

COMPETENCIA DE VIDA ARTIFICIAL

Introducción



LA COMPETENCIA

Esta competencia consiste en diseñar microorganismos que sepan moverse para conseguir alimento, lograr reproducirse, luchar contra otra especie o, en caso de tener poca energía, huir de ellos para no morir en un eventual combate.

Cada concursante desarrollará el algoritmo de supervivencia de sus microorganismos (MO) que luego competirán con otras colonias por la exclusividad del medio. El lenguaje utilizado podrá ser C++ o java.

ORGANIZACIÓN DEL ENCUENTRO

Con todos aquellos que estén interesados en participar, se realizará una reunión introductoria y se definirán el día y hora de los encuentros semanales.

Los documentos, códigos fuentes del entorno donde se desarrolla la competencia, ejemplos de MO simples para probar sus algoritmos de competencia y otra información están en SourceForge, en los siguientes sitios:

Reglas del juego de la competencia:
<http://alifecontest.wikidot.com>

Artificial Life Contest

Download aLifeC Past Winners Search this site

>Welcome to the ALife Contest!

Artificial Life Contest (ALC) is a competition of artificial microorganisms (MOs). The artificial life models are simulated by software. Each contender develops the survival algorithm for his MOs. Documentation, game rules and source code are provided, as well as several examples.

The documentation, game rules and source codes for the simulation of the artificial environment and graphics are all freely available: the petri class, the agar, the colonies, an abstract microorganism and several examples of simple MOs. They are available in this site.

The last sourcecode version can be downloaded from [here](#).

Besides, we suggest you to subscribe you to the [email list](#) to be always informed.

Organizing Universities from Argentina: [FI-UNER](#), [FICH-UNL](#) ([sinc\(i\)](#)), [FRSF-UTN](#) ([CIDISI](#)).


page_revision: 7, last_edited: 20 Jan 2010, 14:31 GMT (221 days ago)

Edit Tags History Files Print Site tools + Options

La documentación y las reglas del juego de la competencia están disponibles en este Wiki. Podés bajarte la última versión del código fuente desde:

<http://sourceforge.net/projects/alifecontest/files>

SourceForge.net > Find Software > Artificial Life Contest > Browse Files




Artificial Life Contest Beta

by [cesarmart](#), [dmilone](#), [mboscovich](#), [motoko_kusanagi](#), [nandekudasai](#), ...

[Summary](#) | [Files](#) | [Support](#) | [Develop](#)

This project is aimed to the development of a framework for alife contests: the main rules of life, a graphical interface and some examples of elementary microorganisms (more complex survival strategies are developed by the contestants).

Download Now!




















OR

View all files >

Com_vida para Java y C++ (709.7 KB)

Browse Files for Artificial Life Contest

| File/Folder Name | Platform | Size | Date ↓ | Downloads | Notes/Subscribe |
|--|---|----------|------------|-----------|---|
| Newest Files | | | | | |
|  alifecontest-java-003.zip Com_vida para Java y C++ |  | 709.7 KB | 2010-08-16 | 32 | |
| All Files  | | | | | |
| ▼  alifecontest-java | | 1.1 MB | 2010-08-16 | 283 |   |
|  alifecontest-java-003.zip Com_vida para Java y C++ |  | 709.7 KB | 2010-08-16 | 32 | |
|  alifecontest-java-002.zip | | 208.4 KB | 2009-11-24 | 141 | |
|  alifecontest-java-001.zip | | 136.5 KB | 2009-03-13 | 110 |  |
| ▶  microorganisms | | 254.9 KB | 2008-06-22 | 1,567 |   |
| ▶  alifecontest | | 1.0 MB | 2008-03-22 | 847 |   |

En este sitio también se publica el código fuente del ganador de cada encuentro.

Además, te recomendamos suscribirte ahora mismo a la lista de correo electrónico así estás siempre informado en intercambiar comentarios, ideas, propuestas, etc.:

