Instituto Politécnico Nacional
Escuela Superior de Cómputo
ESCOM

Trabajo Terminal

"Modelado del Physarum Polycephalum con autómatas celulares para el enrutado de robots mensajeros"

2021 - B013

Presenta:
Ramírez Olvera Guillermo
Sánchez Méndez Edmundo Josué

Directores:

Dr. Juárez Martínez Genaro Dr. Oliva Moreno Luz Noé

México CDMX, 09 de enero de 2023

Instituto Politécnico Nacional
Escuela Superior de Cómputo
Subdirección Académica

Trabajo Terminal No. 2021 - B013 09 de enero de 2023

Documento Técnico

"Modelado del Physarum Polycephalum con autómatas celulares para el enrutado de robots mensajeros" Trabajo Terminal No. 2021 - B013

Presentan:
Ramírez Olvera Guillermo[1]
Sánchez Méndez Edmundo Josué[2]

Directores:

Dr. Juárez Martínez Genaro Dr. Oliva Moreno Luz Noé

Resumen – En el presente reporte se presenta la documentación técnica del Trabajo Terminal 2021-A013 "Modelado del Physarum Polycephalum con autómatas celulares para el enrutado de robots mensajeros " el cual incluye la información acerca del desarrollo de los prototipos realizados para este trabajo en los cuales se incluye un prototipo de un simulador que trabaja con un modelado del comportamiento del hongo Physarum Polycephalum y un prototipo de robot mensajero haciendo uso del modelado mencionado con anterioridad.

Palabras Clave - Academia de Ciencias de la Computación, Autómatas celulares, Instrumentación, Sistemas Digitales

[1] *e-mail: memo0p2@hotmail.com*

[2] *e-mail: edmundoelpro1@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 derecho 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 podrá obtenerse 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.

Resumen

El presente trabajo terminal propone un nuevo modelado del Physarum Polycephalum el cual como veremos a través de este texto se plantea el uso de un autómata celular para su modelado, así como el tipo de vecindad a usar y el porqué de usar esta vecindad, aunque veremos mas adelante, que este modelado se puede usar con diferentes tipos de vecindades, este modelado nos permite encontrar al menos una ruta dentro de un universo de posibilidades, que aunque la ruta puede no ser la más optima, el modelado nos garantiza encontrar al menos una. Debido a este gran potencial del modelado se plantea el uso de este para pode controlar un robot, el cual es capaz de realizar entregas de paquetes y siendo como tal un sistema embebido móvil, ya que para controlar el robot se hace uso de una Raspberry Pi 4 que es un sistema en chip, ademas de que todo los sistmas digitales que se ocupan residen en dicho robot, también el robot al tener sensores ultrasónicos para buscar obstáculos y gracias al modelado propuesto del Physarum Polycephalum nos permite que el robot pueda adaptarse en tiempo real a los obstáculos que encuentre en el camino y así poder llegar al destino sin importar cuantos obstáculos pueda encontrarse.

1. Introducción

El modelado de la naturaleza ha sido, históricamente, antiguo a la par que interesante, partamos del punto de si se trata de ciencia o ingeniería: un científico se guía por la curiosidad y un ingeniero por la necesidad, el primero preguntaría cómo se formó un valle y el segundo cómo cruzarlo. Algo que crea la ingeniería es propenso a que exista algo mejor, una ley que describe la ciencia es más seguro a que perdure mucho más. En perspectiva, un algoritmo resulta caer en la primera categoría (puede existir uno mejor), sin embargo, en este caso es creado por la curiosidad más que la necesidad. Y si lo analizamos, es natural el uso de algoritmos, pues tienen un lenguaje rico y expresivo: el nuestro. Una hoja de un árbol, por ejemplo, se maneja por el software de la naturaleza, por ello profesionales de la computación observan a la naturaleza como algo complejo y cada vez menos ajeno a su área de estudio. El motivo recae en que, si se logra modelar un organismo, se obtienen ciertas ventajas del organismo, y en nuestro caso, se sabe que el Physarum es muy bueno obteniendo rutas, por ello se parte el esfuerzo por modelarlo y determinar qué tan bueno es para otorgar rutas en un sistema discreto [8].

