INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

ESCOM

Trabajo Terminal

"Modelado del Physarum Polycephalum implementado en robot basado en autómatas celulares"

2017 - A056

Presenta

Marín Alavez Edgar Yair

Directores

Dr. Juárez Martínez Genaro

Dr. Oliva Moreno Luz Noé

Mayo 2018

INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADÉMICA

No. de TT: 2017 - A056 Mayo 2018

Documento Técnico

"Modelado del Physarum Polycephalum implementado en robot basado en autómatas celulares"

Presenta

Marín Alavez Edgar Yair [1]

Directores

Dr. Juárez Martínez Genaro **Dr. Oliva Moreno Luz Noé**

RESUMEN

En el presente reporte se presenta la documentación técnica del Trabajo Terminal 2017-A056 ” Modelado del Physarum Polycephalum implementado en robot basado en autómatas celulares ” el cual incluye información acerca del desarrollo del mismo con el fin de obtener el producto final.

[1] sebasyairv@gmail.com

Advertencia

" Este documento contiene información desarrollada por la Escuela Superior de Cómputo del Instituto Politécnico nacional, a partir de datos y documentos con derechos de propiedad y por lo tanto, su uso quedará restringido a las aplicaciones que explícitamente se convengan."

La aplicación no convenida exime a la escuela su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico puede obternerse en:

La Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional, situada en Av. Juan de Dios Bátiz s/n Teléfono: 57296000, extensión 52000.

Índice

1. Introducción **9**
1.1. Objetivo . . . 10
1.1.1. Objetivos Específicos . . . 10

2. Marco Teórico **11**
2.1. Slime Mould . . . 11
2.1.1. Características del Slime Mould . . . 12
2.1.2. Estructura del plasmodio . . . 14
2.1.3. Caraterística por su grupo protista . . . 14
2.1.4. Clasificación dada su estructura . . . 15
2.1.5. Interpretación computacional del hongo . . . 15
2.1.6. Aplicaciones del Physarum en la robótica y la computación . . . 16
2.1.7. Antecedentes del Physarum en la ciencia . . . 16
2.1.8. Aplicación en México . . . 17
2.1.9. ¿ Por qué considerarlo como un sistema complejo ? . . . 18
2.2. Autómatas Celulares . . . 19
2.2.1. Definición Formal . . . 19
2.2.2. Elementos de un AC . . . 21
2.2.3. Vecindad del AC . . . 21
2.3. El Juego de la Vida . . . 23
2.3.1. Patrones en la regla . . . 23
2.4. Expresión regular . . . 24
2.4.1. Operadores . . . 24

3. Análisis de Diseño **26**
3.1. Requerimientos . . . 26
3.1.1. Funcionales . . . 26
3.1.2. No funcionales . . . 26
3.1.3. Restricciones . . . 27
3.2. Diagramas . . . 27
3.2.1. Arquitectura . . . 28
3.2.2. Casos de Uso . . . 29
3.2.3. Clases . . . 31
3.2.4. Diagrama de Actividades . . . 32
3.3. Requerimientos de la herramienta . . . 33
3.3.1. Software . . . 33
3.3.2. Hardware . . . 33
3.4. Justificación de lenguajes a emplear . . . 36
3.4.1. C++ . . . 36

4. 2DCAvN(9) **38**
4.1. Vecindad a emplear . . . 39
4.2. Condiciones de transición . . . 40
4.3. Ejemplo de aplicación . . . 41

5. eBots **45**
5.1. Comportamiento del Robot . . . 46
5.2. Ajuste de avance . . . 48

6. Pruebas **51**
6.1. Requerimientos . . . 51
6.2. Entorno gráfico . . . 51
6.3. Prueba del simulador con Red de Tokyo . . . 52
6.3.1. Comparación de 6 simulaciones . . . 57
6.4. Prueba del simulador con laberinto . . . 58
6.4.1. Comparación de 6 simulaciones . . . 63
6.5. Prueba del simulador con ejemplo de Jeff . . . 64
6.6. Comparación con otros modelos y el organismo real . . . 69
6.6.1. Modelo de Adamatzky . . . 69
6.6.2. Modelo de Jeff Jones . . . 70
6.6.3. Modelo de Gunji . . . 72
6.6.4. Resumen de los modelos . . . 73
6.6.5. Comparación con el modelo real . . . 74
6.6.6. Observaciones de nuestro modelo . . . 76
6.7. Prueba piloto del robot . . . 76
6.7.1. Pruebas de movimiento básicas . . . 76
6.7.2. Prueba del sensor de proximidad . . . 78
6.7.3. Observaciones o problemáticas presentadas . . . 78
6.8. Evolución continua del Physarum . . . 79
6.9. Prueba con maqueta . . . 80

7. Conclusiones **86**

8. Trabajo a futuro **86**
8.1. Repositorio y derivado . . . 86

9. Anexo **88**
9.1. Glosario . . . 88

10. Apendíce A **88**
10.1. Análisis de Riesgos y Estimaciones . . . 88
10.1.1. Planteamiento del problema . . . 88
10.1.2. Estudio de Factibilidad . . . 89
10.1.3. Factibilidad Técnica . . . 89
10.1.4. Factibilidad Económica . . . 89
10.1.5. Factibilidad Operativa . . . 89
10.2. Estimación de Métricas . . . 89
10.2.1. Métrica Orientada a la Función . . . 89
10.3. COCOMO . . . 93
10.4. Análisis de Riesgos . . . 97
10.4.1. Tabla general de riesgos . . . 98

10.5. Semáforo de Riesgos 99

11. Apendíce B **100**
11.1. Código del programa 100
11.1.1. Main 100
11.1.2. MainWindow 100
11.1.3. Rectangulo 123
11.1.4. Hormiga 127
11.2. Código del robot 130

Índice de figuras

1. Physarum Polycephalum real 11
2. Ciclo de vida del Physarum 12
3. Estructurura del Physarum Polycephalum 14
4. Ciclo de vida del Physarum 15
5. Experimento de Tokio 16
6. Resolución del laberinto por el Physarum 17
7. Estados seleccionados para emplear nutriente 17
8. Ejemplo del un autómata celular con el juego de la vida 19
9. Vecindad de von Neumann 22
10. Vecindad de Moore 23
11. Ejemplos de patrones del Juego de la vida 24
12. Arquitectura General. 28
13. Diagrama de Casos de Uso General. 29
14. Diagrama de Clases. 31
15. Diagrama de Actividades. 32
16. Microcontrolador MSP430G2 33
17. Módulo HC-05 34
18. Pilas AA marca Duracell 34
19. Llantas DC con motor. 35
20. Sensor HC-SR04. 35
21. Fuente de regulación. 35
22. Driver de los motores. 36
23. Comparación de tiempo de ejecución Java vs C+ 37
24. Ejemplo del caso ideal. 38
25. Ejemplo de la regla, T = 0. 41
26. Ejemplo de las regla, T = 1. 41
27. Ejemplo de las reglas, T = 2. 41
28. Ejemplo de las reglas, T = 3. 42
29. Ejemplo de las reglas, T = 10. 42
30. Ejemplo de las reglas, T = 11. 42
31. Ejemplo de las reglas, T = 12. 43
32. Ejemplo de la regla 5. 43
33. Ejemplo de las reglas T = 39. 44

34. Ejemplo de lasreglas T = 49. . . . 44
35. Ejemplo de lasreglas T = 51. . . . 44
36. Ejemplo de lasreglas T = 69. . . . 45
37. eBot . . . 45
38. Diagrama del robot aproximado . . . 46
39. Diagrama de flujo del robot. . . . 47
40. Ejemplo de cambio de dimensiones. . . . 48
41. Ejemplo de división para el simulador. . . . 49
42. Dibujo del simulador. . . . 49
43. Pantalla del simulador . . . 51
44. Tokyo en T = 0 . . . 52
45. Tokyo en T = 40 . . . 53
46. Tokyo en T = 75 . . . 53
47. Tokyo en T = 110 . . . 54
48. Tokyo en T = 160 . . . 54
49. Tokyo en T = 200 . . . 55
50. Tokyo en T = 225 . . . 55
51. Tokyo en T = 260 . . . 56
52. Tokyo en T = 330 . . . 56
53. Simulación de la red de Tokyo a 6 salidas. . . . 57
54. Laberinto en T = 0 . . . 58
55. Laberinto en T = 106 . . . 58
56. Laberinto en T = 156 . . . 59
57. Laberinto en T = 252 . . . 59
58. Laberinto en T = 345 . . . 60
59. Laberinto en T = 421 . . . 60
60. Laberinto en T = 503 . . . 61
61. Laberinto en T = 586 . . . 61
62. Laberinto en T = 655 . . . 62
63. Simulación del laberinto a 6 salidas . . . 63
64. Jeff mapa en T = 0 . . . 64
65. Jeff mapa en T = 82 . . . 64
66. Jeff mapa en T = 170 . . . 65
67. Jeff mapa en T = 270 . . . 65
68. Jeff mapa en T = 371 . . . 66
69. Jeff mapa en T = 470 . . . 66
70. Jeff mapa en T = 571 . . . 67
71. Jeff mapa en T = 720 . . . 67
72. Simulación con un ejemplo de Jeff a 6 salidas. . . . 68
73. Representación del agente de acuerdo al modelo de Jeff . . . 71
74. Ejemplo de aplicación del modelo de Gunji . . . 72
75. Simulación de la red de Tokyo con exploración continua del Physarum . . . 80
76. Configuración inicial . . . 81
77. Configuración final del AC . . . 81
78. Trayectoria del robot. . . . 82

79. Punto de partida 82
80. Robot a 20 segundos 83
81. Robot a 50 segundos 83
82. Robot a 75 segundos 84
83. Robot a 90 segundos 84
84. Robot a 100 segundos 85
85. Robot a 130 segundos 85
86. Estimación de LDC por punto de función en diferentes lenguajes 93
87. Constantes de modelos en proyectos de software 94

1. Introducción

Durante el paso de los años junto con el desarrollo de la tecnología ha sido de interés la implementación de diversos modelos en relación al comportamiento o procesos con base a las necesidades de sus desarrolladores. Los autómatas celulares se han convertido en modelos computacionales sobre agentes, fuerza-social, interacción de individuos y auto organización, describiendo sistemas complejos.

Es por ello que como objeto de interés es la implementación de dicho modelo junto con el desarrollo de un robot, además que los autómataas celulares han tenido un mayor enfoque sobre el software, siendo el presente Trabajo Terminal una opción mas en esta área. Con respecto al Physarum Polycephalum el propio organismo se ha expuesto a la resolución de problemas sobre la solucón de laberintos, compuertas lógicas, fenómenos de evacuación o redes de transporte, donde en este caso se empleará sobre búsqueda de salidas. Dicho estudio se ha llevado a cabo por diversos autores entre los más conocidos Andrew Adamatzky y Jeff Jones que justifican al Slime Mould como un organismo candidato a ser computable debido a que se constituye de ser simple y autónomo que puede tener el control de sus propiedades, Adamatzky en la computación no clásica menciona a que es simple de cultivar, es configurable y sujeto a influencia externa.

El trabajo terminal propone la implementación del Physarum Polycephalum con el fin de lograr un modelo computacional de propagacón por medio del uso de un autómata celular. En motivación por el comportamiento de organismos y necesidad de describir este sistema complejo siendo consecuencia de una interacción de individuos basados en reglas simples de naturaleza paralela, que junto con diversos lenguajes de programación se pueden emplear en sistemas de cómputo en conjunto con dispositivos electrónicos al ser una buena opción en el desarrollo de sistemas complejos de bajo costo.

1.1. Objetivo

Desarrollar un modelo basado en autómatas celulares que simule el comportamiendo del organismos Physarum Polycephalum en robots de bajo costo para la solución de ruteo.

1.1.1. Objetivos Específicos

- Realizar el estudio del Slime Mould a fin de obtener información relevante del organismo.
- Obtener las funciones de transición del autómata celular que se adecúen al comportamiento del organismo.
- Implementar el autómata celular de acuerdo a los criterios necesarios para emplearse junto con el robot.

2. Marco Teórico

2.1. Slime Mould

El Physarum Polycephalum o Slime Mould **moho de fango** parte de la familia Physaridae, organismos con la característica de tener una estructura viscosa que son estimulados por el entorno en el que se encuentran, estos estímulos pueden ser iluminación, temperatura o niveles de pH. El organimo es visto como una célula gigante que el ojo humano puede observar con facilidad, un ejemplo del organismo lo observamos en la figura 1, como un organismo de estructura amarilla y viscosa.

El organismo desarrolla una estructura de ramificaciones que se le denomina **plasmodio**, esto ocurre a consecuencia de la exploración por necesidad de la búsqueda de alimento a lo largo del entorno en el que se encuentra. La ingesta de los nutrientes tiene un proceso llamado fagocitosis que consiste en rodear el recurso e introducirlo en el cuerpo celular, conforme va explorando el organismo, empieza a crear una red con el propósito de conectar todos los puntos o nutrientes que haya encontrado en su exploración discriminando zonas que no vaya a emplear obteniendo una red parecida a las venas que conectan dichos puntos.

Figura 1: Physarum Polycephalum real [10].

Una parte importante del Physarum es el ciclo de vida que desarrolla, inicialmente parte como una célula llamada cigoto, este cigoto se le puede considerar como un punto inicial siendo una concentración de muchas células que poco a poco iran explorando con el fin de hacer crecer el plasmodio, una vez creada la red final y agotar los recursos, puede ocurrir que siga explorando o bien entrar a una fase de comportamiento fructífero, con el que forma pequeñas plantas llamadas esporangios que una vez al madurar, explotan esparciendo esporas en el entorno, creando nuevas células que después de un proceso de reproducción forman nuevamente un cigoto, este ciclo se ilustra en la figura 4.

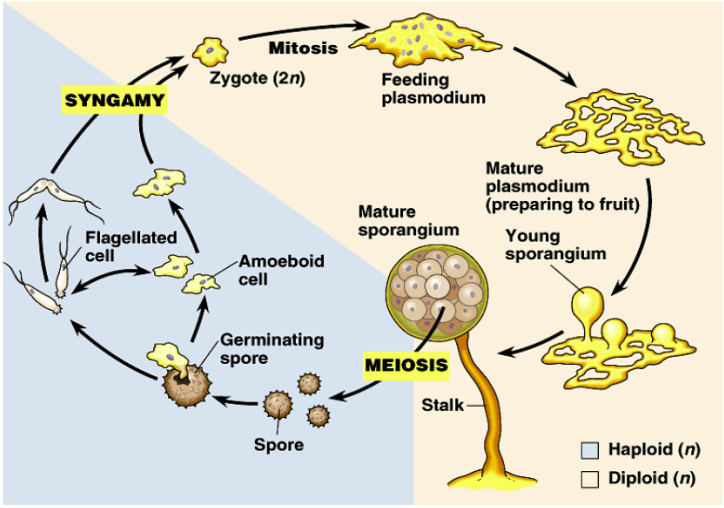


Figura 2: Ciclo de vida del Physarum. Inicialmente a partir de las esporas formadas, crea un cigoto que será el punto inicial para empezar su exploración, al final empieza a crear nuevas esporas para repetir el proceso [8].

Las condiciones que emplea el organismo para poder nacer y crecer son generalmente sobre zonas húmedas con poca luz como lo son las áreas tropicales o bosques templados en especial sobre la corteza en descomposición, también se llega a dar muy rara vez en el propio suelo o césped. En la ecología tienen la función de ser descomponedores o recicladores de nutrientes, además sobre la cadena alimentaria al consumir microorganismos permiten transportar energía a la red trófica siendo consumidos por insectos, por mencionar ejemplos, moscas y escarabajos. Además llegan a ser importantes para la vegetación ya que su estructura contiene celulosa.

2.1.1. Características del Slime Mould

Enunciaremos algunas de las caraterísticas mas relevantes del organismo, especialmente sobre las condiciones que emplea para poder sobrevivir.

- Se dice que el organismo tiene la capacidad de saber donde ha estado, por ello se le ha nombrado como un organismo inteligente, esto es gracias a que deja pequeños rastros en zonas que haya explorado, haciendole saber que dicha zona ya anteriormente alguna parte del plasmodio estuvo presente.
- Puede llegar a crecer hasta más de un metro cuadrado.
- La luz es uno de sus principales repelentes del organismo, otros ejemplos pueden ser sustancias como café o sal aunque se han echo experimentos donde se ha puesto a prueba al organismo ante estas sustancias, en un principio el Physarum mostraba un comportamiento de rechazo sin embargo al paso del tiempo, el organimo logró comprender que las sustancias no le eran nocivas.
- Compuesto de una fase vegetativa "Plasmodio", que se caracteriza de poder expandirse o deslizarse explorando todo su medio.

- Crece bajo temperaturas de 14 a 27 grados Celsius.
- Su velocidad de exploración es aproximadamente 1 cm por hora.
- Una vez hallado un nutriente, el Physarum se encarga de que parte del alimento llegue a toda la red.
- Tiene la capacidad de reorganizar su red a medida que busca alimento.
- Niveles de pH: 6.5 - 7.3 para el crecimiento del organismo.

Para resumir, se exponen las caraterísticas generales del Physarum en la siguiente tabla 1.

Tabla 1: Caraterísticas generales del Physarum Polycephalum.

Caraterística	**Descripción**
Estructura	Viscosa, amarillenta, gelatinosa
Hábitat	Zonas húmedas con poca luz.
Alimentación	Esporas, bacterias, microbios, compuestos orgánicos y materia orgánica muerta
Proceso de alimentación	fagocitosis
Velocidad de exploración	1 $\frac{cm}{h}$
Temperatura ideal	14 a 27 grados celsius
pH ideal	6.5 - 7.3

Nutrición.

Como se mencionó anteriormente el proceso de alimentación lo lleva a cabo por fagocitosis entre los alimentos comunes son hongos microscópicos, esporas de plantas, protozoarios, fragmentos de materia orgánica muerta. Ante la absorción o ingesta de dichos alimentos no se sabe completamente como realizan este proceso sin embargo se piensa que una vez dentro del plasmodio se engloban en vacuolas digestivas que después serán absorbidas por la acción de enzimas para eliminar los desechos por deposición.

Esclerotización.

Esta propiedad es la respuesta a condiciones que le afecten al Physarum, provoca que entre a un estado de masa dura compuesta de pequeñas esferas con un número variable de núcleos.Esto es un método de supervivencia temporal y si las condiciones le favorecen, vuelva nuevamente a su estado de plasmodio.

Esporulación.

En el ciclo de vida se mencionó que existe una etapa donde el plasmodio entra a una fase de producir esporangios que se convierte en su función reproductiva, esto se convierte en un esporangio maduro que al explotar, el aire se convierte en su medio de transporte, aunque se llega el caso en que insectos lo emplean como alimento, aunque no esta comprobado que dentro de la enzina digestiva de los insectos puedan germinar o afectar.

2.1.2. Estructura del plasmodio

El organismo como tal llega a ser una red de tubos, constituido por elementos importantes una membrana celular, una cápsula externa viscosa, un ectoplasma exterior (Gel) y un núcleo endoplásmico en su interior veasé en la Figura 3.

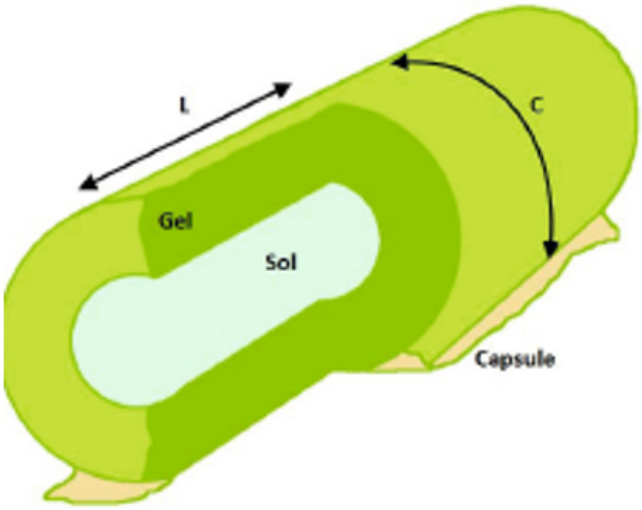


Figura 3: Estructurura del Physarum Polycephalum [14].

En conjunto llegan a formar filamentos o redes parecido a una vena humana, como resultado de haber explorado su medio con una capa de protoplasma. Dentro de los tubos existen ciertas estimulaciones que fluyen en una dirección, que llegan a cambiar de dirección, dándole la oportunidad de crecer mientras consume nutrientes.

2.1.3. Caraterística por su grupo protista

El Physarum está considerado dentro del grupo Mixomycota [12], conocidos como hongos mucilaginosos acelulares, que son organismos que carecen de una pared celular, la forma en que se delimitan es por una membrana celular junto con una envoltura gelatinosa que son microfibrillas, interviniendo en el movimiento del plasmodio, su clasificación se encuentra dada por su origen:

- **Plasmodio típico:** Nacen a partir de una división celular donde inicialmente se encuentran uninucleadas (solo un núcleo), que aumenta de volumen por la asimilación de sustancias.
- **Plasmodio de fusión:** Se da por la reunión de células ameboides y la fusión de un plasmodio con otros.
- **Seudoplasmodio:** Se forma por la agregaciíon de mixamebas que son amebas con capacidad de moverse.

2.1.4. Clasificación dada su estructura

Una clasificación que se le da al Physarum dentro de los grupos de los hongos es dependiendo la estructura, el Physarum se encuentra dentro de un grupo denominado **Faneroplasmodio**, donde se le da la característca de tener una red visible , presentando colores llamativos y creando formas o ramificaciones.

Esta clasificación se encuentra junto al **Afanoplasmodio**, donde la red que conforma es de menor grado que la del Physarum y no es tan fácil visualizarlo, por último tenemos el **Protoplasmodio**, el cual tiene un tamaño muy diminuto y no forman ramificaciones.

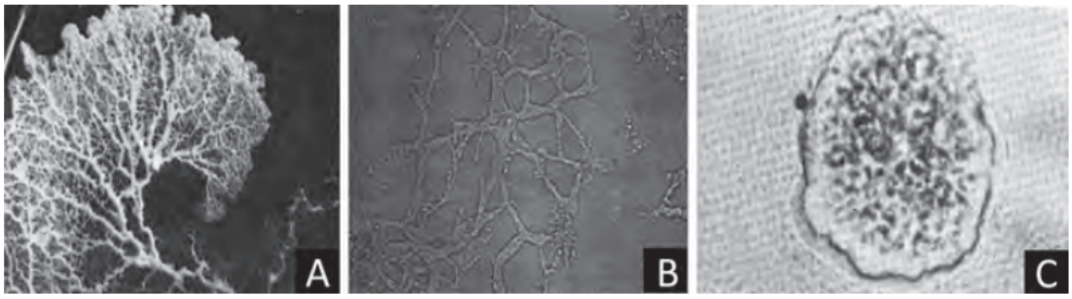


Figura 4: Plasmodios por su estructura. a) Faneroplasmodio, b) Afanoplasmodio, c) Protoplasmodio [13].

2.1.5. Interpretación computacional del hongo

El physarum puede ser visto como un organismo de comportamiento complejo de patrones dinámicos tal que contiene 2 características esenciales de supervivencia: búsqueda o exploración de su medio en un área máxima y la eficiencia en el transporte de los nutrientes o dibujo de caminos. Se puede clasificar en una computación no convencional al aprovechar el comportamiento del organismo con el fin de obtener funciones de cómputo.

La computación no convencional tiene diferentes motivaciones a partir del uso de dichos comportamientos físicos, como propiedades que en sistemas informáticos llegan a ser redundantes (no dependientes), además de mostrar comportamientos emergentes (a partir de pequeñas estructuras, se generan comportamientos complejos como resultado de interacciones). Sin embargo, tambíen tiene ciertas dificultades al caer en que los sistemas tienden a no detenerse al momento de resolver un problema pero pueden adoptar un equilibrio o un estado estable, aquí es importante la interpretación del comportamiento que va generando debido a que como tal lo que esté procesando el organismo, para él no tienen alguna interpretación relevante que a diferencia de nosotros como observadores empleamos ese comportamiento para ser investigado u obtener alguna información.

2.1.6. Aplicaciones del Physarum en la robótica y la computación

El organismo ha sido fuente de inspiración para construir robots que asemejen su comportamiento, como Umedachi [2] que empleando la robótica ameboidea junto con osciladores externos acoplados a un interior lleno de líquido imitando la contracción del gel del plasmodio. El Physarum llegan a ser una máquina biológica caracterizado de movimiento, adaptación y fácil control de sus propiedades con base de material simple.

Ante la computación se ha visto que el hongo ha tenido importancia en aspectos como fenómenos oscilatorios, formación de patrones o adaptación, explicando memorias primitivas de acuerdo a su comportamiento de poder contraerse y saber en dónde ya ha estado, donde un ejemplo ha sido en aproximar caminos en resolución de laberintos.

2.1.7. Antecedentes del Physarum en la ciencia

- Por el año 2000, investigadores de Japón emplearon el hongo con el fin de trazar rutas de Tokio donde cada uno de los puntos importantes colocaban algún nutriente para que el hongo pudiera llegar a él y trazar su respectiva ruta. El hongo al final logró trazar rutas que incluso se vieron optimizadas con respecto a la red en aquellos tiempos veasé Figura 5.

- Otro experimento fue llevado a cabo empleando diferentes sustancias con el fin de ver si el Physarum podía recordar el nivel de peligro que tenía ante ellas. Junto con un puente, el Physarum fue obstaculizado con cafeína, pocos días después, se volvió a realizar el experimento con el mismo organismo, donde no mostraba dificultad de cruzar al saber que la cafeína no le era nocivo.

- Resolución de laberintos, este experimento es uno de los más famosos en la aplicación del Physarum, ya que como tal lo que hace el hongo es extenderse en toda su área para al final trazar un camino para conectar un punto inicial y final como lo muestra la Figura 6.

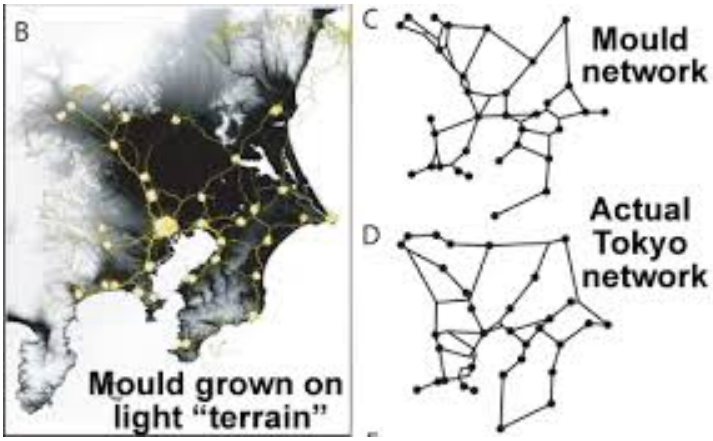
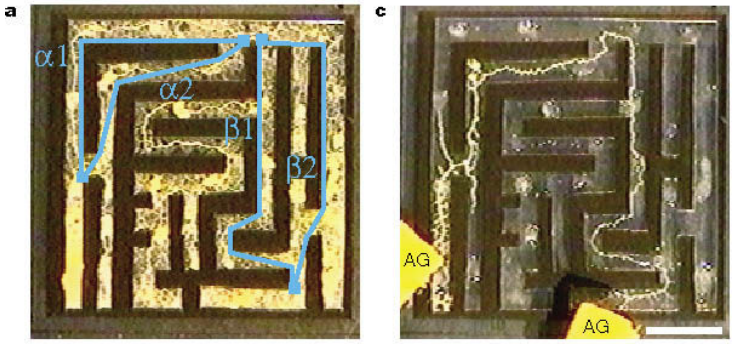


Figura 5: Experimento de Tokio [23].

[2] Umedachi Takuya. (2010). Fully decentralized control of a soft-bodied robot inspired by true slime mold



	None	β1	β2	β1, β2
None	2	0	0	0
α1	0	0	0	0
α2	0	5	6	3
α1, α2	0	0	0	3

Figura 6: Resolución del laberinto por el Physarum. [19].

2.1.8. Aplicación en México

Para este experimento el objetivo principal es unir carreteras de la República Mexicana donde fueron participes integrantes de la University of the West of England, la UNAM, Cinvestav, entre otros [4], empleando una mapa de 120 x 120 mm se seleccionaron 19 puntos para poner nutrientes en ciertos estados tal como se observa en la Figura 7.

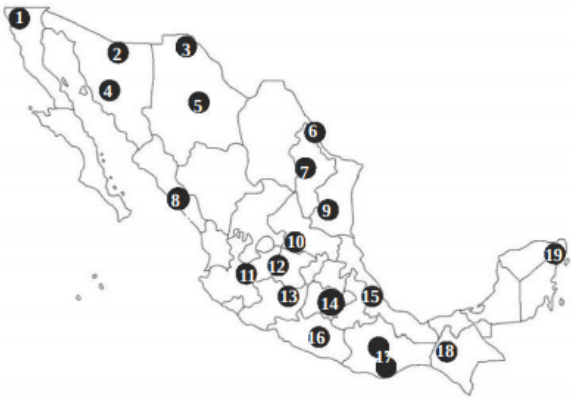


Figura 7: Estados seleccionados para emplear nutriente [5].

En el experimento se empleó una temperatura de 22 - 25 grados centígrados, para las 12 primeras horas el Physarum empezó a propagarse por las regiones de Huatulco, Oaxaca y Tuxtla Gutiérrez. A las 24 horas Monterrey y Nuevo Laredo. A las 38 horas el Physarum ya habia cubierto casi gran parte de la República Mexicana para que después de 52 horas estuviera

colonizada a excepción de Baja California y Xalapa. Completamente el Physarum tardó cerca de 61 horas en realizar la traza de rutas de todo el mapeo de México.

2.1.9. ¿ Por qué considerarlo como un sistema complejo ?

Un sistema complejo se comporta como un grupo donde la interacción de estos elementos considerados primitivos conducen a comportamientos de gran escala que no son fáciles de predecir conocido como un sistema emergente. Para el Physarum se le considera complejo ya que las siguientes características se encuentran en él:

Comportamiento colectivo.

Se le considera como la conducta que producen masas o grupos, en donde el Physarum es consecuencia de un grupo de células o ameboides en una sola gran célula que conforme va creciendo el trabajo colectivo de los elementos que se encuentran dentro del plasmodio, por lo cual cada elemento reacciona dependiendo a los estímulos de su entorno y como tal no existe una célula jefe que ordene a las demás.

Autoorganización.

Esta propiedad le da la característica de tener un comportamiento emergente dado el orden de sistemas fśicos y naturales, en otras palabras dependiendo de las condiciones que vayan apareciendo el sistema pareciera que toma decisiones con el fin de obtener ventajas o disminuir aspectos negativos. En el caso del Physarum observamos que el organimos entra a una etapa que si llega a tener condiciones desfavorables puede mantenerse vivo, discriminar zonas que no le sirvan para su red o el transportar alimento a todo el plasmodio.

Elementos primitivos.

A pesar de que en realidad el plasmodio en su estructura biológica es compleja, como tal las células que lo componen se les considera simples ya que se pueden ver como simples agentes que trabajan en conjunto y la función que tienen es meramente el sobrevivir, se les considera inteligentes ya que el comportamiento que van originando tiene la capacidad de aprender o saber en donde ha estado.

2.2. Autómatas Celulares

Un autómata celular (**AC**) es un sistema dinámico discreto que empleando una regla de evolución bajo variables dentro de una rejilla donde las variables se llaman células cambian de estado o valor según el el criterio de células vecinas. Parten de una configuración inicial generando comportamientos complejos, donde los elementos que conforman dicha configuración están distribuidos en su espacio teniendo la idea de que entre más cerca esté una región con ciertas carecterísticas, esta se verá influencia mucho más que otras aisladas.

El aspecto de los autómatas celulares es el tener la capacidad de lograr comportamientos o propiedades a partir de la dinámica global, esto hace que un AC se deba de analizar a partir de las evoluciones que va mostrando, partiendo de una configuración inicial cambiando sus estados o configuración del sistema de manera síncrona, la ventaja de estos modelos es el obtener dinámicas complejas por elementos o reglas simples. (Figura 8)

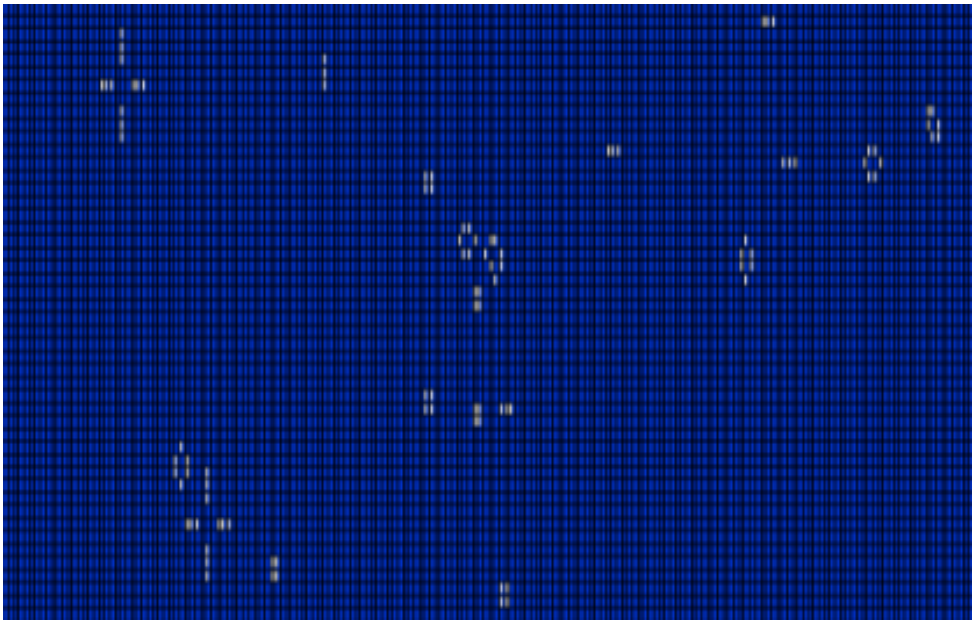


Figura 8: Ejemplo del un autómata celular con el juego de la vida.

2.2.1. Definición Formal

Formalmente un autómata celular lo definimos a partir de lo siguiente. Sea d un entero positivo se define como una **dimensión** de Z^d que representa las células en el espacio o en otras palabras es el arreglo multimensional uniforme. Sea S un **conjunto de estados** que cada célula dentro de Z^d puede adquirir, esa asignación de estados para cada célula en el sistema con dimension d, y conjunto de estados S se convierte en una función.

$$c : Z^d \to S \tag{1}$$

La configuración a palabras simples se debe entender como una descripción instantánea de todas las células dentro de la rejilla. Ahora bien, cada célula tiene una posición relativa representado por un vector $\overrightarrow{n}$, que retomando la configuración, decimos que el estado de la célula la

denotamos como $c(\overrightarrow{n})$.

Teniendo $\overrightarrow{n}$, podemos definir la **vecindad**, o vector de vecindad de espacio m, con la siguiente tupla.

$$N = (\overrightarrow{n_1}, \overrightarrow{n_2}, ..., \overrightarrow{n_m}) \tag{2}$$

Donde para cada célula $\overrightarrow{n} \in Z^d$, con m vecinos $\overrightarrow{n} + \overrightarrow{n}$ para $i = 1, 2, ..., m$. Entonces para alguna célula $\overrightarrow{n} \in Z^d$ denotamos.

$$N(\overrightarrow{n}) = \{\overrightarrow{n} + \overrightarrow{n}_1, \overrightarrow{n} + \overrightarrow{n}_2, ..., \overrightarrow{n} + \overrightarrow{n}_m\} \tag{3}$$

La vecindad que más adelante se va a ilustrar se debe de ver como una región de influencia, donde cada célula $\overrightarrow{n} \in Z^d$ es afectada por sus vecinos adyacentes para provocar cambios en su estado $c(\overrightarrow{n})$.

Por último tenemos la **regla de transición local**, que se emplea para definir el nuevo estado de la célula con base a los estados actuales de sus vecinos,

$$f : S^m \to S \tag{4}$$

es decir que para los vecinos de una célula con estados $s^1, s^2, ..., s^m$ el nuevo estado de la célula de referencia será $f(s^1, s^2, ...s^m)$.

En un AC la misma regla se aplica para cada una de las células en el espacio de manera simultanea, esto causa un cambio global en la configuración c que al aplicar las reglas se convierte en $c^{'}$, donde para todo $\overrightarrow{n} \in Z^d$.

$$c^{'}(\overrightarrow{n}) = f[c(\overrightarrow{n} + \overrightarrow{n}_1), c(\overrightarrow{n} + \overrightarrow{n}_2), ..., c(\overrightarrow{n} + \overrightarrow{n}_m))] \tag{5}$$

Donde simplemente estamos definiendo dicha transición para las células en el espacio. Por lo tanto la transformación $c \mapsto c^{'}$ es la función global del sistema denotandolo como.

$$G : S^{Z^d} \to S^{Z^d} \tag{6}$$

Que dicho de otra manera es que cualquier configuración c es transformada en otra $c^{'}$, donde este último es considerado como $c^{'} = G(c)$, que aplica la función de transición local a cada celda del arreglo.