<http://lists.sourceforge.net/mailman/listinfo/alifecontest-microorganisms>

Email Archive: [alifecontest-microorganisms](#) (read-only)

| | | | | | | | | | | | | |
|-------|-----|-----|----------|----------|---------|----------|---------|---------|----------|----------|----------|----------|
| 2007: | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct (9) | Nov (1) | Dec |
| 2008: | Jan | Feb | Mar (33) | Apr (25) | May (6) | Jun (23) | Jul (2) | Aug (1) | Sep (35) | Oct (13) | Nov (36) | Dec (13) |
| 2009: | Jan | Feb | Mar (8) | Apr (2) | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec |

alifecontest-microorganisms Ultimate Show 25 Change View

| Topic | Topic Starter | Thread Posts | Last Post |
|---|---|--------------|------------------|
| Re: [Alifecontest-microorganisms] ACM | Diego Milone <d.milone@ie...> | 2 | 2009-04-24 15:01 |
| Re: [Alifecontest-microorganisms] [Alifecontest-developers] Nueva versión del comvidal | Sergio Del Castillo <yeyo_32141759@ya...> | 5 | 2009-03-30 00:45 |
| Re: [Alifecontest-microorganisms] Invitación ACM | Mariano Obeld <geobeld@gm...> | 2 | 2009-03-29 22:00 |
| [Alifecontest-microorganisms] Virtual Embryos | Diego Milone <d.milone@ie...> | 1 | 2009-03-06 13:36 |
| [Alifecontest-microorganisms] 3 Equipos argentinos van a la final de la competencia ACM de programación | Diego Milone <d.milone@ie...> | 1 | 2008-12-09 14:34 |
| Re: [Alifecontest-microorganisms] Alifecontest-microorganisms Digest, Vol 9, Issue 1 | Juan Bertinetti <juanbertinetti@gm...> | 7 | 2008-12-05 18:25 |

Las listas de correo son el principal medio de comunicación entre los participantes, así que es muy importante suscribirse. Al enviar un correo a esta lista el mensaje es distribuido automáticamente a todos los suscriptos. Todos los correos son archivados.

Se solicita a los competidores que utilicen un ID de email que incluyan su nombre y apellido para que podamos identificarnos fácilmente.

LOS ELEMENTOS DE LA SIMULACIÓN

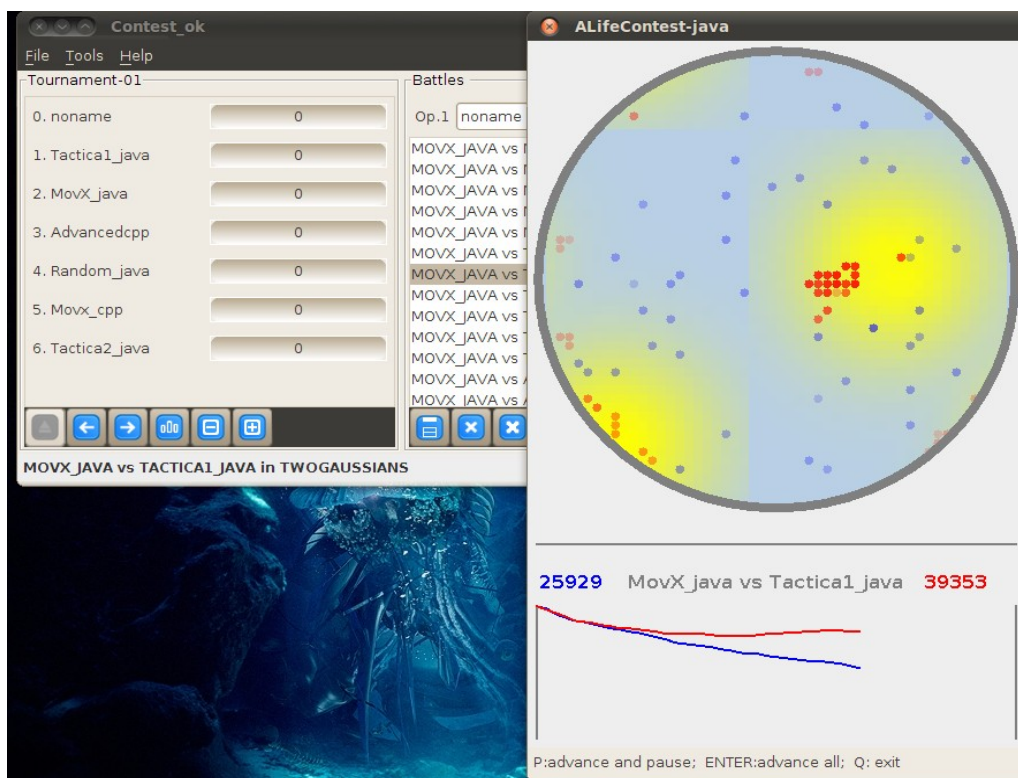
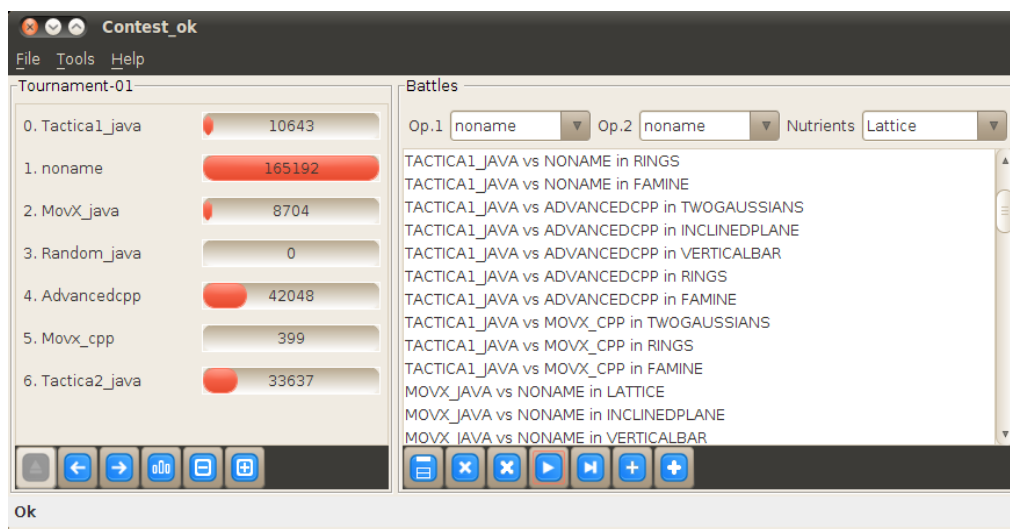
Teniendo en cuenta que el desarrollo del código es en el lenguaje C++ y en base a un Diseño Orientada a Objetos, se detallan a continuación los lineamientos generales de las clases que forman parte del programa de vida artificial.

