CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Matemáticas e Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Desarrollo de una plataforma web para la especificación de protocolos biológicos basada en BIOBLOCKS

Autor: AITOR MOYA LÓPEZ

Director: ALFONSO RODRÍGUEZ PATÓN

MADRID, ENERO 2018

ÍNDICE GENERAL

Introducción

Gro es un simulador bacteriano basado en agentes (MBA), esto quiere decir que GRO sigue un modelo computacional en el cual se permite la simulación de acciones e interacciones de individuos autónomos. En el caso de GRO son bacterias autónomas, dentro de un entorno especifico en el que se puede analizar los efectos que se producen en conjunto sobre el sistema.

Actualmente, para comenzar a realizar simulaciones en GRO es necesaria la instalación de varios programas externos (Qt, Bison, Flex), cosa que en muchos casos es una tarea ardua...Sin contar con la configuración del set up del sistema en el que se esté instalando.

Por otro lado, si conseguimos instalar GRO satisfactoriamente, nos encontramos otra barrera, la especificación de las simulaciones. Actualmente GRO carece de una interfaz gráfica de usuario (GUI) a la hora de crear las simulaciones, por lo tanto, para realizar simulaciones debemos conocer la especificación del simulador. Como es un software que se utiliza en entornos multidisciplinares, es un impedimento para los investigadores realizar los experimentos mediante una especificación tan compleja como la que se utiliza actualmente en GRO, haciendo que estos tengan que recurrir muchas veces al manual de GRO para ver la sintaxis de las sentencias o las funciones que pueden simular.

Solución propuesta al problema

La solución que hemos decidido tomar es la implementación de una GUI que trabaje sobre una estructura basada en HTML y JavaScript para poder utilizarla tanto para la versión de escritorio de GRO como para la versión de GRO web también conocida como GROW.

El software con el que se implementará la GUI es Blockly, una librería "*open source*" que proporciona Google. Actualmente hay otras distribuciones basadas en esta librería como puede ser Scracth y más específicamente en el ámbito de la biología sintética podemos encontrar la plataforma BioBlocks desarrollada por investigadores de LIA.

La decisión de utilizar Blockly como GUI viene dada ya que esta librería nos permite la posibilidad de ser utilizada en una gran cantidad de entornos, dándonos la posibilidad de hacer conversiones desde bloques a otros lenguajes como pueden ser Python, JavaScript y JSON.

Pero no solo nos ofrece la posibilidad de hacer una multiconversión de manera eficaz, sino también nos permite que las simulaciones funcionen de manera rápida en dispositivos de todo tipo, tanto ordenadores como dispositivos inalámbricos como tablets o móviles, dándonos la posibilidad de poder realizar simulaciones desde cualquier lugar.

Otra de las ventajas que nos ofrece Blockly, es la capacidad de crear piezas y controlar la conexión entre ellas, de este modo, podemos evitar fallos en la especificación desde el lado del desarrollador y facilitar la creación de simulaciones para los usuarios. Por lo tanto, nunca se podrán realizar simulaciones que GRO no pueda realizar.

Trabajo Previo

3.1. Blockly (1), Scratch (2) y Bioblocks (3)

a) Introducción a Blockly

Blockly es una librería desarrollada por Google, basada en interfaces gráficas mediante bloques. La filosofía que sigue esta librería es la implementación de interfaces gráficas o aplicaciones, con una forma visual basada en bloques que resulta muy intuitiva para los usuarios.

Google nos proporciona una API con las características y funcionalidades que nos proporciona esta librería y un generador online de bloques para que podamos construir bloques nuevos de forma muy sencilla. Mediante la plataforma podremos seguir de manera visual y muy intuitiva la el bloque que queramos implementar, luego solo habrá que programar la codificación que haremos del bloque.

Bioblocks esta creado a partir de las librerías de Blockly.

3.2. Funcionamiento y características de GRO

GRO es un simulador de bacterias desarrollado por Eric Klavins (4), para este proyecto trabajaremos con una versión modificada de GRO. Esta versión está desarrollada por Martin Guitierrez(5).Esta versión amplia algunas características del GRO original y sigue una modelización propia de bacterias mediante la descripción de sucesos con "*Probabilistic Time Autómata*" (PTA).El manual de esta versión de GRO la podemos encontrar en la página de LIA(6).

Revisando el manual del simulador podemos ver que a grandes rasgos la especificación del simulador se basa en dos grandes partes, la especificación de los componentes biológicos que siguen una jerarquía de la forma (Bacteria-Plásmidos-Operones-Promotor-Proteínas) y las acciones que se pueden realizar sobre dichas estructuras.

a) Bacteria

Es la estructura de más alto nivel biológico que contempla GRO. Este simulador trabaja con solo un tipo de bacteria, la *"Ecoli"*.

b) Plásmido

Los plásmidos son moléculas de ADN extracromosómico generalmente circular que se replican y transmiten independientes del ADN cromosómico. Estos se encuentran dentro de las bacterias. El número de plásmidos contenidos en una bacteria puede ser desde ninguno hasta N.

c) Operón

Un operón se define como una unidad genética funcional formada por un grupo de genes capaces de ejercer una regulación de la expresión proteica asociada a este, por medio de los sustratos(proteínas) con los que interaccionan los promotores mediante su factor de transcripción.

En GRO la regulación de la síntesis de proteínas se codifica mediante la terna operón-promotor-proteína.

d) Promotor

En genética un promotor es una región de ADN que controla la iniciación de la transcripción de una determinada porción del ADN a ARN. Un promotor, por lo tanto, promueve la transcripción de un gen.

En GRO el promotor se ha modelizado como una puerta lógica, donde la entrada es un factor de transcripción y la salida es la síntesis de una o un conjunto de nuevas proteínas.

e) Proteínas

Sustancia bioquímica constituida por una hebra o cadena lineal de unidades llamadas aminoácidos. Las proteínas son los productos primarios codificados por los genes, encargadas de organizar la actividad bioquímica celular.

f) Acciones

Las acciones que el simulador puede realizar son las siguientes:

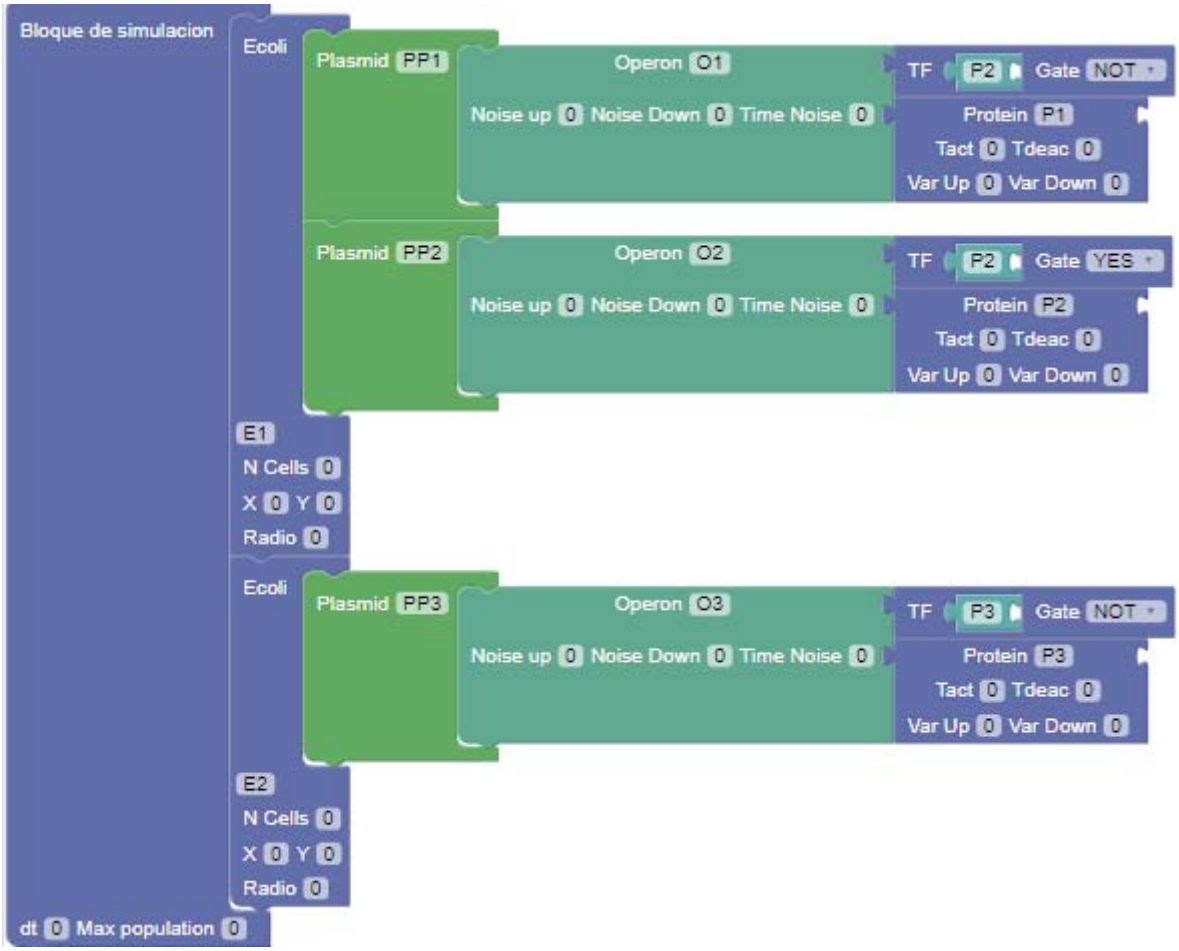
- Fluorescencia: Colorea las bacterias que contienen la proteína indicada en su interior.
- Conjugación: Transferencia de una cadena de ADN de una bacteria donadora a otra receptora.
- Perdida de plásmidos: Acción que elimina un plásmido de la bacteria seleccionada
- Set/Remove Eex: Fija o remueve una restricción de plásmidos sobre la bacteria (mediante conjugación)
- Muerte: Elimina a la bacteria deseada.
- Quorum Sense: Mecanismo de regulación de la expresión genética.

g) Otros componentes

- Dump Single: Programa complementario que guarda los datos de la simulación en un fichero externo
- Módulo de señales: Señales binarias utilizadas para distintos propósitos en el simulador
- Módulo de nutrientes: Distribución de nutrientes que se distribuyen por la zona de crecimiento de las bacterias.

