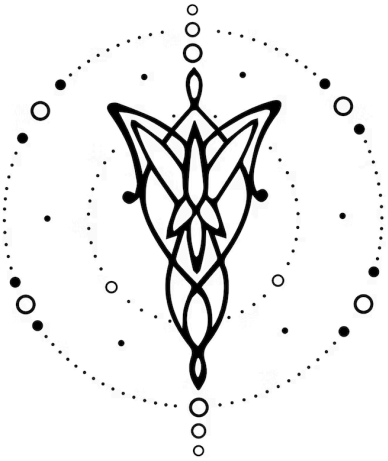

Especificación

Diseño e Implementación de un Lenguaje

v1.0.0

1. Equipo

Nombre	Apellido	Legajo	E-mail
Filipo	Ardenghi	64306	FILIPO ARDENGHI
Agustín Tomás	Romero	64268	AGUSTÍN TOMÁS ROMERO
Federico Ignacio	Ruckauf	64356	FEDERICO IGNACIO RUCKAUF
Pedro Salinas	Salinas	64388	PEDRO SALINAS

2. Repositorio

La solución y su documentación serán versionadas en: **Repositorio**.

3. Dominio

Desarrollar un lenguaje que permita definir configuraciones de autómatas celulares de dos dimensiones, que luego serán ejecutados con una interfaz gráfica interactiva. El lenguaje debe permitir definir la función de vecindario y la regla de evolución o, en su defecto, una función "transición" que englobaría a estas dos y daría lugar a mayor complejidad. Además debe permitir definir las dimensiones, los colores, la cantidad de estados y el tipo de fronteras del autómata.

Se podrán definir varios estados para las células, asignándoles colores y reglas específicas para su transición. Si se definen las funciones vecindario y evolución separadas, no se podrá contar con más de 2 estados, ya que solo se permitirá definir la evolución a partir de la notación S/B, donde S (Survival) lista las cantidades de vecinos vivos que permiten que una célula viva siga viva en el siguiente paso, y B (Birth) lista las cantidades de vecinos vivos que hacen que una célula muerta esté viva en el siguiente paso.

Por otro lado, si se define la función transición, se permitirá desarrollar reglas de evolución más complejas utilizando aritmética y estructuras de control. La función de transición no sólo puede considerar cantidades sino también posiciones.

La implementación satisfactoria de este lenguaje permitiría visualizar e interactuar con autómatas celulares únicamente definiendo su configuración y las reglas que los rigen, simplificando el proceso de generación.

4. Construcciones

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

(I). Se podrá definir la función de vecindario del autómata, permitiendo vecindarios complejos más allá de las células circundantes.
(II). Se podrá definir, a través del atributo Evolution, la regla de evolución en formato S/B la cual solo se regirá en base a cantidades.
(III). Se podrá definir la función transición, que engloba a las funciones vecindario y evolución.
(IV). Para definir las funciones vecindario y transición se proveerán

(IV.1). Operadores relacionales como <, >, =, ≠, ≤ y ≥.
(IV.2). Operaciones aritméticas básicas como +, -, *, % y /.
(IV.3). Operaciones lógicas básicas como AND, OR y NOT (&&, ||, !).
(IV.4). Estructuras de control básicas de tipo IF-THEN-ELSE y FOR.

(V). En la definición de las función de transición y de vecindario, se podrá acceder a una variable cell que permitirá las siguientes operaciones:
  (V.1). Referenciar celdas utilizando coordenadas relativas x e y a través de cell(x,y).
  (V.2). Acceder a la posición absoluta de la celda en cuestión a través de cell.x (devuelve la coordenada x de la celda) y cell.y (devuelve la coordenada y de la celda).
  (V.3). Referenciar celdas en posiciones especiales a través de funciones auxiliares como:
  `cell.diag_asc(displacement)`, `cell.diag_dec(displacement)`, `cell.vert(displacement)` y `cell.hor(displacement)`.

(VI). En la definición de las función de transición y de vecindario, se podrá acceder a cualquier celda en la grilla de manera absoluta a través de una variable grid al igual que con cell (grid(x,y)).

(VII). En la definición de la función vecindario, se utilizará la sentencia add(x,y) para indicar que se quiere agregar al vecindario la celda con coordenadas x e y. También se podrá utilizar la sentencia remove(x,y) que removerá del vecindario la celda con dichas coordenadas.

(VIII). Se podrá, a través del atributo Frontier, elegir entre una frontera abierta (todas las células fuera de la grilla tienen un estado fijo), una frontera reflectora (las células fuera de la grilla toman los valores que están dentro, como si se tratara de un espejo) o una frontera periódica o cíclica (las células que están en la frontera interaccionan con sus vecinos inmediatos y con las células que están en el extremo opuesto de la grilla) con las palabras clave open, mirror y periodic respectivamente.

(IX). Se podrá definir el ancho y el alto de la ventana a través de los atributos Width y Height.

(X). Se podrán definir vecindarios conocidos a través de constantes definidas. Por ejemplo:
  (X.1). VON_NEWMAN: vecindario compuesto por las 4 celdas que comparten aristas con la celda en cuestión.
  (X.2). MOORE: vecindario compuesto por las ocho celdas que rodean a la celda en cuestión.
  (X.3). K_NEIGHBORHOOD: vecindario de radio K.