Existe una clase a partir de la cual el concursante deberá desarrollar el método propio de supervivencia de sus microorganismos. Hay otras clases de las cuales se crearán objetos que serán de los organizadores del encuentro y que el concursante no podrá modificar pero si estará disponible su código para un mejor conocimiento de su funcionamiento. Algunos de estos objetos estarán accesibles para que los microorganismos puedan obtener datos útiles en el momento de tomar decisiones estratégicas.

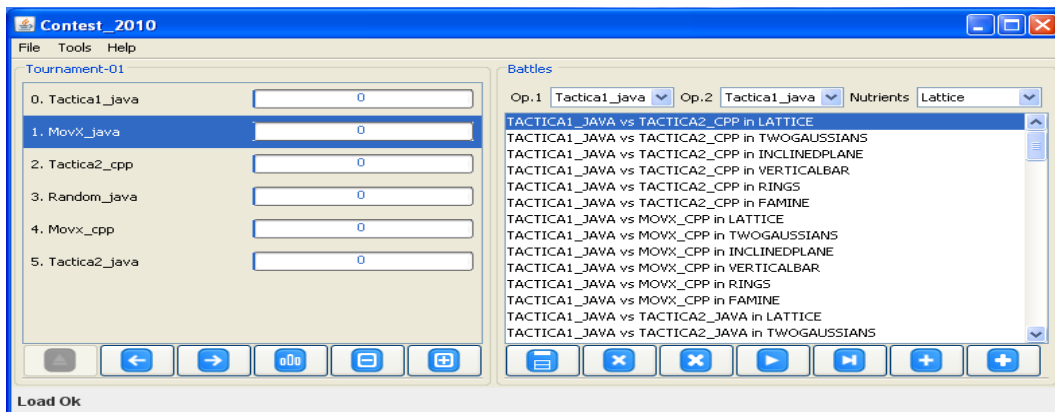
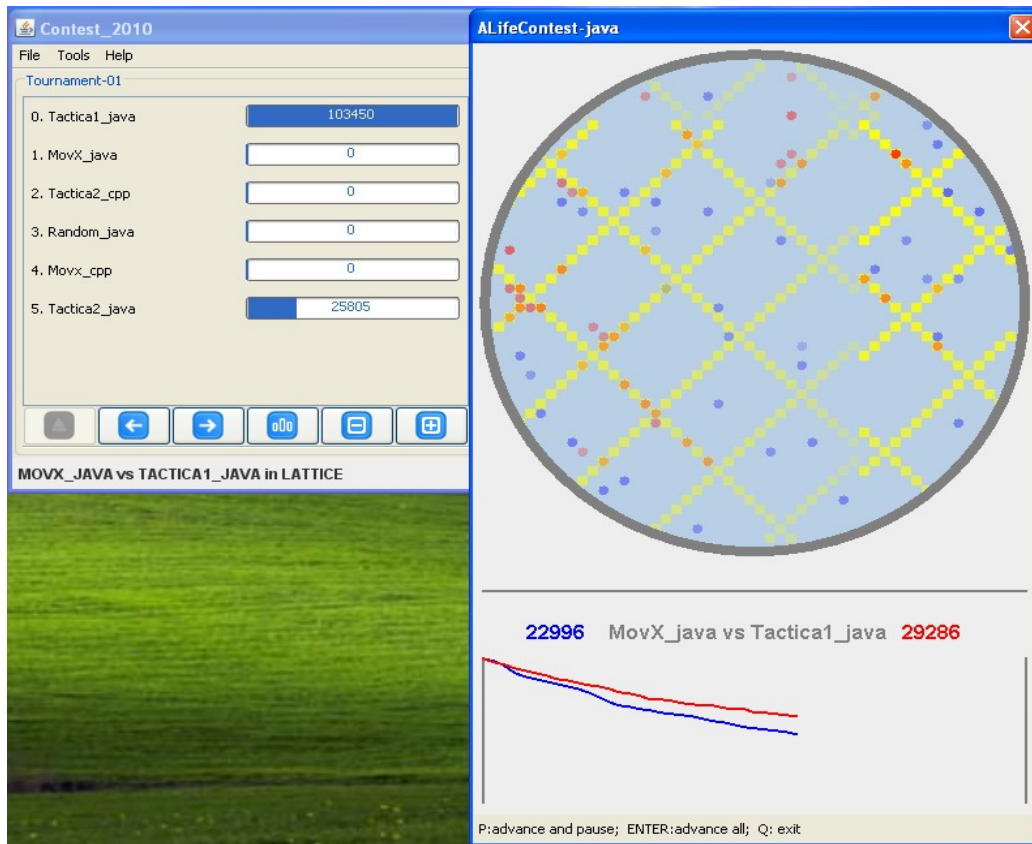