El Physarum Polycephalum ha sido motivo de estudio en diversas investigaciones, por su parte, la comunidad científica se pregunta cómo es que un organismo unicelular gigante puede realizar tareas consideradas complejas en el ramo de la computación, dado su gran tamaño, la complejidad de su flujo interno y su comportamiento motor-sensorial, el mayor de los intereses se halla en sus propiedades computacionales complejas que exhibe en su forrajeo, crecimiento y adaptación. Este organismo no solo encuentra rutas óptimas en los tres procesos mencionados, sino que también lo hace encontrando su comida y evadiendo peligros para su supervivencia [17].

Diversas investigaciones han utilizado al Physarum para encontrar rutas de cómo se dispersan poblaciones por las vialidades del país [5], por ejemplo, de ciudades grandes de

África, Australia, Bélgica, Brasil, Canadá, China, Alemania, Iberia, Italia, Malasia, México, Reino Unido, Países bajos [3] , o para ver las rutas de migración en Estados Unidos[6]. Ambos manejan un simulado con objetos tangibles, es decir, imprimen un mapa a escala de la zona de estudio o tienen papel con la forma de la región, ponen el organismo en los puntos donde se conoce que comienzan la distribución, por ejemplo, en la frontera con México y esperan a que cubra la superficie aproximadamente 5 días. Ambas investigaciones han arrojado resultados esperados con respecto a la realidad. Por ejemplo, en [6] los autores notan que en Chicago se forma un punto más aglomerado que el resto, haciendo que concuerde con lo que pasa actualmente.

Se tiene como principal objetivo modelar el comportamiento del Physarum Polycephalum con autómatas celulares con la finalidad de explorar su uso como proveedor de rutas a un robot mensajero, la expansión de nuestro conocimiento acerca de nuestros intentos por imitar la naturaleza, y en especial a este organismo, son motivo suficiente para calificarlo de relevante en el marco actual, considerando que pertenecemos a una institución donde predominan las ciencias exactas, y para ser concretos, una rama de las ciencias de la computación, en este caso, la computación no convencional.

Existe una amplia variedad en las aplicaciones del modelado del organismo, sin embargo, en lo que concierne a este trabajo solo se enfocará en determinar qué tanto aporta como proveedor de rutas alternas al robot mensajero en el envío de mensajes en el edificio de gobierno de la Escuela Superior de Cómputo del Instituto Politécnico Nacional.

El presente trabajo pretende ser un pequeño bloque en la aportación de aplicaciones conocidas del modelado del Physarum Polycephalum mediante autómatas celulares, sin embargo, resulta importante realizar una aportación, que, aunque, pequeña, pueda resultar útil en futuras investigaciones, en alcances más grandes, como lo podría ser considerar un espacio tridimensional, e incluso en otras escuelas. Creemos que tenemos un compromiso con la sociedad por todas las cosas que nos ha brindado en nuestra educación, por lo que hemos decidimos tomar esta ruta de investigación de la computación no convencional para

agradecer por todas las oportunidades que nos ha brindado no solo nuestra alma máter sino, la sociedad en sí.

1.1. Objetivo general

Modelar el comportamiento del organismo Physarum Polycephalum usando autómatas celulares para otorgar rutas alternas a un robot mensajero dentro del edificio de gobierno de la Escuela Superior de Cómputo del Instituto Politécnico Nacional; con la finalidad de enviar paquetes menores a 5 kilogramos entre las personas que laboran en ese edificio.

1.2. Objetivos específicos

- Explorar las configuraciones del autómata celular en su forma bidimensional del Physarum Polycephalum.
- Diseñar un robot que pueda mover paquetes menores a 5 kilogramos entre dos puntos.
- Enviar paquetes en el edificio de gobierno de la Escuela Superior de Cómputo del Instituto Politécnico Nacional usando el robot desarrollado.

.

2. Marco teórico

2.1. Physarum Polycephalum

La manera informal de llamar a los organismos eucariotas que suelen ser de pequeño tamaño, donde usualmente no tienen relación, pero se caracterizan por expandirse en el terreno donde estén, es *slime mold* o *slime mould* (moho de fango), que eran considerados parte del reino Fungi, pero, debido a algunas investigaciones encontraron que no tienen relación con estos y ahora son considerados dentro del reino Protista.