(XI). Se podrán definir reglas de evolución de autómatas celulares conocidos a través de constantes definidas. Por ejemplo:
  (XI.1). CONWAY: reglas del clásico "Juego de la vida": B3/S23 (nace con 3 vecinos; sobrevive con 2 o 3).
  (XI.2). HIGHLIFE: variante de Conway con regla B36/S23. Permite patrones replicadores.
  (XI.3). SEEDS: regla B2/S. Las células vivas mueren siempre; las muertas nacen solo con 2 vecinos vivos.
  (XI.4). DAY_NIGHT: regla B3678/S34678. Tiene estructuras complejas y estables.

(XII). A través del atributo States, se podrán definir los estados que puede tomar una célula, siempre y cuando se utilice una función de transición si estos son más de dos.
  (XII.1). Cada estado deberá tener un identificador único.
  (XII.2). Las función de transición deberán contemplar todos los estados personalizados.

(XIII). A través del atributo Colors, se podrá definir el color para los estados definidos.

5. Casos de Prueba

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

(I). Un programa que genere el autómata del "Juego de la vida" de John Horton Conway.
(II). Un programa que genere un autómata de tres estados y una función transición que considere a los tres.
(III). Un programa que define los estados SANO, INFECTADO y RECUPERADO, con transiciones que simulan la propagación de una enfermedad.
(IV). Un programa que genere un autómata con el vecindario VON_NEWMAN y la regla SEEDS.
(V). Un programa que genere un autómata cuya función transición use todos los operadores lógicos (AND, OR, NOT) en la función transición.
(VI). Un programa que genere un autómata cuya función transición use la estructura de control IF-THEN-ELSE.
(VII). Un programa que genere un autómata cuya función transición use la estructura de control FOR.
(VIII). Un programa que genere un autómata cuya función transición use operadores relacionales (<, >, =, ≠, ≤, ≥) correctamente.
(IX). Un programa que utilice las funciones de la variable cell para moverse por las células diagonales, verticales y horizontales a esta en la función transición.
(X). Un programa que genere un autómata cuya función transición utilice operadores aritméticos combinados.

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

(I). Un programa mal formado.
(II). Un programa cuya función de vecindad no utilice la función add(x, y).
(III). Un programa cuya función de transición no devuelva un estado válido para la célula.
(IV). Un programa con más de 2 estados que defina una función de vecindario y regla de evolución separadas.
(V). Un programa cuya definición S/B cuente con S o B mayores a la cantidad de células en el vecindario.
(VI). Un programa que utilice el valor custom en el atributo Neighbourhood y no especifique una función neighbourhood.
(VII). Un programa que defina la función neighbourhood sin definir el atributo Evolution.

6. Ejemplos

Generar un autómata celular de dos estados (viva, muerta), cuyas celdas pasen o se mantengan en estado vivo cuando se cumplen las siguientes condiciones:

v	v/m	v	v/m	v
v/m	v/m	v/m	v/m	v/m
v	v/m	C	v/m	v
v/m	v/m	v/m	v/m	v/m
v	v/m	v	v/m	v

```
configuration:
   Height: 100;
   Width:  100;
   Frontier: Open;
   States: {dead, alive};
   Colors: {white, red};
transition:
   if cell(2,2) == alive && cell(-2,-2) == alive && cell(-2,2) == alive &&
cell(2,-2) == alive && cell(0,2) == alive && cell(0,-2) == alive && cell(2,0) ==
alive && cell(0,2) == alive
   then
      ->alive;
   else
      ->dead;
   endif
```

Alternativamente:

```
configuration:
   Height: 100;
   Width: 100;
   Frontier: Open;
   States: {dead, alive};
   Color: {white, red};
transition:
   if cell.diag_asc(2) == alive && cell.diag_asc(-2) == alive &&
cell.diag_dec(-2) == alive && cell.diag_dec(2) == alive && cell.vert(2) == alive
&&  cell.vert(-2) == alive && cell.hor(2) == alive && cell.hor(-2) == alive
     then
        ->alive;
     else
        ->dead;
    endif
```

Generar el “Juego de la vida” de John Horton Conway.

```
configuration:
   Height: 100;
   Width:  100;
   Frontier: Periodic;
   States: {alive, dead};
   Colors: {white, black};
   Neighborhood: MOORE;
   Evolution: CONWAY;
```

Generar un autómata celular de dos estados (on, off) cuyo vecindario sea una cruz de 7x7 celdas. La celda pasará a estar en estado on si 6 de sus vecinos están on y pasará a estado off si más de 10 están on o si lo están menos de 3.

```
configuration:
   Height: 100;
   Width:  100;
   Frontier: Mirror;
   States: {on, off};
   Colors: {white, black};
   Evolution: 10,9,8,7,6,5,4/6;
   Neighborhood: custom

neighborhood:
      for i in [-3,3] do
         add(cell(i,0));
         add(cell(0,i));
      end
```