La salida gráfica de la simulación estará a cargo de los árbitros. Se agradece toda propuesta de mejora en la simulación: diseño, codificación, presentación gráfica, etc.

SCREENSHOTS

Las siguientes pantallas son ejemplos de colonias de microorganismos y la cantidad de nutrientes en la superficie. Se presenta el sistema que simula el entorno de la competencia donde se observa la cantidad de MO de las colonias oponentes y ventanas de selección de la distribución de nutrientes.



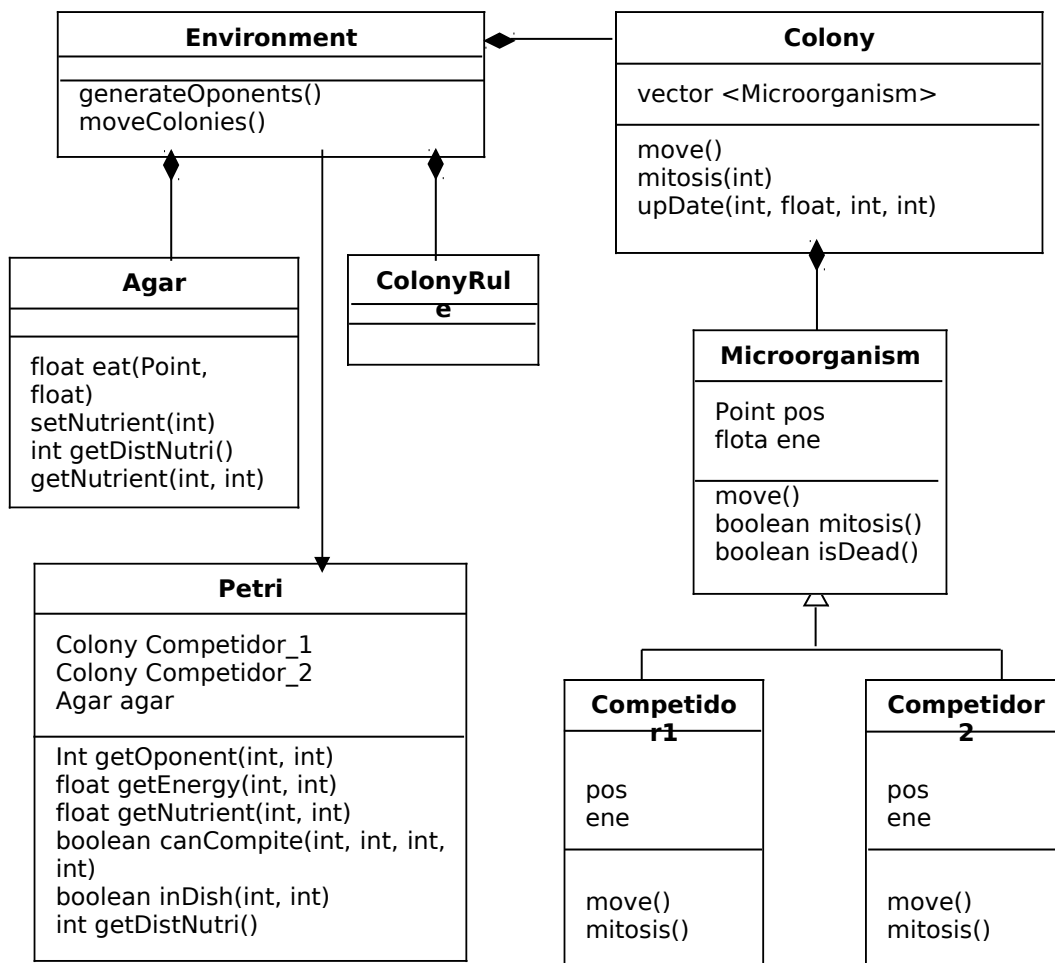
Pantallas de la versión 0.03 en Ubuntu 10.04 LTS.

