Programación y Prototipado

# El juego de la vida de Conway

Universidad de Las Palmas de Gran Canaria

Máster SIANI

Leopoldo Lopez Reveron

Repositorio: Conway_game_matlab

Fecha de entrega: 25/01/2022

El juego de la vida se basa en la premisa de las propiedades emergente, lo que supone que a partir de reglas simples surgen comportamientos complejos que son difícilmente previsibles en primera instancia.

En concreto las reglas designadas en el juego de Conway son:

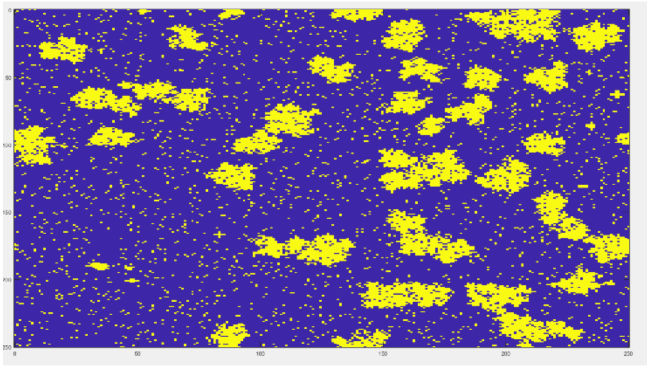
- Una célula muerta con exactamente 3 células vecinas vivas "nace" (es decir, al turno siguiente estará viva).

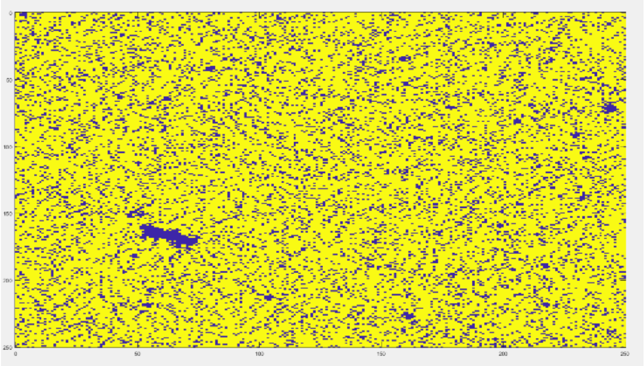
- Una célula viva con 2 o 3 células vecinas vivas sigue viva, en otro caso muere (por "soledad" o "superpoblación").

Con estas dos reglas sencillas se crean comportamientos complejos.

Para la simulación del escenario descrito se ha utilizado Matlab como lenguaje base del prototipo, como base del problema se utilizó una matriz binaria para representar el estado del tablero de las células vivas o muertas, además, durante el desarrollo del prototipo se partió de dos funciones separadas para la actualización del tablero y la actualización de los vecinos de las células.

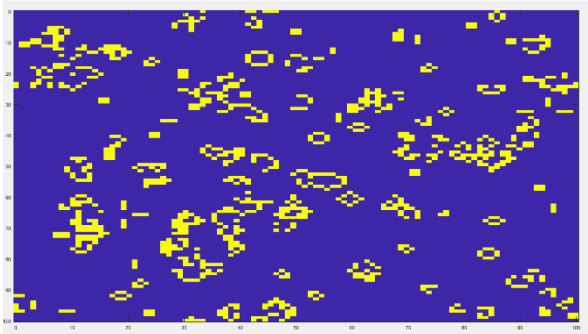
En la implementación de las reglas se cometieron errores dando resultados inesperados, en concreto que se malinterpreto la regla “Una célula viva con 2 o 3 células vecinas **vivas sigue viva**, en otro caso muere (por "soledad" o "superpoblación").” De forma que “Una célula viva con 2 o 3 células vecinas **vivas sigue muere**”, este cambio supuso que el sistema se volviera de crecimiento en donde la “corrupción se comía el tablero”, además se quedaban pequeños osciladores atrapados dentro de las cavidades