Para terminar sea la configuración inicial c^0, representamos la evolución o generaciones de un AC como:

$$c^0 \to c^1 \to ...c^t \to c^{t+1} \to ...c^{t+p} = x^t \tag{7}$$

Donde existe un ciclo con período p > 0, que decimos que el autómata llega a una configuración fija o sin cambio si p = 1.

Entonces, un autómata celular se define de acuerdo a la Ecuación 8, donde:

$$A = (d, S, N, f) \tag{8}$$

- d es la dimensión tal que $d \in Z^+$
- S es el conjunto finito de estados.
- N como el vector de vecindad $N = \{\overrightarrow{n_1}, \overrightarrow{n_2}, ..., \overrightarrow{n_m}\}$
- f es la función de transición local $f : S^m \rightarrow S$

2.2.2. Elementos de un AC

1. **Espacio celular**: Los autómatas celulares tienen un arreglo de células en una dimensión d. Cada elemento o celda dentro del espacio es una célula que puede tomar algun valor en S.
2. **Estados**: Es el valor que cada celular tiene en cada generación o puede llegar a obtener representando alguna propiedad, estado físico o característica en el sistema.
3. **Configuración de vecindad**: Cada célula es afectada por el entorno que las rodea, donde dicha célula es denominada " célula central ", existe una función de transición que dado el radio de vecindad determinará su siguiente estado.
4. **Función de transición**: Cada célula toma un valor con base a la configuración de la vecindad, cayendo en un estado.

2.2.3. Vecindad del AC

Cada elemento dentro del sistema es necesario evaluarlo, la vecindad se comporta como un región donde agentes o elementos a evaluar se consideran como vecinos. Esto en términos generales, se trata de asociar objetos dadas ciertas coordenadas sea una distancia o radio con respecto a un punto de referencia llamado " Célula Central o de Referencia ", en los autómatas celulares podemos encontrar diferentes tipos de vecindad, por mencionar algunos se tiene los siguientes:

Vecindad de von Neumann [3]

Esta vecindad está definida por cuatro celdas que rodean en forma de cruz u ortogonal la célula central, empleado como criterio de autómatas bidimensionales, puede tener extensiones o incrementar su radio, sin embargo el más empleado es el que se puede observar en la Figura 9.

[3] John von Neumann: Matemático hungaro-estadounidense (1903-1957)

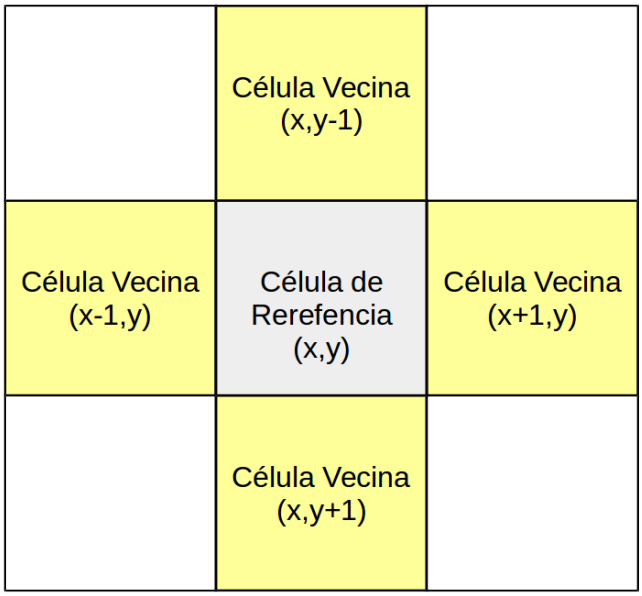


Figura 9: Vecindad de von Neumann.

Vecindad de Moore [4]

A diferencia de von Neumann, esta vecindad está totalmente rodeada de las ocho celdas observado en la Figura 10, tiene un gran impacto al haberse empleado para desarrollar el autómata celular del ” Juego de la vida de Conway ”.

[4]Edward Forrest Moore: Profesor estadounidense de matemáticas y ciencias de la computación (1925-2003)

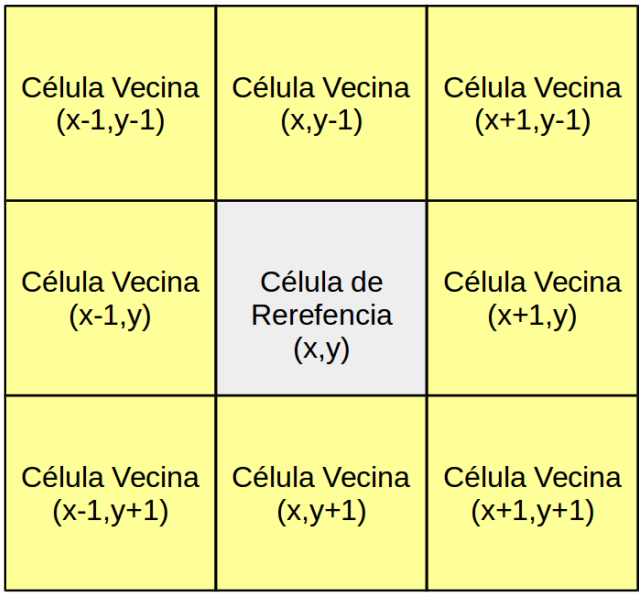


Figura 10: Vecindad de Moore.

2.3. El Juego de la Vida

Los autómatas celulares pueden existir en un espacio de cierta dimensión, donde los casos comunes son de 1 ó 2 dimensiones, el Juego de la Vida desarrollado por Conway [5], muestra el ejemplo de un AC de 2 dimensiones mostrado como un tablero de ajedrez empleando la vecindad de Moore definiendo las siguientes reglas.

1. **Nacimiento**: Sea una célula muerta, esta dará nacimiento a una nueva célula, si y solo si, existen 3 células activas en su vecindad.

2. **Muerte**: Sea una célula activa o vida, esta va a morir por aislamiento si existen menos de 2 células activas en su vecindad ó por sobrepoblación si existen más de 3 células vivas.

3. **Supervivencia**: Sea una célula activa, esta va a sobrevivir si y solo si existen 2 ó 3 células activas en su vecindad.

2.3.1. Patrones en la regla

Los patrones son aspectos que dicho modelo bajo un comportamiento complejo se han visto generados y bautizados de acuerdo a comportamientos que se observan en función de sus dinámicas o movimientos por mencionar algunos, blinkers u osciladores, gliders, vida estática generados por evaluaciones dado un tiempo n, más ejemplos pueden observarse en la Figura 11.

[5]John Horton Conway: Matemático británico enfocado en teoría de conjuntos, nudos, números y códigos (1937-Actualidad)

Así uno de ellos como los gliders siendo partículas que pueden moverse en el espacio moviéndose en líneas aproximadamente de 45 grados, esta serie de patrones son generados a partir de un llamdo glider-gun que para cierto número de evoluciones lanza un glider.

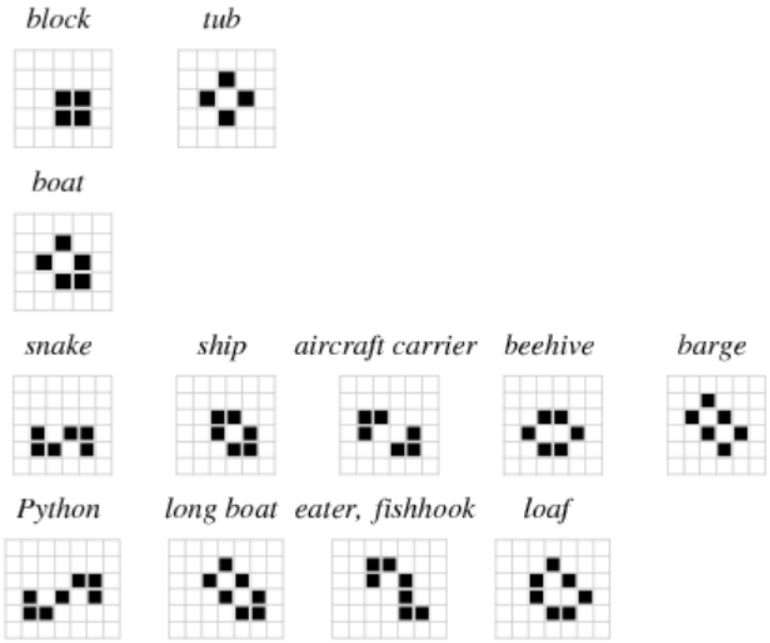


Figura 11: Ejemplos de patrones del Juego de la vida [9].

2.4. Expresión regular

Tienen el objetivo de representar cadenas definidas dentro de un alfabeto Σ, con base a lenguajes primitivos y operadores de composición. Los **lenguajes primitivos** comprenden los símbolos que conforman el alfabeto, el lenguaje vació y el lenguaje formado por la palabra vacía descritos en.

- ϕ
- λ
- $a \in \Sigma$

Su empleo es describir un patrón de texto de manera sencilla que en muchos de los casos se somete a evaluarse dentro de una validador de cadenas, para saber si dicha palabra o cadena pertenece a una cierta expresión regular.

2.4.1. Operadores

Sea α y β expresiones regulares tenemos los siguientes operadores, para ser expresiones derivadas

- $\alpha + \beta$: Unión.
- $\alpha.\beta$ ó $\alpha\beta$: Concatenación.
- α^* : Cierre o cerradura de Kleene.
- (α)

Donde estos operadores tienen una prioridad o procedencia.

1. ()
2. Cerradura
3. Concateniación
4. Unión

Ejemplo.

Sea $M = \{11, 00\}$ y $N = \{0, 1\}$ dos lenguajes tenemos lo siguiente.

- Por unión: $M + N = \{0, 1, 00, 11\}$
- Por concatenación: $MN = \{110, 111, 000, 001\}$
- Por cerradura de Kleene: Si L es un lenguaje la cerradura se denota L^* donde:
  Si $L^0 = \{\epsilon\}$
  Si $L^1 = \{L\} = \{0, 11\}$
  Si $L^2 = \{LL\} = \{00, 011, 110, 1111\}$

Sea $L(0 + 10^*)$ las cadenas que genera la expresión regular son $L = \{0, 1, 100, 1000, ...\}$.

3. Análisis de Diseño

3.1. Requerimientos

3.1.1. Funcionales

- **RF1**: El sistema deberá proveer una GUI donde el usuario pueda interactuar ante la formación de las diferentes configuraciones del AC.
- **RF2**: Deberá manejar entrada de datos archivo o manualmente, donde este último se le darán las opciones o estados.
- **RF3**: La interfaz deberá mostrar datos importantes tales como generación, número de células en determinado estado o algún otro dato importante.
- **RF4**: El sistema debe permitir al usuario generar la configuración inicial del AC, donde el robot solo va a esperar las acciones sin necesidad que el usuario realice alguna configuración previa a este mismo, solamente lo posiciones en donde vaya a ser usado.
- **RF5**: El sistema proveerá de una gráfica de densidad donde mostrará la expasión de Physarum conforme avancen sus generaciones.
- **RF6**: El sistema deberá permitir al usuario dar una corrida paso a paso, automática, pausa y continuar. Así como opciones de limpiar o algún otro que sea útil.
- **RF7**: Al momento de la entrada de archivos este tiene que validar que el archivo sea adecuado para procesar.
- **RF8**: Además para cierta configuración inicial o final, se debe de proveer al usuario poder guardar dicha configuración en archivo.

3.1.2. No funcionales

- **RFN1**: El sistema debe de soportar un cierto número máximo de celdas pensadas en función del tamaño de la resolución de pantalla con el fin que sea apreciada la evolución.
- **RFN2**: Para el manejo de algunos métodos, especialmente de tiempo se deben usar hilos.
- **RFN3**: El tiempo de aprendizaje del sistema por el usuario debe ser lo más rápido posible donde por ende no debe de complicarse su uso.
- **RFN4**: Debe de contar con algunos módulos o explicaciones para su fácil manejo así como un manual de preferencia, mensajes informativos o de errores.
- **RFN5**: Debe existir integridad de la información para el manejo adecuado del AC y que no tengan algún problema sobre su evolución.

3.1.3. Restricciones

- Para el presente modelado, existen ciertos aspectos del Physarum que al ser propiedades físicas son difícilmente modelables en aspecto hardware por aspectos de costos, recursos o factibilidad.
- Para fines de experimento, la zona donde va a trabajar el robot debe ser compacta y lo más pequeña posible, en función de esto el ejemplo que se provará para experimentación tendrá dimensiones definidas.

3.2. Diagramas

Los siguientes diagramas muestran un panorama general de lo que se pretende desarrollar, principalmente el usuario podrá establecer configuraciones iniciales con el fin de manipularlas si lo deseam además de que el robot reciba las señales mostrado en la Figura 12 con base a diferentes entradas.

El diseño del sistema dado el diagrama de clases de la Figura 14 estará consitituido de que una clase Célula teniendo características de cada elemento que compondrá el Lattice , siendo otra clase que determina el tamaño o espacio de evoluciones, teniendo la configuración inicial y células en un espacio.

3.2.1. Arquitectura

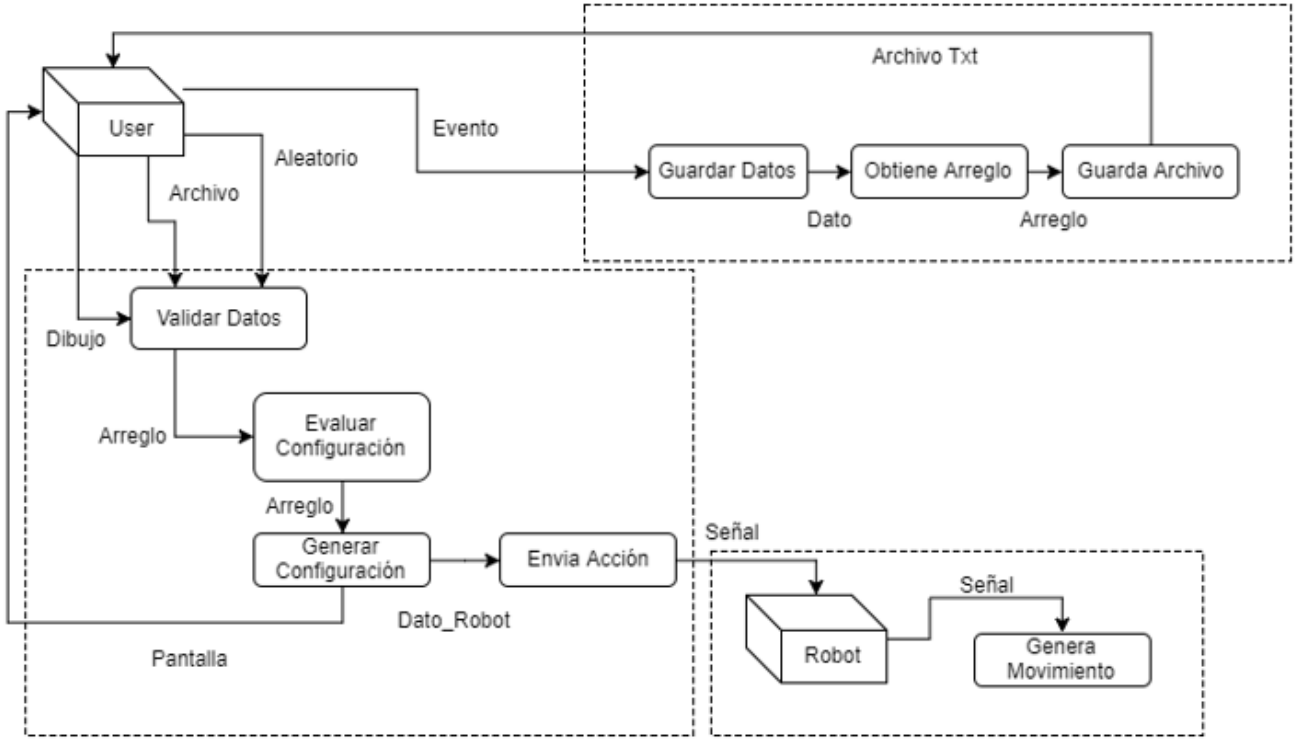


Figura 12: Arquitectura General.

3.2.2. Casos de Uso

Muestra los principales casos de uso sobre el usuario siendo fundamentales para la relación sistema-usuario.

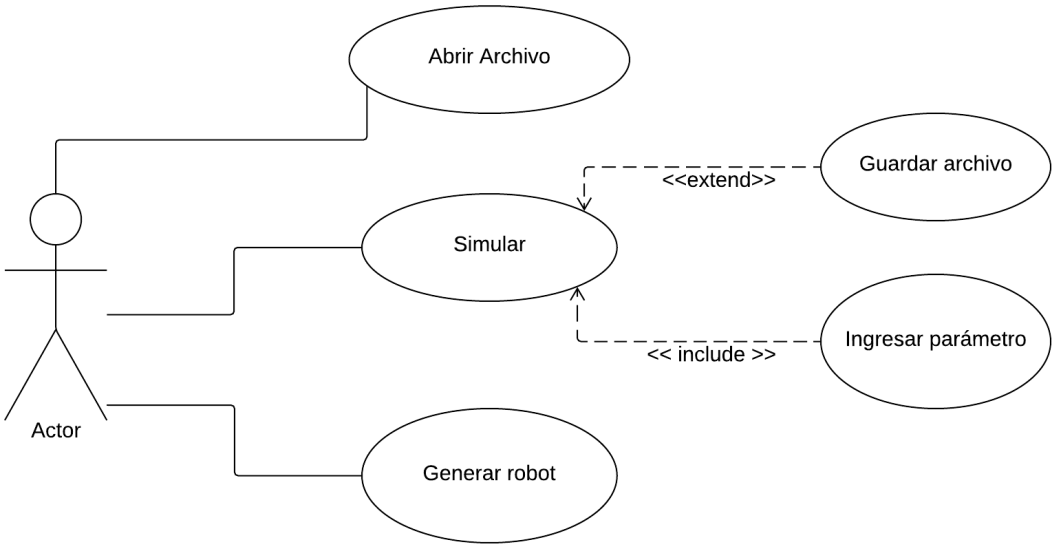


Figura 13: Diagrama de Casos de Uso General.

Tabla 2: Ficha descriptiva C1.

Caso de Uso	**Nombre: Abrir archivo**	
Requisitos asociados	RF1,RF2,RF4,RF7	
Descripción	Permite al usuario buscar dentro del sistema alguna configuración previamente guardada ya sea para emplear como simulación o ver resultados.	
Precondición	Exista un archivo guardado.	
Secuencia Normal	Paso	Acción
	1	El usuario elige la opción de abrir archivo
	2	Se abre el cuadro de diálogo para seleccioanr el archivo
	3	Accede a la información que contiene el archivo
	4	Se visualiza la información en la ventana del simulador

Tabla 3: Ficha descriptiva C2.

Caso de Uso	**Nombre: Simular**	
Requisitos asociados	RF1,RF3,RF5,RF6	
Descripción	Permite al usuario una vez puesto parámetros iniciales poder iniciar y manipular la simulación.	
Precondición	Tener alguna configuración ya en el simiulador	
Secuencia Normal	Paso	Acción
	1	El usuario elige la opción empezar o iniciar
	2	Comienza la evolución en la ventana
	3	Termina la simulación
	4	Se presentan los datos

Tabla 4: Ficha descriptiva C3.

Caso de Uso	**Nombre: Generar robot**	
Requisitos asociados	RF1,RF4	
Descripción	Permite el usuario poder establecer el robot y del mismo modo comunicarlo con el simulador para llevar a cabo el movimiento del robot	
Precondición	Haber terminado la simulación	
Secuencia Normal	Paso	Acción
	1	El usuario elige la opcion de colocar el robot
	2	Conecta con el robot
	3	Empieza la simulación del robot
	4	Para su simulación del robot
	5	Genera comportamiento

3.2.3. Clases

Comprende las clases que conformarán el modelo así como aspectos que son necesarios ya sea para la reutilización de código o diseño del mismo.

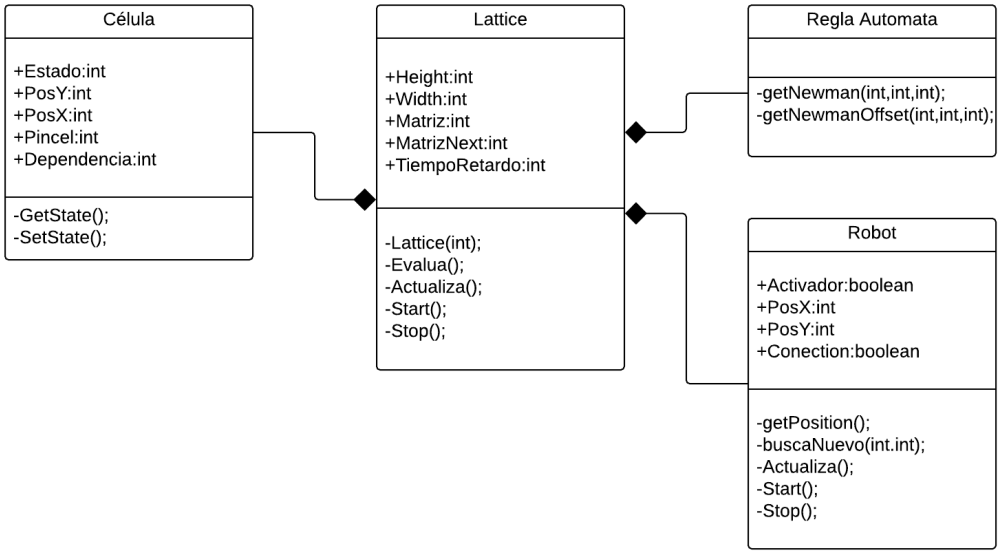


Figura 14: Diagrama de Clases.

3.2.4. Diagrama de Actividades

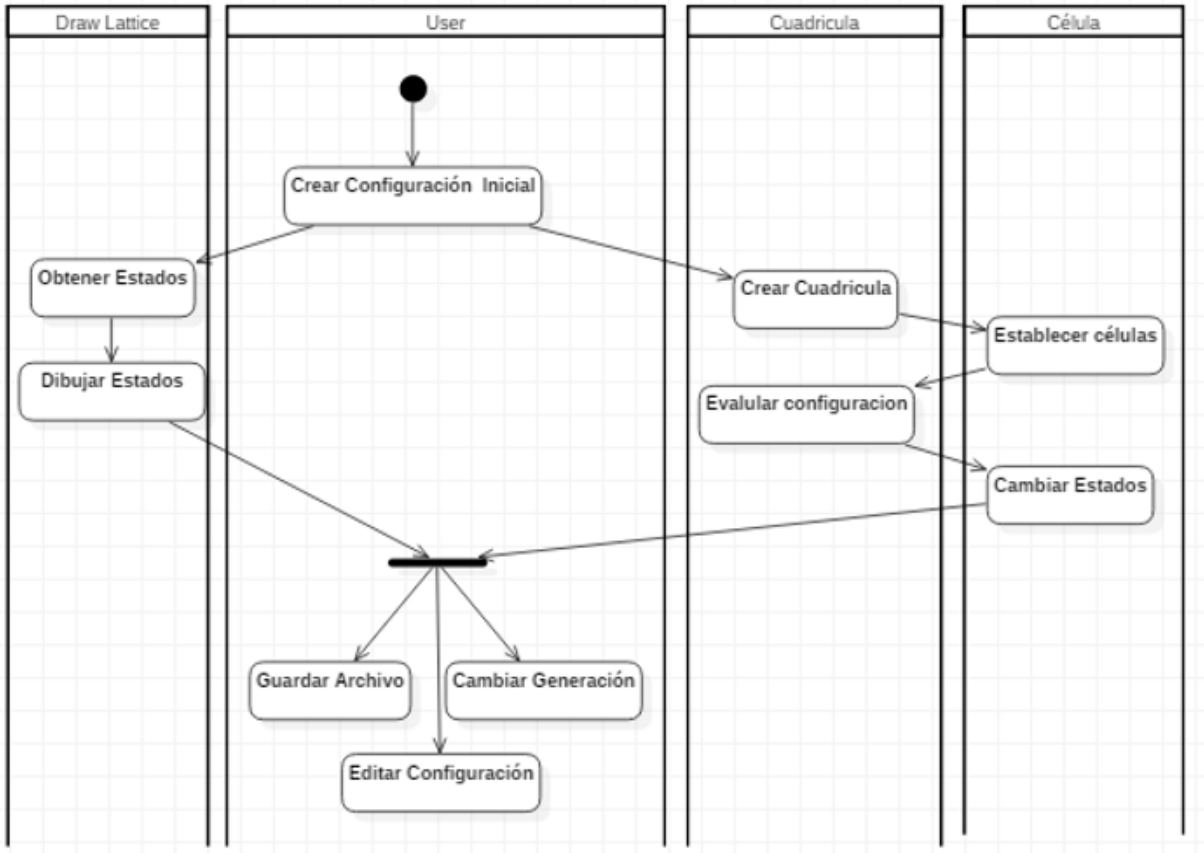


Figura 15: Diagrama de Actividades.

3.3. Requerimientos de la herramienta

Una de las primordiales características que el modelo debe soportar, son espacios de al menos 100 x 100, reflejado en un mapeo que el usuario podrá manipular bajo ciertas condiciones tales como pausa de evolución, diferentes tipo de entradas, mostrar cierta información o gráficos, guardar datos, entre otros.

3.3.1. Software

- **Qt Creator**: Este IDE soporta C++, tiene características similares a NetBeans, fue elegido principalmente por la creación de GUI permitiendo diseñar widgets o diálogos con fácil visualización, además de poder trabajar con las ventajas de C++.
- **Code Composer Studio**: El IDE soporta microcontroladores del mismo Texas Instruments que será el fabricante del cual estará basado el robot teniendo la función de poder programas la propia tarjeta de manera fácil.

3.3.2. Hardware

Los siguientes componentes son parte de hardware que conforma el robot en aproximación se realiza un costo de un poco más de 40 USD.

- **Launchpad MSP430**: Esta tarjeta de desarrollo es el que será empleado tanto por su facilidad, costo y poder soportar 20 pines usando USB para poder programarla, para el proyecto se emplea la versión G2553 con 28 pines.

Figura 16: Microcontrolador MSP430G2.

- **Módulo HC-05**: Este módulo de bluetooth será empleado para el uso junto con la tarjeta al poderse adaptar a esta misma para recibir señales que serán enviadas desde el computador.

Figura 17: Módulo HC-05.

- **4 Pilas AA**: Serán las necesarias para alimentar el robot, es preferible emplear las no recargables debido a que el módulo bluetooth emplea mínimo los 5V para trabajar sin problemas (desconexión), aunque del mismo modo las recargables pueden funcionar hasta un cierto intervalo de tiempo.

Figura 18: Pilas AA marca Duracell.

- **Motores con llanta**: Con un voltaje de operación de 3V a 6V, y diámetro aproximado de las llantas de 66mm.

Figura 19: Llantas DC con motor.

- **Sensor HC-SR04**: Operación de voltaje de 5V, con un rango de muestreo de 1.7cm a 4.5m.

Figura 20: Sensor HC-SR04.

- **Fuente de regulación UA78M33CDCYR**: Alimentando al microcontrolador con los 3.3 Volts requeridos, a diferencia de entre 5V a 6V que nos entregan las pilas AA.

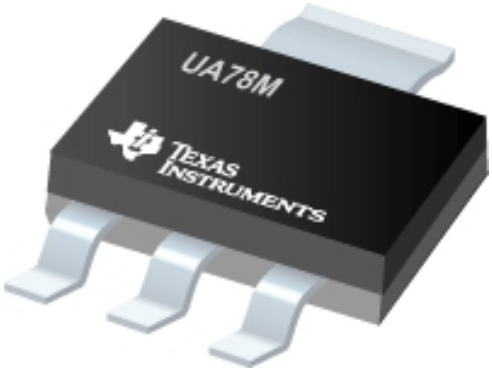


Figura 21: Fuente de regulación.

- **Controlador LV8548MC-AH**: Empleado para el funcionamiento de los motores.

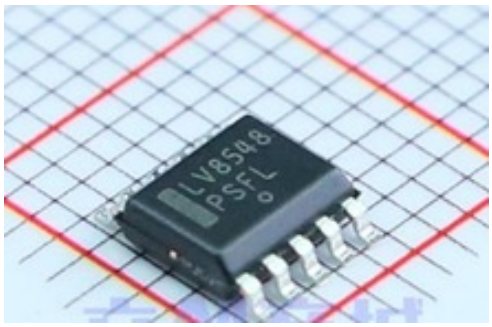


Figura 22: Driver de los motores.

3.4. Justificación de lenguajes a emplear

3.4.1. C++

Siendo su base en 1979 por ” C with Classes ” creado por Bjarne Stroustrup con la inteción de proporcionar un ambiente distribuido para Unix. Con la ventaja de poder desarrolar sistemas complejos además de dar una extensión al lenguaje C.

En 1983 el lenguaje se hace llamar C++, siendo su nombre actual junto con la primera edición del libro ” The C++ Programming Language ”, para poder ser estandarizado en ISO C++.

Es un lenguaje de programación con el objetivo de ser una extensión de C al emplear uso de objetos, para una visión orientada a objetos al ser un lenguaje hibrido.

Puntos.

1. A diferencia de otros lenguajes de programación como Java que es muy común emplearlo, C++ tiene un mejor desempeño en tanto procesamiento de información.
2. Es más fácil el poder crear nuevos tipos de datos, o en este caso objetos.
3. El microcontrolador que se va a emplear, se va a programa en C, por lo que va a optar por compatibilidades.
4. Ya que estamos hablando de un sistema que necesita procesar mucha información y de forma paralela, se necesita rapidez que en comparación contra Java como lo muestra la Figura 23, existe una gran diferencia.
5. El empleo de módulos será otra parte importante en el desarrollo del sistema, por el cual C++ ayuda mucho en este aspecto y del mismo modo, reutilizar código.

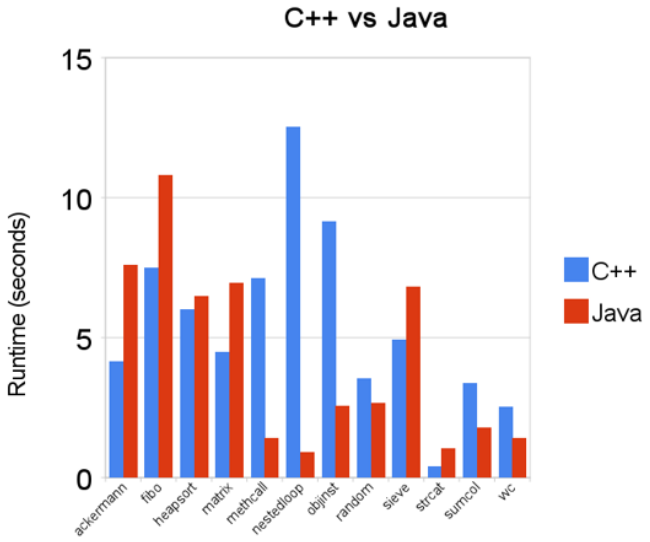


Figura 23: Comparación de tiempo de ejecución Java vs C++ [11].

Portabilidad.

La característica sobre este aspecto, es que el sistema será desarrollado en Linux donde tradicionalmente trata de seguir los lineamientos de los creadores en el mismo sistema, C++ sigue dicho lineamiento, aunque no estrictamente es un lenguaje distribuido para Linux tiene la ventaja de poder ejecutar variantes GNU GCC/C++, sin embargo también existen ciertas dificultades sobre este lenguaje ante la portabilidad.

- Algunas caraterísticas tienen que depender de la implementación, como ejemplo sobre tamaños de diversos tipos de datos o ciertos caracteres.
- A pesar de que C++ es un lenguaje que tenga el empleo de librerías del sistema operativo, las librerías principales no tienen una estandarización tan rigurosa como lo es el lenguaje propio, por ello han cado en ciertas incompatibilidades, esto tiene más importancia sobre distribuciones de Linux.

Eficiencia.

C++ tiene un gran fuerte por la traducción a lenguaje máquina, sin embargo esto no siempre fue así en un principio era complicado el carecer de operaciones u operadores en el procesador, en cambio en la actualidad llegan a poder manipular la memoria con el fin de optimizar procesos o ejecución, a diferencia de un programa que debe traducirse a partir de Código Byte.

4. 2DCAvN(9)

El modelo que se propone para el TT parte de la inspiración de obtener las características de expansión y retracción del Physarum, estas dos etapas comparten durante el desarrollo de la red del organismo el plasmar a la par cada una de las decisiones que se emplean para llegar a la solución que arrojará el hongo, por lo cual partimos de las siguientes ideas.

Criterios base

- El Physarum en un caso ideal donde las zonas por las que pueda pasar estén limitadas a una sola celda (figura), en consecuencia, es fácil obtener una regla simple ya que solo basta con discriminar las zonas donde no exista un nutriente o bien esté rodeado de repelentes, sin embargo la realidad no es así para cualquier experimento, el organismo tiene la libertad de ocupar zonas muy abiertas que debe discriminar.

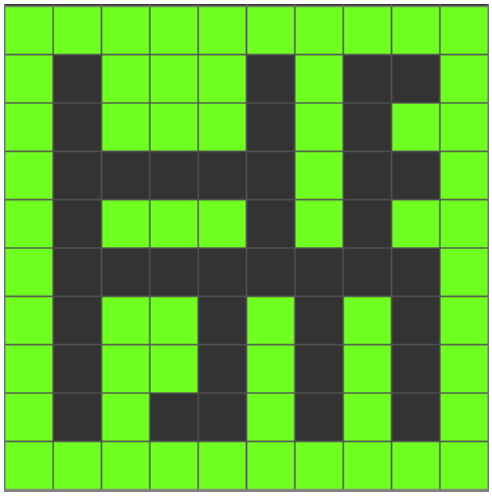


Figura 24: Ejemplo del caso ideal.

- Para el caso del robot, un caso básico para ocupar es simplemente que tenga movimientos de 90 grados, donde la vecindad que apoya a realizar esta implementación es la de von Neumann.

- Al trabajar en un comportamiento colectivo, es necesario meter parámetros que faciliten la función de transición donde no solamente serán evaluados los valores de los estados, sino que cada célula contiene información adicional para construir la red del Physarum, ya que por si sola es imposible generar tanto la variabilidad de resultados esto es debido a que existen ciertas configuraciones dentro del espacio de evoluciones que siempre van a generar los mismo resultados, como ver la evolución del organismo.

- Pensando en un sistema de red, las células deben trabajar bajo dependencias, es decir, si una célula nace o muere, esto debe provocar que las células que dependen de ella estén envueltas a una decisión si son importantes o no en la red.

- Por el momento el interés de modelar el Physarum es obtener soluciones variables, rápidas, ver el desarrollo del Physarum, y emplear dichas soluciones a un robot como una búsqueda de desarrollo de robots de comportamientos no triviales.

- Unas de las opciones que se tienen que pensar es que el organismo puede o no seguir explorando. El seguir explorando llega a ser conveniente para obtener una cercanía del organismo donde, a pesar de que se unieron los puntos o nutrientes, el Physarum sigue explorando su entorno y no recae en una configuración estática. Por el contrario el no seguir explorando es conveniente para obtener una solución inmediata para emplear ya sea con el robot, obtener información de la red y realizar la comparación de resultados.

4.1. Vecindad a emplear

- Vecindad de von Neumann: La decisión de tomar esta vecindad es debido a que el robot va a emplear las rutas trazadas por el AC, luego entonces esta vecindad será la adecuada para obtener rutas sin diagonales.

Entonces definimos el autómata celular de radio 1 de la siguiente manera.

- $d = 2$
- $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$
- $N = \{(-1, 0), (0, 1), (0, 0), (1, 0)(0, -1)\}$
- $f : S^5 \to S$

Además por la definición del AC, recordemos que empleamos la denotación $c(\overrightarrow{n})$.

Los estados propuestos en la tabla de estados son representación que tienen dentro del sistema, la transición que tendrán siempre y cuando cumplan con determinadas características que a continuación se hablará.

Tabla 5: Tabla de estados del AC.