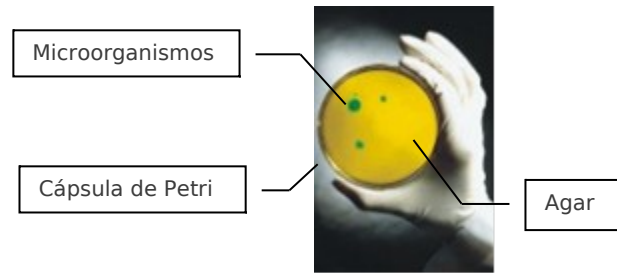
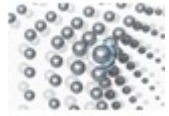


Pantallas de la versión 0.03 en Windows XP.

DISEÑO DEL PROGRAMA

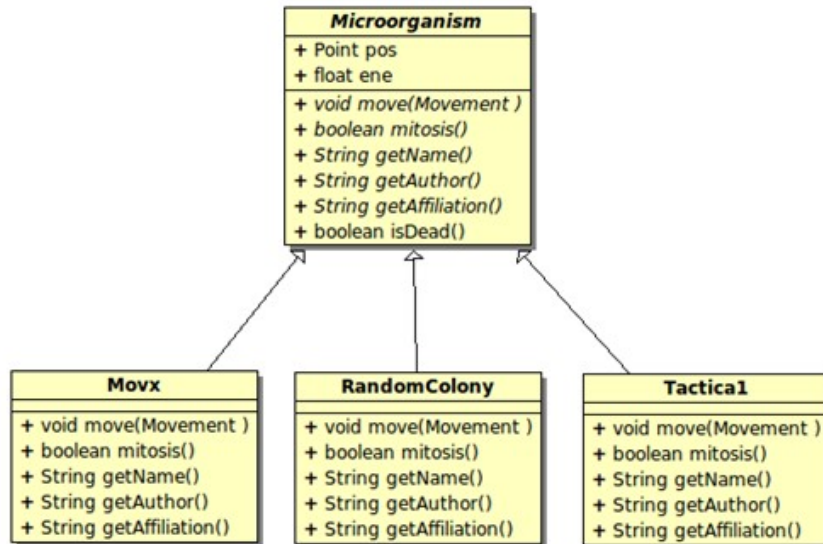
El siguiente diagrama simplificado de las clases muestra el modelo del *Competidor1* y del *Competidor2*, que son las clases de los microorganismos desarrollados por los competidores a partir de los cuales se realizarán las colonias. El método más importante es **move**. El alimento para los MO es administrada por la clase **Agar** y la determinación de las relaciones entre los microorganismos y su supervivencia las resuelve **Environment**.





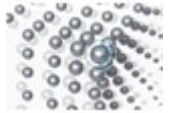
LA CLASE MICROORGANISM

Esta clase declara los métodos básicos del comportamiento, de la cual deberá heredar el MO concursante.