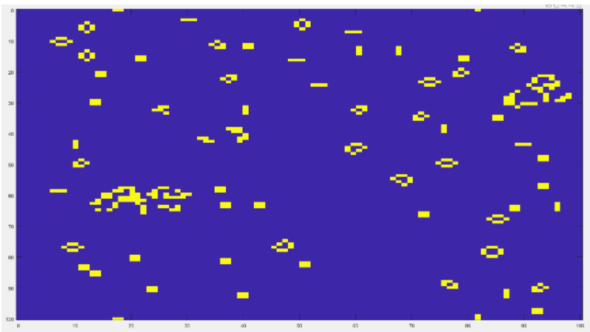




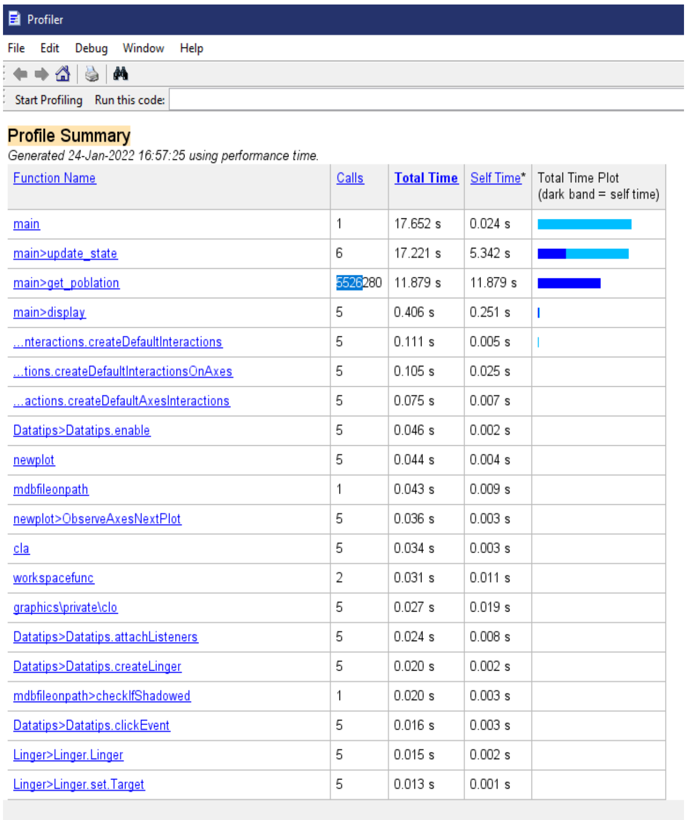
Luego se corrigió esta regla consiguiendo el resultado esperado, dentro de la simulación se observaron distintas estructuras:

- osciladores estructuras cíclicas en el tiempo de simulación
- vidas estáticas en el tiempo
- vidas móviles que se desplazan por el tablero
- Generadores de vidas, que ya sea a su paso o de manera estática producen vidas estáticas o móviles





Durante la simulación se observó una gran limitación a la hora de ampliar la dimensión del problema, resultando en una simulación lenta en espacios de tableros grandes. Como podemos ver el profile de rendimiento de Matlab. La función con mayor carga durante la ejecución es Get_poblation() que se encarga de obtener los vecinos más próximos a una célula, esto se debe a que se utilizan 4 for anidados, 2 para recorrer el tablero y dos para obtener los vecinos mas cercanos a una célula.