Existen tres grandes grupos que clasifican a los slime mold: los plasmodios, los celulares y los Labyrinthulomycetes, donde la familia Physarum pertenece a la de los plasmodios, que, simplificando la idea, consisten en miles de núcleos diploides que conforman un enorme organismo unicelular, creados cuando las células flageladas de un enjambre se fusionan, donde resulta una inmensa bolsa de citoplasma [34]. Los plasmodios han sido usados en diversos estudios debido a su tamaño, ya que con poca magnificación de imágenes se puede ver el flujo de citoplásmico (el movimiento del contenido de las células).

En específico, el Physarum Polycephalum tiene un color amarillo y se le conoce como blob por la película The Blob de 1988, puede expandirse en un área grande y tiene la capacidad de moverse si las condiciones lo permiten, tiene un ciclo de vida complejo con fases haploides y diploides. Además, responde a las condiciones del ambiente, por ejemplo, se vuelve más resistente si las condiciones son menos favorables y espera hasta que mejoren, por el contrario, condiciones muy favorables hace que muestre un gran movimiento [17].

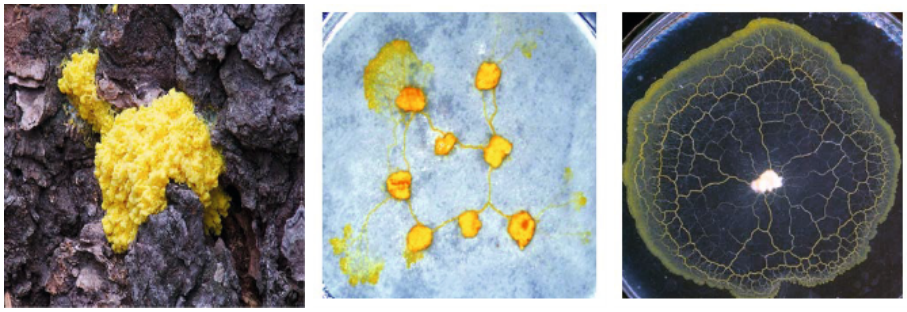


Figura 1: Physarum Polycephalum en la naturaleza y en laboratorio [17].

Su estructura es viscosa y recibe estímulos de diversas fuentes, como la temperatura, la humedad, la iluminación o los niveles de pH. Desarrolla una estructura de ramificaciones a las que se le conoce como plasmodio, como se observa en la figura 1. Esto lo hace para buscar alimento en su entorno, al encontrarlo, entra en un proceso llamado fagocitosis, donde introduce el recurso a su cuerpo celular. Dependiendo de los puntos o nodos donde encuentra comida, cambia la forma de su red de ramificaciones para discriminar zonas que ya exploró y no va a emplear [10].

Bajo condiciones favorables, el Physarum Polycephalum se reproduce formando tallos con esporas, estos lucen de forma esférica o como paletas de caramelo, y, con algo de tiempo, estos tallos liberan las esporas, cada una se vuelve en una célula y cuando estén en un estado flagelado, entonces de nuevo pasa a ser un plasmodio, o bien, sean ameboides, ambos casos pasan a ser cigotos y por mitosis vuelven a conformar un plasmodio que busque alimento y repita el ciclo, por ello es considerado un ciclo de vida complejo, ya que tiene una fase donde es una sola célula, se alimenta, crea esporas, las libera, cada una es una célula, tienen dos caminos, ameboide o célula flagelada y pueden cambiar entre ellos, por singamia se unen y pasa a ser plasmodio, todo este gran ciclo es ilustrado en la figura 2 [26].