En la clase descendiente que debe desarrollar el concursante, a través del método **move** le indicará a donde quiere moverse cuando sea su turno. Cuando la colonia invoca el método **mitosis** del microorganismo, este indica si se quiere dividir en dos. El método **update** es desde donde recibe el identificador (**id**) con el cual está compitiendo, sus coordenadas y energía antes de pedir que devuelva el movimiento deseado, es decir que estos atributos del MO le son informados. Este método **update** es implementado únicamente en la clase madre **Microorganism** por lo que no es necesario que el competidor lo redefina en su MO.

El siguiente código es un ejemplo simple de la clase de un microorganismo creado para la competencia. Se llamó **BuscaN** porque su estrategia simplemente consiste en moverse al casillero de su entorno que posea más nutrientes.



```

#ifndef BUSCAN_H
#define BUSCAN_H

#include<cstdlib>
#include <iostream>
using namespace std;
#include "Microorganism.h"
#include "Petri.h"

class BuscaN: public Microorganism{
public:
BuscaN(){}

void move(Movement &mov) {
    int x_rel, y_rel; // posición relativa de prueba.
    int x_max, y_max; // posición relativa con más nutrientes.

    x_max= y_max=0;

    for(x_rel=-1; x_rel<2; x_rel++){
        for(y_rel=-1; y_rel<2; y_rel++){
            if(!petri.inDish(pos.x+x_rel,pos.y+y_rel)) continue; //verifica que esté
dentro.

            if (petri.getNutrient(pos.x+x_rel,pos.y+y_rel) >
                petri.getNutrient(pos.x+x_max,pos.y+y_max)){
                x_max=x_rel;
                y_max=y_rel;
            }
        }
    }

    mov.dx=x_max;
    mov.dy=y_max;
};

bool mitosis(){
    return false;
};

string getName(){
    return "BuscaN";
};

string getAuthor(){
    return "Author";
};

string getAffiliation(){
    return "UTN-FRSF";
};
};

```

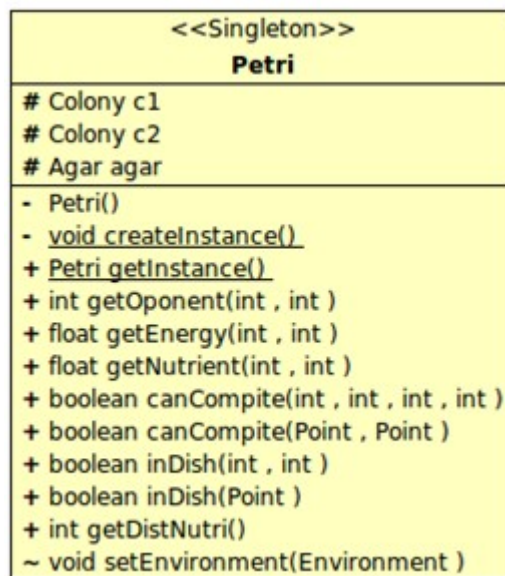

#endif

A través de `petri.getNutrient()` se pide colaboración al agar para conocer la cantidad de nutrientes que hay en cualquier posición de la cápsula de Petri.

En este caso se implementó una mitosis tan simple como: “si el MO tiene más de 5000 unidades de energía entonces pide dividirse”

Este MO es muy simple debido a que solamente considera la posición donde hay más nutrientes y no tiene en cuenta las posiciones de los enemigos, ni las de su propia colonia. Algoritmos más desarrollados podrán hacer uso de más información para conseguir estrategias más efectivas. Por ejemplo la de realizar mitosis de acuerdo a la cantidad de nutrientes disponibles, atacar a los MO contrarios, buscar y conservar copadas las regiones con más nutrientes, etc.

LA CLASE PETRI

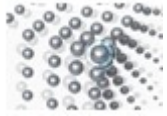


Esta es la clase más importante para el competidor por que es la encargada de brindar información a los microorganismos. Cada competidor puede invocarla para obtener información necesaria para tomar decisiones estratégicas. Los métodos disponibles son:

int **getOponent**(int x, int y): retorna un identificador de la clase del MO que se encuentra en la posición (x,y). si la posición está libre se retorna 0.

float **getEnergy**(int x, int y): retorna la energía del MO que se encuentra en la posición (x,y). si la posición está libre se retorna 0.

float **getNutrient**(int x, int y): retorna la cantidad de nutrientes que hay en la posición (x, y).



int **getDistNutri()**: retorna un identificador único que representa la distribución de nutrientes actual que se utiliza en una batalla.