Color	Estado	Evoluciona	Representa	Parte del Physarum
	q_0	q_7	Campo libre	
	q_1	q_6	Nutriente no encontrado	
	q_2	q_2	Repelente	
	q_3	q_3	Punto inicial	X
	q_4	q_5	Gel en contracción	X
	q_5	q_0,q_8	Gel con compuesto	X
	q_6	q_6	Nutriente hallado	
	q_7	q_4	Expansión del Physarum	X
	q_8	q_5	Gel sin compuesto	X

4.2. Condiciones de transición

- **Campo libre (estado 0):** Se convertirá en expasión del Physarum, si y solo si, existe en su vencindad de Neumann un punto inicial, gel en contracción o nutriente hallado.

$$c(\overrightarrow{n})_{t+1} = \left\{ 7 \quad \text{if} \quad c(\overrightarrow{n})_t = 0 \quad \text{and} \quad q_j > 0 \quad j = 3, 4, 6 \right. \tag{9}$$

- **Nutriente no encontrado (estado 1):** Cambiará a nutriente hallado si existe al menos un gel con compuesto o nutriente hallado.

$$c(\overrightarrow{n})_{t+1} = \left\{ 6 \quad \text{if} \quad c(\overrightarrow{n})_t = 1 \quad \text{and} \quad q_j > 0 \quad j = 5, 6 \right. \tag{10}$$

- **Repelente (estado 2) :** Estado periódico.

$$c(\overrightarrow{n})_{t+1} = \left\{ 2 \quad \text{if} \quad c(\overrightarrow{n})_t = 2 \right. \tag{11}$$

- **Punto inicial (estado 3):** Estado periódico.

$$c(\overrightarrow{n})_{t+1} = \left\{ 3 \quad \text{if} \quad c(\overrightarrow{n})_t = 3 \right. \tag{12}$$

- **Gel en contracción (estado 4):** Evoluciona a gel con compuesto, si existe otro gel con compuesto, punto inicial o nutriente hallado con el que pueda realizar la unión.

$$c(\overrightarrow{n})_{t+1} = \left\{ 5 \quad \text{if} \quad c(\overrightarrow{n})_t = 4 \quad \text{and} \quad q_j > 0 \quad j = 3, 4, 6 \quad connect = true \right. \tag{13}$$

- **Gel con compuesto (estado 5):** Morirá si no existen uniones dependiendes de él, para ellos se emplea un parámetro para determinar dicha dependencia, de lo contrario pasará a gel sin compuesto.

$$c(\overrightarrow{n})_{t+1} = \left\{ \begin{array}{ll} 0 & \text{if} \quad c(\overrightarrow{n})_t = 5 \quad \text{and} \quad q_j < 2 \quad j = 5 \quad dep = 0 \\ 8 & \text{else} \end{array} \right. \tag{14}$$

- **Nutriente hallado (estado 6):** Sin transición.

$$c(\overrightarrow{n})_{t+1} = \left\{ 6 \quad \text{if} \quad c(\overrightarrow{n})_t = 6 \right. \tag{15}$$

- **Expansión del Physarum (estado 7):** Evoluciona a gel en contracción si existe en su vecindad un punto inicial, gel con contracción, con nutriente o nutriente hallado.

$$c(\overrightarrow{n})_{t+1} = \left\{ 4 \quad \text{if} \quad c(\overrightarrow{n})_t = 7 \quad \text{and} \quad q_j > 0 \quad j = 0, 4, 1, 6 \right. \tag{16}$$

- **Gel sin compuesto (estado 8):** Evoluciona a gel con compuesto directamente al ser una representación de transporte en su red.

$$c(\overrightarrow{n})_{t+1} = \left\{ 5 \quad \text{if} \quad c(\overrightarrow{n})_t = 8 \right. \tag{17}$$

4.3. Ejemplo de aplicación

Partimos de la siguiente configuracón inicial, con un arreglo de 11 x 11.

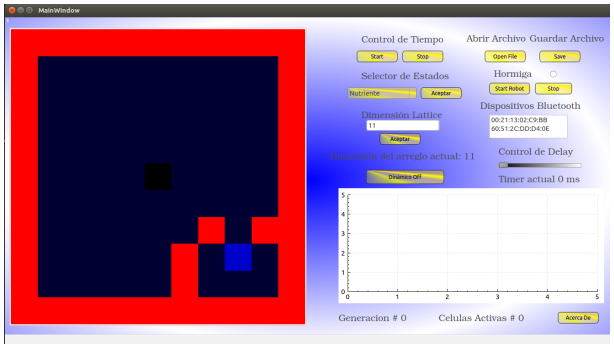


Figura 25: Ejemplo de la regla, T = 0.

Para la primera evolución se aplica la regla 1, Ecuación 9 , donde el campo libre va a detectar el punto inicial, sin embargo dentro de cada celda que es candidato a nacer o bien pasar al estado q_7, se toma en cuenta si tiene las condiciones o no para nacer, de ser así, pasa a dicho estado como se observa en la Figura 26.

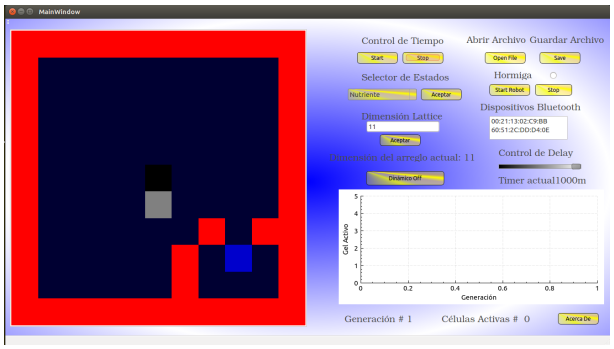


Figura 26: Ejemplo de las regla, T = 1.

En la segunda evolución nuevamente se toma la regla 1 para el campo libre que evolucionó, sin embargo se observa que la célula en estado q_{s7}, pasa a un estado q_4, por criterio de la regla 8, Ecuación 16, mostrado en la Figura 27.

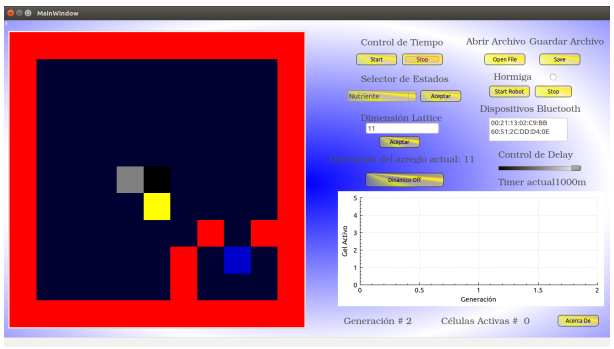


Figura 27: Ejemplo de las reglas, T = 2.

Observamos que nace una nueva célula con q_7, por la misma regla 1 y anteriormente la célula que estaba en estado q_7, cambio a q_4, por la regla 2.

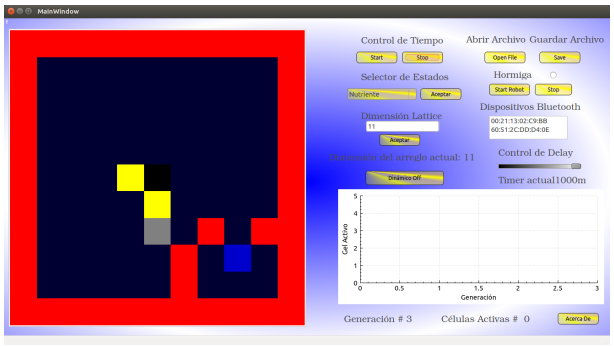


Figura 28: Ejemplo de las reglas, T = 3.

Seguimos bajo las mimas reglas, hasta que por criterio de la regla 5, Ecuación 13 , una célula en q_4, toma la decisión de convertirse en parte de la red de solución al poder realizar alguna conexión con alguno de sus vecinos de estados que menciona dicha regla, vease en la Figura 29.

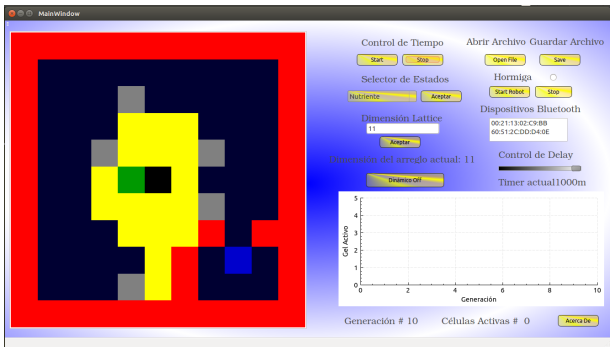


Figura 29: Ejemplo de las reglas, T = 10.

En la siguiente evolución, la célula en q_5, pasa a q_8 por la regla 6, Ecuación 14, este caso solamente representa como un transporte o movimiento dentro de la estructura del plasmodio, además al tener dependencia, no puede morir, vease la Figura 30.

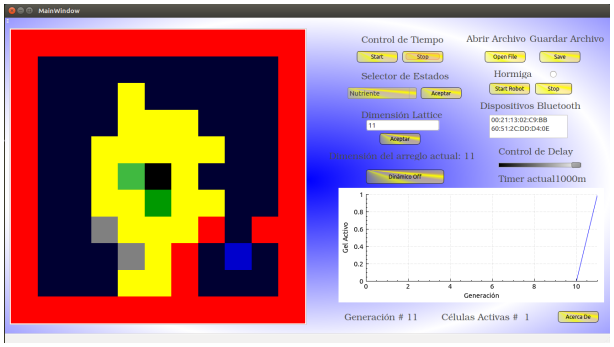


Figura 30: Ejemplo de las reglas, T = 11.

En la siguiente transición la célula en estado q_8, regresa a q_5, por la regla 9, Ecuación 17, el resto de elementos siguen teniendo su evolución a consecuencia de reglas anteriores. Figura 31.

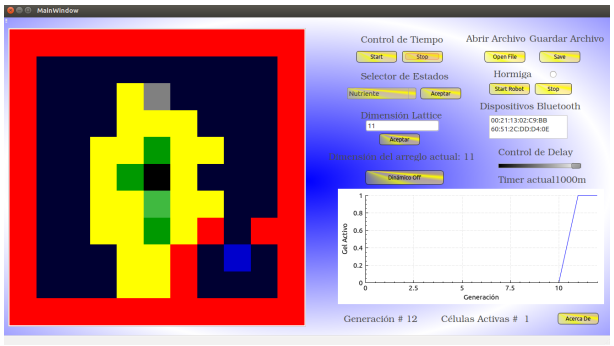
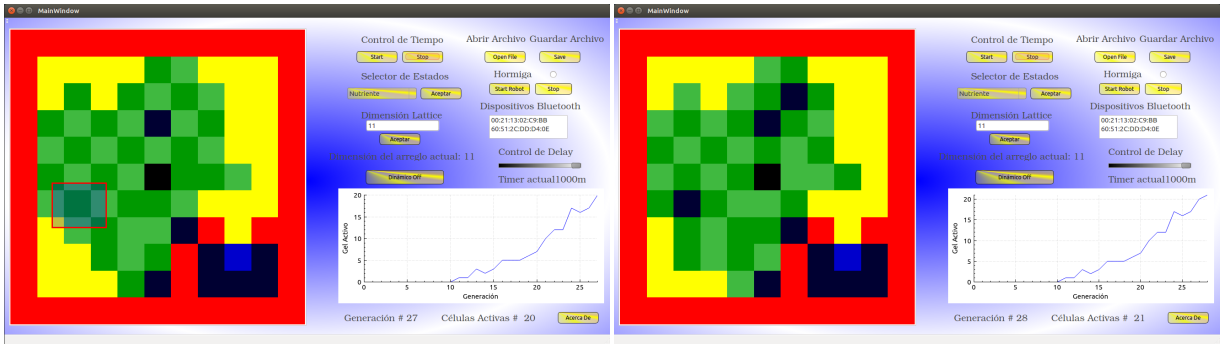


Figura 31: Ejemplo de las reglas, T = 12.

Tras una serie de evoluciones empiezan a morir células, con base a la regla 5 Ecuación 13, a) donde la célula marcada, b) morirá debido a que no puede extenderse o encontrar alguna dependencia con algun vecino o célula adyacente esto hace que dicha célula ya no sea de utilidad para la red.

Lo siguiente es que si alguna célula estaba conectada a ella, también morirá por la misma regla 5, traducido al Physarum, recordemos que el organimos puede saber donde ya ha estado, entonces para el ya no es necesario volver a explorar la zona donde desaparecio puesto que sabe que ahi al menos en su primer exploración habrá nada Figura 32.



(a) (b)

Figura 32: Ejemplo de la regla 5.

Bajo esas reglas el Physarum llega a retraerse hasta ser un poco más pequeño después de ciertas evoluciones, donde aunque su plasmodio ya haya tocado el nutriente, no existe expansión a esa zona. Figura 33.

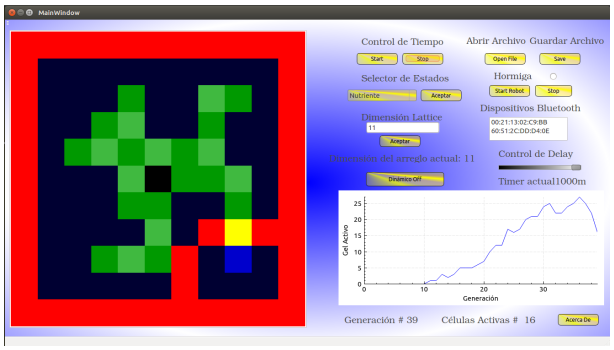


Figura 33: Ejemplo de las reglas T = 39.

Toma hasta la generación 49, que el Physarum se extiende el nutriente donde este cambia de estado q_1 a q_6, por la regla 2, Ecuación 10, vease en la Figura 34.

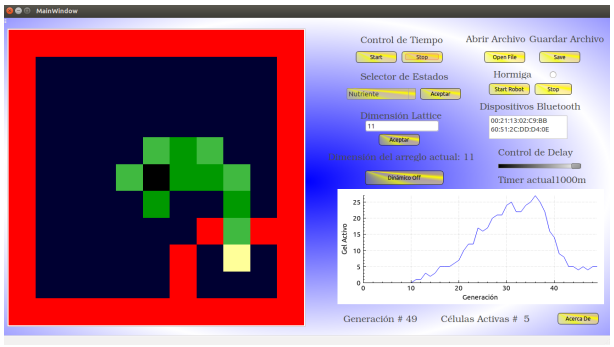


Figura 34: Ejemplo de lasreglas T = 49.

Para la generación 51, las zonas libres que se encuentran en el nutriente las puede explorar el Physarum ya que una vez hallado el nutriente se convierte en una entidad que el organimos logra cubrir y para este caso, puede pasar sobre él para seguir explorando, entonces la zona libre pasa a estado q_7 por la regla 1. Figura 35.

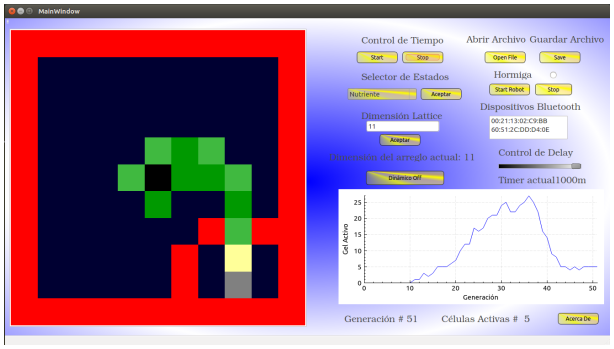


Figura 35: Ejemplo de lasreglas T = 51.

Una vez evaluado el sistema, bajo los criterios ya mencionados, el autómata celular arroja el siguiente resultado, donde por último las células que forman parte de la red en el nutriente persisten y no mueren por la regla 5.

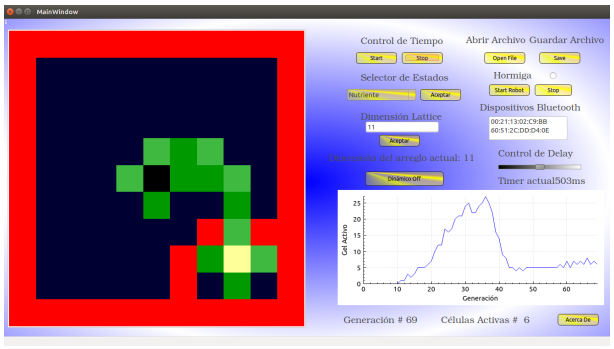


Figura 36: Ejemplo de lasreglas T = 69.

5. eBots

Para aplicar la simulación presentada se empleó una arquitectura denominada eBots [7], desarrollada en la Escuela Superior de Cómputo bajo la asesoría del Dr. Luz Noé Oliva Moreno y el Dr. Juárez Martínez Genaro junto con la alumna Rosa Graciela Chávez Barrera (figura 37).

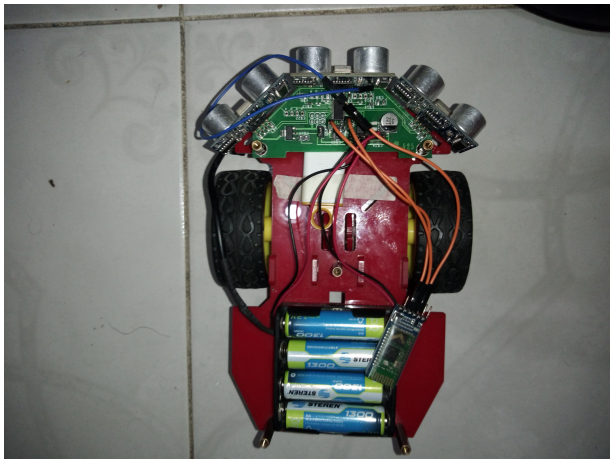


Figura 37: eBot

Donde el objetivo de este proyecto fue el presentar un comportamiento colectivo por medio de colonias, empleando sensores de proximidad junto con el microcontrolador MSP430G23 de Texas Instruments, que parten de la idea de emplearlos para el estudio de comportamientos swarm, estos robots tiene una dimensión aproximada de 4.72 x 7.48 x 3.15, conformado de 4 pilas AA, una base, dos motores, dos llantas, 3 sensores de distancia, un regulador de voltaje, microcontrolador y un capacitor. Dichos componentes previamente especificados anteriormente estan pensados para obterner una arquitectura simple con un número bajo de componentes.

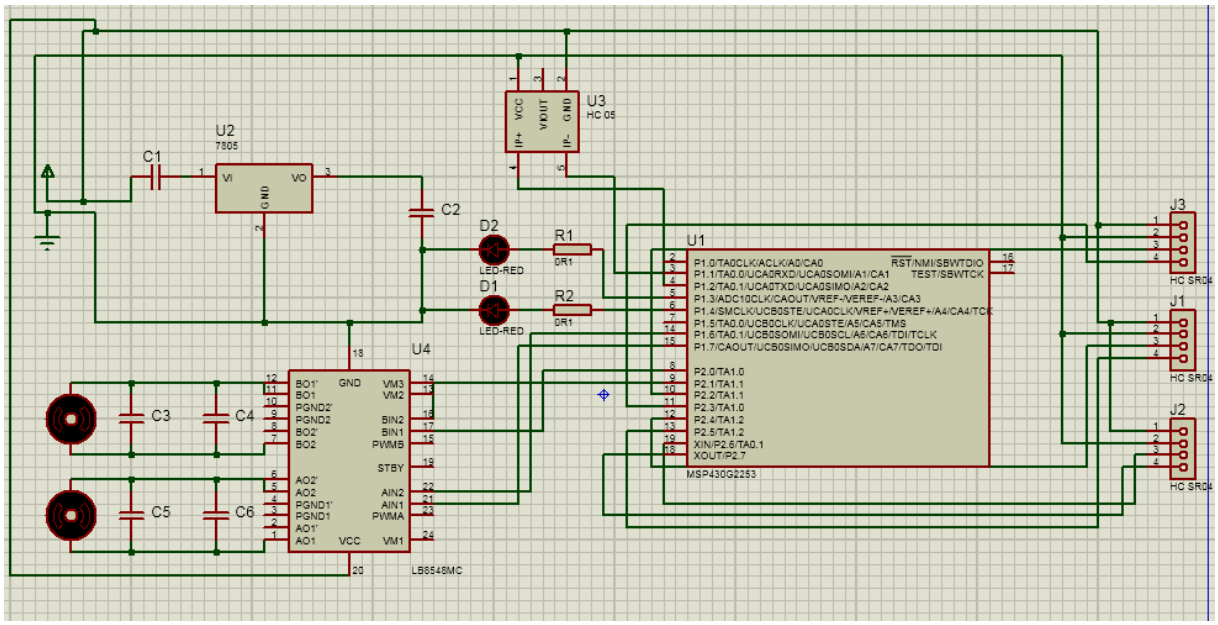


Figura 38: Diagrama del robot aproximado.

5.1. Comportamiento del Robot

Para el presente apartado nos basamos en el funcionamiento de que el Robot deba recibir los comandos de movimiento para poder moverse, para ello tendremos 3.

- 1: Rotar a la izquierda 90 grados.
- 2: Rotar a la derecha 90 grados.
- 3: Avanzar.

Por el momento se maneja una cierta distancia sobre el avance, donde el motor se activará n microsegundos para realizar dicha acción. Los pasos que sigue el robot para poder leer los comandos que se le mandan desde el programa son los siguientes y pueden ser observados en la Figura 39 :

1. El robot espera la conexión del dispositivo donde una vez que lo haya hecho, entra a la lectura de datos que se le vayan a enviar, en donde se recibirá los movimientos antes mencionados con los mismos números.
2. Por cada movimiento recibido lo almacena en un arreglo, mientras la acción no sea 0 (parar lectura), seguirá leyendo.
3. Al parar la lectura el robot empieza a emplear el arreglo, leyendo lo que se haya almacenado para darle el movimiento.

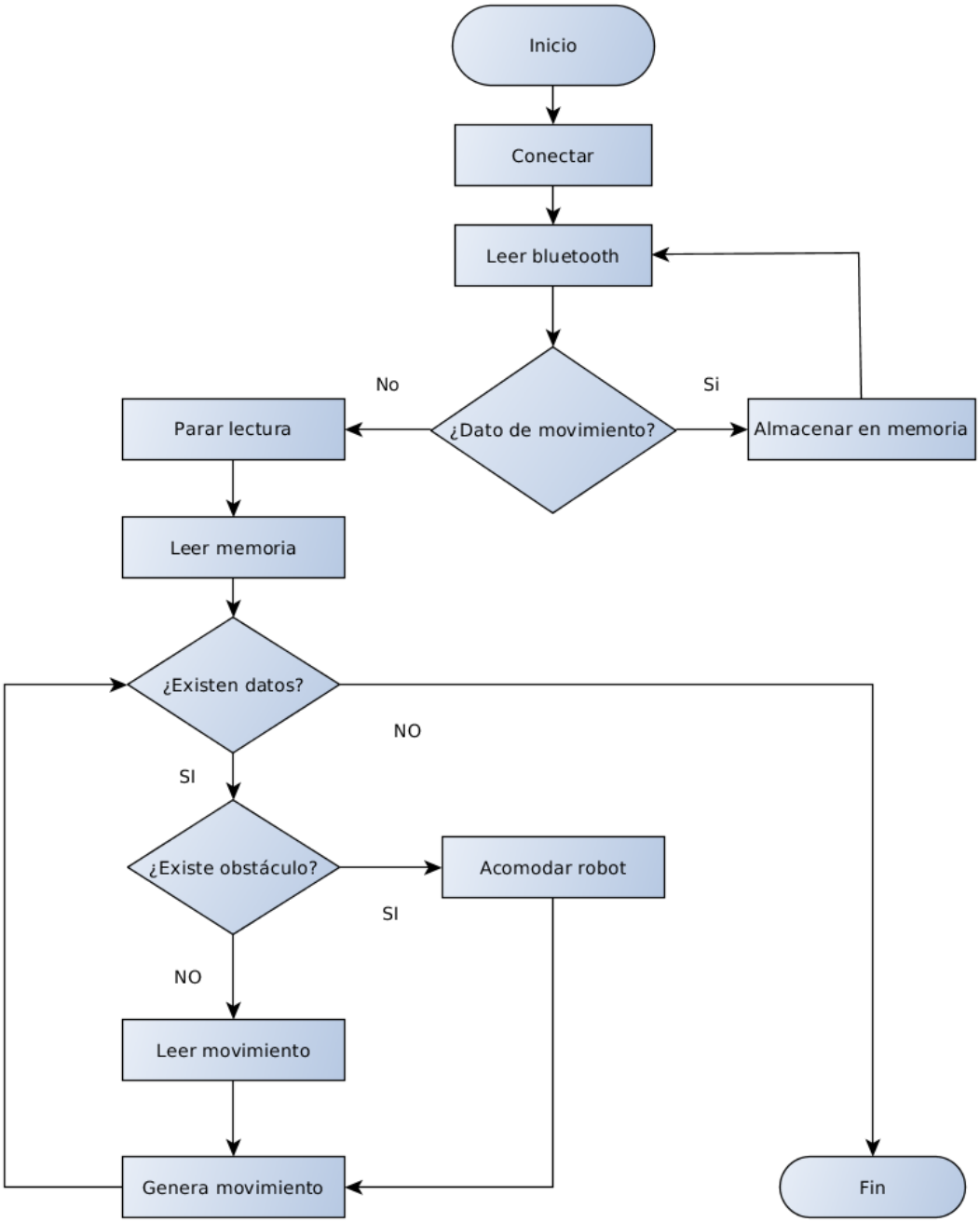


Figura 39: Diagrama de flujo del robot.

Si desearamos ver los comandos como una secuencia o cadena, se habla de que el robot al final del día lo que está leyendo es una expresión regular, por el cual el patrón a describir es la secuencia de movimientos o acciones del robot. Para el robot denotamos las cadenas con alguna acción de movmiento, seguido de cualquier cantidad de acciones que definiremos a continuación.

Sea A la acción de avanzar, I giro a la izquierda y D giro a la derecha, definimos nuestro alfabeto:

$$\Sigma = \{A, I, D\} \tag{18}$$

Luego definimos la siguiente expresión regular

$$ER = (A + I + D)(A + I + D)^* = (A \cup I \cup D)(A \cup I \cup D)^* = \{A, I, D\}\{A, I, D\}^* \tag{19}$$

5.2. Ajuste de avance

Para realizar dichas pruebas se toma una referencia que se puede acoplar dependiendo a la distancia que debe avanzar el robot como parámetros actuales tiene lo siguientes: avance de 8 cm aproximadamente y duración de los motores 250 microsegundos. Entonces la razón en la que avanzan es $3.2cm/s$. Para la dimensiones de la maqueta más adelante expuesta se emplean celdas de 18 x 18 cm, luego entonces ¿ Cómo acoplarlo a diferentes dimensiones?.

A fin de acoplarlo con diferentes dimensiones tenemos que pensar que el similador se comporta con cuadros regulares, esto hacer pensar que en un caso de aplicación a diferentes maquetas o tamños tenemos que buscar la manera de ver que sean cuadros los que hay que obtener, sin importar tal vez el tamaño de las celdas que como se mencionó para el caso de la maqueta a emplear es de 18 x 18 aproximadamente.

- Pongamos un ejemplo en que en que tengamos las dimensiones que presenta la Figura 40. Dichas dimensiones se tienen que pensar de tal forma que se pueda dividir en cuadros, con el fin de poder meterlo al simulador.

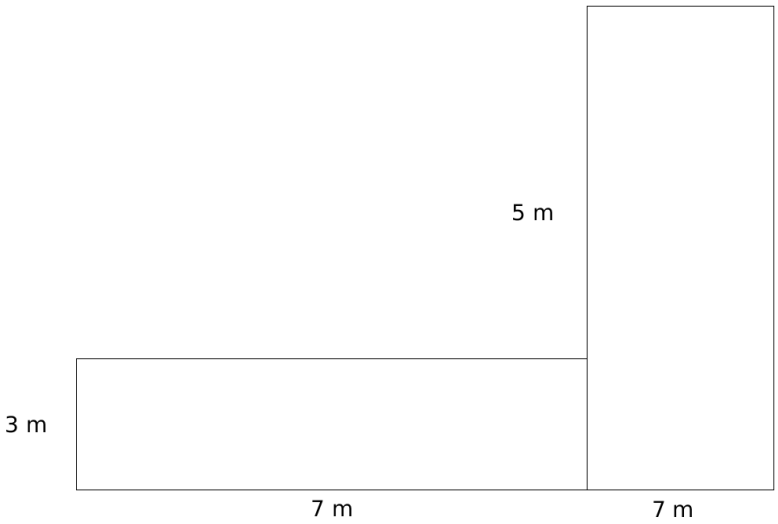


Figura 40: Ejemplo de cambio de dimensiones.

- La propuesta de división será la que se muestra en la siguiente Figura 41.

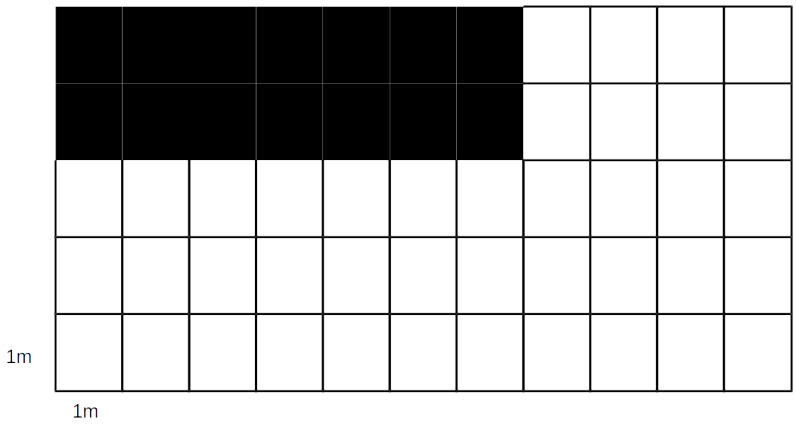


Figura 41: Ejemplo de división para el simulador.

- Ahora dicha división la pasamos al diagrama, recordando que es una rejilla regular, donde para la condición más larga será necesario de 13 x 13, debido a que las orillas son los repelentes o limites del Physarum, quedando de la siguiente manera Figura 42.

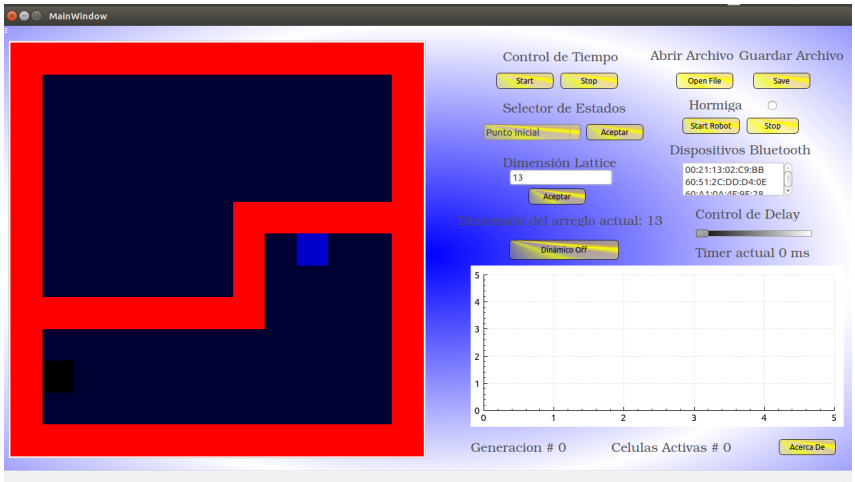


Figura 42: Dibujo del simulador.

Ahora bien, el robot originalmente avanza 8 cm aproximadamente, sin embargo 8 cm es poco para lograr pasar de una celda a otra a las distancias nuevas, pensando en que el robot idealmente queda en medio de cada celda la distancia promedio que hay entre los puntos medios de cada una es de 1 m, entonces empleando una regla de 3 dado los datos que obtuvimos anteriormente. Sea $d = 8cm$ y $t = 250\mu s$ el tiempo de activación de los motores.

Por regla de 3

$$\frac{8cm}{250\mu s} = \frac{100cm}{x\mu s} \tag{20}$$

Donde x es el valor que requerimos para avanzar 100 cm o 1 m siendo la activación de los motores.

$$x = 3,125\mu s \tag{21}$$

El cual ese será el dato para activar los motores.

6. Pruebas

6.1. Requerimientos

Para las simulaciones, las características del equipo en el que fue desarrollado son las siguientes.

- **Sistema Operativo**: Ubuntu 16.04 LTS 64 bits.
- **Procesador**: Intel Celeron N2830, 2.16 Ghz.
- **Memoria Ram:** 4 Gb.
- **Bluetooth**: CSR 4.0 Dongle.

Como requerimientos importantes el Sistema Operativo debe de ser alguna distribución de Linux, debido a que el IDE en el que fue desarrollado el sistema, **el framework del Bluetooth que proporciona QtCreator no trabaja en sistemas Windows** [6], el resto de requerimientos se pueden considerar mínimos.

6.2. Entorno gráfico

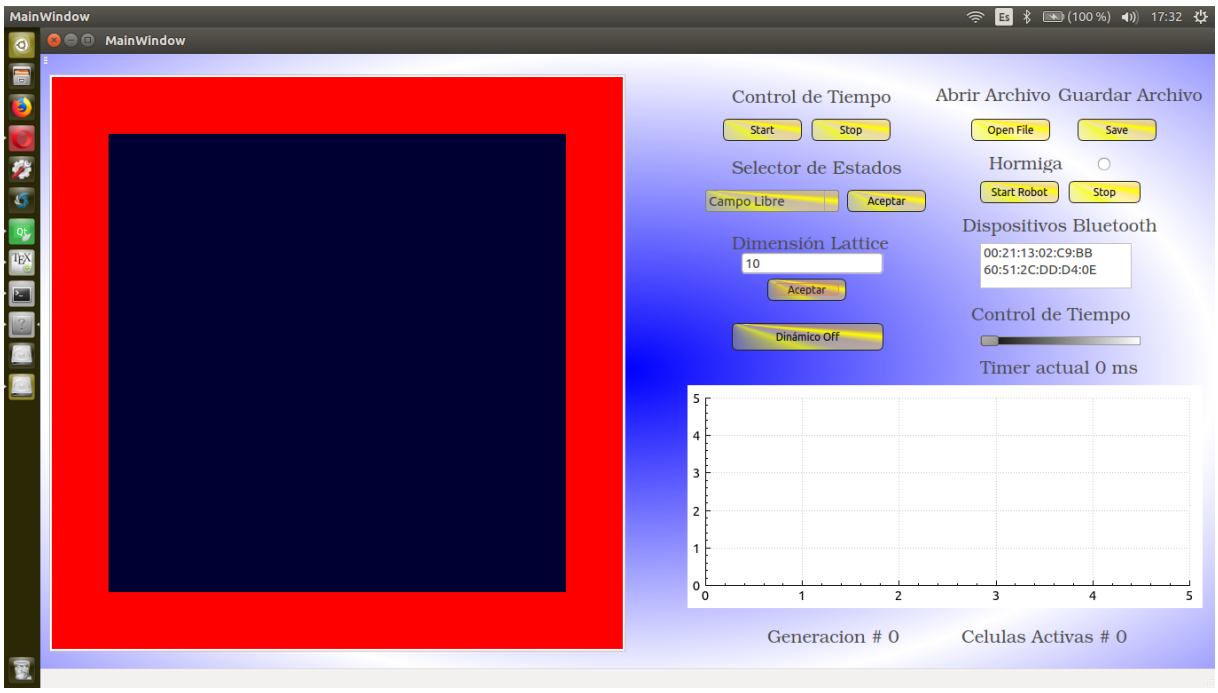


Figura 43: Pantalla del simulador.

- **Control de tiempo:** Empleado para parar (botón Start) o continuar (botón Stop) la simulación.

[6] Véase en: https://doc.qt.io/qt-5.10/qtbluetooth-index.html

- **Controles de archivos:** Ya sea que guarde una configuración dentro de la simulación (botón Save) o bien emplear alguna configuración previamente creada (botón Open File).
- **Selector de estados:** Provee de los estados que puede tomar cada celda, para ello se selecciona el estado que se desea en la lista y después en el botón aceptar para establecer el pincel.
- **Botón de comportamiento dinámico:** Este botón inicialmente en "Dinámico Off"determina si la simulación o bien el Physarum seguirá creciendo o explorando su entorno.
- **Dimensión de Lattice:** Se emplea para cambiar la longitud o dimensión de la cuadrícula de la simulación, tan solo es ingresar un número entero positivo dentro de la caja de texto y hacer click en aceptar.
- **Herramientas del robot:** Para el robot tenemos varias herramientas, primeramente un radio button empleado para meter el robot o en este caso que llamaremos hormiga dentro de la simulación, su botón para empezar su recorrido (botón Start Robot) y uno para parar el recorrido (botón Stop), por último un área de texto para mostrar los dispositivos Bluetooth que nuestro equipo detecte para conectarse en este caso tenemos que buscar el que sea del robot.
- **Control de Delay:** El Slider proporciona el poder cambiar que tan rápido o lento vaya la simulación.

