**

**Autómatas Celulares**

***Game of Life***

72.25 - Simulación de Sistemas

Abril 2022

**Autores:**

Galende, Lautaro (60287)

Ratti, Valentín Segundo (60031)

**Docentes:**

Parisi, Daniel Ricardo

Patterson, German Agustin

Wiebke, Lucas

# Introducción

El objetivo de este trabajo práctico es implementar la simulación del modelo de autómatas celulares Game of Life, visto en clase. Se simulará el mismo para analizar comportamientos en base a distintos parámetros mediante tablas de resultados y animaciones.

# Fundamentos Teóricos

Un autómata celular es un modelo matemático para un sistema dinámico compuesto por un conjunto de celdas o células que adquieren distintos estados o valores. Estos estados son alterados de un instante a otro en unidades de tiempo discreto, es decir, que se puede cuantificar con valores enteros a intervalos regulares. De esta manera este conjunto de células logran una evolución según una determinada expresión matemática, que es sensible a los estados de las células vecinas, y que se conoce como regla de transición local.

# Implementación

Para la simulación de los diferentes tipos de autómatas celulares que se estudiaron, se realizó un proyecto en Java. Para el mismo, se crearon las clases Simulator, State, Rule, Initialization Grid, Grid. Muchas de ellas son clases abstractas que luego tienen una implementación particular dependiendo si son en 2D o 3D.

Las partículas ‘spawnean’ en una sub-grilla autocontenida dentro de la grilla de dimensión mayor. Una vez creado todo el estado inicial de la simulación, se comienzan a realizar múltiples iteraciones secuenciales del autómata estudiado hasta llegar a una iteración límite o a una situación de borde. En nuestro caso, estas son que una partícula llegue a un extremo de la grilla principal, o que no queden más partículas con vida.



Para calcular los estados intermedios, se utilizó la siguiente estrategia: en cada estado, se persiste una lista de las celdas que fueron modificadas, esto es, que pasaron de viva a muerta o de muerta a viva. Esto permite que para el siguiente estado, solo se chequeen las celdas que fueron modificadas en el último estado y los vecinos de ellas. Lo cual se debe a que una celda sólo puede modificar su estado respecto al estado anterior si alguna de sus vecinas cambió.

# Simulaciones

Se generarán celdas vivas en un dominio central y acotado. Es decir, lejos de los bordes y con una cantidad de celdas vivas considerablemente menor a la cantidad total de celdas. Las posiciones de dichas celdas se generan en posiciones aleatorias dentro del dominio inicial (subgrilla).

La transición del sistema se realizará siguiendo una determinada regla. Las reglas posibles se detallarán más adelante.

Se utilizarán como criterios de corte el hecho de que una celda viva esté en el borde de la grilla, que se supere el número máximo de iteraciones recibido por parámetro, o que no queden celdas vivas.

Parámetros

* Cantidad inicial de celdas vivas: se calcula a partir del input (porcentaje inicial de celdas vivas) y de la dimensión.
* Dimensión de la grilla: cantidad de celdas, equivalente a la dimensión al cuadrado.
* Número máximo de iteraciones: se utilizará como criterio de corte en caso que ninguna celda viva llegue al borde.
* Regla a utilizar: hay 6 posibles, se detallarán más adelante.
* Dimensión del dominio inicial: subgrilla donde se generarán las celdas vivas inicialmente.

Input

Porcentaje de celdas vivas dentro del dominio inicial. Se tomarán 6 valores en el intervalo (0, 100%].

Output

Se define como observable el tiempo de saturación del sistema, medido en cantidad de iteraciones.

Se define como métrica el porcentaje de celdas vivas vs el número de iteraciones. El primero se calcula como el número de celdas vivas dividido el número total de celdas.

El programa Java exporta archivos en formato .xyz los cuales funcionan como input en el software de visualización Ovito. En el mismo archivo se proveen las posiciones xyz y la distancia al centro de cada celda, la cual se utiliza para colorear las partículas. Lo cual se puede observar en las figuras correspondientes a las simulaciones en la sección Resultados y Conclusiones.

Además, para cada input y para cada sistema se generará un archivo csv con 3 columnas: regla, cantidad total de iteraciones y porcentaje inicial de celdas vivas. Por ejemplo, para 6 sistemas y 6 valores en el input, el csv tendrá 36 líneas (sin contar el header).

Reglas

Se definieron 6 reglas, de las cuales las primeras 3 son para sistemas 2D y las restantes para sistemas 3D.