Bloques Genéticos

Los llamados bloques genéticos definen las estructuras biológicas presentes en las bacterias que el simulador es capaz de procesar. En estos bloques se definen las propiedades que se contemplan en GRO, como pueden ser el tiempo de sinterización de las proteínas implementadas, la localización espacial de la cepa de bacterias o el tipo de puerta lógica que modeliza el promotor.



Implementación de un circuito genético sin acciones en GROCKLY

4.1. Introducción

La especificación que GRO utiliza para la realización de sus simulaciones no fue pensada en un principio para una interfaz gráfica de estas características. Lo primero que hay que comprender son los objetivos que debemos alcanzar para implementar la GUI.

En primer lugar queremos una interfaz sencilla y visualmente atractiva, por eso no debemos sobrecargar los bloques con información y/o hacer macro bloques que generen estructuras de código.

En segundo lugar, debemos realizar una implementación de algoritmos no demasiado compleja para que cualquier persona con unos conocimientos mínimos de programación sea capaz de insertar nuevos bloques para que la plataforma no se quede obsoleta.

En este punto es cuando nos encontramos con el primer problema, la compilación de bloques que utiliza Blockly. Blockly utiliza una compilación de bloques en profundidad. Esto, para la implementación de los bloques genéticos, es un problema ya que se expanden en anchura, por lo tanto, a la hora de compilar los bloques y extraer de ellos los datos necesarios para crear la especificación nos resulta muy complejo mantener el orden correcto de compilación para generar una especificación en GRO.

Por otro lado, aunque consigamos realizar un algoritmo sencillo y optimo la codificación de los bloques solo devuelve un string, lo que es otro problema, ya que habría que hacer lecturas sobre estos (1 por atributo retornado), cosa que es poco eficiente.

La solución que se ha desarrollado para este problema es la implementación de una gramática formal propia generada por los bloques genéticos, de la cual se pueden extraer los datos que necesitemos para generar nuestro código.

4.2. Ecoli

El bloque Ecoli como su propio nombre codifica la información relativa a la bacteria con ese nombre (por el momento sólo existe un tipo de bacteria).

El simulador necesita los siguientes datos para poder realizar las simulaciones:

a) Numero de Bacterias
   - Este número define el número de bacterias que tendrá la cepa especifica.

b) Posición X
   - GRO codifica las bases de los experimentos reales (palcas de Petri), en forma de plano por lo tanto este atributo define la posición sobre el eje X del plano.

c) Posición Y
   - Posición sobre el eje Y del plano para la cepa especifica.

d) Radio
   - GRO codifica el crecimiento de las bacterias de forma radial, por tanto, este atributo define el radio del circulo que se genera con centro (X, Y) definidos anteriormente.

e) Plásmidos
   - Este atributo define el conjunto de plásmidos que contiene en el momento inicial de la simulación, este dato puede variar en el paso de la simulación.

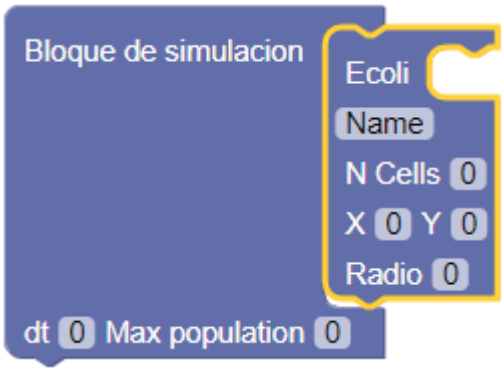
f) Programa de ejecución
   - Define el programa que deberá ejecutar la bacteria durante el tiempo de simulación.

La especificación en GRO de esta estructura es la siguiente:

```
c_ecolis(10,0,0,0,{"PP1"},program p());
```

Ejemplo de la especificación de una bacteria de Ecoli para el simulador GRO, los atributos que presenta el código de izquierda a derecha son: 1-Numero de bacterias , 2-Posición X, 3-Posición Y, 4-Radio , 5-Lista de plásmidos que contiene , 6-Nombre del programa de ejecución.

Y el bloque implementado en GROCKLY :



Representación del bloque Ecoli en GRO

4.3. Plásmido

El bloque plásmido modeliza dicha estructura biológica. Los datos que implementan este elemento en el lenguaje del simulador son:

a) Nombre

Este campo es necesario para localizar el tipo del plásmido sobre el que se realizan ciertas acciones como la conjugación.

b) Conjunto de operones que contiene

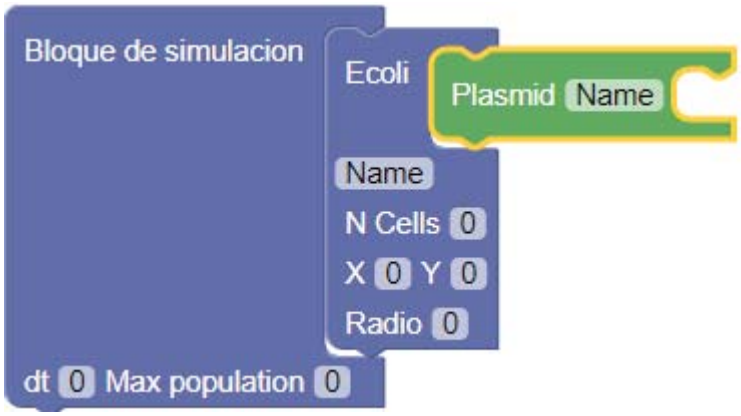
Este campo relaciona el conjunto de operones que contiene el plásmido.

La especificación de este elemento en GRO es la siguiente:

```
plasmids_genes([PP1:={"O1","O2"}]);
```

En primer lugar, se instancia el nombre del plásmidos, en este caso PP1, y en segundo lugar la familia de operones, en este caso solo presenta dos elementos O1 y O2.

Y el bloque implementado en GROCKLY:



Representación del bloque plásmido en GROCKLY

4.4. Operón

El bloque operón codifica la información relativa a esta estructura genética. En GRO el operon se expresa como una terna operon-promotor-proteina, por lo tanto, estos dos elementos deben ser incluidos como atributos dentro de estos elementos.

El simulador necesita como atributos de un operon los siguientes elementos:

a) Nombre del operon

Nombre asignado al operon en cuestión que estamos implementando

b) Familia de proteínas que contiene

Conjunto de proteínas que es capaz de sintetizar el operon.

c) Promotor

Instanciación del promotor asignado a este operon.

d) Tiempo de ruido

Indica el tiempo en el que el operon deja de funcionar correctamente, este fenómeno se explica en el paper publicado en ACS.

e) Tiempo de sintetización

Tiempo en el que tarda cada proteína de la familia antes especificadas en sintetizarse en el medio.

f) Tiempo de degradación

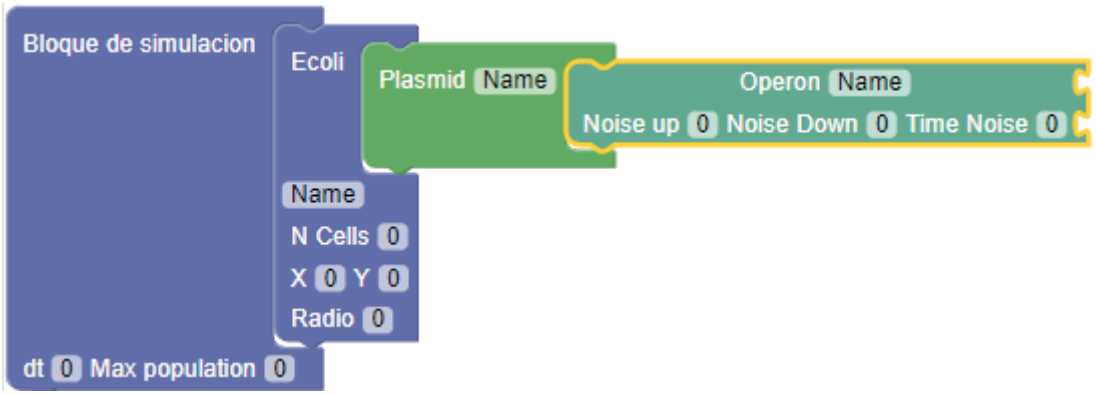
Tiempo en el que tarda cada proteína de la familia antes especificadas en degradarse en el medio.

La especificación de este elemento en GRO es la siguiente:

```
genes([name:="O1",
       proteins:={"P1"},
       promoter:=[function:="YES",
       transcription_factors:={"P1"}],
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={10},variabilities:={0}],
       prot_deg_times:=[times:={5},variabilities:={0}]]);
```

En orden descendente y de derecha a izquierda tenemos los siguientes datos,1-Nombre del promotor ,2-Familia de proteínas,3-Promotor,4-Ruido del promotor, 5-Tiempo de sintetización,6-Tiempo de degradación.

Y el bloque implementado en GROCKLY:



Representación del bloque operón en GROCKLY

Como se puede observar, analizando la especificación podemos simplificar la estructura de este para una mejor comprensión del suceso biológico aislando los sucesos específicos del operon.

4.5. Promotor

En GRO, como ya hemos explicado se modeliza el operon como la terna Operon-promotor-proteina. Para modelizar esto en GROCKLY los distintos operones solo pueden tener un único promotor.

Los atributos que necesita esta estructura para funcionar en el simulador son los siguientes:

a) Factores de Transcripción

Conjunto de proteínas que producen la sintetización o degradación el conjunto de proteínas del operon. La familia de proteínas que forman el factor de transcripción se modeliza mediante bloques auxiliares, con identidad de bloques de proteínas explicados más adelante.

