, ver la siguiente figura:



- luego agregamos una llamada al diagrama desde el **go**

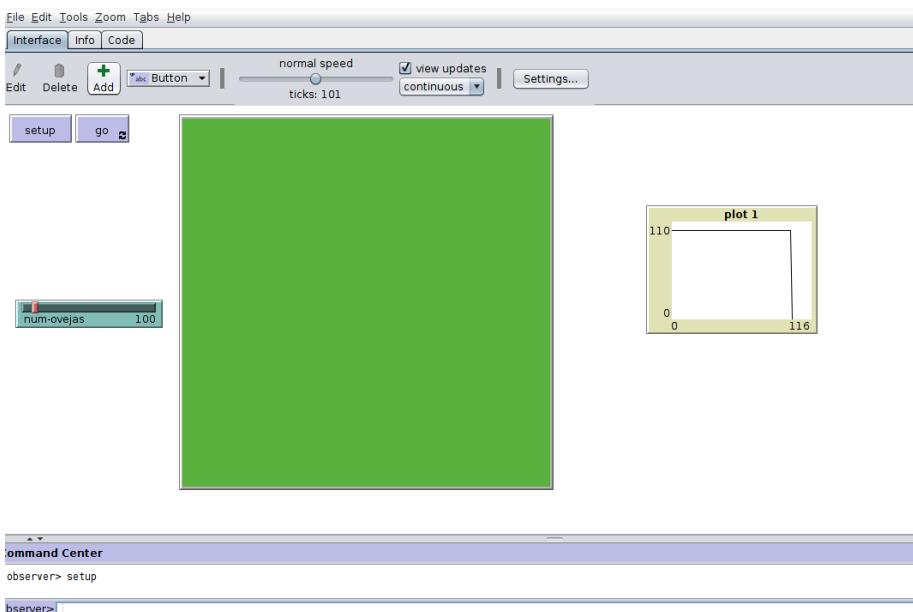
```
to go
  if not any? ovejas [stop]
  ask ovejas [
    wiggle ; gire al azar en cierta dirección
    muevase ; luego muevase
    verifique-si-hay-muertes
  ]
  tick
  actualice-gráficas ; Nuevo llamado a actualizar gráficas
end
```

Necesitamos definir el procedimiento **actualice-gráficas**:

```
;; actualiza gráficas en la interfaz
to actualice-gráficas
  plot count ovejas ; Gráfica el número de ovejas dado el tiempo (tick)
end
```

(Nota Importante: Podríamos haber usado “plot count ovejas” directamente al final del procedimiento go, pero sabemos que es probable que tengamos que trazar otras gráficas (por ejemplo más adelante de número de lobos y de cantidad de pasto), Entonces podemos usar este procedimiento para otras gráficas también)

Ejecutemos el modelo (setup luego go):



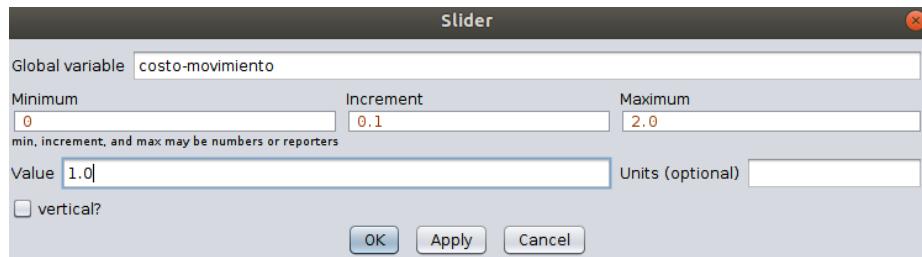
la gráfica nos muestra que hubo un población completa de ovejas hasta el final y luego todas murieron al tiempo, la muerte de todas las ovejas en el tick 101 es el resultado de la energía inicial y el costo del movimiento que le hemos definido a las ovejas.

Costo del Movimiento como parámetro

Recordemos que queríamos que el costo del movimiento fuera un parámetro del modelo. Necesitamos entonces:

- Agregar otro deslizador para controlar el costo del movimiento
- Modificar el procedimiento muevase para tomar en cuenta el valor del deslizador.

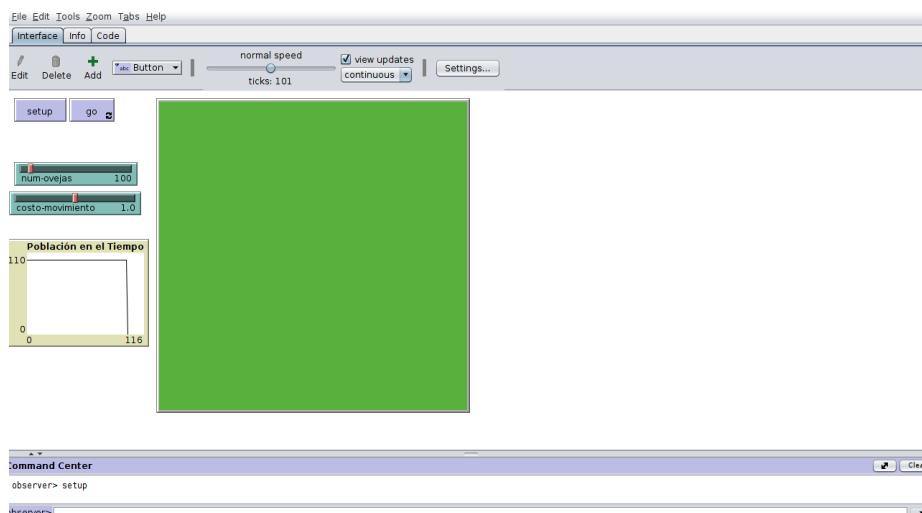
Entonces: 1. Cree un deslizador llamado costo-movimiento de la siguiente manera:



Modifique el procedimiento muevase:

```
to muevase
  forward 1
  set energía energía - costo-movimiento ; Nuevo reducir energía por deslizador
end
```

¡¡¡ ufff !!!! Listo tenemos la versión 2 de nuestro modelo!!!



Modelo Versión Dos, con dos deslizadores, una Gráfica y Ovejas muertas

7.3.3 Version tres

Introducción versión tres

En este momento, el modelo exhibe un comportamiento muy predecible. Cada vez el modelo funciona durante ($100 / \text{costo-movimiento}$) ticks, luego todas las ovejas desaparecen y el modelo se detiene. La razón es porque las ovejas actualmente gastan energía (moviéndose) pero no tienen forma de adquirir

energía. Por lo tanto, necesitamos dar a las ovejas la capacidad de comer pasto y de esta manera ganar energía. Sin embargo, primero debemos crear el pasto!!!

¿Cómo hacer esto?

Vamos a definirle a las parcelas (que funcionaran como manojo de pasto para este modelo), una propiedad llamada **cantidad-de-pasto**, que mide la cantidad de pasto que hay disponible en esa parcela, debemos entonces agregar a nuestro modelo la siguiente línea (después de la línea `ovejas-own[energía]`):

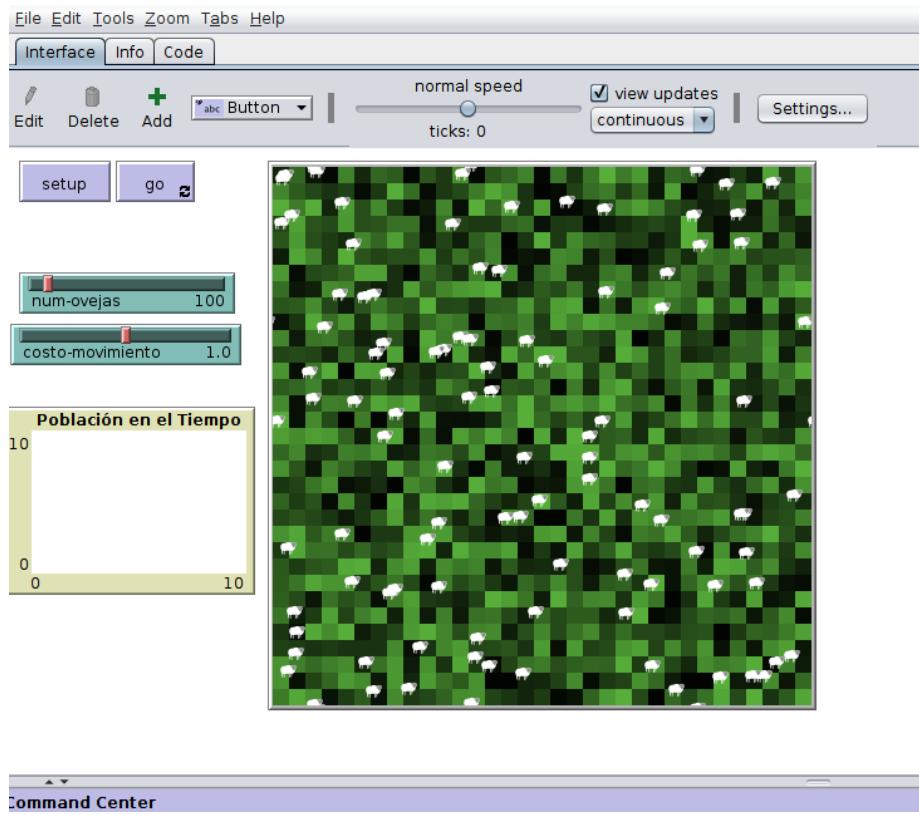
```
patches-own[cantidad-de-pasto] ;; las parcelas contienen pasto, cantidad variable
```

Ahora tenemos que colocar el pasto, pero mientras estamos en ello, modificaremos el color de las parcelas para que indiquen cuánto pasto tienen. Hacemos esto configurando la cantidad inicial de pasto a un número aleatorio real entre 0.0 y 10.0. Usaremos números reales para el pasto, ya que a diferencia de las ovejas, que son individuos, cada parcela contiene un “manojo” de pasto, no hojas individuales. Esto asegura cierta variabilidad en la cantidad de pasto y crea cierta heterogeneidad. Luego establecemos el color del pasto a un tono verde tal que si no hay pasto en absoluto, la parcela será negra, y si hay mucho pasto en la parcela, está será de color verde brillante.

```
to setup
  clear-all

  ask patches [
    ;; parcelas se habitan con un número aleatorio de pasto
    set cantidad-de-pasto random-float 10.0 ;; colorear las parcelas con la cantidad de pasto
    set pcolor scale-color green cantidad-de-pasto 0 20
  ]

  create-ovejas num-ovejas [ ;; crea ovejas y sus propiedades iniciales
    setxy random-xcor random-ycor
    set color white
    set shape "sheep"
    set energía 100
  ]
  reset-ticks
end
```



Command Center

Ahora necesitamos modificar el procedimiento **go** para que las ovejas puedan comer pasto. Como mencionamos en el diseño, colocamos este procedimiento después del procedimiento de verificación de muerte, llamaremos al procedimiento **comer**:

```

to go
  if not any? ovejas [stop]
  ask ovejas [
    wiggle ;; gire al azar en cierta dirección
    muevase ;; luego muevase
    verifique-si-hay-muertes
    comer ; Nuevo las ovejas comen pasto
  ]
  tick
  actualice-gráficas
end

to comer
  if cantidad-de-pasto >= 1 [
    set energía energía + 1 ; incremente energía de la oveja
  ]

```

```

set cantidad-de-pasto cantidad-de-pasto - 1 ; reduzca el pasto de la parcela donde está
set pcolor scale-color green cantidad-de-pasto 0 20 ; actualiza el color del pasto
]
end

```

Se verifica, para cada oveja, si hay suficiente pasto en la parcela donde está. Si hay suficiente para comer, la oveja convierte el pasto en energía , y se disminuye la cantidad de pasto en la parcela.Al mismo tiempo, recoloreamos la parcela para reflejar la nueva cantidad de pasto, el comportamiento del modelo todavía no es muy interesante. Las ovejas deambulan, comen tanto pasto como pueden, y eventualmente todos se extinguen. La única variación en el modelo es el nivel de pasto en las parcelas. Debido a la distribución aleatoria de pasto originalmente y debido a que las ovejas se muevan al azar alrededor del paisaje, habrá algunas áreas de pasto que son completamente consumidas por las ovejas y otras áreas que serán solo parcialmente consumidas.

Para que el modelo sea un poco más interesante, agregaremos un procedimiento para que el pasto vuelva a crecer. Al permitir que el pasto vuelva a crecer, **debería ser posible mantener la población de ovejas a lo largo del tiempo**, ya que existirá una fuente renovable de energía para ellos. Empezamos modificando el procedimiento **go**, añadiendo el procedimiento **renace-pasto**:

```

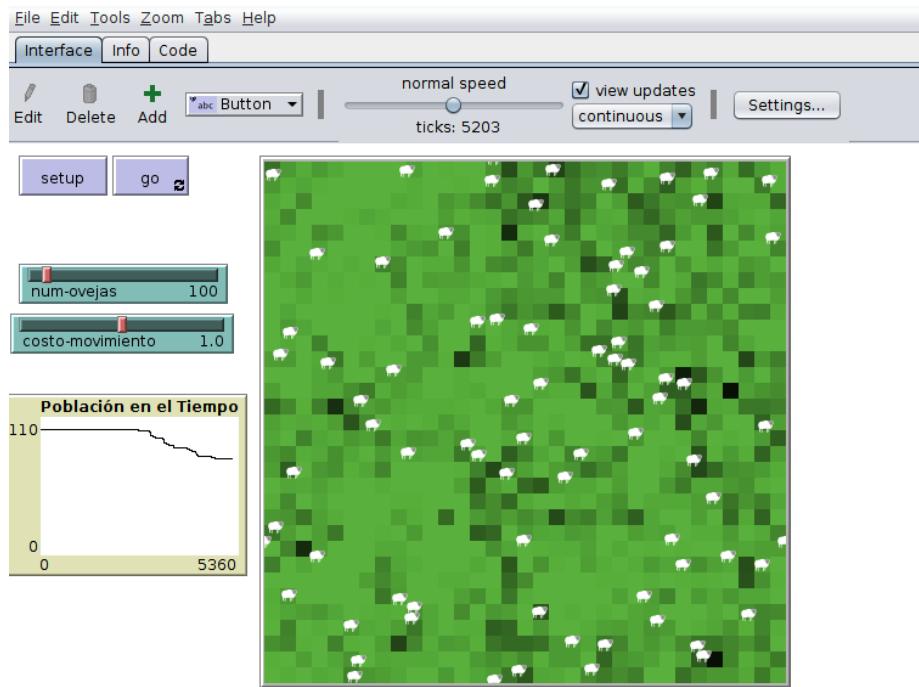
to go
  if not any? ovejas [stop]
  ask ovejas [
    wiggle ;; gire al azar en cierta dirección
    muevase ;; luego muevase
    verifique-si-hay-muertes
    comer
  ]
  renace-pasto ; nuevo el pasto vuelve a crecer
  tick
  actualice-gráficas
end

;; renace el pasto
to renace-pasto
  ask patches [
    set cantidad-de-pasto cantidad-de-pasto + 0.1
    if cantidad-de-pasto > 10 [
      set cantidad-de-pasto 10
    ]
    set pcolor scale-color green cantidad-de-pasto 0 20
  ]
end

```

Este procedimiento dice a todos las parcelas que aumenten la cantidad de pasto que tienen en una décima parte. También nos aseguramos de que el pasto nunca

excede de 10 unidades, que sería la cantidad máxima para cualquier parcela. Luego se recolorea el pasto, con este pequeño cambio las oveja deberían sobrevivir a una corrida del modelo. ¡¡¡pruébelo!!!!



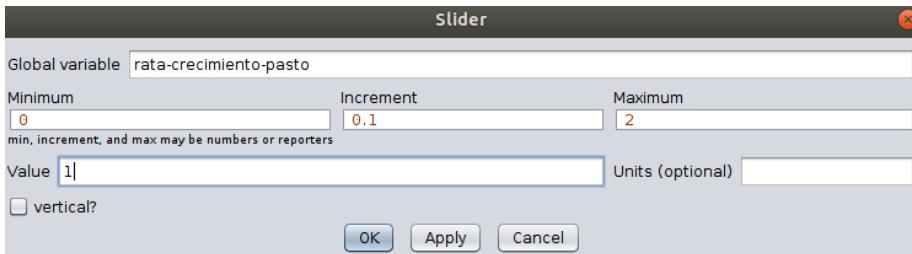
Ahora tenemos recoloración de pasto en tres lugares diferentes del modelo, por lo que también sería bueno colocar un procedimiento. A menudo, cuando comenzamos a duplicar código, vale la pena colocarlo en un procedimiento separado; de esa manera tenemos que modificar el código en una sola ubicación si necesitamos cambiarlo más tarde (por ejemplo, si queremos que el césped sea de color amarillo en lugar de verde). Mantener el código más conciso y colocar nombres apropiados ayuda a que su código sea más legible para otros. Entonces definimos un procedimiento **recolorear-pasto**:

```
to recolorear-pasto
  set pcolor scale-color green cantidad-de-pasto 0 20
end
```

(Nota: Coloque este procedimiento en el modelo y coloque en los sitios donde se recolorea el pasto el nombre de este procedimiento)

Reflexion en torno al modelo

Al ejecutar el modelo varias veces con cien ovejas iniciales, queda claro que cien ovejas no pueden consumir todo el pasto y, por lo tanto, eventualmente todo el mundo se convierte en un sólido tono verde. Sin embargo, si aumenta el número de ovejas iniciales a un número mayor, digamos setecientos, y luego se corre el modelo, las ovejas consumirán casi todo el pasto en el modelo, y luego muchas de ellos morirán. Sin embargo, algunas de ellas, que tenían una gran cantidad de energía antes de que desapareciera todo el pasto sobrevivirán y eventualmente el pasto volverá a crecer, lo que les permitirá persistir, ya que ya hay muchas ovejas compitiendo por el pasto. Otro parámetro que queremos introducir y que puede afectar la dinámica del modelo es la velocidad a la que el pasto vuelve a crecer. Vamos a agregar un deslizador llamado **rata-crecimiento-pasto**, le damos valores límite de 0 y 2 y un incremento de 0.1:

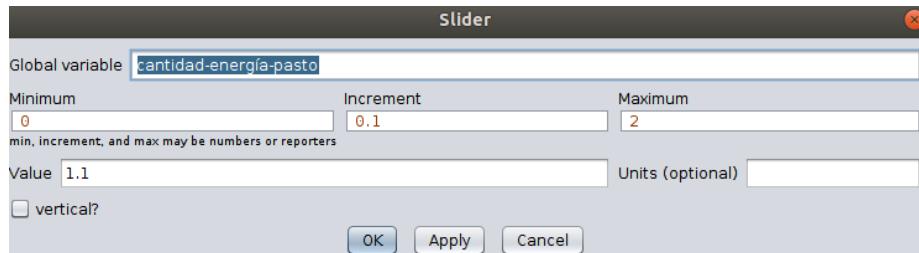


modifiquemos el procedimiento **renace-pasto** para reflejar el uso de este nuevo parámetro:

```
;; renace el pasto
to renace-pasto
  ask patches [
    set cantidad-de-pasto cantidad-de-pasto + rata-crecimiento-pasto ; Nuevo : deslizador
    if cantidad-de-pasto > 10 [
      set cantidad-de-pasto 10
    ]
    recolorear-pasto
  ]
end
```

Si colocamos **rata-crecimiento-pasto** en un valor lo suficientemente alto (por ejemplo 2.0), entonces incluso con setecientas ovejas en el modelo, se puede mantener la población total de ovejas. Esto se debe a que las ovejas pueden obtener una unidad completa de energía del pasto, y si este vuelve a crecer esta cantidad en un solo tick, las ovejas gastan esa energía cuando se mueven, pero esa energía se reemplaza inmediatamente. Sin embargo, si se cambia el deslizador **costo-movimiento** a un valor mayor que uno, las ovejas eventualmente morirán, esto se debe a que están gastando energía más rápido de lo que pueden recolectar del medio ambiente(incluso si no hay escasez de pasto).

Para que nuestro modelo sea más flexible, podemos agregar otro parámetro (deslizador), llamado **cantidad-energía-pasto**, que controla la cantidad de energía que las ovejas pueden obtener al comer pasto, al igual que con los deslizadores anteriores, tendremos que establecer límites razonables y un incremento para este nuevo deslizador.



Para usar este nuevo deslizador necesitamos modificar el procedimiento comer:

```
to comer ; Nuevo : Deslizador cantidad-energía-pasto
  if cantidad-de-pasto >= cantidad-energía-pasto [
    set energía energía + cantidad-energía-pasto ; incremente energía oveja
    set cantidad-de-pasto cantidad-de-pasto - cantidad-energía-pasto ; decrementa el pasto
    recolorear-pasto ; actualiza color del pasto
  ]
end
```

Tenga en cuenta que utilizamos el parámetro **cantidad-energía-pasto** tanto para incrementar la energía de las ovejas como para disminuir el valor del pasto. Podríamos haber usado dos diferentes parámetros para estas dos funciones, pero podemos pensar en el sistema "ovejas / pasto" como un sistema de conversión de energía, donde la energía del pasto fluye hacia las ovejas.

En este momento podemos observar una dinámica interesante:

Comenzando con setecientas ovejas, duran alrededor de trescientos ticks. Pero luego hay una hambruna masiva, que se hace cada vez más gradual, hasta que alrededor de quinientos ticks, la población se mantiene estable con un poco más de cuatrocientas ovejas. Luego de que han muerto suficientes ovejas, el pasto se regenera, mantiene a las ovejas vivas y el sistema alcanza un estado de **equilibrio**. Como el movimiento de las ovejas es aleatorio, es posible que una gran cantidad de ovejas se agrupen en las mismas pocas parcelas durante mucho tiempo, y por lo tanto mueran de hambre, pero esto no es probable.

Dependiendo de la selección de los parámetros del modelo, también son posibles muchos otros comportamientos.

¡¡ Siéntase libre de experimentar y explorar antes de pasar a la próxima versión del modelo!!

7.3.4 Version cuatro

Introducción versión cuatro

El modelo tiene ovejas moviéndose en un paisaje, consumiendo recursos y muriendo. Sin embargo, no hay forma de que la población de ovejas **aumente!!!**, de hecho solo puede bajar. Por lo tanto, para que vuelva a subir, agregaremos reproducción al modelo. Construir un modelo reproductivo completo con parejas sexuales y tener una oveja embarazada tomaría mucho tiempo, y no está claro de que serviría para responder nuestra pregunta inicial.

En cambio, haremos dos simplificaciones:

1. Una sola oveja puede producir nuevas ovejas!!!. Se puede ver esto como reproducción asexual o se puede pensar que cada oveja representa a un par de ovejas macho y hembra unidas. Esta suposición puede parecer extraña al principio y ciertamente es obviamente contraria a la realidad. Este es un buen momento para recordar las palabras de George Box: “todos los modelos están equivocados, pero algunos son útiles”. Está bien errar en nuestro modelo sobre un hecho tan básico **si el modelo simplificado sigue siendo útil para nuestros propósitos**. Pero si luego vemos que con esta simplificación se ha perdido alguna utilidad del modelo, siempre podremos agregar verdadera reproducción sexual más tarde.
2. La segunda simplificación es esta: en lugar de preocuparse por el proceso de gestación, asumiremos que las ovejas dan a luz de manera inmediata a una ovejita, **cuando alcanzan un cierto nivel de energía**. Esta energía puede verse como una aproximación a tener la capacidad de reunir suficientes recursos para sobrevivir durante todo el período de gestación. Para implementar esto, comenzamos agregando código al procedimiento go:

```

to go
  if not any? ovejas [stop]
  ask ovejas [
    wiggle ;; gire al azar en cierta dirección
    muevase ;; luego muevase
    verifique-si-hay-muertes
    comer
    reproducirse ; Nuevo: procedimiento de reproducción de las ovejas
  ]
  renace-pasto ; nuevo el pasto vuelve a crecer
  tick
  actualice-gráficas
end

to reproducirse
  if energía > 200 [

```

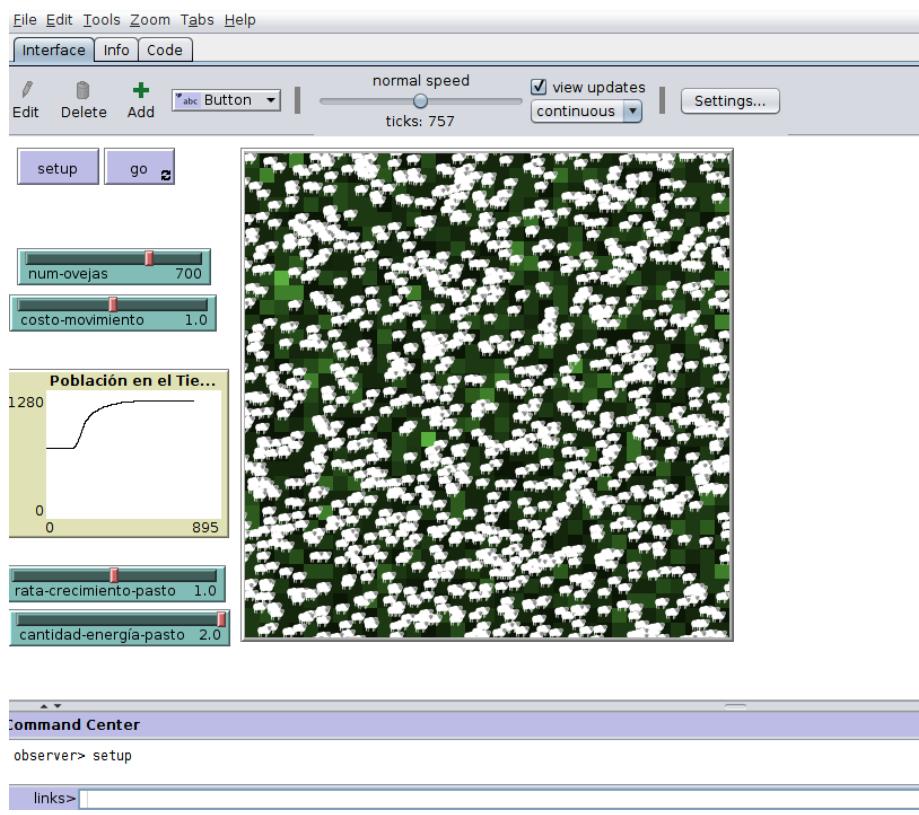
```

set energía energía - 100 ;; reproducción transfiere energía
hatch 1 [ set energía 100 ] ;; energía de la nueva oveja
]
end

```

Se verifica si la oveja actual tiene suficiente energía para reproducirse (dos veces la cantidad original de energía (100)). Si la oveja lo cumple, entonces disminuye su energía en 100, y crea una nueva ovejita (**hatch** crea un clon del agente en la misma parcela) y establece su energía también a 100.

Ahora, si ejecutamos el modelo con un bajo **costo-movimiento** en comparación con la **cantidad-energía-pasto**, y lo arrancamos con 700 ovejas, la población aumenta con el tiempo, y finalmente se nivela cerca de 1.300 ovejas, como se muestra en la figura :



Modelo con Reproducción (Aumento de 700 a 1300 ovejas)

7.3.5 Version cinco

Introducción versión cinco

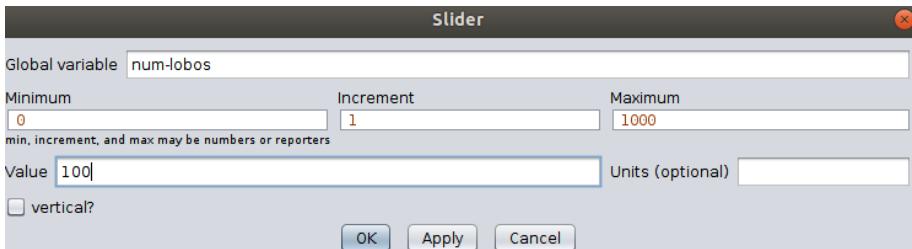
Tenemos a las ovejas comportándose de la manera que describimos en nuestro diseño pero nuestro objetivo original era tener dos especies. Entonces ahora necesitamos agregar a los lobos. Lo primero que debemos hacer es crear una segunda raza (breed) llamada lobos. Al mismo tiempo necesitamos dar a los lobos energía.

Podríamos hacer esto agregando un comando lobos-own como nuestro ovejas-own, pero dado que los únicos agentes de tortuga en el modelo son ovejas y lobos, podemos hacer de energía una propiedad **genérica** de todas las tortugas. Hacemos esto cambiando la declaración: **ovejas-own[energía]** por **turtles-own[energía]**, entonces añadimos la nueva raza lobos y hagamos esta modificación :

```
breed[ovejas oveja]
breed[lobos lobo]

turtles-own[energía]
```

Después de esto, necesitamos crear los lobos, tal como lo hicimos con las ovejas. Primero, agregamos un deslizador llamado **num-lobos**:



Ahora modificamos el procedimiento SETUP:

```
to setup
  clear-all

  ask patches [
    set cantidad-de-pasto random-float 10.0 ;; parcelas se habitan con num aleatorio de pasto
    recolorear-pasto ;; colorear las parcelas de acuerdo con la cantidad de pasto
  ]

  create-ovejas num-ovejas [;; crea ovejas y sus propiedades iniciales
    setxy random-xcor random-ycor
    set color white
    set shape "sheep"
```

```

set energía 100
]

;; NUEVO : CREACIÓN DE LOBOS
create-lobos num-lobos [ ;; crea lobos y sus propiedades básicas
  setxy random-xcor random-ycor
  set color brown
  set shape "wolf"
  set size 1.5
  set energía 100
]

reset-ticks
end

```

Ahora que hemos agregado lobos al modelo, también necesitamos agregar sus comportamientos. Observe que todos los comportamientos son comunes tanto para los lobos como para las ovejas, incluso si los detalles exactos difieren (por ejemplo, los lobos comen ovejas, mientras que las ovejas comen pasto, pero ambos “comen”). Entonces simplemente debemos reemplazar “ovejas” por “turtles” en las dos primeras líneas de nuestro procedimiento **go**:

```

to go
  if not any? turtles [stop]    ;; NUEVO: CAMBIAR ovejas POR turtles
  ask turtles [    ; cambiar ovejas por turtles
    wiggle ; gire al azar en cierta dirección
    muevase ; luego muevase
    verifique-si-hay-muertes
    comer
    reproducirse
  ]
  renace-pasto  ; nuevo el pasto vuelve a crecer
  tick
  actualice-gráficas
end

```

Todos los comportamientos que le dimos a las ovejas se aplican igualmente bien a los lobos, entonces el modelo correrá como está. Sin embargo:

¡¡¡los lobos comen pasto en este modelo tal y como está!!!!!!

Pero el comportamiento alimenticio de los lobos es diferente del comportamiento alimenticio de las ovejas, por lo que tendremos que modificar nuestro procedimiento de “comer”:

```

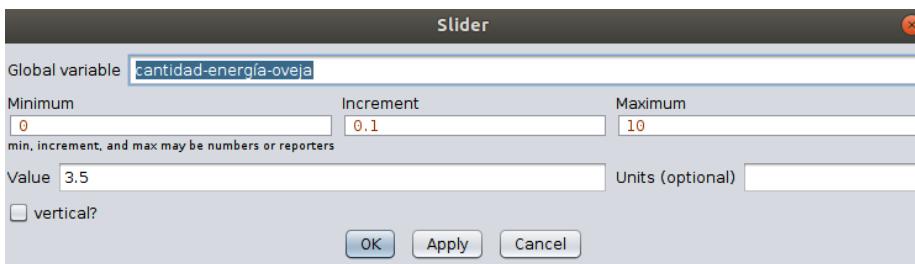
to comer
  ifelse breed = ovejas [
    comer-pasto
  ]

```

```
[  
  comer-oveja  
]  
end
```

Ahora nuestro comportamiento alimentario será diferente para las ovejas y los lobos. Las ovejas comerán pasto y los lobos comerán ovejas

- Cambiemos el nombre de nuestro antiguo procedimiento “comer” por “comer-pasto” ya que ese es el comportamiento que definimos para las ovejas. Ahora falta definir el comportamiento de “comer-ovejas”.
- Agregue un deslizador **cantidad-energía-oveja**



coloquemos el procedimiento **comer-oveja**:

```
to comer-oveja
  if any? ovejas-here [ ;; si hay ovejas coma
    let target one-of ovejas-here ;; seleccione una oveja al azar de la parcela
    ask target [ ;; coma la oveja seleccionada
    die
  ]
  ;; incremente energía de acuerdo con deslizador
  set energía energía + cantidad-energía-oveja
]
end
```

En este procedimiento, el lobo primero verifica si hay ovejas disponibles para comer en la parcela donde se encuentra. Si las hay, entonces consume a una de ellas (elige una al azar de la parcela) y obtiene un aumento de energía de acuerdo con el deslizador de energía que acabamos de definir.

Actualizando las graficas

Ahora nuestro modelo tiene todos los agentes, comportamientos e interacciones que nos propusimos crear. Sin embargo, nuestro gráfica (plot) aún no contiene toda la información. Sería útil si también el gráfico muestra la población de lobos, y al mismo tiempo podemos agregar una visualización de cantidad de pasto en el mundo. Para hacer esto, primero agregamos dos “esferas”(pen) adicionales a

la gráfica (lobos y pasto). También cambiamos el nombre del lápiz de trazado predeterminado a oveja.

Para que quede claro, la modificación al procedimiento actualice-gráficas es la siguiente:

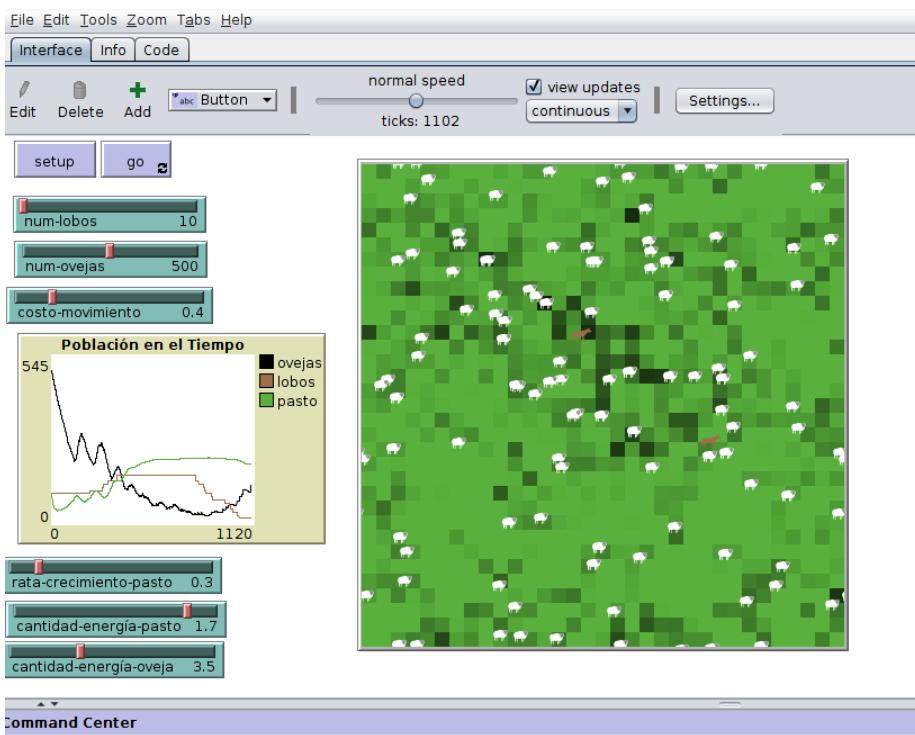
```

to actualice-gráficas
  set-current-plot-pen "ovejas"
  plot count sheep
  set-current-plot-pen "lobos"
  plot count wolves * 10 ;; se escala para que se vea bien la gráfica
  set-current-plot-pen "pasto"
  plot sum [cantidad-de-pasto] of patches / 50
  ;; se escala para que se vea bien la gráfica
end

```

Este código es bastante sencillo. El “* 10” y “/ 50” son solo factores de escala para que la gráfica sea legible cuando todos los datos se trazan en el mismo eje. (Pero tenga en cuenta que al leer el número de lobos fuera de la gráfica, el recuento real de la población es diez veces más pequeño.) A menudo también es útil agregar **monitores** para estas variables para poder leer por fuera de la gráfica los valores exactos.

Ahora podemos experimentar el modelo con una variedad de configuraciones de parámetros. Muchos ajustes de parámetros provocarán la extinción de una o ambos especies. Pero podemos encontrar parámetros que den como resultado un ecosistema autosostenible (en donde los niveles de población de las especies varían de manera cíclica) Se muestra uno de estos conjuntos de parámetros en la siguiente figura:



Con esos parámetros, las poblaciones de lobos y ovejas se mantienen en el tiempo de manera cíclica.

Chapter 8

Analizando Modelos Basados en Agentes

8.1 Modelos e investigación

Cuando se habla de usar modelos en investigación, típicamente queremos usar un modelo para obtener entendimiento de una pregunta específica de investigación. Construir una buena pregunta de investigación es uno de los aspectos más difíciles en el ciclo de modelaje. Una manera común de hacerlo es comenzar con algo de teoría y definir que pregunta puede ser contestada con el modelo basado en agentes, una pregunta como:

- ¿Por qué no más personas usan los carros eléctricos?

es difícil de contestar con un modelo basado en agentes ya que es una pregunta empírica para lo cual lo mejor es realizar sondeos y encuestas con compradores potenciales de carros. Sin embargo podríamos formular una pregunta como:

- ¿ Basados en nuestro entendimiento del comportamiento de los consumidores y en las tendencias de ventas de carros, cuáles pueden ser las proyecciones de ventas de carros eléctricos hacia el futuro?

Esta pregunta puede ser abordada por una variedad de modelos incluyendo modelos estadísticos. Una pregunta más enfocada a la metodología de los MOBAs podría enfocarse en el rol de la influencia social y la estructura de las redes sociales, asumiendo que se tienen datos relevantes de este aspecto. ¿Cuáles son las estrategias que podrían ayudar a la formulación de una pregunta de investigación apropiada?

Una es leer publicaciones de implementación de modelos basados en agentes en el área de interés y ver que tipo de preguntas se tratan de resolver. Buenas

preguntas de investigación que usan MOBAs se concentran en como diferentes mecanismos que afectan el comportamiento de agentes y su entorno impactan los resultados que se quieren investigar. Esto no sorprende ya que los modelos basados en agentes funcionan de esa manera. Una vez se define la pregunta de investigación, se pueden definir hipótesis, que ayudan a formular que resultados hay que investigar y que mecanismos incluir en el modelo, por ejemplo:

*; Es la discriminación racial la única causa para la segregación?

Para probar esto podríamos derivar la segregación en un agente donde los agentes son “felices” si la mayoría de sus vecinos lo son. El modelo de Schelling demuestra que la segregación solo sucede si los agentes se sienten bien viviendo en vecindarios donde la mayoría es diferente a ellos. Una vez tenemos una idea clara de las preguntas de investigación, podemos comenzar a trabajar en el modelo, también es útil mirar publicaciones de trabajos sobre modelos parecidos. De hecho, es conveniente replicar uno o varios de estos modelos para tener un mejor idea de como funcionan estos tipos de modelos y que retos y desafíos han tenido sus autores. No es inusual que los que construyen modelos sean sobre-optimistas en sus ambiciones, entonces observar estos modelos alternativos ayuda a ser más realista sobre el alcance de este tipo de modelos. Un aspecto importante al construir un modelo MOBA, como ya se ha mencionado, es construirlo de la manera más simple posible, pero no tan simple. Esto es más fácil decirlo que hacerlo. De hecho, modelos simples son difíciles y dispendiosos de construir mientras que modelos complicados son más fáciles de crear. Modelos simples, que capturan la esencia de un tema de interés son el resultado de mucha iteraciones y ensayos con el modelo, el modelaje es un arte, como crear una escultura o una pintura, en donde el producto final termina siendo unos pocos brochazos que muestran la esencia de la obra de arte.

8.2 Herramientas básicas

Use diagramas de flujo para establecer conexiones entre los diferentes componentes de un modelo:

- ¿Cuáles son los procesos de retroalimentación?
- ¿Cuáles son las escalas espaciales y temporales?
- ¿Qué tipo de agentes incluir y cuales son sus interacciones?

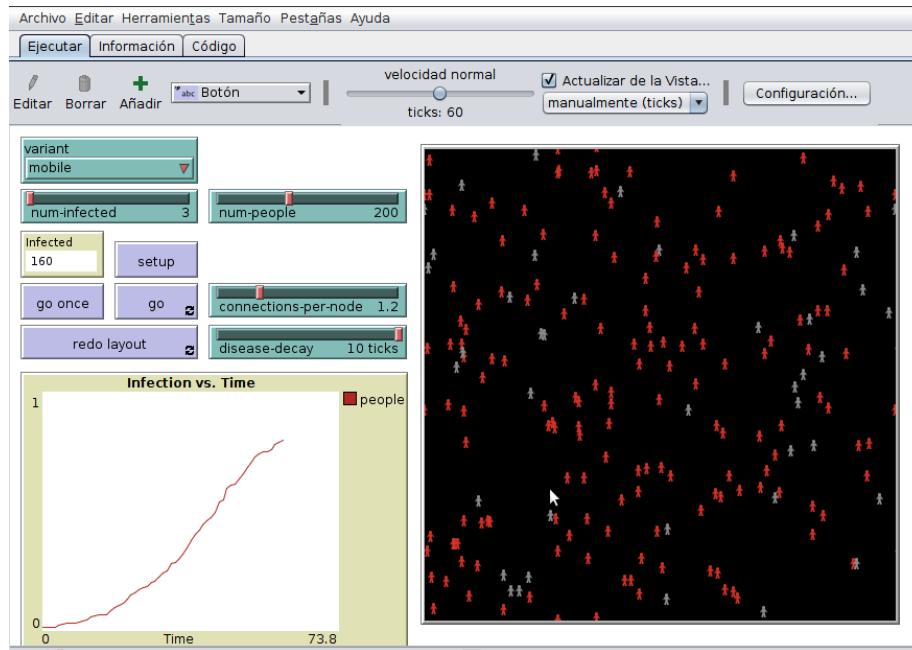
Sabiendo que el modelaje es un proceso, no demore todo su tiempo diseñando el modelo. El replicar otros modelos relacionados ayuda a no comenzar de ceros. Implemente sus primeras ideas y encontrará que sus ideas son **menos** brillantes de lo que pensaba. El proceso de modelación y el observar los resultados de las suposiciones del modelo, es un proceso de aprendizaje que le ayudará a redefinir la estructura del modelo. Una vez se tiene una versión inicial del modelo, haga un análisis adecuado. Se dará cuenta que se utilizará mucho más tiempo en el análisis que en la construcción del modelo, explore condiciones extremas del

modelo y pruebe si el modelado todavía produce resultados relevantes. Cuando empieza a tener confianza en el funcionamiento del modelo puede comenzar a hacer un análisis más formal. Un análisis básico puede ser un análisis de sensibilidad donde se varían los parámetros del modelo de manera sistemática. Puede comenzar a variar los parámetros uno a uno para identificar qué parámetros impactan de mayor manera los resultados, luego varíe dos parámetros y observe su covariación, el análisis de sensibilidad le ayudará a entender mejor el modelo y puede producir que se generen modificaciones al modelo e inclusive a reconsiderar la pregunta inicial que dio origen al modelo. Con datos cualitativos se puede construir un modelo cuantitativo, como las feromonas impactan el movimiento de las hormigas, como se forman caminos de hormigas, como los patrones de vuelos de aves se forman, etc... si se tiene información cuantitativa, esta se puede usar para definir valores de los parámetros del modelo y también se puede usar para evaluar el modelo, acá entramos en el importante tema de la **calibración** de un modelo que veremos más adelante. Cuando se hace un análisis sistemático de un modelo hay que ser crítico acerca del modelo, desconfie de los resultados del modelo y pregúntese siempre por qué se dan estos resultados, sobre todos si estos no son obvios ni intuitivos, es frecuente sorprenderse con algunos resultados del modelo, pero al final se debería entender porque los resultados a la larga resultan no ser tan sorprendentes. Durante el proceso de implementación del modelo, es una buena práctica comenzar a documentarlo. La buena documentación de un modelo basado en agentes no es fácil de obtener ya que estos modelos van más allá de un conjunto de ecuaciones. Un protocolo que puede ayudar a obtener una buena y sistemática documentación es el protocolo ODD (ODD protocol). Cuando se tenga el modelo documentado, puede guardar su modelo y su documentación. Un repositorio recomendado es la Librería de Modelos Computacionales COMSES NET. Guardar su modelo en la nube ayuda a otros a construir sobre su trabajo y se asegura que hacia el futuro la documentación no se pierda. Finalmente, ¿Qué se aprende del proceso de modelado? Se pensará que ya no se necesita modelar más cuando ya se tiene un mejor entendimiento del problema, pero siempre es posible mejorar un modelo y aprender de él, cuando vaya a comunicar los resultados de su trabajo y escriba, por ejemplo, un artículo de investigación, haga énfasis en las preguntas de investigación y enfoque la discusión en el análisis del modelo y en cómo este responde a las preguntas. No tiene que incluir todos los detalles en el artículo, mueva los detalles técnicos a un apéndice.

