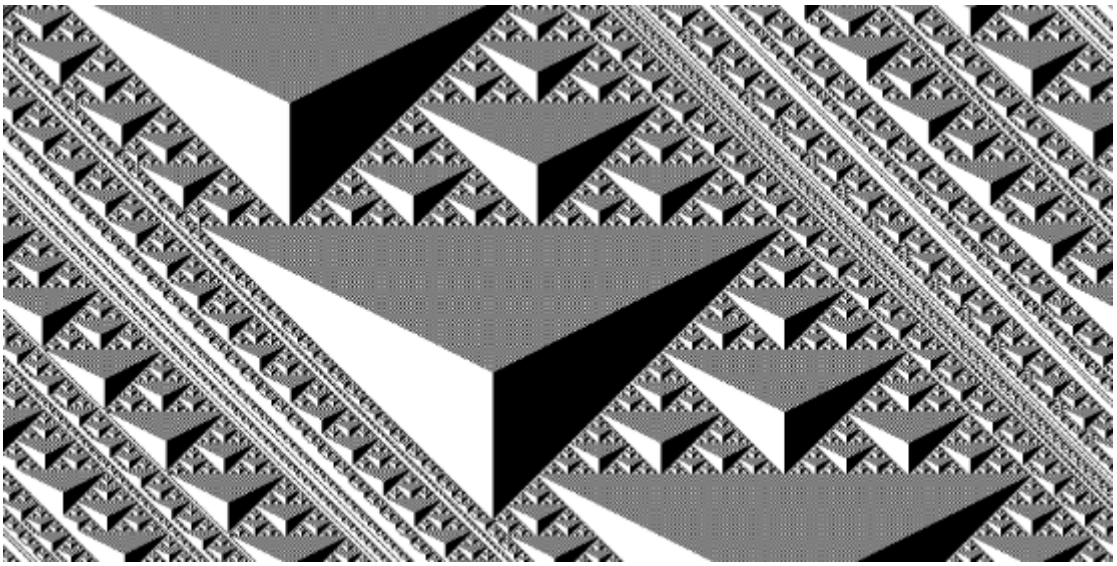

Especificación: CAL 9000

Cellular Automata Language 9000

Diseño e Implementación de un Lenguaje

`v1.2.0`

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**, 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

La extensión de los archivos será *.cal9k*.

4. Construcciones

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

(I). Se podrán crear autómatas celulares en 2D con reglas definidas por el usuario.
(II). Se podrán modificar propiedades de la simulación.

(III). Se podrán definir los colores utilizados para el *output*.
(IV). Se proveerán dos secciones distintas del código: *automata* y *rule*.
(V). Se podrán definir vecindades.
(VI). Se proveerán tipos de datos *built-in*: INT, BOOLEAN y COLOR (representación hexadecimal).

5. Casos de Prueba

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

(I). Un programa que define primero *automata* y después *rule*.
(II). Un programa que define primero *rule* y despues *automata*.
(III). Un programa que solo define un *automata*.
(IV). Un programa que simule un autómata tipo 1/2/1 en una grilla de 10 x 10 con 1 chequeo positivo y 1 regla con 1 parámetro (int).
(V). Un programa que simule una regla tipo 5/6/3 en una grilla de 100 x 100 con 6 chequeos negativos y 5 reglas con hasta 3 parámetros (int, boolean, color).
(VI). Un programa que simule una regla tipo 8/9/7 en una grilla de 500 x 500 con 9 chequeos positivos/negativos y 7 reglas con hasta 3 parámetros o más (int, boolean, color).
(VII). Un programa con líneas vacías dentro de *automata* y *rule*.
(VIII). Un programa con espacios extra entre cada string.
(IX). Un programa que utilice comentarios multilínea dentro de *automata* y *rule*.
(X). Un programa que utilice comentarios multilínea fuera de *automata* y *rule*.
(XI). Un programa que crea una grilla con x distinto de y.

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

(I). Un programa con capitalización incorrecta.
(II). Un programa sin una función *automata* definida.
(III). Un programa que no define chequeos.
(IV). Un programa vacío.
(V). Un programa que define dos funciones *rule*.
(VI). Un programa que define dos funciones *automata*.
(VII). Un programa con un comentario sin cerrar.
(VIII). Un programa con una propiedad inexistente.
(IX). Un programa con un parámetro inválido.
(X). Un programa con una propiedad con cantidad de parámetros inválida.
(XI). Un programa con un parámetro inválido incluído en una lista de parámetros válidos.
(XII). Un programa con una propiedad sin parámetros.
(XIII). Un programa con número de autómata ilegal.
(XIV). Un programa con una grilla vacía.
(XV). Un programa con checks afuera de la grilla.
(XVI). Un programa con múltiples parámetros que no están separados por una coma.
(XVII). Un programa con propiedades duplicadas.

(XVIII). Un programa con un número de grilla negativo.
(XIX). Un programa con número de autómata negativo (A).
(XX). Un programa con número de autómata negativo (D).
(XXI). Un programa con número de autómata negativo (B).

6. Ejemplo

Crear un autómata con una regla 3/4/3, con propiedades definidas por el usuario

```
/*
  Custom automata 3/4/3
*/

automata (3,4,3), grid (100,100):
     check(1,0);
     check(-1,-1);
     check(2,-1);
     check(-2,0);
automatan't

rule:
     prop:color = (#FFFFFF, #000FFD, #12DCFF);
     prop:bg_color = (#000000, #FFFFFF);
     prop:wrapping = (true);
rulen't
```