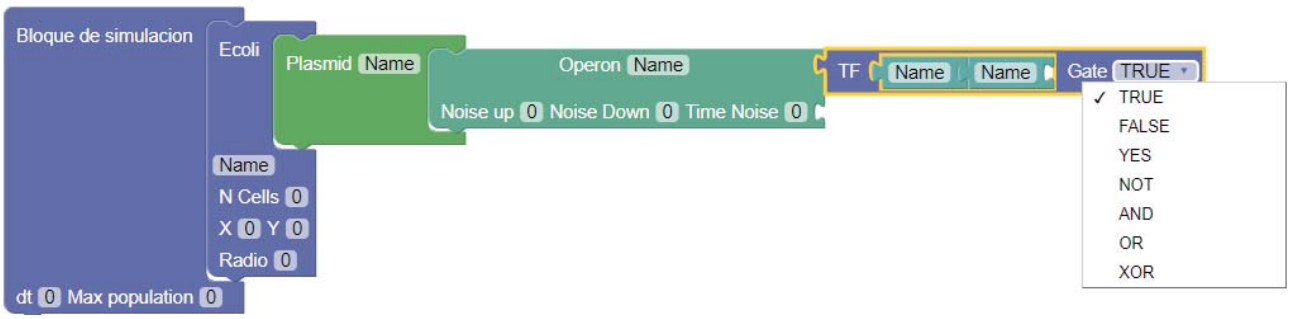
b) Tipo de puerta lógica

La codificación en GRO del promotor se incluye dentro de la implementación de los operones que es la siguiente:

```
genes([name:="O1",
       proteins:={"P1"},
       promoter:=[function:="YES",
       transcription_factors:={"P1"}],
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={10},variabilities:={0}],
       prot_deg_times:=[times:={5},variabilities:={0}]]);
```

Resaltado la implementación en GRO de un promotor

Y el bloque implementado en GROCKLY es el siguiente:



Implementación del bloque operon en GROCKLY, los atributos de este bloque de izquierda a derecha son:

1-Factores de transcripción 2-Tipo de puerta lógica

4.6. Proteína

El bloque proteína codifica una de las proteínas que forman parte de la terna operon-promotor-proteina.

Los atributos necesarios para este componente son:

a) Tiempo de activación

Este parámetro fija el tiempo en que la proteína se sintetiza en el medio.

b) Tiempo de degradación

Este parámetro fija el tiempo en que la proteína se degrada en el medio.

c) Varianza positiva

Varianza del tiempo de activación.

d) Varianza negativa

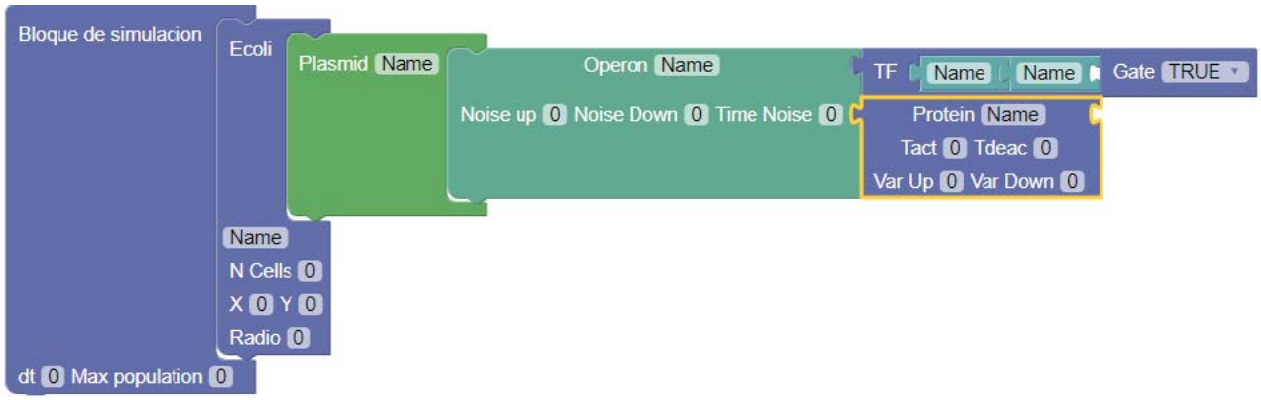
Varianza del tiempo de desactivación.

Como ocurría anteriormente en el promotor, la codificación de las proteínas que controla el operon se incluye dentro de él.

```
genes([name:="O1",
       proteins:={"P1"},
       promoter:=[function:="YES",
       transcription_factors:={"P1"}],
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={10},variabilities:={0}],
       prot_deg_times:=[times:={5},variabilities:={0}]]);
```

Codificación de la proteína dentro del bloque operon, los parámetros de arriba hacia abajo y de izquierda a derecha son:
1-Nombre de la proteína, 2-tiempo de activación ,3—Varianza de activación ,4-tiempo de degradación ,5-Varianza de degradación.

Y su implementación en bloques de GROCKLY es la siguiente:



Implementación del bloque de proteínas, sus datos de arriba hacia abajo y de izquierdas a derecha son:
1-Nombre de la proteína,2-Tiempo de activación, 3-tiempo de desactivación, 4-Varianza de activación ,5-Varianza de degradación

4.7. Bloques auxiliares

Los bloques auxiliares que deberemos añadir para conseguir una interfaz clara y auto resolutiva es la introducción de un bloque auxiliar a modo de localizador de proteínas.

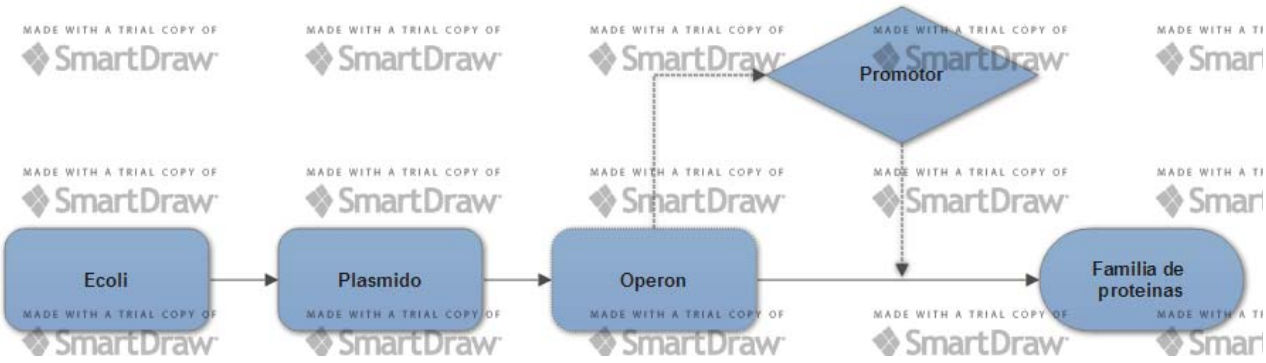
La idea de este bloque viene de la mano con la la especificación de las acciones en GRO, estas, van relacionadas con la presencia o falta de alguna proteína, por lo tanto, es lógico útil implementar una subfamilia de bloques que nos faciliten esta estructura.

Como podemos observar la definición de acciones anteriormente dada y la de operon son parecidas, por lo tanto, utilizaremos este estilo de bloque en el también.

Como este elemento es solo un localizado no le hace falta ningún parámetro más que el nombre, que se utilizara a modo de ID.

4.8. Diagrama de conexiones entre los bloques

La arquitectura genética que sigue GRO es la siguiente:



4.9. Descripción del algoritmo

El algoritmo implementado para la construcción del código mediante la implementación de bloques Grockly esta divida en dos grandes grupos.

Por un lado, se encuentran los bloques de acciones, estos bloques se codifican de forma inmediata. Esto se puede realizar gracias a que son auto resolutivos, lo que quiere decir que en ellos encuentro toda la información necesaria para poder generar su propio código GRO.

Por otro lado, tenemos los bloques genéticos, los cuales no son auto resolutivos porque que su estructura en código GRO se codifica de forma inclusiva Esto significa que el código GRO que genera ciertos tipos de estructuras necesita dentro de sus atributos de un nivel inferior en su interior, por ejemplo el código que codifica el operón contiene dentro de él la implementación de las proteínas que sintetiza este y el promotor que lo regula ,como hemos visto anteriormente.

Otro problema con el que nos hemos topado a la hora de implementar la codificación de los bloques es la forma que tiene Blockly de recorrer los bloques. Blockly, crea una distribución en forma de Grafo, en la que cada bloque representa un nodo y las aristas que unen estos son las conexiones que unen los bloques entre ellos. A la hora de recorrer el árbol creado el algoritmo que sigue es una busque en profundidad, lo que es un problema ya que nuestra distribución de bloques se expande en anchura.

Teniendo en cuenta todas estas cosas la solución que hemos aplicado es generar una gramática propia que genere nuestro código de bloques genéticos de forma independiente, extrayendo de los bloques los datos pertinentes de cada tipo de bloque y mediante identificadores para padres e hijos combinándolo con ciertas reglas para generar nuestro código GRO.

Bloques de Acciones

5.1. Introducción

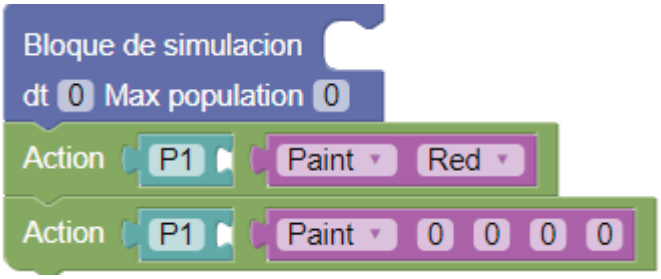
Los bloques de acciones en Grockly, son la representación gráfica de las acciones que se pueden realizar en GRO. Este tipo de bloques poseen, al igual que los bloques que codifican la información relativa a módulos externos de GRO (signals y nutrients), una estructura auto resolutiva, lo que quiere decir, que no nos hace falta nada más para realizar la codificación de la información que estos bloques nos aportan.

5.2. Estructura General

Observando el código de GRO de las acciones podemos observar que se sigue una estructura similar independiente de la acción que realice, esta estructura es de la forma acción- objetivos de la acción- tipo de acción.

Implementación de una acción en GRO, los elementos de izquierda a derecha son:
1-Conjunto objetivo de la acción ,2-Tipo de acción ,3-Parámetros de la acción

De modo que la implementación en bloques Grockly es la siguiente:



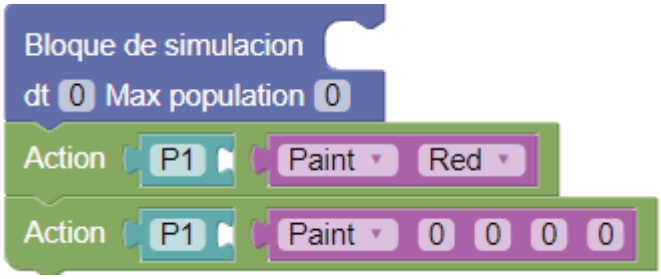
Implementación de un bloque de acciones en Grockly

Al poder encontrar una estructura simétrica independiente de la acción, esto nos permite poder implementar un bloque contenedor (bloque verde) al que se le incrusten un conjunto de objetivos mediante bloques auxiliares (bloques cian) y una acción determinada (bloque morado).