8.3 Un Ejemplo de Análisis

Si alguien se resfria y está tosiendo, podría infectar a otros. Los que entran en contacto con él, sus amigos, compañeros de trabajo e incluso extraños, pueden atrapar la gripe. Si un virus del resfriado infecta a alguien, esa persona podría transmitir esa enfermedad a otras cinco personas (seis ahora infectadas). A su vez, esas otras cinco personas podrían extenderse el resfriado a cinco personas más.

cada uno (treinta y uno ahora están infectados), y esos veinticinco las personas pueden propagar el resfriado a cinco personas adicionales (ciento cincuenta y seis personas son ahora infectados). De hecho, la tasa de infección inicialmente aumenta exponencialmente. Sin embargo, dado que este recuento de infecciones crece tan rápido, cualquier población eventualmente alcanzará el límite de la cantidad de personas que pueden infectarse. Por ejemplo, imagine que las 156 personas mencionadas anteriormente trabajan para la misma compañía de 200 personas. Este modelo simple supone que cada persona infecta el mismo número de personas, lo que evidentemente no es el caso en contextos reales. A medida que una persona se mueve en su espacio de trabajo, podría ser el caso de que, por casualidad, no vean a muchas personas en un día, mientras que otra persona puede ver a muchas personas. Además, nuestra descripción inicial supone que si una persona infecta a cinco personas y otra persona infecta a cinco, no habrá superposición. En realidad, es probable que haya una superposición sustancial. Por lo tanto, la propagación de la enfermedad en un lugar de trabajo no es tan sencillo como sugiere nuestra descripción inicial. Supongamos que estamos interesados en comprender la propagación de la enfermedad, y queremos construir un MOBA de la propagación. ¿Cómo deberíamos hacerlo? Primero, necesitamos algunos agentes que hagan un seguimiento de si están infectados con un resfriado o no. Además, estos agentes necesitan una ubicación en el espacio y la capacidad de moverse. Finalmente, necesitamos la capacidad de inicializar el modelo infectando a un grupo de individuos. Ese es exactamente cómo se comporta el modelo de NetLogo que discutiremos en otro capítulo del libro (capítulo ??) :



Las personas se mueven aleatoriamente en un paisaje e infectan a otras personas cada vez que entran en contacto con ellos. Aunque este modelo es simple, exhibe un comportamiento interesante y complejo. Por ejemplo:

- ¿Qué sucede si aumentamos el número de personas en el modelo? ¿Se propaga la enfermedad más rápido en toda la población, o lleva más tiempo porque hay más personas?

Podemos simular el modelo con poblaciones de 50, 100, 150 y 200, y examinar los resultados. Mantendremos constante el tamaño del mundo, de modo que, a medida que aumentemos el número de individuos, también estamos aumentando la densidad de población. En el proceso anotaremos el tiempo para que se infecte toda la población :

Con base en estos resultados, concluimos que a medida que aumenta la densidad de población, el tiempo para la infección completa disminuye drásticamente. Esto tiene sentido, al comienzo cuando la primera persona se infecta, si no hay muchas otras personas alrededor, la persona no tiene a nadie a quien infectar y, por lo tanto, la infección aumenta lentamente. Sin embargo, si hay muchas personas alrededor, habrá muchas oportunidades de infección. Al final, cuando solo hay uno o dos agentes no infectados, será más probable que la infectemos si la población de infectados es alta

Datos de Infección				
Población	50	100	150	200
Tiempo para 100% de Infección	419	188	169	127

Supongamos que mostramos estos datos a un amigo nuestro y él no lo cree. El cree que el tiempo de infección al 100 por ciento debería crecer linealmente con la población. Después de eso, corre el modelo y recopila los mismos datos que hicimos. Sus datos están en la siguiente tabla:

Los Datos de Un amigo				
Población	50	100	150	200
Tiempo para un 100% de Infección	305	263	118	126

Estos resultados no respaldan la predicción de nuestro amigo de que el tiempo hasta el 100 por ciento de infección crecerá a medida que aumente la población, pero, por otro lado, son bastante diferentes a los resultados que fueron recolectados originalmente. De hecho, el tiempo hasta el 100 por ciento de infección para 150 y 200 aumenta en los datos de nuestro amigo, aparentemente contradiciendo nuestros resultados originales. Si ejecutamos el modelo varias veces más, es posible que nuevamente se obtengan resultados diferentes. Necesitamos entonces determinar si hay tendencias en los datos. Los datos son inconsistentes

porque la mayoría de los modelos MOBA usan aleatoriedad, cómo se mueven los agentes alrededor del paisaje no se determina específicamente, sino que es un movimiento azaroso. Claramente, entonces, un conjunto de corridas del modelo no es suficiente para caracterizar el comportamiento de este modelo. Supongamos, entonces, que recopilamos datos para diez corridas diferentes del modelo como en la siguiente tabla:

Datos Brutos										
Población	Co1	Co2	Co3	Co4	Co5	Co6	Co7	Co8	Co9	Co10
50	419	365	305	318	323	337	432	380	430	359
100	188	263	256	205	206	205	201	181	202	231
150	169	118	163	146	143	167	137	121	140	140
200	127	126	113	111	133	129	109	101	105	133

Aunque la mayoría de las corridas, se parece a nuestros resultados originales que a los del amigo, puede ser difícil ver tendencias claras. Por lo tanto, para describir estos patrones de comportamiento tiene sentido recurrir a algunas estadísticas.

8.3.1 Análisis estadístico de ABM: (más allá de los datos)

Los resultados estadísticos son la forma más común de ver cualquier tipo de información científica, si se trata de modelos computacionales, experimentos físicos, encuestas sociológicas u otros métodos que generan datos. La metodología general detrás de la estadística descriptiva es proporcionar medidas numéricas que resuman un gran conjunto de datos y describan el conjunto de datos de tal manera que no sea necesario examinar cada valor individual. Por ejemplo, supongamos que estamos interesados en determinar si una moneda es justa (es decir, es tan probable que salga cara) como sello), podemos realizar una serie de experimentos donde lanzamos la moneda y observamos los resultados. Una forma de determinar si la moneda es justa sería simplemente examinar todas las observaciones: CCCCCSSCSSS y determinar si la moneda era justa o no. Sin embargo, si quisieramos examinar mil, diez mil, o incluso un millón de tales observaciones, tomaría demasiado tiempo examinarlos todos. Una mejor manera es simplemente contar la frecuencia con la que se produce una cara, es decir, la probabilidad observada de éxito y la desviación estándar de este probabilidad observada. Es mucho más fácil ver los promedios y las desviaciones estándar al examinar grandes series de datos (por ejemplo, para CCCCCSSCSSS, la probabilidad observada es 0.5, y el resultado esperado para diez ensayos es observar cinco cabezas con una desviación estándar de 1,58). Para aplicar esta técnica a nuestro modelo de propagación de enfermedades , podemos crear estadísticas resumidas de los resultados de la tabla anterior que mostramos en la siguiente tabla:

Resumen Estadístico		
Población	Media	Desviación
50	366.8	47.39
100	213.8	27.4
150	144.4	17.65
200	118.7	12.13

A partir de estos resultados resumidos, vemos que el tiempo medio de infección al 100%, disminuye a medida que aumenta la densidad de población. Otro resultado interesante es que a medida que la población (la densidad) aumenta, la desviación estándar disminuye. Esto significa que los datos son menos variados, en otras palabras, más pruebas están cerca de la media que lejos de ella. Esto pasa porque cuando hay pocos agentes, existe la posibilidad de que los individuos no se encuentren entre sí para transmitir la enfermedad durante bastante tiempo, pero cuando hay una alta densidad de individuos, hay menos probabilidad de que esto ocurra, lo que significa que el tiempo hasta 100% de infección permanece más cerca del tiempo promedio. Estos resultados parecen confirmar nuestra hipótesis original de que a medida que aumenta la densidad de población el tiempo medio de infección disminuye. El análisis estadístico es un método común de confirmar o rechazar hipótesis. Al examinar inicialmente un MOBA, podemos comenzar por explorar el espacio de posibilidades (el espacio de parámetros). Diseñando un experimento como el anterior, y analizando los resultados es cómo comenzamos a confirmar o rechazar estas hipótesis. Los MOBAs crean grandes cantidades de datos (el modelo de propagación de la enfermedad es solo un pequeño ejemplo), y si podemos resumir estos datos podemos examinar grandes cantidades de resultados de manera eficiente. Numerosas herramientas fácilmente disponibles pueden facilitar la realización de análisis estadísticos. Por ejemplo, Microsoft Excel, el paquete R de código abierto, SAS, Mathematica y Matlab, todos tienen paquetes y conjuntos de funciones que ayudan en el análisis de grandes conjuntos de datos.

NetLogo tiene una poderosa herramienta para generar experimentos llamado Analizador de Comportamiento (BehaviorSpace), (ver C.1) que miraremos en capítulos posteriores

Part V

Un Modelo de Infección

Construiremos un importante modelo de infección

Chapter 9

El Modelo SI de Infección

9.1 Construcción del Modelo

9.1.1 Introducción

En términos históricos, las enfermedades infecciosas han constituido una amenaza muy grave para la sociedad. Durante la mayor parte del siglo XX las pandemias (epidemias que se propagan por áreas y poblaciones de enorme tamaño) se habían ya considerado amenazas del pasado; la medicina moderna se había ocupado para siempre de la peste, la viruela y otras catástrofes de carácter contagioso. No obstante, los cambios ambientales actuales han propiciado cambios en las distribuciones geográficas de organismos en general y de parásitos en particular. La resistencia a los agentes antimicrobianos también se ha convertido en un grave problema mundial. Algunas infecciones, antes fáciles de tratar con antibióticos, representan ahora una grave amenaza para la salud en todas partes. El caso de Toronto (Canadá), la única ciudad de un país occidental en la que la epidemia del síndrome respiratorio agudo grave (SRAG) se ha extendido de forma local, es un claro ejemplo de ello. Por lo tanto, en años recientes, las enfermedades infecciosas como malaria, tuberculosis, VIH/SIDA, SRAG y la posibilidad del bioterrorismo han provocado de nueva cuenta un gran efecto económico y de salud, sea en países desarrollados o del tercer mundo, lo cual indica que esta amenaza sigue presente. Por ello. El uso de modelos basados en agentes es muy útil para estudiar la dinámica de transmisión y control de las enfermedades infecciosas e idear programas efectivos de control.

Describiremos e implementaremos un modelo básico de Infección que luego iremos extendiendo con mayores funcionalidades y paralelamente iremos realizando análisis del modelo.

9.1.2 Descripción del Modelo

Se tiene un número de personas en un espacio cerrado, las personas se mueven al azar dentro del espacio cerrado todas a una misma velocidad. Cuando una persona sana se encuentra “cerca” de una persona infectada, esta adquiere inmediatamente la infección, se asume que una persona infectada no vuelve a quedar sana.

Inicialmente solo **una** persona se encuentra infectada y las demás sanas

Queremos entender como evoluciona la infección en este espacio cerrado, las dimensiones del espacio cerrado son fijas y también fijaremos el número de personas infectadas inicialmente (en este caso una). El número de personas en el cuarto es variable (digamos de 50 a 200 personas) y queremos medir:

- el tiempo que transcurre hasta que **todas** las personas quedan infectadas

9.1.3 Pregunta

La pregunta inicial que queremos responder con el modelo es:

- ¿Qué relación hay entre el tiempo para la infección total y la densidad de personas en el cuarto?

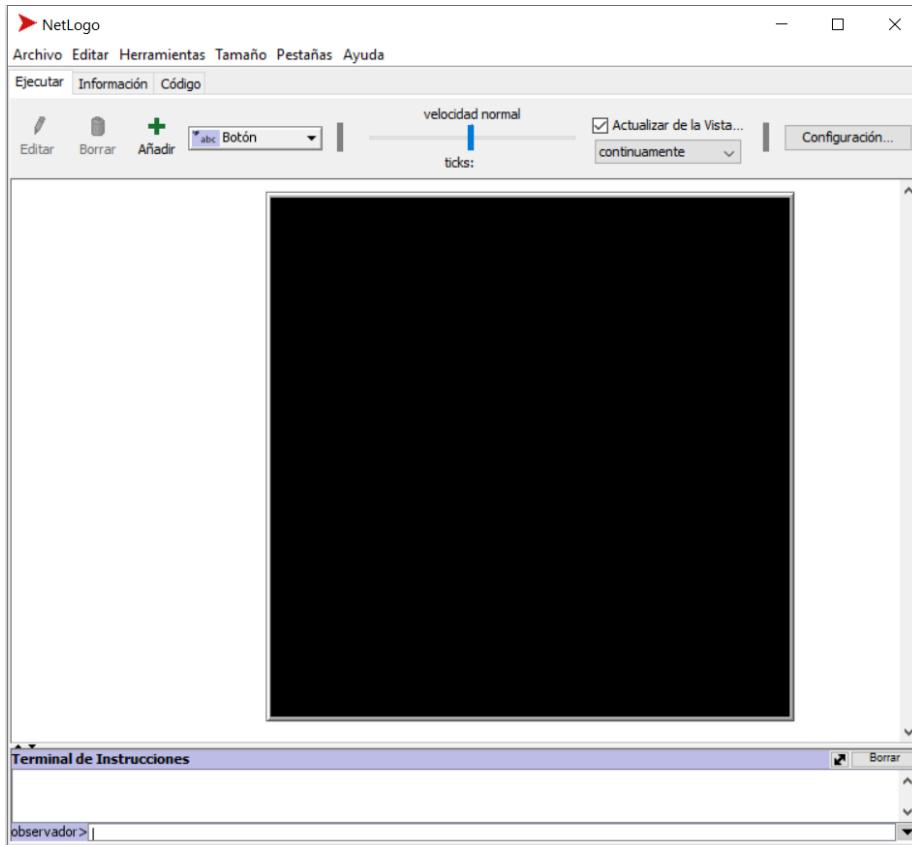
Por ejemplo:

*¿Entre más densidad de personas hay en el cuarto el tiempo de infección total es mayor o menor?

Por ejemplo si tengo 200 personas, estas se demorarán más en infectarse comparado con 50 personas, o por el contrario las 50 personas demoran más en infectarse.

Implementando el Modelo

Abra el Programa NetLogo:



9.1.3.1 Setup

Haga clic en la pestaña de código y coloque lo siguiente:

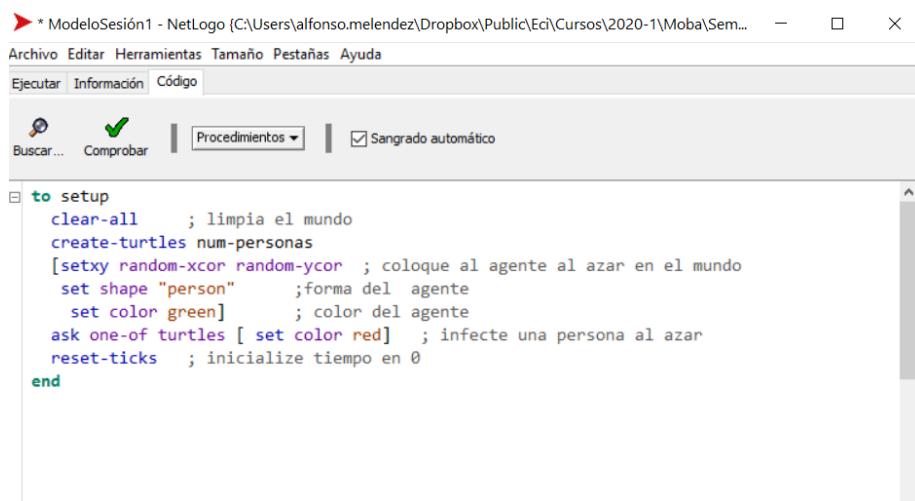
```
to setup
  clear-all      ; limpia el mundo
  create-turtles num-personas
  [setxy random-xcor random-ycor ; coloque al agente al azar en el mundo
   set shape "person"           ; forma del agente
   set color green]            ; color del agente

  ask one-of turtles [ set color red]    ; infecte una persona al azar
  reset-ticks    ; inicialize tiempo en 0
  end
```

El procedimiento setup en su orden:

- limpia el mundo : *clear-all*

- crea agentes de acuerdo con el deslizador num-personas : *create-turtles num-personas*
- se define atributos a los agentes:
- forma : *set shape "person"*
- color : *set color green*
- se selecciona un agente al azar y se infecta : *ask one-of turtles [set color red]*
- Se pone el tiempo del modelo en cero : *reset-ticks*



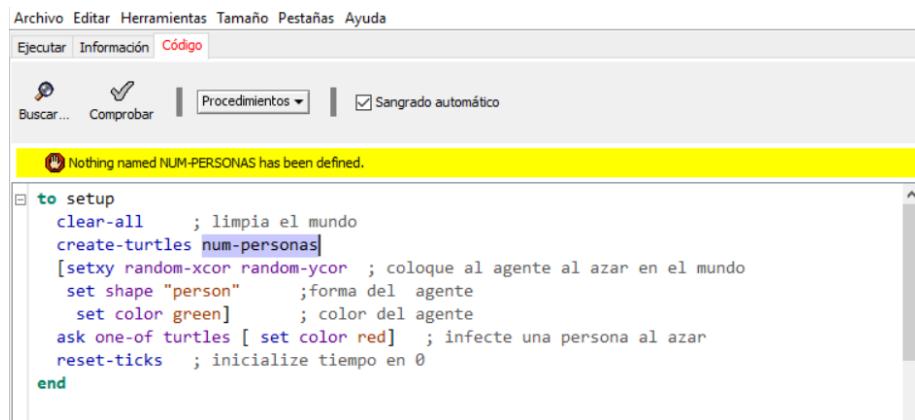
The screenshot shows the NetLogo interface with the title bar 'ModeloSesión1 - NetLogo (C:\Users\alfonso.melendez\Dropbox\Public\Eci\Cursos\2020-1\Moba\Sem...)' and menu bar 'Archivo Editar Herramientas Tamaño Pestañas Ayuda'. The 'Código' tab is selected. The code area contains the following NetLogo code:

```

to setup
  clear-all      ; limpia el mundo
  create-turtles num-personas
  [setxy random-xcor random-ycor ; coloque al agente al azar en el mundo
   set shape "person"           ; forma del agente
   set color green]            ; color del agente
  ask one-of turtles [ set color red] ; infecte una persona al azar
  reset-ticks                ; inicialize tiempo en 0
end

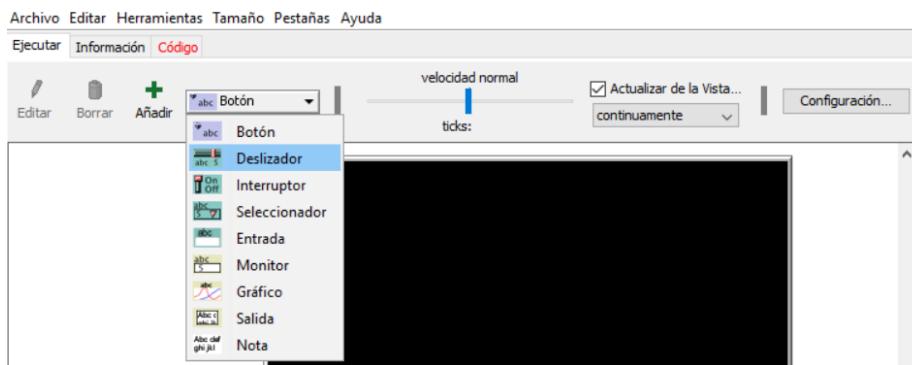
```

Oprima el botón comprobar, aparece el siguiente error en amarillo:

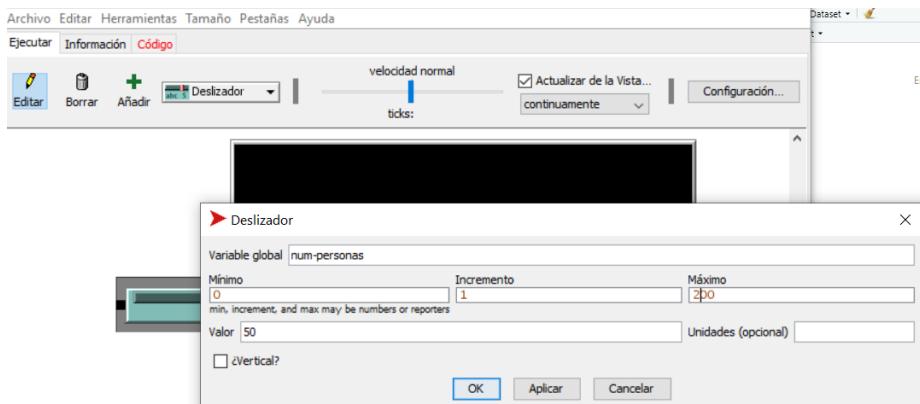


The screenshot shows the NetLogo interface with the same title and menu bar. The 'Código' tab is selected. A yellow error message box at the top says 'Nothing named NUM-PERSONAS has been defined.' Below it, the code area shows the same NetLogo code as the previous screenshot.

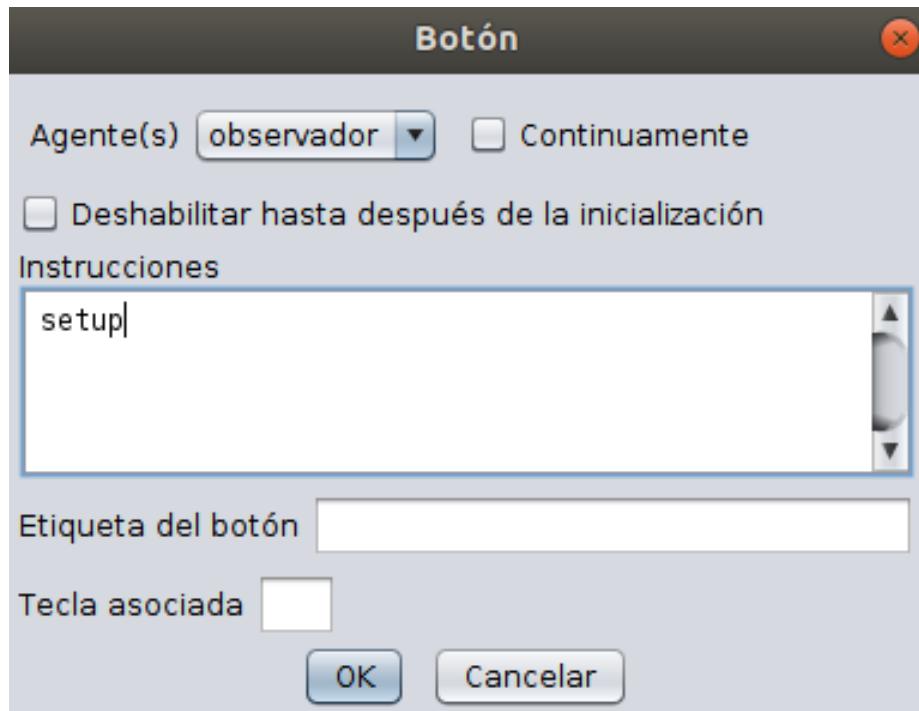
Este error indica que num-personas no está definida, la mejor manera es definirla a través de un deslizador para que sea un parámetro del modelo, entonces: Seleccione la pestaña Ejecutar y del menú que dice Botón seleccione la opción deslizador:



Haga clic a la izquierda de la pantalla negra, aparece una ventana, llénela de la siguiente manera y oprima ok.



Haga clic en la pestaña código de nuevo y oprima el botón comprobar ya no debe aparecer el error. Ahora creamos un botón pra el procedimiento **setup**: Seleccione la opción Botón, haga clic en la parte izquierda de la pantalla y llene la ventana que aparece de la siguiente manera:



Haga clic varias veces en el botón set up para comprobar que los agentes (personas) se crean en el mundo.

9.1.3.2 Go

Una vez definido el procedimiento de setup, vamos a definir el comportamiento de los agentes, vaya a la pestaña de código y coloque lo siguiente:

```

to go
  if all? turtles [color = red] [stop] ; si todas las personas se infectan pare
  ask turtles [
    move ] ; los agentes (personas) se mueven al azar en el mundo

  ask turtles with [color = red] [
    infect] ; las personas infectadas miran si pueden infectar a otras
    tick
  end

  to infect
    ask (turtles-on neighbors) with [color = green]
      [set color red] ; si una persona es vecina de una infectada, se infecta
    end
  
```

```

to move
  move-to one-of neighbors ; los agentes seleccionan al azar una párcele vecina y se mueven
end

```

Este procedimiento go en su orden:

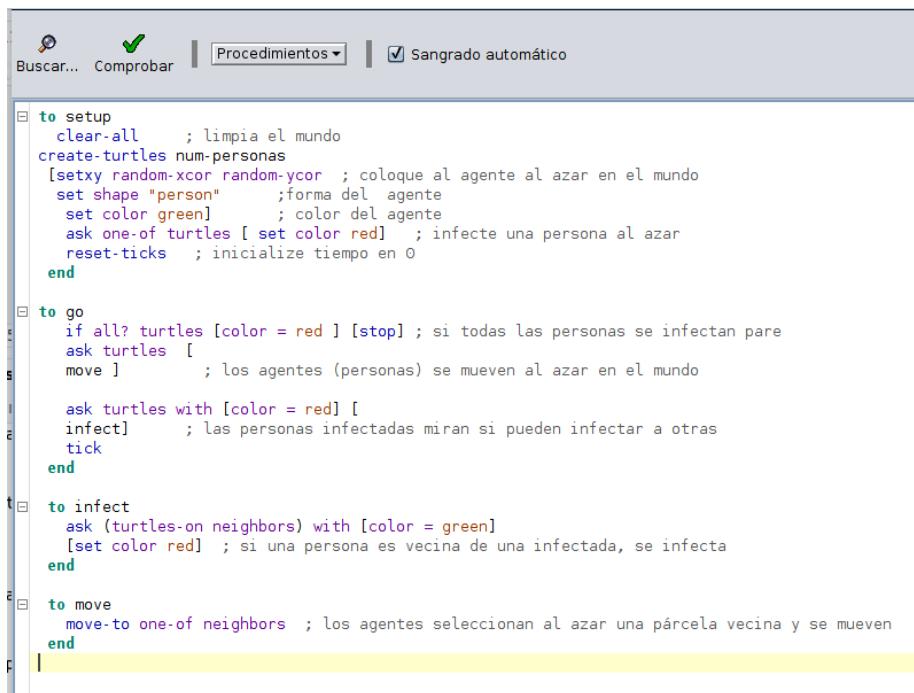
- Determina cuando el modelo para: *if all? turtles [color = red] [stop]*
- pone a mover los agentes al azar en el mundo : *ask turtles [move]*
- digale a los agentes rojos: infecten : *ask turtles with [color = red] [infect]*

¿Como infecta un agente rojo? (procedimiento infect) :

1. dígale a mis agentes vecinos que estén sanos: (*turtles-on neighbors*) with *[color = green]*
2. inféctense: *[set color red]*

La frase completa es: “vecinos mios sanos, enférmense”: *ask (turtles-on neighbors) with [color = green] [set color red]*

(Nota: *neighbors* es el conjunto de parcelas vecinas y (*turtles-on neighbors*) es el conjunto de agentes(personas) que están sobre esas parcelas)



```

Buscar... Comprobar | Procedimientos | Sangrado automático

to setup
  clear-all      ; limpia el mundo
  create-turtles num-personas
  [setxy random-xcor random-ycor ; coloque al agente al azar en el mundo
  set shape "person"           ;forma del agente
  set color green]            ; color del agente
  ask one-of turtles [ set color red] ; infecte una persona al azar
  reset-ticks ; inicialize tiempo en 0
end

to go
  if all? turtles [color = red] [stop] ; si todas las personas se infectan pare
  ask turtles [
    move ]          ; los agentes (personas) se mueven al azar en el mundo

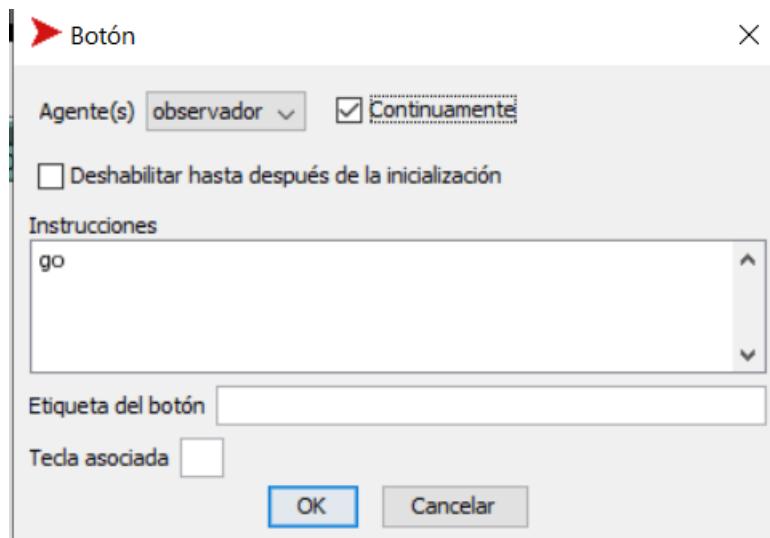
    ask turtles with [color = red] [
      infect]       ; las personas infectadas miran si pueden infectar a otras
      tick
    end
  end

  to infect
    ask (turtles-on neighbors) with [color = green]
    [set color red] ; si una persona es vecina de una infectada, se infecta
  end

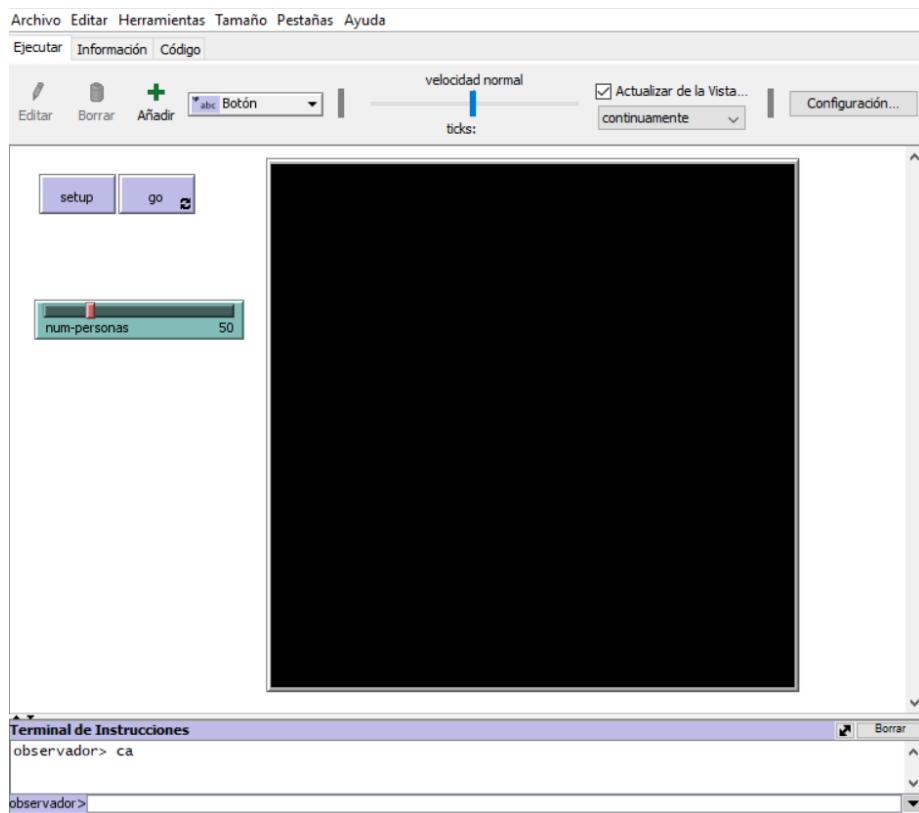
  to move
    move-to one-of neighbors ; los agentes seleccionan al azar una párcele vecina y se mueven
  end

```

Seleccione de nuevo la opción botón y haga clic a la derecha del botón setup, aparece una ventana llénela de la siguiente manera:



Listo, ya podemos observar el comportamiento del modelo, la interfaz es la siguiente:



Haga clic en Setup y luego en Go para observar la funcionalidad del modelo.

(Nota: si el modelo corre muy rápido ajuste el deslizador de velocidad situado en la parte superior central)

Puede ensayar la funcionalidad del modelo en el siguiente applet:

```
## TypeError: Attempting to change the setter of an unconfigurable property.  
## TypeError: Attempting to change the setter of an unconfigurable property.
```

9.2 Análisis del Modelo

9.2.1 La pregunta

Recordemos la pregunta planteada que queremos investigar:

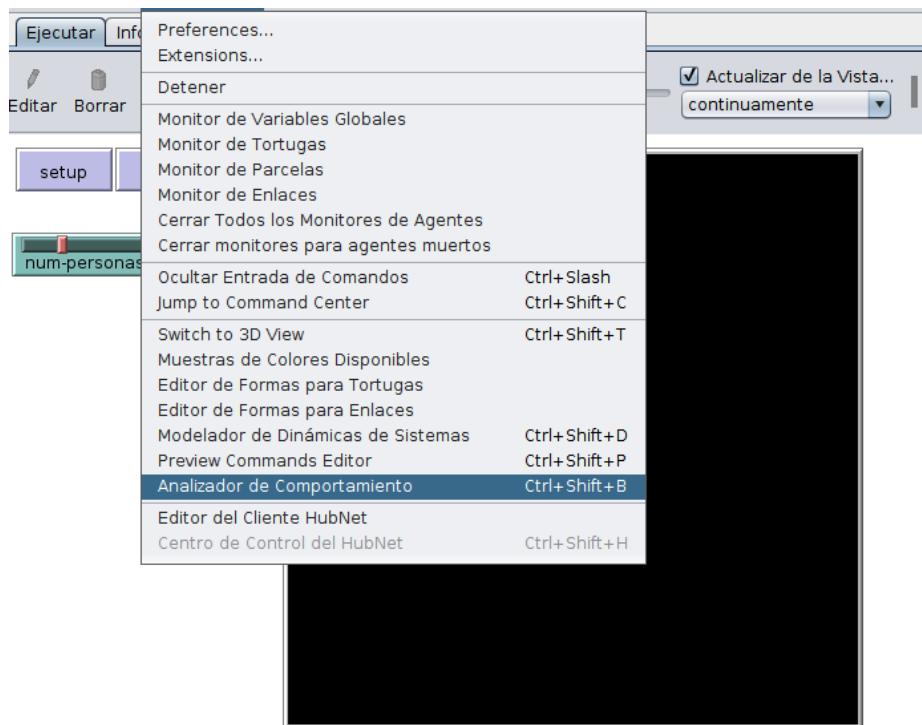
- ¿Entre más densidad de personas en el cuarto, el tiempo que transcurre hasta una infección de todas las personas es mayor o menor?

Por ejemplo si tengo 200 personas, estas se demorarán más en infectarse comparado con 50 personas, o por el contrario las 50 personas demoran más en infectarse. Recuerde también que inicialmente hay solo **una** persona contagiada.

9.2.2 Diseñando el experimento

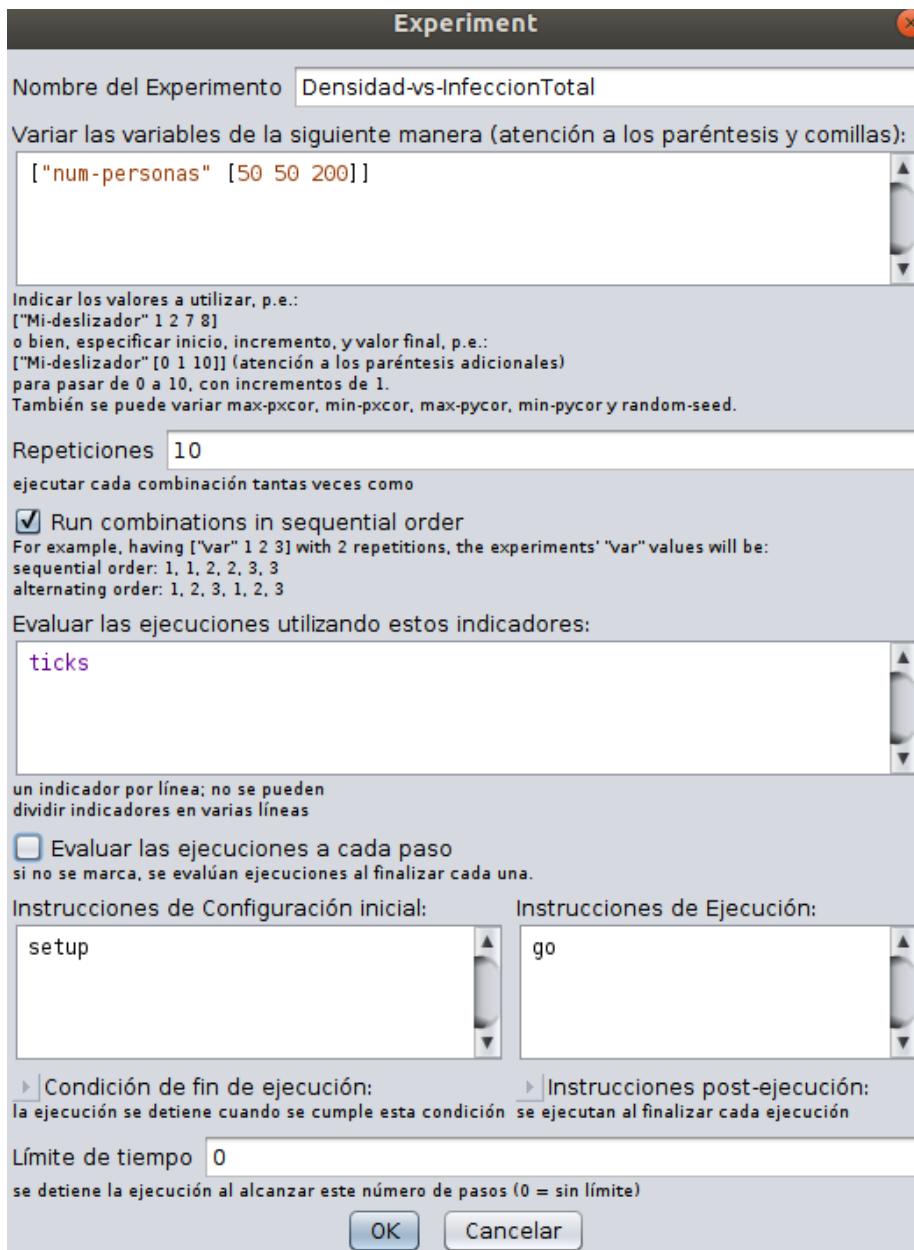
Vamos entonces a definir el experimento que nos permitirá responder a la pregunta

Abra el modelo de infección que acabamos de construir y seleccione la opción Analizador de Comportamiento del Menú Herramientas



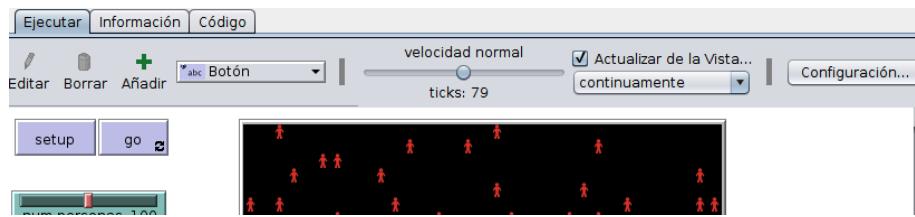
(Nota: Para mayor información del Analizador de Comportamiento puede leer el tutorial en el siguiente enlace: ??

Se necesita calcular para diferentes densidades (Número de personas en el cuarto) el tiempo de infección total, entonces vamos a seleccionar cuatro valores de densidad (509,100,150,200) y para cada una de estas densidades vamos a calcular el tiempo de infección total, realizaremos para cada una de estas densidades 10 experimentos, entonces la ventana del analizador de comportamiento la llenamos de la forma siguiente:



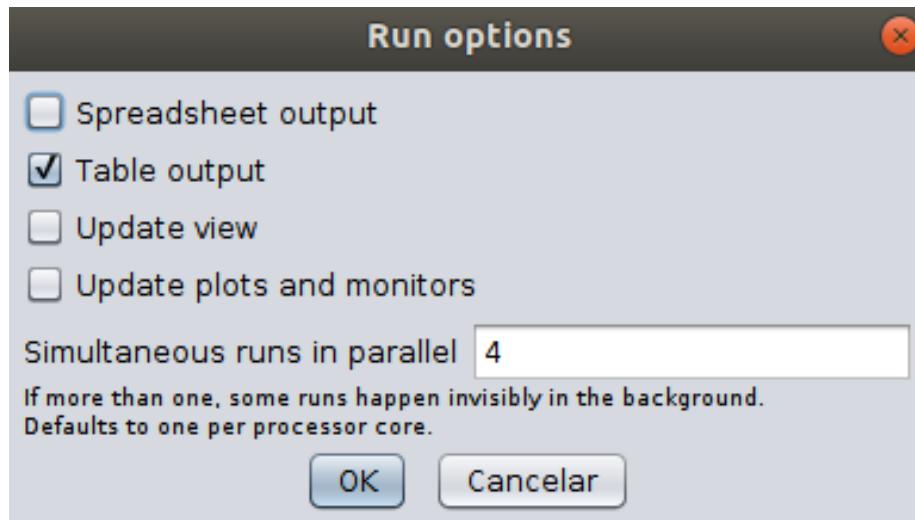
- Observe que le dimos un nombre al experimento, en este caso “Densidad-vs-InfeccionTotal”.
- las variables a medir se colocan en la ventana “Evaluar las ejecuciones utilizando estos indicadores”, en este caso vamos a medir el tiempo para la infección total, este tiempo será el valor de la variable ticks cuando el

modelo deja de correr, es el número de ticks que aparece en la parte superior central de NetLogo debajo de Velocidad Normal, cuando el modelo termina de correr.



- Al deshabilitar la opción “Evaluar las ejecuciones a cada paso” solo se registraran los datos cuando el modelo termina de correr y no en cada tick del modelo ya que lo que queremos es el tiempo final cuando todas las personas se infectan.

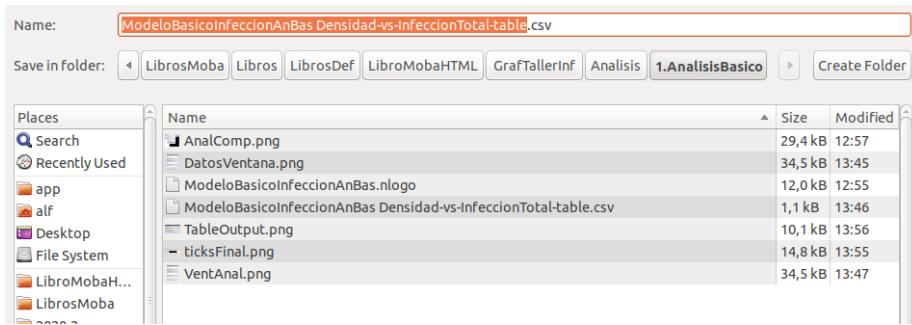
Listo, ahora hay que ejecutar el experimento oprima el botón Ejecutar aparece:



Seleccione la opción Table Output y Oprima Ok.