Además, la clase Petri provee dos métodos para facilitar la programación de un MO:

boolean **canCompite()**: retorna true si el MO en la posición (x1,x2) es oponente del MO en la posición (y1,y2).

boolean **inDish()**: retorna true si el MO en la posición (x,y) esta dentro del entorno, es decir, pertenece al círculo del centro (Defs.Radius, Defs.Radius) y radio Defs.Radius. **Defs** es una clase que define las constantes importantes para la competencia, por ejemplo: cantidad de energía inicial de cada MO (E_INITIAL), radio del entorno (RADIOUS), cantidad de MOs que se crean al inicio de la competencia (MO_INITIAL), entre otras

int **getBattleId()**: retorna un identificador que es único para cada batalla.

int **getLiveTime()**: representa el tiempo de simulación en una batalla.

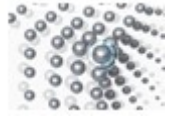
La posición y la energía actual del MO están siempre actualizadas en sus variables internas pos y ene (esto se hace siempre desde la clase **Environment** antes de llamar al método move). Mediante el método move el competidor simplemente debe devolver, a través del parámetro por referencia mov, el movimiento relativo que quiere realizar.

Si por ejemplo se quiere desarrollar una estrategia que tenga en cuenta a los MO adversarios, se podrá hacer preguntas como las siguientes:

```
If (id != petri.getOponent(miX+1,miY))
{ // tomará acción con el adversario que está a la derecha
  ...
  If (ene > petri.getEnergy(miX+1,miY))
  { // tomará acción a partir de las energías, por ejemplo: luchar...
    ...
  }
}
```

LA CLASE COLONY

Modela una colonia de microorganismos. Cada colonia de microorganismo puede estar implementada tanto en C/C++ como en Java. Inicialmente cada participante tiene, en su colonia, 50 microorganismos que luego pueden ir duplicándose (este proceso se llama **mitosis**) e incrementando el número de microorganismos, pero éstos también pueden morir.



| Colony |
|---|
| ~ int id |
| ~ String path |
| # String name= "" |
| # String author= "" |
| # String affiliation= "" |
| + Colony(int , String) |
| # boolean createMO(Point , float) |
| # void kill(int) |
| # Movement move(int) |
| # boolean mitosis(int) |
| + float getEnergy() |
| + String getAuthor() |
| + String getName() |
| + String getAffiliation() |
| + boolean isDied() |
| + void update(int , float , int , int) |

LA CLASE ENVIRONMENT

La competencia administra la clase **Environment**, que aplica las reglas de convivencia de los MO.

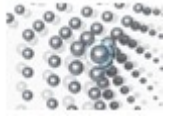
| Environment |
|-------------------------------------|
| + Environment(String) |
| + void generateOponents(BattleRun) |
| + boolean moveColonies() |

Para cada movimiento, los MO son ordenados al azar y luego se va solicitando a cada uno el movimiento deseado y si quiere dividirse (mitosis) y luego se aplican las reglas de la competencia. Con cada movimiento se actualiza el **agar**, o sea que siempre van a esta viendo en el petri lo que realmente va a haber al momento de realizar su movimiento.

Cuando la colonia solicita a cada MO que proponga su movimiento, la respuesta es analizada por **Environment** y entonces puede pasar que se cumplan las siguientes reglas:

1) El lugar esté vacío y dentro de los límites de la cápsula: el movimiento será realizado por Petri.

2) El lugar esté ocupado por un MO de otra colonia: se origina una lucha. En una lucha gana el que tiene mayor energía pero el ganador no mata simplemente al perdedor sino que le quita una cantidad de energía igual a la diferencia de energía entre ambos. Si después de esto el perdedor quedara con energía menor a cero entonces sí se muere. Ambos quedan en el mismo lugar que estaban antes de la lucha y el ganador decremента su energía en un 7.5% de la que tenga el perdedor.



En caso de que se solicite una división mediante mitosis, la reproducción tendrá éxito siempre que exista un lugar libre para poner al hijo en alguna de las 8 celdas adyacentes. Se busca un lugar vacío al rededor recorriendo las celdas con un orden al azar y si hay lugar se crea al hijo en ese lugar y ambos (padre e hijo) quedan con el 49% de la energía original del padre.

Otras reglas que se aplican siempre son:

De cada colonia nacen inicialmente 50 MO con 1000 unidades de energía cada uno y en posiciones al azar.