6.3. Prueba del simulador con Red de Tokyo

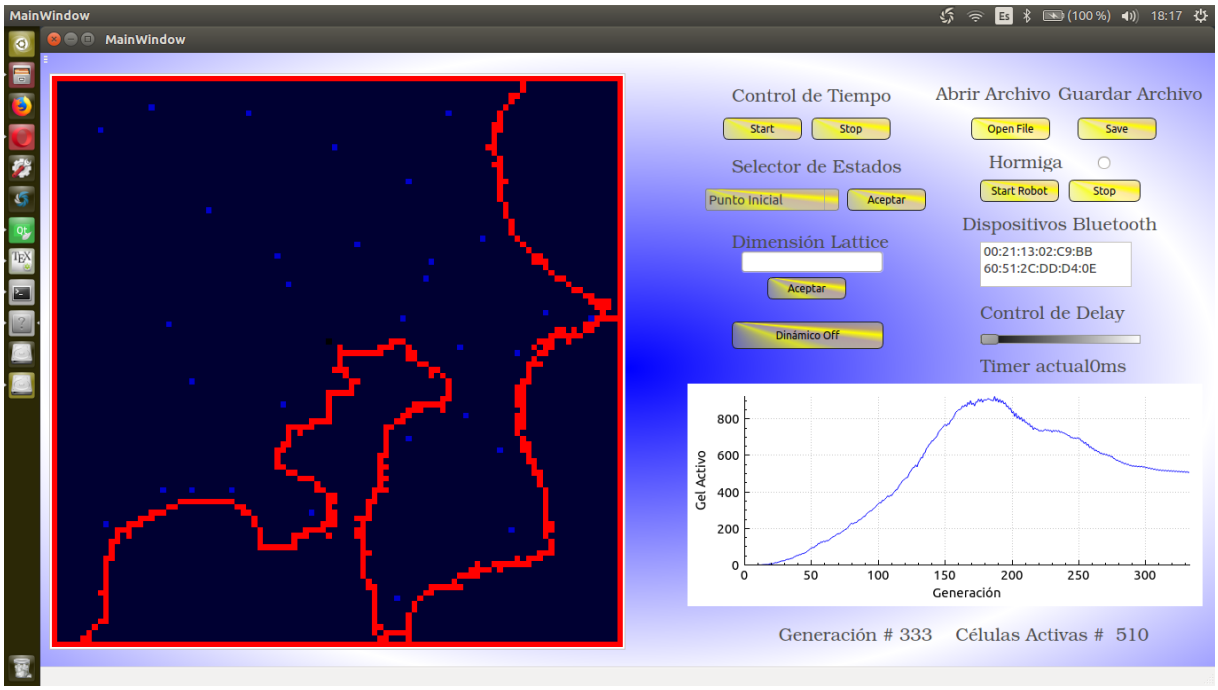


Figura 44: Tokyo en T = 0.

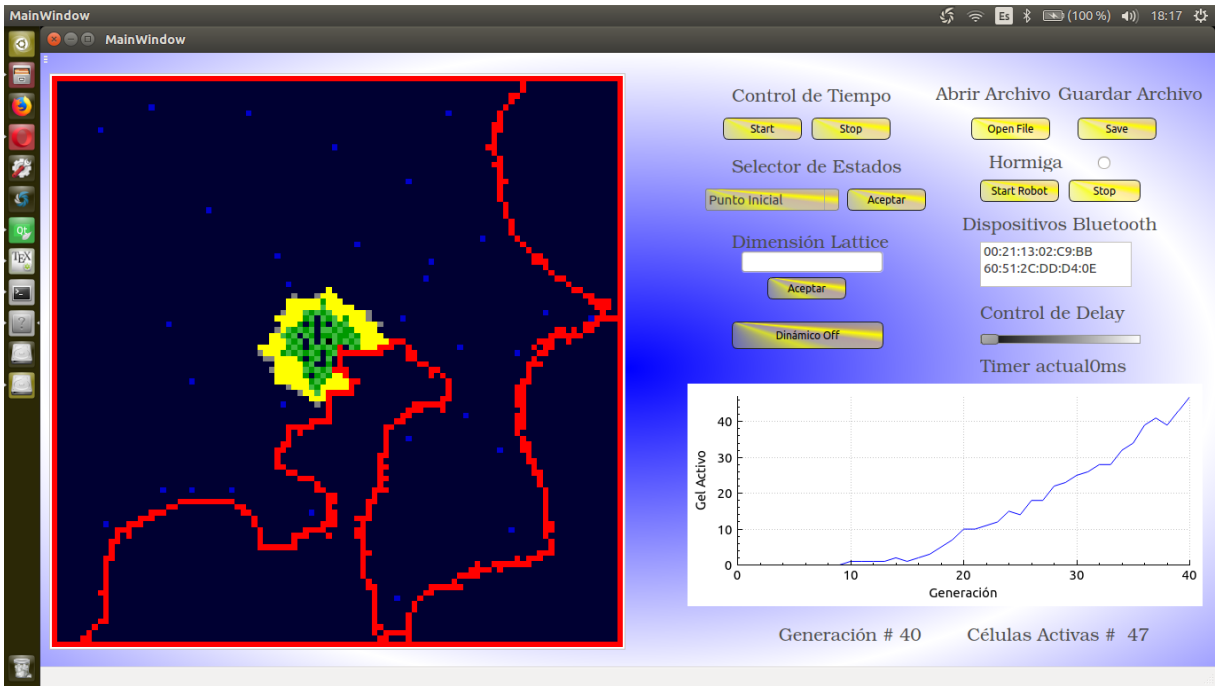


Figura 45: Tokyo en T = 40.

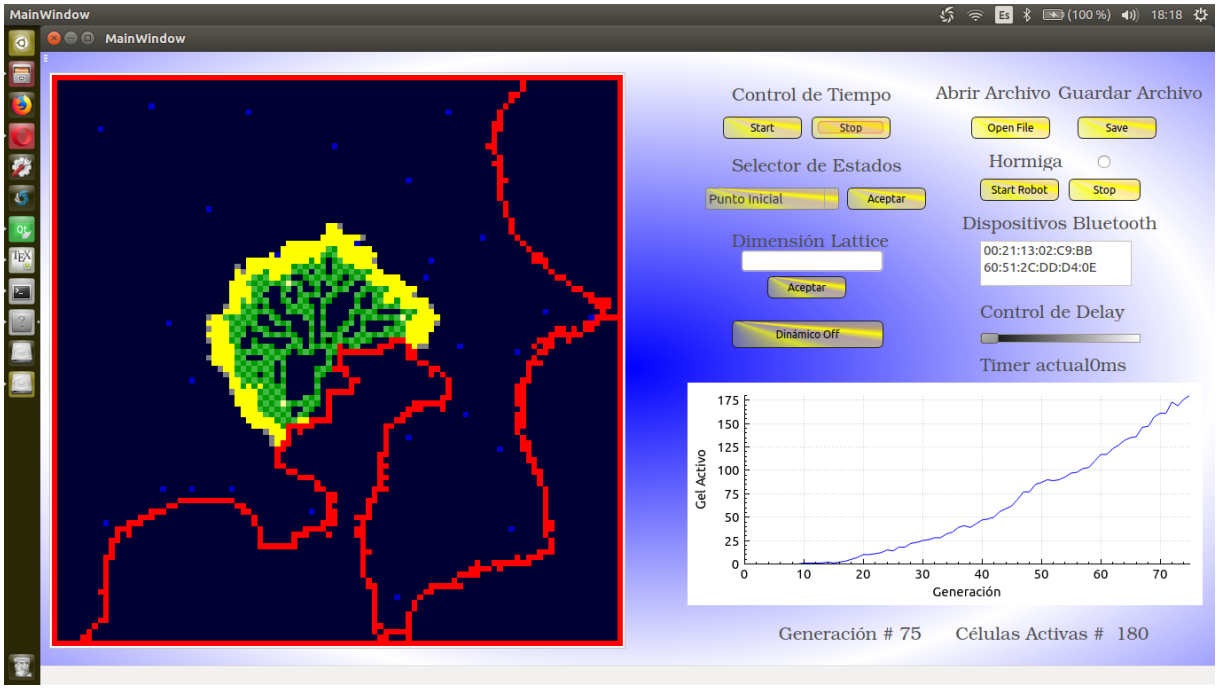


Figura 46: Tokyo en T = 75.

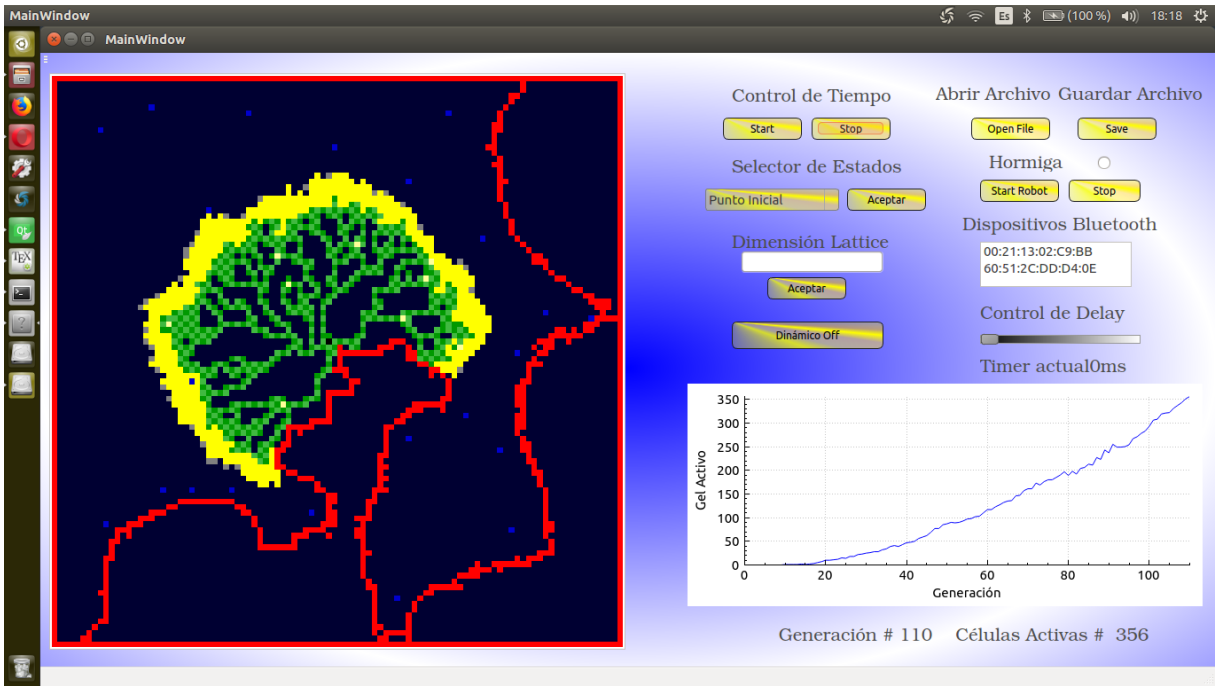


Figura 47: Tokyo en T = 110.

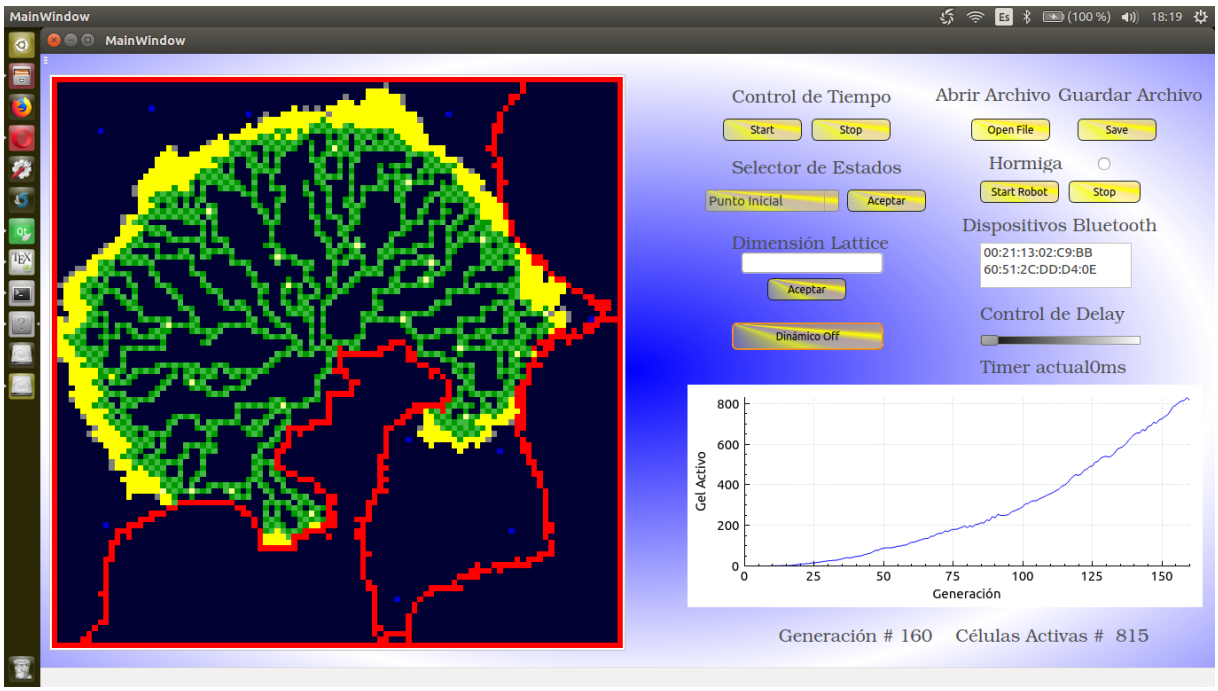


Figura 48: Tokyo en T = 160.

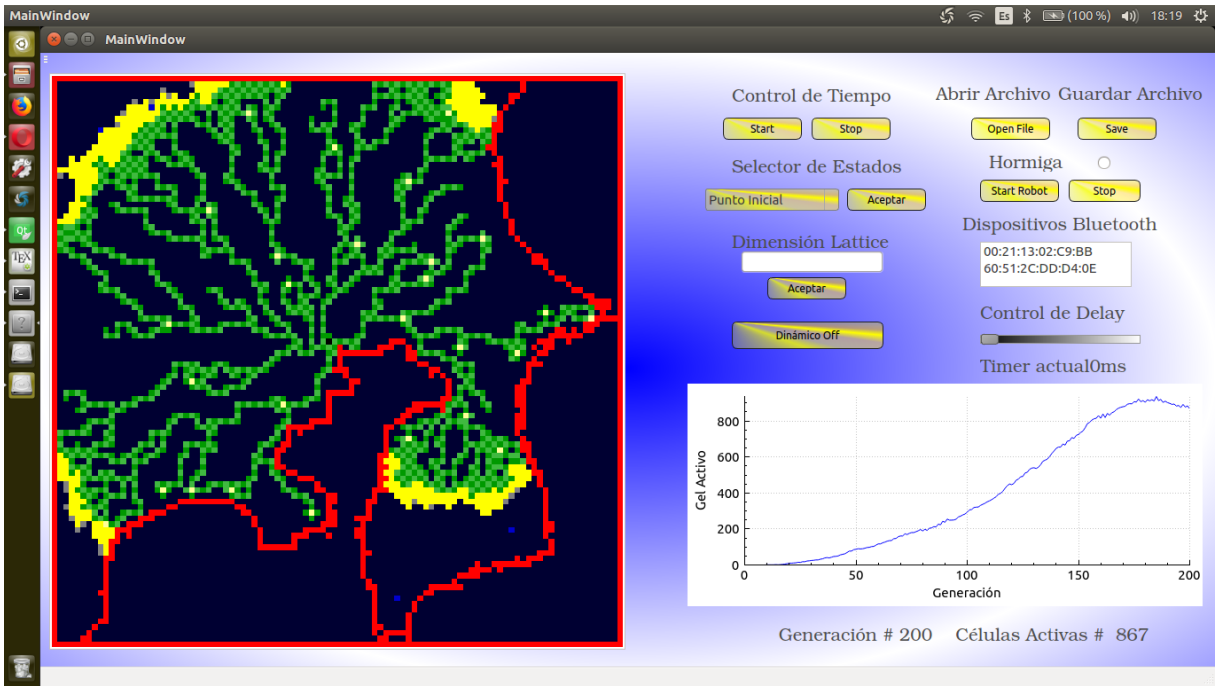


Figura 49: Tokyo en T = 200.

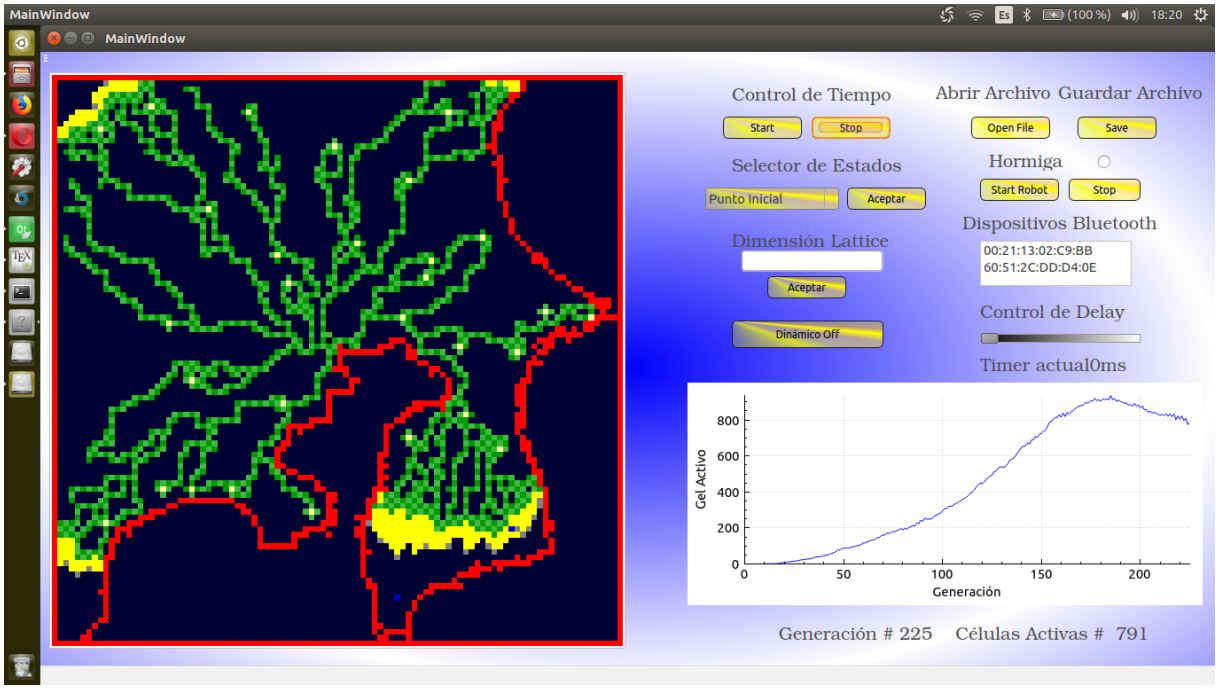


Figura 50: Tokyo en T = 225.

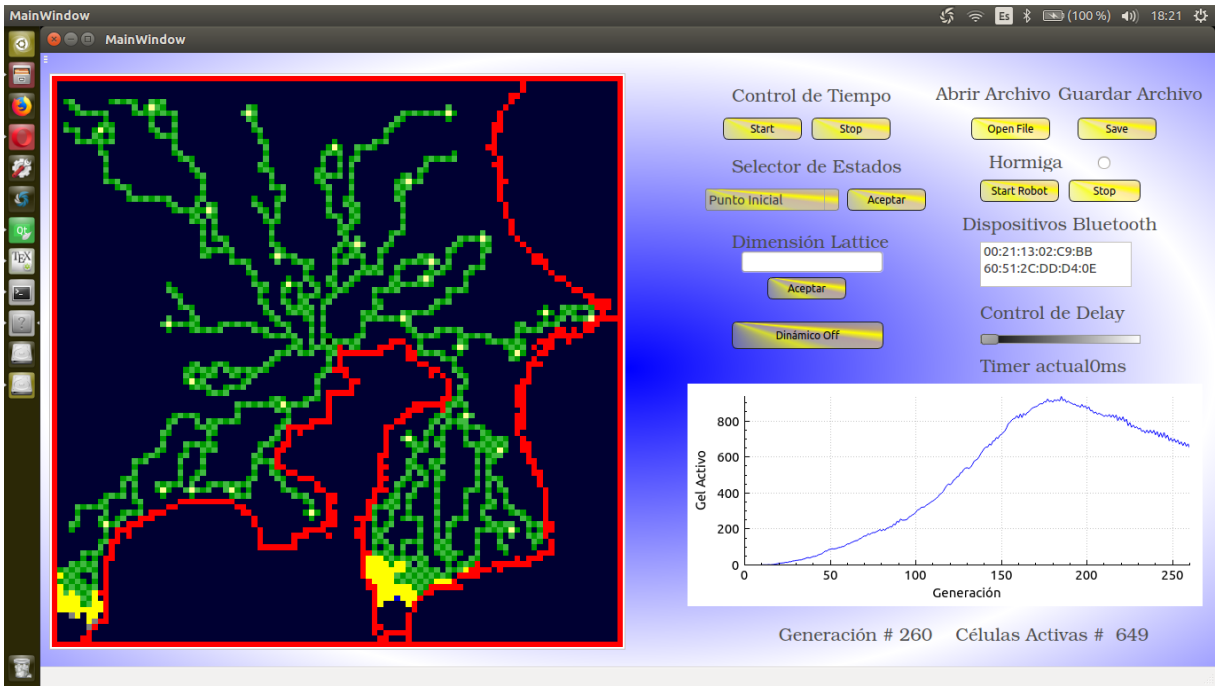


Figura 51: Tokyo en T = 260.

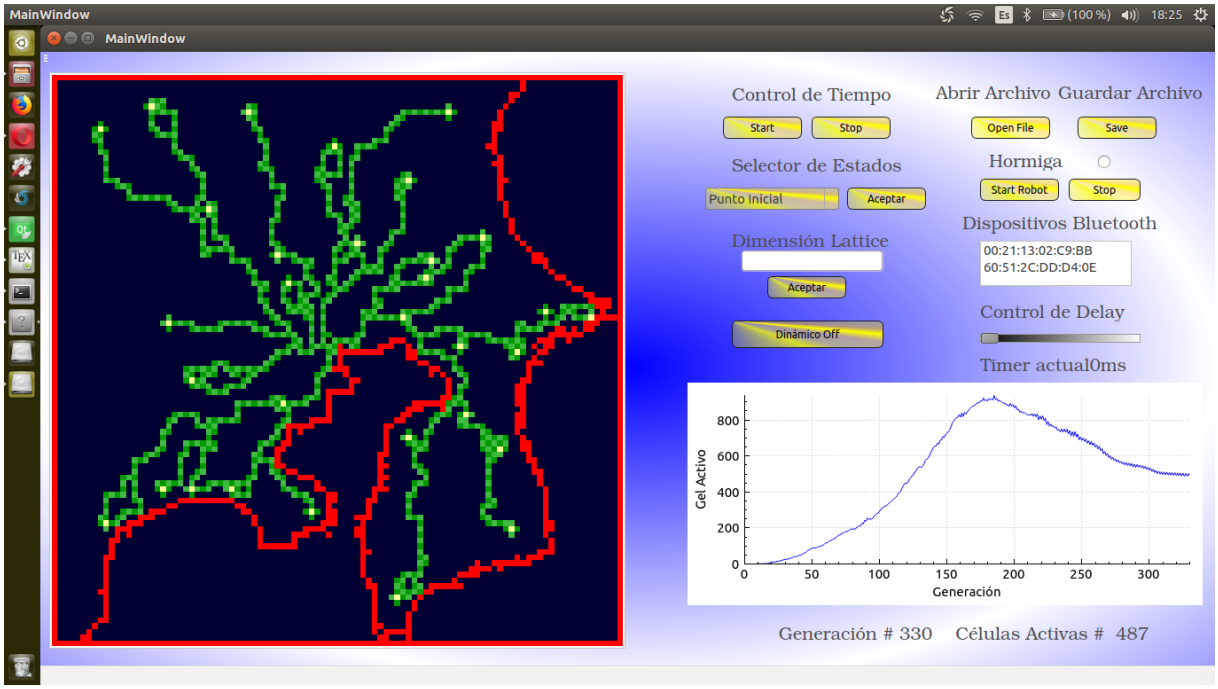
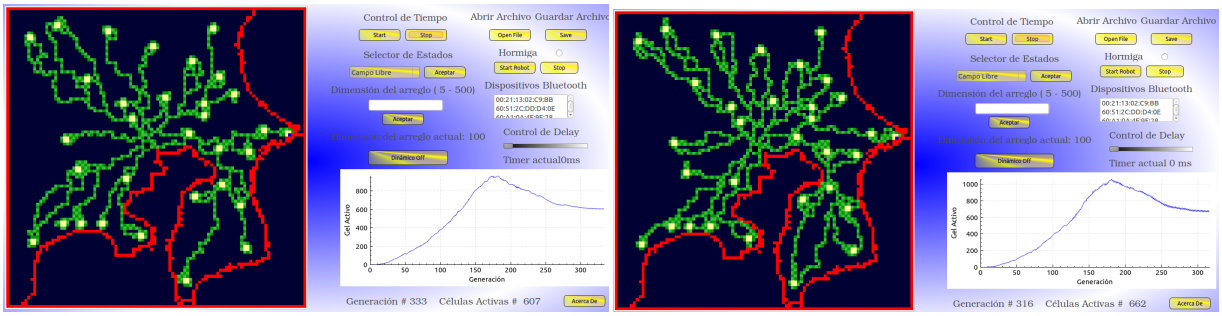


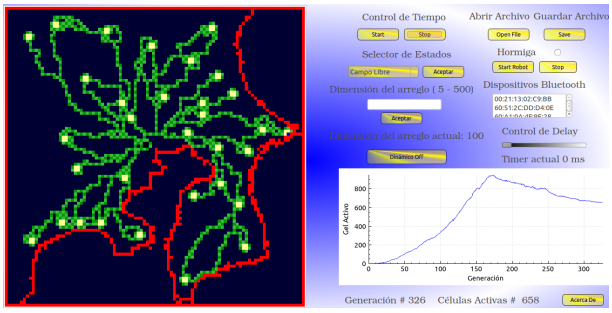
Figura 52: Tokyo en T = 330.

6.3.1. Comparación de 6 simulaciones

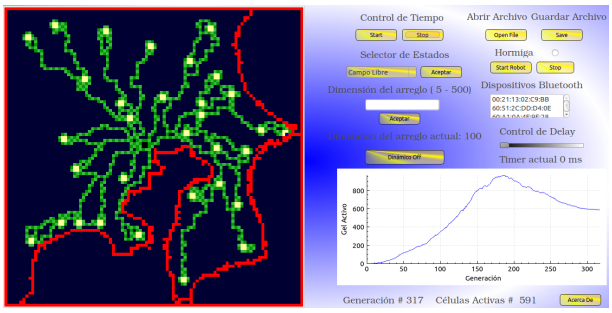


(a) T = 333 con 607 células en el plasmodio.

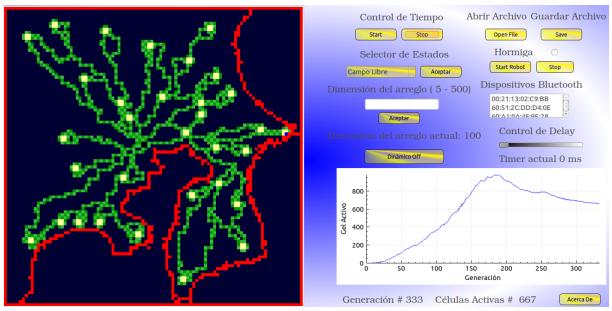
(b) T = 316 con 662 células en el plasmodio.



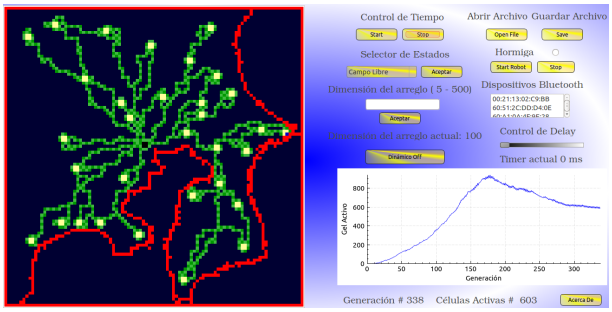
(c) T = 326 con 658 células en el plasmodio.



(d) T = 317 con 591 células en el plasmodio.



(e) T = 333 con 667 células en el plasmodio.



(f) T = 338 con 603 células en el plasmodio.

Figura 53: Simulación de la red de Tokyo a 6 salidas.

El promedio de evoluciones que le toma al AC para la red de Tokyo es de 327 generaciones, para obtener cerca de 631 células que conforman el plasmodio. En las redes se observa que es común ver huecos de diferentes tamaños con las conexiones que realiza entre los nutrientes, como tal no hay una forma que tienda a formar ya que las variaciones del mapa de solución varía demasiado.

6.4. Prueba del simulador con laberinto

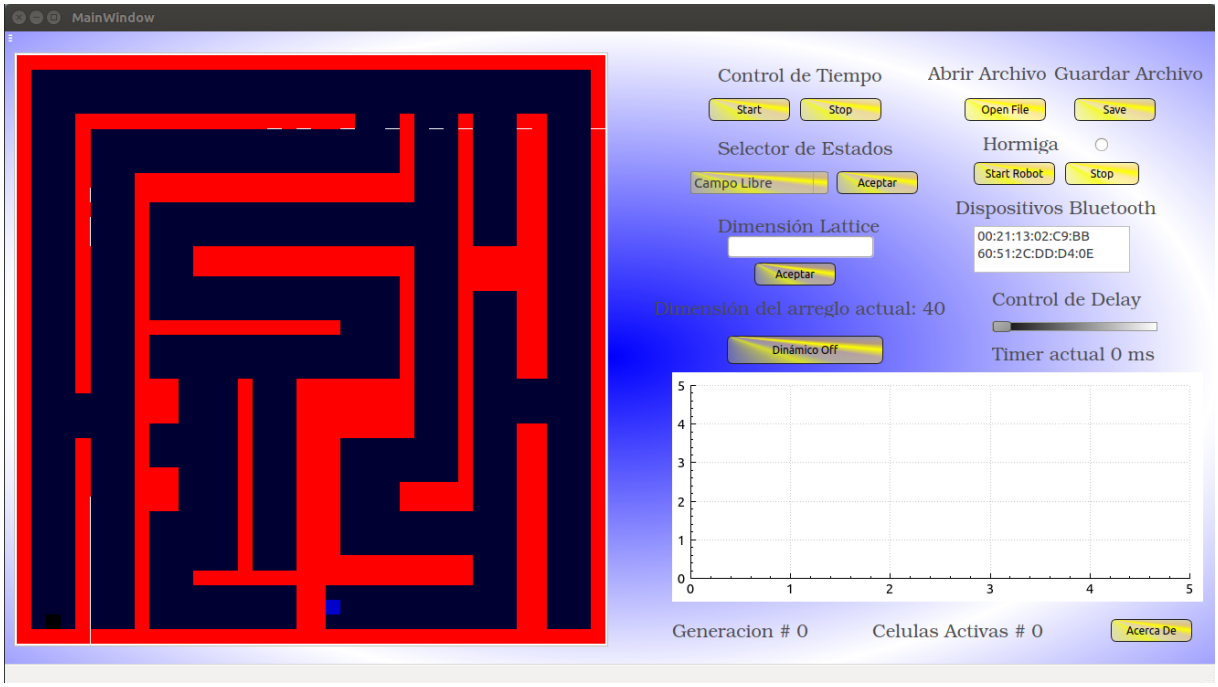


Figura 54: Laberinto en T = 0.

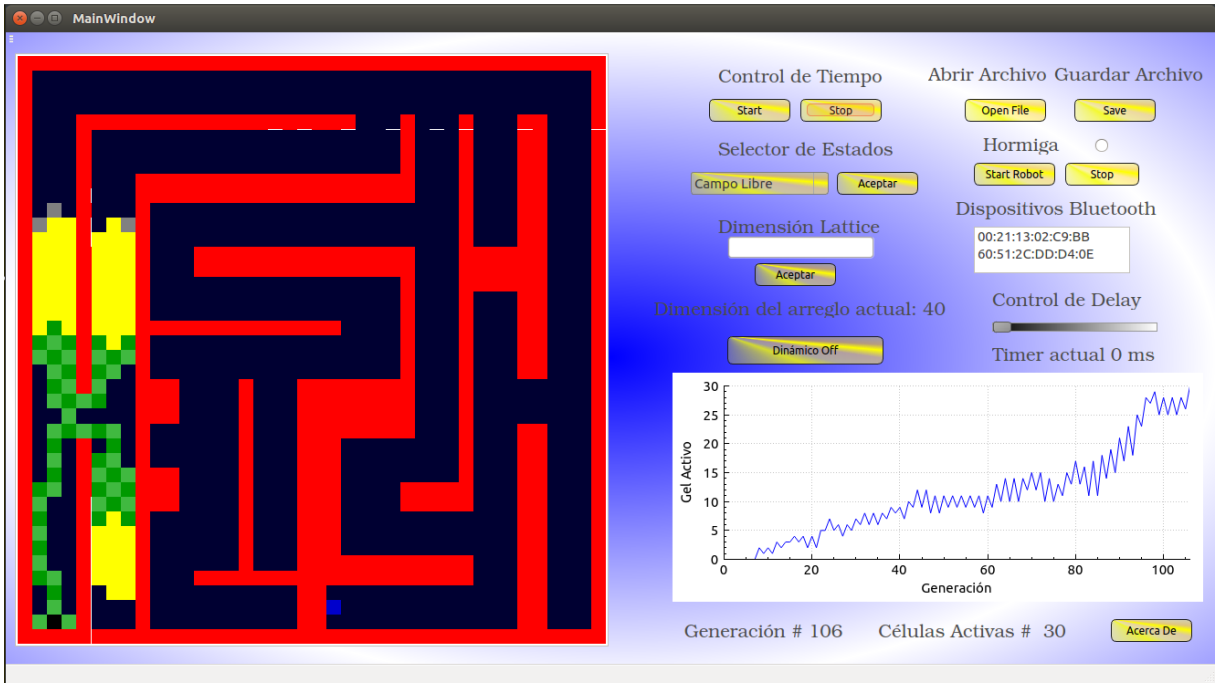


Figura 55: Laberinto en T = 106.

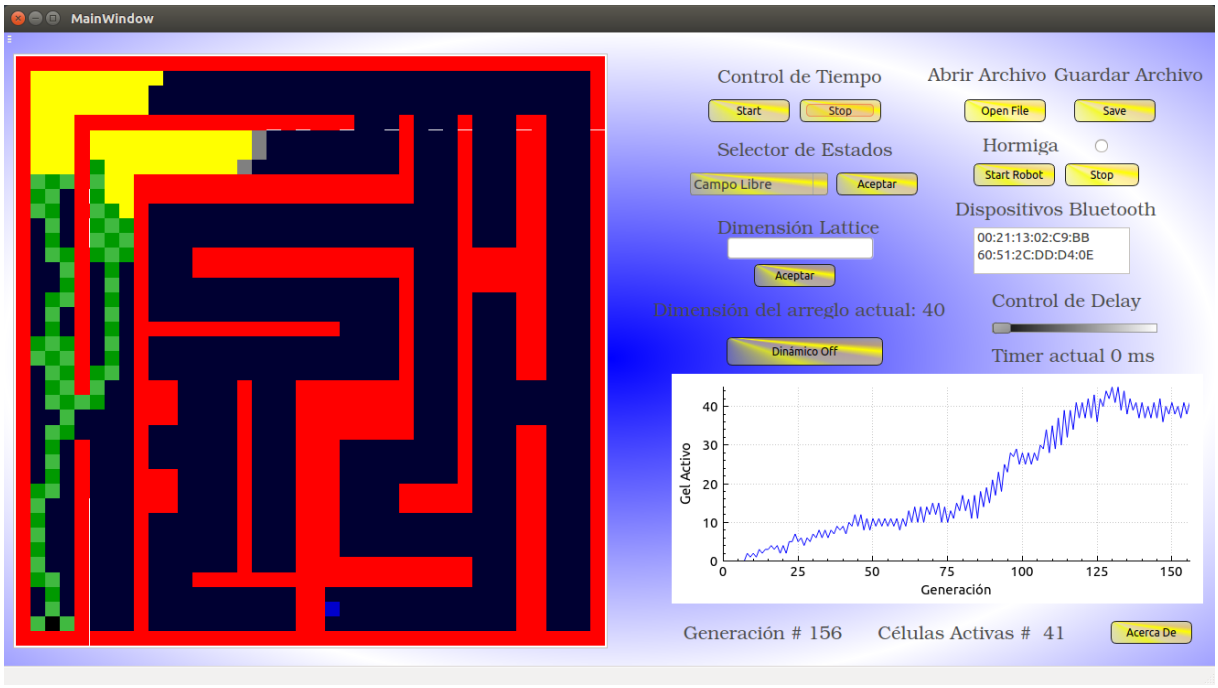


Figura 56: Laberinto en T = 156.