(Nota: La opción “Simultaneous runs in paralell” indica cuántos procesadores tiene su computador y ayuda a acelerar el proceso de generación de experimentos, la recomendación es no modificar este número)

Aparece la siguiente ventana:

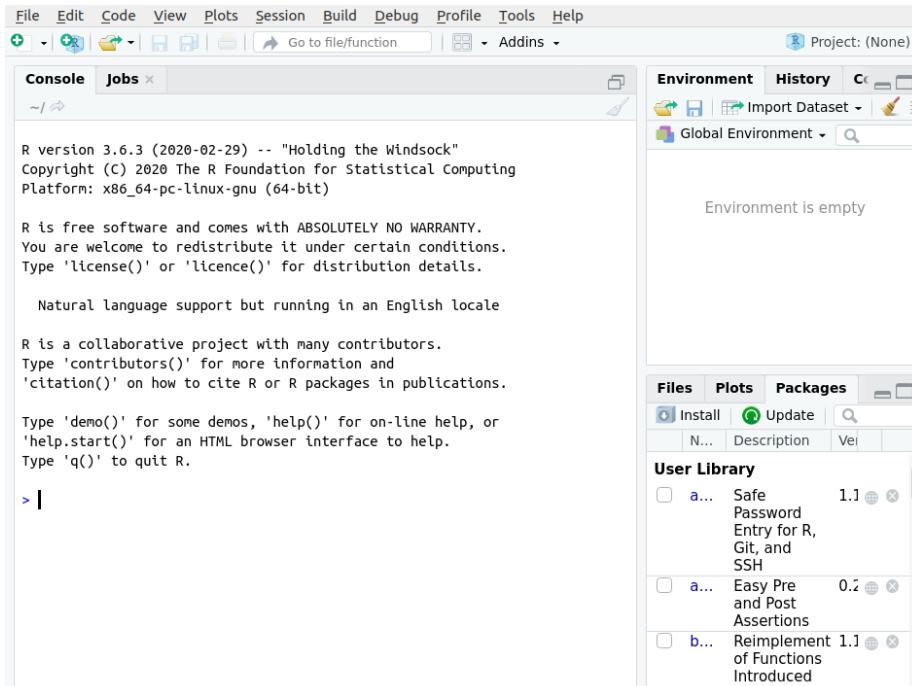


Es conveniente que guarde este archivo en el mismo directorio donde se encuentra el Modelo de Infección NetLogo, también es conveniente que **no le cambie** el nombre al archivo. Una vez guardado este archivo de datos que contiene el resultado de los experimentos definidos vamos a importarlo a R (Rstudio) para realizar el análisis de los datos generados:

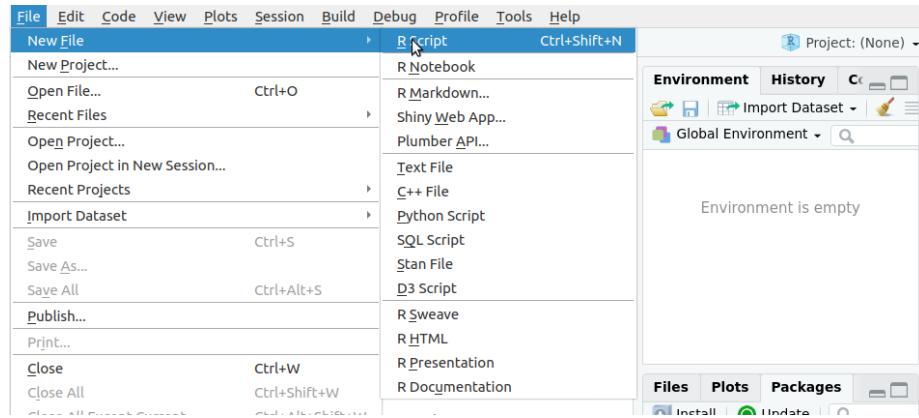
9.2.3 Análisis del Modelo en Rstudio

9.2.3.1 Configurando R studio

Abra el programa RStudio:

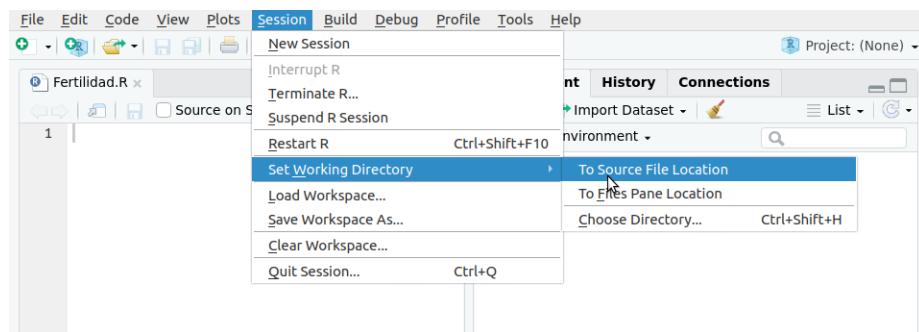


Cree un Nuevo archivo en Rstudio (Script):



Guardelo en el mismo directorio donde se encuentran el Archivo NetLogo y el archivo de datos que se acaba de generar, llame el archivo Densidad-InfTotal.R:

Ahora hay que definir el directorio de trabajo, para ello seleccione del Menú “Session”, la opción To “Source File Location” de “Set Working Directory”:



Todos los archivos que usemos, Rstudio supondrá que están en este directorio.

9.2.3.2 Leyendo los datos de NetLogo a R

Coloquemos en la Primera linea del Archivo Script que acabamos de crear lo siguiente:

```
library(tidyverse)
```

Coloque el cursor en esta ,linea y oprima el botón Run situado en la parte derecha de la ventana de Rstudio:

```
1 library(tidyverse)
2
```

(Nota: Si la librería tidyverse no carga debe instalar previamente el paquete tidyverse, para ello hay que colocar en la ventana inferior izquierda de R studio (Consola) el comando:install.packages("tidyverse"))

Con esta linea se carga la librería que usaremos para el análisis Vamos ahora a importar los datos del experimento, coloque la siguiente linea:

```
data <- read_csv("ModeloBasicoInfeccionAnBas Densidad-vs-InfeccionTotal-table.csv", skip=6)
```

Observe que:

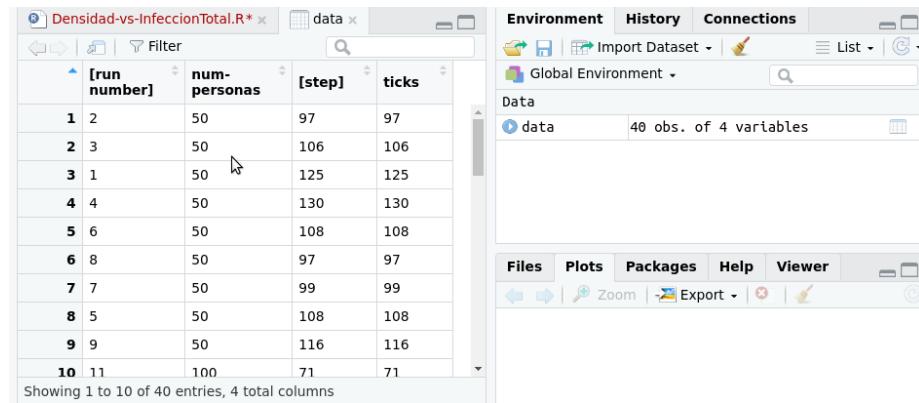
- “ModeloBasicoInfeccionAnBas Densidad-vs-InfeccionTotal-table.csv”, es el nombre del archivo de datos que generamos.
- la opción skip=6 hay que usarla en R para todos los archivos que importemos de NetLogo ya que NetLogo coloca en las primeras cinco líneas de los archivos quem genera la infoamación del Archivo (MetaDatos) y esta información no la usaremos en nuestro análisis.
- data es el nombre que tendrá el archivo en R de los datos que vamos a importar
- read_csv es el comando para leer archivos de datos este comando es un comando muy potente para leer datos y es parte de la librería tidyverse.

Coloque el cursor en la línea que acabamos de escribir y oprima el botón Run.

```
1 library(tidyverse)
2
3 data <- read_csv("ModeloBasicoInfeccionAnBas Densidad-vs-InfeccionTotal-table.csv", skip=6)
4
```

	Global Environment
data	40 obs. of 4 variables

Si todo salió bien, Rstudio lee el archivo NetLogo (data) y lo coloca en la parte derecha de la pantalla (data, 40 obs of 4 variables). Si hace clic en data, puede visualizar en la parte izquierda el archivo de datos:



	[run number]	num-personas	[step]	ticks
1	2	50	97	97
2	3	50	106	106
3	1	50	125	125
4	4	50	130	130
5	6	50	108	108
6	8	50	97	97
7	7	50	99	99
8	5	50	108	108
9	9	50	116	116
10	11	100	71	71

Showing 1 to 10 of 40 entries, 4 total columns

Listo!! Ya tenemos los datos en Rstudio, puede observar los datos en la tabla y comenzar a ver cuál puede ser la respuesta a nuestra pregunta, pero es mejor usar unos pocos comandos de la librería tidyverse para analizar y visualizar los datos de una mejor manera.

9.2.3.3 Limpiando los datos

Cambiemos los nombres de las columnas, coloquemos en R y oprimanos el botón Run:

```
colnames(data) <- c("corrida", "personas", "paso", "ticks")
```

El nombre de las columnas cambia:



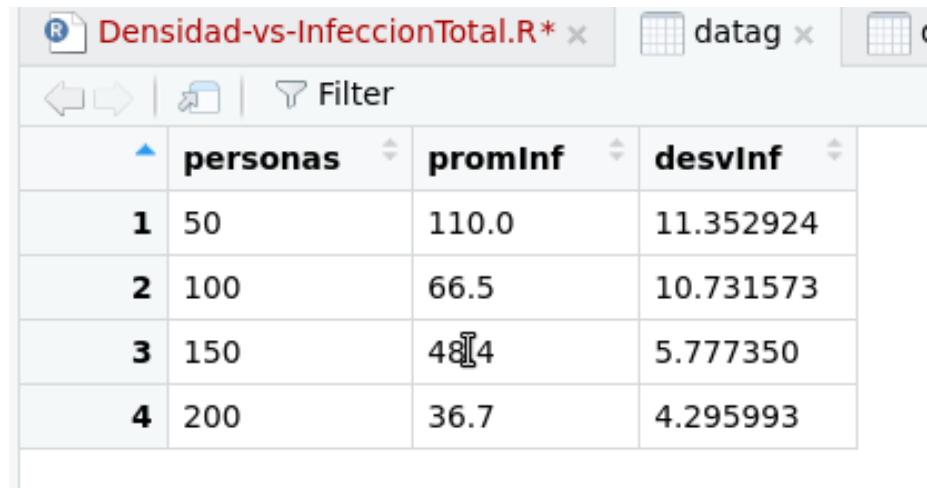
	corrida	personas	paso	ticks
1	2	50	97	97
2	3	50	106	106
3	1	50	125	125
4	4	50	130	130
5	6	50	108	108
6	8	50	97	97
7	7	50	99	99
8	5	50	108	108
9	9	50	116	116
10	11	100	71	71

Agrupemos nuestros datos de acuerdo al número de personas y generemos los promedios de infección total y las desviaciones:

```
data %>% group_by(personas) %>%
  summarise(promInf=mean(ticks),
            desvInf=sd(ticks)) -> datag
```

- `group_by` agrupa los datos de acuerdo al número de personas
- `summarise` calcula promedio y desviaciones de los datos agrupados
- estos nuevos datos se colocan en una nueva tabla (`datag`)

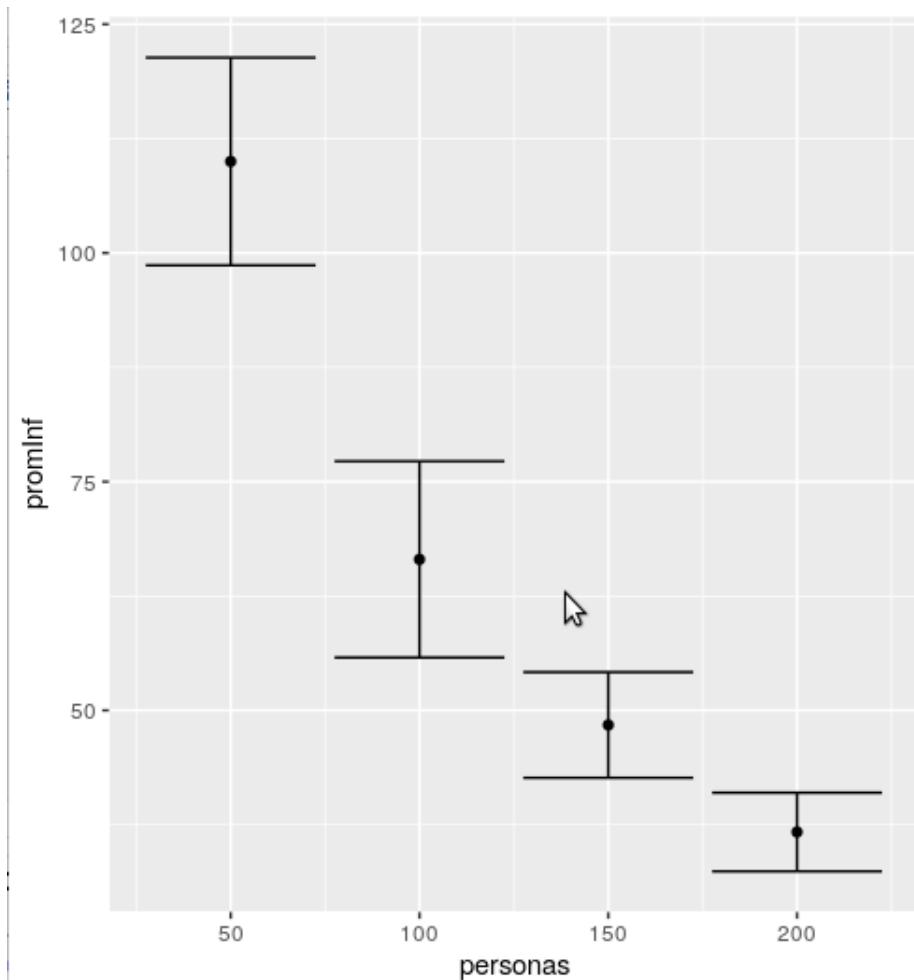
La tabla `datag` contiene la información que necesitamos graficar:



	personas	promInf	desvInf
1	50	110.0	11.352924
2	100	66.5	10.731573
3	150	48.4	5.777350
4	200	36.7	4.295993

Construyamos un gráfico de estos datos:

```
ggplot(datag)+  
  geom_point(aes(personas,promInf))+  
  geom_errorbar(aes(personas,promInf,ymax=promInf + desvInf,ymin=promInf - desvInf))
```



En esta gráfica observamos claramente que a mayor densidad menor tiempo de infección (O sea entre más personas hay en el cuarto más rápido se infectan)

9.2.4 Un Segundo Análisis (Infección Ambiental)

```
library(tidyverse)
library(plotly)
library(reshape2)

leer datos
data <- read_csv("ModeloInfeccionEspacialAmbiental_Ambiental-vs-Espacial-table.csv", skip=6)

## Parsed with column specification:
```

```
## cols(
##   `run number` = col_double(),
##   variant = col_character(),
##   `disease-decay` = col_double(),
##   `num-infected` = col_double(),
##   `num-people` = col_double(),
##   `connections-per-node` = col_double(),
##   `[step]` = col_double(),
##   ticks = col_double()
## )
```

renombras

```
colnames(data) <- c("run", "v", "decay", "w", "personas", "z", "b", "ticks")
```

filtrar

```
data %>% select(decay, personas, ticks) -> data
```

agrupar

```
data %>% group_by(decay, personas) %>%
  summarise(pticks=mean(ticks)) -> datag
```

Contour 2D

```
mat <- acast(datag, decay ~ personas, value.var="pticks")

ejey <- as.numeric(rownames(mat))
ejex <- as.numeric(colnames(mat))

fig <- plot_ly(
  x=ejex, y=ejey, z=mat, type="contour" )

fig
```

```
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

Contour 3D

```
library(reshape2)
library(plotly)
mat <- acast(datag,decay ~ personas,value.var="pticks")

ejey <- as.numeric(rownames(mat))
ejex <- as.numeric(colnames(mat))

fig <- plot_ly() %>%
  add_surface(x=~ejex, y = ~ejey, z = ~mat) %>%
  layout(title="decay-personas vs ticks",
         scene=list(
           xaxis=list(title="personas"),
           yaxis=list(title="decay"),
           zaxis=list(title="ticks"))

  ))
fig

## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```


Chapter 10

Verificación, Validación Replicación

En los capítulos anteriores, hemos defendido la importancia y utilidad de los MOBAs, aprendido a extender un modelo existente y construir modelos nuevos. En este capítulo, aprenderemos a evaluar la corrección y validez de un MOBA

- ¿Cómo podemos saber si nuestro MOBA implementado corresponde a nuestro modelo conceptual?
- ¿Cómo podemos medir la correspondencia entre nuestro MOBA y el mundo real?

10.1 Corrección de un modelo

Si un modelo es útil para responder preguntas del mundo real, es importante que el modelo proporcione resultados que abordan los problemas relevantes y que los resultados sean precisos. El modelo debe proporcionar resultados que sean útiles para el usuario del modelo. La precisión del modelo puede ser evaluada a través de tres procesos diferentes:

- Validación, Verificación y Replicación.

La validación del modelo es el proceso de determinar si el modelo implementado corresponde a, y explica, algún fenómeno en el mundo real. Verificación del modelo es el proceso de determinar si un modelo implementado corresponde al modelo conceptual que se usó para su construcción, este proceso es equivalente a asegurarse de que el modelo ha sido implementado correctamente. Por último, la replicación es la coherencia entre el modelo original y un modelo implementado por un investigador o Grupo de investigadores distinto. Al garantizar que un modelo implementado corresponde a un modelo conceptual (verificación) cuyas

salidas se reflejan en el mundo real (validación), la confianza crece en la corrección y el poder explicativo de los modelos. Además, mientras otros científicos y constructores de modelos replican el trabajo original, la comunidad científica específica en su conjunto es la que llega a aceptar el modelo como correcto. Verificación validación y replicación sustentan colectivamente la corrección y, por lo tanto, la utilidad de un modelo.

Sin embargo, demostrar que un conjunto particular de resultados de un modelo corresponde al mundo real no es suficiente. Como se discutió en capítulos anteriores, debido a la naturaleza estocástica de los MOBAs, a menudo se necesitan múltiples ejecuciones para confirmar que un modelo es exacto. Por lo tanto, las metodologías de verificación, validación y replicación a menudo se basan en métodos de estadística. Comenzamos nuestra discusión mirando más de cerca la verificación.

10.2 Verificación

A medida que un modelo basado en agentes crece, se hace más difícil simplemente mirar su código para determinar si realmente está llevando a cabo su función prevista. El proceso de verificación aborda este problema, teniendo como objetivo la eliminación de “errores” del código. Sin embargo, esto no es tan simple como puede parecer, y si los diseñadores e implementadores de modelos son personas diferentes, el proceso de depuración puede volverse mucho más complejo. Si un modelo es simple para empezar, es más fácil de verificar que un modelo complejo. Igualmente, si las partes adicionales agregadas al modelo también son de naturaleza incremental, construyendo hacia la pregunta de interés en vez de tratar de desarrollar totalmente el modelo, estos modelos serán más fáciles de verificar. Aun así, debe tenerse en cuenta que incluso si todos los componentes de un modelo son verificados, el modelo en sí puede no serlo, ya que pueden surgir complicaciones adicionales en las interacciones entre los componentes del modelo. A lo largo de esta sección, examinamos el tema de la verificación en el contexto de un simple MOBA del comportamiento de un sistema de votación, utilizaremos la siguiente narrativa ficticia para guiar nuestra discusión:

Imagine que se nos acerca un grupo de polítólogos que desean desarrollar un modelo simple de comportamiento de votación. Ellos explican que piensan que las interacciones sociales son las que determinan en gran medida la votación en las elecciones. Con base en sus observaciones de encuestas y resultados electorales, piensan que las personas tienen alguna inclinación inicial de votación, y cuando son encuestados inicialmente, expresan esos sentimientos. Sin embargo, en el intervalo entre cuando son encuestados y cuando realmente emiten su voto, hablan con sus vecinos y colegas y discuten la forma en que planean votar; esto puede cambiar la forma en que deciden votar. De hecho, esto puede suceder varias veces durante el período previo a una elección. Los polítólogos nos piden que construyamos un MOBA que ilustre este fenómeno.

10.3 Comunicación

A menudo, el implementador del modelo y el autor del modelo no son la misma persona, pero esto no es siempre el caso. A veces, un equipo de personas construye un modelo, en el que una o más personas describen el modelo conceptual mientras que otros miembros del equipo realmente implementan el modelo. Esto sucede con frecuencia cuando el experto en el dominio no tiene las habilidades técnicas para crear el modelo por su cuenta. En estas situaciones, la verificación se vuelve especialmente crítica, ya que ningún individuo tiene un conocimiento completo de todas las partes del proceso de modelado. Cuando los modelos están construidos de esta manera, la comunicación es crítica para asegurar que el modelo implementado refleje correctamente el modelo conceptual del experto en el dominio. La mejor manera de verificar modelos construidos en este tipo de equipos es que el experto en el dominio (o expertos) se familiaricen con las herramientas del modelo y, asimismo, los implementadores aprendan sobre el tema del modelo. Si bien uno no puede esperar que las dos partes se conviertan en expertos en los dominios del otro, construir este terreno común es esencial para garantizar que las ideas se comuniquen de manera efectiva y el modelo refleja correctamente las intenciones de los modeladores.

Por ejemplo, en nuestro modelo de votación, sería útil que los polítólogos conocieran la diferencia entre los tipos de vecindades (Moore y von Neumann) que existen en los modelos basados en agentes, este conocimiento les permitiría tomar decisiones informadas sobre cómo su modelo conceptual debería ser implementado. Además, ayudaría si el implementador tuviera una idea básica de cómo los mecanismos de votación se conceptualizan dentro de la disciplina de la ciencia política, ya que podría ayudarlos a darse cuenta de posibles simplificaciones para el modelo o incluso potenciales “trampas” en el modelo a medida que se implementa. Por ejemplo:

- ¿es razonable suponer que solo hay dos partidos?
- ¿es razonable suponer que el grupo de amigos de cada persona amigos no cambia durante el período de tiempo modelado en la simulación?

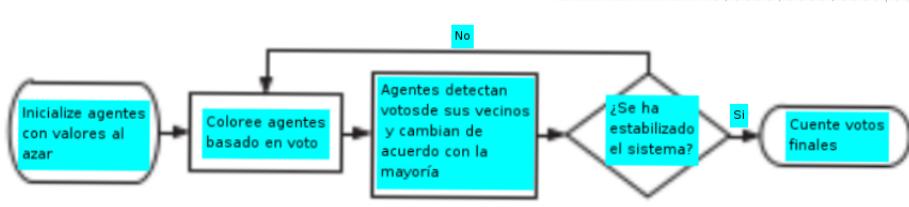
Cuando se trata de la comunicación del modelo conceptual, a menudo hay espacio para errores humanos y malentendidos, en una situación ideal, el autor del modelo y el implementador son la misma persona, lo que evita el tipo de errores de comunicación que pueden resultar de diferentes vocabularios y supuestos diferentes. Sin embargo, el tiempo requerido para los expertos en el dominio para aprender programación de computadora, o por el contrario, el tiempo requerido para los programadores de computadora para aprender un dominio particular, puede ser sustancial. Esto fue particularmente cierto en las décadas pasadas, cuando a menudo era inviable convertirse en un implementador de modelos y un experto. Sin embargo, los nuevos lenguajes de MOBAs de bajo umbral, como NetLogo, tienen un objetivo explícito de disminuir la cantidad de tiempo necesario para aprender a escribir MOBAs, por lo tanto reducir (o eliminar) la brecha entre el autor y el implementador.

10.4 Descripción de Modelos Conceptuales

A medida que comenzamos a implementar el modelo de votación, podemos darnos cuenta de que hay algunos mecanismos y propiedades de agentes que no entendemos completamente cuando hablamos con los politólogos. Para aliviar este problema, se decide escribir un documento que describa cómo planeamos implementar el modelo para que podamos verificar que nosotros y los expertos en ciencia política tienen el mismo modelo conceptual en mente. Este documento servirá como un documento más formal.

10.4.1 Descripción del modelo conceptual.

Una forma de describir modelos conceptuales en términos más formales es usar diagramas de flujo. Un diagrama de flujo es una descripción gráfica del modelo que describe el flujo de decisiones que ocurren durante la ejecución de un procedimiento de software. Para el modelo conceptual descrito arriba, podríamos usar un diagrama de flujo como el de la siguiente figura.



Los diagramas de flujo usan cuadrados redondeados para indicar los estados de inicio y final del sistema, cuadrados para indicar procesos y diamantes para indicar puntos de decisión en el código. Estos símbolos proporcionan una forma clara de entender cómo fluye el control a través del software. También podemos tomar de diagrama de flujo y reescribirlo en pseudocódigo. El objetivo es servir como punto intermedio entre el lenguaje natural y el programa formal. El pseudocódigo puede ser leído por cualquier persona, independientemente de su conocimiento de programación, mientras que, al mismo tiempo, contiene una estructura algorítmica que lo hace más fácil para implementar directamente en código real. Por ejemplo, al describir el modelo de votación, podríamos usar un pseudocódigo como este:

```

Los votantes tienen votos = {0, 1}
Para cada votante:
  Establezca el voto 0 o 1, elegido con igual probabilidad
  Loop hasta la elección
    Para cada votante
      Si la mayoría de los votos de los vecinos = 1 y voto = 0, establezca el voto 1
      De lo contrario, si la mayoría de los votos de los vecinos = 0 y voto = 1, establezca el voto 0
      Si voto = 1: establece el color azul
  
```

```

De lo contrario: color = verde
Mostrar recuento de votantes con voto = 1
Mostrar recuento de votantes con voto = 0
Bucle final

```

10.5 Verification Testing

Después de diseñar el modelo conceptual con nuestros colegas de ciencias políticas, podemos comenzar la codificación. Seguimos el principio de diseño central de MOBA y comenzamos a verificar de forma simple e incremental. La alineación entre nuestro modelo conceptual y el código. Por ejemplo, podemos escribir el procedimiento de configuración, como aquí:

```

patches-own
[
  vote ;; my vote (0 or 1)
  total ;; sum of votes around me
]
to setup
  clear-all
  ask patches [
    if (random
      [ set vote
    ]
    ask patches [
      if (random 2 =
        [ set vote
      ]
      ask patches [
        recolor-patch
      ]
      end
      2 = 0) ;; half a chance of this
      1 ]
      0) ;; half a chance of this
      0 ]
    to recolor-patch ;; patch procedure
      ifelse vote = 0
      [ set pcolor green ]
      [ set pcolor blue ]
    end

```

Luego podemos escribir una pequeña prueba que examine si el código creó el número correcto de votantes verdes y azules. En el estado inicial del modelo de votación, el número de votantes con voto 0 debería ser aproximadamente

igual al número de votantes con voto 1. Podemos verificar esto fácilmente al configurar nuestro modelo varias veces, comparando los recuentos de cada voto. Si la diferencia en estas muchas configuraciones es más de aproximadamente el 10 por ciento, por ejemplo, nuestro código podría tener un error. Si la diferencia es inferior al 10 por ciento del número total, podemos sentir relativamente confiamos en que las poblaciones se generan como pretendíamos. Este código se vería así:

```
to check-setup
let diff abs ( count patches with [ vote = 0 ] - count patches
with [ vote = 1 ] )
if diff > .1 * count patches [
print "Warning: Difference in initial voters is greater than 10 %."
]
end
```

Podemos insertar una llamada a CHECK-SETUP en la parte inferior del procedimiento de “setup”. Esta prueba luego se ejecutará cada vez que el modelo ejecute setup” y, si hay un problema, alertará quien está ejecutando el modelo y si hay un desequilibrio de votación. Usando este procedimiento de setup, la advertencia aparece casi cada vez que ejecutamos el modelo, lo que nos dice que el código no está logrando lo que pretendíamos. Además, es visualmente evidente que este código crea muchos más parcelas que son verdes (voto = 0) que azules(voto = 1). Debido a que todos los errores no son aparentes visualmente, es importante escribir pruebas de verificación (Se alienta a los lectores a examinar el código de configuración anterior y determinar su falla). Después de descubrir el error, podemos reescribir el procedimiento de configuración con lo siguiente (más simple) código, que logra el equilibrio correcto de los votantes iniciales.

```
to setup
clear-all
ask patches [
set vote random 2
]
ask patches [
recolor-patch
]
check-setup
end
```

Esta técnica de verificación es una forma de prueba unitaria, es un enfoque que implica escribir pequeñas pruebas que verifican si las unidades individuales funcionan correctamente en el código. Al escribir pruebas unitarias a medida que desarrollamos nuestro código, podemos asegurarnos de que los cambios futuros en nuestro código no interrumpan el código anterior. Dado que esta prueba unitaria se ejecutará cada vez que ejecutemos este modelo, garantiza que podamos modificar el código sin temor a que nuestros cambios alteren indetectable el código anterior. Por supuesto, esta es solo una prueba unitaria, y

hay muchas más que podrían escribirse. También es posible escribir por separado un conjunto de pruebas unitarias que no forman parte del modelo implementado, sino que están escritas para ejecutar el modelo con entradas particulares y comprobar si las salidas corresponden a los resultados esperados. Este enfoque a veces se denomina prueba unitaria “fuera de línea”. Después de estar seguros de que se verifica el código de setup, se puede comenzar un proceso similar para el procedimiento go” Traducimos el pseudocódigo en NetLogo de la siguiente manera:

```
to go
ask patches [
  set total (sum [vote] of neighbors)
]
;; this is equivalent to count neighbors with [vote = 1]
;; use two ask patches blocks so all patches compute "total"
;; before any patches change their votes
ask patches [
  ifelse vote = 0 and total >= 4 [
    set vote 1
  ]
  [if vote = 1 and total <= 4 ] [
    [set vote 0
  ]
  recolor-patch
]
tick
end
```

En este código, primero pedimos a las parcelas que calculen el número total de vecinos con voto= 1. Si la parcela está votando 0 y tiene un total mayor o igual a 4, entonces cambia su votación a 1. Del mismo modo, si la parcela está votando 1 y tiene un total menor o igual a 4, cambia su voto a 0. Después de verificar los procedimientos SETUP y GO, podemos comenzar a investigar los resultados del modelo

10.6 Más allá de la verificación

A pesar de los mejores esfuerzos para verificar que el modelo implementado corresponde al modelo conceptual, a veces producirá resultados que no parecen corresponder a lo que los implementadores y los autores planearon. Con el tiempo, puede quedar claro que no hay “error” en el modelo, sino que los resultados emergentes y sorprendentes del modelo son consecuencias no deseadas de decisiones a nivel individual que el modelador tiene codificado.

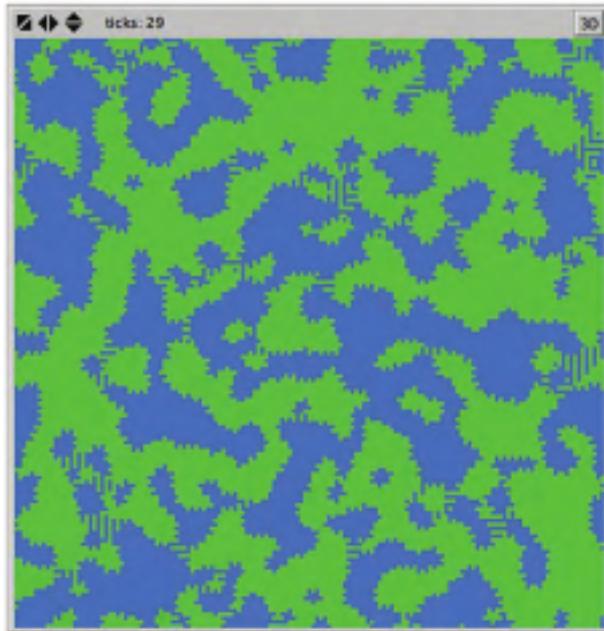


Figure 7.2
Voting model, first version.

Por ejemplo, después de codificar el modelo anterior, presentamos los resultados a nuestros colegas de ciencia política, los resultados del modelo confunden a los politólogos, porque esperaban que el modelo uniera a los votantes formando bloques estáticos con bordes suaves, en lugar de bordes irregulares que aparecen en la figura. De hecho, este modelo nunca alcanza el equilibrio; Continuamente recorre un conjunto de estados, después de un examen más detallado, queda claro que lo que está causando el ciclo del modelo y produce los bordes dentados es cuando los votos de los vecinos están empatados. Como lo codificamos, el modelo hace que los votantes cambien su voto, entonces los bordes siguen yendo y viniendo entre los dos votos. Podemos cambiar el modelo para que los votantes no cambien su voto si los votos de sus vecinos están empatados. Después de hacer esto , los bloques se unen con bordes lisos:

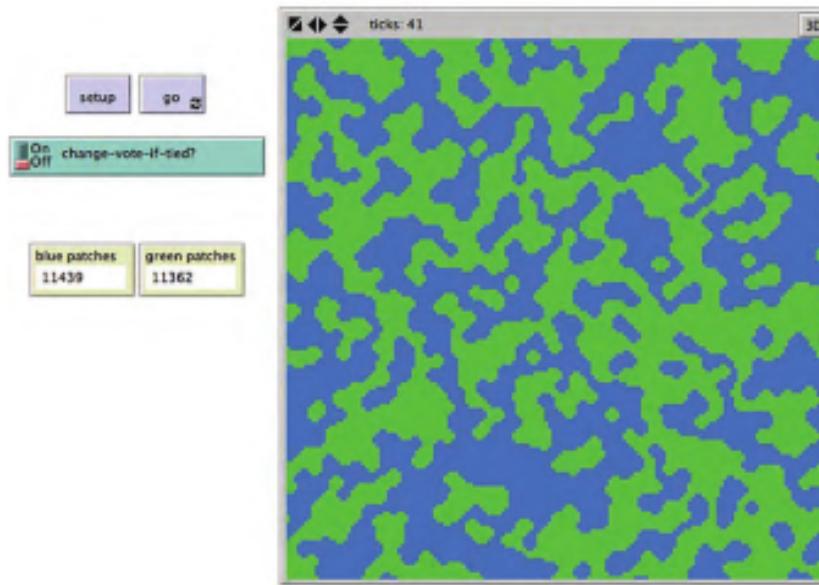


Figure 7.3
Voting model, ties stay with same vote.

Sin embargo, como los políticos se han interesado en cómo un cambio aparentemente pequeño crea una gran diferencia en el resultado, deciden hacer de este un elemento del modelo que puedan controlar agregando un interruptor, llamado CAMBIAR-VOTO-SI-EMPATE?. Tener esta opción como elemento controlable nos permite explorar nuevos comportamientos de votación, por ejemplo:

- ¿qué pasaría si los votantes decidieran ponerse del lado del partido minoritario en su barrio?

Es decir, imagine una comunidad donde, cuando los vecinos de un miembro están divididos estrechamente entre dos candidatos, el miembro decide votar por el desvalido (quien quiera que haga que la mayoría de sus vecinos no voten), ya que el miembro podría ser capaz de asegurar la victoria del desvalido. Tenga en cuenta que esta perversidad hipotética de los agentes puede no reflejar las prácticas de votación del mundo real. Sin embargo, esta es una cuestión de validación del modelo (que discutiremos más adelante), en lugar de un problema de verificación. A continuación se muestra el código para el procedimiento go con las dos opciones agregadas:

```
to go
ask patches
  [ set total (sum [vote] of neighbors) ]
  ;; use two ask patches blocks so all patches compute "total"
  ;; before any patches change their votes
```

```

ask patches
[
  if total > 5 [ set vote 1 ]
  if total < 3 [ set vote 0 ]
  if total = 4
    [ if change-vote-if-tied?
      [ set vote (1 - vote) ] ] ;; switch vote
  if total = 5
    [ ifelse award-close-calls-to-loser?
      [ set vote 0 ]
      [ set vote 1 ] ]
  if total = 3
    [ ifelse award-close-calls-to-loser?
      [ set vote 1 ]
      [ set vote 0 ] ]
  recolor-patch ]
  tick
end

```

Este ejemplo ilustra que, al examinar los resultados del modelo, puede ser difícil descifrar si el resultado de un modelo es el resultado de errores en el código, una falta de comunicación entre el autor del modelo y el implementador, o un resultado “correcto” pero no anticipado de las reglas de los agentes. Por lo tanto, es vital que el implementador del modelo y el autor del modelo discutan ambos la modelación de las reglas y los resultados con la mayor frecuencia y regularidad posible, y no simplemente cuando finalice la implementación del modelo. Saltarse este proceso de comunicación puede resultar en que el implementador termine creando un modelo con comportamientos de agentes que el autor del modelo no había definido. Sin embargo, al mantener estas comunicaciones, el autor del modelo puede descubrir que las variantes en su modelo conceptual resultan en resultados dramáticamente diferentes. Incluso si el autor y el implementador del modelo son la misma persona, es útil revisar las reglas del modelo y los resultados de forma iterativa y discutirlos con personas que están familiarizadas con el fenómeno modelado.

10.7 Análisis de sensibilidad y robustez

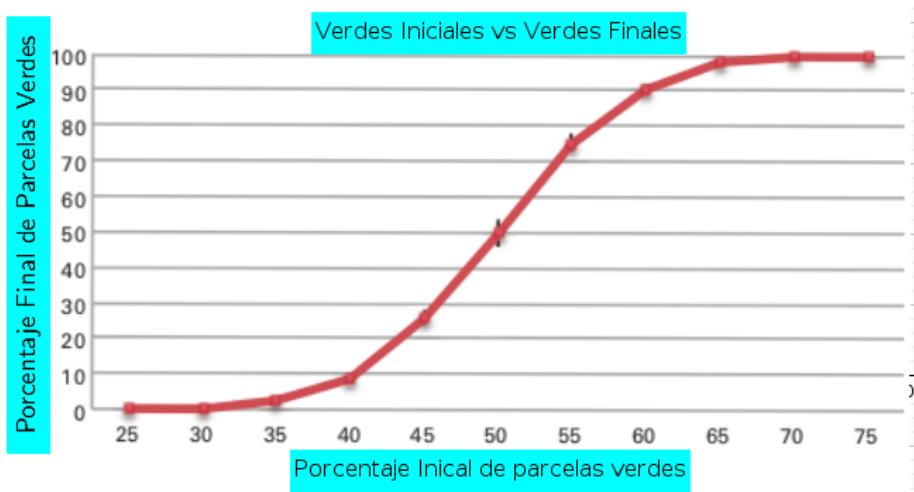
Después de que hayamos terminado de construir un modelo y encontrado algunos resultados interesantes, es importante explorar estos resultados para determinar qué tan sensible es nuestro modelo al conjunto particular de condiciones iniciales que estamos usando. A veces, esto solo significa variar un grupo de parámetros que ya tenemos dentro de nuestro modelo, pero otras veces, esto implica agregar nuevos parámetros al modelo. Este proceso, llamado análisis de sensibilidad, crea una comprensión de cuán sensible (o robusto) es el modelo a varias condiciones.

Un tipo de análisis de sensibilidad implica alterar los valores de entrada del modelo. Por ejemplo, uno de los polítólogos está preocupado por las condiciones iniciales del modelo, piensa que el comportamiento actual del modelo puede depender inicialmente de tener un equilibrio entre el número de votantes para cada partido (color). y se pregunta si inclinar el estado inicial en una dirección u otra daría como resultado que un color domine el paisaje. Para probar esta hipótesis, creamos un parámetro (un deslizadores en la interfaz del modelo) que controla el porcentaje de agentes verdes en el estado inicial del modelo. Usando El Analizador de Comportamiento C.1, ejecutamos un conjunto de experimentos donde variamos este porcentaje del 25 por ciento al 75 por ciento en incrementos del 5 por ciento. Al establecer la condición de parada para este experimento, debemos tener en cuenta que es posible tener patrones oscilantes interminables. Por esta razón, decidimos establecer dos condiciones de parada diferentes:

1. el modelo se detiene si ningún votante cambió de votos en el último tick.
2. el modelo se detendrá después de que se han ejecutado cien pasos de tiempo (ticks), ya que parece ser tiempo suficiente para llegar a un estado final del modelo. Esta investigación significará que tenemos que reevaluar nuestro proceso de verificación inicial, ya que ahora estamos deliberadamente alterando completamente la distribución inicial. Por lo tanto, podemos cambiar el código para que tome en cuenta el nuevo parámetro:

```
to check-setup
  let expected-green (count patches * initial-green-pct / 100)
  let diff-green (count patches with [ vote = 0 ] ) - expected-green
  if diff-green > (.1 * expected-green) [
    print "Initial number of green voters is more than expected."
  ]
  if diff-green < (- .1 * expected-green) [
    print "Initial number of green voters is less than expected."
  ]
end
```

Podemos examinar la relación entre la distribución inicial de votantes y el recuento final de votos haciendo diez ejecuciones para cada porcentaje inicial. Cada partido político debe contar el porcentaje de votantes finales que son verdes o azules. Después de ejecutar nuestro experimento BehaviorSpace, podemos graficar los resultados del porcentaje final contra el porcentaje de entrada, obteniendo el gráfico siguiente:



Estos resultados muestran que, a medida que nos alejamos de una distribución media inicial de votantes azules y media de votantes verdes, hay un efecto no lineal en la distribución final de los votantes. El modelo es sensible a estos parámetros, y por lo tanto, un ligero cambio en el número inicial de votantes de un partido dan como resultado un mayor número de votantes finales para ese mismo partido. Sin embargo, la definición de “sensibilidad” depende de los resultados del modelo que se esté considerando. Por ejemplo, la sensibilidad en los resultados cuantitativos no significa necesariamente que habrá sensibilidad en los resultados cualitativos. Si el resultado de su modelo principal es el hallazgo cualitativo de que se forman islas sólidas de votantes de ambos colores, dejando la segregación, entonces este resultado cualitativo sigue siendo cierto incluso si se perturba la distribución inicial de los votantes en un 10 por ciento en cualquier dirección.

Cuando están perturbadas, las islas de un color u otro pueden ser mucho más pequeñas, pero aún habrá bloques sólidos. Por lo tanto, para esta medida cualitativa, podríamos concluir que este modelo es insensible a pequeños cambios en la configuración inicial de votantes. El análisis de sensibilidad es un examen del impacto de diferentes parámetros del modelo en sus resultados. Para determinar qué tan sensible es un modelo, examinamos el efecto que diferentes condiciones iniciales y los mecanismos del agente tienen en los resultados del modelo. Además, podemos examinar el entorno en el que opera el modelo. Por ejemplo, en el modelo de votación, estamos usando una cuadrícula bidimensional, pero estos resultados podrían cambiar drásticamente si los votantes se ubicaron en una cuadrícula hexagonal, en una red o en alguna otra topología.

10.8 Beneficios de la verificación

Existen muchos beneficios al realizar análisis de verificación, que incluyen el desarrollo de una comprensión de la causa de resultados inesperados y una exploración del impacto de los pequeños cambios en las reglas de un modelo, el nivel básico de verificación es que el implementador del modelo compare la descripción conceptual del modelo con el código implementado para determinar si el modelo implementado corresponde al modelo conceptual. Cuanto más riguroso sea el proceso de verificación del modelo, es más probable que el modelo implementado resultante corresponda al modelo conceptual. Si los dos modelos corresponden exactamente, entonces el modelador y los autores entienden las reglas del modelo. Esto significa que entienden cómo el modelo genera sus resultados. Aún así, comprender los componentes del modelo no garantiza una comprensión de todas las interacciones de estos componentes o de por qué el modelo genera los resultados. La verificación es importante porque ayuda a garantizar que el autor (o autores) entiendan los mecanismos que subyacen el fenómeno que se está explorando. Sin pasar por este proceso, los autores no pueden confiar en las conclusiones extraídas del modelo. La verificación puede ser difícil de lograr porque es difícil determinar si un resultado inesperado es el producto de un error en el código, una falta de comunicación entre el autor e implementador del modelo, o un resultado inesperado de una regla, además aún puede ser difícil aislar y eliminar un error, o corregir el error incluso si estamos seguros de que el resultado es una consecuencia inesperada pero precisa. Puede ser difícil descubrir qué causó que los resultados fueran diferentes de lo esperado. El proceso de comprender cómo funciona un modelo también puede ayudarnos a comprender la pregunta de “por qué”. Por ejemplo, en el modelo anterior, como examinan los politólogos en el modelo, comienzan a comprender la razón por la que el segundo modelo se une en bloques. Al pensar en las reglas del modelo desde el punto de vista del agente, queda claro que una vez que se forma un bloque de individuos permanecerá constante, mientras que si la mayoría de los vecinos del agente votan de una manera y ninguno de ellos está cambiando, entonces el agente lo hará seguir votando de la misma manera. Por lo tanto, una vez que todos los agentes hayan alcanzado la concenso mayoritario de sus vecinos, se quedarán así para siempre. Es solo cuando damos a los agentes la capacidad de cambiar colores según los vecinos que los rodean, como en las otras dos reglas, que vemos un cambio perpetuo en los resultados. El proceso de verificación no es binario. Un modelo no está verificado o no verificado, pero más bien existe a lo largo de un continuo de verificación. Siempre es posible escribir más pruebas de o realizar más análisis de sensibilidad. Por lo tanto, depende del autor del modelo y implementador (y más tarde un replicador de modelos) para decidir cuándo terminar la verificación.

