Práctica 1

Introducción a Agentes

1.1. Objetivo

Que el alumno se familiarice con la abstracción del concepto de agente mediante la programación de un sistema de simulación biológico que muestra las bases de un autómata celular, programación dirigida a agentes, sistemas complejos y emergencia de propiedades como la auto-organización.

1.2. Introducción

Un autómata celular es un modelo discreto que consiste en una cuadrícula de células, cada una con un número finito de estados. La cuadrícula puede estar en cualquier número finito de dimensiones pero lo más común es encontrar autómatas en una, dos y tres dimensiones para que tengan sentido geométrico. Cada célula tiene definido un conjunto de células llamado vecindad [Fig. 1.1] [Fig. 1.2].

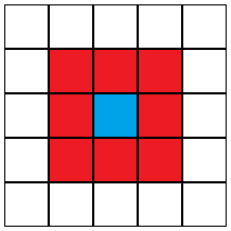

Figura 1.1: Vecindad de Moore

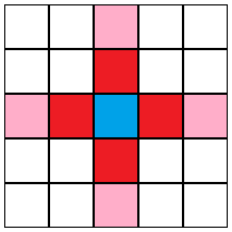

Figura 1.2: Vecindad de Von Neumann de radio 1 (rojo) y radio 2 (rojo y rosa).

Se tiene un estado inicial (al tiempo t=0) en el que se asigna un estado a cada célula. Una nueva generación es creada (avanzar t en 1) según alguna regla que determina el nuevo estado de cada célula en términos del estado actual de la célula y la de sus vecinos [Fig. 1.3].

1.2.1. Modelo basado en agentes

Un modelo basado en agentes es un tipo de modelo computacional que permite la simulación de acciones e interacciones entre individuos autónomos dentro de un entorno, y permite determinar qué efectos producen en el conjunto del sistema.

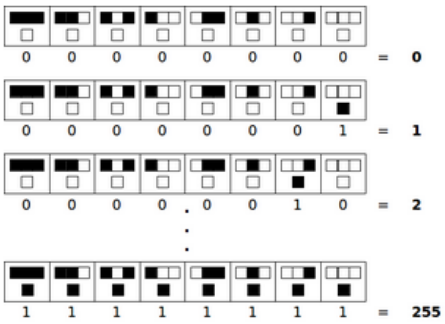

Figura 1.3: Numeración de reglas de transición para autómatas celulares unidimensionales.

Combina elementos de teoría de juegos, sistemas complejos, emergencia, sociología computacional, sistemas multi-agente y programación evolutiva. Los modelos simulan las operaciones simultáneas e interacciones de múltiples agentes, en un intento de recrear y predecir la apariencia de fenómenos complejos. El proceso de emergencia surge de un nivel bajo hacia niveles del sistema más altos. La clave es notar que reglas de comportamientos sencillos generan comportamientos complejos.

1.2.2. Agentes autónomos y auto-organización

Un agente autónomo es una unidad que interactúa con su entorno (el cual probablemente consta de otros agentes) pero actúa independientemente de todos los demás agentes porque no toma decisiones con respecto a algún líder o plan global a seguir. Es decir, cada agente actúa por sí mismo.

Así, veremos cómo múltiples agentes pueden desempeñar tareas que aparentan seguir un plan global. A este proceso en el que cada agente autónomo interactúa a su propia manera para crear un orden global se le conoce como auto-organización y se observa como modelos simples son capaces de generar comportamientos complejos.

1.3. Planteamiento

1.3.1. Modelo de termitas

Mitchel Resnick (1994) estudió varios sistemas de agentes primitivos, uno de ellos fueron las termitas teóricas dentro de un espacio con astillas esparcidas que seguían las siguientes reglas:

- Caminar aleatoriamente hasta encontrar una astilla.
- Si la termita está cargando una astilla, la suelta y continua caminando aleatoriamente.
- Si la termina no está cargando una astilla, la toma y continua caminando aleatoriamente con la astilla.

Claramente, las reglas definidas por Resnick son tan simples como es posible. No parece haber lugar para un comportamiento inteligente en un modelo como éste, tampoco

parece que las termitas puedan producir nada con algún sentido más allá de la aleatoriedad de las astillas distribuidas en el entorno.

La Figura 1.4 muestra seis escenarios de la simulación del conjunto de reglas simples con un conjunto pequeño de termitas. En la configuración inicial el universo de termitas consiste en una cuadrícula con astillas aleatoriamente distribuidas. La representación de la cuadrícula consta de una frontera periódica, es decir, un punto en una de las aristas de la cuadrícula tiene como vecinos a los puntos en la arista opuesta. Al comenzar la simulación, las termitas mueven las astillas en pequeños grupos o clusters. Conforme pasa el tiempo, los clusters se vuelven más grandes y más definidos.