5.3. Bloques de pintado

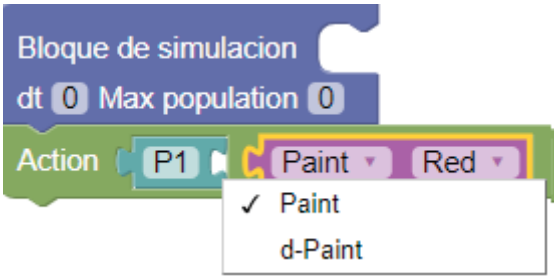
Los bloques de pintado codifican la coloración que pueden sufrir ciertas proteínas cuando se producen las condiciones adecuadas

En Grockly existen dos bloques que codifican esta propiedad, uno con los argumentos clásicos que se dan en el código de GRO y otra para usuarios nuevos que facilita la asignación de colores.



Implementación de los dos bloques de pintado en Grockly, el primero es la versión simplificada del segundo para usuarios nuevos.

Del mismo modo GRO nos permite colorear las proteínas en dos modos distintos, Paint y d_Paint. El primero nos produce un pintado plano y permanente, el segundo un pintado con degradación. Estas características también se encuentran implementadas en los bloques de Grockly mediante un desplegable.

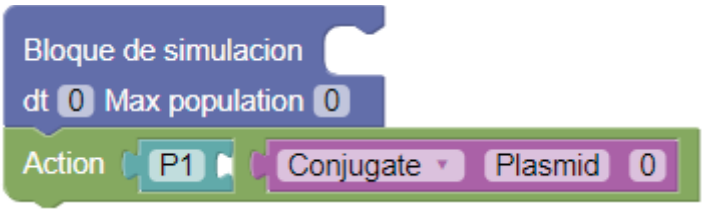


Tipos de pintado posibles en el bloque de acción Paint.

5.4. Bloque de conjugación

La conjugación es un proceso muy importante a la hora de las simulaciones. Gro permite realizar este proceso en modo de acción. La conjugación es un proceso de transferencia de información genética desde una célula donadora a otra receptora, promovido por un tipo de plásmidos.

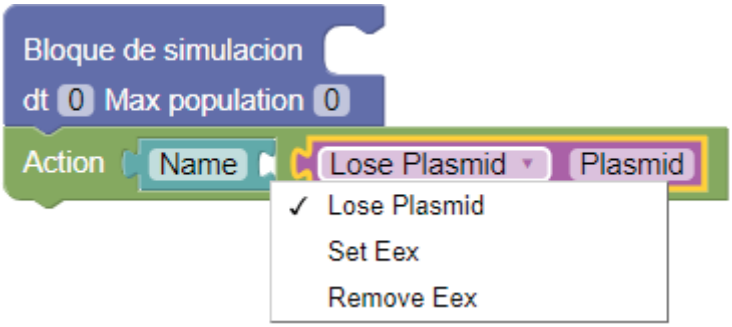
La implementación de los bloques Grockly de esta acción es la siguiente:



Implementación de la acción de conjugación.

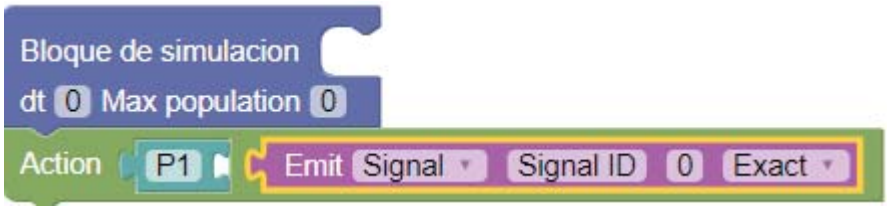
5.5. Lose Plasmid, Set/Remove Eex

En este bloque de acciones de Grockly hemos agrupado tres acciones que tienen una estructura similar a la hora de generar su código. Esto es un beneficio para nosotros ya que de esta forma podemos reducir el número de bloques y maximizar la utilidad de estos.



5.6. Emisión de señales

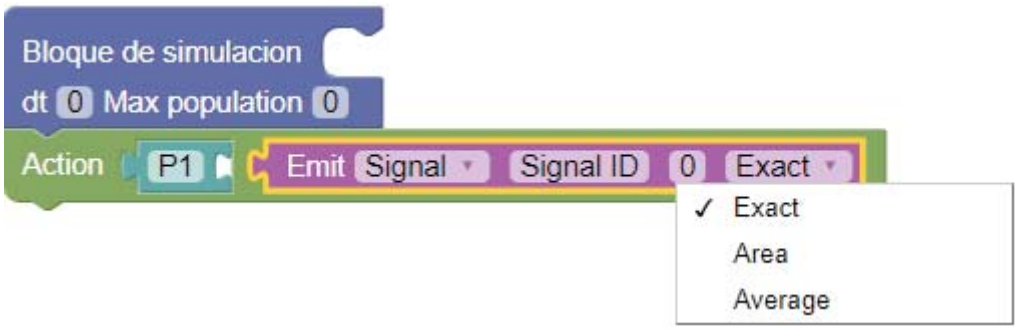
El bloque de emisión de señales vincula la señal con su objetivo. Su representación en Grockly es la siguiente:



Implementación de la acción de emisión de señales

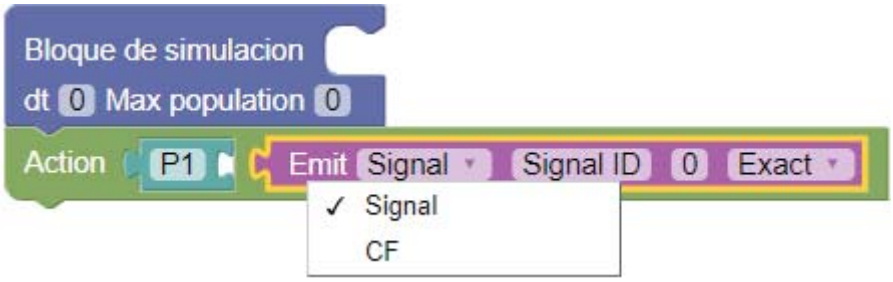
En donde la ID de la señal se instancia en el bloque de señales.

En GRO también podemos especificar como se distribuirá la señal, teniendo las siguientes opciones:



Tipos de emisión de las señales en Grockly.

De la misma manera que podemos tener distintos tipos de emisiones de señales:



Tipos de emisión de señales en Grockly.

5.7. Quorum Sensing

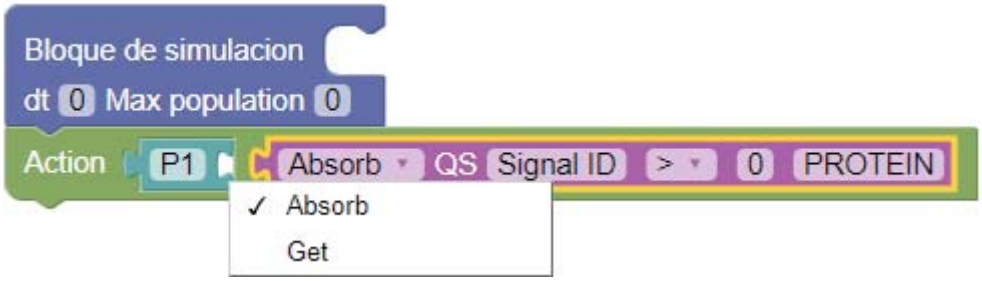
La autoinducción o *quorum sensing* es un proceso biológico de regulación de la expresión genética en respuesta a la densidad de la población.

En gro este proceso se modeliza mediante una acción y el bloque encargado de

codificar la información relativa a esta acción es el siguiente:

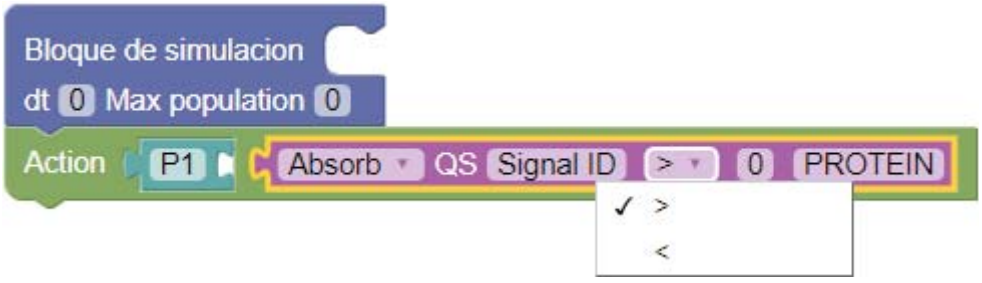
Implementación de la acción QS en Grockly.

En donde podemos encontrar varios tipos de autoinducción:



Tipos de QS posibles en Grockly.

Y varios operadores de densidad:



Operadores para la acción de QS en Grockly.

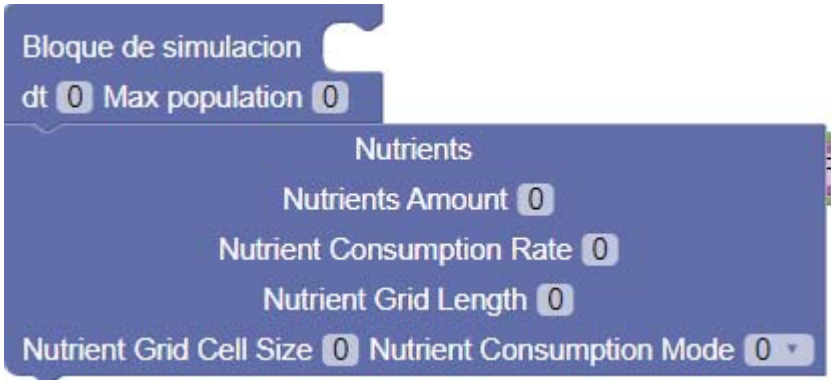
Bloque de Nutrientes

6.1. Introducción

Los bloques de nutrientes codifican la implementación de dicho módulo de GRO.

