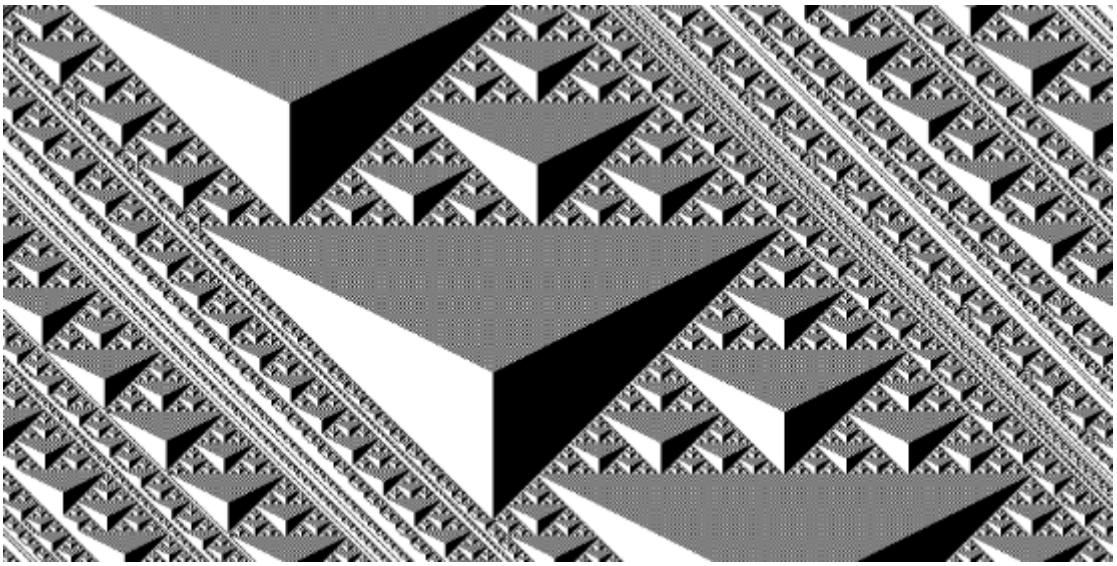

Especificación: CAL 9000

Cellular Automata Language 9000

Diseño e Implementación de un Lenguaje

v1.0.0

1. Equipo

Nombre	Apellido	Legajo	E-mail
Nicolas	Casella	62.311	ncasella@itba.edu.ar
Micaela	Perillo	62.625	miperillo@itba.edu.ar
Timoteo	Smart	62.844	tsmart@itba.edu.ar

2. Repositorio

La solución y su documentación serán versionadas en: **CAL 9000**.

3. Dominio

Desarrollar un lenguaje que permita generar autómatas celulares en una y dos dimensiones. El lenguaje debe permitir especificar las reglas que cada célula debe seguir (condiciones para cada estado, color); al igual que las reglas globales de la simulación (condición de frontera, tiempo transcurrido entre cada estado).

El proyecto buscará abstraer lo máximo posible la generación visual de los autómatas celulares. Para ello se proporcionará al usuario un lenguaje con formato similar a Python.

Se brindarán dos modalidades distintas de uso del lenguaje. La primera consistirá en especificar una regla predefinida de vecindad, o bien una especificada por el usuario. utilizando una nomenclatura provista por el lenguaje. Por otro lado, la segunda le permitirá al usuario dibujar los autómatas celulares elementales.

El lenguaje generará una salida en *Python*, utilizando la librería *PyGame* para poder visualizar el autómata y que el usuario interactúe con el mismo en caso de que sea en dos dimensiones.

La nomenclatura para los autómatas en dos dimensiones será la siguiente: **A/D/B/V**, en donde cada letra significa

- A: Cantidad de vecinos vivos para que la célula siga viva, sino muere de soledad
- D: Cantidad de vecinos vivos para que la célula muera por sobrepoblación
- B: Cantidad de vecinos vivos para que una célula nazca en una celda vacía
- V: Regla de vecindad, podra ser Moore (M), Von Neumann (VN) o creada por el usuario (U) y el rango especificado. Por ejemplo. M(2) es la distancia de Moore con rango dos.

La extensión de los archivos será *.cal9000*.

4. Construcciones

El lenguaje desarrollado debería ofrecer las siguientes construcciones, prestaciones y funcionalidades:

(I). Se podrán crear autómatas celulares en 1D con reglas predefinidas según los autómatas celulares elementales.
(II). Se podrán crear autómatas celulares en 2D con reglas definidas por el usuario.
(III). Se podrá crear por default el Game of Life de Conway en 2D.
(IV). Se podrá crear por default la Regla 30 en 1D.
(V). Se podrá utilizar un estado inicial predefinido en 2D.
(VI). Se podrá utilizar un estado inicial customizado en 2D.
(VII). Se podrán definir los colores utilizados para el *output*.
(VIII). Se proveerán el objeto: *Simulation*.
(IX). Se podrá utilizar la Vecindad de Moore con rango customizable.
(X). Se podrá utilizar la Vecindad de Von Neumann con rango customizable.

(XI). Se podrán definir vecindades customizables.
(XII). Se proveerán tipos de datos *built-in*: INT, BOOLEAN.

5. Casos de Prueba

Se proponen los siguientes casos iniciales de prueba de **aceptación**:

(I). Un programa que simule una regla tipo 3/4/3/M(1) en una grilla de 100 x 100.
(II). Un programa que simule una regla tipo 3/4/3/M(3) en una grilla de 300 x 300.
(III). Un programa que simule una regla tipo 3/4/3/N(2) en una grilla de 200 x 200.
(IV). Un programa que simule una regla 30 en una grilla de 100 x 100.
(V). Un programa que simule una regla 25 en una grilla de 100 x 100.
(VI). Un programa que simule una regla tipo 3/4/3 con la vecindad creada por el usuario en una grilla de 100 x 100.
(VII). Un programa que usa el default para una dimensión, Regla 30 en una grilla de 200 x 200.
(VIII). Un programa que usa el default para dos dimensiones, el *Game of Life* en una grilla de 200 x 200.
(IX). Un programa que utilice el Game of Life con una vecindad creada por el usuario en una grilla de 200 x 200.
(X). Un programa que implemente la regla 3/4/3/M(2) con colores customizados según la grilla, la misma de 100 x 100.

Además, los siguientes casos de prueba de **rechazo**:

(I). Un programa malformado.
(II). Un programa sin función *Initialize* definida.
(III). Un programa que no redefine.
(IV). Un programa que utilice una regla predefinida que no existe.
(V). Un programa que utilice el formato A/D/B/V, pero donde D es menor a A o B.
(VI). Un programa que haga *check* en una coordenada *out of bounds*.

6. Ejemplos

Crear un autómata con una regla 3/4/3/U, definida por el usuario

```
simulation Automata over (100,100) grid:
     setNeighborhoodRule(customUserRule);
     setBidimensionalRule(3,4,3,U);
     setColor(CYCLIC, Blue, Red, Yellow);
     setGridColors(Black, White); # Black background, white lines
simulation't
# User defined grid does not get a range parameter
```

```
rule customUserRule over (3,3) grid:
     check(x+1,y);
     check(x-1,y-1);
     check(x+2, y-1);
rulen't
```

Crear un autómata 1D con la regla 29

```
simulation Automata over (100,100) grid:
    setElementalRule(29);
    setColor(COORDINATE, Red, Green, Blue);
    setGridColors(Black, White);
simulation't
```