Tras cientos de miles de pasos en la ejecución de la simulación, como se muestra en la última imagen de la Figura 4, las astillas están claramente bien definidas en una colección. Obviamente esto es un método poco óptimo para coleccionar astillas, sin mencionar lo frustrante que es observar el proceso. Sin embargo, con el paso del tiempo es un hecho que el orden del sistema es evidente como resultado.

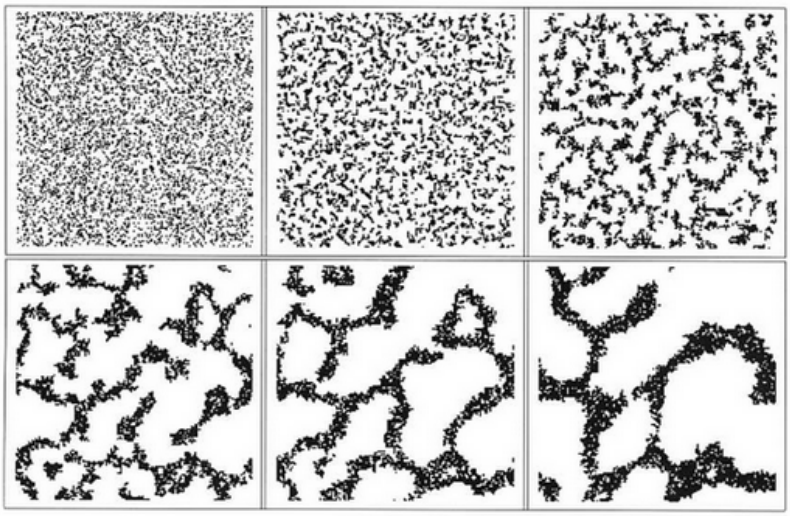

Figura 1.4: Termitas colocando aleatoriamente astillas con las reglas definidas anteriormente.

1.4. Desarrollo e implementación

Los resultados anteriores son reportados por el propio Resnick, sin embargo, para mostrar que es posible llegar al mismo resultado se lleva a cabo una implementación con el lenguaje de programación `Processing`. Al alumno se le proporciona parte de la implementación, de manera que se enfoque únicamente en programar el comportamiento descrito y evitando perder tiempo en la interfaz gráfica.

Las clases, métodos y variables más relevantes son las siguientes:

Clase Celda
Representación de cada espacio dentro de la cuadrícula donde estarán las termitas y astillas. Cada celda tiene coordenadas (x,y) y un valor booleano para indicar si hay una astilla.

Clase Termita
Representación de una termita. Se representan las coordenadas (x,y) de su posición, la dirección en la que está observando (auxiliar que más adelante será mencionado a detalle) y un valor booleano para indicar si está cargando una astilla.

Clase ModeloTermitas
Representación de una colonia de termitas. Principalmente contiene una matriz de celdas (la representación del mundo) y una lista de termitas (nuestros agentes). Adicionalmente se define la cantidad de celdas a lo ancho y alto, un valor auxiliar para conocer la cantidad de iteraciones, un objeto de la clase Random (para hacer decisiones aleatorias) y el tamaño en pixeles de cada celda (para la visualización con Processing).

1.4.1. Implementación

El constructor de la clase `ModeloTermitas` ya está implementado (principalmente para inicializar el espacio y termitas). También se encuentra implementado el método `moverTermita`, que mueve la termita dada como parámetro en la dirección indicada. Cada termita tiene una vecindad de Moore, es decir, tienen 8 celdas adyacentes (considerando un espacio periódico) que se enumeran según la Fig. 1.5

0	1	2
7		3
6	5	4

Figura 1.5: Las 8 posibles direcciones y vecindades de cada termita o celda.

De la misma manera se indica la dirección en la que puede mirar una termita. Por ejemplo, si la termita está mirando en dirección con valor 1 significa que está observando hacia arriba. Si tuviera el valor 4 significa que está observando en diagonal inferior derecha.

Existen 3 maneras diferentes de simular e implementar el modelo de termitas:

1. **Usando las 8 posibles direcciones**
   Siguiendo la idea original con caminatas aleatorias en las 8 posibles direcciones para las termitas.