1. Conway’s Game of Life: 2-3/3/2/M
2. Rule B: 4-9/6/2/M (r=2)
3. Rule C: 3/2-3/2/M
4. Builder 2: 5-7/1/2/M
5. Architecture: 4-6/3/2/M
6. Clouds 2: 13-26/13-14/2/M

Tomaremos como ejemplo la primera regla para explicar la notación.

Si una celda está viva, se mantendrá en este estado si tiene entre 2 y 3 vecinos vivos. De lo contrario morirá.

En cambio, si una celda está muerta, se transformará en viva si tiene exactamente 3 vecinos vivos.

El tercer parámetro indica la cantidad de estados posibles. En este caso es 2 para todas las reglas: viva o muerta.

Por otro lado, para todas las reglas se utiliza vecindad de Moore con radio 1 para calcular los vecinos, con excepción de la segunda que se utiliza radio 2.

# Resultados y conclusiones

Los sistemas estudiados se corresponden con las reglas definidas en la sección anterior. En otras palabras, son 6 los sistemas estudiados: tres 2D y tres 3D. Además, los inputs utilizados para las simulaciones son los siguientes: 0,5; 3; 10; 30; 40; 45.

El escalar definido como observable es el número de iteraciones hasta que se cumple el criterio de corte, ya sea la saturación del sistema, que no haya más celdas vivas o que se llegue al límite de iteraciones.

|  |
| --- |
| Figura 1: Sistema 1. Conway’s Game of Life. |

| Gráfico |
| --- |
| Figura 2: Sistema 1. Número de iteraciones vs % de celdas vivas. |

| Gráfico |
| --- |
| Figura 3: Sistema 1. Input vs observable. |

|  |
| --- |
| Figura 4: Sistema 2. |

| Gráfico |
| --- |
| Figura 5: Sistema 2. Número de iteraciones vs % de celdas vivas. |

| Gráfico |
| --- |
| Figura 6: Sistema 2. Input vs observable. |

|  |
| --- |
| Figura 7 : Sistema 3. |

| Gráfico |
| --- |
| Figura 8: Sistema 3. Número de iteraciones vs % de celdas vivas. |

| Gráfico |
| --- |
| Figura 9: Sistema 3. Input vs observable. |

|  |
| --- |
| Figura 10: Sistema 4. Builder 2. |

# 

| Gráfico |
| --- |
| Figura 11: Sistema 4. Número de iteraciones vs % de celdas vivas. |

# 

| Gráfico |
| --- |
| Figura 12: Sistema 4. Input vs observable. |

|  |
| --- |
| Figura 13: Sistema 5. Architecture. |

| Gráfico |
| --- |
| Figura 14: Sistema 5. Número de iteraciones vs % de celdas vivas. |
| Gráfico |
| Figura 15: Sistema 5. Input vs observable. |
|  |

# 

|  |
| --- |
|  |
| Figura 16: Sistema 6. Clouds 2. |

# 

| | Gráfico | | --- | | Figura 17: Sistema 6. Número de iteraciones vs % de celdas vivas. | |
| --- | --- | --- |
| | Gráfico | | --- | | Figura 18: Sistema 6. Input vs observable. | |

# Conclusiones

1. Sistema 1

Tienden a estado estacionario con un % menor al 5. Cuando el % inicial es mayor, llega rápido al borde.

1. Sistemas 2 y 3

Si se inicia con un alto % de celdas vivas, este baja bruscamente tras la muerte de muchas celdas. Luego, crece el % de celdas vivas y rápidamente se cumple que llegan al borde.

En cambio, si se inicia con poca cantidad de celdas vivas, crece casi linealmente hasta cumplir la condición de corte.

1. Sistemas 4 y 5

Los casos que inician con un % inicial de celdas vivas alto, luego de pocas iteraciones, tienen menos % de celdas vivas que los que iniciaron con un % bajo. Sin embargo, logran llegar antes al borde.

1. Sistema 6

Sin importar el % inicial de celdas vivas, ante todos los inputs se llega cumplir la condición de corte debido que no quedan celdas vivas.

1. General

En los primeros 5 sistemas, la cantidad de iteraciones decrece a medida que se aumenta el % inicial de celdas vivas. En cambio, en el sexto sucede lo contrario.

## Referencias

Junto con el material proporcionado por la cátedra, se utilizó la siguiente bibliografía:

[3D Cellular Automata | Softology's Blog](https://softologyblog.wordpress.com/2019/12/28/3d-cellular-automata-3/)