10.9 Validación

La validación es el proceso de asegurar que haya una correspondencia entre el modelo implementado y realidad. La validación, por su naturaleza, es compleja, multinivel y relativa. Los modelos son simplificaciones de la realidad; Es imposible que un modelo exhiba las mismas características y patrones que existen en la realidad. Al crear un modelo, queremos incorporar los aspectos de la realidad que sean pertinentes a nuestras preguntas. Así, cuando se emplea un proceso de validación, es importante tener en cuenta las preguntas del modelo conceptual y validar aspectos del modelo que se relacionan con estas preguntas. Hay dos ejes diferentes para considerar los problemas de validación (Rand & Rust, 2011). El primer eje es el nivel en el que se produce el proceso de validación (Microvalidación) es asegurarse de que los comportamientos y mecanismos codificados en los agentes del modelo coinciden con sus análogos del mundo real. La Macrovalidación es el proceso de asegurar que las propiedades agregadas y emergentes del modelo corresponden a las propiedades agregadas en el mundo real. El segundo eje de validación es el nivel de detalle del proceso de validación. La validación facial (face-validation) es el proceso de mostrar que los mecanismos y propiedades del modelo parecen mecanismos y propiedades del mundo real. La validación empírica asegura que el modelo genera datos que pueden demostrarse que corresponden a patrones similares de datos en el mundo real. Para ayudar a ilustrar el proceso de validación de un MOBA, utilizaremos el modelo Flocking de la sección de Biología de la biblioteca de modelos NetLogo. Este modelo intenta recrear patrones de aves que acuden como existen en la naturaleza :

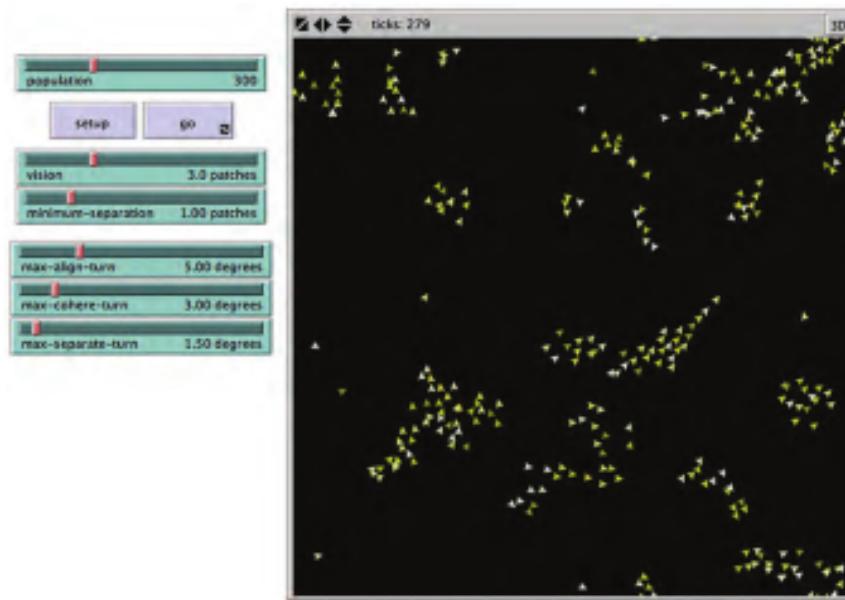


Figure 7.7
NetLogo Flocking model. <http://ccl.northwestern.edu/netlogo/models/Flocking> (Wilensky, 1998a).

Un modelo clásico basado en agentes basado en los modelos originales de Boids de Reynolds (1987). El modelo demuestra que pueden surgir bandadas de pájaros sin ser guiados de ninguna manera por Aves líderes especiales. Por el contrario, cada ave sigue exactamente el mismo conjunto de reglas, y de ahí las bandadas emergen. Cada ave sigue tres reglas: “alineación”, “separación” y “coherencia” * Alineación: significa que un pájaro gira para que se mueva en la misma dirección que cerca aves.

- Separación: significa que un pájaro gira para evitar golpear a otro pájaro.
- Cohesión: significa que un pájaro se mueve hacia otras aves cercanas.

La regla de “separación” anula las otras dos, lo que significa que si dos pájaros se acercan, siempre se separarán. En este caso, las otras dos reglas se desactivan hasta que se logra la separación mínima. las tres reglas afectan solo el rumbo del ave. Cada ave siempre avanza al mismo tiempo y con velocidad constante. Las reglas son notablemente robustas y se pueden adaptar a enjambres de insectos, colonias de peces (Stonedahl y Wilensky, 2010a). Describamos brevemente la implementación de estas tres reglas. La regla de alineación es codificada de la siguiente manera

```
to align
;; turtle procedure
turn-towards average-flockmate-heading max-align-turn
```

```
end
```

Este código le dice al pájaro que gire hacia el rumbo promedio de sus compañeros de manada, pero no más que el control deslizante MAX-ALIGN-TURN, que especifica el ángulo máximo que puede girar un pájaro con el fin de alinearse con sus compañeros de rebaño. Este código requiere dos procedimientos auxiliares: uno para encontrar los compañeros de bandada de un pájaro y otro para encontrar su rumbo promedio. El primero es sencillo:

```
to find-flockmates ;; turtle procedure
set flockmates other turtles in-radius vision
end
```

10.10 Replicación

Uno de los componentes fundamentales del método científico es la idea de replicación.(Latour y Woolgar, 1979). Desde este punto de vista, para que un experimento sea considerado aceptable por la comunidad científica, los científicos que originalmente realizaron el experimento deben publicar los detalles de cómo se realizó el experimento. Esta descripción puede ayudar a los equipos de científicos posteriores a realizar el experimento ellos mismos para determinar si sus resultados son lo suficientemente similares como para confirmar los resultados originales. Ese proceso confirma el hecho de que el experimento no dependía de ninguna condición local, mientras que la descripción escrita del experimento es lo suficientemente satisfactoria como para registrar el conocimiento y la ventaja obtenida en el registro permanente. La replicación es una parte importante del proceso científico y es tan importante dentro del ámbito de los modelos computacionales como lo es dentro del ámbito de los experimentos físicos. La reproducción de un experimento físico ayuda a demostrar que los resultados originales no se deben a errores o descuidos en la ejecución probando y comparando tanto la configuración experimental como los resultados posteriores. La replicación de un modelo computacional sirve para este mismo propósito. Adicionalmente, replicar un modelo computacional aumenta nuestra confianza en la verificación del modelo ya que una nueva implementación del modelo conceptual ha arrojado los mismos resultados que el original. La implementación del modelo replicado debe diferir de alguna manera del modelo original y también debe ser ejecutable (en lugar de otro modelo conceptual formal). Un modelo original y un modelo replicado asociado pueden diferir en al menos seis dimensiones: (1) tiempo, (2) hardware, (3) idiomas, (4) juegos de herramientas, (5) algoritmos y (6) autores.

Esta lista está ordenada según la probabilidad de que el esfuerzo de replicación produzca resultados diferentes del modelo original. Por lo general, más de una de estas dimensiones son consideradas en el curso de un modelo de replicación. Un modelo puede ser replicado por el mismo individuo en el mismo hardware

y en el mismo kit de herramientas y entorno de lenguaje pero reescrito en un momento diferente. En este cambio es menos probable que se produzcan resultados significativamente diferentes pero, si lo hiciera, lo haría indicar que la especificación publicada es inadecuada, ya que incluso el investigador original no se pudo volver a crear el modelo a partir del modelo conceptual original. Este es la única dimensión de replicación que siempre será variada. El modelo puede ser replicado por el mismo individuo pero en diferente hardware, por un cambio en el hardware queremos decir que el modelo implementado puede correr en una máquina diferente. Los cambios de hardware también se pueden obtener replicando el modelo en una plataforma de hardware diferente. El modelo podría replicarse en un lenguaje de computadora diferente. Por un lenguaje informático, nos referimos al lenguaje de programación que se utilizó para codificar las instrucciones en el modelo implementado. Java, Fortran, Objective-C y NetLogo son ejemplos de diferentes lenguajes, a menudo, la sintaxis y la semántica de un lenguaje tienen un efecto significativo en cómo el investigador traduce el modelo conceptual en una implementación real. Por lo tanto, la replicación en un nuevo lenguaje puede destacar las diferencias entre el modelo conceptual y la implementación. Incluso apareciendo detalles menores en lenguaje y especificaciones algorítmicas, como los detalles de punto flotante o las diferencias entre implementaciones de protocolos, pueden causar diferencias en los modelos replicados (Izquierdo y Gotts, 2005, 2006). Para que un modelo sea ampliamente aceptado como parte de la investigación científica, debe ser robusto a tales cambios. NetLogo es un ejemplo interesante porque, si bien el software de NetLogo está escrito en una combinación de Java y Scala, los modeladores usan otro lenguaje (también referido como NetLogo) para desarrollar modelos. Por lo tanto, clasificamos NetLogo como un kit de herramientas y un lenguaje. Con muchos kits de herramientas de modelado diferentes disponibles para su uso, los resultados de la replicación. pueden iluminar problemas no solo conceptuales sino también en el mismo kit de herramientas los propios, el modelo podría replicarse usando diferentes algoritmos. Por ejemplo, hay muchas formas de implementar algoritmos de búsqueda (por ejemplo, primero en amplitud, primero en profundidad) o actualizar una gran cantidad de objetos (por ejemplo, en orden de creación de objetos, en un orden aleatorio). De hecho, un modelo replicado puede simplemente realizar los pasos de un modelo en un orden diferente al del modelo original. Todas estas diferencias pueden potencialmente crear disparidades en los resultados. Por otra parte, también es posible que la descripción algorítmica difiera, pero que los resultados no lo hagan. Esto podría suceder porque dos individuos describen el mismo algoritmo de manera diferente, o que las diferencias algorítmicas no afectan los resultados. Esto es una prueba sólida de la replicabilidad de un modelo. Si otro investigador puede tomar una descripción formal del modelo y recrearlo para producir los mismos resultados, tenemos evidencia razonable de que el modelo se describe con precisión y los resultados son robustos a los cambios.

Part VI

Un Modelo Económico

Chapter 11

Modelo de Inversión

11.1 Descripción del Modelo

11.1.1 Introducción

Este modelo simula cómo las personas deciden en qué negocio invertir cuando las alternativas de inversión que se eligen difieren en sus ganancias anuales (profit) y en su riesgo de fracaso (risk). Estas alternativas comerciales (podían pensarse que son bancos) se van a modelar como parcelas.(Observe que este es un uso distinto a representar parcelas como espacios geográficos).

En esta primera versión, los inversores (agentes) perciben la información financiera de manera local, sólo de las parcelas vecinas (en posteriores versiones ampliaremos el “alcance” de los agentes).

11.1.2 Descripción del Modelo Odd Protocol

Especificaremos nuestro modelo de Negocio Simple usando el Protocolo ODD (RailsBack, Grimm), protocolo muy usado para especificar o definir MOBAs. La siguiente es la descripción ODD del modelo:

11.1.3 Objetivo del Modelo

El objetivo principal es explorar los efectos de la “Adquisición de Información” (qué información tienen los agentes y cómo la obtienen), y como se toman decisiones y se “adaptan” estas al utilizar la información adquirida. El modelo tiene que ver con decisiones de inversión, pero no pretende representar ningún enfoque de inversión real o sector empresarial, se podría considerar que este modelo representa aproximadamente a personas que compran y operan en negocios

“locales”, en este caso los inversores están familiarizados con oportunidades de inversión dentro de un rango limitado (las parcelas vecinas) y asumiremos que seleccionar una alternativa de decisión de inversión no acarrea ningún costo (podría asemejarse por ejemplo a cambiar de banco).

11.1.4 Tipos de Agentes, Variables y escalas

Los agentes en este modelo son;

Agentes inversores (tortugas) y Alternativas comerciales ó bancos (parcelas)

Las alternativas comerciales reciben dinero de sus clientes y varían en la *ganancia* que brindan a sus clientes y en el *riesgo* que tiene esa inversión, para ser específicos El Mundo es una cuadrícula de parcelas comerciales, cada uno de los cuales tiene dos variables:

- La ganancia anual que se proporciona allí (P , en unidades monetarias como dólares por año).
- el riesgo anual de que el negocio falle y el inversor pierda toda su riqueza (F , como probabilidad anual).

Los inversores tienen una riqueza actual (W , en unidades monetarias).

11.1.5 Escalas Espacio Temporales

- Escala de Espacio: El Mundo tiene un tamaño de 19×19 parcelas y tiene bordes.
- Escala de tiempo : los ticks corresponden a un año, y las simulaciones del modelo se realizaran por 50 años.

11.1.6 Procesos y ordenación temporal (scheduling)

El modelo incluye las siguientes acciones que se realizarán en el siguiente orden en cada tick:

- Repositionamiento: Los inversores deciden si un negocio similar (parcela adyacente) les ofrece una mejor opción que la actual; si es así, “repositionan” y transfieren su inversión a esa parcela o banco, moviéndose allí. **Solo un inversor puede ocupar un parcela a la vez.**
- Contabilidad. Los inversores actualizan sus variables de riqueza, esta se calcula incrementando la riqueza actual más el beneficio anual que le aporta la parcela (“banco”) a donde se desplazó.

(Nota: El fracaso inesperado del negocio también está incluido en la acción contable. La variable F en las parcelas miden el riesgo de quiebra, en este caso el negocio falla y la riqueza W del inversor queda en cero, sin embargo el inversor permanece en el modelo)

11.1.7 Conceptos de diseño

- Principios básicos. El tema básico de este modelo es cómo los agentes toman decisiones que involucran compensaciones (trade-offs) entre varios objetivos: en este caso aumentar las ganancias y disminuir el riesgo.
- Emergencia: El resultado importante que se quiere mirar del modelo es que riqueza promedio tendrán los inversores luego de 25 años, tiempo en el cual se correrá el modelo

(Nota: Otros tipos de resultado podrían ser la máxima ganancia obtenida ó el número de inversores que han sufrido un fracaso. Estos resultados surgen de cómo los inversores individuales balancean (trade-off) sus beneficios y el riesgo involucrado, pero también del “clima de negocios”)

- Comportamiento adaptativo. Este es la decisión de a qué parcela vecina deben moverse (o quedarse en su parcela), considerando el beneficio y el riesgo de las diferentes alternativas. En esta versión del modelo, los inversores utilizan un sistema simplificado de análisis microeconómico para tomar su decisión, moviéndose a la parcela que proporciona el valor más alto de una función objetivo, que se describe a continuación.
- Objetivo: En economía, el término “utilidad” se utiliza como sinónimo del objetivo que buscan los agentes, los inversores califican las alternativas mediante una medida de utilidad que represente su riqueza futura esperada al final de un horizonte temporal (T), en este caso se usarán “5 años”. Esta riqueza futura es una función de la riqueza actual y del riesgo que ofrece la alternativa. La función objetivo es la siguiente:

$$U = (W + T \cdot P)(1 - F)^T$$

P: ganancia anual.

F : probabilidad de riesgo.

W : riqueza del agente.

T: Horizonte (5 años).

- Detección(Sensing): Los agentes inversores conocen la ganancia y el riesgo en la parcela donde se encuentra y en las parcelas vecinas adyacentes **sin error**.
- Interacción. Los inversores interactúan entre sí indirectamente a través de la “competencia” por las parcelas:

1. Un inversor no puede hacerse cargo de un negocio (pasar a una parcela) si está ya ocupada por otro inversor.
2. Los inversores ejecutan su acción de reposicionamiento en orden aleatorio, por lo que no hay preferencias (por ejemplo los inversores con mayor riqueza no tienen ventaja sobre otros)
- Aleatoriedad (Estocasticidad) : El estado inicial del modelo es estocástico: los valores de P y F de cada parcela y las ubicaciones iniciales de los inversores, se establecen al azar (ver más adelante). No hay correlación entre ganancia y riesgo (P y F), son independientes.

11.1.8 Inicialización del modelo

Se utilizan cuatro parámetros del modelo para inicializar este mundo inversionista. Estos definen:

- Los valores de P Distribución Exponencial con media 5000.
- Los valores de F, Dstribución Uniforme (desde 0.01 y 0.1).
- 100 agentes inversores se colocan en parcelas al azar en el mundo, evitando que dos invesores queden en una misma parcela. Las variable de riqueza W de cada inversor se coloca en cero.

11.1.9 Datos de entrada

No se utilizan datos de entrada.

11.1.10 Submodelos

- Reposicionamiento: Un agente identifica todas las empresas en las que podría invertir, las parcelas vecinas (ocho o menos si el agente está en el borde del mundo) que están desocupados, más su parcela actual. El agente determina cuál de estas alternativas proporciona el valor más alto de función de utilidad,y se mueve a esta parcela (o permanece en su parcela si no hay una mejor utilidad en las parcelas vecinas).

*Contabilidad. Esta acción se describió en un apartado anterior (Procesos y ordenación temporal (scheduling)).

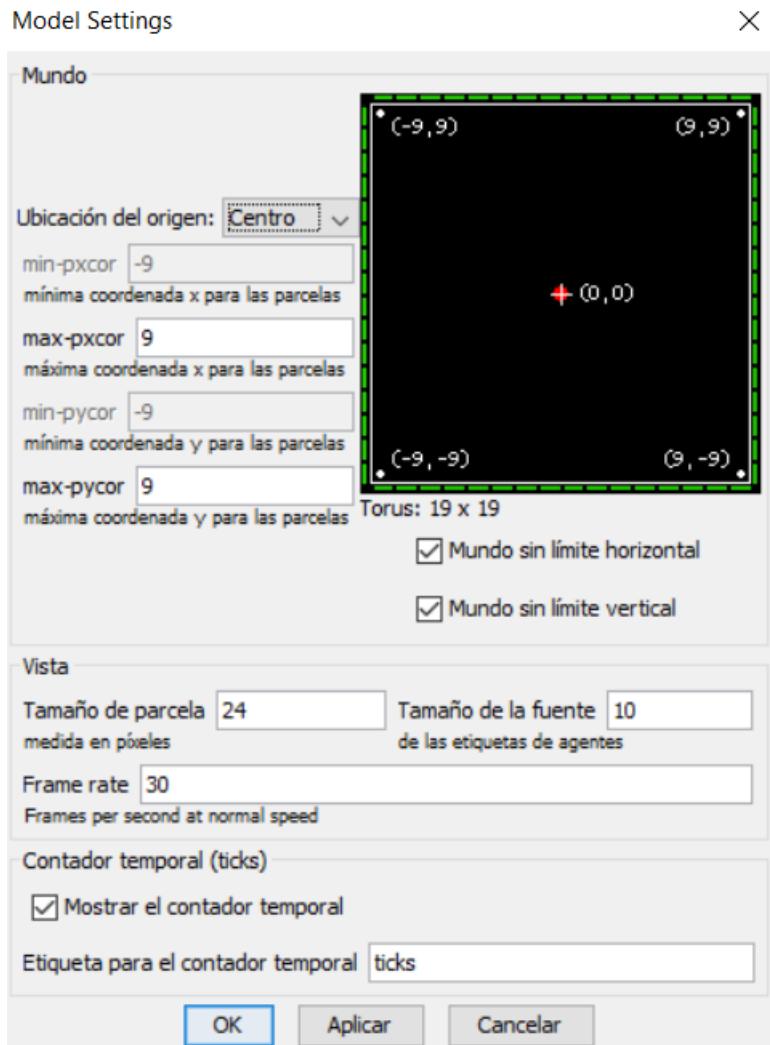
11.2 Construcción del Modelo

Abra Netlogo y Guarde el Archivo vacío como ModeloNegocio.nlogo

11.2.1 Configuración del Mundo:

Definiremos el mundo del Modelo de Negocio como un mundo e 19x19 parcelas conmlímites tanto horizontales como verticales

19 x 19 con límites horizontales y verticales



11.2.2 Procedimiento Setup

1. Modelación de las parcelas (Bancos)

Cada parcela(banco) tendrá dos atributos asociados:

- la ganancia anual que ofrece esta parcela para los inversores
- El riesgo anual que tiene la inversión

```
patches-own
[
  profit      ; ganancia anual que da cada parcela (banco)
  annual-risk ; ganancia anual que tiene cada parcela (banco)
]
```

Oprimamos botón comprobar

El procedimiento setup:

- limpia el mundo clear-all
- Configura las parcelas (bancos): procedimiento setup-patches.
- cronómetros en 0 : reset-ticks

```
to setup
  clear-all
  ask patches [setup-patches]
  reset-ticks
end

to setup-patches
end
```

setup-patches (profit annual-risk):

El procedimiento setup-patches:

- Inicializa la ganancia anual de cada parcela (aleatorio entre 1000 y 10000)
- inicializa el riesgo de cada parcela (probabilidad entre 0.01 y 0.1)

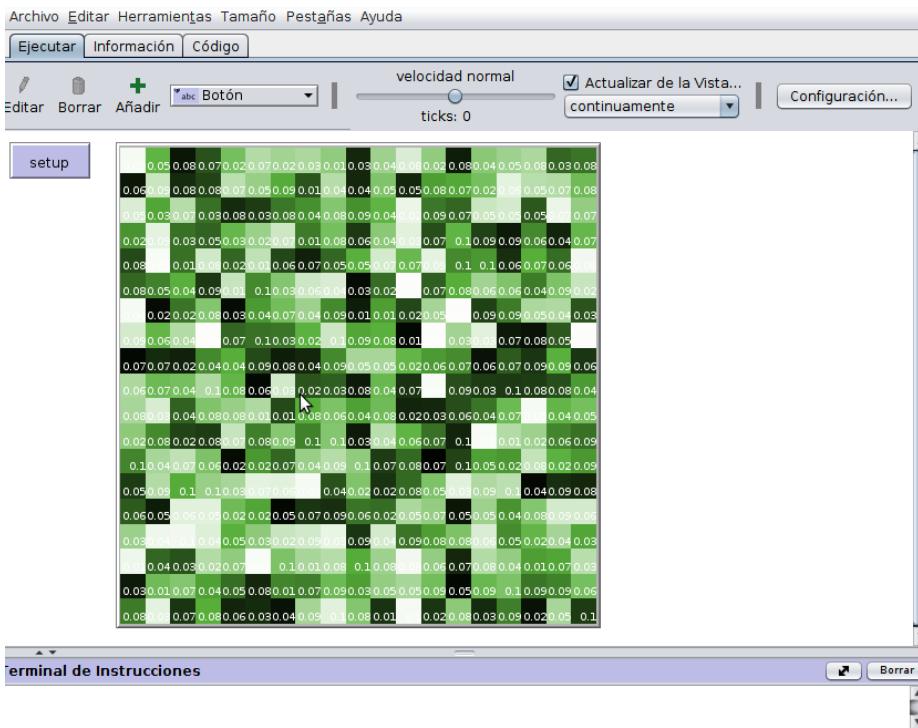
```
to setup-patches
  set profit random-exponential 5000
  set risk 0.01 + random-float ( 0.1 - 0.01 )
end
```

Para una mejor visualización del modelo cada parcela mostrará:

- el riesgo como un número
- la utilidad de color verde (entre más claro el verde, mayor la utilidad)

Estas dos líneas se añaden a setup-patches

```
set plabel (precision risk 2)
set pcolor scale-color green profit 1000 10000
```



2. Modelación de los agentes (Inversores)

Añadir al setup la linea:

```
setup-inversores
```

y colocar debajo del setup:

```
to setup-inversores
end
```

Inicialmente se usaran 25 inversores, cada uno con un monto de dinero (wealth) inicialmente en 0:

Crear la variable global num-investors y la propiedad de riqueza (wealth) de los agentes, colocar en la parte superior de la pestaña de código lo siguiente:

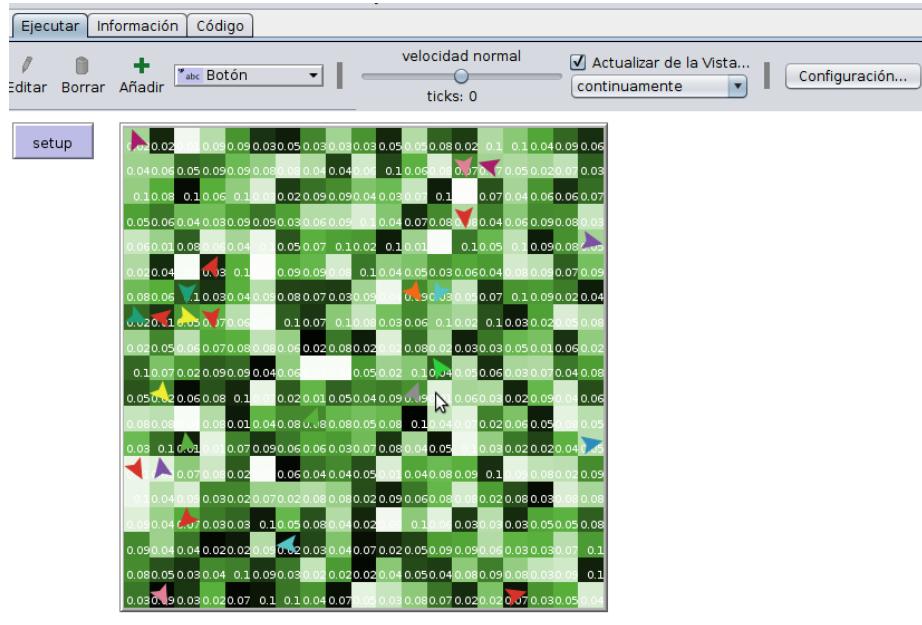
```
globals
[
  num-investors ; variable global número de inversores
]
turtles-own
[
  wealth ; riqueza de l agente inicialmente en cero
]
```

Definir dentro de setup el número de inversores:

```
set num-investors 25
```

y ahora colocar dentro de setup-inversores lo siguiente:

```
create-turtles num-investors
[
  move-to one-of patches with [not any? turtles-here] ; ocupen una parcela no ocupada
  set wealth 0.0 ; riqueza en cero
  pen-down ; activar rastro de agentes para observar su trayectoria
]
```



Este es el procedimiento setup total:

```
globals
[
  num-investors ; variable global número de inversores
]

turtles-own
[
  wealth ; riqueza de l agente inicialmente en cero
]
patches-own
[
  profit ; ganancia anual que da cada parcela (banco)
  risk ; ganancia anual que tiene cada parcela (banco)
```

```

    ]

to setup
  clear-all
  set num-investors 25
  ask patches [setup-patches]
  setup-inversores
  reset-ticks
end

to setup-patches
  set profit 1000 + random ( 10000 - 1000 ) ; profit entre 1000 y 10000
  set risk 0.01 + random-float ( 0.1 - 0.01 ) ; risk entre 0.01 y 0.1
  ; para visualización
  set plabel precision risk 2 ; visualizar numéricamente el riesgo en cada
  set pcolor scale-color green profit 1000 10000 ; visualizar por color la ganancia
  ; verde más claro más ganancia
end

to setup-inversores
  create-turtles num-investors
  [
    move-to one-of patches with [not any? turtles-here] ; ocupen una parcela no ocupada!!!!
    set wealth 0.0 ; riqueza en cero
    pen-down ; activar rastro de agentes para observar su trayectoria
  ]
end

```

11.2.3 Procedimiento Go

Lo primero es determinar cuando el modelo para, usaremos una variable global years-simulated:

```

globals
[
  num-investors ; variable global número de inversores
  years-simulated ; número de años en que el modelo se va a correr
]

```

inicializamos en el setup la variable years-simulated:

```
set years-simulated 50
```

y construimos la primera versión del procedimiento go:

```
to go
```

```

if ticks >= years-simulated [stop]
tick
end

```

Antes de continuar debemos acordarnos que el cálculo de la utilidad de un agente:

$$U = (W + T.P)(1 - F)^T$$

requiere de un “horizonte-de-tiempo” T, que nos dará el horizonte de tiempo en el que estamos considerando la inversión, entonces la definiremos como variable global:

```

globals
[
num-investors ; variable global número de inversores
years-simulated ; número de años en que el modelo se va a correr
time-horizon ; horizonte de tiempo para el calculo de la utilidad
]

```

el valor que le daremos a esta variable es 5 años, coloquemos en el setup la siguiente linea:

```
set time-horizon 5 ; horizonte de tiempo 5 años
```

Ahora si podemos continuar, El procedimiento go va a tener dos subprocedimientos:

1. reposition: determina cual parcela, no ocupada , vecina me proporciona mejor inversión para moverme a ella (en caso de que no exista esa mejor parcela me quedo quieto).
2. do-accounting: una vez me muevo actualizo mi dinero (wealth)

Para determinar que inversión es la mejor debo calcular para cada parcela vecina no ocupada la utilidad de la inversión, para ello definiremos un reportero (que usaran las parcelas) que hace esta cálculo dado un agente (que tiene determinada cantidad de dinero), el reportero es el siguiente:

```

to-report utilidad-para [un-inversor]
let riqueza-inversor [wealth] of un-inversor ; riqueza del inversor
let utilidad
  riqueza-inversor + (profit * time-horizon) * ((1 - risk) ^ time-horizon)
  ; aplica fórmula de utilidad al inversor
report utilidad ; reporta la utilidad
end

```

finalmente definimos las funciones repositioning y accounting de la siguiente manera:

Finalmente definamos las funciones : reposition y do-accounting.
coloquemos en el go lo siguiente, antes de tick:

```
ask turtles [reposition]
ask turtles [do-accounting]
```

y definamos estos dos procedimientos:

```
to reposition
  let potential-destinations neighbors with [not any? turtles-here]
  set potential-destinations (patch-set potential-destinations patch-here)
  let best-patch max-one-of potential-destinations [utility-for myself]
  move-to best-patch
end

to do-accounting
  set wealth (wealth + profit) ; Primero , añada ganancia anual
  if random-float 1.0 < risk ; Mire si la inversión quiebra
  [
    set wealth 0
  ]
end
```

El texto completo del go es el siguiente:

```
to go
  if ticks >= years-simulated [stop]
  ask turtles [reposition]
  ask turtles [do-accounting]
  tick
end

to reposition
  let potential-destinations neighbors with [not any? turtles-here]
  set potential-destinations (patch-set potential-destinations patch-here)
  let best-patch max-one-of potential-destinations [utilidad-para myself]
  move-to best-patch
end

to do-accounting
  set wealth (wealth + profit) ; Primero , añada ganancia anual
  if random-float 1.0 < risk ; Mire si la inversión quiebra
  [
    set wealth 0
  ]
end

to-report utilidad-para [un-inversor]
  let riqueza-inversor [wealth] of un-inversor ; riqueza del inversor
```

```

let utilidad
  riqueza-inversor + (profit * time-horizon) * ((1 - risk) ^ time-horizon)
  ; aplica fórmula de utilidad al inversor
  report utilidad    ; reporta la utilidad
end

```

El texto completo del modelo es el siguiente:

```

globals
[
  num-investors    ; variable global número de inversores
  years-simulated ; número de años en que el modelo se va a correr
  time-horizon    ; horizonte de tiempo paara el calculo de la utilidad
]

turtles-own
[
  wealth    ; riqueza de l agente inicialmente en cero
]
patches-own
[
  profit      ; ganancia anual que da cada parcela (banco)
  risk        ; ganancia anual que tiene cada parcela (banco)
]

to setup
  ca
  set num-investors 25
  set years-simulated 50
  set time-horizon 5
  ask patches [setup-patches]
  setup-inversores
  reset-ticks
end

to setup-inversores
  crt num-investors
  [
    move-to one-of patches with [not any? turtles-here]
    set wealth 0.0
    pen-down
  ]
end

to setup-patches

```

```

set profit random-exponential 5000
set risk 0.01 + random-float ( 0.1 - 0.01)
set plabel (precision risk 2)
set pcolor scale-color green profit 1000 10000
end

to go
  if ticks >= years-simulated [stop]
  ask turtles [reposition]
  ask turtles [do-accounting]
  tick
end

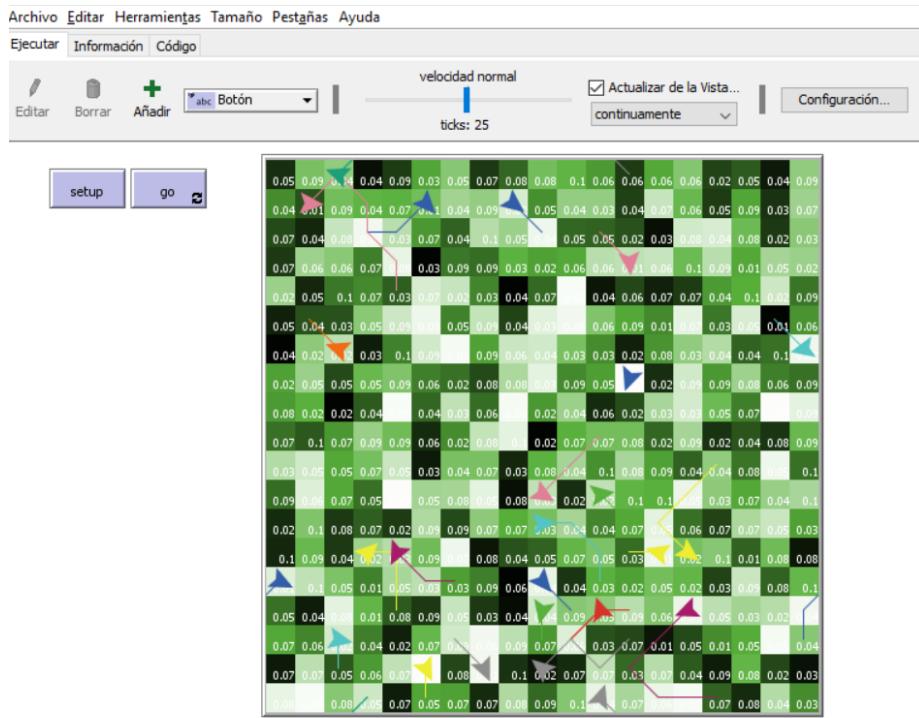
to reposition
  let potential-destinations neighbors with [not any? turtles-here]
set potential-destinations (patch-set potential-destinations patch-here)
  let best-patch max-one-of potential-destinations [utilidad-para myself]
  move-to best-patch
end

to do-accounting
  set wealth (wealth + profit) ; Primero , añada ganancia anual
  if random-float 1.0 < risk ; Mire si la inversión quiebra
  [
    set wealth 0
  ]
end

to-report utilidad-para [un-inversor]
  let riqueza-inversor [wealth] of un-inversor ; riqueza del inversor
  let utilidad
    riqueza-inversor + (profit * time-horizon) * ((1 - risk) ^ time-horizon)
    ; aplica fórmula de utilidad al inversor
  report utilidad ; reporta la utilidad
end

```

Finalmente coloquemos un botón go y corramos el modelo:



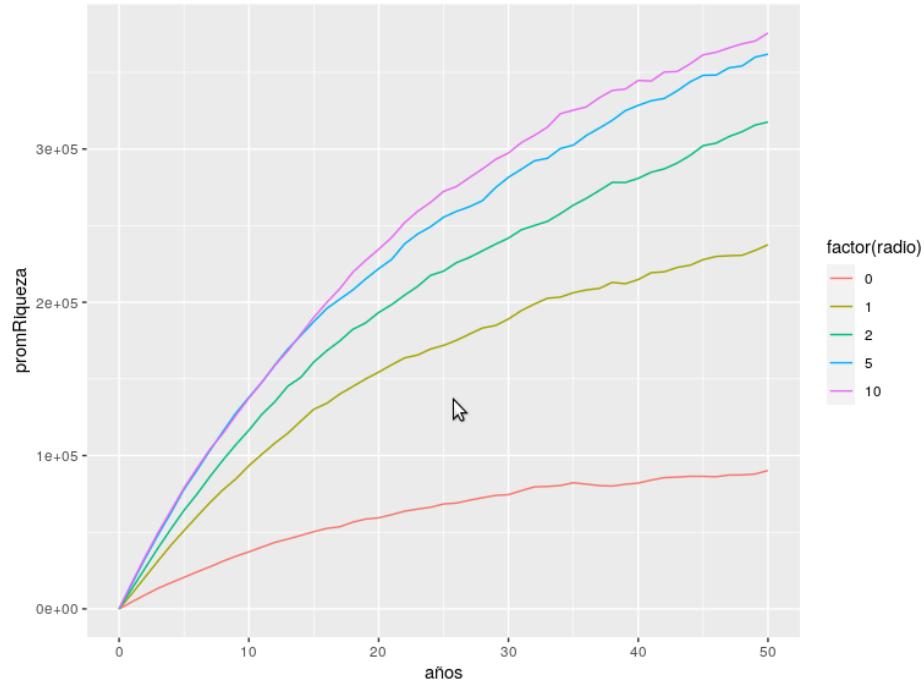
El modelo lo puede correr en el siguiente applet:

```
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

11.3 Análisis del Modelo

11.3.1 Análisis Básico

¿Tienen más éxito los inversores (promedio de riqueza) si pueden percibir más oportunidades de negocio en un radio más amplio que el de sus parcelas vecinas?. Podemos abordar esta cuestión con un experimento de simulación que varíe el radio sobre el cual los inversores pueden detectar parcelas (bancos). Para ampliar esta noción de vecindad, definiremos una variable (en forma de deslizador) que nos defina el radio que los inversores pueden detectar y entonces podemos usar El Analizador de Comportamiento para simular un experimento que varíe este parámetro para ver el efecto del radio sobre la riqueza media de los inversores. Usaremos el comando **in-radius** (puede ver lo que este comando hace en el diccionario de NetLogo in-radius) para crear un conjunto de parcelas dentro de un radio determinado. Si variamos el parámetro radio-detección, obtenemos resultados como los que se muestran en la figura:

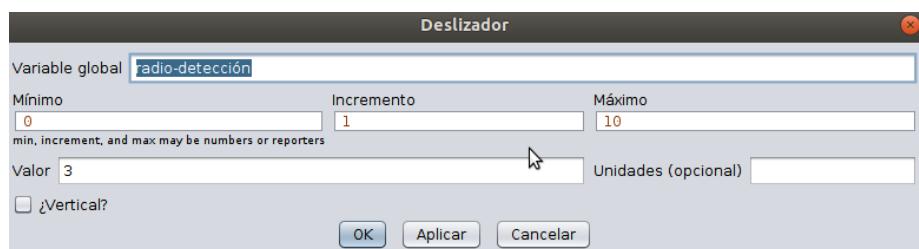


Estos resultados indican que ser capaz de detectar y ocupar oportunidades en una parcela es mucho mejor que no poder reposicionar en absoluto, pero que a medida que el radio de detección aumenta cada vez existe menos beneficio adicional.

- ¿Qué hace que la variación de la riqueza entre los inversores individuales aumente con el radio de acción pero no aumenta tanto como lo hace la riqueza media?
- ¿Cómo podemos construir este gráfica en RStudio?

11.3.2 Modificando el Modelo

Modifiquemos nuestro modelo básico de acuerdo a lo anterior , lo primero es definir una nueva variable global radio-detección, la definiremos a través de un deslizador:



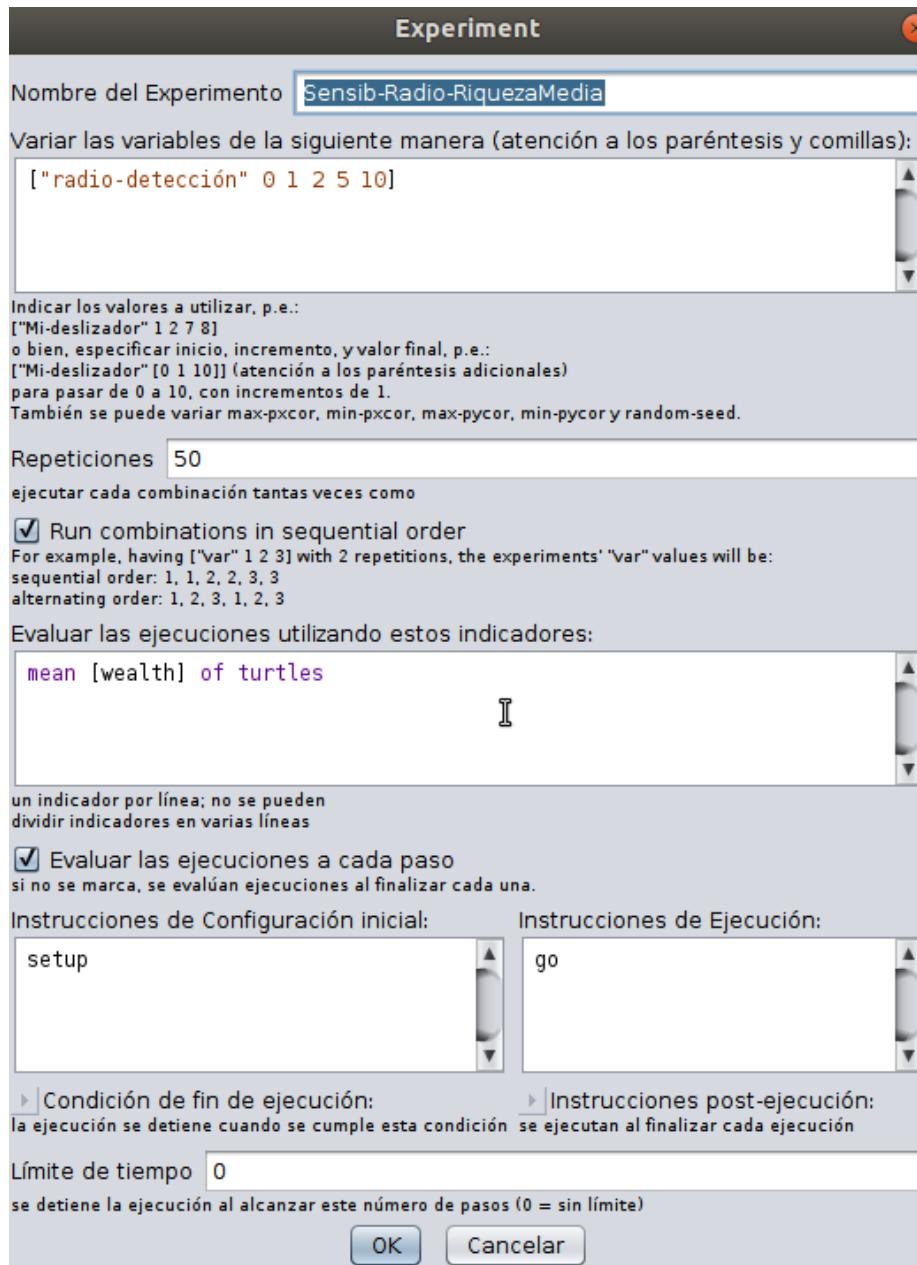
Ahora debemos cambiar el procedimiento go, para calcular las parcelas potenciales donde los inversores buscarán su mejor utilidad:

```
to reposition let potential-destinations patches in-radius radio-detección with  
[not any? turtles-here] ; ¡¡Nuevo!! posibles destinos dependen del radio de  
detección set potential-destinations (patch-set potential-destinations patch-here)  
let best-patch max-one-of potential-destinations [utilidad-para myself] move-to  
best-patch end
```

11.3.3 Generando el Experimento

Usaremos el Analizador de Comportamiento para definir el experimento que nos permitirá observar la sensibilidad de la utilidad de los inversores al radio de detección de las parcelas (BANCOS)

Abramos el analizador de Comportamiento de NetLogo y definamos el siguiente experimento:



Observe que:

- El nombre del Experimento es: "Sensib-Radio-RiquezaMedia"
- Los radios de detección toman 5 valores diferentes: 0,1,2,5,y 10
- Para cada radio de detección realizaremos 50 experimentos, o sea que en total son 250 experimentos

- Mediremos la riqueza media de los inversores (**mean [wealth] of turtles**)
- Mediremos la riqueza media de los agentes en cada paso (tick), ya que la opción “Evaluar las ejecuciones a cada paso” está activada.

(Nota: cada corrida del experimento terminará después de 50 iteraciones esto está especificado dentro del procedimiento Go de NetLogo y no hace parte de la definición del experimento, pero podríamos definirlo colocando en la ventana “Condición de fin de ejecución: **ticks >= years-simulated**”)

11.3.4 Análisis en R

Abramos R y definamos un Archivo Script, llamémolo “Sens-Radio-RiquezaMedia.R”, coloque este archivo en el mismo directorio donde se encuentre el Modelo NetLogo y el archivo de datos generado por el analizador de comportamiento.

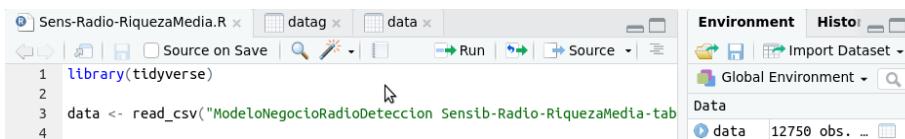
Defina como directorio de Trabajo en Rstudio el Directorio donde se encuentran estos tres archivos, usando el menu Session y la opción “To Source Fil Location” de la opción “Set Working Directory”

Coloque las dos primeras lineas del archivo Script la siguiente manera:

```
library(tidyverse)
```

```
data <- read_csv("ModeloNegocioRadioDeteccion Sensib-Radio-RiquezaMedia-table.csv", skip=6)
```

Debe aparecer en la parte derecha de Rstudio, el Archivo de Datos leido y con nombre “data”:



Renombre las columnas de la tabla de datos:

```
colnames(data) <- c("corrida", "radio", "años", "riqueza")
```

Agrupe los datos por año y radio y calcule los promedios de riqueza de los datos agrupados:

```
data %>% group_by(años, radio) %>%
  summarise(promRiqueza=mean(riqueza)) -> datag
```

La nueva agrupada y con la riqueza promediada (datag) debe verse de la siguiente manera:

Sens-Radio-RiquezaMedia.R x datag x dat

Filter

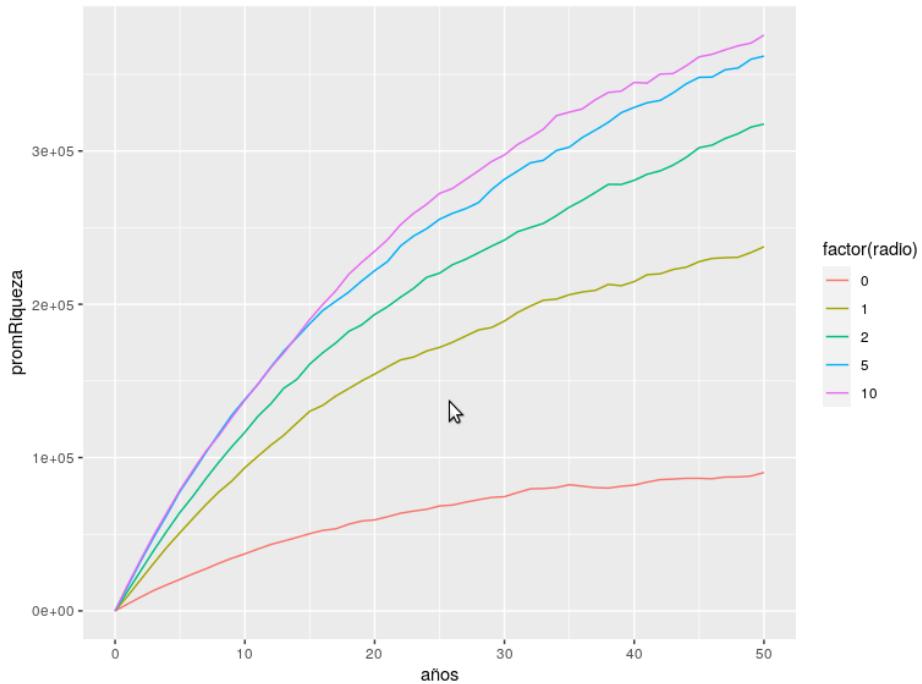
	paso	radio	prom
5	0	10	0.000
6	1	0	4657.103
7	1	1	10028.987
8	1	2	13318.366
9	1	5	16448.524
10	1	10	17465.537
11	2	0	9106.128
12	2	1	20708.212
13	2	2	26504.999
14	2	5	32860.912

Showing 5 to 15 of 255 entries, 3 total columns

Con esta tabla podemos graficar para cada valor del radio de detección como evoluciona la riqueza promedio de los inversores:

```
ggplot(datag)+  
  geom_line(aes(años,promRiqueza,color=factor(radio)))
```

La gráfica es la siguiente:



Se observa que:

- A mayor radio de detección mayor riqueza promedio de los inversores, pero..
- A medida que aumenta el radio el aumento de riqueza promedio es cada vez menor.

11.3.5 Un Estudio Bivariado

11.3.5.1 Introducción

Vamos a estudiar el efecto de dos variables claves del modelo que hasta ahora no hemos tomado en cuenta:

- El número de inversores
- El tamaño del mundo

¿Qué efecto tiene estas dos variables (de manera simultánea) sobre la riqueza de los inversores después de 25 años?

(El modelo básico lo puede bajar del siguiente enlace:

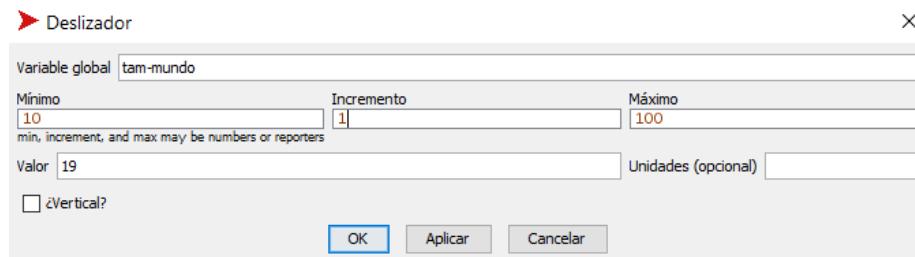
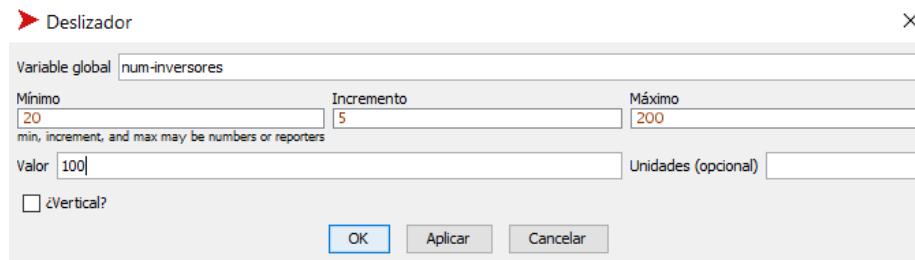
Enlace al modelo)

11.3.5.2 Modificando el Modelo

11.3.5.3 Deslizadores

Definamos dos deslizadores:

- num-inversores: para variar el número de inversores
- tam-mundo: para variar el tamaño del mundo

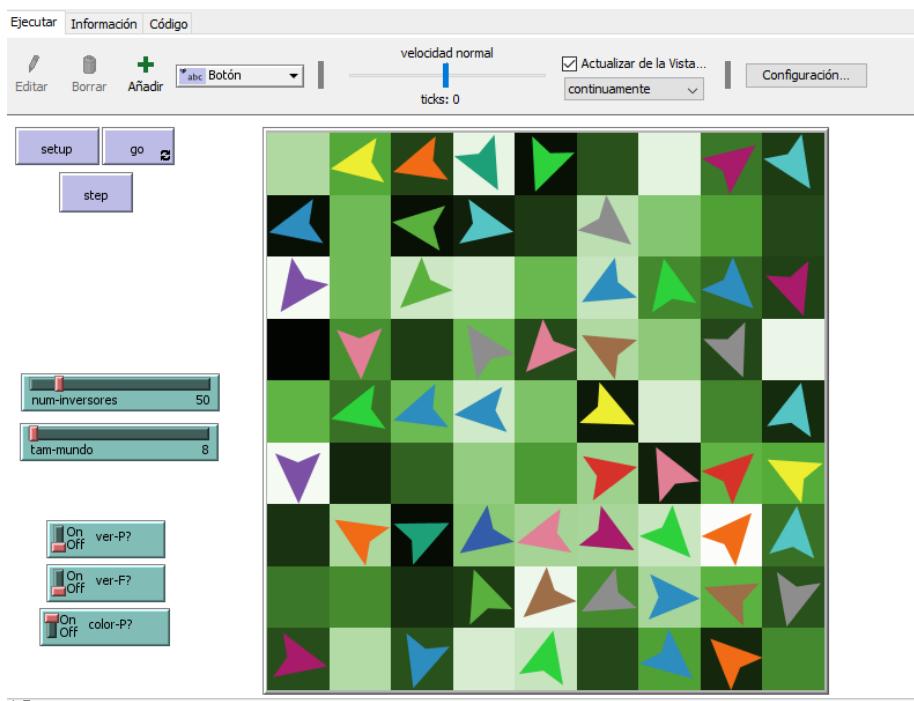


11.3.5.4 Procedimientos

Ahora modifiquemos el modelo, primero modifiquemos setup-inversores para incluir el deslizador:

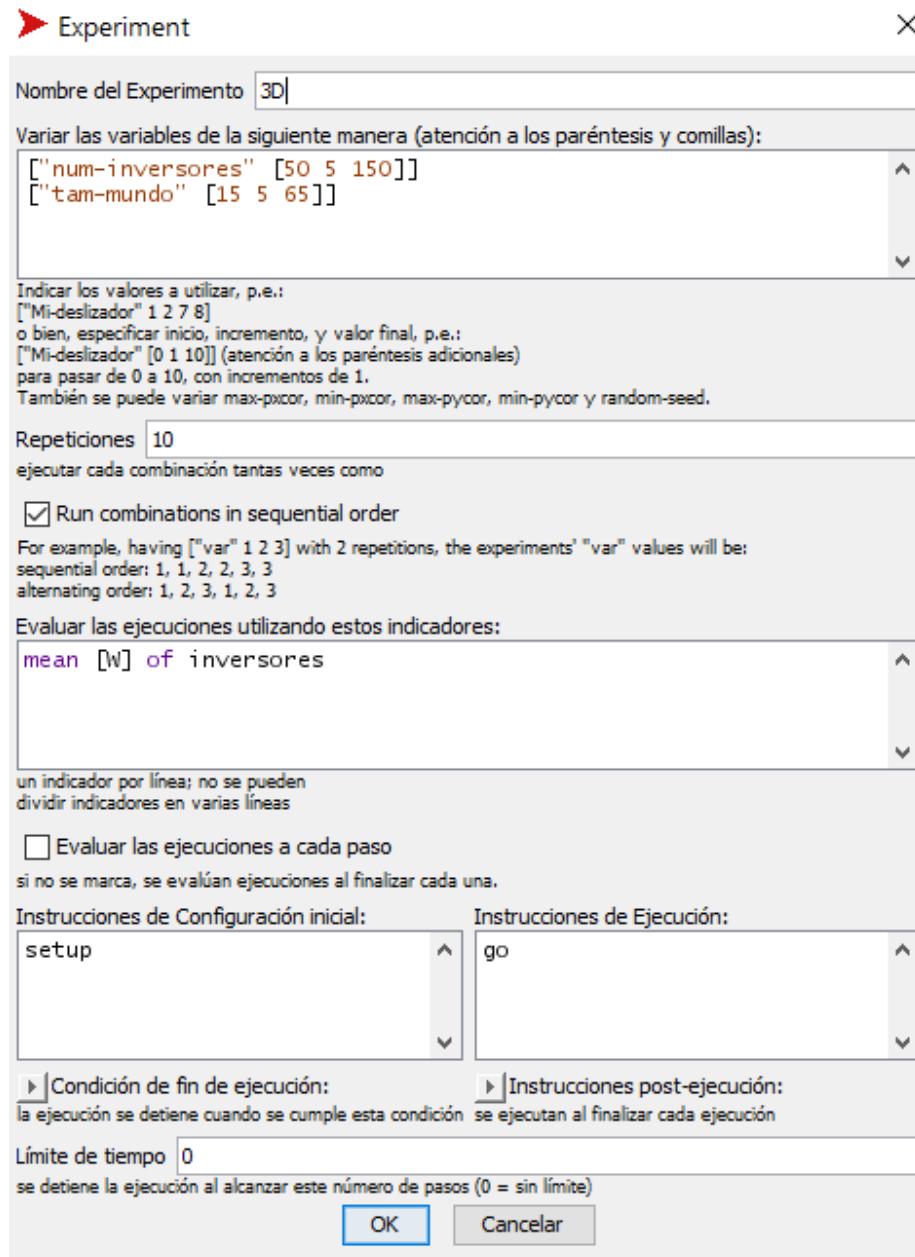
Ahora modifiquemos el setup, añadiendo un procedimiento setup-mundo para poder cambiar con el deslizador construido el tamaño del mundo

setup-mundo cambia tanto el tamaño del mundo (resize-world) como el de las parcelas (set-patch-size) de tal manera que el mundo se visualize correctamente. El modelo con un tamaño de 8 y 50 inversores sería el siguiente:



11.3.6 El Experimento

Creemos el experimento para poder analizar el efecto del número de inversores y el tamaño del mundo sobre la riqueza:



Solo variamos los deslizadores que definimos y desactivamos “Evaluar las ejecuciones a cada paso” ya que solo queremos observar resultados finales (a los 25 años). Corra el experimento y guarde los datos resultantes.

11.3.7 Rstudio

Importemos los datos a Rstudio

```
data <- read.csv("ModeloNegocioSimpleMundoVsInversores 3D-table.csv",skip=6)
head(data)

##   X.run.number. num.inversores tam.mundo X.step. mean..W..of.inversores
## 1             4          50        15       25      135951.9
## 2             3          50        15       25      128363.4
## 3             1          50        15       25      122975.7
## 4             6          50        15       25      124873.4
## 5             5          50        15       25      134091.1
## 6             7          50        15       25      163788.2
```

Seleccionemos las tres columnas que nos interesan:

```
library(tidyverse)
data <- data %>% select(num.inversores,tam.mundo,mean..W..of.inversores)
colnames(data)<- c("inversores","mundo","riqueza")
head(data)

##   inversores mundo riqueza
## 1          50    15 135951.9
## 2          50    15 128363.4
## 3          50    15 122975.7
## 4          50    15 124873.4
## 5          50    15 134091.1
## 6          50    15 163788.2
```

Agrupemos los datos por inversores y tamaño del mundo:

```
datag <- data %>% group_by(inversores,mundo) %>%
  summarise(riquezaP=mean(riqueza))
head(datag)

## # A tibble: 6 x 3
## # Groups:   inversores [1]
##   inversores mundo riquezaP
##       <int> <int>     <dbl>
## 1         50    15  135799.
## 2         50    20  141836.
## 3         50    25  139714.
## 4         50    30  155572.
## 5         50    35  147588.
## 6         50    40  157463.
```

Con los datos agrupados podemos construir una matriz con :

- filas(y) : tamaño del mundo

- columnas (x) : número de inversores
- valores matriciales (z): riqueza

Usaremos la librería plotly para visualizar la matriz en 2D:

```
library(reshape2)
library(plotly)
riqueza <- acast(datag, mundo ~ inversores,value.var="riquezaP")
ejey <- as.numeric(rownames(riqueza))
ejex <- as.numeric(colnames(riqueza))
fig <- plot_ly(
  x=ejex,y=ejey,z=riqueza,
  type = "contour"
)

fig

## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

Usaremos la libreria plotly para visualizar la matriz en 3D:

```
library(reshape2)
library(plotly)
riqueza <- acast(datag, mundo ~ inversores,value.var="riquezaP")

ejey <- as.numeric(rownames(riqueza))
```

```
ejex <- as.numeric(colnames(riqueza))

fig <- plot_ly() %>%
  add_surface(x=~ejex, y = ~ejey, z = ~riqueza) %>%
  layout(title="mundo-inversores vs riqueza",
         scene=list(
           xaxis=list(title="Inversores"),
           yaxis=list(title="mundo"),
           zaxis=list(title="riqueza"))

))

fig

## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

11.3.8 Preguntas

Estas gráficas (contour 3D plots) son muy útiles para analizar modelos basados en agnetes cuando queremos ver la influencia simultanea de dos diferentes variables sobre una tercera. A partir de esta gráfica se pueden responder muchas preguntas como:

- ¿Para que valores de tamaño del mundo e inversores la riqueza de estos es máxima?
- ¿Para que valores de tamaño del mundo e inversores la riqueza de estos es mínima?
- ¿En donde hay mayor y menor variación de la riqueza?

¿Qué otras preguntas se podrían formular? ¿Se le ocurren algunas?

Part VII

Proyectos Propuestos

Chapter 12

Proyectos Básicos

12.1 La Hormiga Atomica

Una condición que a veces produce un comportamiento complejo es el alto rendimiento. Este rendimiento puede ser el producto de ;

- Una gran cantidad de agentes que actúan e interactúan simultáneamente; una gran cantidad de algún material físico o propiedad (por ejemplo, carga eléctrica, fluido, moléculas de gas) que actúan en un espacio o medio restringido
- Un número mucho menor de agentes que repiten comportamientos simples una gran cantidad de veces en un espacio finito.

Tenemos un mundo de parcelas sin pintar (negro) y pintadas (blanco), habitadas por una o más hormigas que siguen una regla simple, en cada paso de tiempo (tick), cada hormiga realizará las siguientes acciones: Reglas:

1. La hormiga cambia el color de la celda donde se encuentre
2. Avanza una unidad de distancia a una nueva celda
3. a. Si la celda donde llega es blanca gira 90 grados a la derecha
 b. Si la celda donde llega es negra gira 90 grados a la izquierda

En el Mundo inicial todas las parcelas son blancas, una hormiga se coloca en el centro del mundo mirando hacia arriba.

Las primeras ocho iteraciones del modelo son las siguientes:

Uno:

Dos:

Tres:

Cuatro:

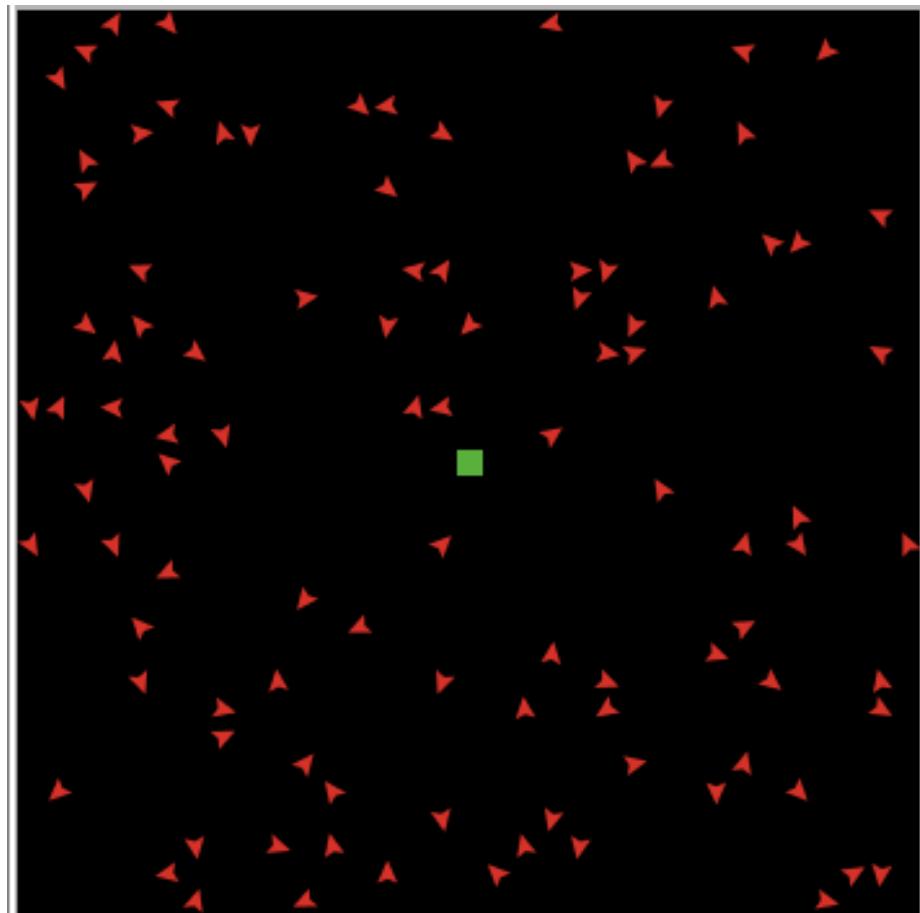
Cinco:

Seis:

Esta hormiga virtual se conoce como la hormiga de Langton, por su inventor Christopher Langton, pionero en el campo de la vida artificial. Con una sola hormiga situada en el centro del mundo, el comportamiento es el siguiente.:

12.2 Partículas

Tenemos una gran cantidad de partículas rojas (agentes) deambulando por el mundo y una parcela verde fija colocada en el centro del mundo.



Las partículas rojas se mueven de manera aleatoria hasta que:

- Encuentran que alguna parcela vecina es verde

En este caso la partícula muere, pero antes deja su huella en el mundo, convirtiendo la parcela donde se encuentra en verde. Si se implementa el modelo y se corre durante un tiempo suficiente, se obtendrá un patrón interesante.

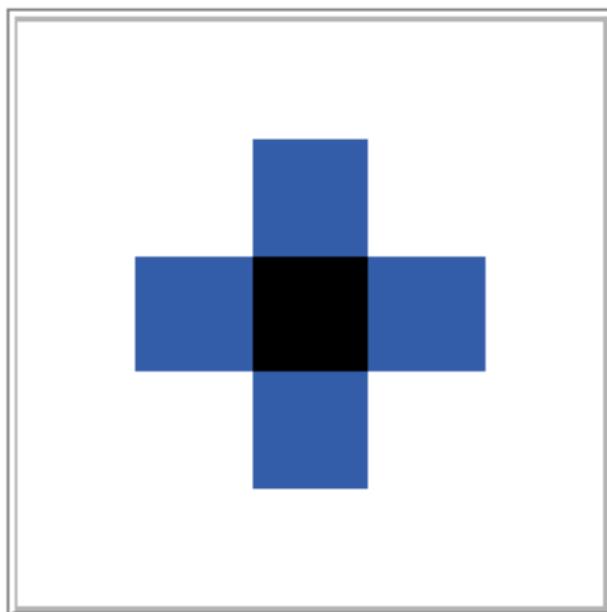
12.3 El modelo de paridad

Un modelo de cierta importancia para modelar sistemas físicos es el modelo de paridad. El modelo de paridad solo tiene una regla para actualizar el estado de una celda:

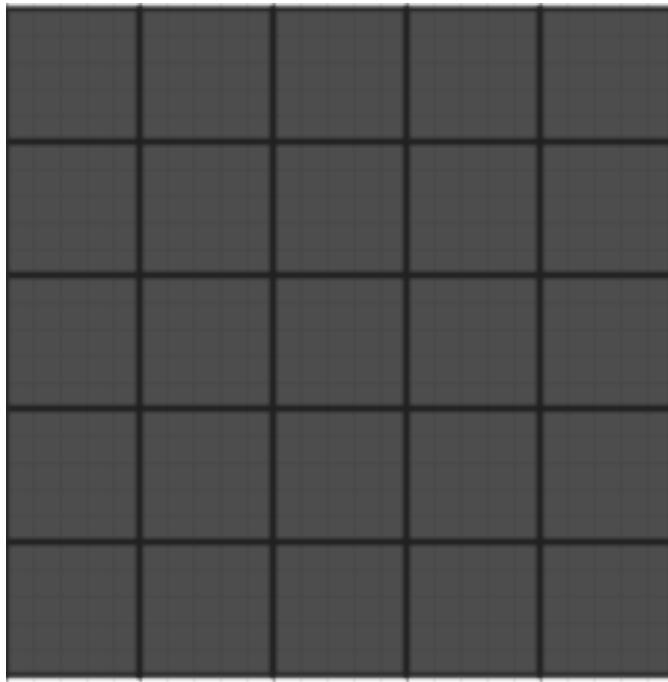
- una celda se vuelve “viva” (color negro) o “muerta” (color blanco) dependiendo de si la suma del número de células vivas vecinas es par ó impar.

(Nota: Este modelo usa solo cuatro celdas vecinas, las del norte, este, sur y oeste, este se conoce como el vecindario de von Neumann y se muestra en Figura)

Vecinos en azul



La configuración inicial que usaremos será un bloque cuadrado de cinco por cinco de células vivas situadas en el centro del mundo.



12.4 El Modelo del Chisme (Gossip Model)

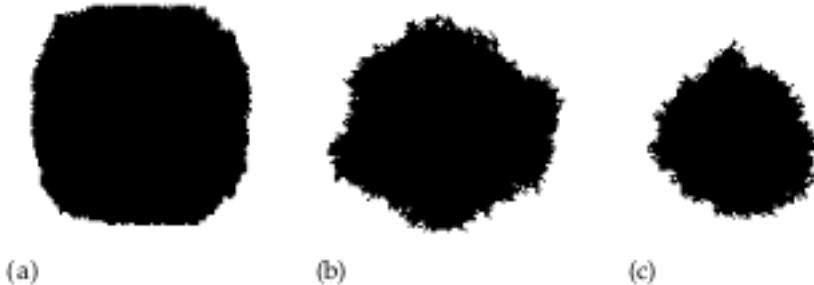
Con frecuencia, los individuos se modelan como células y la interacción entre individuos se modela utilizando reglas para estas celdas. Por ejemplo, se puede modelar la difusión de conocimientos, innovaciones o actitudes de esta manera. Considere por ejemplo, la propagación de un chisme que genera una sola persona a una audiencia interesada. Cada individuo se entera del chisme si lo escucha de un vecino quien ya ha escuchado la noticia y luego puede pasársela a algún vecino suyo (pero si no ve a su vecino ese día, no tendrá la oportunidad de difundirlo). Una vez alguien escucha el chisme, él o ella lo recuerda y no necesita escucharlo nuevamente. Este escenario se puede modelar muy fácilmente. Cada celda del modelo tiene dos estados: ignorancia sobre el tema del chisme (el equivalente de lo que en el discusión anterior que hemos llamado una celda “muerta”) o conocer el chisme (el equivalente a estar “vivo”). Colorearemos de blanco una celda que no conoce el chisme y de negro una celda que sí lo conoce. Una celda cambia de estado blanco a negro cuando uno de sus cuatro vecinos von Neumann conoce el chisme (y entonces pasa a color negro) y a partir de este momento puede difundirlo. Hay una probabilidad constante de que en un instante e tiempo (tick), una celda blanca reciba el chisme de un vecino negro y se convierta en negra. Una vez que una célula ha escuchado el chismes, nunca lo olvida, así que una celda negra nunca vuelve a ser blanca. Las reglas de cambio de estado son

las siguientes:

1. Si la celda es blanca y tiene uno o más vecinos negros cambie a negro con alguna probabilidad específica, de lo contrario quedese en blanco.
2. Si la celda es negra, la celda permanece negra.

Todas las reglas que hemos mencionado anteriormente han sido deterministas. Es decir, dada la misma situación, los resultados de la regla siempre serán las mismas. Sin embargo, el modelo de chismes es estocástico: solo existe la posibilidad de que una celda escuchará los chismes de un vecino. Podemos simular lo estocástico con un generador de números aleatorios. Supongamos que el generador produce un flujo aleatorio de números enteros entre 0 y 99. Una probabilidad del 50 por ciento de transmitir chismes puede simularse implementando la primera regla de la siguiente manera:

1. Si la celda es blanca, y tiene vecinos negros obtenga un número del generador de números aleatorios. Si este número es menor que 50, cambia el estado de blanco a negro.



La Figura a muestra la simulación cuando la fuente del chisme es una sola persona y utilizando una probabilidad del 50 por ciento de transmitir chismes. El chisme se propaga aproximadamente igualmente en todas las direcciones. Porque solo hay una probabilidad de transmitir el chisme, el área formada por las células negras no es un círculo perfecto sino desviaciones de una forma circular que tiende a alisarse con el tiempo. Con este modelo, podemos investigar fácilmente el efecto de diferentes probabilidades de comunicar los chismes haciendo un cambio apropiado en las reglas La figura (b) muestra el resultado del uso de una probabilidad del 5% (la primera regla se reescribe para que una celda solo cambie a negro si la probabilidad es menor que 5%, en lugar de 50%). Sorprendentemente, el cambio hace poca diferencia. La forma de las celdas negras es un poco más irregular y claro el chnisme viaja más lentamente porque la probabilidad de transmisión es mucho más baja (La figura 7.7 (b) requirió aproximadamente 250 pasos de tiempo, en comparación con 50 pasos para la Figura 7.7 (a)).

Sin embargo, incluso con esta probabilidad bastante baja de transmisión, se difunde el chisme. Podemos bajar aún más: La Figura 7.7 (c) muestra el

resultado de una probabilidad de transmisión del 1%. La forma de las celdas negras siguen siendo similares a las dos simulaciones anteriores, aunque la velocidad de transmisión es aún más lenta (la figura muestra la situación después de 600 pasos de tiempo). El modelo demuestra que la propagación de chismes (ó “Noticias” como innovaciones tecnológicas o incluso infecciones transmitidas por contacto) a través de interacciones locales de persona a persona no son impedidos, incluso teniendo una baja probabilidad de transmisión , aunque bajas probabilidades hacen más lenta su difusión.

Chapter 13

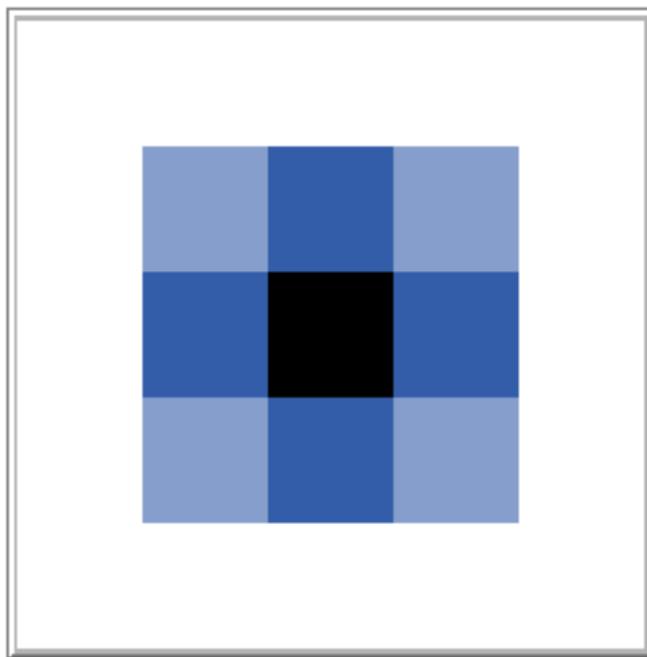
Proyectos Intermedios

13.1 Majority model

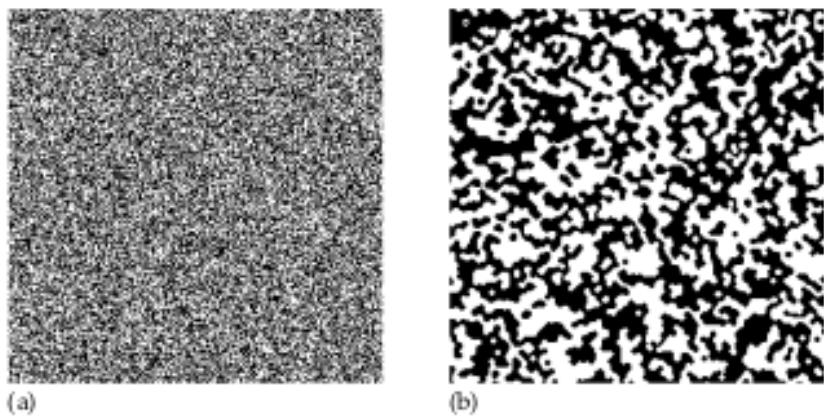
En el modelo de chismes, una celda se vuelve negra si escucha el chisme de uno de sus vecinos, por lo tanto este es un modelo de interacción persona a persona. Ahora consideremos un modelo en el que una célula cambia de estado de acuerdo con la información de todos sus vecinos. Por ejemplo, las personas pueden adoptar una moda solo si la mayoría de sus amigos ya lo han adoptado. Una vez más, la simulación consistirá en celdas cada una de las cuales tiene dos estados: blanco y negro. El modelo más simple tiene una sola regla:

1. El nuevo estado de una celda es el estado de la mayoría de los vecinos de Moore de esa celda (o el estado anterior de la celda si el número de vecinos blancos es el mismo) hay ocho vecinos de Moore.

8 Vecinos (los Azules)



Por lo tanto, la regla dice que una celda pasa a blanca si hay cinco o más celdas blancas que la rodean, y pasa a negra si hay cinco o más celdas negras a su alrededor, o permanece en su estado anterior si hay cuatro blancos y cuatro negros.

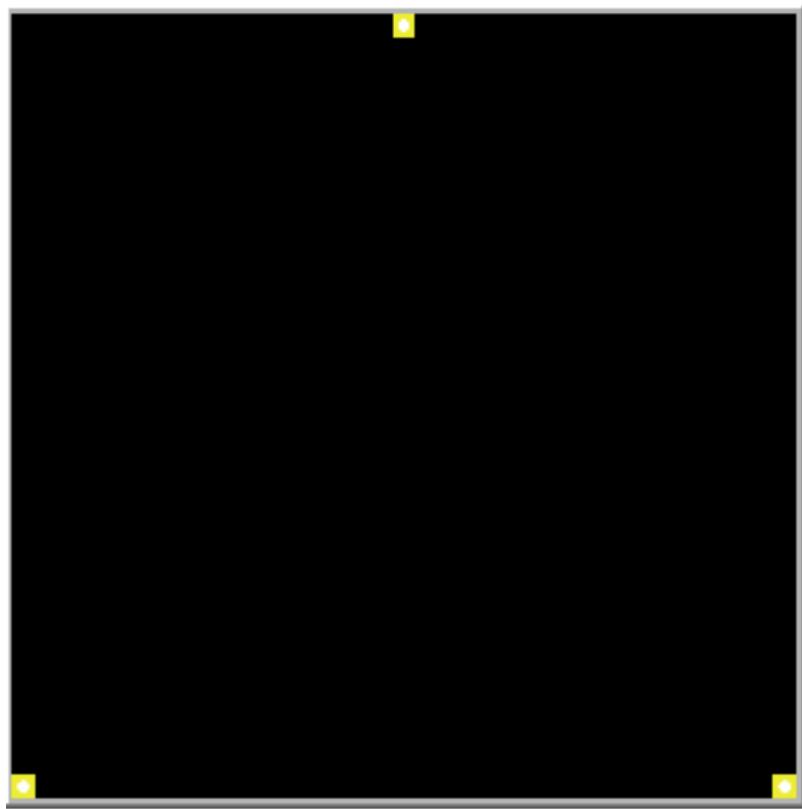


A partir de una distribución aleatoria de celdas blancas y negras, el resultado de ejecutar esta regla es un mosaico de pequeños bloques blancos y negros (Fig.

ure 7.9 (b)). Las celdas rodeadas por celdas de otro color cambian al color de la mayoría, de modo que las celdas aisladas se unen para formar bloques de un color. celdas que tienen celdas mitad blancas y mitad negras como vecinas permanecer sin cambios y forman límites estables a estos bloques. Una vez que las celdas han logrado este patrón moteado, ya no hay ninguna oportunidad para cambio. Sin embargo, la situación es muy diferente, con una pequeña alteración en el regla. Supongamos que algunas personas son a veces más susceptibles a los cambio0 de la moda que otros. Algunas celdas blancas cambiarán a negro si tienen tan solo cuatro vecinos negros, mientras que otras solo cambiarán si tienen al menos seis vecinos negros. Del mismo modo para las células negras. La probabilidad de ser susceptible o resistente a la moda se distribuye de manera aleatoria en el modelo, de modo que, en general, exista el mismo número de celdas que requieran seis vecinos del otro color para cambiar que de aquellas que requieren solo cuatro para cambiar. En resumen, en esta modificación del modelo, ya no tenemos todas las celdas iguales, sino una variación en la individual de la celda.

13.2 Patron Misterioso

A veces encontramos que simples comportamientos, cuando son repetidos por muchos individuos o incluso por un pequeño número de ellos pero repitiendo muchas veces algún comportamiento, se pueden producir patrones inesperados. En esta actividad, construiremos un modelo sencillo que tiene individuos (que llamaremos agentes) con un comportamiento muy sencillo. Colocaremos un número fijo de agentes en cada uno de los tres vértices de un triángulo equilátero.



El comportamiento de cada uno de estos agentes es muy sencillo y es el siguiente, en cada paso del proceso el agente debe:

- Seleccionar al azar una de los vértices del triángulo
- Moverse la mitad de la distancia hacia el vértice seleccionado y a continuación determinar si se unen los puntos blancos dejados por los agentes.
- Marcar su nueva ubicación con un punto blanco.

A veces llamaremos a nuestros agentes tortugas por razones que serán claras más adelante, para empezar entonces, cada tortuga o agente estará ubicado en uno de los vértices del triángulo. ¿Qué patrón (si existe) dará el resultado de las tortugas siguiendo estas sencillas reglas definidas? ¿Qué figura se puede formar si se unen esos puntos blancos dejados por los agentes? ¿Cuántas repeticiones de los pasos anteriores tardará antes de que emerge o surga un patrón interesante? Podemos intentar responder a las preguntas arriba haciendo el ejercicio sobre un papel, pero sería probablemente muy tedioso, y demoraríamos mucho antes de que ese “patrón misterioso” se revele.

13.3 Heroes y Cobardes

Ahora vamos a crear un modelo de otro juego, un juego que llamamos Heroes y Cobardes. Los orígenes de este juego son difíciles de precisar. En las décadas de 1980 y 1990, una compañía italiana llamado el grupo de teatro Fratelli utilizó el juego como una actividad de improvisación. En el 1999 en la conferencia Embracing Complexity, celebrada en Cambridge, Massachusetts, el grupo Fratelli usó este juego con los participantes de la conferencia, y esta parece ser la primera instancia pública del juego. El juego también está relacionado con un juego llamado “Party Planner” propuesto por A. K. Dewdney en su columna “Computer Recreations” en Scientific American (Septiembre de 1987) y reimpreso en el libro The Magic Machine (1990). Para jugar se necesita un grupo de personas. El juego comienza pidiéndole a cada persona que eliga a otra persona para que sea su “amigo” y otra para que sea su “enemigo”. Hay dos etapas del juego. En la primera etapa, a todos se les dice que actúen como “coabredes”

- Actuar como cobarde implica asegurarse que su amigo esté siempre entre usted y su enemigo (efectivamente, escondiéndote de tu enemigo detrás de tu amigo de una manera cobarde).

Cuando la gente jugaba en esta etapa del juego, el centro de la sala quedaba vacía ya que la gente “huye” de su enemigo.

En la segunda etapa del juego, se les pidió a las personas que se comportaran como “héroes”, es decir:

- Moverse entre su amigo y enemigo (protegiendo efectivamente a tu amigo de tu enemigo de manera heroica).

Cuando la gente jugaba la segunda etapa, el centro de la habitación queda muy llena. Fue una diferencia dramática y generó risas y curiosidad. Desde la conferencia, el juego se extendió a la incipiente comunidad de académicos de complejidad, ya que fue un buen ejemplo de comportamiento emergente sorprendente. Eric Bonabeau, presidente de Icosystems y un conocido científico de complejidad y modelador basado en agentes, creó un versión del juego en 2001. Más tarde, Bonabeau creó un modelo del juego basado en agentes que se ejecuta en el sitio web de Icosystems (Bonabeau, 2012). Stephen Guerin, el presidente del grupo Redfish, también creó una temprana versión del juego y lo presentó en 2002 en la Duodécima Conferencia Internacional Anual de la Sociedad de Teoría del Caos en Psicología y Ciencias de la Vida. Algunas imágenes del juego se encuentran en la siguiente figura:



Desde entonces, muchas versiones diferentes del juego han aparecido en varios lugares dentro de la comunidad de sistemas complejos. Además de codificar las primeras versiones, Eric Bonabeau, Stephen Guerin y otros han llevado a la gente a jugarlo en las conferencias. Una variante de tres jugadores aparece en la Guía de pensamiento de sistemas (Sweeney & Meadows, 2010). El juego no siempre aparece bajo el nombre de “Héroes y cobardes”; de hecho, tiene muchos nombres diferentes. Bonabeau lo llama “agresores y defensores” (Bonabeau y Meyer, 2001). También lo hemos escuchado llamado “Amigos y enemigos” y “Espadas y escudos. Construir el modelo es bastante sencillo, lo invito a construirlo:

13.4 Termitas

Philip Morrison, el físico y educador científico del MIT, tiene una historia sobre su infancia. Cuando Morrison estaba en la escuela primaria, uno de sus maestros describió la invención del arco como uno de los hitos centrales y definitorios de la civilización humana. El arco adquirió un especial significado para el joven Morrison. Sentía cierto tipo de orgullo cada vez que veía un arco. Muchos años después, cuando Morrison aprendió que las termitas también construyen arcos, estaba bastante sorprendido, ganó un nuevo escepticismo sobre todo lo que le enseñaron en la escuela, y un nuevo respeto por las capacidades de las termitas. Desde entonces, Morrison se ha preguntado sobre los límites de lo que las termitas pueden hacer. Si pueden construir arcos, ¿por qué no estructuras más complejas? Dado suficiente tiempo, Morrison se preguntaba si las termitas podrían construir un radiotelescopio, Probablemente no. Pero las termitas se encuentran entre los maestros de la arquitectura del mundo animal. En las llanuras de África, las termitas construyen montículos gigantes, nidos que se elevan más de diez pies de altura, miles de veces más alto que las termitas mismas. Dentro de los montículos hay intrincadas redes de túneles y cámaras. Ciertas especies de termitas incluso usan trucos arquitectónicos para regular la temperatura dentro de sus nidos, en efecto convierten sus nidos en elaborados sistemas de aire acondicionado. Como señala E. O. Wilson, “Toda la historia de las termitas … puede verse como un escape lento por medios de innovación arquitectónica a partir de la dependencia de la madera podrida para hacer refugios” Cada colonia de termitas tiene una reina. Pero, como en las colonias de hormigas, la termita reina no le “dice” a los trabajadores de termitas qué hacer. (De hecho, parece justo preguntarse si la designación “reina” es un reflejo de prejuicios humanos. “Reina” parece implicar “líder”. Pero la reina es más una madre para la colonia que un líder.) En el sitio de construcción de termitas no hay unjefe de construcción, nadie a cargo del plan maestro. Más bien, cada termita lleva a cabo una tarea relativamente simple. Las termitas son prácticamente ciegas, por lo que deben interactuar entre sí (y con el mundo que los rodea) principalmente a través de sus sentidos del tacto y el olfato. Pero de las interacciones locales entre miles de termitas, emergen estructuras impresionantes. la construcción de un nido de termitas completo sería un proyecto monumental (que va más all+ a de nuestros alcances). Así que considere un objetivo más modesto:

- constuir un modelo de termitas artificiales para recolectar astillas de madera y colocarlas en pilas.

Las termitas reales en realidad no llevan astillas de madera de lugar a lugar. Por el contrario, comen trozos de madera, luego construyen estructuras con “cemento fecal” que producen de la madera digerida). El desafío es encontrar una estrategia descentralizada para agregar algún orden a la colección desordenada de astillas de madera. Al inicio las astillas de madera se dispersan al azar en el mundo. Pero a medida que se ejecuta el modelo, las termitas deben organizar la madera, en unas pocas pilas ordenadas. ¿Existirá una estrategia simple?

Suponga que cada termita individual sigue las siguientes reglas:

- Si no lleva nada y choca con una astilla de madera la recoje.
- Si lleva una astilla de madera y choca con otra astilla de madera, baja la astilla de madera que lleva.

Estas reglas parecen demasiado simples. No hay un mecanismo para evitar que las termitas saquen las astillas de las pilas existentes. Entonces mientras las termitas colocan nuevas astillas de madera en una pila, otras termitas podrían estar quitándolas, las reglas decritas parecen una buena receta para no llegar a ninguna parte. Pero, si dejamos de lado el escepticismo , podemos intentar seguir estrategia simple y construir los siguientes procedimientos:

1. Avanzar de manera aleatoria (wiggle)
2. Recojer astillas de madera si encuentran una y no llevan ninguna
3. Depositar astillas de madera si llevan una y se encuentran con otra.

¿Podría construir el modelo?

Chapter 14

Proyectos Avanzados

14.1 Simple economy

En las últimas dos décadas, ha habido un creciente interés y uso de modelos basados en agentes en las ciencias sociales. Los métodos basados en agentes pueden ser particularmente valiosos en la ciencia social donde los agentes son heterogéneos y las descripciones matemáticas a menudo no ofrecen suficiente poder descriptivo. Varias comunidades prominentes se han organizado en torno al uso de sistemas complejos y modelos basados en agentes en las ciencias sociales. Entre estos son la Asociación de Ciencias Sociales de Sistemas Complejos (CSSSA) y el Congreso Mundial en Simulación Social (WCSS). En los últimos años, organizaciones como la Asociación Estadounidense de Investigación Educativa (AERA), la Asociación de Economía del Este y Asociación Americana de Geógrafos (AAG) han organizado sesiones sobre la intersección de modelos sociales y modelos basados en agentes. Un área que ha recibido una atención creciente de la comunidad MOBA son las ciencias económicas. Las economías consisten en actores heterogéneos como compradores y vendedores, de ahí que es natural el uso de estos modelos y métodos a la economía. En 1996, los economistas, Josh Epstein y Robert Axtell publicaron un libro que representaba un mundo artificial llamado SugarScape, que fue poblado por agentes económicos. En esta sección, crearemos un modelo económico muy simple que tiene algunos resultados sorpresivos. Supongamos que se tiene un número fijo de personas, digamos 500, cada una con la misma cantidad de dinero, digamos U\$ 100. En cada instante de tiempo (tick), cada persona da uno de sus dólares a cualquier otra persona al azar.

- ¿Qué pasará con la distribución del dinero a largo plazo?

Una importante restricción de este modelo es que la cantidad total de dinero permanece fija, por otro lado nadie puede tener menos que cero dinero. Si alguien

se queda sin dinero, nadie le puede prestar o regalar y debe esperar nada hasta que alguien al azar le entrege un billete. Refinando un poco más la pregunta, podría ser:

¿Existirá una distribución limitante estable del dinero? Si es así, ¿cuál? ¿se concentrará toda la riqueza en unos pocos manos o se distribuirá equitativamente?

Muchas personas, cuando se les plantea esta pregunta, tienen la intuición de que existe una distribución y que es relativamente plana. El razonamiento detrás de la intuición es que dado que ninguna persona comienza con ventaja y la selección de personas a quienes se les transfiere dinero es totalmente aleatorio, ninguna persona debería tener una gran ventaja sobre las demás. Por lo tanto, la distribución de riqueza resultante debería ser relativamente plana y todos deberían terminar con aproximadamente la misma cantidad de dinero con la que comenzaron. Otras personas tienen la intuición de que la riqueza debería distribuirse de forma normal. Para explorar esta cuestión lo invito a construir el modelo.

14.2 Fuego en Línea

Considere un mundo en el que los árboles crecen a lo largo de una línea recta. cada parcela de la recta es adecuada para que crezca un árbol. Cada primavera hay una probabilidad fija, q , de que un árbol nazca en una parcela desocupada. Para hacer las cosas simples, una vez el árbol nace, crece de inmediato a su tamaño completo y permanece así a menos que caiga al suelo. En el verano, tormentas eléctricas caen sobre los árboles y Cada celda tiene una probabilidad f de ser alcanzada por un rayo. Si un árbol es alcanzado, se incendia y la conflagración se extiende a todos los árboles contiguos, las eldas vacías actúan como amortiguadores del fuego ya que evitan su propagación. La tabla 7.2 ilustra un ejemplo de incendio forestal:

TABLE 7.2
A Simple Forest Fire Model

<i>Time</i>	<i>Forest</i>
1	- -
1.G	- t - t t - t t - t t - - t - - - - t t
1.F	- t - t t - t t - t T - - T - - - - t t
2	- t - t t - t t - - - - - - - - - - t t
2.G	- t t t t - t t - - - t t - - t - - t t
2.F	- t T t t - t t - - - t t - - T - - t t
3	- - - - - - t t - - - t t - - - - t t

Note: A t indicates a newly grown or existing tree and a T designates a tree struck by lightning.

En el período 1, el bosque esta vacio. Durante la fase de crecimiento (1.G), los árboles surgen espontáneamente en la linea en lugares designados por “t”. Durante la temporada de rayos, algunos árboles son golpeados y prendidos en llamas; así en el tiempo 1.F dos árboles (designados por “T”) han sido golpeados. Al comienzo del período 2, los árboles golpeados y cualquier vecino conectado ha sido quemados y caen al suelo. Este ciclo de crecimiento y fuego continúa durante los períodos de tiempo posteriores.

Producción del Bosque

Este modelo basado en agentes es muy simple. Los agentes son las parcelas individuales. Cada agente sigue una regla idéntica y fija. En la primavera, si su estado está actualmente vacío, cambie su estado a tener un árbol con probabilidad g ; de lo contrario, mantiene su estado actual. En el verano, si en la parcela hay un árbol, hay una probabilidad f de incendiarse o incendiarse si algún árbol contiguo se ha incendiado. Una vez un agente se incendia, vuelve al estado vacío. A pesar de su simplicidad, este modelo produce algunos resultados provocativos. Defina la **producción** del bosque como el número promedio de árboles que se encuentran de pie al final del verano. En La siguiente figura se muestra la producción para varias tasas de crecimiento en un bosque con cien parcelas y una probabilidad de rayo del 2 por ciento. La producción alcanza su pico a una tasa de crecimiento del 43 por ciento.

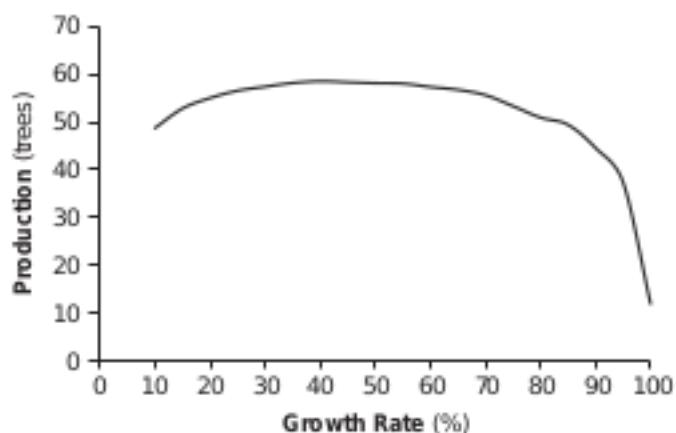


Figure 7.1. Tree production in a Forest Fire model with $f = 0.02$ and homogeneous, fixed rules.

Los cambios en la producción a medida que aumentan las tasas de crecimiento es el resultado de dos fuerzas compensatorias :

- La primera fuerza es la tasa de crecimiento misma, como aumenta, es más

probable que los lugares vacantes estén ocupados por árboles y es mayor la productividad potencial del bosque.

- La segunda fuerza son los rayos,a medida que aumentan los rayos, es más probable que un árbol sea destruido por el fuego. Además, los rayos no solo impactan en árboles individuales sino los árboles contiguos. Por lo tanto, la densidad de los árboles es importante.

A altas tasas de crecimiento, casi todos los árboles son contiguos, y todo lo que se necesita es un rayo para derribar todo el bosque. Esto implica una compensación entre estas dos fuerzas: un crecimiento más rápido significa más árboles, pero más árboles implican colecciones contiguas más grandes de árboles, que promueven fuegos más grandes. En Una versión bidimensional (piense en los árboles que crecen en un tablero de ajedrez), el modelo muestra una conexión mucho más dramática entre producción y crecimiento. En tal modelo hay un cambio muy dramático en la producción a medida que se modifican las tasas de crecimiento. En términos técnicos, tal cambio dramático se conoce como una **transición de fase**, y se puede demostrar (a través de la teoría de percolación) que hay un “valor crítico” de g que resulta en que el sistema pasa de una colección de árboles en gran parte desconectada a una en la que todos los árboles están conectados entre sí como uno solo.

14.3 Un modelo básico de comercio (Toy Trader)

Como consumidor, una vez se identifica lo que quiere, en la mayoría de los casos, el siguiente paso más importante es determinar dónde puede estar ese bien o servicio al precio más bajo. Como vendedor, usted ajusta sus precios de acuerdo con la demanda. Muchas empresas usan el inventario como un proxy de la demanda:

- Si el producto no se mueve o el inventario se está acumulando, entonces se supone que la demanda es demasiado baja a ese precio y, por lo tanto, el precio se reduce.
- Si el inventario está disminuyendo rápidamente, el producto se está agotando, aumentando el precio reduce la demanda y le permite al vendedor ganar más dinero.

Una forma simple de administrar el inventario es tener un objetivo de inventario y usar el precio para mantenerlo en niveles deseados. Estas son las dinámicas que forman la base del Toy Trader modelo que exploraremos y que llamaremos “Toy Trader”

Imagine que Ud se encuentra en un mundo de 51 metros \times 51 metros dividido en celdas de un metro por un metro, en el se encuentran 400 personas distribuidas de manera aleatoria, Usted (y todos los demás) reciben inicialmente US100

y 5 juguetes. El precio de cada juguete se coloca inicialmente en US10 cada uno. Primero, usted (y todos los demás) deben encontrar el precio más bajo de un juguete a una distancia de un metro. Si el precio más bajo disponible lo tienen dos ó más juguetes con el mismo precio, se elige uno al azar. Si hay un juguete disponible, y ud puede pagar el precio, debe comprar el juguete. Al mismo tiempo, las personas que se encuentran dentro del alcance de un metro de ud, verificarán el precio de sus juguetes para ver si son los más baratos de la zona y en ese caso lo comprarian. Su objetivo es utilizar el precio para reducir y mantener su inventario en **3 juguetes**. Despues de cada instancia de venta de un juguete, se revisa el inventario y, si es demasiado alto (más de 3), ud baja el precio multiplicandolo por 0,99. Si es demasiado bajo (menos de 3), aumenta su precio en multiplicandolo por 1.0101. Despues de que todos hayan intentado comprar un juguete, usted (como todos los demás) gira de manera aleatoria y avanza un metro a una celda contigua. Esto concluye una iteración del modelo Toy Trader.

Resumiendo:

El modelo Toy Trader utiliza un población de 400 agentes. Cada agente se inicia con Us 100 y 5 juguetes. En todo el sistema, el stock de juguetes y dinero es fijo e inmutable ($40,000 (400 \times 100)$ dinero y $2000 (5 \times 400)$ juguetes.) Inicialmente, 400 agentes se colocan aleatoriamente dentro de un cuadrado de 51×51 topología de red toroidal, lo que significa que si alguien se aleja de un borde, reaparecen en el lado opuesto. Durante cada iteración, todos los agentes tratan de comprar un solo bien de otro agente al precio más bajo entonces la demanda total es de 400 juguetes por iteración (también es fija). Como los agentes quieren 3 juguetes en su inventario, a largo plazo, se demandan 1200 juguetes de un sistema de suministro de 2000. El sistema está en perpetuo exceso de oferta. No hay razones sistémicas para que cualquiera se vuelva pobre.

Definiciones de Riqueza

En el modelo Toy Trader, hay muchas definiciones posibles de riqueza, se pueden usar tres:

- solo dinero.
- solo juguetes ó
- activos (dinero + juguetes x precio promedio).

Esto nos permite rastrear tres posibles distribuciones de riqueza. Un artículo de Dragulescu y Yakovenko sugiere que si el suministro de bienes es constante, como es el caso en estos experimentos, todas las cantidades deben ir a un Distribución de Boltzmann-Gibbs (0.5 coeficiente de Gini).

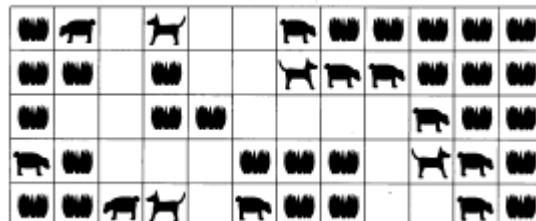
¿Puede construir el modelo y verificar esta hipótesis?

14.4 Un Eco-Sistema

Introducción

El modelo Lotka-Volterra brinda una visión general al dinamismo entre varias especies en un ecosistema, al ignorar variaciones en individuos y asumiendo que todos los miembros de una especie son idénticos o lo suficientemente similares como para que las diferencias entre ellos se puedan ignorar. Una técnica de modelado relativamente nueva en la disciplina de la ecología es la modelación basada en agentes (MOBA), y está basado en el individuo (agente), ya que permite modelar cada miembro de una población por separado.

Descripción del Modelo



En nuestro ecosistema hay tres tipos de especies:

- plantas
- herbívoros
- carnívoros

Tenemos una cuadrícula finita como se muestra en la anterior figura, cada celda está vacía ó tiene una sola planta o animal en ella. Las plantas crecen en espacios vacíos, los herbívoros comen plantas y se mueven por el espacio, y los carnívoros comen herbívoros y también se mueven. Ya que cada criatura consume un tipo de recurso distinto (la planta “consume” un espacio vacío para crecer), puede ser útil consultar la siguiente figura, que ilustra el manejo de recursos en nuestro ecosistema:

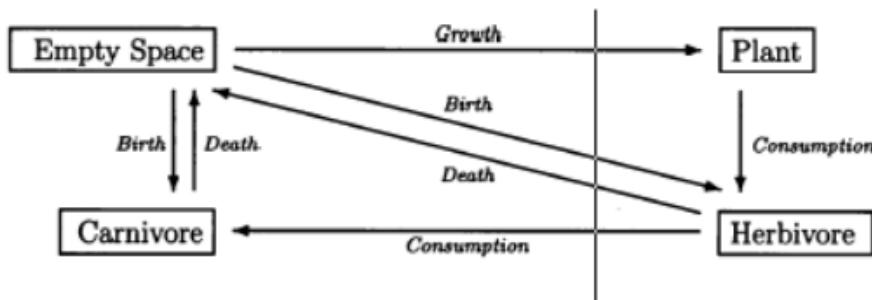


Figure 12.6 Flow of resources in the three-species individual-based ecosystem

Los tipos de acciones o interacciones (comportamiento) obedecen las siguientes reglas:

- Para cada instante de tiempo (tick):
 - Para cada celda vacía, e:
 - Si se tiene 3 o más vecinos que son plantas, entonces se convierte en planta (asumiendo que en la celda no hay ni un herbívoro ni un carnívoro)
 - Para cada herbívoro, h:
 - Disminuya reservas de energía de h una cantidad fija.
 - si h no tiene energía, h muere y la celda queda vacía.
 - Sino, si hay una planta vecina a h, h se mueve sobre la planta, la consume y gana la energía de la planta
 - * si h tiene suficiente reserva de energía, entonces produce un bebé herbívoro en la celda en la que estaba.
 - * Sino, h se mueve a un espacio vacío al azar (si existe) cerca a donde se encuentra.
 - Para cada carnívoro, c:
 - Disminuya reservas de energía de c una cantidad fija.
 - si c no tiene energía, c muere y la celda queda vacía.
 - Sino, si hay un herbívoro vecino a h, h se mueve sobre el, lo consume y gana la energía del herbívoro.
 - * si c tiene suficiente reserva de energía, entonces produce un bebé carnívoro en la celda en la que estaba.
 - * Sino, c se mueve a un espacio vacío al azar (si existe) cerca a donde se encuentra. Si no hay espacio vacío c se mueve donde hay una planta

Comentarios del modelo

Hay varios lugares en los que se utiliza la aleatoriedad. Además de utilizar un orden de actualización aleatorio para los animales, suponemos que los animales tomará una decisión aleatoria cuando se le presente más de una opción para una acción. Por ejemplo, si un herbívoro tiene tres plantas como vecinas cercanas, el herbívoro puede moverse a cualquiera de las tres plantas. Del mismo modo, un carnívoro dará un paso al azar si no hay herbívoros en sus proximidades. Por lo tanto, el modelo individual combina procesos deterministas y aleatorios, al igual a como sucede en el mundo real.

Pregunta

¿Podría construir un modelo de 100x100 celdas y correrlo durante 1500 ticks y observar que sucede con las poblaciones de las tres especies?

Appendix A

Redes

A.1 Introducción

En capítulos anteriores, las interacciones fueron definidas por el entorno y los agentes se comportan de manera aleatoria, estos modelos no incluyen el hecho de que en los sistemas sociales reales las interacciones no son aleatorias. Hoy en día, muchas personas forman parte de sitios web de redes sociales como Facebook. Los miembros no se conectan de forma aleatoria a otros miembros, pero están conectados con sus “amigos.” Si miramos las redes en los sitios web de redes sociales observaremos una estructura, la estructura de una red puede afectar una serie de fenómenos sociales, por ejemplo, la propagación de un rumor. Si un miembro de Facebook bien conectado coloca un rumor en su página, esta información se difundirá mucho más rápido entre la comunidad de Facebook en comparación con el caso en el que el mismo rumor lo pone en la página una persona que solo tiene un amigo. Cuando incluimos estructuras de red en nuestros modelos basados en agentes, podremos estudiar el efecto de la estructura de la red en la difusión de información, normas, enfermedades, comportamiento, etc...

A.2 Conceptos Básicos de Redes.

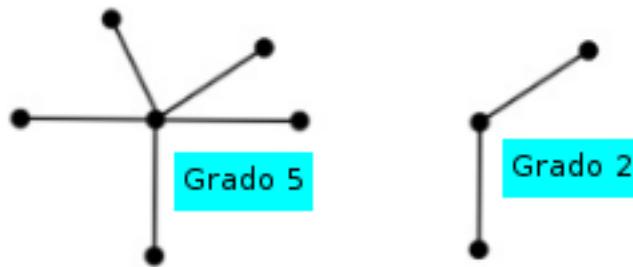
Una red es una colección de nodos que están conectados por enlaces. Un nodo puede representar una persona, una familia, una nación, una computadora, una especie, etc. Los enlaces definen una relación entre nodos como la amistad, el parentesco, una carretera, quién-come-a quién, etc. En el modelo NetLogo tendrímos agentes definidos como los nodos. Los enlaces están definidos por una categoría especial llamada enlaces (links) En la siguiente figura, un enlace representa que el agente A es el padre del agente B y es un enlace dirigido, pero

un vínculo que representa una amistad entre los agentes A y B puede ser un enlace no dirigido. Es decir, si ambos agentes se ven como amigos.

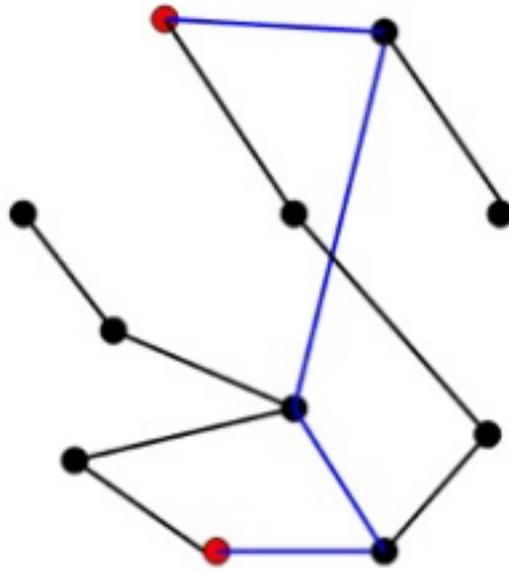
Agente A → Agente B (enlace dirigido)

Agente A ↔ Agente B (enlace no dirigido)

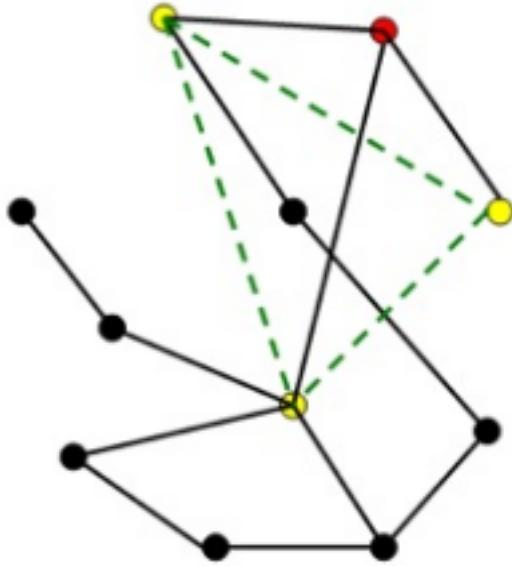
Cada agente puede tener varios enlaces, la cantidad de enlaces que tiene un agente se llama el grado del agente, si un agente tiene cinco conexiones no dirigidas, diríamos que el grado del agente es cinco:



En algunas redes el grado es el mismo para cada agente, mientras que en otras, el grado es diferente para cada agente. La distribución de grados en una población de agentes es una característica importante de una red. Otra característica de las redes está definida por los caminos entre los nodos. En la figura a continuación, verá un camino entre los dos nodos rojos:



Hay varios caminos posibles, podemos calcular el camino más corto entre cada dos nodos de una red y luego tomar el promedio de estos caminos. Esto conduce al promedio de las longitudes de ruta más cortas de una red. La estructura de la red afecta la longitud del camino más corto y por tanto se convierte en un indicador importante para comparar redes. Cuanto menor el promedio de los caminos más cortos, más rápido esperamos que la información se difunda dentro de una red. Finalmente, miremos el coeficiente de agrupamiento de una red (clustering). Este coeficiente indica con qué frecuencia los enlaces de un nodo están vinculados entre sí. En las redes sociales podemos pensar en amigos de la persona A que también son amigos entre sí, una mayor cantidad de amigos de amigos conduce a mayor agrupamiento. En la siguiente figura, puede ver que el nodo rojo tiene tres amigos (amarillos), pero ninguno de estos amigos está conectado. Por tanto, el coeficiente de agrupamiento del nodo rojo es cero:



Para calcular el coeficiente de agrupamiento de una red, tomamos el promedio del agrupamiento para todos los nodos de una red.

Una red se visualiza como un grafo. A menudo, estos grafos no tienen representación espacial, la tienen cuando representan carreteras u otras redes físicas. De lo contrario, estos grafos son solo una representación de las relaciones. Miremos una serie de diferentes tipos de redes y cómo se diferencian las métricas de red (como longitud de la ruta más corta y agrupamiento (clustering))

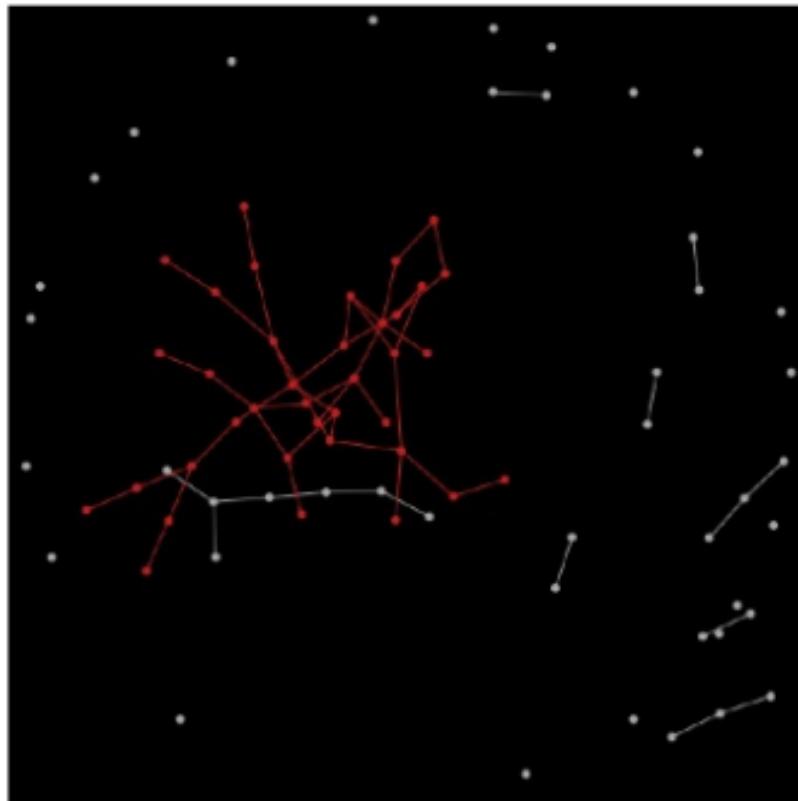
A.3 Redes aleatorias

Comencemos mirando una red simple, la red aleatoria. En esta red hay una población de n nodos. Uno a uno agregamos enlaces entre nodos. Hacemos esto al azar eligiendo uno de los nodos y vinculándolo aleatoriamente con otro nodo. En NetLogo, verá que agregar un enlace aleatorio se realiza de la siguiente manera: Seleccione dos agentes y compruebe si ya existe un vínculo entre ellos. Para hacer esto utilizamos **link-neighbor?** X que es verdadera si hay un vínculo entre el agente y X . Si no hay ningún enlace, podemos crear un nuevo enlace entre los dos tortugas usando la función **create-link-with** X .