2. **Modificar la caminata para que sea aleatoria y restringida**
   Una manera de avanzar totalmente aleatoria como en el primer caso implica que pueden existir muchos movimientos innecesarios (considerese la situación en la que una termita se cicla moviéndose en la casilla delante de ella y detrás de ella). Empleando la dirección en la que está observando la termita se puede restringuir

su movimiento a solo 3 opciones: a la izquierda, al frente o a la derecha. Adicionalmente al soltar una astilla, la termita da media vuelta y se coloca en la dirección opuesta de donde soltó la astilla (esto evita una situación similar en la que se cicle una termita moviendo la misma astilla al mismo lugar).

3. **Brindar un salto a las termitas**
   Esto significa que en el momento en que una termita suelta una astilla, en lugar de moverse en la dirección opuesta, las termitas "brincan" o se mueven a una celda sin astilla y continúan caminando de manera aleatoria y restringida.

Cada una de las diferentes maneras de implementación se encuentran asignadas a los métodos `evolucion1`, `evolucion2` y `evolucion3`, respectivamente.

El archivo `Termitas.pde` contiene parte del código de la simulación y solamente tiene implementado la visualización de las termitas como cuadrados verdes moviéndose aleatoriamente (Fig. 1.6). Al implementar todos los métodos faltantes se darán cuenta de que cuando una termita está cargando una astilla (cuadros amarillos) cambia de color a rojo.

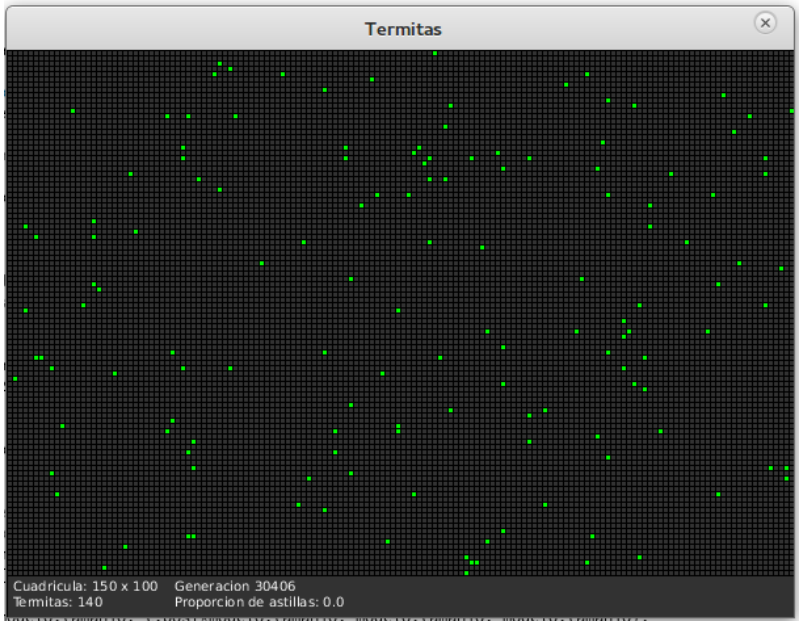

Figura 1.6: Captura de pantalla del código `Termitas.pde` de Processing.

Se debe implementar el comportamiento de las termitas para simular todo el sistema de la mejor manera. Dado que la interfaz gráfica está dada, solamente es necesario implementar los siguientes métodos:

- `int direccionAleatoriaFrente(int direccion)`
- `boolean hayAstilla(Termita t, int direccion)`
- `void dejarAstilla(Termita t, int direccion)`
- `void dejarAstilla(Termita t)`
- `void dejarAstillaConSalto(Termita t)`
- `void tomarAstilla(Termita t, int direccion)`

Cada método se encuentra especificado dentro del archivo `Termitas.java`.

1.5. Requisitos y resultados

Para evaluar y calificar la práctica es necesario que se implementen todos los métodos mencionados e indicados en el código, respetando implementar sólo lo que se pide (para evitar comportamientos extraños de la simulación). Es completamente válido utilizar bibliotecas adicionales si lo consideran necesario, así como la creación y uso de sus propios métodos auxiliares si lo desean.

Debe notarse una mejora significativa mediante la implementación de la caminata restringida o el uso del salto. Es decir, verifiquen que tras varias iteraciones su implementación del modelo actúe como se espera. Las siguientes imágenes ilustran parte de los resultados esperados [Fig. 1.7] [Fig. 1.8] [Fig. 1.9].