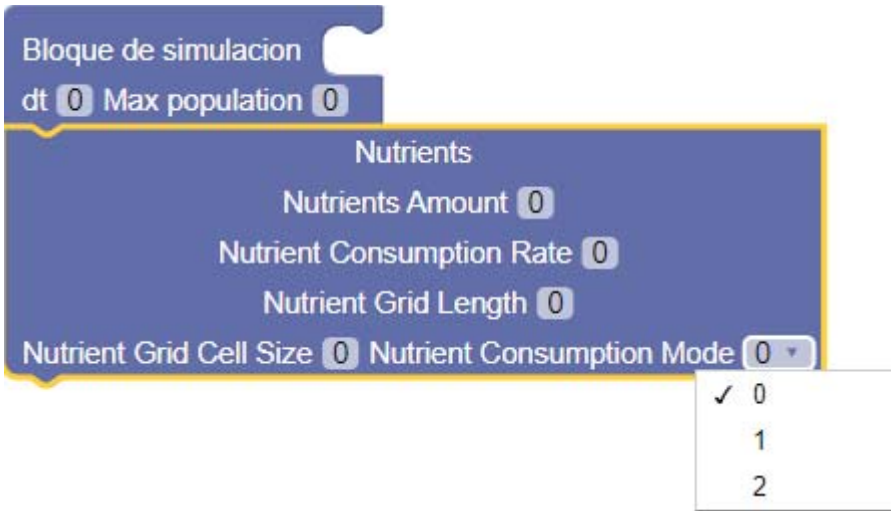
El módulo de nutrientes se utiliza para crear una distribución de nutrientes sobre la superficie de crecimiento de las cepas. Esto nos permite crear zonas donde al tener una mayor cantidad de nutrientes el crecimiento sea más rápido que en otras con menos crecimiento.

6.2. Implementación del bloque en Grockly



Implementación del bloque de Nutrientes en Grockly.

Y como este módulo nos permite alternar entre distintos modos de consumición de los nutrientes, el bloque en Grockly contiene los siguientes modos:



Tipos de consumición de nutrientes en Grockly.

En donde el parámetro 0 indica un modo de consumición homogéneo, el parámetro 1 indica un modo de consumición por proximidad y el parámetro 2 indica un modo aleatorio de consumición.

La presencia del propio bloque de nutrientes implica la activación de este dentro de la simulación.

Bloques de Señales

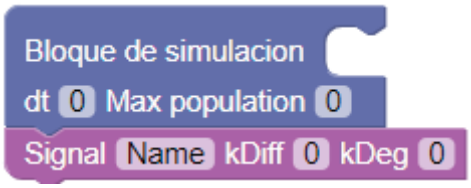
7.1. Introducción

Los bloques de señales codifican la información relativa a dicho módulo de GRO.

El uso de las señales en GRO está muy ligado al de otras acciones, como por ejemplo siendo capaz de reducir la población gracias a la utilización de estas (acción death sobre una bacteria ligada a una señal) o en procesos más complejos como los de QS(Q*uorum Sense*).

7.2. Creación de la señal

El bloque que codifica la instancia de las señales es el siguiente:

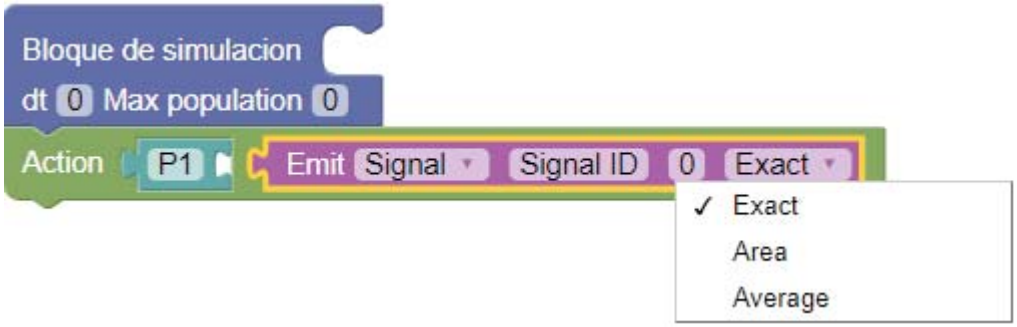


Implementación del bloque que implementa una señal en Grockly.

En donde tendremos que darle un ID a la señal correspondiente, kDiff representa el coeficiente de difusión de las señales y kDeg el de degradación.

7.3. Modificación de la malla de señales

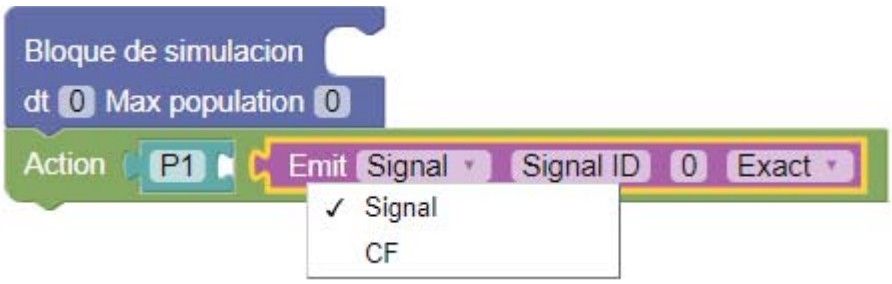
Sobre el módulo de señales que posee implementado GRO, podemos realizar cambios de tal manera que podamos modificar la malla sobre la que crear las señales. El bloque en Grockly que nos permite realizar esto es la siguiente:



Bloque de modificación de la malla de señales en Grockly.

Donde podremos definir la longitud de la maya el tamaño de las celdas y su vecindario.

Del mismo modo podemos hacer que en el modo gráfico de la simulación no aparezcan la representación gráfica de estas señales con el desplegable de la parte superior:



Opción de mostrar señales durante el modo gráfico de la simulación.

Inserción de nuevos bloques

Una de las razones por la que hemos elegido este software, es por la facilidad que nos proporciona a la hora de actualizar la plataforma. Blockly, nos proporciona una plataforma online (7) desde la cual podemos generar bloques de Blockly mediante una interfaz gráfica basada en Blockly de manera rápida y muy intuitiva.

De esta forma podemos generar bloques nuevos que luego codificaremos para que se correspondan con la especificación correspondiente en GRO.

Un punto a tener en cuenta es, que los nuevos fragmentos de código en GRO sean auto resolutivos, para así simplificar la generación de código a través de los bloques.

La generación de código que realiza Grockly, se realiza mediante la concatenación de cadenas de texto.

```
function writeMain(){
  return code=writePlasmid()+actionCode+programCode+"program main():={"+writeCell()+"};";
}
```

Composición del codigo GRO generado por Grockly

En la imagen superior vemos la concatenación de las distintas partes del codigo que se llevan a cabo a la hora de generar la especificación de la distribución de bloques realizada.

De este modo, lo único que tendremos que hacer si nuestro bloque es auto resolutivo, es localizar el tipo de bloque y crear el fragmento de código correspondiente al bloque, extrayendo los parámetros (si se necesitan) y colocándolos en su lugar, como se observa en la siguiente imagen.

```
if(des[h].getInput("Paint2")){
  codeA='},"'+des[h].getFieldValue("TYPE")+'",{'+des[h].getFieldValue("COLOUR")+'});'
}
if(des[h].getInput("ActionR1")){
  codeA='},"'+des[h].getFieldValue("TYPE")+'",{"'+des[h].getFieldValue("NAME")+'"});'
}
```

Ejemplos de codificación de bloques auto resolutivos en Grockly

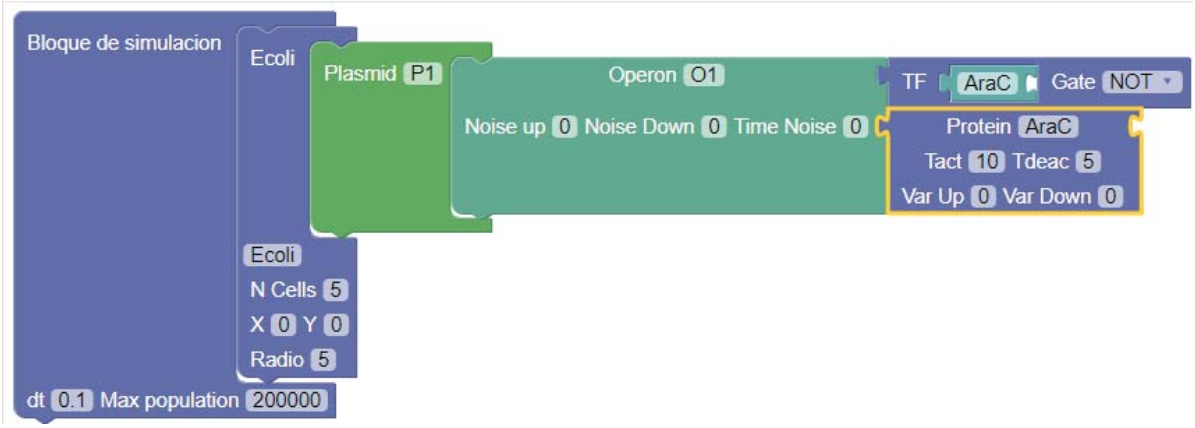
En cambio, si no somos capaces de crear un bloque auto resolutivo, la generación del bloque se deberá hacer creando un método auxiliar con el cual podamos volcar los datos necesarios en una estructura de datos para luego formar con ellos nuestro código GRO.

Validación de simulaciones

La validación de las simulaciones se hará sobre GRO, para ello primero crearemos distribuciones de bloques sencillas, para ir probando todas las partes de la herramienta.

9.1. Generación de una bacteria simple

El primer ejemplo para la validación del código generado por la aplicación será la implementación de una bacteria la cual contenga un plásmido, un operon, un promotor y una proteína.