```
to add-edge
let node1 one-of turtles
let node2 one-of turtles
ask node1 [
ifelse link-neighbor? node2 or node1 = node2 [add-edge]
```

```
[ create-link-with node2 ]  
]  
end
```

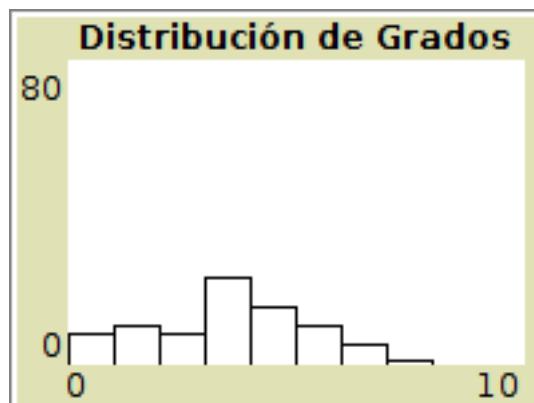
En la siguiente figura , se puede ver cómo se visualiza la red durante la construcción de enlaces. En rojo se representa el componente conectado más grande de la red.



Podemos visualizar el tamaño relativo del componente más grande a lo largo del tiempo :



No todos los agentes tienen el mismo número de enlaces, pero cada uno tendrá un mínimo de un enlace, ya que el modelo se ejecuta hasta que todos los agentes estén conectados. En este ejemplo vemos que el número de enlaces varía de 1 a 12. El grado medio es 4. La coeficiente de agrupamiento (clustering) es de 0.054 y el promedio de longitud de rutas más cortas es de 2.96 enlaces.



A.4 Redes de mundo pequeño

Es posible que tenga un encuentro con un extraño que está a pocos enlaces de distancia de ud en la red, podría estar a solo unos pocos apretones de manos del presidente de los Estados Unidos. Definamos aquí “apretón de manos” como literalmente estrechar la mano de alguien mano como te encuentras en persona. Excluyamos los apretones de manos que los presidentes dan a la multitud cuando tienen mitines. Entonces:

- ¿A cuántos apretones de manos se está del presidente?

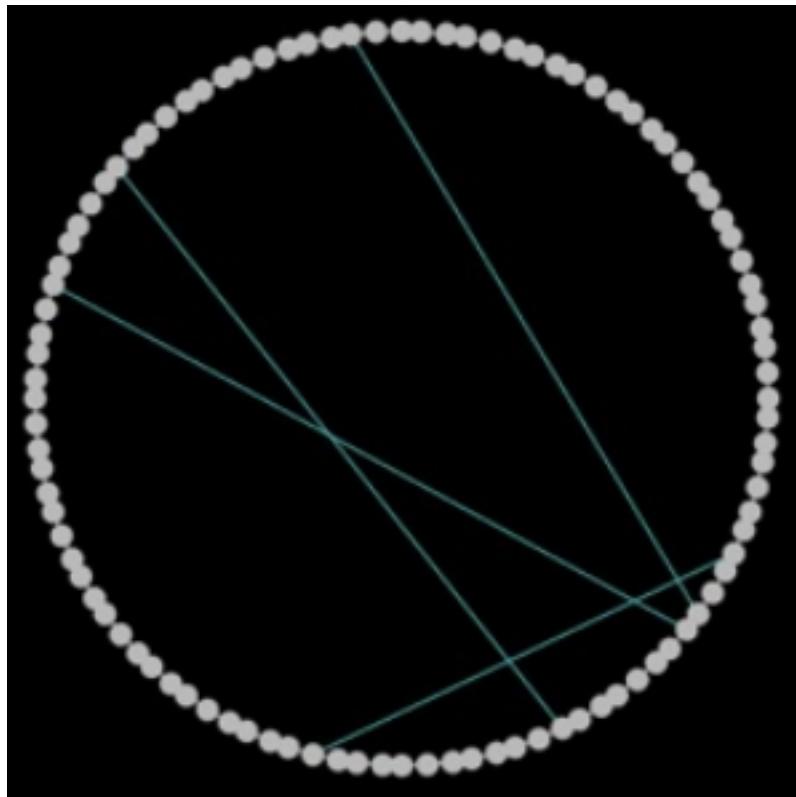
Puede sorprenderse de lo pequeño que es este número para la mayoría de las personas. Hagamos esto con una persona elegida al azar de los miles de millones de adultos que viven en este planeta y tratar de definir a cuántos apretones de manos están lejos. ¿Alguna conjeta? Aquí esperaríamos que este número no sea mucho más que un puñado de apretones de manos.

Esto sugiere que la longitud promedio de caminos de las redes sociales en todo el mundo es pequeña. El psicólogo Stanley Milgram hizo un experimento en la década de 1960 en el que solicitó a varios personas en el oeste medio de los Estados Unidos, enviar una carta a alguien en Boston. Sin embargo, no se recibía la dirección de la persona, sino otro tipo de información como la ocupación. Para alcanzar su objetivo (la persona con la que quieren tener contacto), las personas tenían que enviar la carta a alguien que pensaban que tendría más éxito en encontrar el objetivo. Milgram estaba interesado en si las cartas llegarían a su objetivo y cuántas conexiones se necesitarían. No todas las cartas alcanzaron su objetivo, pero de aquellos que alcanzaron el objetivo, el número medio de enlaces fue de **seis**. Esto llevó al dicho que todas las personas están en promedio a solo seis apretones de manos. Este experimento se replicó 35 años después con correos electrónicos, y nuevamente el número promedio de enlaces entre la fuente y el objetivo se encontró que era de seis.

- ¿Es la red aleatoria un buen modelo para las redes sociales?

No, ya que las redes sociales tienen otro atributo distintivo. Muchos amigos de una persona también son amigos entre sí. Esto es llamado agrupamiento (clustering) dentro de las redes. Para calcular el coeficiente de agrupamiento de un nodo buscaremos a todos los amigos de este nodo, y contamos cuántas amistades posibles existen entre estos nodos. El total de amistades reales entre amigos se divide por el total posible de amistades. Este es el coeficiente de agrupamiento, que es de 0.054 en la red aleatoria del ejemplo. En redes aleatorias este coeficiente no es muy alto ya que muchas conexiones son al azar, y entonces los amigos no son amigos entre sí. Ahora veremos una red de mundo pequeño que captura tanto el alto nivel de agrupación como una longitud de trayectoria media corta. El modelo fue desarrollado por Duncan Watts y Steven Strogatz. Comenzaron con una red regular de nodos que están dispuestos en círculo. Cada nodo está conectado a dos nodos a la izquierda y dos nodos a la derecha, esto significa que el coeficiente de agrupamiento es 0.50, ya que el 50% de los enlaces

entre los amigos de un nodo están conectados entre sí. Para una red de 100 nodos, la longitud de ruta promedio es de aproximadamente 13 enlaces. Cuando se vinculan algunos nodos sin vínculo, la longitud de ruta promedio disminuye significativamente. En la siguiente figura, el reenlace de cuatro nodos conduce a una caída en la longitud promedio de la ruta de 13 a 9 enlaces.



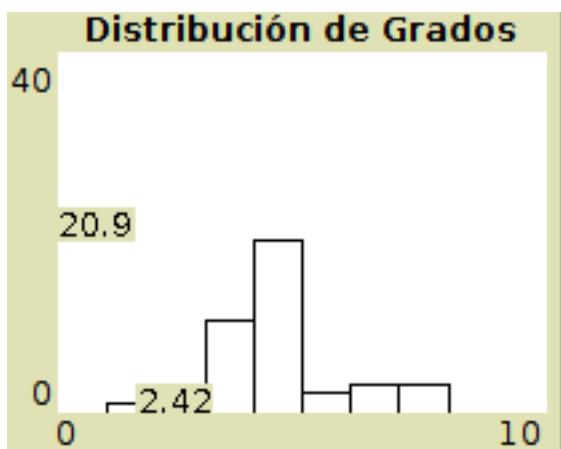
Al volver a vincular un nodo, elegimos un nodo A aleatorio y luego elegimos aleatoriamente de este nodo uno de los enlaces con otro nodo B. El nodo B luego se reemplaza con un nodo elegido al azar de toda la población de nodos, excepto el nodo A y los nodos existentes del nodo A. Una nueva función que usamos es **end1**, que indica el primer agente de un enlace. En un enlace dirigido esta será la fuente, y para los enlaces no dirigidos será la tortuga con el número who más bajo. Primero verificamos si aún se pueden agregar nuevos enlaces al agente de interés. Si este es el caso, se coloca un agente que aún no está conectada con el agente de interés. Luego creamos un enlace entre ambas tortugas. En NetLogo esto se convierte en lo siguiente:

```
let node1 end1
if [ count link-neighbors ] of end1 < (count turtles - 1)
[
let node2 one-of turtles with [

```

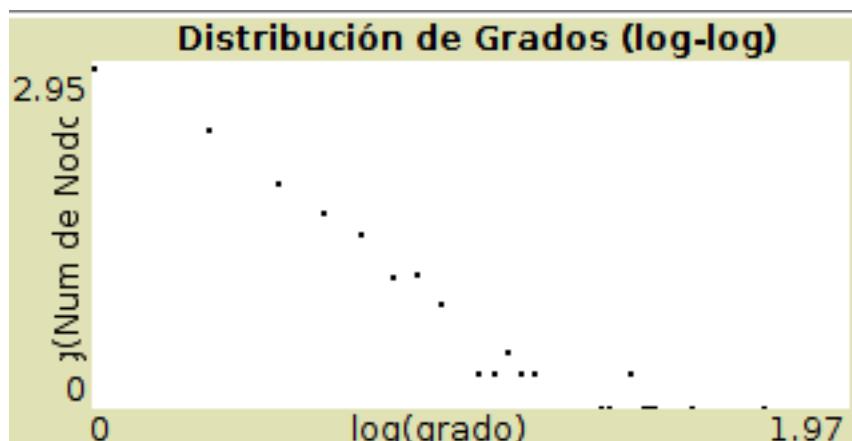
```
(self != node1) and (not link-neighbor? node1) ]
ask node1 [create-link-with node2]
]
```

Cuando comenzemos a volver a enlazar, haremos cortes transversales de una parte de la red a otra. Esto conducirá a caminos más cortos entre los extremos de la red. Si volvemos a vincular muchos nodos, la longitud promedio de los caminos inicialmente disminuirá rápidamente. El coeficiente de agrupamiento se mantiene inicialmente alto, los atajos reducen la ruta longitud entre nodos de ambos lados de la red, pero no afectan la vinculación entre amigos. Solo unos pocos enlaces conducen a redes con alta agrupación (clustering) y una longitud promedio de caminos baja. Ésta es la característica de las redes de mundo pequeño, este es también el tipo de red que encontramos en las redes sociales. La mayoría de las personas están agrupadas y los amigos de los amigos son amigos. Pero si solo un pequeño porcentaje de las amistades son con personas muy diferentes, el promedio de longitudes de camino será pequeña.



A.5 Redes sin escala

La última red que discutiremos es la red sin escala, la red sin escala captura la observación de que la distribución de grados no es normalmente distribuida en muchas redes del mundo real, por ejemplo, muchos sitios web no se enlazan con otros, pero algunos tienen muchos enlaces, estos se denominan centros o “hubs” Si trazamos la distribución de grados en una escala logarítmica , vemos una gráfica lineal. Si fuera una distribución normal, la frecuencia de observaciones con un alto grado, digamos 100, estaría cerca de cero. Esto significa que en una distribución sin escala, la distribución tiene colas grandes, hay más valores atípicos de los que habría en una distribución normal. Este tipo de distribuciones de grados de una red sin escala se han encontrado en muchas redes. A menudo no siguen una línea recta para toda la distribución, sino para una gran parte de ella



Barabasi y Albert (1999) publicaron un modelo simple que genera redes que tienen un distribución de grados libre de escala. El modelo se basa en el llamado apego preferencial:

Empezando con dos nodos, cada nuevo nodo estará vinculado a la red existente, pero con un mayor preferencia a los nodos con mayor número de grados. Por tanto, los nodos con alto grado tienen un mayor probabilidad de conectar más nodos. Esto también se llama “los ricos se vuelven más ricos”. Para modelar el proceso de crear nuevos enlaces con base en los nodos y enlaces ya existentes en la red e implementarlo en Netlogo hay que hacer lo siguiente:

Primero calculamos un valor aleatorio entre 0 y el número total de grados en la red. Luego, en orden aleatorio, los grados de los agentes se agregan hasta que el algoritmo sobrepasa el número acumulativo de grados del número aleatorio generado inicialmente, cuanto más alto sea el nivel del grado, más probable es que se dibuje una tortuga.

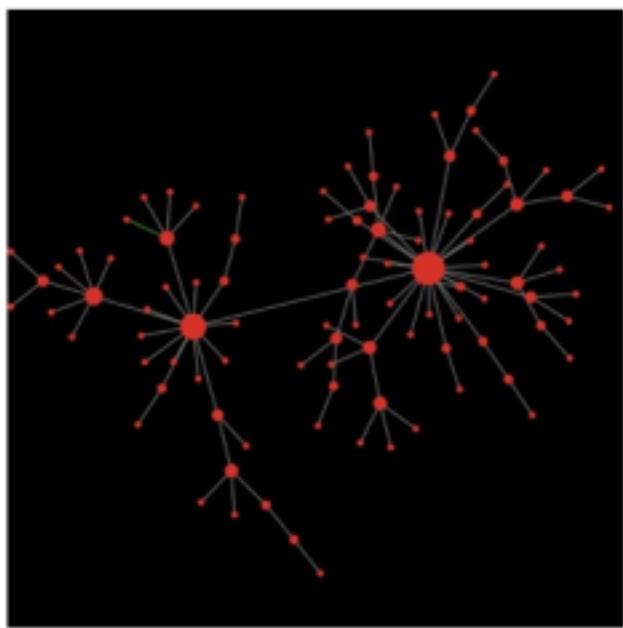
```
let total random-float sum [count link-neighbors] of turtles
```

```

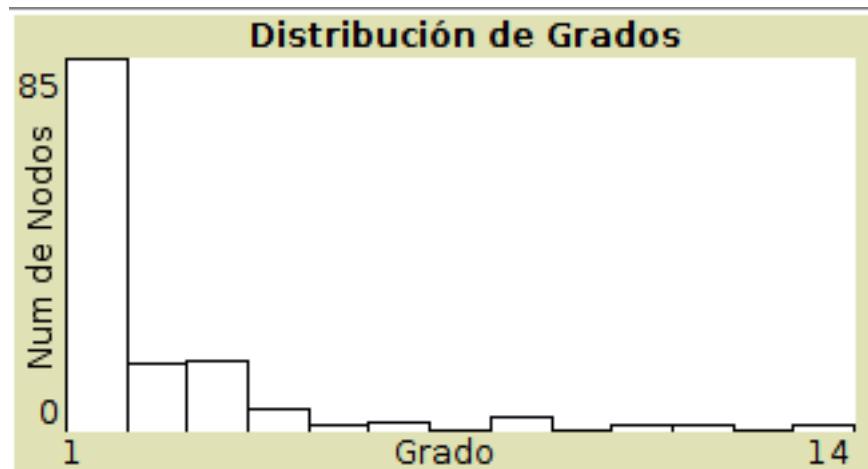
let partner nobody
ask turtles
[
let nc count link-neighbors
if partner = nobody
[
ifelse nc > total
[ set partner self ]
[ set total total - nc ]
]
]

```

En la Figura se ve una instantánea del proceso de generación de la red preferencial, el tamaño de los nodos es proporcional al grado. Vemos algunos hubs y muchos nodos que simplemente están conectados a otro nodo.



La siguiente figura muestra la distribución de grados de una red de 100 nodos. La distribución se aproxima a una distribución libre de escala, lo que significa que es una línea recta en una escala logarítmica del número de nodos y el grado. La red tiene 66 nodos con grado uno, y solo dos nodos con un grado superior a 8. El coeficiente de agrupamiento es 0, ya que los amigos de los amigos no son amigos. El camino promedio longitud es 4.14, que es mayor que la longitud de ruta promedio cuando la red es generada (2,96).



A.6 El Modelo SIR

Una forma clásica de describir la difusión de una enfermedad en una población es distinguir tres tipos de agentes:

- Susceptibles, Infectados y Recuperados.

El llamado modelo SIR, se estudia típicamente usando ecuaciones diferenciales. Hay tres variables, S el número de susceptibles, I el número de infectados y R el número de recuperados. Si no consideramos el cambio en la población y solo miramos el virus, podemos observar que el número de susceptibles disminuye con el tiempo. La tasa de esta disminución depende del número de personas infectadas y depersonas susceptibles. Cuantas más personas susceptibles haya, más personas pueden infectarse y cuantas más personas infectadas haya, más pueden infectar a las personas susceptibles. beta es la tasa de contacto entre las personas:

$$\frac{dS}{dt} = -\beta IS$$

El número de personas infectadas es el número de personas que se infectan menos el número de personas que se recuperan, las personas se recuperan a una tasa fija dada por

$$v$$

:

$$\begin{aligned}\frac{dI}{dt} &= \beta IS - vI \\ \frac{dR}{dt} &= vI\end{aligned}$$

Ahora podemos definir R_0 , que se llama el coeficiente básico de reproducción, este coeficiente se obtiene considerando el número esperado de nuevas infecciones de una infeción en una población de susceptibles. Si

$$R_0 > 1$$

, más personas se infectan y la infección se expande, para combatir la epidemia se necesita que

$$R_0 < 1$$

$$R_0 = \frac{\beta}{v}$$

El Modelo SIR en una red

El modelo basado en ecuaciones diferenciales como se describe arriba asume que hay muchos agentes que interactúan aleatoriamente. Ahora discutimos un modelo basado en agentes en el que se distinguen agentes susceptibles, infectados y recuperados. Se colocan agentes conectados en una red. La red se genera de la siguiente manera. Primero todos los agentes se colocan al azar en el paisaje.