**Profile Summary**

*Generated 24-Jan-2022 16:57:25 using performance time.*

| Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| main | 1 | 17.652 s | 0.024 s | |
| main>update_state | 6 | 17.221 s | 5.342 s | |
| main>get_poblation | 5526280 | 11.879 s | 11.879 s | |
| main>display | 5 | 0.406 s | 0.251 s | |
| ...nteractions.createDefaultInteractions | 5 | 0.111 s | 0.005 s | |
| ...tions.createDefaultInteractionsOnAxes | 5 | 0.105 s | 0.025 s | |
| ...actions.createDefaultAxesInteractions | 5 | 0.075 s | 0.007 s | |
| Datatips>Datatips.enable | 5 | 0.046 s | 0.002 s | |
| newplot | 5 | 0.044 s | 0.004 s | |
| mdbfileonpath | 1 | 0.043 s | 0.009 s | |
| newplot>ObserveAxesNextPlot | 5 | 0.036 s | 0.003 s | |
| cla | 5 | 0.034 s | 0.003 s | |
| workspacefunc | 2 | 0.031 s | 0.011 s | |
| graphics\private\clo | 5 | 0.027 s | 0.019 s | |
| Datatips>Datatips.attachListeners | 5 | 0.024 s | 0.008 s | |
| Datatips>Datatips.createLinger | 5 | 0.020 s | 0.002 s | |
| mdbfileonpath>checkIfShadowed | 1 | 0.020 s | 0.003 s | |
| Datatips>Datatips.clickEvent | 5 | 0.016 s | 0.003 s | |
| Linger>Linger.Linger | 5 | 0.015 s | 0.002 s | |
| Linger>Linger.set.Target | 5 | 0.013 s | 0.001 s | |

Para resolverlo se sustituyó la función Get_poblation() por una convolución 2-D que nos da una matriz(n-1, m-1) del tablero donde tenemos las poblaciones de cada célula. El kernel que se utilizo fue:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$$

de esta forma obtenemos el número de células vivas que tenemos alrededor de la célula sin contarla a ella misma. Con este cambio obtuvimos un orden de magnitud mayor de simulación en tiempos similares.