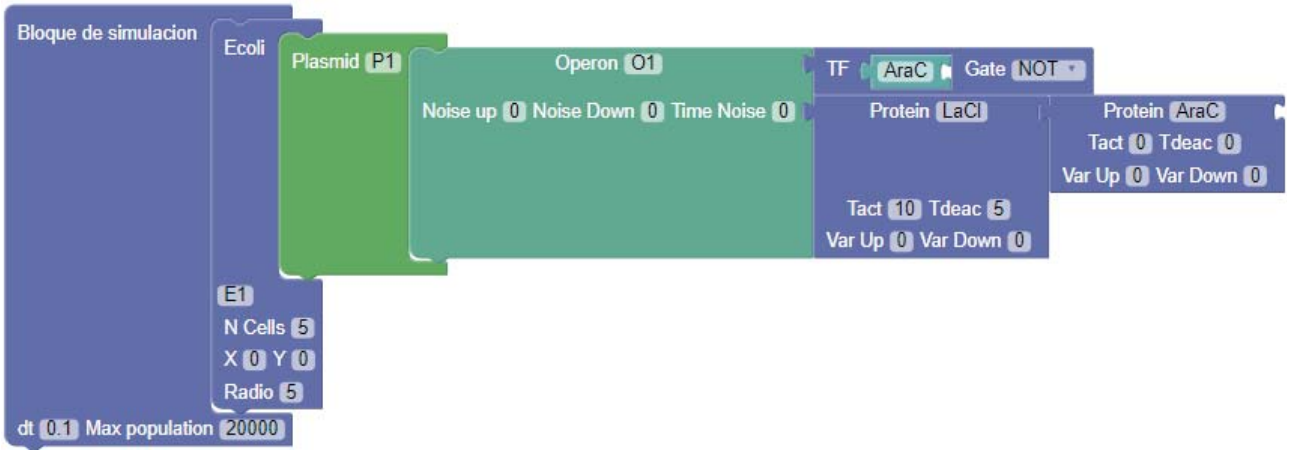
Esta distribución genera el siguiente codigo GRO:

```
include gro
set("dt",0.1);
set("population_max",200000);
genes([name:="O1",proteins:={"AraC"},
      promoter:=[function:="NOT",transcription_factors:={"AraC"
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={10},variabilities:={0}],
      prot_deg_times:=[times:={5},variabilities:={0}]]);

plasmids_genes([P1:={"O1"}]);set("dt",0.1);
```

9.2. Generación de una bacteria con varias proteínas

La distribución en bloques Grockly es la siguiente:

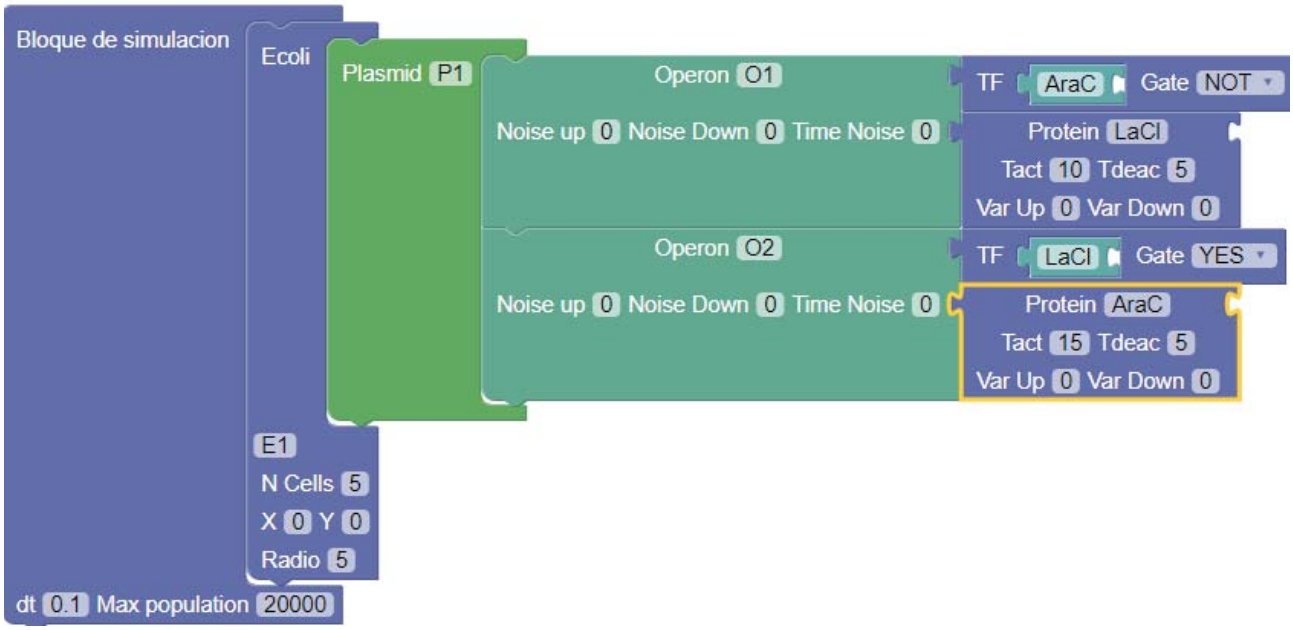


Y el codigo que genera dicha distribución es el siguiente:

```
include gro
set("dt",0.1);
set("population_max",20000);
genes([name:="O1",proteins:={"LaCl","AraC"},
      promoter:=[function:="NOT",transcription_factors:={"AraC"},,
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={10,0},variabilities:={0,0}],
      prot_deg_times:=[times:={5,0},variabilities:={0,0}]]);
plasmids_genes([P1:={"O1"}]);
program main():={c_ecolis(5,0,0,5,{"P1"},program p());};
```

9.3. Generación de una bacteria con varios operones

La distribución en Grockly de este apartado es la siguiente:



Y el codigo GRO generado por esta distribución es el siguiente:

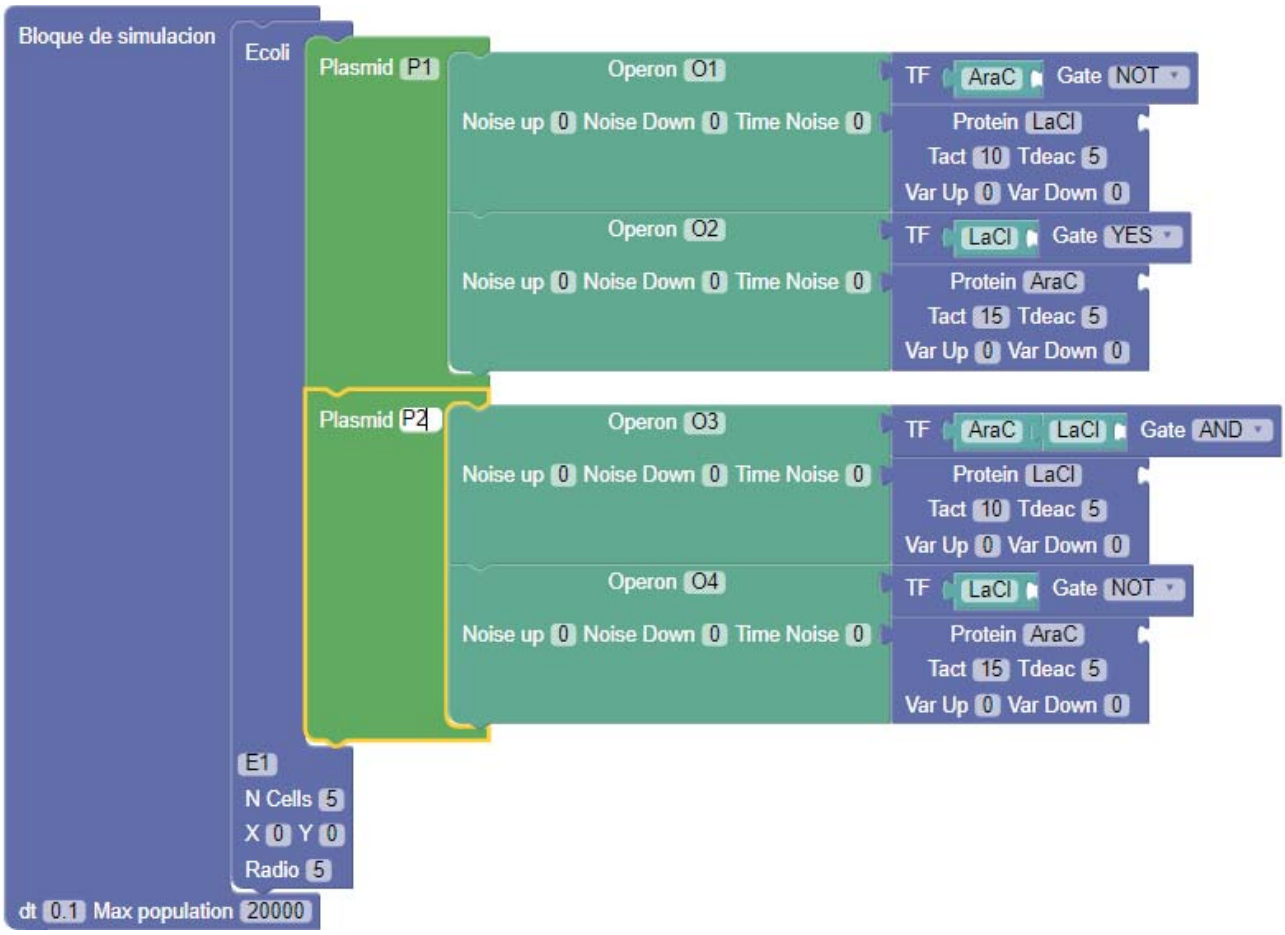
```
include gro
set("dt",0.1);
set("population_max",20000);
genes([name:="O1",proteins:={"LaCl"},
       promoter:=[function:="NOT",transcription_factors:={"AraC"},
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={10},variabilities:={0}],
       prot_deg_times:=[times:={5},variabilities:={0}]]);
genes([name:="O2",proteins:={"AraC"},
      promoter:=[function:="YES",transcription_factors:={"LaCl"},
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={15},variabilities:={0}],
      prot_deg_times:=[times:={5},variabilities:={0}]]);

plasmids_genes([P1:={"O1","O2"}]);

program p():={skip();};program main():={c_ecolis(5,0,0,5,{"P1"},program p());};
```

9.4. Generación de una bacteria con varios plásmidos

La distribución de bloques Grockly en este apartado es la siguiente:



Y el codigo GRO que genera esta distribución de bloques es la siguiente:

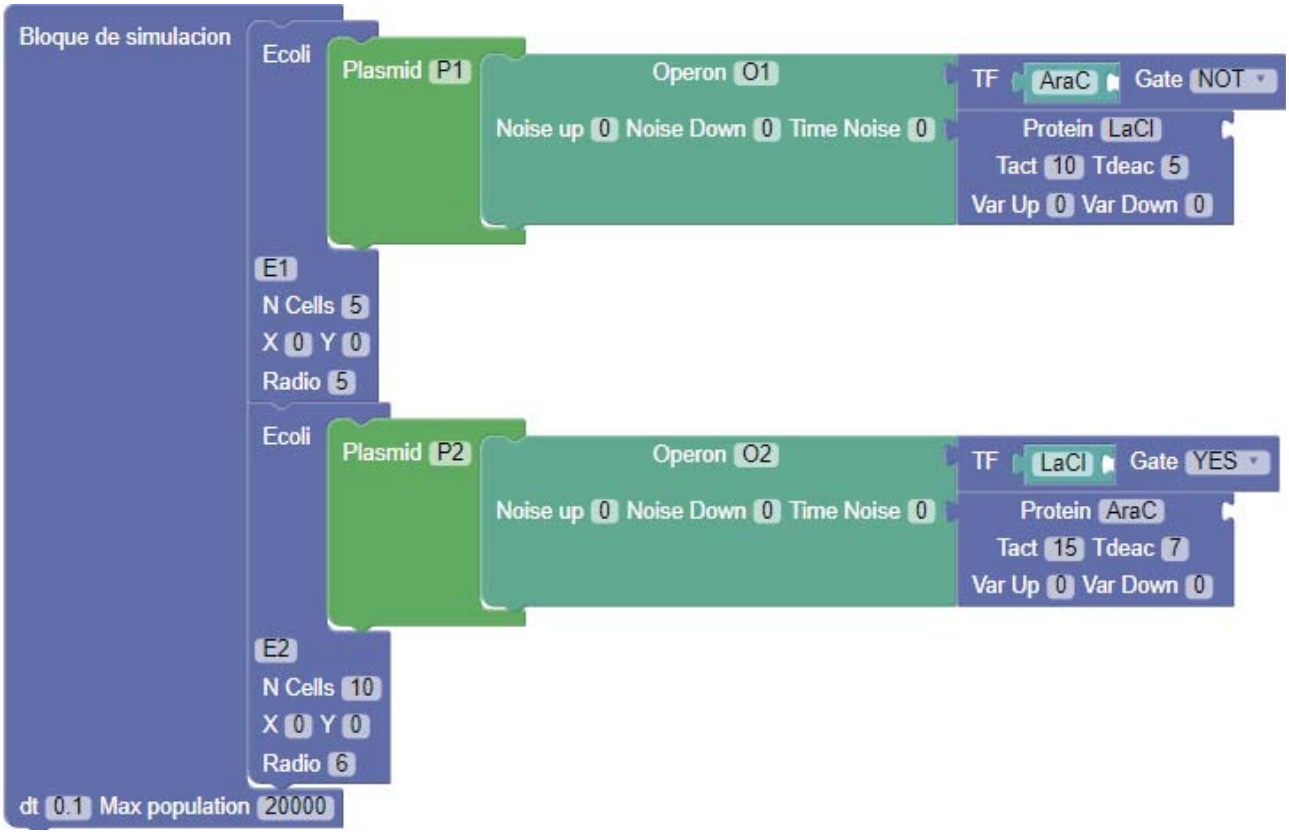
```
include gro
set("dt",0.1);
set("population_max",20000);
genes([name:="O1",proteins:={"LaCl"},
       promoter:=[function:="NOT",transcription_factors:={"AraC"},
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={10},variabilities:={0}],
       prot_deg_times:=[times:={5},variabilities:={0}]]);
genes([name:="O2",proteins:={"AraC"},
       promoter:=[function:="YES",transcription_factors:={"LaCl"},
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={15},variabilities:={0}],
       prot_deg_times:=[times:={5},variabilities:={0}]]);
genes([name:="O3",proteins:={"LaCl"},
       promoter:=[function:="AND",transcription_factors:={"AraC","LaCl"},
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={10},variabilities:={0}],
       prot_deg_times:=[times:={5},variabilities:={0}]]);
genes([name:="O4",proteins:={"AraC"},
       promoter:=[function:="NOT",transcription_factors:={"LaCl"},
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={15},variabilities:={0}],
       prot_deg_times:=[times:={5},variabilities:={0}]]);

plasmids_genes([P1:={"O1","O2"},P2:={"O3","O4"}]);

program p():={skip();};program main():={c_ecolis(5,0,0,5,{"P1","P2"},program p());};
```

9.5. Generación de varias bacterias (simples)

La implementación e varias bacterias con circuitos simples en su interior es la siguiente:

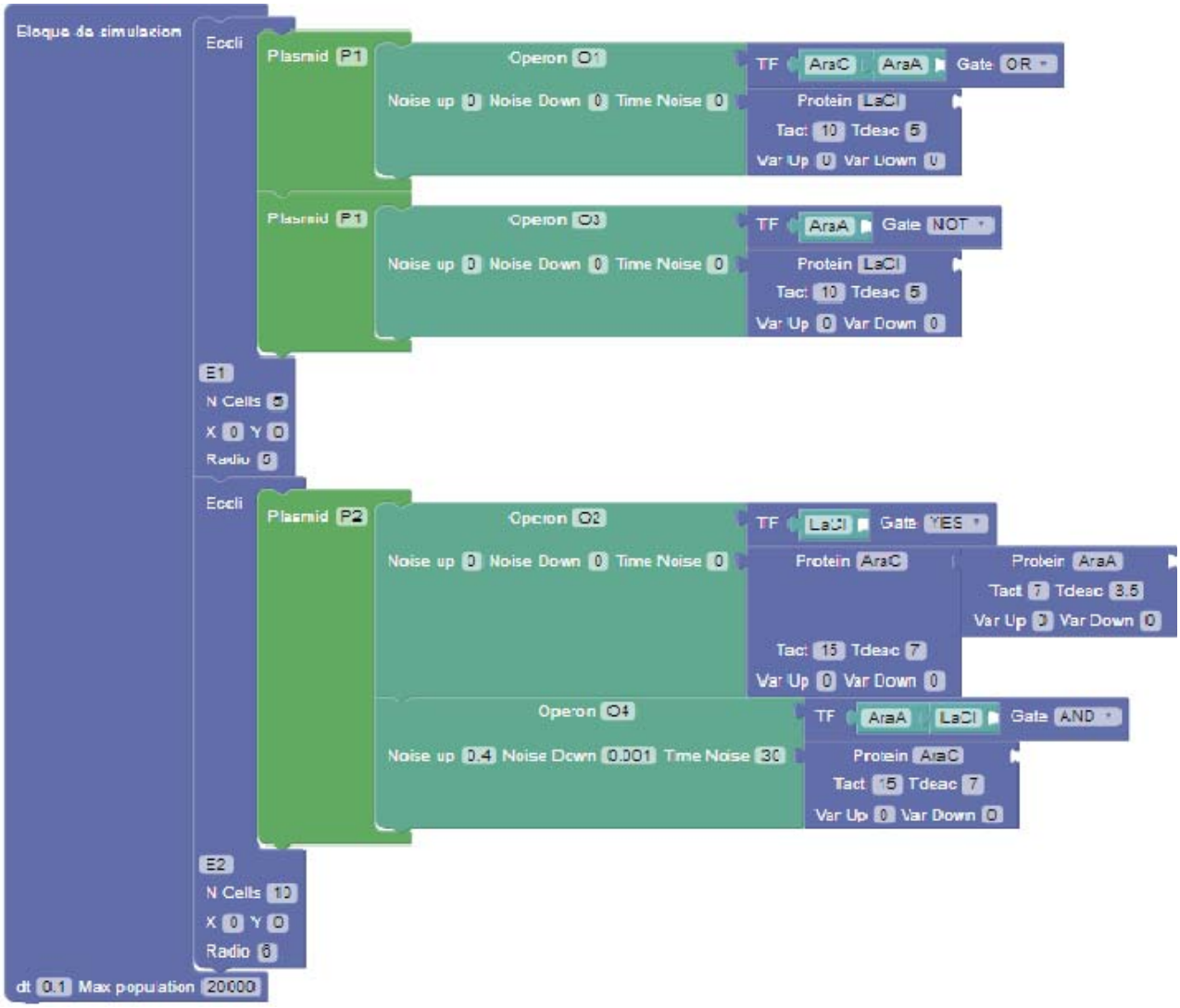


La cual genera la el siguiente codigo para GRO:

```
include gro
set("dt",0.1);
set("population_max",20000);
genes([name:="O1",proteins:={"LaCl"},
      promoter:=[function:="NOT",transcription_factors:={"AraC"},
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={10},variabilities:={0}],
      prot_deg_times:=[times:={5},variabilities:={0}]]);
genes([name:="O2",proteins:={"AraC"},
      promoter:=[function:="YES",transcription_factors:={"LaCl"},
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={15},variabilities:={0}],
      prot_deg_times:=[times:={7},variabilities:={0}]]);
plasmids_genes([P1:={"O1"},P2:={"O2"}]);
program main():={c_ecolis(5,0,0,5,{"P1"},program p());c_ecolis(10,0,0,6,{"P2"},program p());};
```

9.6. Generación de varias bacterias (complejas)

La distribución de bloques Grockly de este apartado es la siguiente:



La cual genera el siguiente codigo GRO:

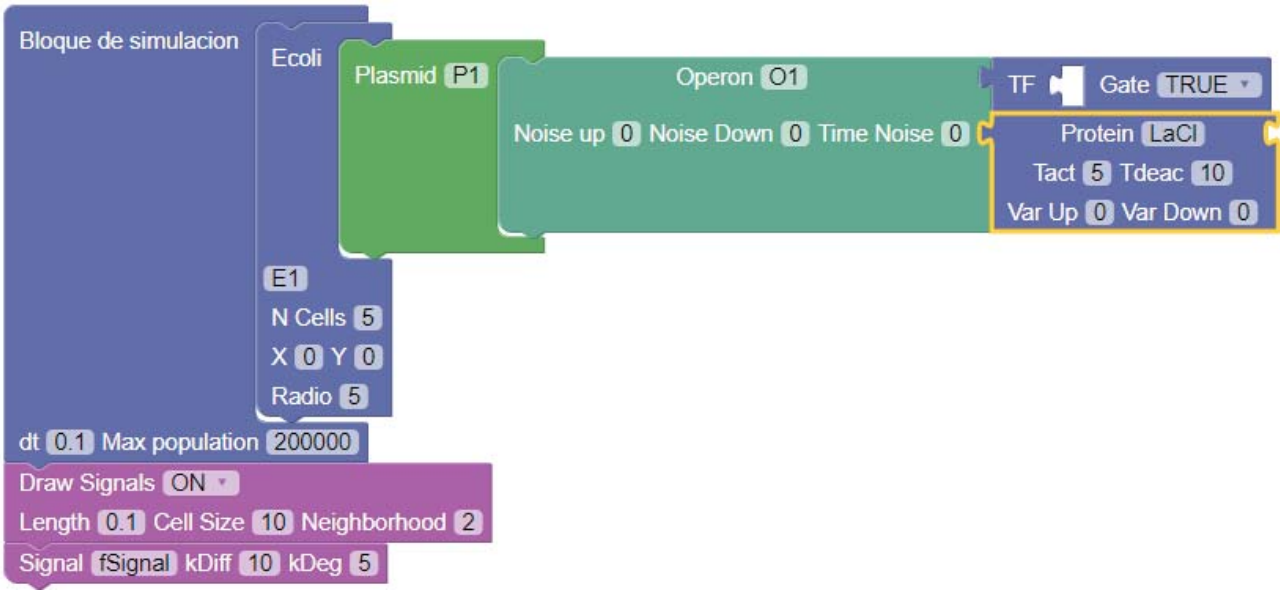
```
include gro
set("dt",0.1);
set("population_max",20000);
genes([name:="O1",proteins:={"LaCl"},
      promoter:=[function:="OR",transcription_factors:={"AraC","AraA"},
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={10},variabilities:={0}],
      prot_deg_times:=[times:={5},variabilities:={0}]]);
genes([name:="O3",proteins:={"LaCl"},
      promoter:=[function:="NOT",transcription_factors:={"AraA"},
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={10},variabilities:={0}],
      prot_deg_times:=[times:={5},variabilities:={0}]]);
genes([name:="O2",proteins:={"AraC","AraA"},
      promoter:=[function:="YES",transcription_factors:={"LaCl"},
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={15,7},variabilities:={0,0}],
      prot_deg_times:=[times:={7,3.5},variabilities:={0,0}]]);
genes([name:="O4",proteins:={"AraC"},
      promoter:=[function:="AND",transcription_factors:={"AraA","LaCl"},
      noise:=[toOff:=0.4,toOn:=0.001,noise_time:=30]],
      prot_act_times:=[times:={15},variabilities:={0}],
      prot_deg_times:=[times:={7},variabilities:={0}]]);

plasmids_genes([P1:={"O1"},P2:={"O3"},P2:={"O2","O4"}]);
program p():={skip();};
program main():={c_ecolis(5,0,0,5,{"P1","P2"},program p());c_ecolis(10,0,0,6,{"P2"},program p());};
```