```
create-turtles num-nodos
[
  setxy random-xcor random-ycor
]
```

Luego definimos la cantidad de enlaces que se generarán de manera que la red tenga un grado promedio medio específico (grado-promedio-de-nodos)

```
let num-enlaces (grado-promedio-de-nodos * num-nodos) / 2
```

Los enlaces se agregan uno por uno hasta que se alcanza el número de enlaces. En cada tick uno de los agentes se selecciona al azar, se selecciona el agente más cercano a esta y se establece un enlace (link) entre estos dos agentes

```
while [ count links < num-links ]
[
  ask one-of turtles
  [
    let choice
      (min-one-of (other turtles with [not link-neighbor? myself])
      [distance myself])
    if choice != nobody [ create-link-with choice ]
  ]
]
```

Los agentes que están conectados pueden afectar el estado de los demás. Los agentes infectados pueden infectar agentes susceptibles. Cada vecino susceptible de un agente infectado tiene la probabilidad **prob-difusión-virus** de infectarse. Cada agente infectado tiene una probabilidad **prob-recuperar** de

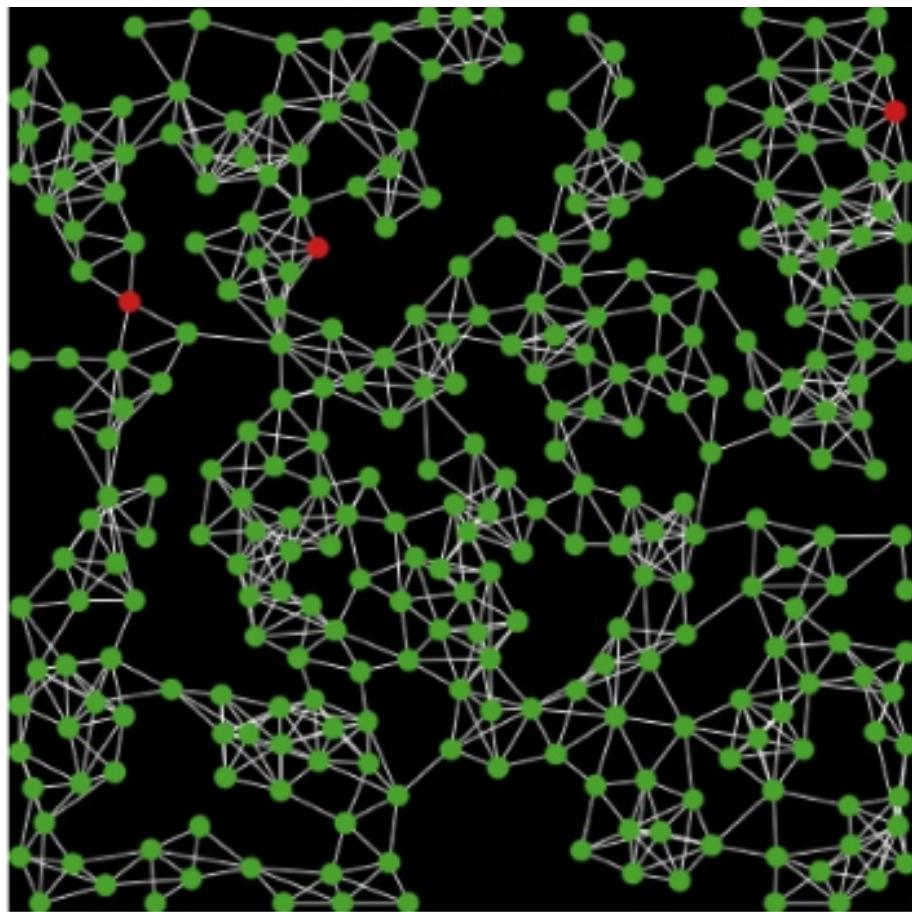
recuperarse. Si un agente se recupera, puede volverse inmune con probabilidad **prob-inmunidad** o volverse susceptible de nuevo. Esto se modela en NetLogo de la siguiente manera:

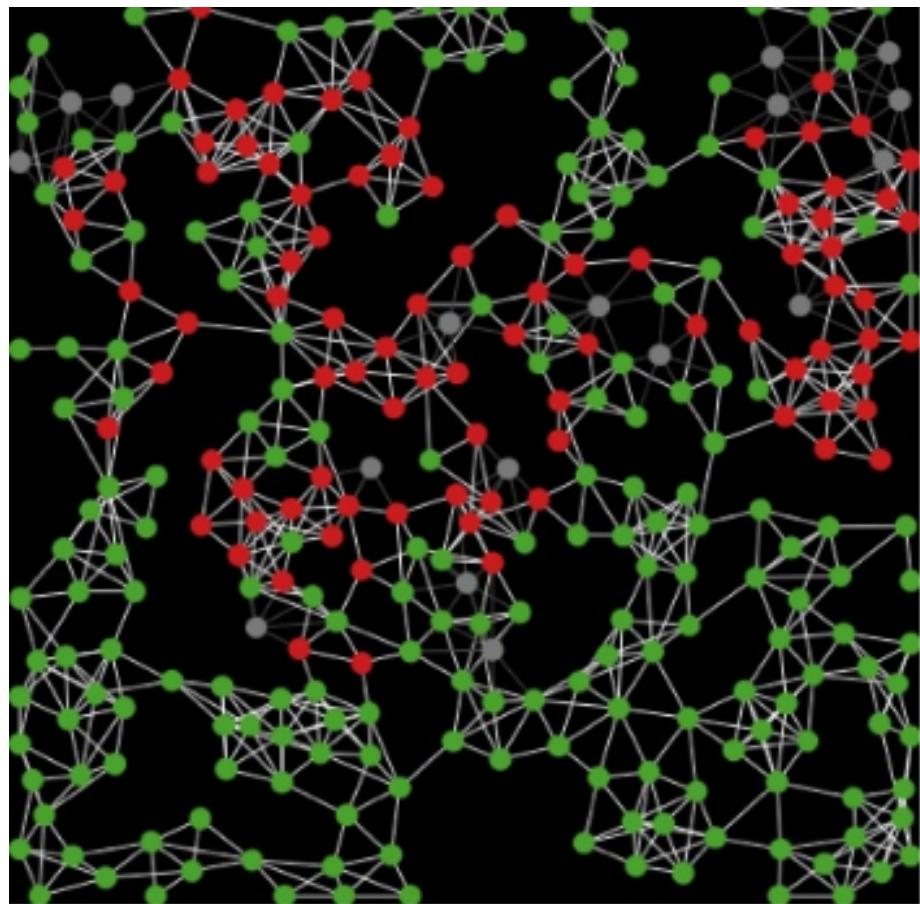
```

ask turtles with [infectado?]
[
  ask link-neighbors with [not resistente?]
  [
    if random-float 100 < prob-difusión-virus
    [
      set infectado? true
      set resistente? false
      set color red
    ]
  ]
]

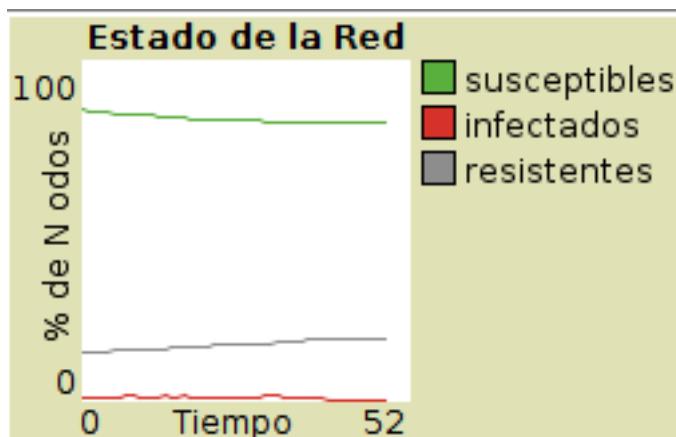
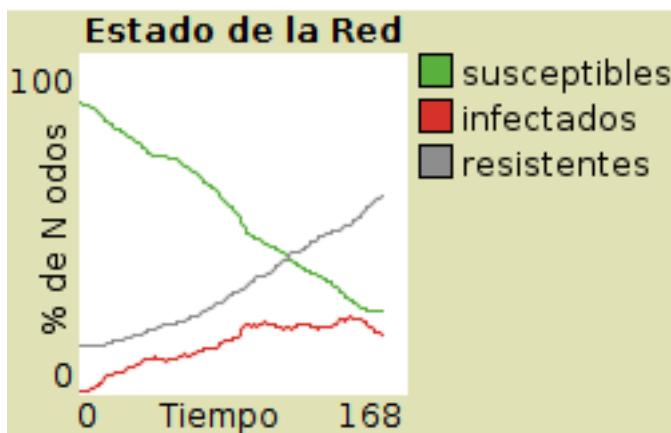
ask turtles with [infectado?]
[
  if random 100 < prob-recuperar
  [
    ifelse random 100 < prob-inmunidad
    [
      set infectado? false
      set resistente? true
      set color gray
    ]
    [
      set infectado? false
      set resistente? false
      set color green
    ]
  ]
]
```

Vemos que el virus se propaga por la red:

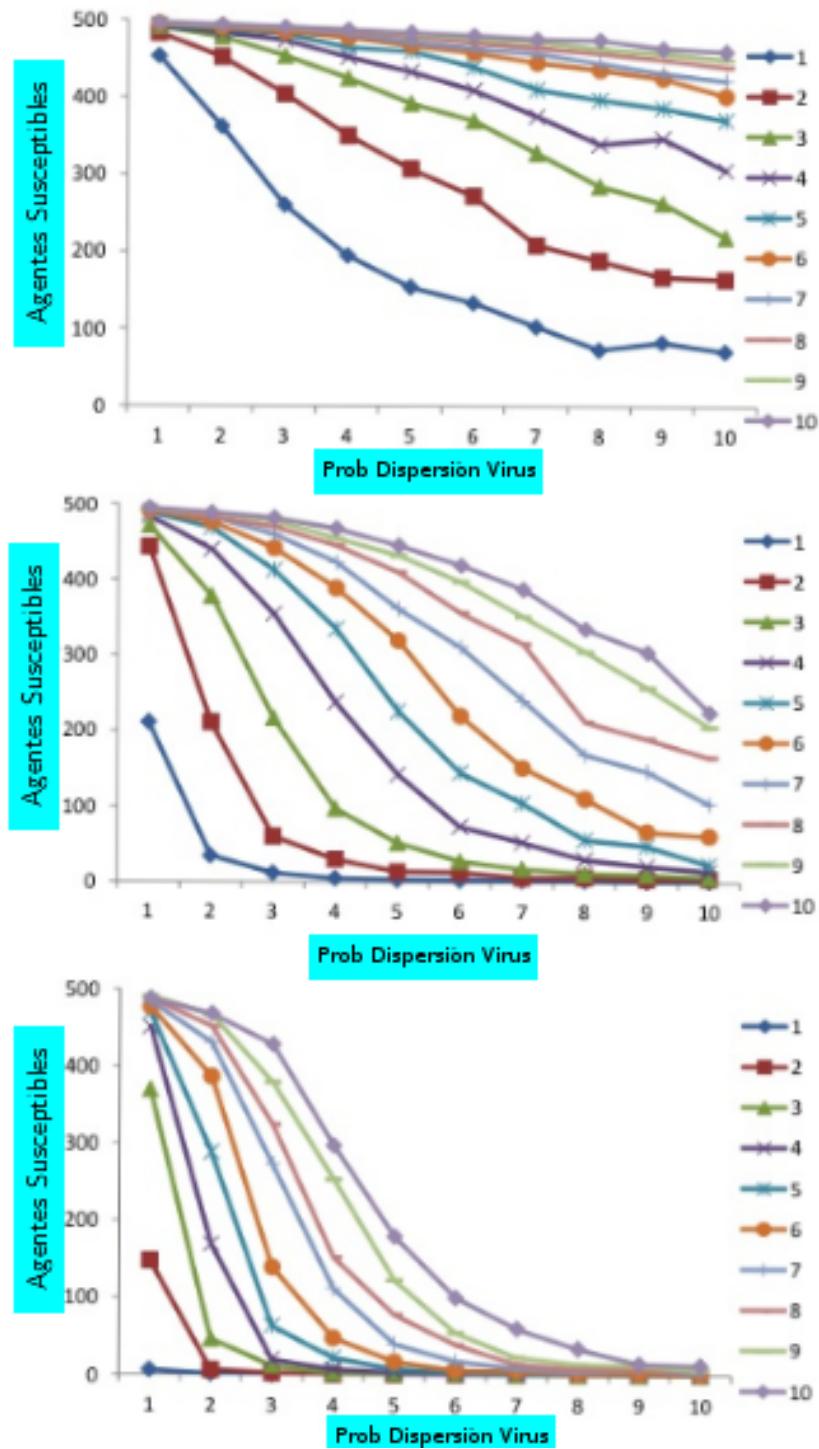




Inicialmente, tres agentes son infectados y en unos pocos pasos vemos que el color rojo comienza a extenderse. Si la probabilidad de volverse resistente a la infección es alta, la enfermedad deja de propagarse de manera temprana en comparación con el bajo nivel de **prob-recuperar**, se muestra que con un valor bajo de **prob-recuperar**, el virus se propaga a aproximadamente el 40% de la población ya que los agentes se enferman nuevamente después de la recuperación (volviéndose susceptibles), y la infección se contiene hasta debajo de un 10% con **prob-recuperar** igual a 50%.



Ahora exploraremos las consecuencias de los parámetros, **prob-recuperar** y **prob-difusión-virus**, asumimos que los agentes que se recuperan no vuelven a ser. Veremos las consecuencias para los grados promedio 4, 6 y 9 para una población de 500 agentes. Cada combinación de parámetros se ejecuta 100 veces. Se muestra que un grado más alto conduce a que más agentes contraigan el virus. Además, vemos que una mayor probabilidad de propagar el virus conduce a que más agentes se enfermen y una menor tasa de recuperación también conduce a que más agentes se enfermen.



A.6.1 Intervención de políticas

Para detener la propagación de enfermedades infecciosas, tenemos varias opciones. Podemos vacunar a las personas si hay una vacuna disponible. Otra opción es reducir las interacciones entre agentes. En la práctica, las escuelas suelen estar cerradas durante un brote de un virus, ya que los niños son un anfitrión eficaz en la propagación de enfermedades como la gripe. Con enfermedades de transmisión sexual como el SIDA, dirigirse a las prostitutas es un enfoque común, ya que son los ejes de interacciones entre agentes. Como ejercicio implementamos la opción de desactivar un agente haciendo clic en ellos con el mouse. Ejecute el modelo y vea si puede aislar el virus:

```
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

Como puede ver, este no es un enfoque fácil, otro enfoque es desactivar los enlaces con los grados más altos. Para modelar esto en NetLogo creamos un bucle while y en cada iteración encontramos uno de los agentes que tiene la mayoría de vecinos y aún no se ha aislado. Luego aislamos ese agente. Cuando un agente es aislado le damos el color azul.

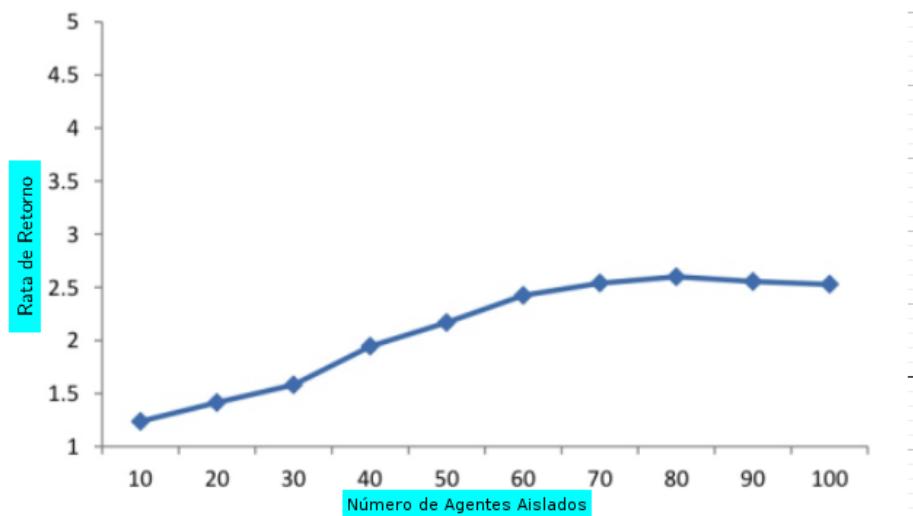
```
let i 0
while [i < numberofisolatedagents]
[
```

```

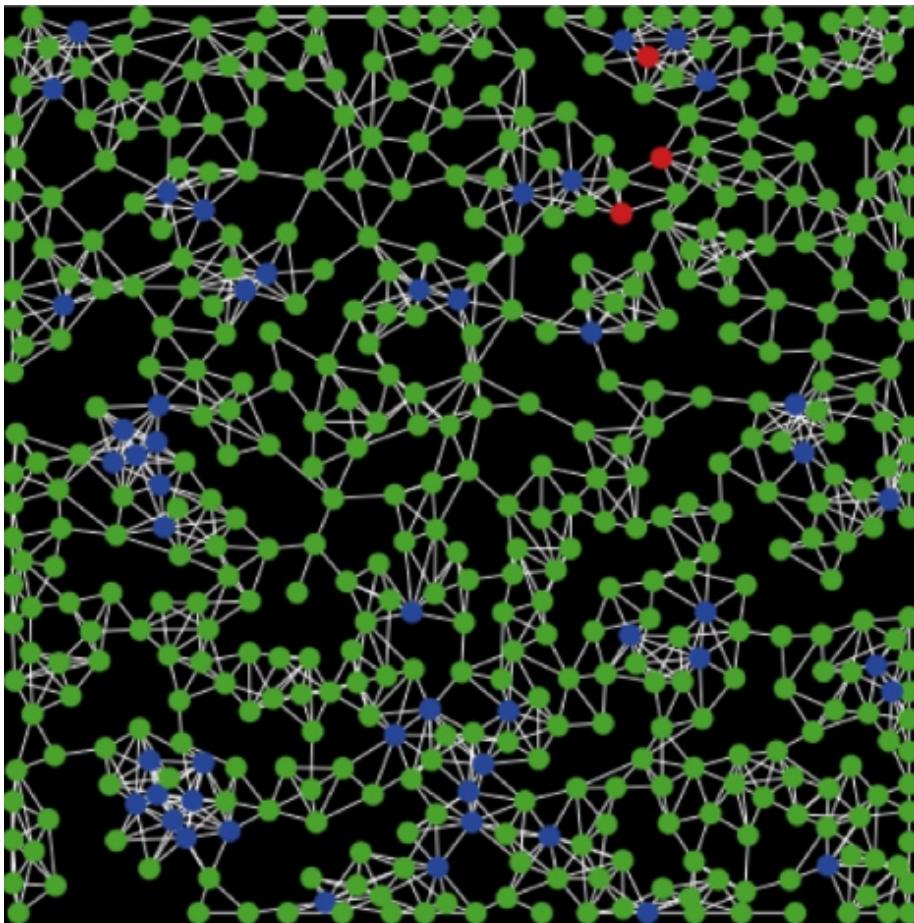
let target max-one-of turtles with [color != blue] [count link-neighbors]
ask target [
  set infectado? false
  set resistente? true
  set color blue
]
set i i + 1
]

```

Simulamos el modelo 1000 veces para aislar diferentes números de agentes y vemos que más agentes no se van a enfermar.



¿Cuál es el beneficio de aislar a una persona? En la gráfica se muestra que por cada persona aislada, en promedio, una o dos personas más se salvan. Sin embargo, después de aislar 80 agentes, el beneficio marginal comienza a disminuir. Al observar un ejemplo de lo que está haciendo la estrategia (Figura 7):

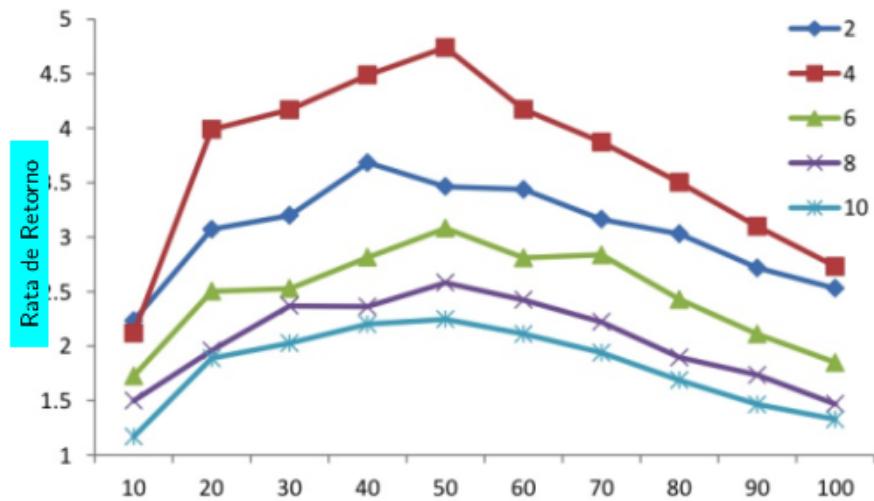


vemos que al aislar los agentes con los grados más altos no es necesario ya que son, de hecho, agentes aislantes lejos de la infección. Además, no aísla la propagación de la enfermedad. Por supuesto, la estrategia más eficaz sería aislar los nodos próximos a los agentes infectados, pero esa no sería una estrategia justa, ya que la computadora es tan rápida como la velocidad de luz a diferencia de las políticas reales. Supongamos que podemos aislar un agente en cada tick:

¿qué agentes querriámos aislar?

Podemos elegir un agente con el grado más alto en cada tick dentro de un radio X de agentes que estén infectados. Un radio más pequeño indicaría que el controlador que está aislando agentes tiene mejor información sobre dónde se encuentran los agentes infectados. No es sorprendente que veamos que una distancia corta conduce a un mejor rendimiento (Figura 8). Pero no demasiado cerca, ya que podría no haber una cantidad suficiente de nodos altamente conectados en un radio pequeño que podrían detener la propagación. La Figura 8 también muestra que más de 50 agentes aislados comienzan a dar menos efecto

por agente aislado. Aunque los agentes no están aislados al comienzo de la simulación, el rendimiento es mucho mejor que aislar agentes con un alto grado al inicio de la simulación.



Appendix B

Tutorial NetLogo

B.1 ¿Qué es NetLogo?

NetLogo es un programa de modelado y programación basado en agentes (MOBA) de código abierto, desarrollado por el Northwestern University Center for Connected Learning (CCL) and Computer-Based Modeling [1]. Se basa en el lenguaje de programación Logo original [2], incorporando y ampliando conceptos y construcciones introducidas en StarLogo y MacStarLogo [3], ambos desarrollados por el Media Lab y el Scheller Education Program del MIT. NetLogo se conoce como una herramienta basada en agentes, debido al hecho de que el lenguaje de programación y la interfaz de usuario están destinadas principalmente para el modelado y simulación de sistemas de múltiples agentes que interactúan (ver “Tipos de Agentes”, p. 13). Por lo general, estos agentes no necesitan tener comportamientos extensos o complicados; se pueden desarrollar útiles modelos a menudo con los agentes siguiendo reglas muy simples. Desarrollado en Java y Scala, y ejecutando en la máquina virtual de Java (JVM), NetLogo es muy portable: modelos escritos en NetLogo para Windows (por ejemplo) pueden ser modificado y ejecutado usando las ediciones OS X y Linux de NetLogo. Los modelos también se pueden ejecutar como applets de Java en las páginas Web (aunque este uso está obsoleto oficialmente), o traducidos “sobre la marcha” al JavaScript para la ejecución basada en navegador. Extensiones a NetLogo pueden ser elaborados en Java o Scala, y el propio NetLogo se pueden crear instancias y controlados por un programa de Java, Scala, o virtualmente cualquier otro idioma que se ejecuta en la JVM. Con cada lanzamiento (versión) importante, NetLogo se ha mejorado de manera significativa. Las características del lenguaje orientadas a los agentes se han racionalizado, para ser más consistentes y coherentes que en las versiones anteriores. Operaciones sobre listas y conjuntos se han ampliado, y el rendimiento de esas operaciones se ha mejorado. Agentes de tipo enlace se ha añadido, en apoyo no sólo para el modelado de las redes (sociales, la

comunicación, etc.), sino también de estructuras físicas y lógicas, y ensamblados.

B.1.1 Terminología de NetLogo

Al igual que prácticamente todos los lenguajes de programación modernos, NetLogo tiene en común con otros lenguajes muchos conceptos y facilidades. Sin embargo, la terminología utilizada en NetLogo a veces difiere significativamente de la de otros idiomas. Algunas de estas diferencias se originaron en dialectos anteriores de Logo, mientras que otros son el resultado de decisiones deliberadas por parte de los diseñadores de NetLogo. En cualquier caso, una familiaridad con algunos de estos términos distintivos, y sus correspondencias con los términos en otros lenguajes, puede ser muy útil. Los siguientes son tres términos que son críticos para empezar: (estos y otros términos se exploran con más detalle, más adelante en este documento)

- Agente

En programación en NetLogo, los agentes son esencialmente objetos: entidades que contienen datos, comportamientos y contextos de ejecución independientes. Para los propósitos de visualización gráfica, un agente en NetLogo (y, en particular, una tortuga, o agente móvil) es más o menos equivalente al concepto de un sprite: una entidad que puede moverse y mostrarse independientemente de otros elementos gráficos. Para obtener más información sobre los tipos de agentes provistos por NetLogo, consulte “Tipos de Agentes”

- Comando

Un comando es conceptualmente equivalente a lo que se suele llamar una sentencia: la especificación de una acción a realizar, para cambiar el estado del sistema. En NetLogo, este estado incluye no sólo las variables globales, el espacio de visualización gráfica y el sistema de archivos, sino también los estados individuales de todos los agentes. Los comandos pueden ser invocaciones de los comandos primitivos (comandos definidos por NetLogo sí mismo) o de los procedimientos comandos (definidos en el código de programación de un modelo).

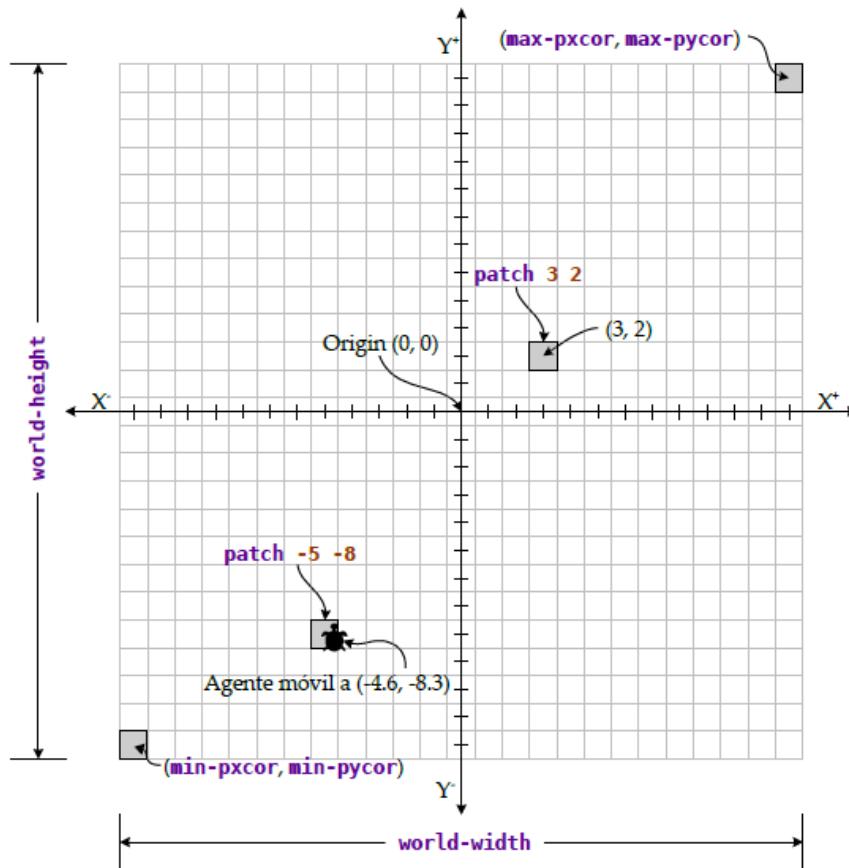
- Reportero

En la mayoría de los lenguajes de programación, llamamos a esto una expresión, considerando que un comando se usa para cambiar el estado del sistema, el propósito de un reportero es calcular y devolver (reportar) algún valor. Esto puede ser un valor de un tipo de datos primitivo (por ejemplo, un número, booleano, o una cadena de caracteres), un agente, o una estructura de datos que contiene (potencialmente) varios elementos de datos. Al igual que los comandos, los reporteros pueden incluir invocaciones de los reporteros primitivos, así como los procedimientos reporteros.

B.2 El mundo NetLogo

Sistema de coordenadas

En la construcción de los modelos de NetLogo es importante entender el sistema de coordenadas utilizado. Este diagrama, con las explicaciones que siguen, ilustran algunos puntos importantes para recordar:



1. Como el sistema de coordenadas cartesianas utilizado tradicionalmente en la geometría analítica, el mundo NetLogo tiene ejes X e Y. El centro del sistema de coordenadas es el origen (que es a menudo – pero no siempre – situado en el centro físico del mundo NetLogo), donde X e Y tienen los valores de cero (0).
2. Superpuesto en el sistema de coordenadas está una rejilla de cuadrículas (parcelas ó cuadrados de tamaño 1 X 1), cada uno de los cuales es un agente fijo (en Inglés y en código de NetLogo, se llama parcela). Cada

tiene un color y una etiqueta opcional. También se puede definir variables adicionales para los agentes fijos.

3. El centro de un agente fijo es un punto en el sistema de coordenadas donde X e Y tienen valores enteros; estas coordenadas se utilizan para referirse al agente fijo. Por ejemplo, patch 3 2 en el diagrama es un cuadrado con su centro en (3, 2); este cuadrado está definido por la región donde $2,5 \leq X < 3,5$ and $1,5 \leq Y < 2,5$. (También podemos hacer referencia a los agentes fijos con coordenadas de punto flotante; se redondearán a enteros si fuera necesario.)
4. Las coordenadas de los agentes fijos son siempre valores enteros, pero eso no es necesariamente así en el caso de un agente móvil. En el diagrama, hay un agente móvil localizada en (-4,6, -8,3), que se encuentra en la cuadrado centrada en (-5, -8). Aunque un agente móvil puede aparecer como si es en dos o más cuadrados a la vez, el punto central del agente móvil es lo que importa: este punto central se trata como la ubicación real del agente móvil, y el cuadrado que contiene ese punto central es considerado como el cuadrado en la que el agente móvil está de pie.
5. El usuario puede cambiar el ancho o alto del mundo NetLogo en cualquier momento; debido a esto, el código de un programa de NetLogo no deben hacer referencia a las dimensiones del mundo con valores literales, a menos que sea absolutamente necesario. Afortunadamente, los programas de NetLogo siempre pueden utilizar world-width y world-height para obtener las dimensiones actuales del mundo (area de simulación).
6. Los agentes fijos en la parte extrema del lado derecho del mundo NetLogo tienen la coordenada X con el valor max-pxcor; los de la cima del mundo tienen Y con el valor max-pycor. Del mismo modo, min-pxcor y min-pycor son las coordenadas X e Y (respectivamente) de los agentes fijos en la parte izquierda extrema y parte inferior (respectivamente) del mundo NetLogo. Estas variables están relacionadas con el tamaño total del mundo, como sigue:

$$\text{world-width} = (\text{max-pxcor} - \text{min-pxcor}) + 1 \quad \text{world-height} = (\text{max-pycor} - \text{min-pycor}) + 1$$

Ángulos y direcciones

Todos los ángulos en NetLogo se especifican en grados, y las direcciones se basan en rumbos de la brújula, con 0 estando “arriba” (es decir, al norte), el 90 hacia la derecha (es decir, al este), etc. Para instruir a un agente móvil que mire a una dirección en particular, podemos hacerlo estableciendo el rumbo del agente a la dirección deseada según la brújula, o diciendo al agente móvil que gire a la derecha o a la izquierda por el número de grados necesarios. También podemos encargar a un agente móvil que mire a otro agente especificando el segundo

agente, con el comando `face`, en lugar de calcular la dirección de la brújula o ángulo de giro requerido.

Topología

Nótese que podemos especificar que el mundo NetLogo tenga continuidad en su borde horizontal, vertical, o ambos, o en ningún borde. Cuando el envolvente (wrapping) está encendido en posición horizontal (por ejemplo), un agente móvil se mueve fuera del borde derecho del mundo y volverá a aparecer en el borde izquierdo, y viceversa. Si el envolvente horizontal no está habilitado, el agente móvil será incapaz de salir por el borde derecho o el izquierdo.

1. ¿Cuál es la “forma” lógica del mundo NetLogo, si el envolvente está encendido en posición horizontal, pero no verticalmente?
2. ¿Cuál es la forma del mundo NetLogo, si el envolvente está encendido verticalmente, pero no horizontalmente?
3. ¿Cuál es la forma del mundo NetLogo, si el envolvente está activado tanto vertical como horizontalmente?

B.3 Programación

Programación en general

Aunque las computadoras (más precisamente, los procesadores) son capaces de manipular los datos de manera muy eficiente, y aunque los procesadores modernos incluyen unidades de procesamiento de punto flotante que pueden realizar cálculos aritméticas, trigonométricas y logarítmicos impresionantes, también son ingenuos: Son generalmente incapaces de realizar la mayor parte de las tareas que el usuario medio consideraría significativa – hasta que se les enseña a hacerlas. Enseñamos a las computadoras a hacer esto a través de la programación: por medio de la codificación de un algoritmo (un procedimiento para completar una tarea o resolver un problema) en una forma que la computadora pueda entender, para lo cual se necesitará insumos específicos, y desde el cual se puede presentar un resultado significativo como salida. Afortunadamente para nosotros, prácticamente todas las computadoras modernas y disponibles en el mercado vienen con millones de líneas de estas instrucciones algorítmicas ya escritas y precargadas en los discos duros, chips de memoria programables, etc. Estas instrucciones forman parte del sistema operativo (que nos permite leer y escribir los datos desde y hacia el teclado, el monitor y archivos), controladores (que indican a la computadora cómo conectarse y hacer uso de los dispositivos de hardware, por ejemplo, adaptadores de visualización de vídeo, unidades de disco, impresoras, dispositivos de memoria externos) y aplicaciones (archivos especiales)

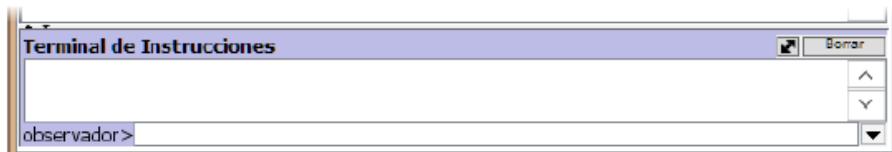
que se pueden ejecutar según la demanda por el usuario, para funciones más específica). Podemos aumentar esta capacidad a través de la instalación o elaboración de nuevos programas para que la computadora las ejecute; cuando hacemos esto, estamos literalmente enseñando a la computadora a realizar nuevas tareas.

Algunos programas de ordenadores son traductores de instrucciones: ellos permiten a los programadores escribir nuevos programas, sin que tengan que entender mucho del funcionamiento interno de la computadora. Estos traductores pueden convertir las instrucciones de los programadores hacia una forma que la computadora puede ejecutar. NetLogo es un tal traductor: nos permite escribir programas en un lenguaje especializado para describir el comportamiento de los agentes; luego convierte estos programas (modelos) en una forma que la computadora puede ejecutar, sin que tengamos que saber nada acerca de cómo la conversión se lleva a cabo. No obstante, podemos considerar los modelos de NetLogo que escribimos como secuencias de instrucciones que enseñamos a la computadora; quizás más útil, podemos considerar nuestra tarea, en la construcción de modelos de NetLogo como el de la enseñanza de NetLogo en sí mismo.

Programación en NetLogo

Damos instrucciones a NetLogo de tres maneras:

1. Podemos escribir instrucciones en el Terminal de Instrucciones (Command Center):



2. Algunas instrucciones pueden ser incluidos en los botones y otros controles en las interfaces de usuario que creamos. Esta funcionalidad es la más utilizada para relacionar los botones que creamos a las nuevas capacidades que hemos enseñado a NetLogo en nuestro programa.
3. Por último, y lo más importante, cuando escribimos instrucciones en la ventana de Código, estamos creando un programa NetLogo, que consiste en uno o más procedimientos. Lo que escribimos en la ventana de Código no se ejecuta inmediatamente, pero se suma a lo que NetLogo sabe hacer. Podemos invocar a esta nueva funcionalidad a través de botones y monitores en la interfaz de usuario, escribiendo comandos en el Centro de Comando, o por referencia en otros procedimientos en el código que hemos escrito en la ventana de Código.

Procedimientos

Para enseñar un procedimiento a otra persona para completar una tarea, es posible comenzar diciendo: “Para hacer X, primero hacer A, luego hacer B”, y así sucesivamente. Enseñando a NetLogo para realizar una tarea es muy similar: utilizamos la palabra clave `to`, seguido del nombre de la tarea, y luego la secuencia de instrucciones que componen el procedimiento; por último, indicamos que no hay más instrucciones para esta tarea con la palabra clave `end`. Por ejemplo, aquí enseñamos a NetLogo un procedimiento que se puede seguir por un agente móvil para dibujar un cuadrado:

```
to draw-square
  pen-down
  repeat 4 [
    forward 10
    right 90
  ]
  pen-up
end
```

Tenga en cuenta que hay palabras con guiones en el código de este ejemplo. Aunque esto no se permite en la mayoría de lenguajes de programación, es válido y común en los dialectos de Logo, y hay varios primitivos (comandos y reporteros definidos por NetLogo sí mismo) con nombres con guiones. Sin embargo, aunque el procedimiento y los nombres de variables pueden incluir guiones – así como muchos otros símbolos de puntuación, no pueden incluir espacios. Ahora que hemos escrito el procedimiento de `draw-square` (dibujar cuadrado), podemos invocarlo por su nombre en el Terminal de Instrucciones, en un botón, o en otro procedimiento. La mayoría de los lenguajes de programación soporta dos diferentes tipos de procedimientos (también llamadas funciones, métodos, subrutinas, etc.): aquellos que modifican el estado del sistema, y los que computan y devuelven un resultado. El procedimiento anterior es un ejemplo del primero: modifica el rumbo y la posición de un agente, pero no devuelve un resultado. En NetLogo, este tipo de procedimiento se llama un procedimiento comando. También podemos escribir un procedimiento que devuelve un resultado; en NetLogo, este tipo se llama procedimiento reportero. Por ejemplo, el siguiente procedimiento calcula y retorna el cuadrado de una entrada:

```
to-report square [input-value]
  report (input-value * input-value)
end
```

La sintaxis para un procedimiento reportero difiere de la de un procedimiento comando en dos aspectos claves:

1. La definición de un procedimiento reportero comienza con `to-report`, en lugar de `to`.

2. El comando primitivo report se utiliza (y se requiere) para salir y devolver un valor de un procedimiento reportero. Como se ha visto anteriormente, parámetros de entrada se incluyen en la definición de un procedimiento (comando o reportero) por encerrándolas entre corchetes después del nombre del procedimiento. Anteriormente, vimos que podemos crear procedimientos comandos y reporteros en nuestro código. De hecho, podemos considerar cualquier programa NetLogo como un conjunto de comandos, reporteros, definiciones y declaraciones. Un comando es una instrucción que invoca (llama) a un procedimiento comando o un comando primitivo (predefinido por NetLogo), especificando los parámetros de entrada requeridos por el primitivo o procedimiento. Por ejemplo, forward 5 es un comando que indica a un agente móvil para mover 5 pasos en la dirección hacia delante; aquí, el comando primitivo es forward (hacia adelante), y la entrada requerida es un valor numérico que indica la distancia a moverse. Por supuesto, mientras que 5 es un valor numérico literal simple, podríamos usar forward (2 + 3) en su lugar, y el resultado será el mismo. Del mismo modo, si existe la variable step-length en nuestro programa, y el valor actual de step-length es de 5, entonces forward step-length también dará lugar al agente móvil a moverse 5 pasos para delante.

¿Qué tienen 5, (2 + 3) y step-length en común? Son expresiones que NetLogo puede evaluar es decir, de las cuales NetLogo puede calcular un valor. Genéricamente, y en la mayoría de los lenguajes de programación, expresiones como éstas se llaman precisamente eso: expresiones. En NetLogo, se llaman reporteros (en algunos contextos, se llaman indicadores). Así que un reportero es un simple valor literal, una variable de referencia, una invocación de un procedimiento reportero o reportero primitivos con las entradas requeridas que son ellos mismos reporteros – o una combinación de éstos, usando operadores aritméticos o lógicos (que también son reporteros primitivos) para calcular el resultado. (Con cualquiera de las posibilidades anteriores, los paréntesis pueden usarse para especificar el orden de evaluación de manera explícita - o incluso sólo para mejor claridad visual.) El ejemplo de un procedimiento reportero en la sección anterior incluye la línea

```
report (input-value * input-value)
```

Aquí, toda la línea es un comando, invocando el comando primitivo report, y proveyendo la entrada esperada por report. Esa entrada es el reportero

```
(input-value * input-value)
```

Este reporte se compone de paréntesis (utilizado aquí para dejar claro que por muy complicado que el reporte sea, estamos calculando y reportando un único valor) que rodea el reporte

```
input-value * input-value
```

El operador aritmético * es un reportero primitivo de un tipo especial: es un reportero infix (uno para el cual se requieren datos antes y después). Así que este reportero consiste en el primitivo *, con input-value (una referencia a una variable) especificada para ambas de las entradas esperadas. Podemos ir por un largo camino con los comandos y reporteros primitivos – pero no podemos realmente escribir programas NetLogo, a menos que también incluyamos declaraciones para la definición de nuevos procedimientos. Como ya vimos, hacemos eso con la palabra clave to o to-report, seguido por el nombre del procedimiento, seguido opcionalmente por una lista entre paréntesis de parámetros de entrada. Entonces, después de que los comandos que conforman el cuerpo del procedimiento, utilizamos la palabra clave end para indicar a NetLogo que se finaliza la definición del procedimiento.

Por último, la mayoría de los programas de NetLogo no triviales también requieren declaraciones. Estas son sentencias escritas al comienzo de nuestro código que declaran información esencial sobre el programa a NetLogo sí mismo: las variables globales que serán asignadas y mencionadas en el código; las razas (breeds) correspondientes a los distintos tipos de agentes móviles y de enlaces que serán utilizados; las variables que serán los atributos de nuestros agentes (más allá de los predefinido por NetLogo); las extensiones NetLogo y archivos adicionales de código fuente que nuestro programa requiere. No miramos a estos en detalle en este resumen, pero los examinaremos en el contexto de ejemplos específicos posteriores; por ahora, sólo nótese que estas declaraciones emplean las palabras claves globals, breed, undirected-link-breed, directed-link-breed, patches-own, turtles-own, links-own, breeds-own, extensions. Un importante comando primitivo es en realidad una combinación de una declaración y un comando: El comando let se utiliza dentro de un procedimiento o bloque de comandos (una secuencia de comandos encerrados por corchetes) para declarar una variable local y para asignarle un valor inicial. Ese valor – y la propia variable – se conserva sólo dentro del procedimiento o del bloque de comandos donde se declara.

Tipos de Agentes

Hay cuatro tipos de agentes en NetLogo; cada uno es capaz de realizar diferentes tipos de acciones, y cada uno sirve a un propósito diferente en un modelo de NetLogo:

1. Observador (observer)
 - Siempre existe solo una instancia de este tipo de agente. Este agente no se visualiza en el mundo NetLogo, pero es el único agente que puede realizar ciertas operaciones globales en un modelo (por ejemplo, clear-all, tick).
2. Parcela (agente fijo, cuadrado, patch)
 - Esto es un agente estacionario; hay exactamente un agente fijo por cada unidad cuadrada en la cuadrícula del mundo NetLogo. Un agente fijo no

se puede mostrar de una manera o forma distinta que un cuadrado, pero cada puede tener su propio color, así como una etiqueta.

3. Tortuga (agente móvil, turtle)

- Esto es un agente que puede moverse por el mundo NetLogo independientemente de otros agentes; las instrucciones que especifican movimiento sólo puede ser ejecutado por los estos agentes móviles. La forma, el color, el tamaño y la etiqueta de un agente móvil pueden ser manipulados por el código de un modelo de NetLogo.

4. Enlace (link)

- Estos son los agentes que conectan un agente móvil con otro. No hay instrucciones para mover enlaces directamente; un enlace se mueve cuando una o ambos de los agentes móviles en sus puntos finales se mueven. Un enlace también se puede configurar como un empalme (tie), donde el movimiento del agente móvil en uno de sus puntos finales se traduce automáticamente en el movimiento del agente móvil en otro punto final. Un enlace puede ser dirigido o no dirigido: con enlaces no dirigidos, no consideramos el enlace como proveniente de un agente móvil a otro, sino simplemente que es entre los dos; un enlace dirigido, por otro lado, es siempre de un agente móvil a otro. Los enlaces y los agentes móviles son los únicos agentes que pueden ser creados o destruidos por las instrucciones contenidas en el código del modelo. Además, los enlaces y los agentes móviles son los únicos agentes que se pueden organizar en razas. Los agentes móviles pueden interactuar con otros agentes móviles mediante la lectura de los atributos de esos agentes, o pidiendo a esos agentes para ejecutar unas instrucciones; también pueden interactuar así con los agentes fijos. Los agentes fijos pueden interactuar con agentes móviles y con otros agentes fijos. Los enlaces generalmente interactúan con los agentes móviles que son sus puntos finales, sino que se pueden interactuar con otros enlaces, agentes móviles y agentes fijos. El observador puede pedir a los agentes móviles, a los agentes fijos y a los enlaces para realizar las operaciones especificadas. Por otro lado, los agentes móviles, los agentes fijos y los enlaces no pueden pedir explícitamente al observador realizar cualquier acción. Sin embargo, los modelos tienen las variables globales (algunos predefinidos por NetLogo, y otros que podemos definir en nuestros deslizadores y en los código del programa); los agentes fijos, los enlaces y los agentes móviles pueden modificar los valores de algunas de estas variables globales – y estos cambios pueden afectar las acciones del observador.

Tipos de Datos

NetLogo es un lenguaje débilmente tipado. Cuando se declara una variable, no se especifica el tipo de datos que se grabará en la variable; la variable puede ser utilizada para almacenar cualquiera de los tipos soportados de datos. De

hecho, durante la vida de una determinada variable, puede contener los datos de diferentes tipos en diferentes momentos (aunque por lo general no es una buena idea). Del mismo modo, la definición de un procedimiento reportero no especifica el tipo de dato devuelto por el procedimiento; es posible (aunque generalmente no es recomendable) que un procedimiento reportero devuelva valores de diferentes tipos en diferentes condiciones. NetLogo nativamente soporta seis tipos de datos:

- Numérico

El formato de punto flotante de doble precisión estándar IEEE 754 se utiliza internamente para todos los valores numéricos en NetLogo [4]. Este tipo de datos utiliza 64 bits para representar números enteros exactos en el rango [-2⁵³ , 2⁵³] y los valores de punto flotante (no exactamente, en su mayor parte) en el rango [-1.797693×10³⁰⁸ , 1.797693×10³⁰⁸]

- Booleano

Este es un tipo que contiene sólo los valores true (veradero) y false (falso). Tenga en cuenta que estos no son los valores de cadena (texto) “true” y “false”; tampoco son intercambiables con los valores de número entero 1 y 0, respectivamente.

- Cadena de caracteres (text)

Una cadena es una secuencia de caracteres, que puede incluir letras, números, signos de puntuación y otros símbolos, así como el espacio en blanco. Tenga en cuenta que los reporteros primitivos para la manipulación de cadena son bastante mínimas: para el procesamiento de texto de uso general, NetLogo es raramente el idioma de su elección. En particular, NetLogo no tiene capacidad integrada para convertir una cadena de dígitos y otros caracteres utilizados para la representación de números en el valor numérico correspondiente. Otra manera en que podríamos pensar en cadenas en NetLogo es como listas de caracteres (consulte “List”, a continuación): muchos de los reporteros primitivos y comandos primitivos que funcionan con listas también operan en las cadenas. Por esta razón y otras, a menudo es útil considerar cadenas no como un tipo de datos distinto, sino como una lista especializada.