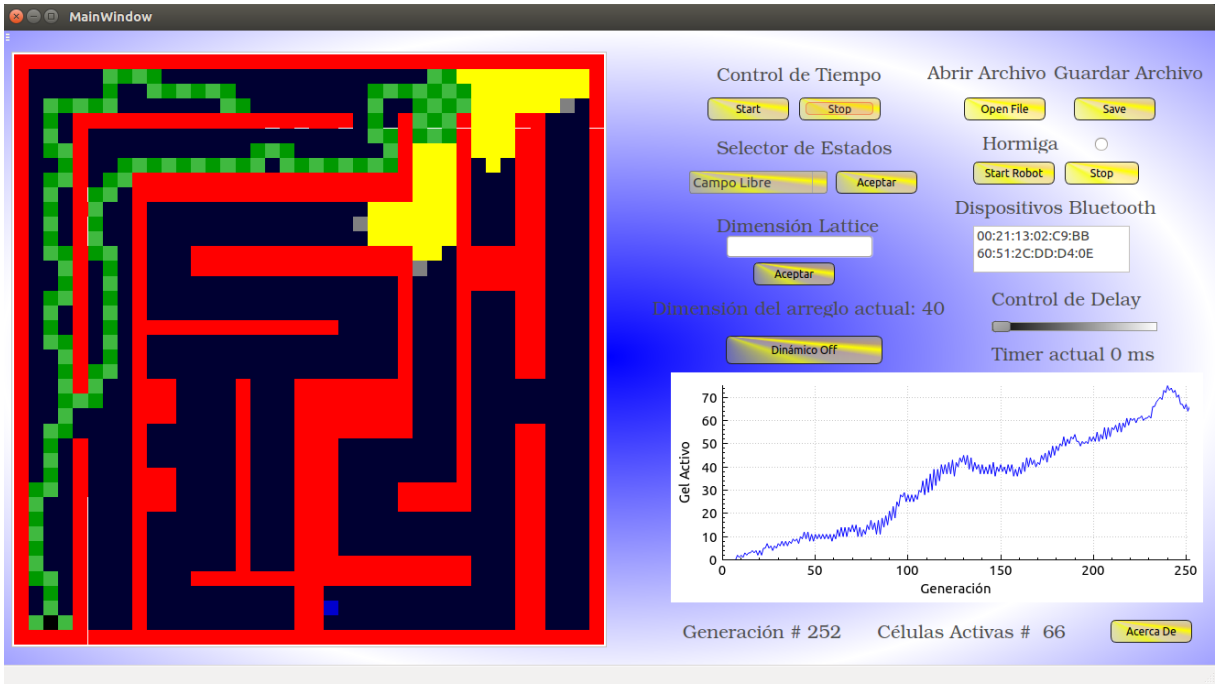


Figura 57: Laberinto en T = 252.

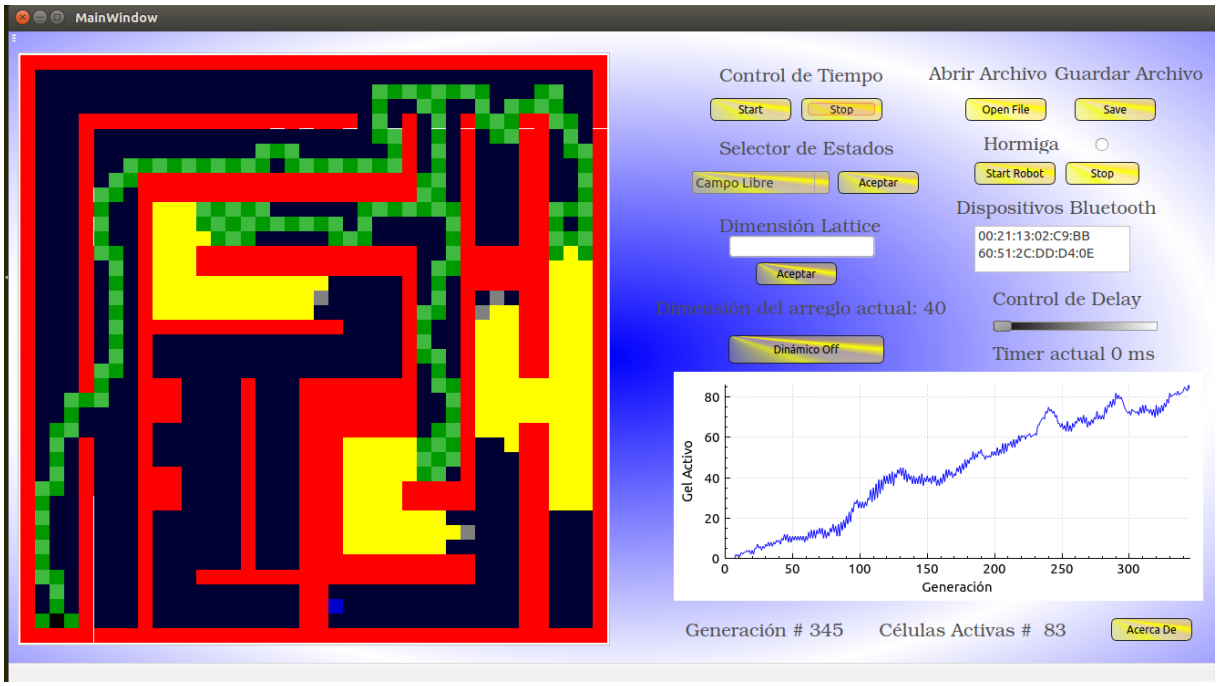


Figura 58: Laberinto en T = 345.

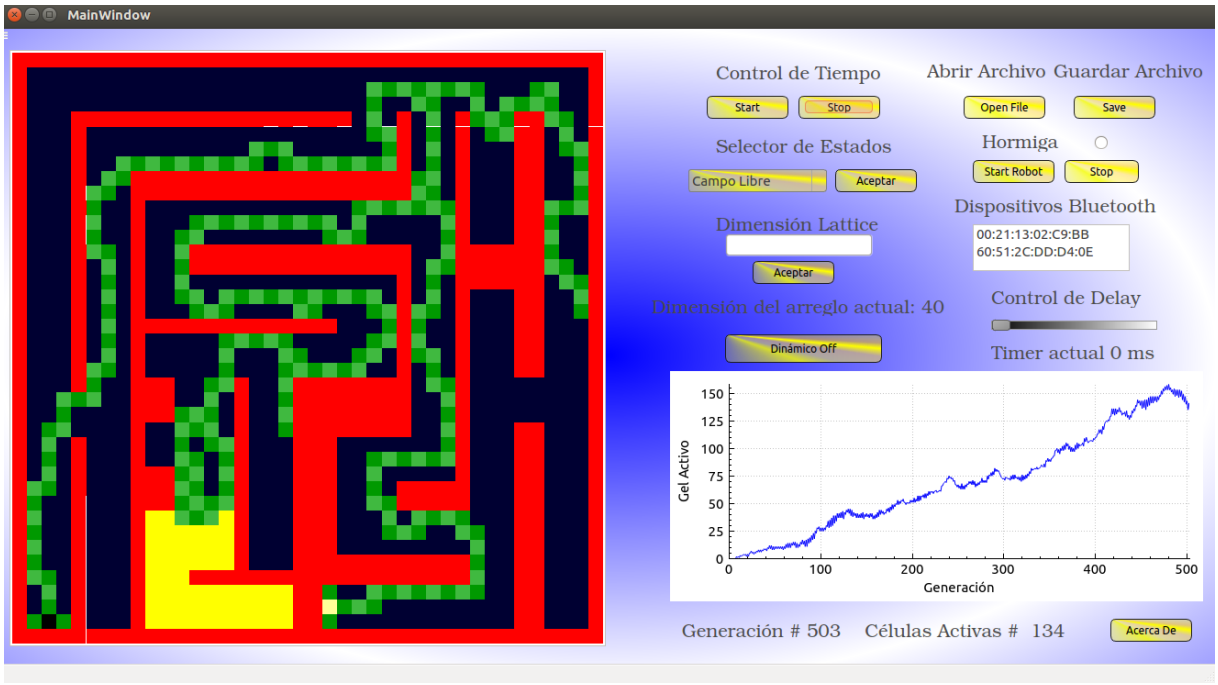


Figura 59: Laberinto en T = 421.

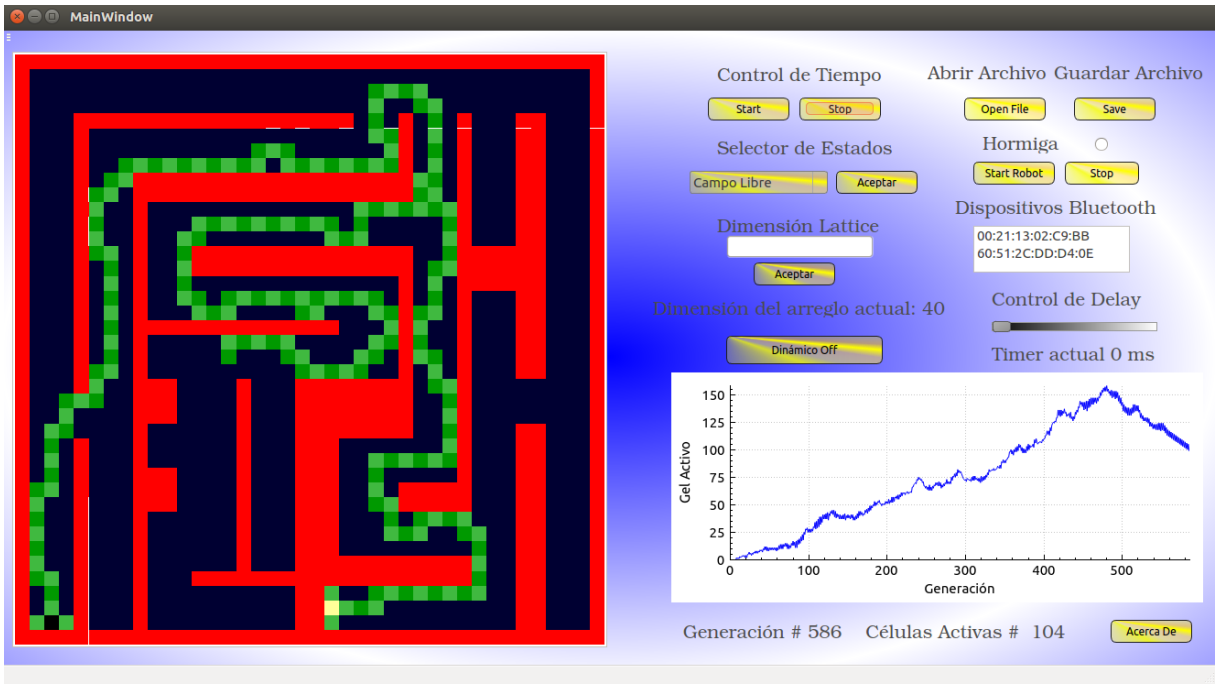


Figura 60: Laberinto en T = 503.

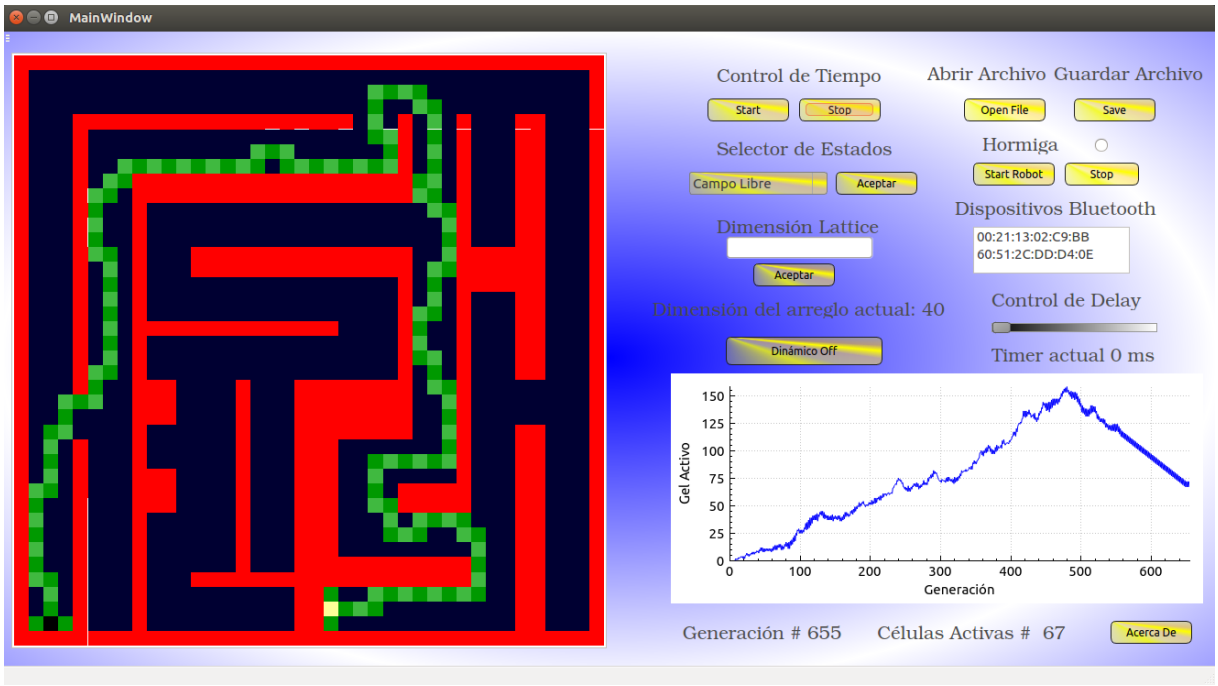


Figura 61: Laberinto en T = 586.

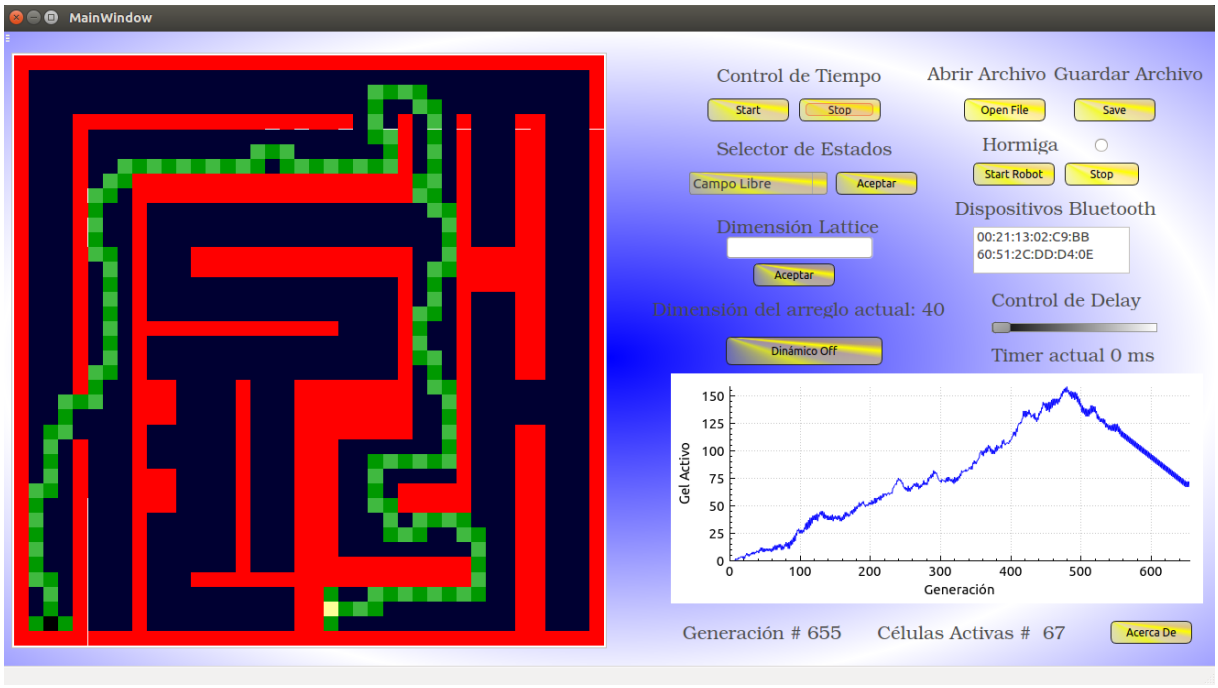
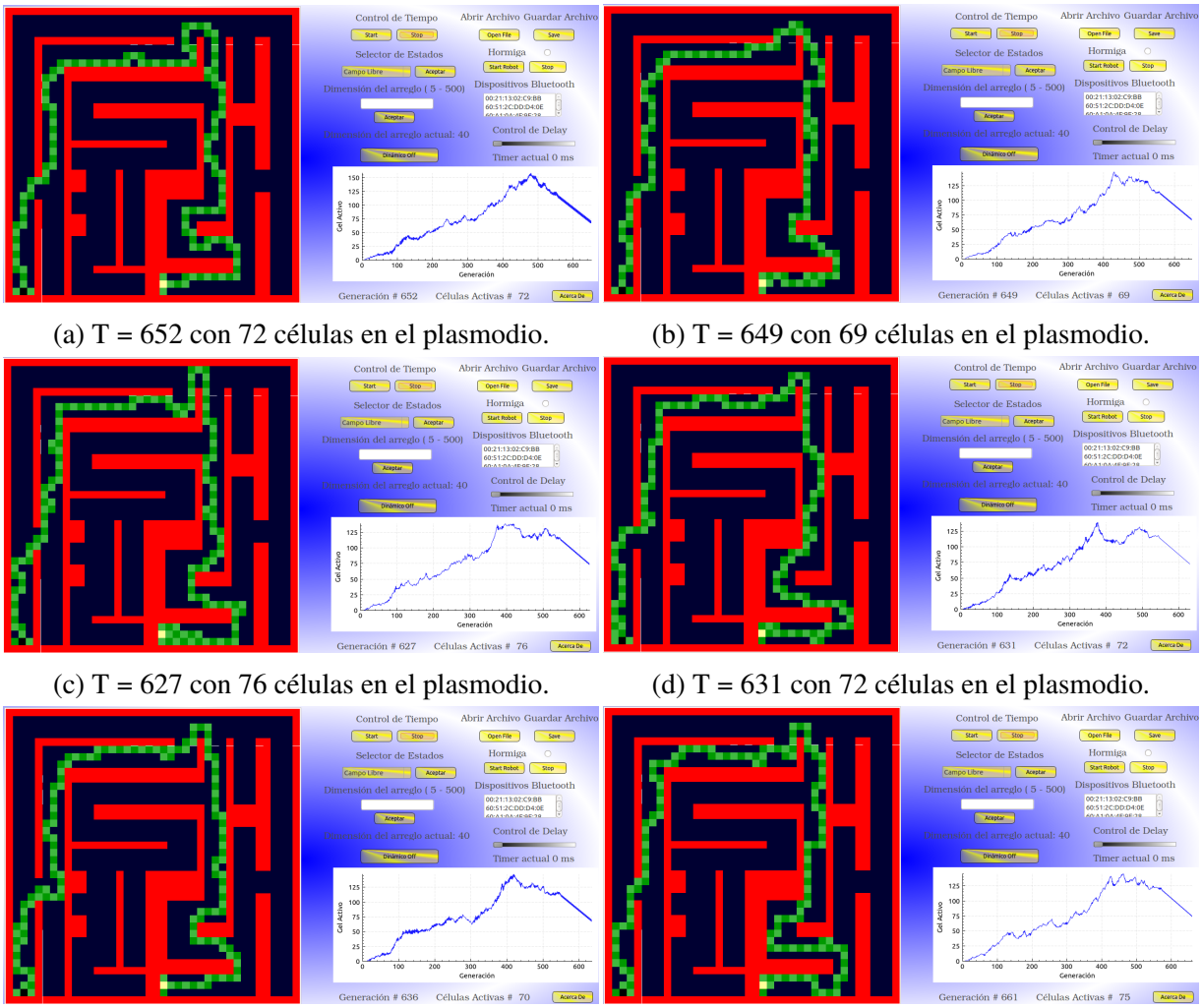


Figura 62: Laberinto en T = 655.

6.4.1. Comparación de 6 simulaciones



(a) T = 652 con 72 células en el plasmodio.

(b) T = 649 con 69 células en el plasmodio.

(c) T = 627 con 76 células en el plasmodio.

(d) T = 631 con 72 células en el plasmodio.

(e) T = 636 con 70 células en el plasmodio.

(f) T = 661 con 75 células en el plasmodio.

Figura 63: Simulación del laberinto a 6 salidas

En el promedio del laberinto se toman 642 generaciones para llegar al ruteo, con cerca de 72 células en la red. Para las salidas no hay demasiada diferencia, sin embargo el Physarum puede a tender a llegar a emplear la ruta norte y cambiar su dirección para obtener otra solución que para estas impresiones, no fue así.

6.5. Prueba del simulador con ejemplo de Jeff

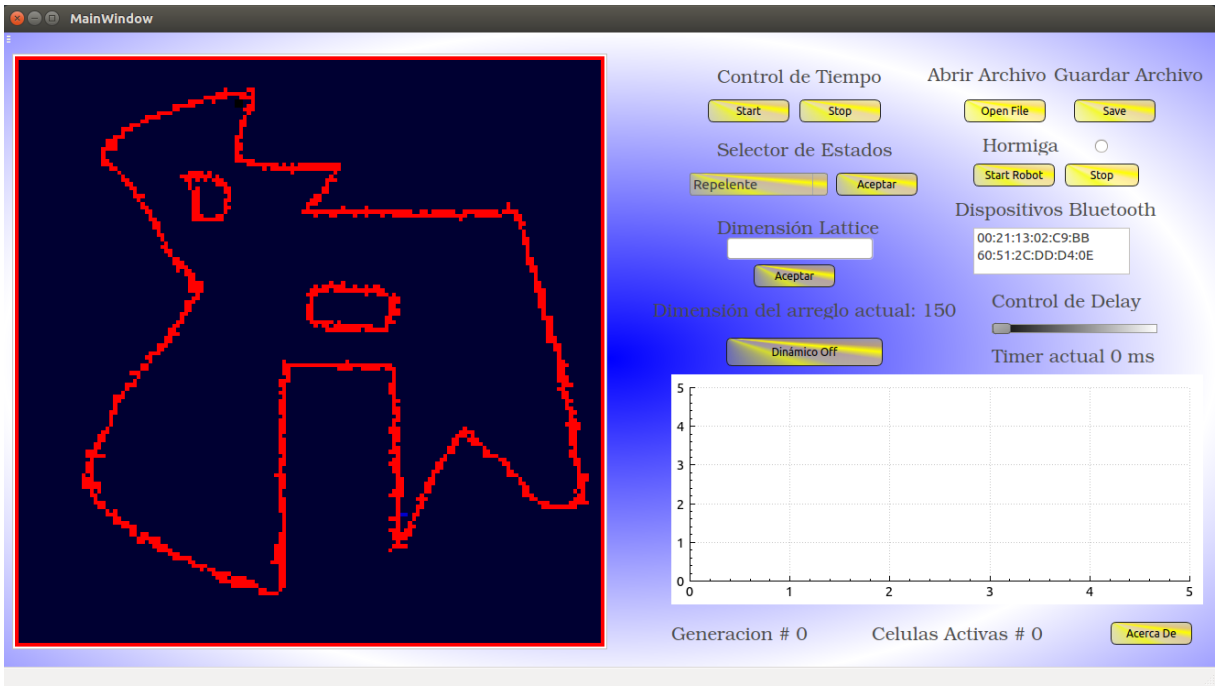


Figura 64: Jeff mapa en T = 0.

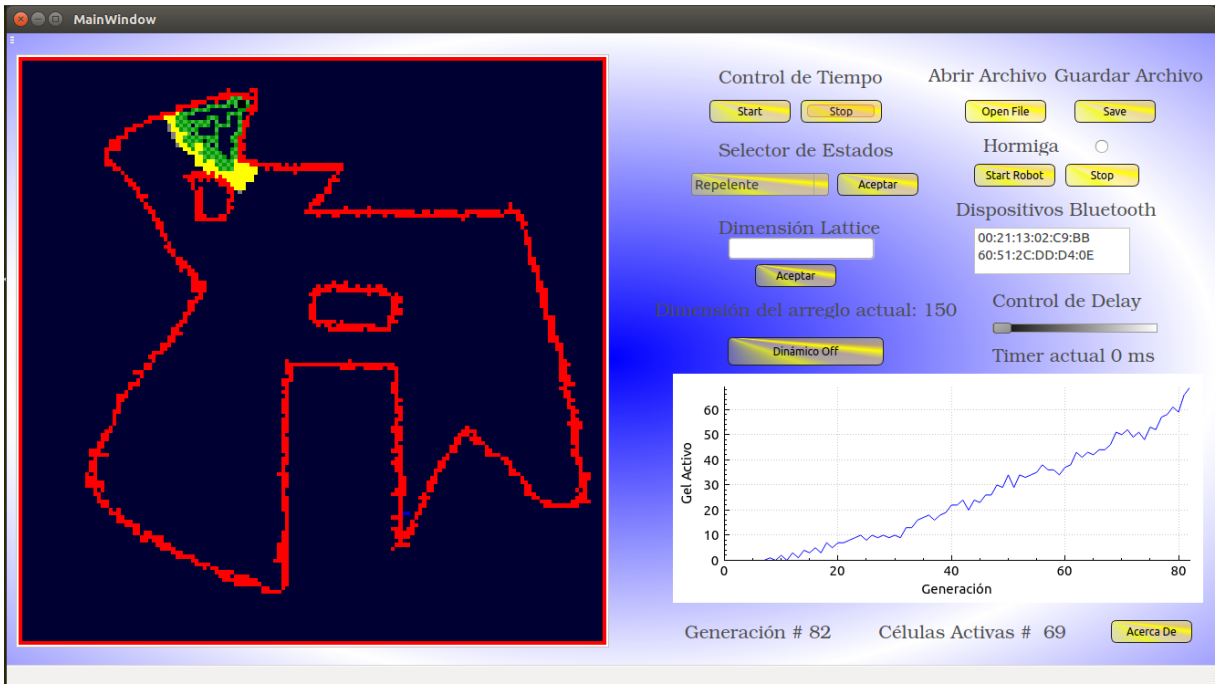


Figura 65: Jeff mapa en T = 82.

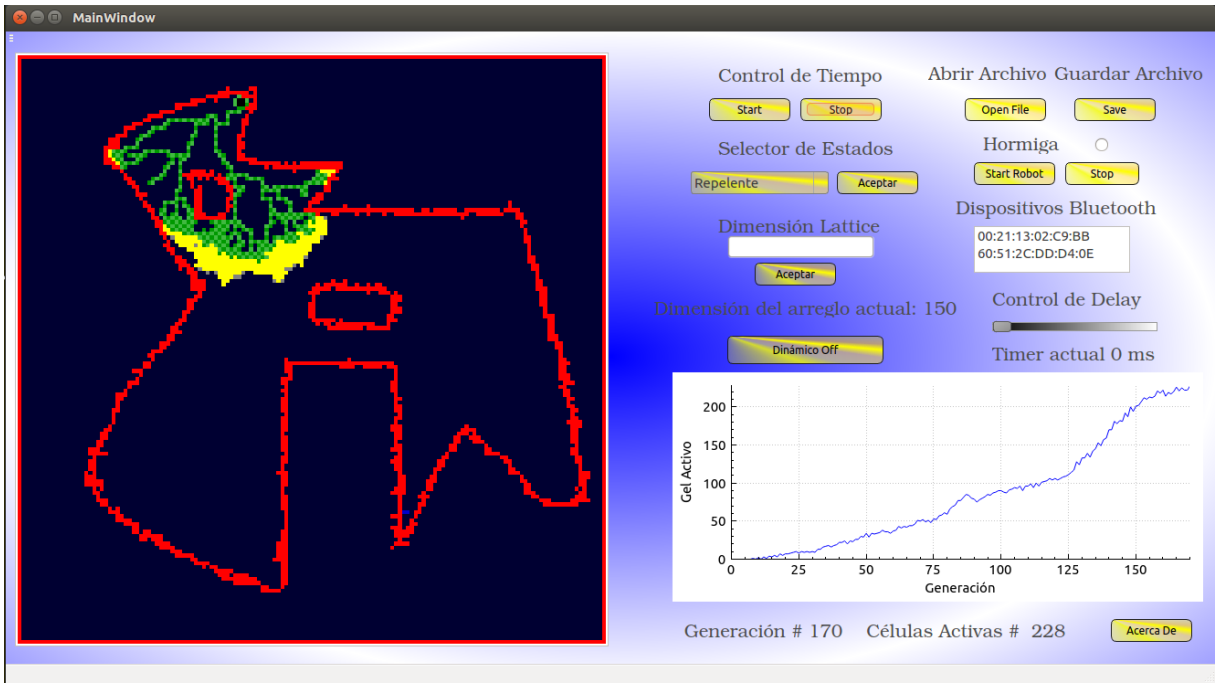


Figura 66: Jeff mapa en T = 170.

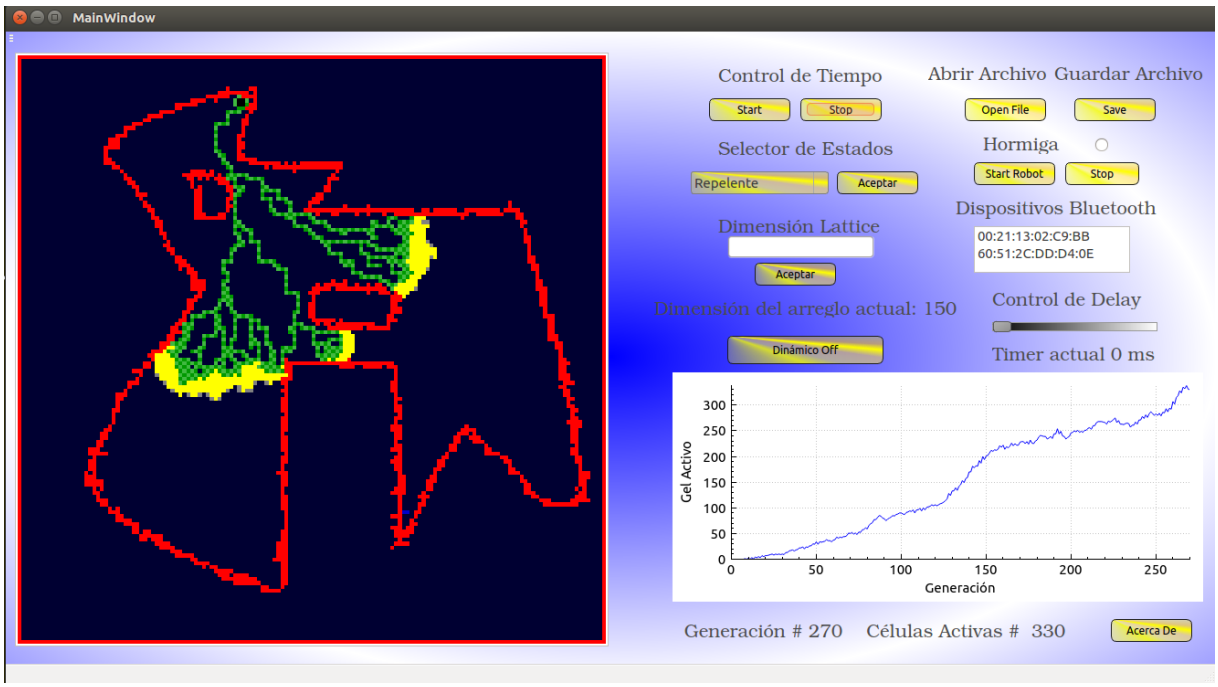


Figura 67: Jeff mapa en T = 270.

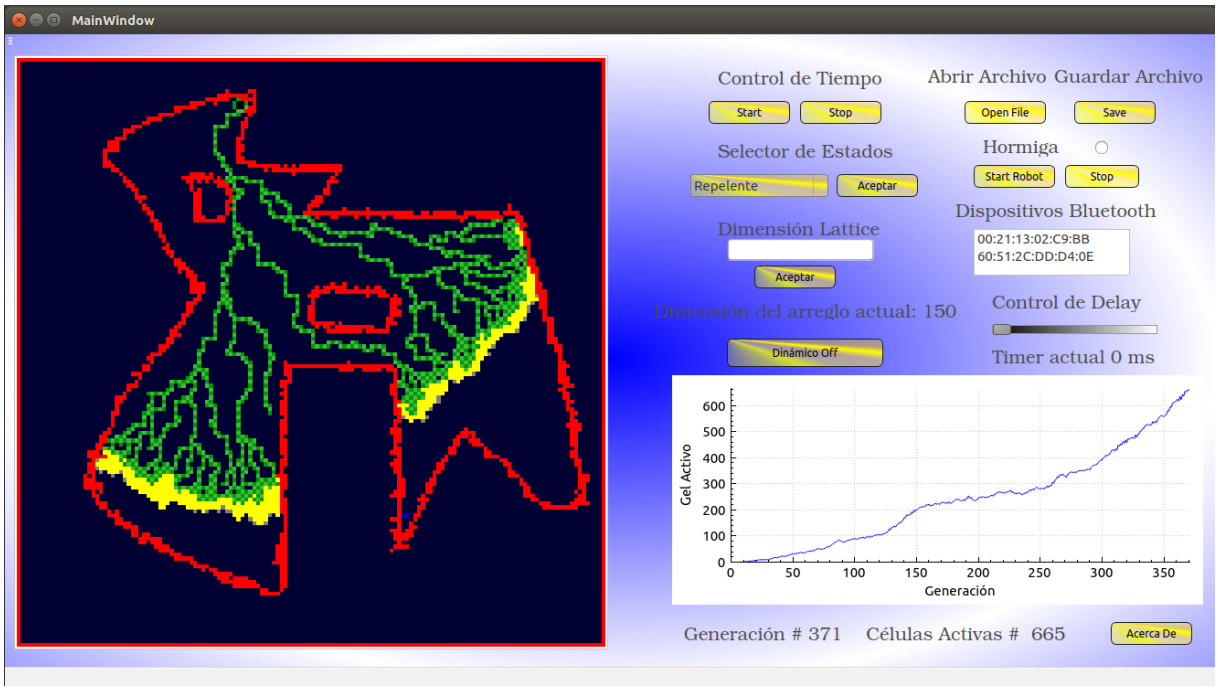


Figura 68: Jeff mapa en T = 371.

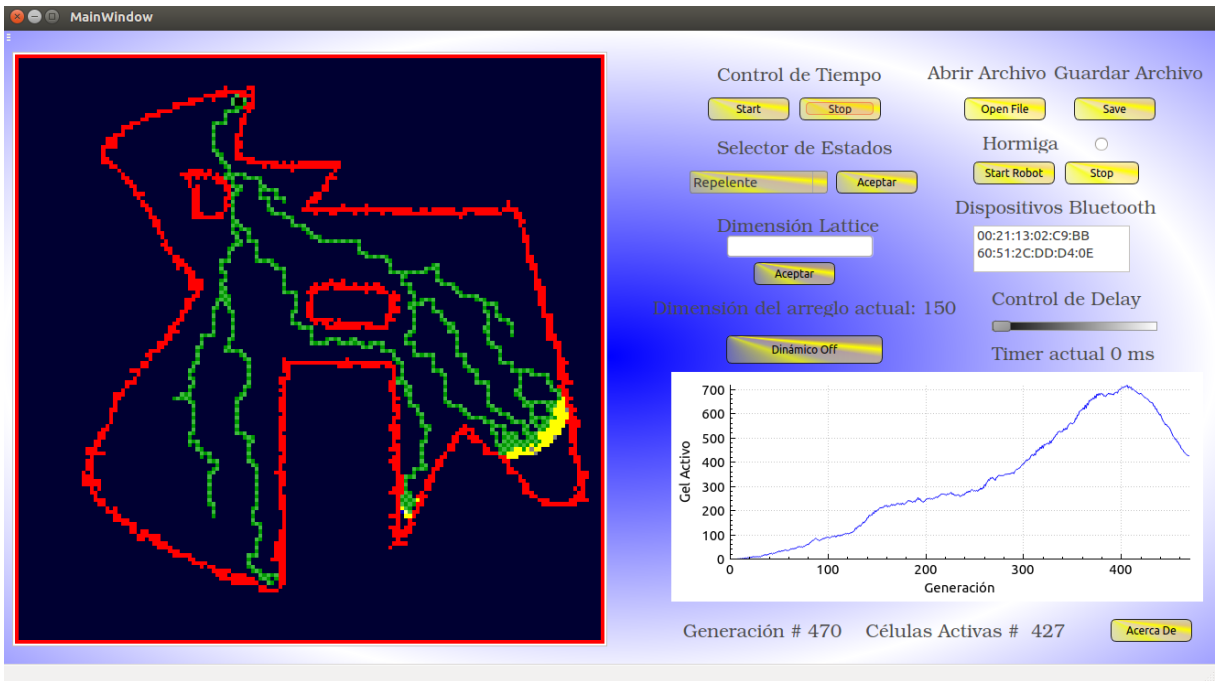


Figura 69: Jeff mapa en T = 470.