9.7. Validación de señales

Para simplificar la distribución de bloques y del codigo generado por estos, utilizaremos una distribución de bloques genéticos simple para realizar la validación de los distintos módulos que posee GRO y por lo tanto Grockly.

La distribución de los bloques que codifican las señales en GRO es la siguiente:



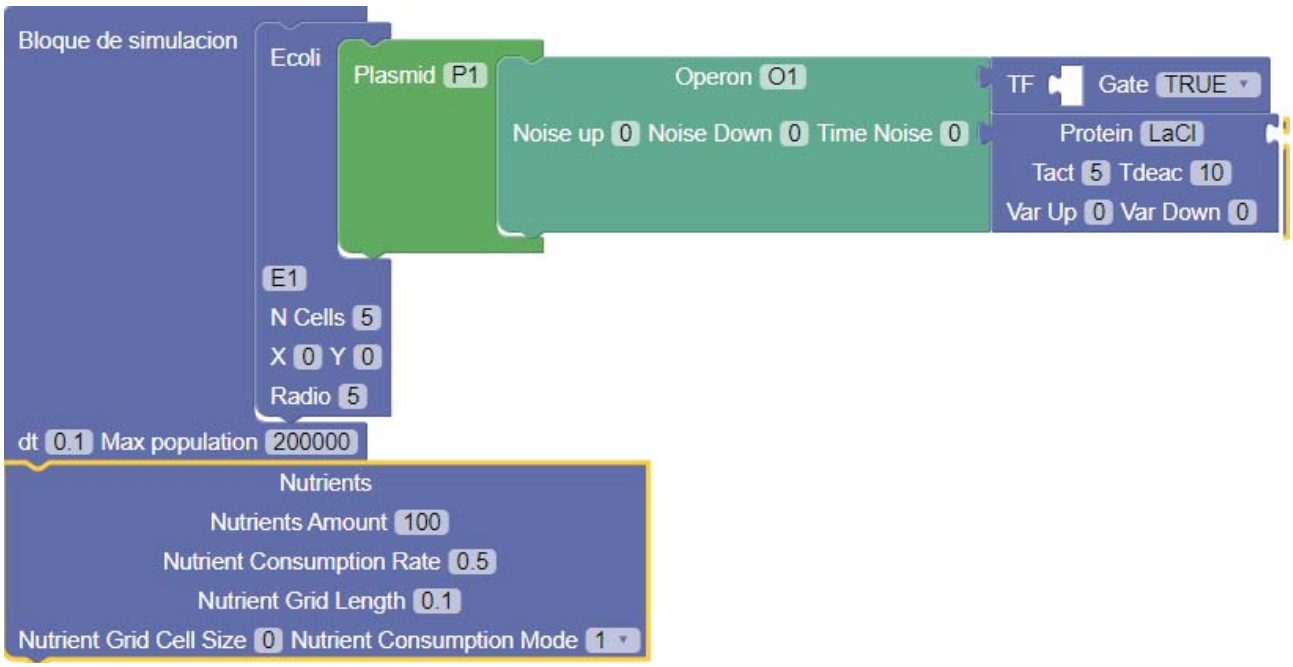
Del mismo modo, el codigo que genera esta distribución es el siguiente:

```
include gro
set("dt",0.1);
set("population_max",200000);
set("signals", 1.0);
set_param("signals_grid_length",0.1);
set_param("signals_grid_neighborhood",2);
fSignal:= s_signal([kdiff:=10,kdeg:=5]);
genes([name:="O1",proteins:={"LaCl"},
       promoter:=[function:="TRUE",transcription_factors:={},
       noise:=[toOff:=0,toOn:=0,noise_time:=0]],
       prot_act_times:=[times:={5},variabilities:={0}],
       prot_deg_times:=[times:={10},variabilities:={0}]]);
       plasmids_genes([P1:={"O1"}]);set("dt",0.1);

program p():={skip();};
program main():={c_ecolis(5,0,0,5,{"P1"},program p());};
```

9.8. Validación de nutrientes

Los bloques que codifican la información relativa al modulo de nutrientes de GRO es:



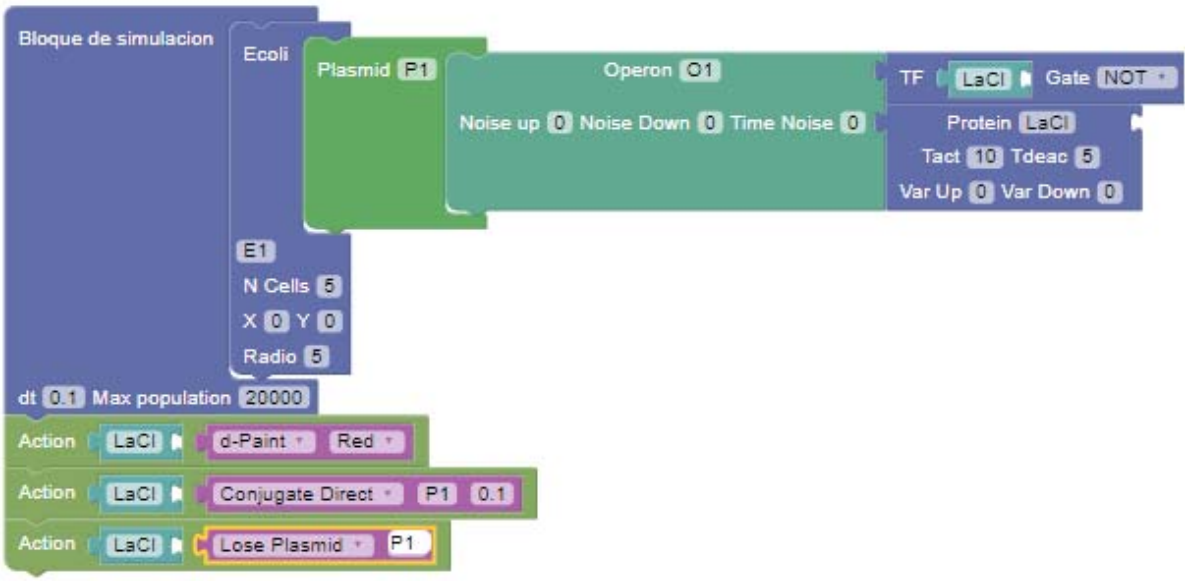
Y el codigo que genera esta distribución es la siguiente:

```
include gro
set("dt",0.1);
set("population_max",200000);
set("nutrients", 1.0);
set("nutrient_consumption_rate",0.5);
set("nutrient_grid_length",0.1);
set("nutrient_grid_cell_size",0);
set("nutrient_consumption_mode",1);
genes([name:="O1",proteins:={"LaCl"},
      promoter:=[function:="TRUE",transcription_factors:={},
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={5},variabilities:={0}],
      prot_deg_times:=[times:={10},variabilities:={0}]]);
plasmids_genes([P1:={"O1"}]);set("dt",0.1);

program p():={skip();};
program main():={c_ecolis(5,0,0,5,{"P1"},program p());};
```

9.9. Validación de acciones

Como las acciones tienen una estructura similar entre ellas, crearemos una distribución con algunas de ellas:



La distribución anterior genera el siguiente codigo:

```
include gro
set("dt",0.1);
set("population_max",20000);
genes([name:="O1",proteins:={"LaC1"},
      promoter:=[function:="NOT",transcription_factors:={"LaC1"},
      noise:=[toOff:=0,toOn:=0,noise_time:=0]],
      prot_act_times:=[times:={10},variabilities:={0}],
      prot_deg_times:=[times:={5},variabilities:={0}]]);
plasmids_genes([P1:={"O1"}]);set("dt",0.1);
action({"LaC1"},"d_paint",{"0","3200","0","0"});
action({"LaC1"},"conjugate_directed",{"P1","0.1"});
action({"LaC1"},"lose_plasmid",{"P1"});
program p():={skip();};
program main():={c_ecolis(5,0,0,5,{"P1"},program p());};
```

Bibliografía

-. https://developers.google.com/blockly/

-. https://scratch.mit.edu/

-. Vishal Gupta, Jesús Irimia, Iván Pau, Alfonso Rodríguez-Patón *"Bioblocks: Programming protocols in biology made easier"*, *ACS Synth. Biol.*,6(7), pp 1230–1232

-. Martín Eduardo Gutiérrez, Paula Gregorio-Godoy, Guillermo Pérez del Pulgar, Luis Enrique Muñoz, Sandra Sáez, Alfonso Rodríguez-Patón *"A new improved and extended version of the multicell bacterial simulator gro"*, *ACS Synth. Biol.*, 6(8), pp 1496–1508

-. http://vps159.cesvima.upm.es/

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Fri Jan 26 11:54:49 CET 2018
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)