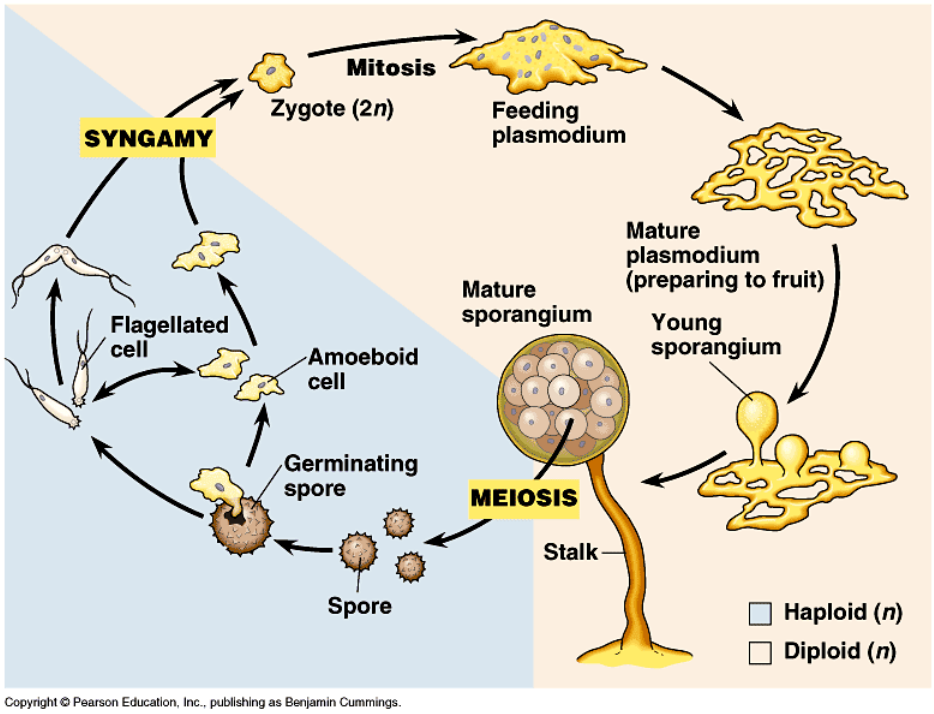


Figura 2: Ciclo de vida de los slime mold [26].

Resumiendo algunas características generales del Physarum las podemos ver en la siguiente tabla.

Característica	**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.

Tabla 1: Características generales del Physarum Polycephalum.

Ahora que conocemos a grandes rasgos el funcionamiento del Physarum Polycephalum, podemos delimitar específicamente las propiedades que se desean modelar para aprovechar su algoritmo de enrutamiento con ramificaciones para la búsqueda de alimento.

2.1.1. Physarum Polycephalum desde la perspectiva computacional

Hemos hablado de su comportamiento y sus respuestas dinámicas al entorno que habita, englobando la idea, se puede partir en dos pedazos: la búsqueda en su medio en un área máxima, es decir, la construcción de ramificaciones; y la eficiencia del transporte de los nutrientes, es decir, la destrucción de ramificaciones innecesarias. Esto, en general, es la idea de trasladar la esencia de un organismo del mundo real al mundo discreto computacional, en otras palabras, se trata de computación no convencional, que abordaremos con más detalle en el apartado de autómata celulares.

Existen diversas investigaciones y enfoques a lo largo de la historia de la investigación del comportamiento del Physarum Polycephalum donde se ha usado tanto en la vida real con mapas a escala, así como en modelados con el uso de autómatas celulares con técnicas muy variadas donde podemos encontrar las siguientes investigaciones, como lo mencionamos en la introducción:

Approximating Mexican highways with slime mould: Se trata de una investigación donde diversos doctores se propusieron aproximar las principales vialidades de México mediante un mapa a escala, considerando a 19 ciudades del país como nodos de alimento para el Physarum Polycephalum. Los investigadores encontraron que las redes que construyó el Physarum fueron similares a las principales vialidades que existen a lo largo del país, aunque, hubo algunos pedazos en donde no cuadró porque no consideraron el relieve del país en su maqueta [5].

Recolonisation of USA: Slime Mould on 3D Terrains: Siguiendo la línea de investigación anterior, los doctores Adamatzky y Juárez plantean obtener las rutas de migración procedentes de México a Estados Unidos mediante el uso de un mapa a escala que considere el relieve del país, por lo que imprimen un Estados Unidos a escala con los detalles de las montañas del país. Al igual que la investigación anterior, los puntos importantes tuvieron comida para el hongo, con ello encuentran rutas que concuerdan con los puntos en los que, hoy en día, se puede localizar gente de habla hispana [6].