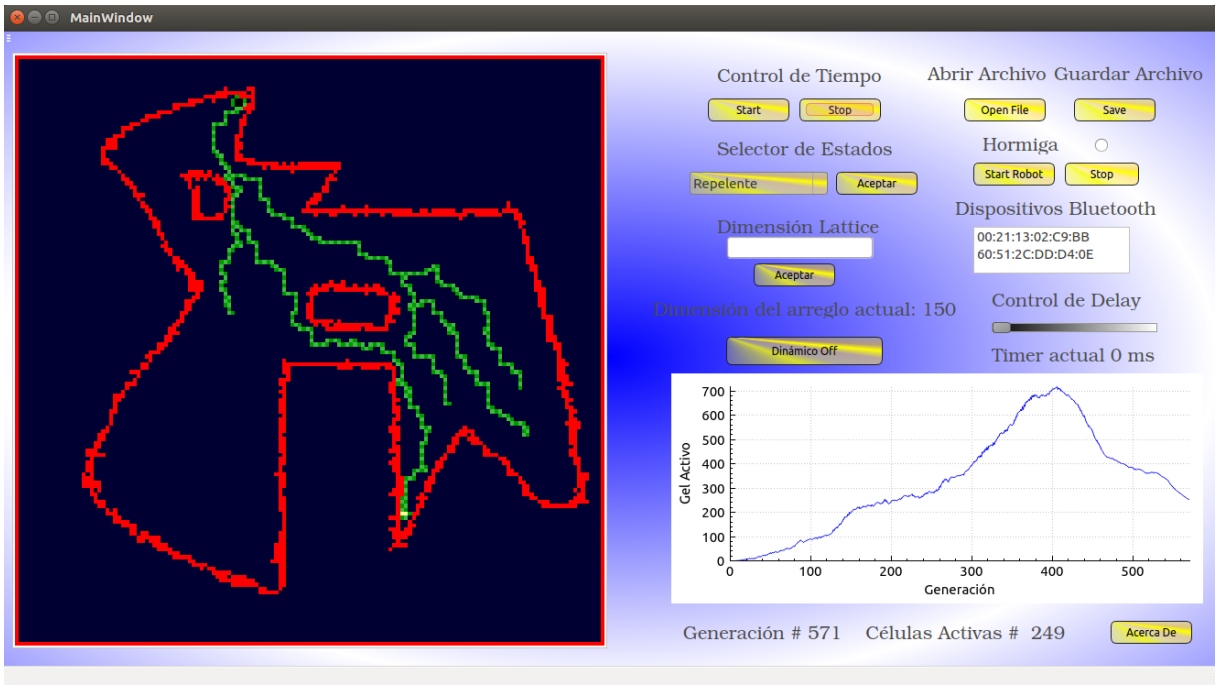


Figura 70: Jeff mapa en T = 571.

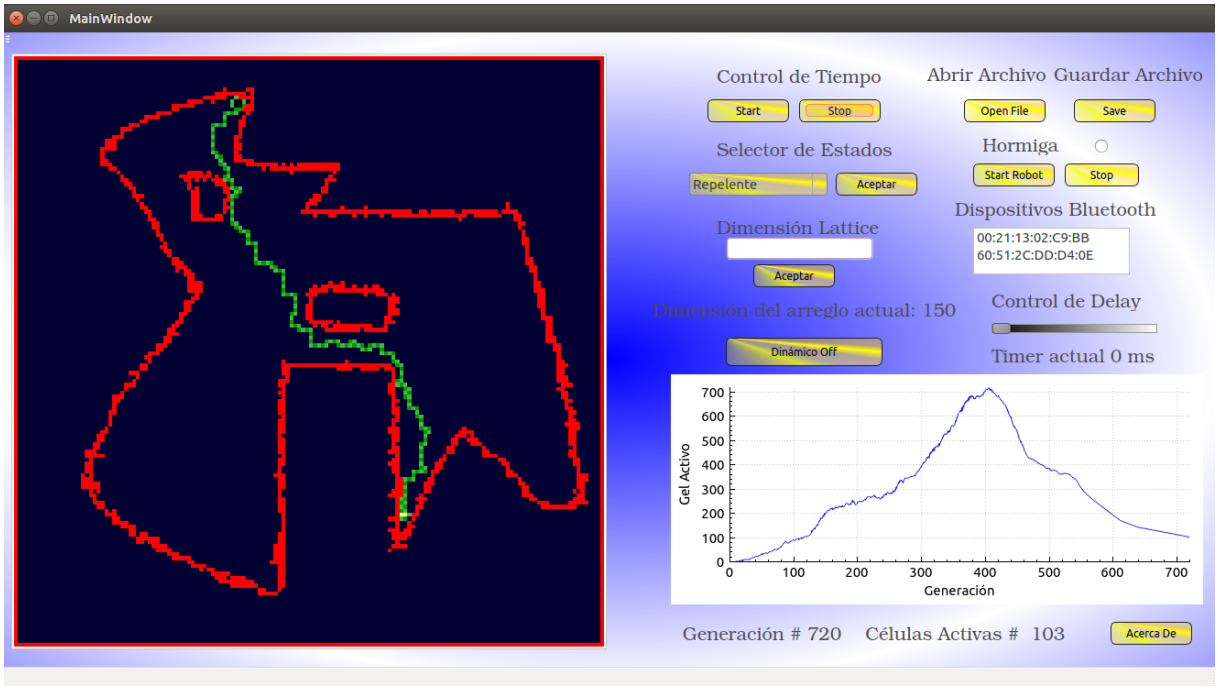
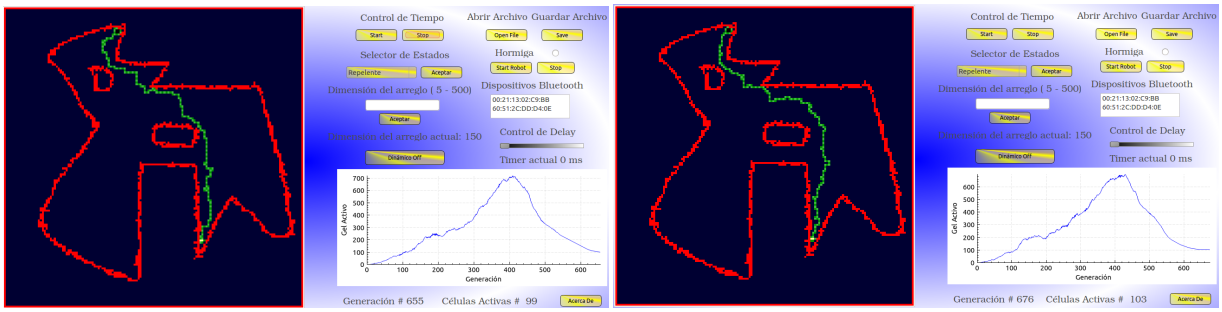
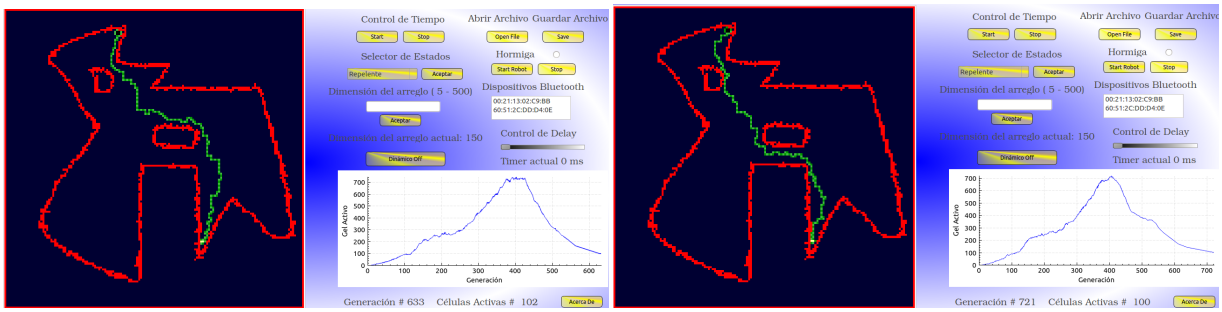


Figura 71: Jeff mapa en T = 720.



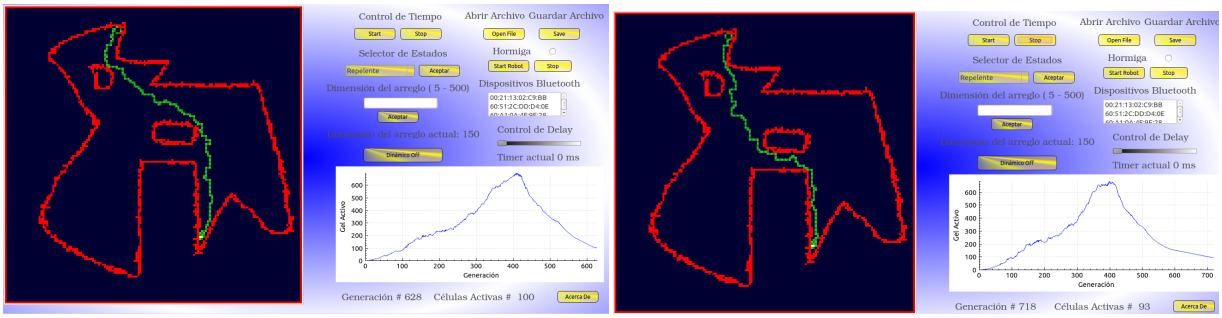
(a) T = 655 con 99 células en el plasmodio.

(b) T = 676 con 103 células en el plasmodio.



(c) T = 633 con 102 células en el plasmodio.

(d) T = 721 con 100 células en el plasmodio.



(e) T = 628 con 100 células en el plasmodio.

(f) T = 728 con 93 células en el plasmodio.

Figura 72: Simulación con un ejemplo de Jeff a 6 salidas.

Para el ejemplo de Jeff son 673 generaciones en promedio que emplea el AC para obtener cerca de 99 células. A pesar de que del mismo modo que el laberinto su mapeo no tiene demasiada variación, para las 6 soluciones, muestra 2 cambios de dirección o decisión en el trazo de la ruta, tanto por arriba del obstáculo y por abajo de este.

6.6. Comparación con otros modelos y el organismo real

Anteriormente se dio mención con respecto a alguno modelos inspirados en el Physarum Polycephalum, en el presente punto vamos a comparar algunos resultados o características no necesariamente sobre tiempo de ejecución, además del organismo real con el fin de observar que tan cercano o lejos se encuentra la propuesta de AC.

6.6.1. Modelo de Adamatzky

El modelo de Andrew Adamatzky [1] se basa principalmente en campos de atracción por parámetros que emplea el organismo con el fin de trazar sus rutas en función de puntos de interés, la vecindad que emplea es la de Moore y en cada paso o generación donde la célula central extrae valores de químico atractor de sus vecinos para saber que tan fuerte o débil es ese punto de interés y con ello redefinir sus rutas.

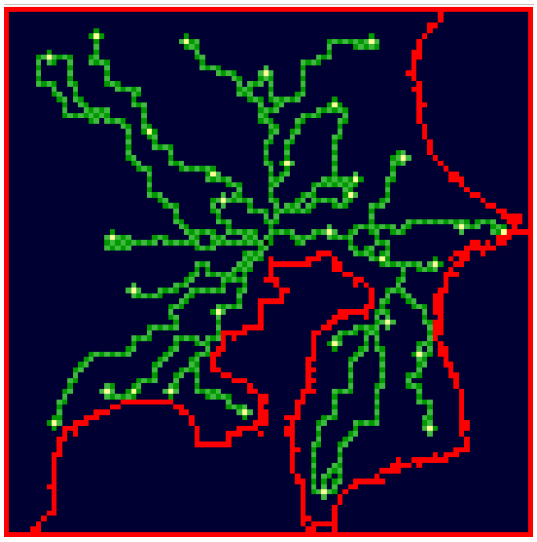
Ventajas

- Algoritmo complejo que gracias al uso de parámetros le da mucha variación no solo a cada nueva corrida, sino que sobre el que se esté trabajando le origina cambios al Physarum.
- A pesar de que emplea dichos parámetros los estados son pocos lo que le causa que sea simple en el aspecto de transiciones siendo 8.

Desventajas

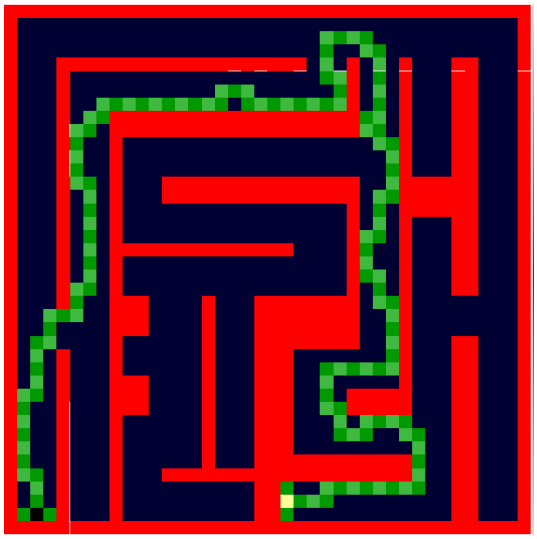
- Las rutas o árbol que crea es de ramificaciones simples, es muy difícilmente ver huecos ó uniones como el modelo real.

Tabla 6: Comparación de la Red de Tokyo Adamazky.

Red de Tokyo	
Modelo de Adamatzky [6]	2DCAvN(9)
	

Se puede observar que las ramificaciones de la salida de Adamatzky no crea algún hueco o los nutrientes sean unidos por varias de ellas, por el cual si existe mucha variación entre ambas redes recordando que el organismo se comporta de ese modo, dependiendo de la exploración que realice, la solución caera dentro de un espectro de soluciones.

Tabla 7: Comparación de solución de laberinto Adamatzky.

Laberinto	
Modelo de Adamatzky [2]	2DCAvN(9)
	

Para este caso no existe mucha variación entre ambas soluciones, debido a que es un espacio de evolución delimitado y no tan abierto como la red de Tokyo, la variación se expone sobre ciertas zonas donde el organismo toma decisión de dirección diferente.

6.6.2. Modelo de Jeff Jones

Jeff Jones [15] propone un modelo basado en agentes o partículas en el espacio distribuidos aleatoriamente. Este modelo también tiene una idea de atracción donde la idea principal se basa en que los agentes roten y tengan dirección sobre la mayor concentración de atractores, debido al uso de sensores como se puede observar en la Figura 73, que pueden escanear su entorno para saber como posicionarse y con ello empezar a desplazarse hacia los campos que tengan mayor fuerza.

Ventajas

- Alto grado de evolución debido al empleo de los atractantes y la interacción de los agentes en el entorno.
- Son apreciables los cambios de retracción o desarrollo de huecos.

Desventajas

- Necesariamente debe existir una población inicial considerable para poder actuar

- A pesar de los cambios que puede realizar el modelo, es distante a ciertas características del organismo real como la exploración del entorno.

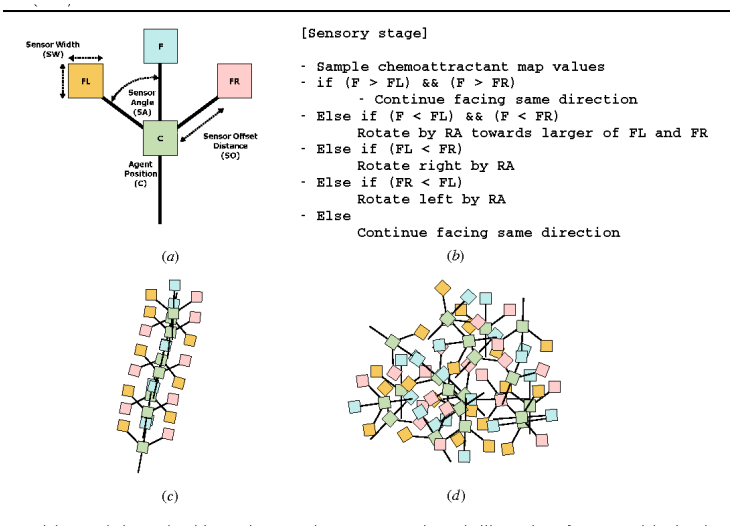
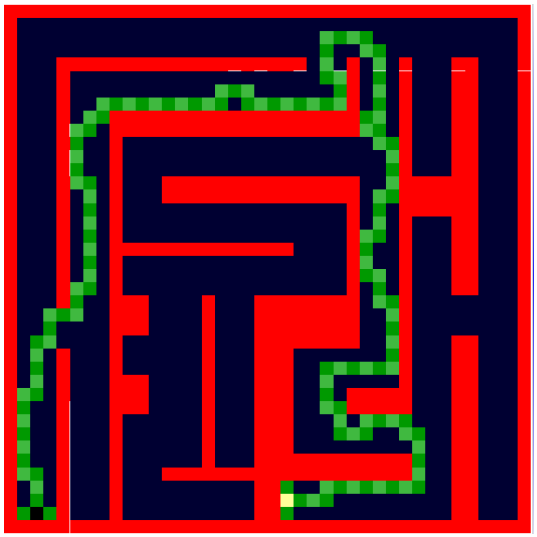


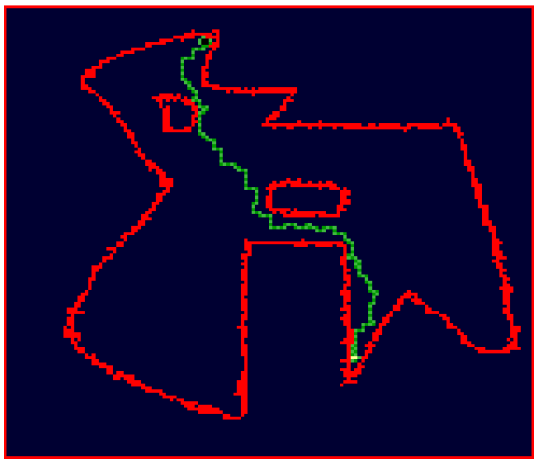
Figura 73: Representación del agente de acuerdo al modelo de Jeff [6].

Tabla 8: Comparación de solución de laberinto Jeff.

Laberinto	
Modelo de Jeff [16]	2DCAvN(9)
	

Para la salida de Jeff, la conexión de las soluciones posibles debido a los propios sensores de los agentes que provocan que se sitúen dentro de una unión de soluciones.

Tabla 9: Comparación de obstáculos Jeff.

Experimento de Jeff	
Modelo de Jeff [17]	2DCAvN(9)
	

El cúmulo de los agentes bajo una línea de solución que a primer viste se ve uniforme, por el contrario el del trabajo terminal existe la traza diferente pero que asemeja a ls misma solución de Jeff siendo no tan uniforme.

6.6.3. Modelo de Gunji

Gunji [24] trata el Physarum como un cúmulo de vecinos donde pueden sobrepasar un cierto umbral para pasar a cierto estado, para su crecimiento inicilamente busca regiones de estímulo donde se propaga poco a poco. El componente principal que emplea lo llama "burbuja", podemos verlo como segregaciones que exploran el entorn,o para saber si sirve o no cierta área la idea es tener un cúmulo de burbujas dentro del Physarum que harán las conexiones un ejemplo lo podemos observar en la Figura 74.

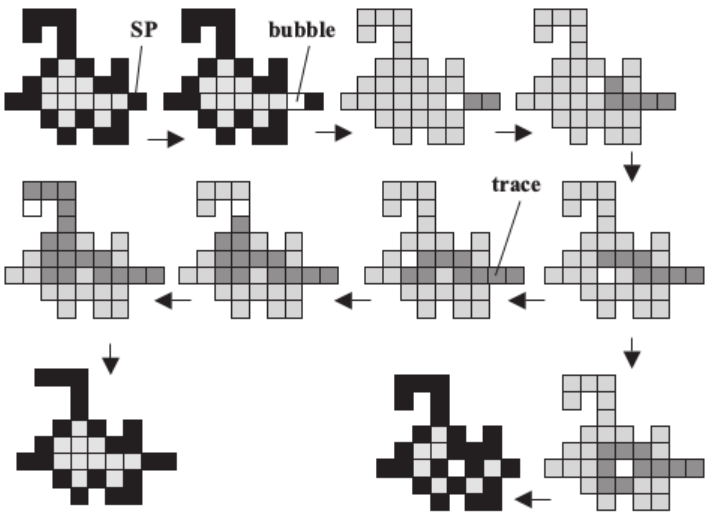


Figura 74: Ejemplo de aplicación del modelo de Gunji [24].

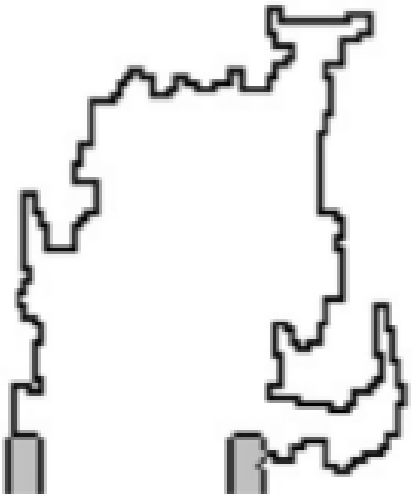
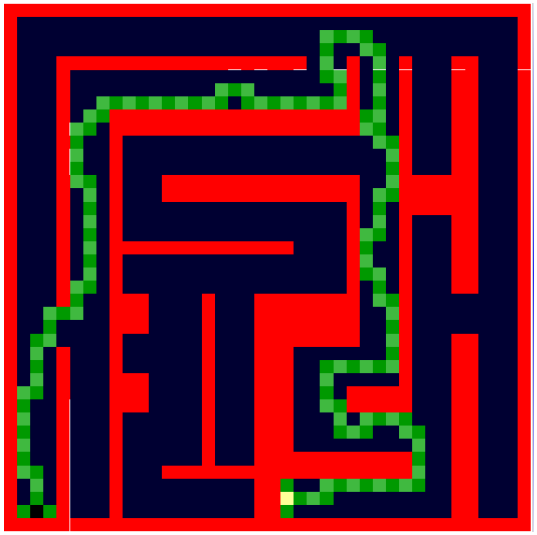
Ventajas

- Reglas simples.
- Solo se necesitan fuentes donde pueda insertar las burbujas y no necesariamente puntos iniciales o finales.

Desventajas

- No se puede apreciar el crecimiento o repulsión dentro del mapa.

Tabla 10: Comparación de solución de laberinto Gunji.

Laberinto	
Módelo de Gunji [25]	2DCAvN(9)
	

Para Gunji a pesar de que de igual modo existe variación con respecto a las decisiones de mapeo que va trazando cada punto de la red, es el impacto que tiene sobre su zona de evolución del cual desaparece al terminar su evaluación.

6.6.4. Resumen de los modelos

Como observaciones para la tabla, el modelo de Jeff como tal es difícil de comparar contra los modelos de AC debido a que se comportan de manera diferente al caer en estados, sino depende de los campos de atracción de las zonas donde los elementos se van moviendo en el espacio. Para el caso de los autómatas celulares, Adamatzky tiene en especial una función de transición que pareciera llevar a n estados, sin embargo solo toma una intensidad de color sobre un estado dependiendo de la cantidad de concentración del plasmodio, mientras Gunji detalladamente muestra las transiciones que tiene sobre cada estado y que valor pueden tomar.

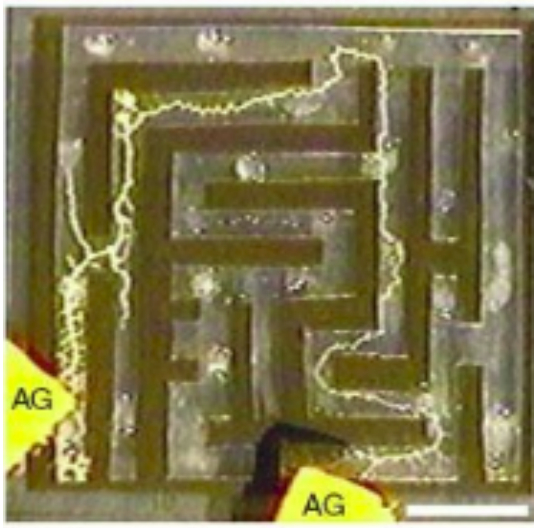
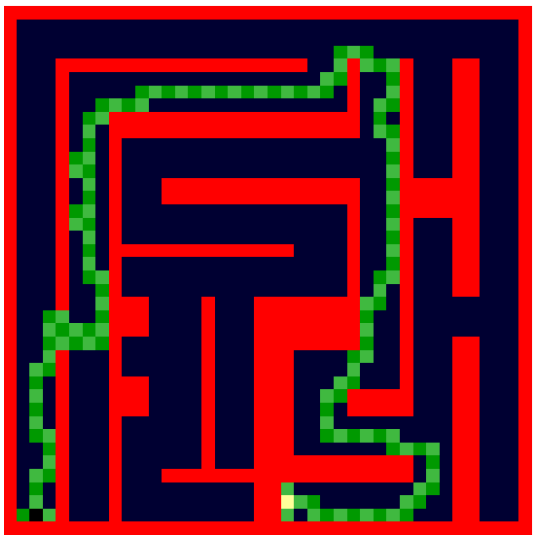
Tabla 11: Resumen de comparaciones de modelos.

Modelo	**Tipo de función**	**Vecindad o conectividad**	**Edos.**	**Zd**	**Numero de trans.**
Adamatzky	Autómata celular	Vecindad de Moore	8	2D	8
Jeff Jones	Agentes	Sensores de radio n	1	2D	-
Gunji	Autómata celular	Vecindad de von Neumann	6	2D	10
2DCAvN(9)	Autómata celular	Vecindad de von Neumann	9	2D	10

6.6.5. Comparación con el modelo real

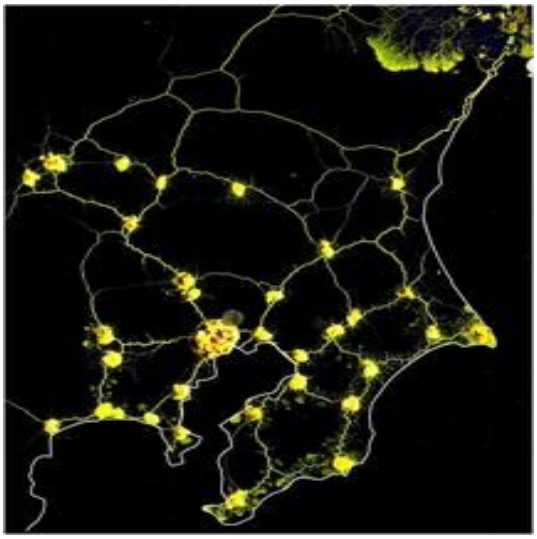
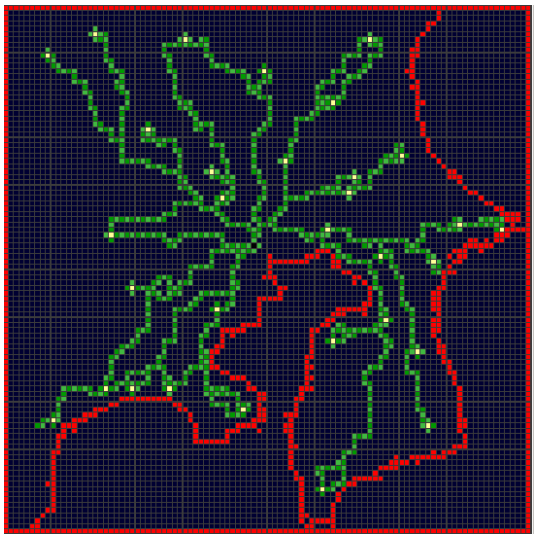
En el siguiente punto, vamos a emplear los experimentos comunes que se aplican al Physarum, recordando que el organismo real se extiende 1 cm cada hora

Tabla 12: Comparación de solución de laberinto real.

Laberinto	
Organismo real [3]	2DCAvN(9)
	

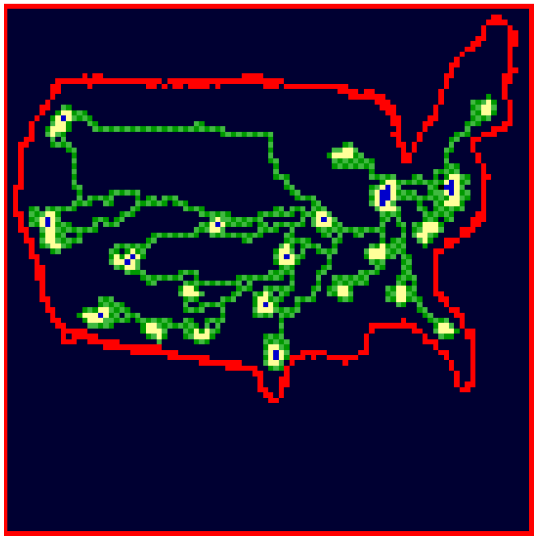
Como dato importante al organismo le tomó 92 horas en realizar el trazo de la red para este experimento, la información de ambos hace ver la simulitud que tiene nuestro modelo con respecto al organismo real, que como observación no siempre el organismo real elegirá las misma solución.

Tabla 13: Comparación de solución de Tokyo.

Red de Tokyo	
Organismo real [21]	2DCAvN(9)
	

Con la red de Tokyo el tiempo fue de 26 horas. Existe mucha diferencia entre ambas redes, propio de la organización que genera tanto el Physarum real y la simulación

Tabla 14: Comparación de solución de USA.

Red de USA	
Organismo real [22]	AC Propuesto
	

En el caso de la Red de USA[7], al organismo real le toma un poco más de 60 horas en realizar la traza de su mapeo.

[7] Veáse en : http://dataphys.org/list/slime-mold-imitates-the-united-states-interstate-system/

6.6.6. Observaciones de nuestro modelo

Después de realizar la comparación con el resto de los modelos junto con el organismo real, se pueden ver algunos puntos importantes acerca de la relevancia de nuestro AC, aunque hay que tener en cuenta que para las comparaciones del cada uno de los modelos de los autores es subjetivo ya que como tal las dimensiones pueden o no ser las mismas con exactitud, sin embargo para fines de la red que trazan sirven como referencia para ver la cercanía con respecto a cada uno.

Ventajas

- Las etapas de crecimiento y retracción se ven en conjunto pues que Physarum en la vida real hace ambas al mismo tiempo, conforme se va expandiendo va discrimando cuales rutas le sirven.
- Algo que deseamos para el AC que por cada corrida, arroje diferentes resultados o soluciones debido a que se busca eso, que no tienda a aspectos predecibles.
- Como nosotros nos concentramos en la propiedad de como el Physarum realiza sus mapeos, ocasiona que la cantidad de evoluciones necesarias para llegar a un resultado sean pocas.

Desventajas

- Una limitante del modelo es el no poder observar cambios en sus rutas debido a las reglas que emplea.
- Los campos de atracción o elementos químico atractores no son vistos en el modelo.

6.7. Prueba piloto del robot

6.7.1. Pruebas de movimiento básicas

En este apartado observaremos cómo el robot realizá los movimientos que va a emplear para su comportamiento donde por el momento se pueden ver un poco los desfases que tienen sobre cada uno, es decir no del todo los realizó como deben de ser sin embargo, como iniciativa se acerca a lo requerido hasta el momento.

Tabla 15: Rotación a la izquierda.

Rotación a la izquierda	
Posición Inicial	Posición Final

Tabla 16: Rotación a la derecha.

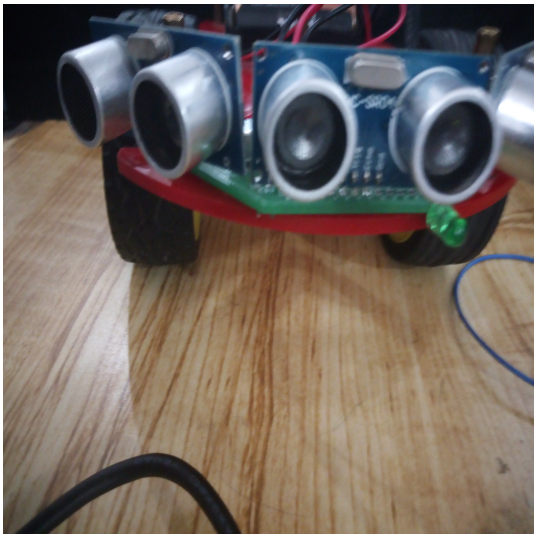
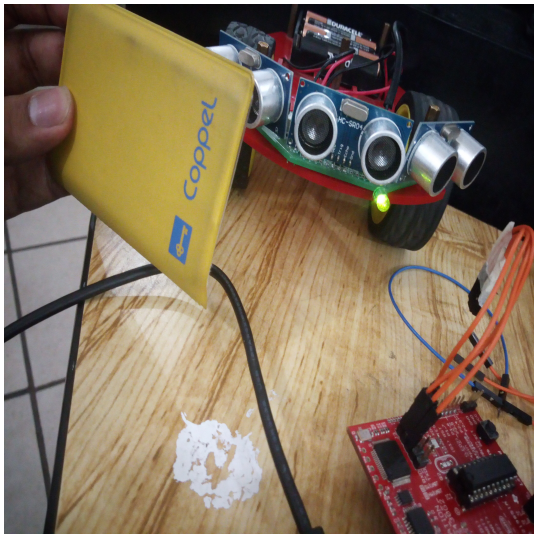
Rotación a la derecha	
Posición Inicial	Posición Final

Tabla 17: Avanzar.

Avanzar	
Posición Inicial	Posición Final

6.7.2. Prueba del sensor de proximidad

Tabla 18: Sensor de Proximidad.

Sensor de Proximidad	
Sin objeto	Con objeto
	

6.7.3. Observaciones o problemáticas presentadas

- Originalmente el robot puede alimentarse con pilas recargables de 1.2 V, sin embargo al emplear el modulo del Bluetooth es necesario utilizar mínimo unas de 1.5 V, aunque otra opción es adaptar otra pila extra de 1.2, debido a que son necesarios 5V para el HC-05.

- Al juntar la parte del sensor de proximidad y el del Bluetooth existe un conflicto donde no establece el correcto funcionamiento del sensor de proximidad.
- El almacenamiento del arreglo se pensó de esta manera debido a que si se usa en un mapa muy grande donde el rango del bluetooth no pueda llegar, se perderá la conexión para las acciones por ello mejor en su punto de partida se le va llenando de las acciones.
- Una limitante es el recibir datos del robot al programa, debido a que el pin Rx ya está ocupado sobre los motores, solo se emplea el Tx.
- Al avanzar el robot tiende a no llevar un camino recto, sino a irse un poco a otras direcciones.

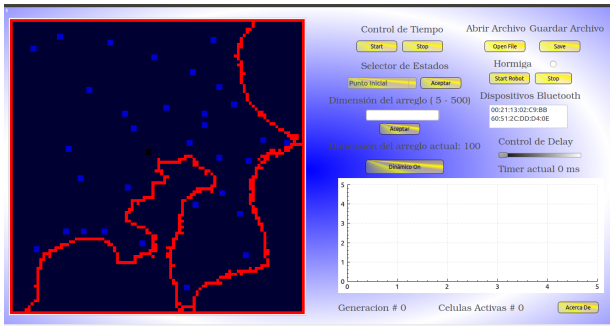
Para los casos anteriores se expone un resumen de solución para los problemas presentados, para trabajar con los 3 sensores es requeridos dentro del código del robot retardos pertinentes para que los muestreos de los sensores se lleven a cabo de manera correcta, de lo contrario eso causó el mal funcionamiento sobre el Bluetooth y los sensores. Los sensores tienen dos utilidades.

1. Primeramente el reconocer obstáculos dentro de la red de solución que pueden o no estar presentes al momento de que el robot empiece a avanzar.
2. Para corregir el problema de no avanzar recto, pensando en que la solución que nos arroja la simulación es ideal y no debe haber razón que se eligan caminos por donde no debe pasar o exista alguna pared, sin embargo a causa de este problema los sensores tienen el objetivo de acomodar el robot en caso de que pegue con alguna pared o dirección que no deba ir el robot.

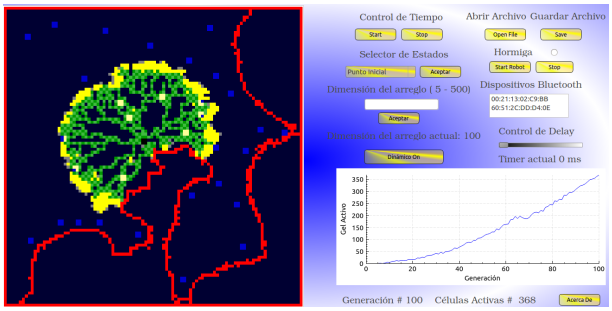
Otra de la solución para la dirección es emplear PWM, la justificación es que los motores conforme al paso de uso tienen un desgaste que provoca el mal funcionamiento de estos mismos provocando que no se detengan cuando deben, con el PWM provoca que si algun motor vaya más o menos rápido que el otro, se acoplen ambos de acuerdo a la salida solicitada.