- Agente

Una variable puede almacenar (y un reportero puede devolver) una referencia a cualquier agente aparte del observador. Por ejemplo, en un modelo de ecosistema con los procesos de nacimiento y muerte, podría ser útil que cada agente móvil (en representación de algún miembro individual del ecosistema) recuerde sus padres - es decir, mantener las referencias a ellos en sus variables. Si una variable está almacenando una referencia a un agente móvil o un enlace, y ese agente muere, la variable se actualizará automáticamente para contener el valor de nobody (nadie), que es una referencia especial que no se refiere a ningún agente.

- List

La lista es la estructura de datos fundamental (un tipo compuesto de datos, posiblemente contiene múltiples componentes) en NetLogo en cuanto a eso, en

la mayoría de dialectos de Logo. Una lista es una secuencia ordenada de cero o más elementos. Los elementos se pueden adjuntar (ser añadido al final de) o antepuesto (ser introducido al principio de) una lista. Cada vez que una lista dada se atraviesa (sin ninguna modificación que altere el contenido de la lista), el orden de los elementos permanecerá consistente. La heterogeneidad es otra propiedad importante de las listas en NetLogo: Cada elemento puede ser de cualquiera de los tipos disponibles de datos (incluyendo listas). En parte debido a la temprana influencia de Lisp (otra lenguaje de programación) en Logo, y en parte debido a la polinización cruzada desde Python y otros lenguajes, NetLogo incluye un amplio conjunto de primitivos para el manejo de listas. Incluso muchos modelos de NetLogo que no utilizan la mayor parte de estos primitivos, o que no asigna explícitamente valores del tipo lista a las variables, todavía emplean listas implícitamente ,por ejemplo, como valores de entrada con los reporteros que calculan estadísticas agregadas (count, max, mean, min, sum, etc.). Muchos de estos usos se encuentran cuando se leen valores de todos los miembros de un conjunto de agentes (a continuación).

B.3.1 Conjunto de agentes (agentset)

El conjunto de agentes es la segunda estructura de datos fundamental en NetLogo. La mayoría de los lenguajes de programación y las bibliotecas que implementan listas y conjuntos diferencian estos dos tipos de datos en la misma forma que se diferencian los correspondientes conceptos matemáticos:

- A pesar de que una lista es ordenada, un conjunto no lo es. Al agregar un miembro a un conjunto, no tenemos control sobre la posición de ese miembro dentro del conjunto , de hecho, “posición” no tiene sentido para los conjuntos. Si leemos un conjunto varias veces, incluso sin la adición o eliminación de miembros entre los recorridos, el orden de recorrido puede ser diferente cada vez.
- Una lista puede contener el mismo valor en múltiples posiciones dentro de la lista. Un conjunto contiene un especificado valor o bien no lo contiene no puede contener varias instancias del mismo valor al mismo tiempo. En la implementación de NetLogo, los conjuntos tienen algunas limitaciones importantes:
- Un conjunto sólo puede contener agentes (más exactamente, contiene referencias a los agentes). NetLogo no admite conjuntos de valores numéricos, conjuntos de cadenas, conjuntos de conjuntos, etc. Por eso los conjuntos de NetLogo se llaman agentsets.
- Conjuntos de agentes son homogéneos: Sólo un tipo de agente se puede contener en un conjunto en cualquier momento dado. Así que un conjunto no puede (por ejemplo) contener agentes fijos y móviles simultáneamente. (También, como las variables simples no pueden referirse al observador, el observador no puede incluirse en un conjunto de agentes.) Aún con estas

limitaciones, los conjuntos de NetLogo son muy útiles y dominar su uso es una parte esencial de convertirse en un experto desarrollador de los modelos de NetLogo. Un conjunto se puede filtrar por una variedad de predicados lógicos, para crear un subconjunto; se puede atravesar con el propósito de obtener información de todos los miembros (esto produce una lista de valores); tan fácilmente, se puede atravesar con el propósito de pedir a cada miembro para llevar a cabo uno o más comandos; se puede combinar con otro conjunto, para construir un nuevo conjunto de la unión, la intersección, o la diferencia de los dos. Las razas de agentes móviles y enlaces pueden considerarse como conjuntos especiales, en los cuales NetLogo maneja la membresía automáticamente. Además de los tipos de datos anteriores, las extensiones de NetLogo pueden definir nuevos tipos de datos simples y estructurados; referencias a instancias de estos tipos de datos se pueden devolver por procedimientos reporteros y asignar a variables. Unos ejemplos notables de esto son las estructuras de datos definidas por las extensiones tabla, matriz y GIS, que se incluyen en la instalación estándar de NetLogo.

Appendix C

Simulación de Experimentos con NetLogo

C.1 Simulacion Biologica

La categoría de Biología de la Biblioteca de modelos de NetLogo incluye un modelo llamado “Simple Birth Rates”. Es un modelo muy simple, diseñado para simular cómo la diferencia en las tasas de natalidad entre dos especies coexistentes (tortugas rojas y azules) afectan el número de individuos en cada población . Las dos especies difieren solo en el número de descendientes que cada individuo produce cada hora de caminar; la probabilidad de muerte es la misma para ambas especies y se ajusta para que el total el número de individuos es aproximadamente constante. (Si estudia negocios, piense en ellos como dos cadenas de restaurantes de comida rápida que cierran al mismo ritmo pero difieren en la frecuencia con la que abren nuevas tiendas.) Abra este modelo, lea su pestaña Información y juegue un poco con él.

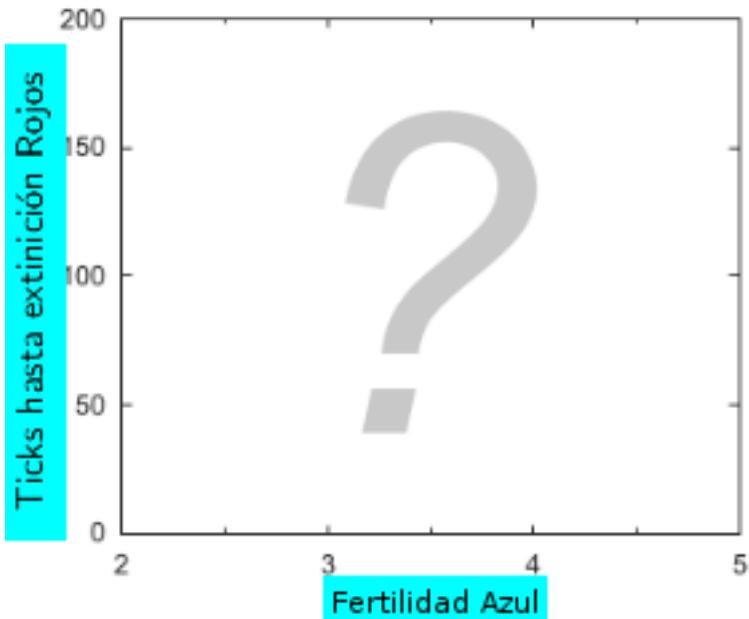
```
## TypeError: Attempting to change the setter of an unconfigurable property.  
## TypeError: Attempting to change the setter of an unconfigurable property.
```

Centrémonos en una cuestión que este modelo puede abordar:

- ¿Cómo el tiempo hasta la extinción de una especie depende de la tasa de natalidad relativa de las dos especies?

Si mantenemos al tasa de nacimiento de la especie roja constante en 2.0 y variamos la fertilidad azul de 2.1 a 5.0, ¿cómo cambia el número de ticks hasta la extinción de la especie roja? En otras palabras,

- ¿qué forma esperamos que tenga la siguiente gráfica ?



Este modelo incluye un botón y un procedimiento llamado run-experimenter" diseñado para ayudar a resolver este problema: ejecuta el modelo hasta que una especie se extingue, nos dice cuánto tiempo tomó y luego comienza otra ejecución. Si pensamos en este problema por un minuto, podemos formarnos una expectativa bastante fuerte de la respuesta. Cuando las tasas de natalidad de las dos especies son cercanas, deberían coexistir durante mucho tiempo. A medida que aumenta la diferencia en las tasas de natalidad, el tiempo hasta la extinción del rojo debería disminuir rápidamente. Pero no importa qué tan rápido se reproduzca el azul, parece poco probable que los rojos se extingan inmediatamente (el modelo se inicializa con 500 de ellos). Por lo tanto, esperamos que la curva de la figura sea alta cuando la fertilidad azul es 2.1, cae rápidamente, luego se aplana a un valor superior a 1 a medida que aumenta la fertilidad azul.

- ¿Son correctas estas expectativas?

Para averiguarlo, necesitamos hacer un experimento de simulación.

C.2 Experimentos de Simulación (Analizador de Comportamiento)

Para ver cómo el tiempo de extinción de los agentes rojos en el modelo varía con el parámetro de fertilidad de los agentes azules, necesitamos ejecutar el modelo en una amplia gama de valores de fertilidad y registrar el tiempo (ticks) en el que el número de agentes rojos llega a cero. Pero hay una complicación: este

modelo, como la mayoría de los MOBAs, es estocástico y produce resultados diferentes cada vez que lo ejecutamos porque las tortugas que mueren en cada tick se eligen al azar. Debido a que el modelo produce resultados diferentes en cada ejecución, para comprenderlo necesitamos calcular los resultados promedio y la variabilidad en torno a esos promedios. Los modeladores a menudo hacen esto de la misma manera que los científicos estudian sistemas reales que son muy variables: ejecutando experimentos que incluyen réplicas de varios escenarios diferentes (los estadísticos a menudo usan la palabra tratamiento , nosotros usamos escenario). En el modelado de simulación, un escenario se define mediante un modelo y un conjunto de parámetros, entradas y condiciones iniciales; si un modelo no tiene elementos estocásticos, produce exactamente los mismos resultados cada vez que ejecuta el mismo escenario. Las réplicas son ejecuciones de modelos en que solo cambian los elementos estocásticos del modelo. Para analizar cómo la fertilidad azul afecta el tiempo de extinción de las tortugas rojas, usaremos un experimento de simulación que varía la fertilidad azul de 2,1 a 5,0 en incrementos de 0,1, produciendo un total de 30 escenarios. Además, ejecutaremos 10 réplicas de cada escenario, la ejecución del modelo continuará hasta que no haya más agentes rojos y se producirá como resultado el tiempo (tick) en el que se produce esta extinción. Estos resultados se registrarán en un archivo que luego podamos importar a R para un análisis estadístico, finalmente, se debe calcular la media y la desviación estándar del tiempo hasta la extinción del rojo y graficar cómo varía la fertilidad del azul.

(Nota: Analizar los resultados en forma gráfica y calcular estadísticas es una parte esencial de modelado que no se puede hacer convenientemente en NetLogo, pór eso usaremos R y Rstudio para ralizar estos análisis).

Este es un ejemplo de un experimento de sensibilidad: un experimento de simulación en el que variamos un parámetro en una amplia gama de valores y luego se ve cómo responde el modelo a él, este diseño experimental es muy útil para comprender cómo un modelo y el sistema que representa, responde a un factor a la vez. Hacer un procedimiento en NetLogo para generar estos datos y que luego se escriban a un archivo tomaría mucho tiempo, y si deseamos hacer un análisis parecido a otro modelo tocaría volver a escribir un procedimiento, es por sto que NetLOgo tiene una herramienta muy útil llamada Analizador de Comportamiento (BehaviorSpace)que nos permite efectuar esots experimentos muy fácilmente. En este punto, debe leer la Guía del Analizador de Comportamiento. Puede pensar en el Analizador de Comportamiento (BehaviorSpace) como un programa separado, integrado en NetLogo, que ejecuta experimentos de simulación en su modelo y guarda los resultados en un archivo para que usted los analice. Completando un cuadro de diálogo simple, puede programarlo para realizar cualquiera o todas estas funciones:

- Crear escenarios cambiando el valor de las variables globales;
- Generar réplicas (llamadas repeticiones en NetLogo) de cada escenario;
- Recopilar los resultados de cada ejecución del modelo y esribalos en un archivo;

- Determinar cuándo detener la ejecución de cada modelo, utilizando un límite de tiempo (por ejemplo, detener después de 1000 ticks) o una condición lógica (por ejemplo, detenerse si el número de tortugas rojas es cero); y
- Ejecutar algunos comandos de NetLogo al final de cada ejecución del modelo.

La información que ingresa en el Analizador de Comportamiento para ejecutar experimentos se guarda como parte del archivo NetLogo del modelo. Por lo tanto, lo primero que debe hacer es guardar su propia copia de este modelo con un nombre único, y de esta manera usar su propia versiones del modelo en lugar de la de la Biblioteca de modelos.

C.2.1 Abra El Analizador de Comportamiento desde el menú Herramientas de NetLogo.

El cuadro de diálogo del Analizador de Comportamiento se abre y le permite crear nuevos experimentos, editar previamente guardados, o copiar o eliminar experimentos. Los términos experimento y configuración del experimento se refieren a un conjunto de instrucciones que definen los escenarios, réplicas, salidas y otras características que deseé. Puede crear y guardar varias configuraciones de experimentos diferentes para el mismo modelo. Esto abrirá el cuadro de diálogo Experimento, que ahora completará modificando los valores predeterminados. A medida que completa el cuadro de diálogo Experimento, puede guardar su configuración haciendo clic en Aceptar en la parte inferior (para reanudar el trabajo en el cuadro de diálogo, vuelva a abrir el experimento haciendo clic en Editar en el cuadro de diálogo del Analizador de Comportamiento).

- Lo primero que debe hacer en el cuadro de diálogo del Experimento es darle al experimento un nuevo nombre. El nombre debe describir el experimento, por lo que debe ser algo como “Efecto-Fertilidad-Azul-Sobre-Fertilidad-Roja”.
- A continuación, puede especificar los escenarios a ejecutar completando el campo etiquetado “Variar las variables de la siguiente manera”. Tenga en cuenta que NetLogo inserta automáticamente las variables globales que están definidas en los elementos de la interfaz, como deslizadores, interruptores y entradas. Desde el Documentación de BehaviorSpace (y el ejemplo proporcionado justo debajo del campo), Puede crear escenarios que varíen la fertilidad azul de 2.1 a 5.0 en incrementos de 0.1 ingresando:

```
["blue-fertility" [2.1 0.1 5.0]]
```

- En el campo “Variar las variables de la siguiente manera”, es una muy buena idea incluir las variables que desea mantener constantes, fijando sus

valores para que no se modifiquen accidentalmente moviendo un control deslizante:

```
["red-fertility" 2]
["carrying-capacity" 1000]
```

- Establezca el valor de “Repeticiones” en 10.

Ahora puede decirle a NetLogo qué resultados desea completando el campo etiquetado “Evaluar las ejecuciones usando estos indicadores”. En este campo, hay que colocar declaraciones de NetLogo que describan los resultados que queremos medir; estas declaraciones se ejecutarán y los resultados se escribirán en el archivo de salida. En este caso, el resultado que desea es el número de ticks en el que los agentes rojos se extinguen. Para hacer esto, puedes decirle al Analizador de Comportamiento que pare el modelo cuando el número de agentes rojos sea cero y que generar el tick (tiempo) cuando esto sucede. El reportero que da el número de ticks es simplemente “ticks” así que:

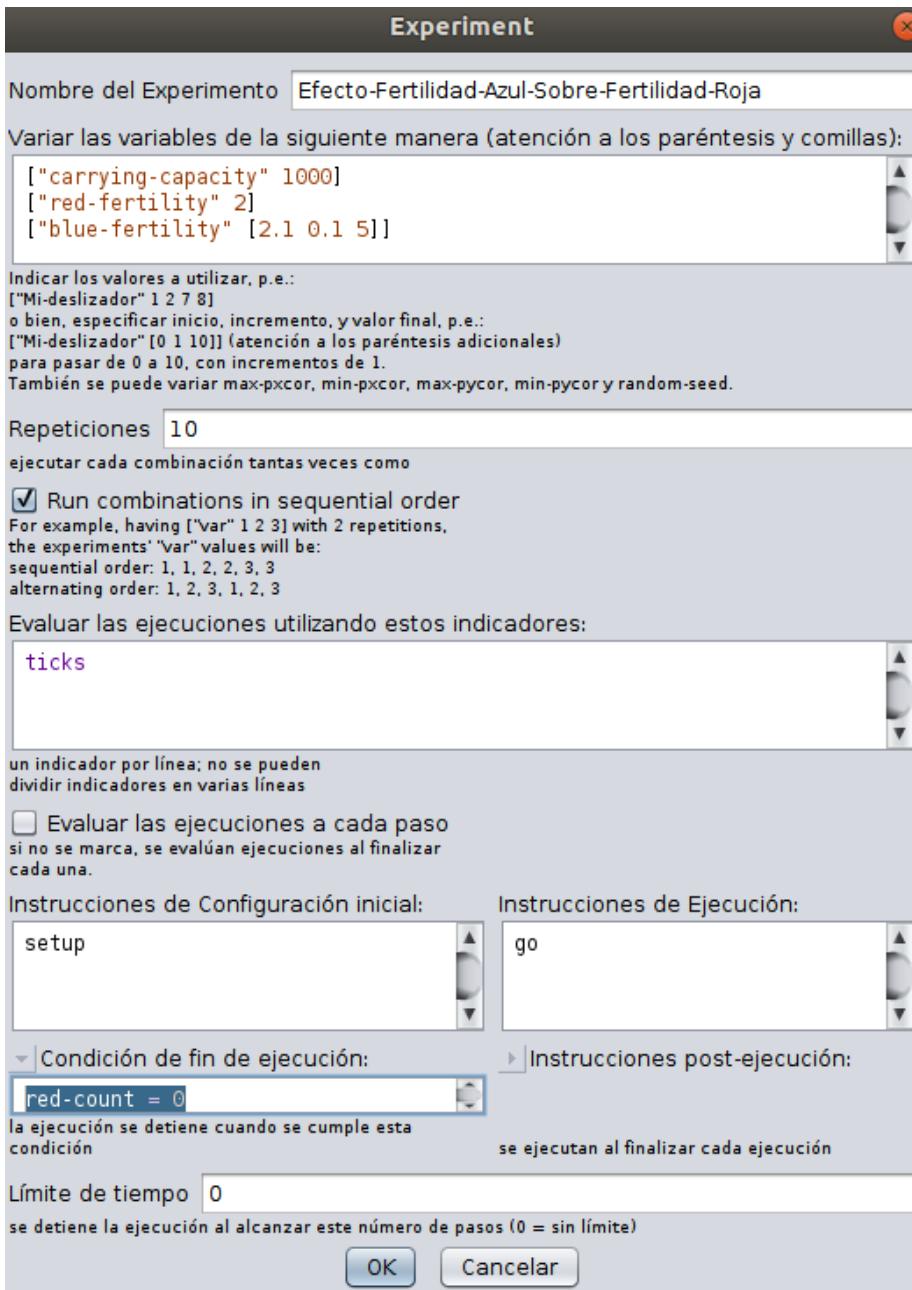
- Coloque la palabra ticks en el cuadro “Evaluar las ejecuciones usando estos indicadores” y desmarque la casilla “Evaluar las ejecuciones a cada paso”

¿Cómo se le dice al Analizador de Comportamiento que detenga el modelo cuando los agentes rojos se extinguen? Se necesita poner una condición en el campo “Condición de fin de ejecución” que sea verdadera cuando el modelo debe detenerse:

- Haga clic en el botón “Condición de fin de ejecución” y coloque la condición:

```
red_count=0
```

C.2. EXPERIMENTOS DE SIMULACIÓN (ANALIZADOR DE COMPORTAMIENTO)301



No es necesario cambiar nada más en el cuadro de diálogo del Experimento, pero asegúrese de entender el significado de los diferentes cuadros (ventanas) de la pantalla * Ahora haga clic en Aceptar para cerrar el cuadro de diálogo del experimento, Cierre el cuadro de diálogo del Analizador de Comportamiento y

Guarde el archivo. Ahora está listo para ejecutar el experimento.

- Abra el Analizador de Comportamiento nuevamente desde el menú Herramientas, seleccione su experimento y haga clic en “Ejecutar”.

Pruebe los formatos de salida de hoja de cálculo y tabla; cada uno de ellos produce archivos en formato .csv diseñados para ser importado en hojas de cálculo u otro software. También tiene la opción de cuántos de los procesadores de su computadora usar, si tiene varios procesadores. También puede anular la selección de las actualizaciones de Vista y gráficos y monitores para acelerar la ejecución.

C.2.2 Nota Importante

El Analizador de Comportamiento es una herramienta extremadamente importante, pero si no comprende exactamente cómo funciona, lo encontrará frustrante y puede malinterpretar sus resultados. Necesita entender varios detalles:

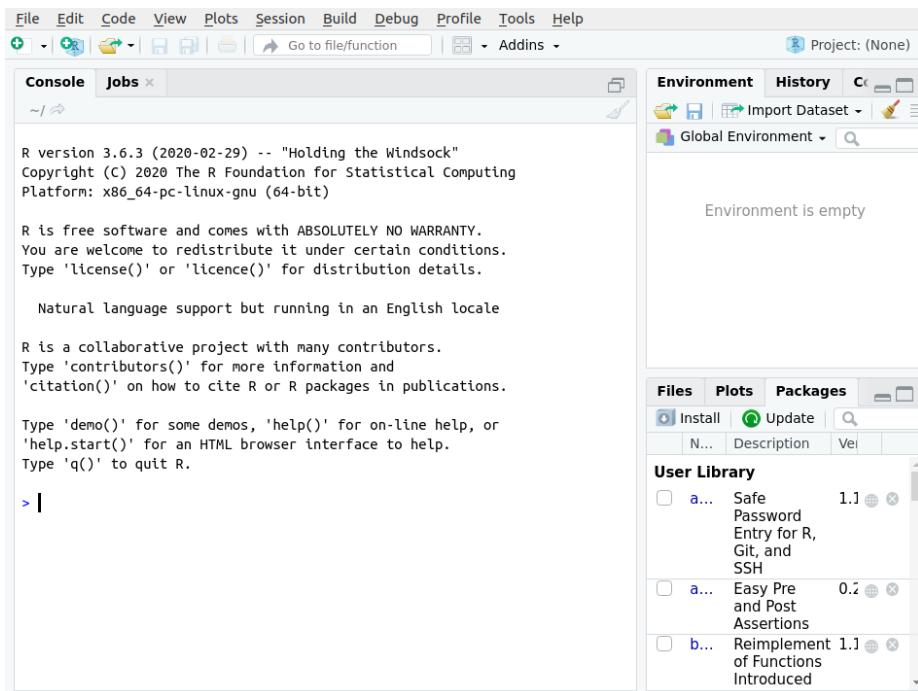
- Al comienzo de cada ejecución, El Analizador de Comportamiento cambia los valores de las variables como se especifica en su caja “Variar las variables de la siguiente manera..” El Analizador de Comportamiento establece estos valores antes de ejecutar el procedimiento **setup**. Por lo tanto, si el setup establece el valor de una variable que se supone que El Analizador de Comportamiento va a controlar., el valor dado por El Analizador de Comportamiento será sobrescrito por el valor establecido en el setup. Además, el setup generalmente comienza con el comando clear-all, que establece el valor de todas las variables en cero, excepto las variables globales que se definen en un deslizador, selector, interruptor o entrada en la interfaz. En consecuencia, cualquier variable que desee que El Analizador de Comportamiento controle debe ser definido e inicializado solo por uno de estos elementos de interfaz.
- Cuando la casilla “Evaluar las ejecuciones a cada paso” no está seleccionada, los resultados se calculan y se escriben en el archivo de salida solo después de que se haya detenido la ejecución del modelo. La corrida puede ser detenida por un stop en el procedimiento go (como en muchos MOBAs), o usando la “Condición de Fin de Ejecución” ó la opción “Límite de Tiempo”
- Cuando la casilla “Evaluar las ejecuciones a cada paso” está seleccionada, El Analizador de Comportamiento evalúa las ejecuciones a cada paso y produce su primer resultado cuando se completa el procedimiento setup. Esto informa el estado del modelo después de inicializado y antes de que comiencen las simulaciones. El Analizador de Comportamiento luego produce resultados cada vez que termina el procedimiento go.

C.3. ANALIZANDO LOS RESULTADOS DEL EXPERIMENTO EN RSTUDIO303

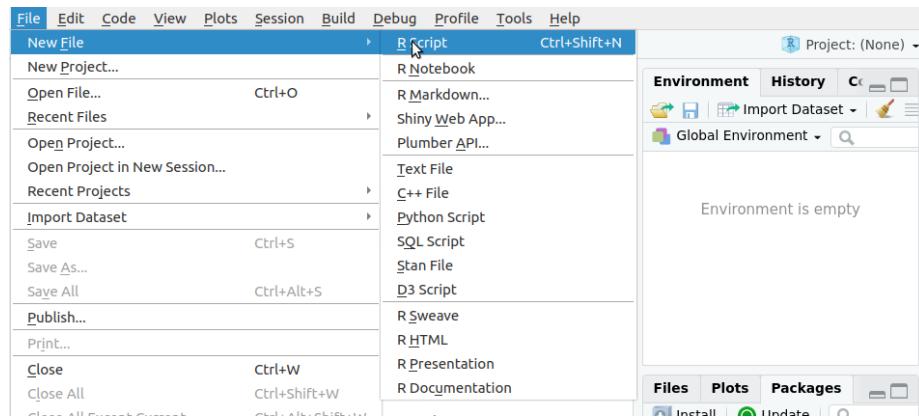
C.3 Analizando los resultados del experimento en Rstudio

Cuando El Analizador de Comportamiento ha terminado de ejecutar el experimento, puede importar los resultados al software de su elección y calcular la desviación estándar y media de las 10 repeticiones del experimento (tiempo hasta la extinción del rojo para cada valor de fertilidad azul) y además dibujar un gráfico ilustrativo.La menera de hacerlo es la siguiente:

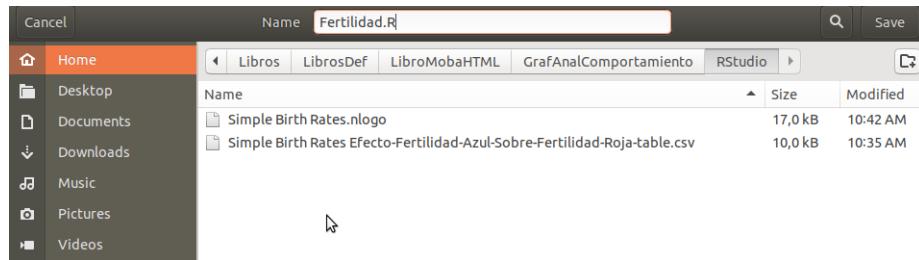
Abra el Programa RStudio:



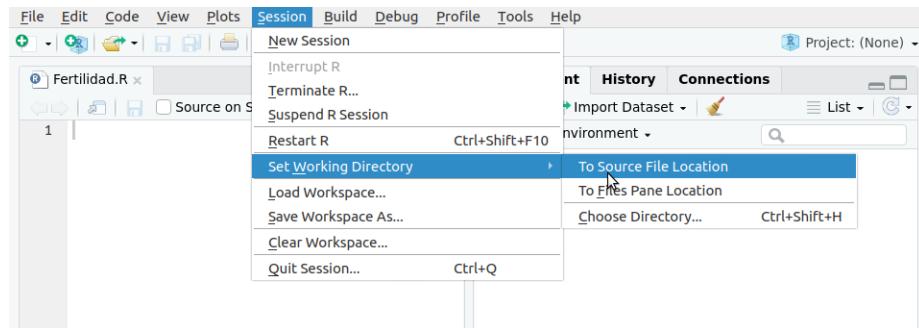
Cree Un Archivo Nuevo (Script) en R-Studio :



Guárdelo con el nombre Fertilidad.R en el mismo directorio donde se encuentra el archivo generado por el Analizador de Comportamiento:



Definamos ahora el directorio de trabajo como el directorio donde se encuentran tanto el Archivo del Experimento como el script que acabamos de crear, paara ello hay que seleccionar la opción:



La primera linea del Archivo de R será el comando que importe la librería de R que usaremos para hacer el análisis del experimento, coloque lo siguiente y luego oprima el botón Run:

```
library(tidyverse)
```

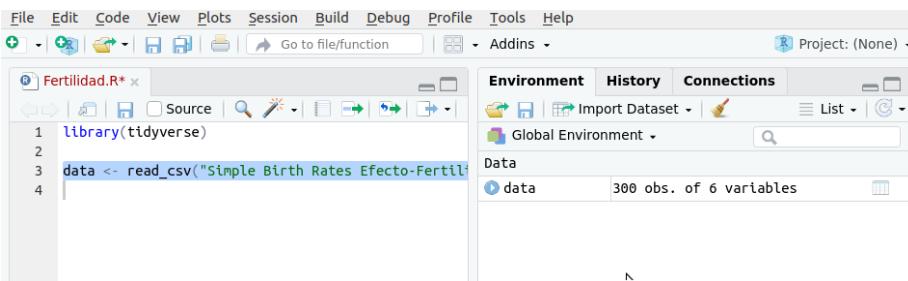
La libreria se debe cargar, esta nos permite graficar y hacer análisis estadísticos

C.3. ANALIZANDO LOS RESULTADOS DEL EXPERIMENTO EN RSTUDIO305

Ahora vamos a importar a R los datos del experimento, usaremos la función `read_csv` para leer el archivo del experimento, coloque lo siguiente y luego oprima el botón Run:

```
data <- read_csv("Simple Birth Rates Efecto-Fertilidad-Azul-Sobre-Fertilidad-Roja-table.csv",skip=1)
```

En la parte derecha superior de la pantalla debe aparecer lo siguiente:



Esto indica que el experimento se cargó a Rstudio, si hace clic en data puede observar los datos del experimento generado por NetLogo:

A screenshot of the RStudio data viewer. It displays a table with 9 rows and 6 columns. The columns are labeled: [run number], carrying-capacity, red-fertility, blue-fertility, [step], and ticks. The data shows values for each row, with the last row being highlighted. The bottom of the viewer shows the text: "Showing 1 to 9 of 300 entries 6 total columns".

[run number]	carrying-capacity	red-fertility	blue-fertility	[step]	ticks
1	2	1000	2	2.1	129
2	4	1000	2	2.1	137
3	3	1000	2	2.1	168
4	1	1000	2	2.1	194
5	5	1000	2	2.1	141
6	8	1000	2	2.1	125
7	6	1000	2	2.1	193
8	7	1000	2	2.1	176
9	11	1000	2	2.2	63

Lo primero que haremos con esta tabla de datos es cambiar los nombres de las columnas a nombre más sencillos, coloque en el Archivo de R lo siguiente y oprima el botón Run:

```
colnames(data) <- c("corrida", "capacidad", "fertRoja", "fertAzul", "paso", "ticks")
```

Debe aparecer la tabla con los nuevos nombres:

	corrida	capacidad	fertRoja	fertAzul	paso	ticks
1	2	1000	2	2.1	129	129
2	4	1000	2	2.1	137	137
3	3	1000	2	2.1	168	168
4	1	1000	2	2.1	194	194
5	5	1000	2	2.1	141	141
6	8	1000	2	2.1	125	125
7	6	1000	2	2.1	193	193
8	7	1000	2	2.1	176	176
9	11	1000	2	2.2	63	63

Dejaremos en la tabla solo las columnas que necesitamos, que son tres:

- fert-roja
- fert-azul
- ticks

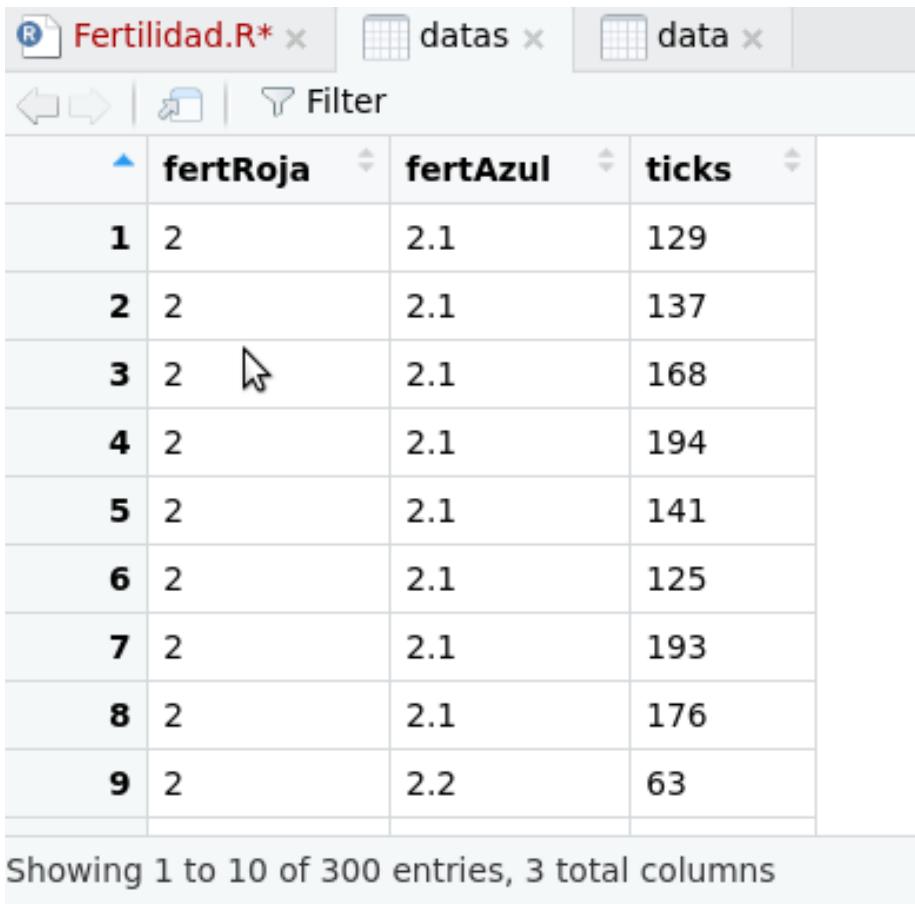
(Nota: observe que paso y ticks son la misma columna)

Coloque el comando siguiente y oprima el botón Run:

```
data %>% select(fertRoja,fertAzul,ticks) -> datas
```

Tenemos entonces una tabla con solo las 3 columnas que necesitamos:

C.3. ANALIZANDO LOS RESULTADOS DEL EXPERIMENTO EN RSTUDIO307



	fertRoja	fertAzul	ticks
1	2	2.1	129
2	2	2.1	137
3	2	2.1	168
4	2	2.1	194
5	2	2.1	141
6	2	2.1	125
7	2	2.1	193
8	2	2.1	176
9	2	2.2	63

Showing 1 to 10 of 300 entries, 3 total columns

Necesitamos ahora agrupar los datos por fertilidad azul, colque lo siguiente y oprima el comando RUN:

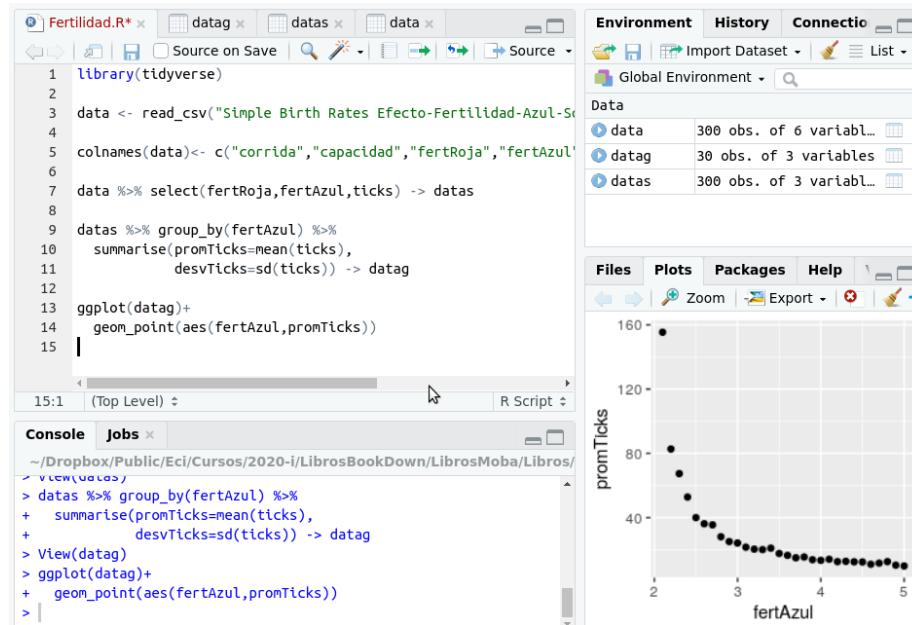
```
datas %>% group_by(fertAzul) %>%
  summarise(promTicks=mean(ticks),
            desvTicks=sd(ticks)) -> datag
```

Obtenemos una tabla (datag) con los datos agrupados y donde se ha calculado el promedio y la desviación para cada vlaor de Fertilidad azul (2.1 a 5.0):

	fertAzul	promTicks	desvTicks
1	2.1	155.4	27.988887
2	2.2	82.7	19.832913
3	2.3	67.4	15.728248
4	2.4	52.8	15.425808
5	2.5	40.0	7.483315
6	2.6	36.2	7.857056
7	2.7	35.5	13.410195
8	2.8	28.1	6.640783
9	2.9	25.1	3.348300

Ahora podemos graficar nuestros datos, coloque en Rstudio lo siguiente y oprima el botón Run:

```
ggplot(datag)+  
  geom_point(aes(fertAzul,promTicks))
```

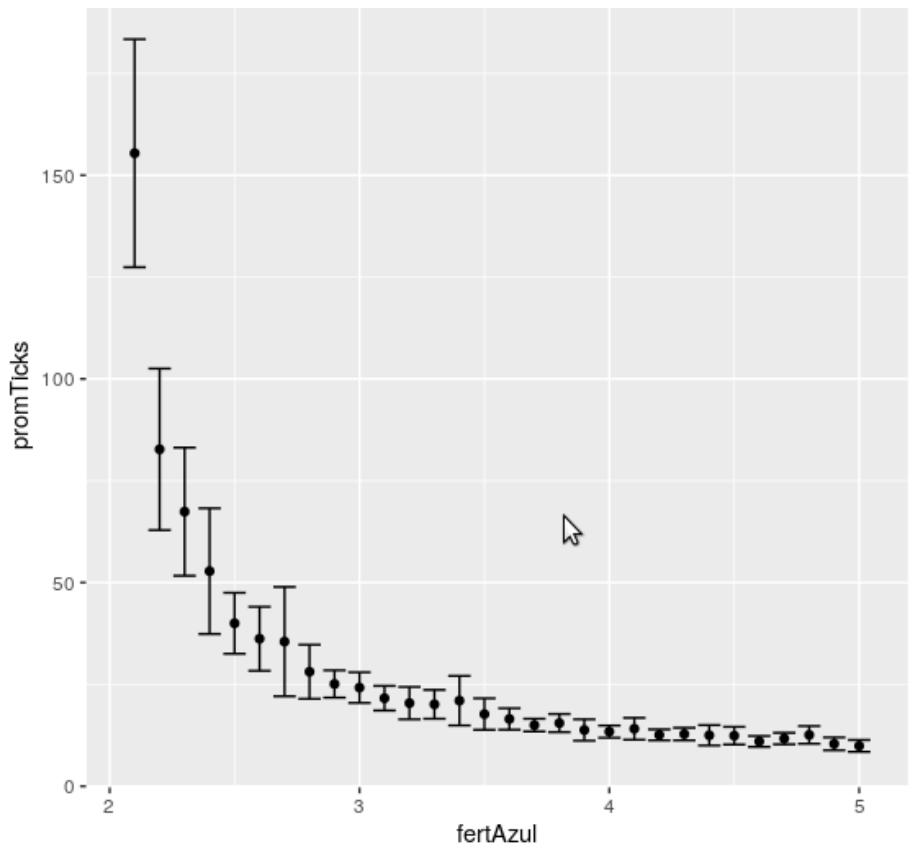


En la parte inferior derecha se observa el gráfico de fertilidad-azul vs Tiempo de permanencia de los agentes rojos

C.3. ANALIZANDO LOS RESULTADOS DEL EXPERIMENTO EN RSTUDIO309

Añadamos la desviación estándar a gráfica, coloque lo siguiente:

```
ggplot(datag)+  
  geom_point(aes(fertAzul,promTicks))+  
  geom_errorbar(aes(fertAzul,promTicks,ymax=promTicks + desvTicks, ymin=promTicks - desvTicks))
```



¿Este gráfico se ve como se esperaba? Si bien parece tranquilizador que el modelo produjo resultados que fueron exactamente los que esperábamos, también nos hace preguntarnos cuál es el valor de la construcción de un MOBA si produce resultados que podríamos anticipar fácilmente. De hecho, elegimos este modelo para examinar primero porque es un claro ejemplo de un MOBA cuya dinámica es fuertemente impuesta por sus comportamientos de agente muy simples y rígidos: los agentes no toman decisiones, no hay variables de estado individuales, y sus muy pocos comportamientos (reproducción, muerte) son estrictamente especificados, o impuestos, por los parámetros globales.

(El modelo también es no espacial porque las ubicaciones de los agentes no tienen ningún efecto en absoluto) Si los agentes en este modelo adaptan su fertilidad a (por ejemplo) la densidad local de otras tortugas o al alimento que puedan encontrar compañeros dentro de un cierto tiempo o distancia antes de

reproducirse, el modelo produciría resultados menos predecibles, más interesantes. Si bien es divertido jugar con este modelo y proporciona un buen ejercicio de uso del Analizador de Comportamiento, no es un buen ejemplo del tipo de problema que requiere un ABM para entender. Así que pasemos a un modelo que, en contraste, es un ejemplo clásico de dinámica emergente.

Appendix D

Bibliografía

- Arthur , W. B. (1994). Inductive reasoning and bounded rationality. *American Economic Review* , 84 (2), 406 – 411
- Axelrod , R. (1984). *The Evolution of Cooperation* . New York : Basic Books .
- Bak , P. (1996). *How Nature Works: The Science of Self-Organized Criticality* . New York : Springer .
- Barabási , A.-L. (2002). *Linked: The New Science of Networks* . Cambridge, MA : Perseus .
- Barabási, Albert-László (2004). *Linked: How Everything is Connected to Everything Else*. Plume, Cambridge, MA.
- Barabási, Albert-László and Albert, Réka (1999). Emergence of scaling in random networks. *Science* 286: 509-512.
- Broadbent , S. R. , & Hammersley , J. M. (1957). Percolation processes I. Crystals and mazes. *Proceedings of the Cambridge Philosophical Society* , 53 , 629 – 641 .
- Casti , J. L. (1995). Seeing the light at El Farol: A look at the most important problem in complex systems theory. *Complexity* , 1 (5), 7 – 10 .
- Gladwell , M. (2000). *The Tipping Point* . New York : Little, Brown .
- Goldbeter, A., and Segel, L. A. A977). “Unified Mechanism for Relay and Oscillation of Cyclic AMP in Dictyostelium Discoideum.” *Proc. of the National Academy of Sciences*, vol. 74, pp. 1543—1547.
- Grimm , V. , & Railsback , S. (2005). Individual-Based Modeling and Ecology . Princeton : Princeton University Press .
- Herman, R., and Gardels, K. A963). “Vehicular Traffic Flow.” *Scientific American*, vol. 209, no. 6 (December), pp. 35—43.

- Holland , J. (1975). Adaptation in Natural and Artificial Systems . Ann Arbor : University of Michigan Press
- Mitchell , M. (2009). Complexity: A Guided Tour . New York : Oxford University Press .
- Newman, M., Barabási, A.-L., Watts, D.J. [eds.] (2006). The Structure and Dynamics of Networks. Princeton, N.J.: Princeton University Press.
- Newman , M. , Girvan , M. , & Farmer , J. D. (2002). Optimal design, robustness, and risk aversion. *Physical Review Letters* , 89 (2), 028301-1- 4 .
- Prigogine, I., and Stengers, I. A984). Order out of Chaos: Man's New Dialogue with Nature. New York: Bantam Books.
- Resnick , M. (1994b). Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds . Cambridge, MA : MIT Press .
- Schelling , T. (1978). Micromotives and Macrobehavior . New York : Norton .
- Stauffer, D. and Aharony, A. (1994) Introduction to Percolation Theory. 2nd Edition, Chapter 7: Percolation and Thermal Phase Transitions. Taylor and Francis, London.
- Strogatz, Steven H. (2001). Exploring complex Networks. *Nature* 410: 268-276.
- Watts, Duncan J. and Strogatz, Steven H. (1998). Collective dynamics of 'small-world' networks. *Nature* 393: 440-442.
- Wilensky , U. , & Rand , W. (2007). Making models match: Replicating agent-based models. *Journal of Artificial Societies and Social Simulation* , 10 , 42 .
- Wolfram , S. (2002). A New Kind of Science . Champaign, IL : Wolfram Media