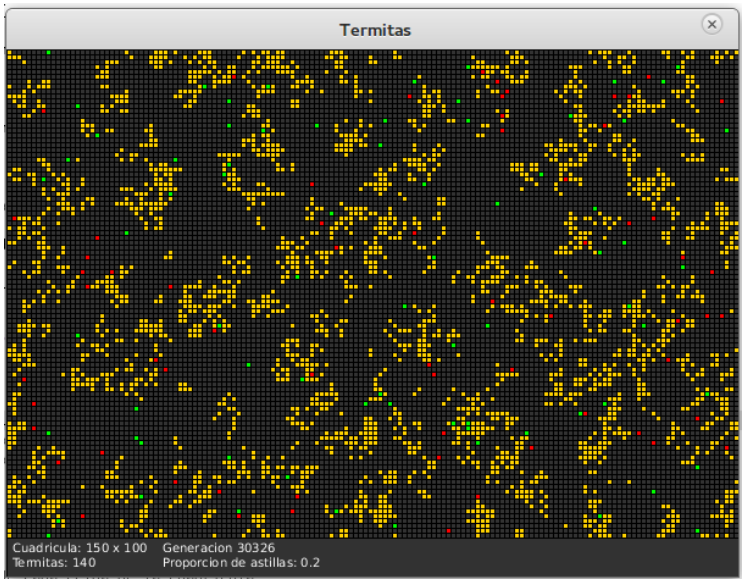

Figura 1.7: Simulación con idea original después de 10000 iteraciones (evolucion1).

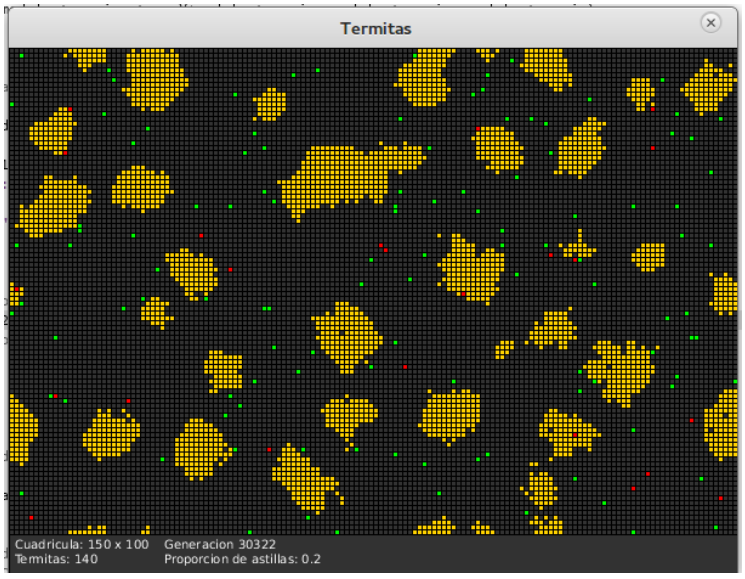

Figura 1.8: Empleando caminata aleatoria restringida tras 5000 iteraciones. La cantidad de astillas es la misma pero se observa un mejor ordenamiento y en menor tiempo.

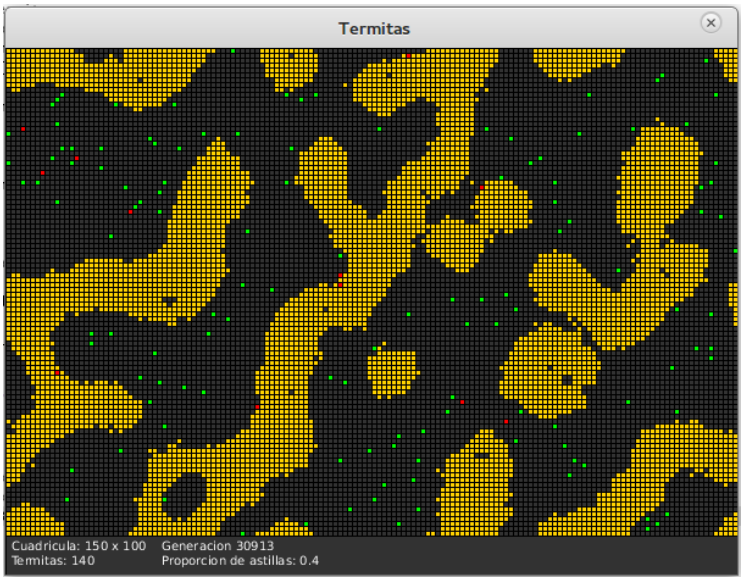

Figura 1.9: El uso de termitas con salto brinda una aproximación similar a la anterior. Cambiando algunos parámetros se pueden obtener resultados similares a colonias de termitas de la naturaleza.