6.8. Evolución continua del Physarum

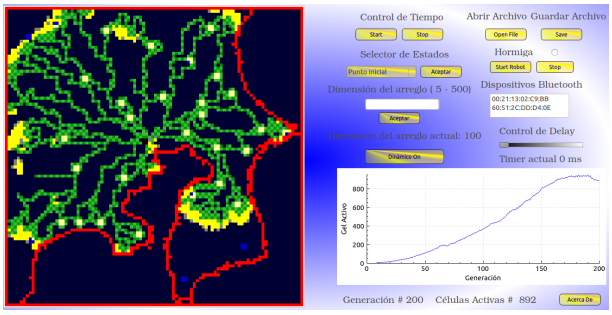
Para esta opción el simulador brinda una opción donde el Physarum puede seguir explorando a pesar de que la simulación obtenga una red solución. El comportamiento que brinda altera la regla para el estado q_4 omitiendo la evaluación para pasar al estado q_5, en este caso pasa directo a dicho estado provocando que el organismo siga invadiendo sin importar la condición. Para poder observar el comportamiento dentro del simulador, se emplea el botón de Dinámico que ofrece la aplicación que una vez activado, solamente es correr la simulación de manera normal.



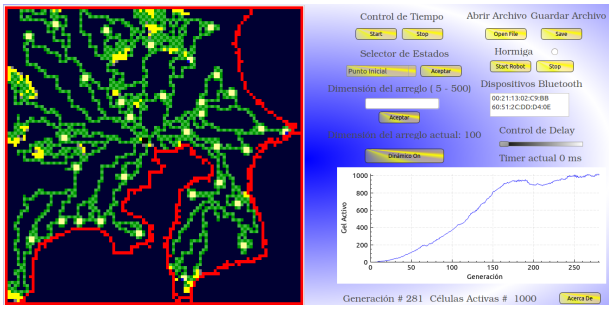
(a) Configuración inicial Red de Tokyo.



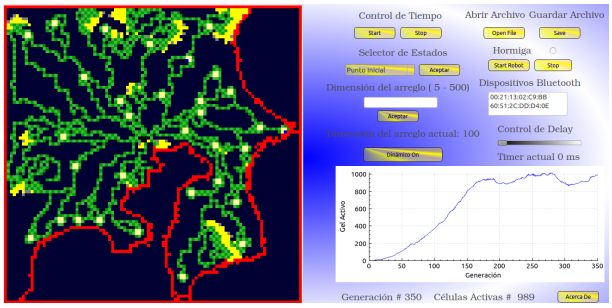
(b) T = 100 con 368 células en el plasmodio.



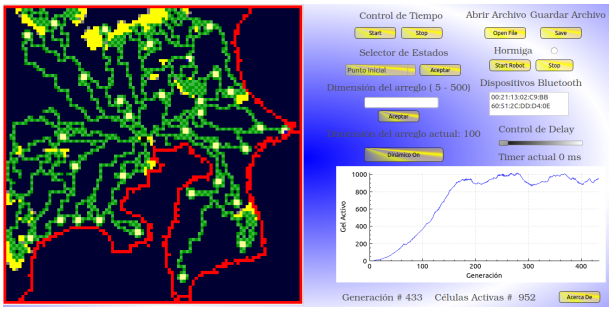
(c) T = 200 con 892 células en el plasmodio.



(d) T = 281 con 1000 células en el plasmodio.



(e) T = 350 con 989 células en el plasmodio.



(f) T = 433 con 952 células en el plasmodio.

Figura 75: Simulación de la red de Tokyo con exploración continua del Physarum

Como se puede observar a diferencia de los experimentos anteriores, la simulación no arroja una solución donde la configuración sea fija, por el contrario, sigue realizando la exploración siempre y cuando la evaluación del autómata le permita crear los estados requeridos que en el caso de los huecos no se observa que siga explorando.

6.9. Prueba con maqueta

Para la maqueta se realizó en una dimensión de 144 x 116, ajustando cada celda de unos 18 cm aproximadamente para representar dentro de la simulación. Primeramente lo que se realiza es la propia simulación para obtener la solución.

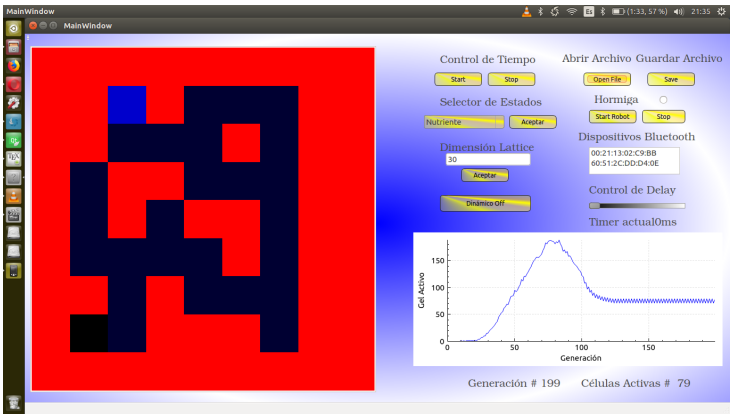


Figura 76: Configuración inicial.

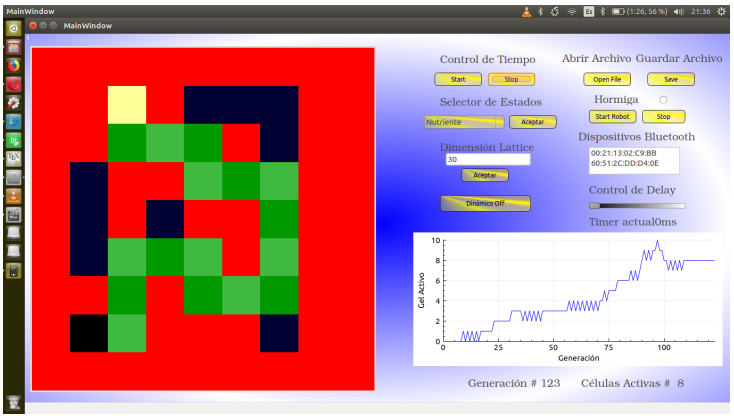


Figura 77: Configuración final del AC.

Al terminar se inserta la representación del robot dentro del simulador, para después conectarlo al eBot, una vez hecho se empiezan a mandar las acciones por medio del Bluetooth con base a la trayectoria que empiece a trazar el punto blanco o representación del robot, para llegar a un momento de dejar de mandar comandos.

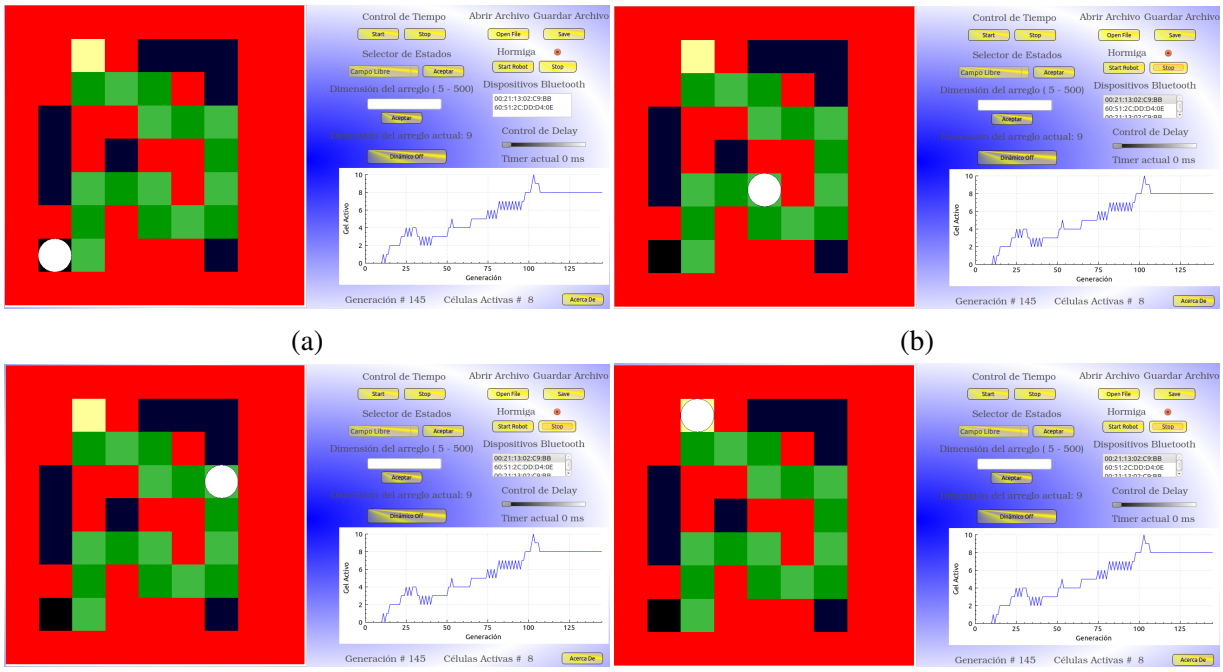


Figura 78: Trayectoria del robot.

Si lo queremos expresar por medio de la expresión regular antes mencionada la cadena que va a emplear será la siguiente.

$$\{A, I, D\}\{A, I, D\}^* = AAIAAADAAADAAIAAAIAAAAIAAADAAIAAADA \tag{22}$$

Una vez que se dejó de mandar acciones, el robot empieza a leer la memoria para realizar las acciones que el simulador le envió de acuerdo a los criterios previamente mencionados para el comportamiento del robot.

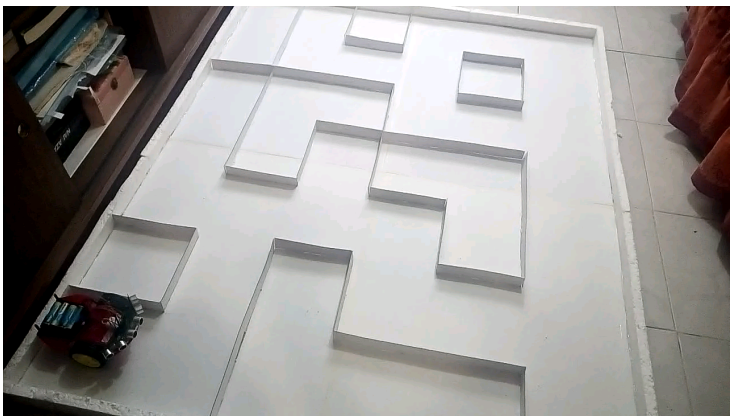


Figura 79: Punto de partida.

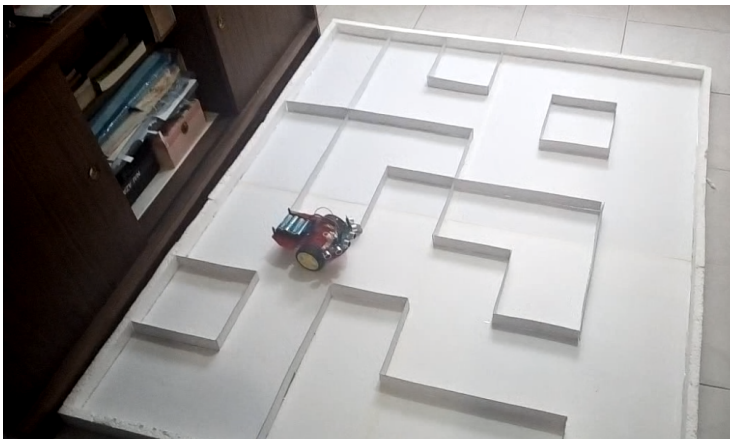


Figura 80: Robot a 20 segundos.

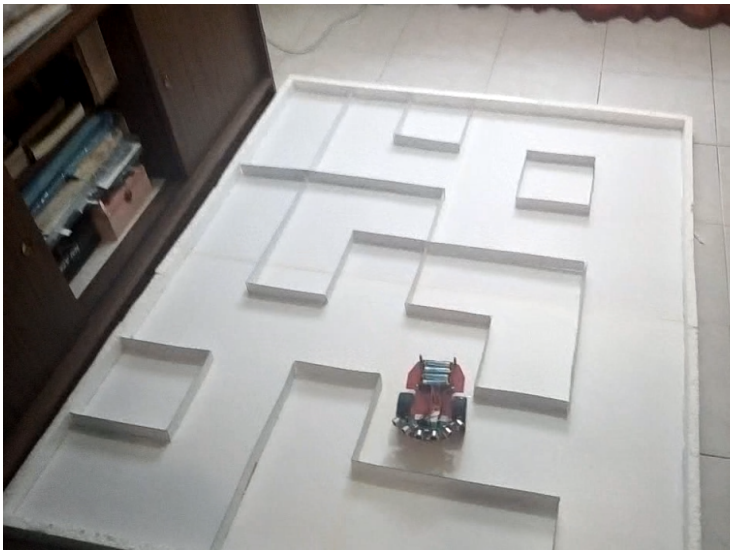


Figura 81: Robot a 50 segundos.

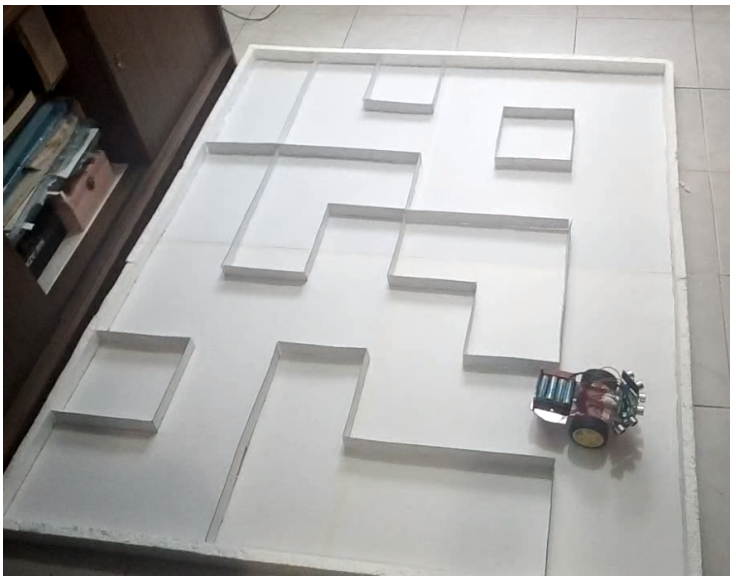


Figura 82: Robot a 75 segundos.

Figura 83: Robot a 90 segundos.

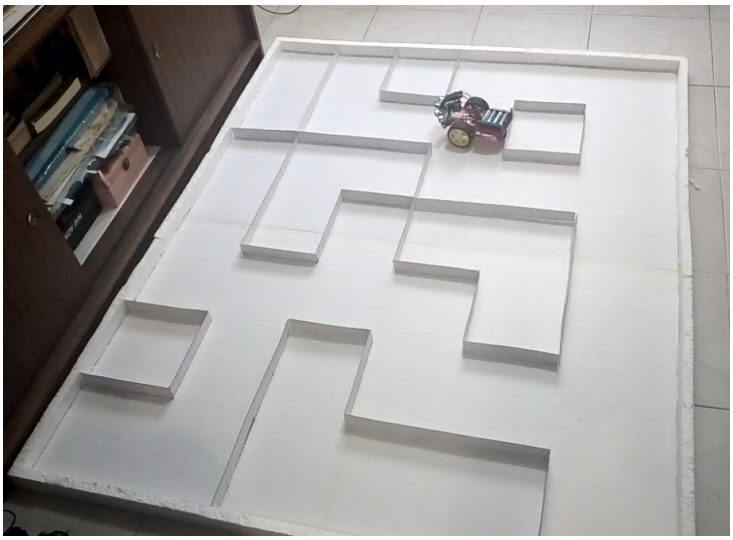


Figura 84: Robot a 100 segundos.

Figura 85: Robot a 130 segundos.

Al final el robot logra cruzar el laberinto por las rutas que el simulador le envió, aquí como observación es de que idealmente existe un tiempo en relación a los cuadros que debe avanzar para estimar el tiempo que tardará en recorrer su trayectoria, sin embargo antes mencionado los sensores y la causa de que pueda irse en dirección diferente provoca que el realizar su ajuste de dirección tome más tiempo del estimado.

7. Conclusiones

Para estos tipos de organismos el modelarlos requiere de saber qué propiedades se desean obtener, más que nada a qué con un solo paradigma o sistema puede darnos un cierto tipo de resultados, o bien obtener cierto grado de dificultad para saber qué tan cercano está el modelo con respecto a lo real. En el caso del Physarum Polycephalum a pesar de tener características simples, se convierte en un interés de que difícilmente puedan obtenerse dichas características de manera no tan compleja, pero que a su vez genere comportamientos que bien pueden llegar a ser caóticos, tal como es el caso del Juego de la Vida, aunque tenga reglas simples, se han obtenido resultados no tan triviales o poco predecibles.

En la presente simulación una de las metas es que el organismo siempre obtenga soluciones diferentes, el cual primordialmente ayuda a ver cómo el organismo se concentra en sobrevivir dependiendo de lo que haya explorado.Ahora bien con la aplicación sobre el robot da otra opción sobre un empleo de rutas poco predecibles o simplemente trayectorias aplicadas a este tipo de entidades.

8. Trabajo a futuro

En el presente modelo se concentran propiedades básicas del organimo que en escencia son características de supervivencia, para trabajo a futuro se pueden proponer los siguientes puntos:

- Incorporar otros tipos de repelentes, como sustancias, que el organismo puede adaptarse o no para saber el grado de rechazo del Physarum.
- Variar la vecindad de ver qué tanto cambia al modelo actual.
- Representar otras etapas, como su nacimiento a partir de una espora que llega a un punto de explotar y esparcir las células, para formar el Physarum hasta que por su muerte, repita el proceso.
- Para el robot, posiblemente que ya no emplee la trayectoria, empleando con base a como se va expandiendo genere algun comportamiento, se puede ver como si el Physarum controlara el robot con base a los estimulos de los nutrientes que el mismo da.

Muchas de estas ideas pueden afectar al modelo más que nada porque es ver qué tan factible es usarlo con un autómata celular.

8.1. Repositorio y derivado

El presente trabajo terminal tendrá el propósito de participar en un repositorio del Dr. Genaro Juárez Martínez [18] donde junto con otros trabajos terminales se convierten en un compilado de fácil acceso para la consulta en cualquier momento, además se puede encontrar material referente a los sistemas complejos y los autómatas celulares donde el Instituto Politécnico Nacional ha sido partice.

Por otra parte el modelo del Physarum forma parte de un proyecto de la Escuela Superior de Cómputo **ALIROB** (Articial Life Robotics Lab) [20] que tiene el objetivo de emplear proyectos basados en inteligencia articial, sistemas complejos y sistemas dinámicos para ser implementados en robtos de bajo costo donde los tópicos es hablar acerca de comportamiento no trivial, organización colectiva y auto organización.

9. Anexo

9.1. Glosario

- **Ameboidea:** Es la característica que tienen algunos protistas en que su forma siempre es cambiante al carecer de pared celular, capturan alimentos por medio de rodear con su membrana el alimento y encerrarlo.
- **Célula:** En la biología es una estructura básica de cualquier ser viviente capaz de realizar funciones de alimentación para convertir dicho alimento en energía y el poder reproducirse. Para un autómata celular se convierte en una entidad visto como una celda en el espacio capaz de cambiar sus propiedades al paso de evoluciones, producto de la influencia del sistema.
- **Célula central:** Célula de referencia dentro de la vecindad, que será influenciado por sus vecinos con el fin de ser evaluada dada una función de transición para determinar su estado de evolucón.
- **Ectoplasma:** Es la región dentro de la red que está en contacto a la membrana plasmica, pero mas lejos que el núcleo de una célula constituido por iones de calcio, magnesio y potasio que tiene la función de dar soporte interno a la céluas para formar el citoesqueleto.
- **Emergente:** Que da la impresión de inteligencia aunque sean consecuencia de comportamientos simples.
- **Plasmodio:** Conjunto de células que comparten un citoplasma, en apariencia comparten un moco amarillento donde no existe como tal una pared que separe una célula de otra en su interior.
- **Physaridae:** Es una familia que se caracteriza de esporoas de estructura viscosa, presentados en plasmodio en forma de moho o gelatina.
- **Protoplasma:** Todo lo que está en el interior de la célula.
- **Vacuola disgestiva:** Son partes de las células que almacenan sustancias ya sea nutrimentos como agua o azúcares, así como aislar materiales dañinos

10. Apendíce A

10.1. Análisis de Riesgos y Estimaciones

10.1.1. Planteamiento del problema

El aut́omata celular como se ha mencionado anteriormente busca funciones de transición con base a su región local o vecindad con el fin de generar algún comportamiento, donde los modelos que han sido desarrollados bajo este principio desarrollados junto con algoritmos de sincronización o de distribución, han tenido exito en dar solución a ciertos problemas sobre campos finitos o simplemente acercarse a la realidad a fin de obtener ventaja por tiempo o recursos.

Si bien, la gran mayoría de los autómatas celulares han tenido mayor enfoque en el software que hardware, por otra parte al hablar de un modelo basado en hardware existen ciertos riesgos ante el modelo llámense costos, restricciones, portabilidad, entre otros. Por ello a continuación se presenta un análisis con el fin de discriminar parámetros que vayan o no adecuarse para el modelo.

10.1.2. Estudio de Factibilidad

La factibilidad del sistema tiene como objetivo, tomar en cuenta 3 aspectos de disponibilidad que son necesarios para llevar a cabo los objetivos o metas del Trabajo Terminal.

- **Técnica**: ¿ Existe la tecnología necesaria ?
- **Económica**: ¿ Relación costo/beneficio ?
- **Operativa**: Capacidades organizacionales

10.1.3. Factibilidad Técnica

Para el sistema se emplearan estructuras electrónicas que van a dotar al robot de movimiento junto con un micro controlador y un receptor bluetooth que tendrá el objetivo de comunicar el robot y el programa, siendo el lenguaje C++ que se encargará de ello.

10.1.4. Factibilidad Económica

Uno de los objetivos del desarrollo del robot es que sea de bajo costo, bajo recursos que sean fáciles de adquirir, en este Trabajo Terminal tienen ya un costo, sin embargo a trabajo a futuro se busca que pueda implementarse en prototipos más grandes que posiblemente tengan un costo mayor.

10.1.5. Factibilidad Operativa

Lo que se busca con un autómata celular es dar una configuración inicial y que el sistema solo vaya actuando sin necesidad de meter algún otro dato a menos que el usuarios lo desee

Tabla 19: Métrica Orientada a la Función

Parámetro	Cuenta	Simple	Medio	Complejo	Subtotal
Entradas	3	3	4	6	9
Salidas	4	4	5	7	28
Peticiones	2	3	4	6	8
Archivos	3	7	10	15	45
Interfaces	5	5	7	10	50
				Total	140

hacer, del mismo modo junto con el robot con el simple hecho que ya esté programado, su única función sea esperar en recibir las acciones que debe tomar.

10.2. Estimación de Métricas

10.2.1. Métrica Orientada a la Función

Son medidas indirectas del sistema enfocadas a la funcionalidad o utlidad del programa bajo 5 parámetros.

- **Número de entradas**: Proporcionan datos de la aplicación donde el usuario debe interactuar.
- **Número de salidas**: Cualquier información que pueda ver el usuario, tales como errores, pantallas o informes.
- **Número de peticiones**: Se conocen coloquialmente como consultas, donde para una entrada existe un resultado inmediato.
- **Número de archivos**: Archivo lógico ya sea de base de datos o independiente generado.
- **Número de interfaces**: Transmiten información a otro sistema.

Para el caso del presente sistema se puede observar dicha evaluación en la Tabla 19.

Ajuste de complejidad

Lo siguiente en hacer es bajo una serie de atributos se busca el impacto que tiene cada uno sobre el mismo para ser evaluadas con el resultado anterior y obtener el Punto de Función mostrado en la Tabla 20

.

Tabla 20: Ajuste de la complejidad.

Atributo	Valor
Requiere el sistema de copias de seguridad y recuperación fiables	3
Se requiere de comunicación de datos	5
Existen funciones de procesamiento distribuido	5
Es crítico el rendimiento	4
Se ejecutaría el sistema en un entorno operativo existente y fuertemente utilizado	5
Requiere el sistema de entrada de datos interactiva	5
Requiere la entrada de datos interactivas que las transacciones de entrada se lleven a cabo sobre múltiples pantallas	3
Se actualizan los archivos maestros de forma interactiva	1
Son complejas las entradas, salidas, los archivos o las peticiones	3
Es complejo el procesamiento interno	4
Se ha diseñado el código para ser reutilizable	4
Están incluidas en el diseño la conversión y la instalación	3
Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones	4
Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario	4
Total	51

Entonces realizando los cálculos siguientes:

$$PF = Cuenta_T * (0.65 + 0.01 * \Sigma(Fi)) \tag{23}$$

$$PF = 140 * (0.65 + 0.01 * 51) = 162.4 \tag{24}$$

Este es nuestro punto de función donde será empleado como dato para el siguiente estudio.

10.3. COCOMO

El modelo ”Constructive Cost Model” engloba una relación matemática con el fin de estimar el esfuerzo y tiempo requerido para el proyecto tomando en cuenta el tipo de desarrollo, en función de las LDC (Líneas de Código), donde hay que tomar en cuenta el tipo de proyecto que se tiene.

- **Orgánico**: Menores a 50 KLDC, son entorno estables y existe experiencia en proyectos similares.
- **Semi-acoplado**: Son intermedio en complejidad y tamaño a no más de 300 KLDC.
- **Empotrado**: Son proyectos muy complejos donde se tiene poca experiencia y engloban una innovación ténica, sobre requisitos.

Para el presente caso hablamos de un proyecto orgánico, debido a que se tiene experiencia tanto de los componentes a emplear como sobre proyectos enfocados los autómatas celulares. Para poder obterner cada una de las estimaciones empleamos las siguientes ecuaciones [8].

$$E = Esfuerzo = aKLDC^{b} * FAE \tag{25}$$

$$T = Tiempo = cEsfuerzo^{d} \tag{26}$$

$$PR = Productividad = \frac{LDC}{Esfuerzo} \tag{27}$$

$$P = Personal = \frac{E}{T} \tag{28}$$

Calculo del Esfuerzo

Para calcular el esfuerzo necesitamos saber el KLDC, para ello empleamos una estimación de LDC por punto de función observada en la Figura 86 donde en este caso emplearemos C++.

LOC por punto de función			
Lenguaje	**LOC/FP**	**Lenguaje**	**LOC/FP**
Ensamblador	320	Basic ANSI/Quick/Turbo	64
Macroensamblador	213	Java	53
C	150	Visual C++	34
Fortran	106	Foxpro 2,5	34
Cobol	106	Visual Basic	32
Pascal	91	Delphi	29
Cobol ANSI 85	91	C++	29
Basic	91	Visual Cobol	20
RPG	80	Clipper	19
PL/I	80	Power Builder	16
Ada	71	Hoja de Calculo	6

Figura 86: Estimación de LDC por punto de función en diferentes lenguajes. [9]

[8] Métrica de Estimación COCOMO - http://slideplayer.es/slide/26622/

Anteriormente calculamos esa PF siendo de 162.4, para nuestro caso de C++ tenemos que es 29, entonces para calculas el KDLC.

$$KLDC = \frac{PF * Constante_PF}{1000} \tag{29}$$

$$KLDC = \frac{162.29 * 29}{1000} = 4.70KLDC \tag{30}$$

Con esto se comprueba que el modelo es orgánico al no superar los 50 KDLC, a continuación se muestra una tabla general en la Figura 87 de los modelos, para las constantes de acuerdo al tipo de software.

MODO	a	b	c	d
Orgánico	2.40	1.05	2.50	0.38
Semilibre	3.00	1.12	2.50	0.35
Rígido	3.60	1.20	2.50	0.32

Figura 87: Constantes de modelos en proyectos de software. [10]

Lo siguiente que haremos es evaluar 15 conductores dada una tabla, con el fin de obtener nuestra FAR (Factor de Ajuste del Esfuerzo) dada la Tabla 21, que será el producto de los atributos seleccionados.

[9]Imagen obtenida de: http://crissangelchoqueacero.blogspot.mx/2015/03/de-funcion-y-de-codigo-puntos-de.html

[10]Imagen obtenida de: https://es.slideshare.net/equipo2/cocomo-ii

Tabla 21: Ponderación de la FAE.

Conductores de coste	Valoración					
	Muy Bajo	Bajo	Nominal	Alto	Muy Alto	Ex. Alto
Fiabilidad del Software	0.75	0.88	1.00	1.15	1.40	-
Tamaño de la DB	-	0.94	1.00	1.08	1.16	-
Complejidad del producto	0.70	0.85	1.00	1.15	1.30	1.65
Restricciones tiempo/ejecución	-	-	1.00	1.11	1.30	1.66
Restricciones de almacenamiento	-	-	1.00	1.06	1.21	1.56
Volatilidad de la máquina virtual	-	0.87	1.00	1.15	1.30	-
Tiempo de respuesta del ordenador	-	0.87	1.00	1.07	1.15	-
Capacidad del analista	1.46	1.19	1.00	0.86	0.71	-
Capacidad de programadores	1.42	1.17	1.00	0.86	0.70	-
Experiencia en la aplicación	1.29	1.13	1.00	0.91	0.82	-
Experiencia en el SO utilizado	1.21	1.1	1.00	0.90	-	-
Experiencia en el lenguaje de prog.	1.14	1.07	1.00	0.95	-	-
Prácticas de programación modernas	1.24	1.10	1.00	0.91	0.82	-
Empleo de herramientos de software	1.21	1.10	1.00	0.91	0.83	-
Limitaciones de planificación	1.23	1.08	1.00	1.04	1.10	-

Por lo tanto tenemos de FAE como 0.9896, asi que procedemos a realizar los cálculos dadas las ecuaciones (4),(5),(6) y (7) de estimaciones.

Esfuerzo.

$$E = aKLDC^{b} * FAE = 3.2 * 4.70^{1.05} * 0.9896 \tag{31}$$

$$E = 16.08 \frac{personas}{mes} \tag{32}$$

Tiempo de Desarrollo.

$$T = cEsfuerzo^{d} = 2.5 * 16.08^{0.38} \tag{33}$$

$$T = 7.1 meses \tag{34}$$

Productividad.

$$PR = \frac{LDC}{Esfuerzo} = \frac{4.70K}{16.08} \tag{35}$$

$$PR = 292.28 \frac{LDC}{Personas_{M}} \tag{36}$$

Personal.

$$P = \frac{E}{T} = \frac{16.08}{7.1} \tag{37}$$

$$P = 2.26 personas \tag{38}$$

Las estimaciones arrojan que se emplearan en promedio 2 personas trabajando a lo largo de 7 meses, sin embargo el Trabajo Terminal tiene duración en promedio de 7 meses o más por ello puede llevarse a cabo.

10.4. Análisis de Riesgos

Para esta sección se van a identificar riesgos que puedan poner en peligro la integridad o el desarrollo del Trabajo Terminal con el fin de asignar un nivel de riesgo de acuerdo a diferentes categorías.

- **Proyecto**: Tienen impacto sobre el desarrollo del sistema sea presupuestal, persona, recursos o calendario, estos tendrán efecto sobre la planificación, costes y calidad final.
- **Producto**: Aquí entran aspectos de diseño, implementación, mantenimiento o interfaz.
- **Negocio**: Afectan a la organización sobre el enfoque del producto.

Para poder hacer dicha estimación se emplea una tabla de valoración de probabilidad e impacto que se muestra en la Tabla 22

Tabla 22: Valoración de Probabilidad e impacto.

Probabilidad	Muy Bajo <10 %	Bajo 10 - 25 %	Moderado 25 - 50 %	Alto 50 - 75 %	Muy Alto >75 %
Impacto	Insignificante 1	Tolerable 2	Grave 3	Catastrófica 4	

10.4.1. Tabla general de riesgos

Tabla 23: Tabla de Riesgos.

ID	PROBLEMA	CATEG.	IMP.	%	SOLUCIÓN O PREVENCIÓN
R1	Perdida de documentación, programas o módulos.	Proyecto	4	Media	Tener algún respaldo en un medio diferente al que se está trabajando.
R2	Disponibilidad de los componentes hardware a emplear.	Proyecto	3	Media	Verificar la disponibilidad y en caso de no existir, buscar alternativas.
R3	Falta de organización.	Negocio	4	Media	Establecer días y prioridades sobre cada componente del sistema sea documentación o implementación.
R4	Incompatibilidad de información entre diferentes módulos.	Producto	3	Media	Buscar la manera de adaptar la información o anticipar limitaciones.
R5	Suspensión de actividades en ESCOM.	Proyecto	3	Media	Esperar alguna reprogramación de la CATT
R6	Falta de integrante del jurado o de TT.	Proyecto	4	Alta	Previamente notificar la asistencia y conocimiento de alguna revisión o presentación en caso de existir notificar al personal pertinente.
R7	Presupuesto final.	Negocio	3	Alta	Se deberá tener algun presupuesto de reserva en caso de existir gastos adicionales.
R8	Sobrecarga de trabajo externo.	Negocio	2	Media	Adecuar tiempos con el fin de no atrasar el TT.
R9	Cambio en el calendario.	Proyecto	2	Baja	Esperar reprogramación y notificar a las personas involucradas.
R10	Subestimar el tamaño del proyecto.	Proyecto	3	Muy alta	Tener un seguimiento, además de desarrollarlo bajo el modelo en espiral para adecuar las etapas.
R11	Disponibilidad de recursos de la sala.	Proyecto	3	Baja	Verificar que lo que se vaya a emplear esté disponible.
R12	Limitaciones sobre el modelo real	Proyecto	3	Alta	Distinguis lo que se puede moderlar fácilmente y lo que no.

10.5. Semáforo de Riesgos

De acuerdo a al semáforo de riesgos mostrado en la Tabla 24 hay dos grandes riesgos, principalmente con la sobre evaluación del proyecto o que algún miembro ya sea del jurado o del TT no esté disponible en el momento de la evaluación por ello hay que notificar previamente y saber sobre lo que se va a desarrollar.

Tabla 24: Semáforo de Riesgos.

	Insignificante	**Tolerable**	**Serio**	**Catastrófico**
Muy Alto			R10	
Alto			R7,R12	R6
Moderado		R8	R2,R4,R5	R1,R3
Bajo		R9	R11	
Muy Bajo				

11. Apendíce B

11.1. Código del programa

En esta sección se muestra el código fuente del simulador, omitiendo el framework empleado para la graficación **QCustomplot** que puede ser adquirido en su página web.

11.1.1. Main

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

11.1.2. MainWindow

Esta clase se encarga de cargar toda la simulación y mostrar la ventana del simulador, contiene todos los componentes que emplea el programa para poder correr.

Mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPainter>
#include <QtCore>
#include <QtGui>
#include "rectangulo.h"
#include "hormiga.h"
#include <qbluetoothdevicediscoveryagent.h>
#include <qbluetoothsocket.h>
#include <QListWidget>
#include "dialog.h"



namespace Ui {
```

```
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_StartButton_clicked();
    void on_StopButton_clicked();

    int getNewman(int i, int j, int crit);
    bool NewmanOffset(int i, int j, int crit, int orien);
    bool NewmanOffset2(int i, int j, int crit);

    int random();
    int randomOffset();

    void newGeneration();
    void evaluar();

    void updateDimension();
    void actualiza(int i, int j,int orientacion);

    void on_colorButton_clicked();
    void on_dimButton_clicked();
    void on_fileButton_clicked();
    void on_salvaButton_clicked();

    void updateGraphic();

    void on_OffB_clicked();
    void deviceDiscovered( const QBluetoothDeviceInfo &device );


    void timerGeneral(int a);
    void Cambio();


    void on_ListaV_itemClicked(QListWidgetItem *item);
    void on_checkHormiga_clicked();
    void on_pushButton_clicked();
    void rastreo();
```

```
    void on_AcercaB_clicked();

private:
    Ui::MainWindow *ui;
    QGraphicsScene *scene;
    Dialog *nuevoD;

    Rectangulo **recta;
    Hormiga *hormiga;
    int **aux;
    int ** min;

    int dimension;

    QTimer* timer;
    QTimer* listen;
    QTimer* send;
    QTimer* rastre;
    QTime time;
    QBrush* color;

    int xInit,yInit;
    int incre, active, mayor;
    int Caso;
    double dm;

    QVector<double> x,y;

    QBluetoothDeviceDiscoveryAgent *agent = new QBluetoothDeviceDiscoveryAgent;
    QBluetoothSocket *socket;
    QString string;

protected:

    bool eventFilter(QObject *target, QEvent *event);

};

#endif // MAINWINDOW_H
```

Mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
#include <QTimer>
#include <QFileDialog>
#include <QList>
```

```
#include <algorithm>
#include <QtBluetooth>
#include <QListWidget>


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    timer(new QTimer(this)),
    ui(new Ui::MainWindow)
{


    Caso = false;

    send = new QTimer();
    rastre = new QTimer();

    send->setInterval(1000);
    timer->setInterval(10);
    rastre->setInterval(1000);

    ui->setupUi(this);

    connect(agent, SIGNAL(deviceDiscovered(QBluetoothDeviceInfo)),
           this, SLOT(deviceDiscovered(QBluetoothDeviceInfo)));
    connect(send, SIGNAL(timeout()), this, SLOT(sendB()));
    connect(timer, SIGNAL(timeout()), this, SLOT(evaluar()));
    connect(rastre, SIGNAL(timeout()), this, SLOT(rastreo()));
    connect( ui->SliderTime,
        SIGNAL(valueChanged(int)),this,SLOT(timerGeneral(int)) );
    connect( ui->Comportamiento, SIGNAL(clicked()),this,SLOT(Cambio()));

    dimension = 4;
    qsrand((uint)time.msec() );


    socket = new QBluetoothSocket(QBluetoothServiceInfo::RfcommProtocol);

    updateDimension();
    agent->start();


    ui->generationLabel->setText("Generacion # 0");
    ui->activeLabel->setText("Celulas Activas # 0");
    ui->timerLabel->setText("Timer actual 0 ms");
```

```
}

MainWindow::~MainWindow()
{
    delete ui;
    connect( agent, SIGNAL( deviceDiscovered( QBluetoothDeviceInfo ) ) , this
        , SLOT ( deviceDiscovered( QBluetoothDeviceInfo ) ) );
    agent->start();

}

void MainWindow::evaluar()
{

    incre++;
    active = 0;


    for(int i = 0; i < dimension; i++)
    {
        for(int j = 0; j < dimension; j++)
        {
            aux[i][j] = recta[i][j].getEdo();
        }
    }

    for( int i = 0; i < dimension; i++)
    {
        for(int j = 0; j < dimension; j++)
        {

            /*
             * 0 campo           3 punto inicial
             * 1 nutriente       4 gel
             * 2 repelente       5 plasmodio
            */

            int orientacion = randomOffset() + 1;
            switch( recta[i][j].getEdo() )
            {

                case 0:

                        if( min[i][j] == 0 )
                        {
                            actualiza(i,j,orientacion);
                        }
                break;
```

```
case 1:

       if( getNewman(i,j,5) != 0 )
       {
           aux[i][j] = 6;
       }
break;

case 4:
       if( Caso )
       {
           if( min[i][j] == 0 )
           {
              if( NewmanOffset(i,j,3,orientacion) ||
                  NewmanOffset(i,j,5,orientacion) ||
                  NewmanOffset(i,j,6,orientacion) )
               {
                  min[i][j] = orientacion;
                  aux[i][j] = 5;
               }
           }

       }
       else
       {
           if( min[i][j] == 0 && getNewman(i,j,7) == 0 &&
               getNewman(i,j,0) == 0 && ( getNewman(i,j,3) != 0
               || getNewman(i,j,6) != 0 || getNewman (i,j,5) != 0
               ) ) // variante estatica del physarum
           {
              if( NewmanOffset(i,j,3,orientacion) ||
                  NewmanOffset(i,j,5,orientacion) ||
                  NewmanOffset(i,j,6,orientacion) )
               {
                  min[i][j] = orientacion;
                  aux[i][j] = 5;
               }
           }

       }
break;


case 5:
       if( Caso )
       {
```

```
                        if(!NewmanOffset2(i,j,8) && !NewmanOffset2(i,j,5)
                            && getNewman(i,j,6) == 0 && getNewman(i,j,4) ==
                            0 && getNewman(i,j,1) == 0 )
                        {

                           aux[i][j] = 0;
                           min[i][j] = 0;

                        }
                        else
                        {
                            aux[i][j] = 8;
                        }
                    }
                    else
                    {
                        if(!NewmanOffset2(i,j,8) && !NewmanOffset2(i,j,5)
                            && getNewman(i,j,6) == 0 && getNewman(i,j,4) ==
                            0 && getNewman(i,j,1) == 0 && getNewman(i,j,3)
                            == 0 )
                        {

                           aux[i][j] = 0;
                           min[i][j] = 0;

                        }
                        else
                        {
                            aux[i][j] = 8;
                        }
                    }
                    active++;
            break;


            case 7:
                if( getNewman(i,j,5) != 0 || getNewman(i,j,6) != 0 ||
                    getNewman(i,j,3) != 0 || getNewman(i,j,4) != 0 )
                {
                    aux[i][j] = 4;
                }
            break;

            case 8:
                aux[i][j] = 5;
            break;
        }
    }
```

```
        //qDebug() << "***";
    }

    newGeneration();

    x.push_front( incre );
    y.push_front( active );

    if( active > mayor )
    {
        mayor = active;
    }
    ui->generationLabel->setText( " óGeneracin # " + QString::number(incre) );
    ui->activeLabel->setText( " éClulas Activas # " + QString::number(active)
        );
    updateGraphic();
}

void MainWindow::updateDimension()
{

    scene = new QGraphicsScene(this);
    scene->installEventFilter(this);

    ui->graphicsView->setScene(scene);



    ui->checkHormiga->setChecked(false);
    int ancho = ui->graphicsView->width();
    ///////////
    dm = ancho / dimension;
    ////////////

    recta = new Rectangulo*[dimension];
    aux = new int*[dimension];
    min = new int*[dimension];

    incre = 0;
    active = 0;
    mayor = 0;
    x.clear();
    y.clear();

    for(int i = 0; i < dimension; i++)
    {
        recta[i] = new Rectangulo[dimension];
```

```
        aux[i] = new int[dimension];

        min[i] = new int[dimension];

        for(int j = 0; j < dimension; j++)
        {
         recta[i][j].setAttrib(j*dm,i*dm,dm);

         min[i][j] = 0;

         if( i == 0 || i == dimension-1 || j == 0 || j == dimension -1 )
         {
             recta[i][j].setEdo(2);
         }

        }
    }

    for(int i = 0; i < dimension; i++)
    {
        for(int j = 0; j < dimension; j++)
        {
            scene->addItem(&recta[i][j]);
        }
    }

    rastre->stop();
    ui->graphicsView->fitInView(scene->sceneRect(), Qt::KeepAspectRatio);
    ui->LabelDim->setText( "óDimensin del arreglo actual: " + QString::number(
        dimension ) );

    delete socket;
    socket = new QBluetoothSocket(QBluetoothServiceInfo::RfcommProtocol);

}

void MainWindow::actualiza(int i, int j,int orientacion)
{
    switch( orientacion )
        {
            case 1:
                   if( recta[i][j+1].getEdo() == 4 || recta[i][j+1].getEdo() ==
                       3 || recta[i][j+1].getEdo() == 6)
                   {
                       aux[i][j] = 7;
                   }
               break;
            case 2:
```

```
                if( recta[i-1][j].getEdo() == 4 || recta[i-1][j].getEdo() ==
                    3 || recta[i-1][j].getEdo() == 6)
                {
                    aux[i][j] = 7;
                }
            break;
        case 3:
                if( recta[i][j-1].getEdo() == 4 || recta[i][j-1].getEdo() ==
                    3 || recta[i][j-1].getEdo() == 6)
                {
                    aux[i][j] = 7;
                }
            break;
        case 4:
                if( recta[i+1][j].getEdo() == 4 || recta[i+1][j].getEdo() ==
                    3 || recta[i+1][j].getEdo() == 6)
                {
                    aux[i][j] = 7;
                }
            break;
    }

}

void MainWindow::newGeneration()
{
    for(int i = 0; i < dimension; i++)
    {
        for(int j = 0; j < dimension; j++)
        {
         recta[i][j].setEdo( aux[i][j] );
         recta[i][j].update();
        }
        //qDebug() << "***";
    }


}


int MainWindow::getNewman(int i, int j, int crit)
{
    int x = 0;


    if( recta[i][j+1].getEdo() == crit ) { x++; }
    if( recta[i-1][j].getEdo() == crit ) { x++; }
    if( recta[i][j-1].getEdo() == crit ) { x++; }
```

```
    if( recta[i+1][j].getEdo() == crit ) { x++; }



    return x;

}

bool MainWindow::NewmanOffset(int i, int j, int crit, int orien)
{
   if( recta[i][j+1].getEdo() == crit && orien == 1) //&& min[i][j+1] == 2 )
   {
      return true;
   }
   else
   if( recta[i-1][j].getEdo() == crit && orien == 2) //&& min[i-1][j] == 3 )
   {
      return true;
   }
   else
   if( recta[i][j-1].getEdo() == crit && orien == 3 ) //&& min[i][j-1] == 0 )
   {
      return true;
   }
   else
   if( recta[i+1][j].getEdo() == crit && orien == 4) //&& min[i+1][j] == 1 )
   {
      return true;
   }
   return false;
}

bool MainWindow::NewmanOffset2(int i, int j, int crit)
{
    if( recta[i][j+1].getEdo() == crit && min[i][j+1] == 3 )
    {
       return true;
    }
    else
    if( recta[i-1][j].getEdo() == crit && min[i-1][j] == 4 )
    {
       return true;
    }
    else
    if( recta[i][j-1].getEdo() == crit && min[i][j-1] == 1 )
    {
       return true;
    }
```

```
    else
    if( recta[i+1][j].getEdo() == crit && min[i+1][j] == 2 )
    {
        return true;
    }
    return false;
}


int MainWindow::random()
{
    return qrand() % ((100 + 1) - 0) + 0;
}

int MainWindow::randomOffset()
{
    return qrand() % ((3 + 1) - 0) + 0;
}

void MainWindow::on_StartButton_clicked()
{
    //generations = number;
    timer->start();
    //evaluar();
}

void MainWindow::on_StopButton_clicked()
{
    timer->stop();
}

void MainWindow::on_colorButton_clicked()
{

    QString edo = ui->stateBox->currentText();

    int valor;

    if( QString::compare("Campo Libre",edo,Qt::CaseInsensitive) == 0 )
    {
        valor = 0;
    }
    else
    if( QString::compare("Nutriente",edo,Qt::CaseInsensitive) == 0 )
    {
        valor = 1;
    }
    else
```

```
    if( QString::compare("Repelente",edo,Qt::CaseInsensitive) == 0 )
    {
        valor = 2;
    }
    else
    if( QString::compare("Punto Inicial",edo,Qt::CaseInsensitive) == 0 )
    {
        valor = 3;
    }
    else
    if( QString::compare("Gel",edo,Qt::CaseInsensitive) == 0 )
    {
        valor = 4;
    }
    else
    if( QString::compare("Plasmodio",edo,Qt::CaseInsensitive) == 0 )
    {
        valor = 5;
    }

    for(int i = 0; i < dimension; i++)
    {
        for(int j = 0; j < dimension; j++)
        {
            recta[i][j].setBrocha( valor );
        }
    }


}

void MainWindow::on_dimButton_clicked()
{
    timer->stop();
    dimension = ui->dimensionText->text().toInt();
    updateDimension();
}

void MainWindow::on_fileButton_clicked()
{

      int newDim = 0;
      int i = 0;
      bool llave = false;

      QString fileName1 = QFileDialog::getOpenFileName(this,tr("Open Text
          File"), "", tr("Text Files (*.txt)"));
```

```
      QFile inputFile(fileName1);
      if (inputFile.open(QIODevice::ReadOnly))
      {
         QTextStream in(&inputFile);
         while (!in.atEnd())
         {

            QString line = in.readLine();
            newDim = line.size();

            if( newDim != 0 )
             {
               if(!llave)
               {
                   //qDebug() << "Actualizado ";
                   llave = true;
                   dimension = newDim;
                   updateDimension();
               }


               //qDebug() << line << newDim;

               for(int j = 0; j < dimension; j++)
               {
                    char chara = line.at(j).toLatin1();
                    aux[i][j] = (int) chara - 48;
               }
               i++;
             }
         }
         inputFile.close();
      }

      newGeneration();

}

void MainWindow::on_salvaButton_clicked()
{
    QString filename= QFileDialog::getSaveFileName( ) + ".txt";

    QFile file( filename );
    if ( file.open(QIODevice::ReadWrite) )
    {
        QTextStream stream( &file );
        QString cadena = "";
```

```
        for(int i = 0; i < dimension; i++)
        {
            for(int j = 0; j < dimension; j++)
            {
                cadena = cadena + QString::number( recta[i][j].getEdo() );
            }

            cadena = cadena + "\n";
            stream << cadena;
            cadena = "";
        }

    }

    file.close();
}

void MainWindow::updateGraphic()
{
    // create graph and assign data to it:
    ui->graphicWidget->addGraph();
    ui->graphicWidget->graph(0)->setData(x, y);
    // give the axes some labels:
    ui->graphicWidget->xAxis->setLabel("óGeneracin");
    ui->graphicWidget->yAxis->setLabel("Gel Activo");
    // set axes ranges, so we see all data:
    ui->graphicWidget->xAxis->setRange(0, incre );
    ui->graphicWidget->yAxis->setRange(0, mayor );
    ui->graphicWidget->replot();
}




void MainWindow::on_OffB_clicked()
{
   rastre->stop();
   socket->write("0");

}

void MainWindow::deviceDiscovered(const QBluetoothDeviceInfo &device)
{
    qDebug() << "Found new device:" << device.name() << '(' <<
        device.address().toString() << ')';
```

```
    ui->ListaV->addItem( device.address().toString() );
}

void MainWindow::on_ListaV_itemClicked(QListWidgetItem *item)
{
    string = item->text();

    ui->ListaV->addItem(string);

    socket->connectToService(QBluetoothAddress(string),
        QBluetoothUuid::SerialPort, QIODevice::ReadWrite);

    qDebug() << "--> " << socket->errorString();

}



void MainWindow::timerGeneral(int a)
{
    timer->setInterval(a);
    ui->timerLabel->setText("Timer actual" + QString::number(a) + "ms");
}

void MainWindow::Cambio()
{
    Caso = !Caso;

    if( Caso )
    {
        ui->Comportamiento->setText( "áDinmico On" );
    }
    else
    {
        ui->Comportamiento->setText( "áDinmico Off" );
    }


}


bool MainWindow::eventFilter(QObject *target, QEvent *event)
{
    int xP;
    int yP;

    //qDebug() << "--> Dim " << dm;
```

```
    if ( target == scene && ui->checkHormiga->isChecked() )
    {
        if( event->type() == QEvent::GraphicsSceneMousePress )
        {
            delete hormiga;
            hormiga = new Hormiga();
            const QGraphicsSceneMouseEvent* const me = static_cast <const
                QGraphicsSceneMouseEvent*>(event);
            const QPointF position = me->scenePos();
            //qDebug() << position.x() << " : " << position.y();
            xP = position.x();
            yP = position.y();

            for(int i = 0; i < dimension; i++)
            {
                for( int j = 0; j < dimension; j++ )
                {

                    if( (j*dm) < xP && (i*dm) < yP && ((j+1)*dm) > xP &&
                        ((i+1)*dm) > yP )
                    {
                        hormiga->setAttrib(j*dm,i*dm,dm);
                        hormiga->setCoord(i,j);
                        hormiga->setFrente(1);
                        scene->addItem( hormiga );
                        break;
                    }

                }
            }


        }
    }

    return QMainWindow::eventFilter(target,event);

}

void MainWindow::on_checkHormiga_clicked()
{

    hormiga = new Hormiga();
    scene->addItem(hormiga);

    for(int i = 0; i < dimension; i++)
    {
        for(int j = 0; j < dimension; j++)
```

```
        {
            recta[i][j].setLlave();
        }
    }
}

void MainWindow::on_pushButton_clicked()
{
    rastre->start();
    //rastreo();
}

void MainWindow::rastreo()
{
    /*
      2
     3 1
      4

      Acciones


      1 Rota Izquierda
      2 Rota Derecha
      3 Avanza

    */

    int posX = hormiga->getX();
    int posY = hormiga->getY();

    qDebug() << "-->Frente: " << hormiga->getFrente();


    switch(hormiga->getFrente())
    {
        case 1:

            qDebug() << "--> Entre en 1";

            if( recta[posY-1][posX].getEdo() == 5 ||
                recta[posY-1][posX].getEdo() == 6 ||
                recta[posY-1][posX].getEdo() == 8 )
            {

                qDebug() << "--> Entre en Izquierdo";

                delete hormiga;
```

```
            hormiga = new Hormiga();

            hormiga->setAttrib(posX*dm,(posY-1)*dm,dm);
            hormiga->setCoord(posY-1,posX);
            hormiga->setFrente(2);

            scene->addItem(hormiga);

            socket->write("1");
            socket->write("3");

        }
        else
        {

            if( recta[posY][posX+1].getEdo() == 5 ||
                recta[posY][posX+1].getEdo() == 6 ||
                recta[posY][posX+1].getEdo() == 8 )
            {
                qDebug() << "--> Entre en Frente";

                delete hormiga;
                hormiga = new Hormiga();

                hormiga->setAttrib((posX+1)*dm,posY*dm,dm);
                hormiga->setCoord(posY,posX+1);

                scene->addItem(hormiga);

                socket->write("3");

            }
            else
            {
                qDebug() << "--> Entre en Cambio";
                hormiga->setFrente(4);

                socket->write("2");

            }
        }


    break;

    case 2:

        qDebug() << "--> Entre en 2";
```

```
    if( recta[posY][posX-1].getEdo() == 5 ||
        recta[posY][posX-1].getEdo() == 6 ||
        recta[posY][posX-1].getEdo() == 8 )
    {

        delete hormiga;
        hormiga = new Hormiga();

        hormiga->setAttrib((posX-1)*dm,posY*dm,dm);
        hormiga->setCoord(posY,posX-1);
        hormiga->setFrente(3);

        scene->addItem(hormiga);

        socket->write("1");
        socket->write("3");

    }
    else
    {

        if( recta[posY-1][posX].getEdo() == 5 ||
            recta[posY-1][posX].getEdo() == 6 ||
            recta[posY-1][posX].getEdo() == 8 )
        {
            delete hormiga;
            hormiga = new Hormiga();

            hormiga->setAttrib(posX*dm,(posY-1)*dm,dm);
            hormiga->setCoord(posY-1,posX);

            scene->addItem(hormiga);

            socket->write("3");

        }
        else
        {
            hormiga->setFrente(1);

            socket->write("2");
        }
    }
break;

case 3:
```

```
    qDebug() << "--> Entre en 3";

    if( recta[posY+1][posX].getEdo() == 5 ||
        recta[posY+1][posX].getEdo() == 6 ||
        recta[posY+1][posX].getEdo() == 8 )
    {

        delete hormiga;
        hormiga = new Hormiga();

        hormiga->setAttrib(posX*dm,(posY+1)*dm,dm);
        hormiga->setCoord(posY+1,posX);
        hormiga->setFrente(4);

        scene->addItem(hormiga);

        socket->write("1");
        socket->write("3");

    }
    else
    {

        if( recta[posY][posX-1].getEdo() == 5 ||
            recta[posY][posX-1].getEdo() == 6 ||
            recta[posY][posX-1].getEdo() == 8 )
        {
            delete hormiga;
            hormiga = new Hormiga();

            hormiga->setAttrib((posX-1)*dm,posY*dm,dm);
            hormiga->setCoord(posY,posX-1);

            scene->addItem(hormiga);

            socket->write("3");

        }
        else
        {
            hormiga->setFrente(2);

            socket->write("2");
        }
    }
break;

case 4:
```

```
        qDebug() << "--> Entre en 4";

        if( recta[posY][posX+1].getEdo() == 5 ||
            recta[posY][posX+1].getEdo() == 6 ||
            recta[posY][posX+1].getEdo() == 8 )
        {

            delete hormiga;
            hormiga = new Hormiga();

            hormiga->setAttrib((posX+1)*dm,posY*dm,dm);
            hormiga->setCoord(posY,posX+1);
            hormiga->setFrente(1);

            scene->addItem(hormiga);

            socket->write("1");
            socket->write("3");

        }
        else
        {

            if( recta[posY+1][posX].getEdo() == 5 ||
                recta[posY+1][posX].getEdo() == 6 ||
                recta[posY+1][posX].getEdo() == 8 )
            {
                delete hormiga;
                hormiga = new Hormiga();

                hormiga->setAttrib(posX*dm,(posY+1)*dm,dm);
                hormiga->setCoord(posY+1,posX);
                scene->addItem(hormiga);

                socket->write("3");

            }
            else
            {
                hormiga->setFrente(3);

                socket->write("2");
            }
        }
    break;
}
```

```
}



void MainWindow::on_AcercaB_clicked()
{
    nuevoD = new Dialog(this);
    nuevoD->show();
}
```

11.1.3. Rectangulo

Tiene la función de dibujar la celda o el rectángulo que será la célula dentro de nuestro sistema. Es requerido crear la clase rectángulo debido a que QtCreator por si solo no puede crear un arreglo de rectángulos con caraterísticas como dar click sobre la figura, cambiarle color en función de un valor. Por ello se emplea para que dentro de Mainwindow simplemente se haga creación del arreglo que ya el IDE y C++ puede manipular con las características mencionadas.

Rectangulo.h

```
#ifndef RECTANGULO_H
#define RECTANGULO_H
#include <QPainter>
#include <QGraphicsItem>
#include <QDebug>


class Rectangulo : public QGraphicsItem
{
public:
    Rectangulo();

    int indX;
    int indY;

    int x;
    int y;
    int dim;
    int estado;
    int brocha;
    int subEdo;
    bool llave;


    QRectF boundingRect( ) const ;
    void setAttrib( int a, int b, int c );
    void getAttrib( );
    void setEdo(int x);
    void setBrocha(int a);
    void setSedo(int a);
    void setLlave();
    int getSedo();
    int getEdo();
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
        QWidget *widget);

private:
```

```
    QRectF** rec;

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event);

};

#endif // RECTANGULO_H
```

Rectangulo.cpp

```
#include "rectangulo.h"

Rectangulo::Rectangulo()
{
    llave = false;
    estado = 0;
    brocha = 0;

}

QRectF Rectangulo::boundingRect() const
{
    return QRectF(x,y,dim,dim);
}

void Rectangulo::setAttrib(int a, int b, int c)
{
    x = a;
    y = b;
    dim = c;


}

void Rectangulo::getAttrib()
{
     qDebug() << x;
     qDebug() << y;
     qDebug() << dim;

}

void Rectangulo::setEdo(int x)
{
    estado = x;
}
```

```
void Rectangulo::setBrocha(int a)
{
    brocha = a;
}

void Rectangulo::setSedo(int a)
{
    subEdo = a;
}

void Rectangulo::setLlave()
{
    llave = !llave;
}

int Rectangulo::getSedo()
{
    return subEdo;
}

int Rectangulo::getEdo()
{
    return estado;
}

void Rectangulo::paint(QPainter *painter, const QStyleOptionGraphicsItem
    *option, QWidget *widget)
{

    QRectF rec = boundingRect();

     switch(estado)
     {
        case 0:  // Campo Libre
               painter->setBrush( QColor(0, 0, 50, 255) );
               break;
        case 1:  // Nutriente
               painter->setBrush( QColor(0, 0, 204, 255) );
               break;
        case 2: // Repelente
               painter->setBrush( QColor(255, 0, 0, 255) );
               break;
        case 3: // Punto Inicial
               painter->setBrush( QColor(0,0,0, 255) );
               break;
        case 4: // Gel
               painter->setBrush( QColor(255, 255, 0, 255) );
               break;
```

```
        case 5: // Plasmodio
               painter->setBrush( QColor(0,153,0,255) );
               break;
        case 6: // Nutriente Exitado
               painter->setBrush( QColor(255,255,153,255) );
               break;
        case 7: // Plasmodio
               painter->setBrush( QColor(128,128,128,255) );
               break;
        case 8:

              painter->setBrush( QColor(64,185,64,255) );

              break;

    }

   painter->setPen(Qt::NoPen);
   painter->drawRect(rec);


}

void Rectangulo::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
   if( !llave )
   {
       estado = brocha;
       update();
       QGraphicsItem::mousePressEvent(event);
   }
}
```

11.1.4. Hormiga

Empleado para representar el robot dentro de la simulación, es parecido a Rectangulo, la diferencia que su representación es un círculo blanco que puede modificar sus coordenadas para posicionarse dentro del simulador.

Hormiga.h

```
#ifndef HORMIGA_H
#define HORMIGA_H
#include <QPainter>
#include <QGraphicsItem>
#include <QDebug>


class Hormiga : public QGraphicsItem
{

public:
    Hormiga();
    int x;
    int y;
    int dim;
    int posX;
    int posY;

    int frente;

    QRectF boundingRect( ) const ;
    void actualizar();
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
        QWidget *widget);

    void setAttrib( int a, int b, int c );
    void setCoord(int a, int b);
    int getX();
    int getY();
    void setFrente(int a);
    int getFrente();

protected:

};

#endif // RECTANGULO_H
```

Hormiga.cpp

```
#include "hormiga.h"

Hormiga::Hormiga()
{

}

QRectF Hormiga::boundingRect() const
{
    return QRectF(x,y,dim,dim);
}

void Hormiga::actualizar()
{
    update();
}

void Hormiga::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
    QWidget *widget)
{
    QRectF rec = boundingRect();

    painter->setBrush( QColor(255,255,255,255) );
    painter->drawEllipse(rec);

}

void Hormiga::setAttrib(int a, int b, int c)
{
    x = a;
    y = b;
    dim = c;
}

void Hormiga::setCoord(int a, int b)
{
    posX = b;
    posY = a;
}

int Hormiga::getX()
{
    return posX;
}

int Hormiga::getY()
{
    return posY;
```

```
}

void Hormiga::setFrente(int a)
{
    frente = a;
}

int Hormiga::getFrente()
{
    return frente;
}
```

11.2. Código del robot

Para el caso del robot, el código esta pensado en el diagrama que se propusó dentro del documento, primeramente guardar todas las acciones que se reciben del Bluetooth, para después al empezar a correr, evalua los sensores para poder leer el valor en su memoria.

```
#define trigPin1 P3_2
#define echoPin1 P3_3
#define LED_first_ping P2_4

#define trigPin2 P3_1
#define echoPin2 P3_0
#define LED_second_ping P2_0

#define trigPin3 P3_6
#define echoPin3 P3_7
//--------------------------------------------------------------------------------

void SonarSensor1();
void SonarSensor2();
void SonarSensor3();


long duration, distance, UltraSensor1, UltraSensor2, UltraSensor3;
bool llave;

char valor = 0;
int array[100];

int i = 0;
int index = 0;

char data;
String SerialData="";
void setup()
{

  Serial.begin (9600);

  pinMode(trigPin1, OUTPUT);
  pinMode(echoPin1, INPUT);
  pinMode(LED_first_ping, OUTPUT);


  pinMode(trigPin2, OUTPUT);
  pinMode(echoPin2, INPUT);
  pinMode(LED_second_ping, OUTPUT);
```

```
  pinMode(P1_2,OUTPUT);
  pinMode(P1_3,OUTPUT);

  pinMode(P1_4,OUTPUT);
  pinMode(P1_5,OUTPUT);

  pinMode(trigPin3, OUTPUT);
  pinMode(echoPin3, INPUT);

  digitalWrite(P1_2,LOW);
  digitalWrite(P1_3,LOW);
  digitalWrite(P1_4,LOW);
  digitalWrite(P1_5,LOW);

  digitalWrite(LED_first_ping,LOW);
  digitalWrite(LED_second_ping,LOW);
}

void loop()
{
  blue();

  do
  {

    digitalWrite(P1_2,LOW);
    digitalWrite(P1_3,LOW);
    digitalWrite(P1_4,LOW);
    digitalWrite(P1_5,LOW);
    llave = false;



    SonarSensor1();
    UltraSensor1 = distance;
    SonarSensor2();
    UltraSensor2 = distance;
    SonarSensor3();
    UltraSensor3 = distance;

    while(Serial.available())
    {
      delay(10);
      data=Serial.read();
      SerialData+=data;
    }

    if(UltraSensor1 <=3)
```

```
{
  digitalWrite(P1_3,HIGH);
  digitalWrite(P1_5,HIGH);
  delay(100);
  digitalWrite(P1_3,LOW);
  digitalWrite(P1_5,LOW);
  llave = true;
}
else
{
  digitalWrite(LED_first_ping,LOW);
}


if(UltraSensor2 <=2)
{
  digitalWrite(P1_2,HIGH);
  digitalWrite(P1_5,HIGH);
  delay(50);
  digitalWrite(P1_2,LOW);
  digitalWrite(P1_5,LOW);
  llave = true;
}
else
{
  digitalWrite(LED_first_ping,LOW);
}

if(UltraSensor3 <=3)
{
  //digitalWrite(LED_first_ping,HIGH);
  digitalWrite(P1_4,HIGH);
  digitalWrite(P1_3,HIGH);
  delay(50);
  digitalWrite(P1_4,LOW);
  digitalWrite(P1_3,LOW);
  llave = true;
}
else
{
  digitalWrite(LED_first_ping,LOW);
}

if( !llave )
{
  if( array[index] == '1' )
  {
     digitalWrite(LED_first_ping,HIGH);
```

```
            analogWrite(P1_2,250);
            digitalWrite(P1_5,HIGH);
            delay(215);
            digitalWrite(LED_first_ping,LOW);
            digitalWrite(P1_2,LOW);
            digitalWrite(P1_5,LOW);
        }
        else
        if( array[index] == '2' )
        {
           digitalWrite(LED_first_ping,HIGH);
           digitalWrite(P1_4,HIGH);
           analogWrite(P1_3,250);
           delay(215);
           digitalWrite(P1_3,LOW);
           digitalWrite(P2_4,LOW);
           digitalWrite(P1_4,LOW);


        }
        else
        if( array[index] == '3' )
        {

            digitalWrite(LED_first_ping,HIGH);
            analogWrite(P1_2,250);
            analogWrite(P1_4,255);
            delay(150);
            digitalWrite(P2_4, LOW);
            digitalWrite(P1_2,LOW);
            digitalWrite(P1_4,LOW);

        }
        index++;
      }

      delay(500);

    }while( index < i );


    i = 0;
    index = 0;
    memset(array, 0, sizeof(array));

}
```

```
void SonarSensor2()
{

digitalWrite(trigPin2, LOW);// put trigpin LOW
delayMicroseconds(2);// wait 2 microseconds
digitalWrite(trigPin2, HIGH);// switch trigpin HIGH
delayMicroseconds(10); // wait 10 microseconds
digitalWrite(trigPin2, LOW);// turn it LOW again

duration = pulseIn(echoPin2, HIGH);
distance= (duration/2) / 29.1;
}


void SonarSensor1()
{

digitalWrite(trigPin1, LOW);// put trigpin LOW
delayMicroseconds(2);// wait 2 microseconds
digitalWrite(trigPin1, HIGH);// switch trigpin HIGH
delayMicroseconds(10); // wait 10 microseconds
digitalWrite(trigPin1, LOW);// turn it LOW again

duration = pulseIn(echoPin1, HIGH);
distance= (duration/2) / 29.1;
}



void SonarSensor3()
{

digitalWrite(trigPin3, LOW);// put trigpin LOW
delayMicroseconds(2);// wait 2 microseconds
digitalWrite(trigPin3, HIGH);// switch trigpin HIGH
delayMicroseconds(10); // wait 10 microseconds
digitalWrite(trigPin3, LOW);// turn it LOW again

duration = pulseIn(echoPin3, HIGH);
distance= (duration/2) / 29.1;
}


void blue()
{
  for(;;)
  {
    if (Serial.available())//Si hay una caracter en el buffer serial
```

```
    {
     valor = Serial.read();// Se lee el valor énumrico en el puerto serie.

     if (valor == '1')//Si el valor es 1.
     {
       digitalWrite(LED_first_ping, HIGH); //Se enciende el led rojo.
       delay(1000);
       digitalWrite(LED_first_ping,LOW);
       array[i] = valor;
       i++;
     }
     else if (valor == '2')
     {
       digitalWrite(LED_first_ping, HIGH); //Se enciende el led rojo.
       delay(1000);
       digitalWrite(LED_first_ping,LOW);
       array[i] = valor;
       i++;
     }
     else if (valor == '3')
     {
       digitalWrite(LED_first_ping, HIGH); //Se enciende el led rojo.
       delay(1000);
       digitalWrite(LED_first_ping,LOW);
       array[i] = valor;
       i++;
     }
     else if (valor == '0')
     {
       digitalWrite(LED_first_ping,LOW);
       break;
     }
   }
  }
}
```

Referencias

[1] Adamatzky, A. (2016a). Advances in physarum machines: Sensing and computing with slime mould. pages 563–594.

[2] Adamatzky, A. (2016b). Advances un physarum machines. Springer. [Figura]. Recuperado de: https://media.springernature.com/lw785/springer-static/image/chp%3A10.1007%2F978-3-319-26662-6_27/MediaObjects/385644_1_En_27_Fig4_HTML.gif.

[3] Adamatzky, A. (2016c). Advances un physarum machines. Springer. [Figura]. Recuperado de: https://media.springernature.com/lw785/springer-static/image/chp%3A10.1007%2F978-3-319-26662-6_27/MediaObjects/385644_1_En_27_Fig4_HTML.gif.

[4] Adamatzky, A., Asoma, R., Chapa, S. V., Juárez, G., and Rhodes, C. (2011a). Approximating mexican highways with slime mould. *Natural Computing: an international journal*, 10:1195,1214.

[5] Adamatzky, A., Asoma, R., Chapa, S. V., Juárez, G., and Rhodes, C. (2011b). Approximating mexican highways with slime mould. *Natural Computing: an international journal*, 10:7.

[6] Adamatzky, A. and Jones, J. (2012). Emergence of self-organized amoeboid movement in a multi-agent approximation of physarum polycephalum. Springer. [Figura]. Recuperado de: https://www.semanticscholar.org/paper/Emergence-of-Self-Organized-Amoeboid-Movement-in-a-Jones-Adamatzky/1196b7e11edfde973ea832974a99b3285a74749b.

[7] Adamaztky, A., Chávez, R., Juárez, G., and Oliva, L. N. (2018). Collective behaviour from simple cellular automata rules in low-cost small robots: the ebots. *Natural Computing: an international journal*, pages 1–13.

[8] B., R. (2006). Slime mold reproduction. [Figura]. Recuperado de: http://bioweb.uwlax.edu/bio203/2010/renner_brad/reproduction.htm.

[9] Eric (2016). Game of life. [Figura]. Recuperado de: http://mathworld.wolfram.com/GameofLife.html.

[10] Fabra, U. P. Descubre ideas sobre hongos. [Figura]. Recuperado de: https://www.pinterest.com.mx/pin/552183604286154239/.

[11] Felde, C. (2010). C++ vs java performance; it's a tie! [Figura]. Recuperado de: https://blog.cfelde.com/2010/06/c-vs-java-performance/.

[12] Frutis, I. and Huidobro, E. (2013a). Micología básica. pages 7–9.

[13] Frutis, I. and Huidobro, E. (2013b). Micología básica. page 14.

[14] Jones, F. (2015a). From pattern formation to material computation: Multi-agent modelling of physarum polychepalum. page 16.

[15] Jones, F. (2015b). From pattern formation to material computation: Multi-agent modelling of physarum polychepalum. pages 33–59.

[16] Jones, F. (2015c). From pattern formation to material computation: Multi-agent modelling of physarum polychepalum. page 111.

[17] Jones, F. (2015d). From pattern formation to material computation: Multi-agent modelling of physarum polychepalum. page 261.

[18] Juárez, G. (2005). Thesis supervised, workpapers, and stuff elaborated by students. http://uncomp.uwe.ac.uk/genaro/Papers/Thesis.html.

[19] Nagaki and Toshiyuki (2000). Intellingence: Maze-solving by an amoeboid organism. [Figura]. Recuperado de: https://www.nature.com/articles/35035159.

[20] Oliva, L. and Juárez, G. (2016). Artificial life robotics lab. http://www.comunidad.escom.ipn.mx/ALIROB/Welcome.html.

[21] Sheetz, L. (2012). Network science. [Figura]. Recuperado de: https://www.pinterest.com.mx/lasheetz/network-science/.

[22] Stormberg, J. (2012). If the interstate system were designed by a slime mold. [Figura]. Recuperado de: https://www.smithsonianmag.com/science-nature/if-the-interstate-system-were-designed-by-a-slime-mold-93309928/.

[23] Yong, E. (2006). Slime mould attacks simulates tokyo rail network. [Figura]. Recuperado de: http://phenomena.nationalgeographic.com/2010/01/21/slime-mould-attacks-simulates-tokyo-rail-network/.

[24] Yukio, G., Haruna, T., Nizato, T., and Shirikawa, T. (2008a). Minimal model of a cell connecting amoebic motion and adaptive transport networks. *Journal of Theoretical Biology*, pages 659–667.

[25] Yukio, G., Haruna, T., Nizato, T., and Shirikawa, T. (2008b). Minimal model of a cell connecting amoebic motion and adaptive transport networks. *Journal of Theoretical Biology*, page 661.