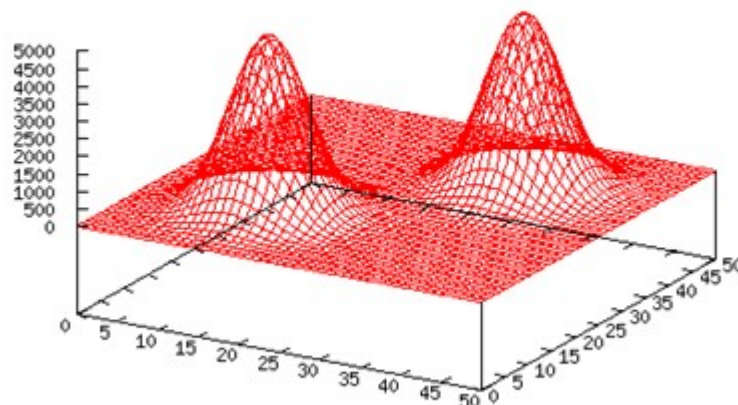
Todos los MO pierden 5 unidades de energía (por envejecimiento) en cada unidad de tiempo.

Cuando un MO se mueve pierde 10 unidades de energía (por el esfuerzo).

Todos los MO comen (incrementan su energía) en un 1% de los nutrientes que existan en su posición.

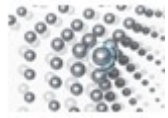
La distribución inicial de energía y la forma en que podrían variar los nutrientes en la superficie a medida que se desarrolla la competencia será elegida por los árbitros en cada encuentro. Hay varias distribuciones y habrá una batalla en cada una ellas.

Por ejemplo, en el gráfico siguiente se muestra una distribución de nutrientes con dos picos (distribución gaussiana bidimensional) donde en este caso en particular la conquista y permanencia de una colonia en las zonas de mayor energía otorgaría una importante ventaja.



DEFINICIÓN DEL GANADOR

Ganar una **batalla** consiste en eliminar a todos los MO adversarios. Se tendrá en cuenta la suma de la energía de los MO que sobrevivieron.



En cada **torneo**, que consistirá en un encuentro con todos los participantes en una fecha indicada, se realizarán 5 batallas de a dos competidores con una distribución de comida distinta. Todas las batallas entre dos colonias se realizarán con las mismas cinco distribuciones de nutrientes. Cada **torneo** será ganado por quien obtenga la mayor energía total, calculada sumando la energía de todos las competencias en que intervino (incluyendo sólo los ganados). El ganador de un torneo sumará 3 puntos en la tabla general de posiciones. De la misma forma se definirá el segundo puesto (obteniendo 2 puntos) y el tercero (obteniendo 1 punto).

En función del número de participantes los árbitros determinarán si el torneo se realiza “todos contra todos” o por “simple eliminación”. Un torneo inicia y finaliza con un único código fuente por participante, el que puede ser mejorado de un torneo a otro. El código fuente del ganador de cada torneo es publicado en el sitio de Internet creado para esta competencia.

Cuando algún participante falte a un torneo, su MO participará con el código presentado en el último torneo. Se considera que un participante que falta a 3 torneos seguidos se ha retirado de la competencia.

Se considerará ganador final de la **competencia** al participante que acumuló más puntos a lo largo de todos los torneos realizados semana a semana. La cantidad de torneos será determinada durante la competencia.

Se tendrá dos tipos de competencias. **Principiantes** (sin participación previa en la competencia) y **Avanzados** (con participación en alguna competencia anterior). Para reducir la complejidad en el caso de los principiantes se competirá con una única distribución de nutrientes.

Al finalizar la competencia se realizará una presentación de las estrategias utilizadas, donde cada participante podrá exponer ante los demás, en una presentación de no más de 15 minutos, una descripción de las estrategias y técnicas de programación utilizadas.

ÁRBITROS

Los árbitros de la competencia son los encargados de:

Organizar cada torneo.

- Mantener las listas de puntuaciones y determinar los ganadores.
- Mantener actualizado el código fuente de la simulación y asegurar que se utilice la versión original en cada batalla.
- Organizar cada encuentro asegurando el lugar físico y equipamiento.
- Mantener informado a todos los participantes.

- Ayudar a los principiantes a desarrollar sus primeros microorganismos.
- Intervenir en casos donde no se respeten los reglamentos o se incurra en cualquier acción inapropiada.
- Servir como interlocutores ante cada una de las instituciones organizadoras.