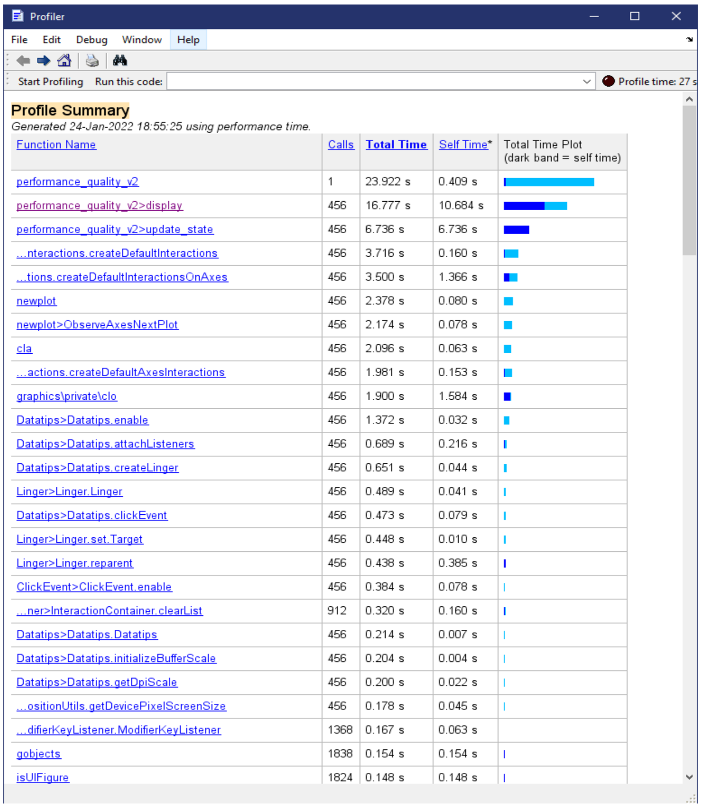

**Profile Summary**

*Generated 24-Jan-2022 18:20:58 using performance time.*

| Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| performance_quality | 1 | 17.622 s | 0.080 s | |
| performance_quality>update_state | 77 | 14.551 s | 14.551 s | |
| performance_quality>display | 76 | 2.991 s | 1.846 s | |
| ...nteractions.createDefaultInteractions | 76 | 0.666 s | 0.028 s | |
| ...tions.createDefaultInteractionsOnAxes | 76 | 0.628 s | 0.232 s | |
| newplot | 76 | 0.479 s | 0.013 s | |
| newplot>ObserveAxesNextPlot | 76 | 0.448 s | 0.012 s | |
| cla | 76 | 0.436 s | 0.011 s | |
| graphics\private\clo | 76 | 0.404 s | 0.258 s | |
| ...actions.createDefaultAxesInteractions | 76 | 0.368 s | 0.029 s | |
| Datatips>Datatips.enable | 76 | 0.252 s | 0.006 s | |
| ...ner>InteractionContainer.clearList | 152 | 0.147 s | 0.025 s | |
| Datatips>Datatips.attachListeners | 76 | 0.127 s | 0.040 s | |
| Datatips>Datatips.createLinger | 76 | 0.118 s | 0.009 s | |
| Datatips>Datatips.delete | 76 | 0.111 s | 0.011 s | |
| Linger>Linger.delete | 76 | 0.099 s | 0.002 s | |
| timer.timer>timer.delete | 1 | 0.091 s | 0.003 s | |
| Linger>Linger.Linger | 76 | 0.089 s | 0.007 s | |
| Datatips>Datatips.clickEvent | 76 | 0.087 s | 0.014 s | |
| timercb | 1 | 0.084 s | 0.001 s | |
| com.mathworks.timer.TimerTask (Java method) | 6 | 0.083 s | 0.083 s | |
| Linger>Linger.set.Target | 76 | 0.081 s | 0.002 s | |
| Linger>Linger.reparent | 76 | 0.080 s | 0.071 s | |
| ClickEvent>ClickEvent.enable | 76 | 0.071 s | 0.017 s | |
| opentoline | 1 | 0.040 s | 0.000 s | |

Sin embargo, este rendimiento se puede mejorar todavía más, en esta ocasión se cambio el doble bucle para recorrer el tablero por selecciones y multiplicaciones de matrices, aprovechando las características binarias de la matriz de representación.

Y una vez mas se mejoro el rendimiento un orden de magnitud adicional en tiempo de simulación similar.




**Profile Summary**

*Generated 24-Jan-2022 18:55:25 using performance time.*

| Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| performance_quality_v2 | 1 | 23.922 s | 0.409 s | |
| performance_quality_v2>display | 456 | 16.777 s | 10.684 s | |
| performance_quality_v2>update_state | 456 | 6.736 s | 6.736 s | |
| ...nteractions.createDefaultInteractions | 456 | 3.716 s | 0.160 s | |
| ...tions.createDefaultInteractionsOnAxes | 456 | 3.500 s | 1.366 s | |
| newplot | 456 | 2.378 s | 0.080 s | |
| newplot>ObserveAxesNextPlot | 456 | 2.174 s | 0.078 s | |
| cla | 456 | 2.096 s | 0.063 s | |
| ...actions.createDefaultAxesInteractions | 456 | 1.981 s | 0.153 s | |
| graphics\private\clo | 456 | 1.900 s | 1.584 s | |
| Datatips>Datatips.enable | 456 | 1.372 s | 0.032 s | |
| Datatips>Datatips.attachListeners | 456 | 0.689 s | 0.216 s | |
| Datatips>Datatips.createLinger | 456 | 0.651 s | 0.044 s | |
| Linger>Linger.Linger | 456 | 0.489 s | 0.041 s | |
| Datatips>Datatips.clickEvent | 456 | 0.473 s | 0.079 s | |
| Linger>Linger.set.Target | 456 | 0.448 s | 0.010 s | |
| Linger>Linger.reparent | 456 | 0.438 s | 0.385 s | |
| ClickEvent>ClickEvent.enable | 456 | 0.384 s | 0.078 s | |
| ...ner>InteractionContainer.clearList | 912 | 0.320 s | 0.160 s | |
| Datatips>Datatips.Datatips | 456 | 0.214 s | 0.007 s | |
| Datatips>Datatips.initializeBufferScale | 456 | 0.204 s | 0.004 s | |
| Datatips>Datatips.getDpiScale | 456 | 0.200 s | 0.022 s | |
| ...ositionUtils.getDevicePixelScreenSize | 456 | 0.178 s | 0.045 s | |
| ...difierKeyListener.ModifierKeyListener | 1368 | 0.167 s | 0.063 s | |
| gobjects | 1838 | 0.154 s | 0.154 s | |
| isUIFigure | 1824 | 0.148 s | 0.148 s | |

Apéndice:

A continuación, se adjunta el código desarrollado y sus respectivas versiones

# “Corrupción” juego de la vida

```
n = 250;
t_0 = zeros(n, n);
t_1 = randi([0 1], n, n);

while (true)

    [t_0, t_1] = update_state(t_0, t_1, n);
    display(n, t_1);

end

function display(n, C)
    x = [0 n];
    y = [0 n];
    image(x, y, C .* 255)
    drawnow
end

function poblation = get_poblation(t, i, j, n)

    poblation = 0;
    for i_inner = i-1:i+1
        for j_inner = j-1:j+1
            poblation = poblation + t(mod(i_inner - 1, n) + 1 ,
mod(j_inner - 1, n) + 1);
        end
    end
end

function [t_0, t_1] = update_state(t_0, t_1, n)
    for i = 1:n
        for j = 1:n
            poblation = get_poblation(t_1, i + n, j + n, n) -
t_1(i, j);

            if (t_1(i, j) == 0)
               if (poblation == 3)
                    t_0(i, j) = 1;
               end
            else
                if (poblation == 2 || poblation == 3)
                    t_0(i, j) = 0;
                end
            end
        end
    end
    t_1 = t_0;
end
```

# # Implementación Naif juego de la vida

```
n = 100;
t_0 = zeros(n, n);
t_1 = randi([0 1], n, n);
pause('on');

while (true)

    [t_0, t_1] = update_state(t_0, t_1, n);
    display(n, t_1);

end


function display(n, C)
    x = [0 n];
    y = [0 n];
    image(x, y, C .* 255)
    drawnow
    pause(0.03);
end


function poblation = get_poblation(t, i, j, n)

    poblation = 0;
    for i_inner = i-1:i+1
        for j_inner = j-1:j+1
            poblation = poblation + t(mod(i_inner - 1, n) + 1 ,
mod(j_inner - 1, n) + 1);
        end
    end
end


function [t_0, t_1] = update_state(t_0, t_1, n)
    for i = 1:n
        for j = 1:n
            poblation = get_poblation(t_1, i + n, j + n, n) -
t_1(i, j);

            if (t_1(i, j) == 0)
               if (poblation == 3)
                    t_0(i, j) = 1;
               end
            else
                if (poblation ~= 2 && poblation ~= 3)
                    t_0(i, j) = 0;
                end
            end
        end
    end
    t_1 = t_0;
end
```

# Implementación Convolucional 2D juego de la vida

```
n = 1000;
t = randi([0 1], n, n);
pause('on');

mask = [1 1 1; 1 0 1; 1 1 1];

while (true)

    t = update_state(t, n, mask);
    display(n, t);
end

function display(n, C)
    x = [0 n];
    y = [0 n];
    image(x, y, C .* 255)
    drawnow
end

function t = update_state(t, n, mask)
    poblations = conv2(t, mask, 'valid');

    for i = 2:n-1
        for j = 2:n-1
            if (t(i, j) == 0)
               if (poblations(i - 1, j - 1) == 3)
                    t(i, j) = 1;
               end
            else
                if (poblations(i - 1, j - 1) ~= 2
&& poblations(i - 1, j - 1) ~= 3)
                    t(i, j) = 0;
                end
            end
        end
    end
end
```

# Implementación Convolucional 2D + multiplicación de matrices juego de la vida

```
n = 1000;
t = randi([0 1], n, n);
pause('on');

mask = [1 1 1; 1 0 1; 1 1 1];


while (true)

    t = update_state(t, mask);
    display(n, t);
end


function display(n, C)
    x = [0 n];
    y = [0 n];
    image(x, y, C .* 255)
    drawnow
end


function t = update_state(t, mask)
    poblations = conv2(t, mask, 'same');

    t_0 = t == 0;
    t_0 = poblations .* t_0 == 3;

    t_1 = t == 1;
    poblations_2 = poblations == 2;
    poblations_3 = poblations == 3;
    poblations_2_3 = poblations_2 + poblations_3;

    t_1 = t_1 .* poblations_2_3;

    t = t_0 + t_1;

end
```