Slime mould attacks simulates tokyo rail network: siguiendo la misma idea, en Japón, se emplea el hongo con el fin de localizar las rutas que dibujaría el Physarum Polycephalum en un terreno basado en la ciudad de Tokio, y, de forma similar, se utilizaron nodos con comida para el plasmodio, además, utilizaron sustancias para poner en peligro al organismo y observar si podía recordar si algún líquido le hacía daño o no, por ejemplo, cafeína, donde la primera vez le costó trabajo, pero, días después, el Physarum se mostró mucho más decidido a cruzarlo al saber que no le hacía daño [37].

En general, hemos descrito que el organismo en cuestión es bueno en las aproximaciones de ramificaciones a escala de las vialidades de diversas ciudades importantes en el mundo, donde el organismo puede recordar, elegir dependiendo del estado, lo que es considerado como si fuera una computadora biológica al poder efectuar un algoritmo de enrutado de forma natural.

El Physarum Polycephalum puede ser interpretado como un sistema complejo de patrones dinámicos, donde su comportamiento está acotado al crecimiento, movimiento y reducción del área, tomando en cuenta que es relativamente un organismo simple, puede crear sistemas de ramificaciones óptimas en áreas grandes, porque, un problema complejo como el de encontrar rutas entre puntos, que lo pueda resolver un organismo de una sola célula que no tiene ningún sistema nervioso, ha sido motivo de investigación y asombro ante la comunidad científica [17].

2.2. Autómatas celulares

Existen diversas formas de modelar al Physarum Polycephalum computacionalmente, donde, históricamente se comenzó por modelar aspectos biológicos de su comportamiento, principalmente la generación, acoplamiento y las interacciones entre los osciladores dentro del plasmodio. Es hasta años recientes que se ha intentado modelar el comportamiento general del organismo para descubrir más sobre sus capacidades desde el punto computacional [17].

Uno de los enfoques para modelarlo es mediante el uso de autómatas celulares (en adelante los llamaremos CA por su abreviación en inglés), donde Gunji consideró el crecimiento del plasmodio y el movimiento de los ameboides. Más tarde se comenzó a usar un CA hexagonal para aproximar los patrones de crecimientos del plasmodio, donde sus resultados fueron buenos, pero, no se apreciaba las adaptaciones morfológicas del organismo. De la misma forma, se ha tratado de usar algoritmos ya conocidos como el de optimización de colonia de hormigas que dio resultados similares a los que el Physarum en la vida real mostró [17].

Un autómata celular es un sistema dinámico discreto que utiliza reglas de evolución en una cuadrícula o reja [10], las celdas de esta cuadrícula contienen valores que representan estados para indicar cosas que deseemos [35], por ejemplo, podemos usar colores para indicar que dada una cuadrícula en una celda tengamos el color verde, entonces significa que existe una célula en esa posición, o, si está en blanco, está vacía. De una forma más forma, su forma más básica consiste en una 4-tupla A = (S, N, Q, δ), donde [14]:

- A es el autómata.
- S es el espacio, cuadrícula o reja en donde existe, lo forma mínima es unidimensional.
- N es una vecindad, representada por posiciones de celdas, usualmente los más cercanos.
- Q es un conjunto de estados, para ver resultados interesantes debe tener al menos 2.
- δ es una función donde δ: $Q^N \rightarrow Q$, es decir, el valor de una celda se determina por sus vecinos en un tiempo discreto, toma los estados de N celdas, y lo mapea a un estado.

Su origen se remonta a la década de 50's entre las primeras personas que los investigaron podemos encontrar a von Neumann, fueron propuestos como un modelo a los sistemas biológicos, aunque recibieron varias críticas y hasta hoy en día siguen siendo motivo de

investigación por lo que podemos encontrar muchas variantes de autómatas celulares [35] [14].

Aunque los avances en su comienzo no fueron muy prometedores, Wolfram decidé investigar a los autómatas unidimensionales y presenta, en 2002, su libro *A new kind of Science* muestra todas las 256 combinaciones de vecindad adyacente 1 de autómatas unidimensionales y crea una nomenclatura de reglas, y para ilustrar cómo funcionan tomaremos el ejemplo de la regla 30 [35]:

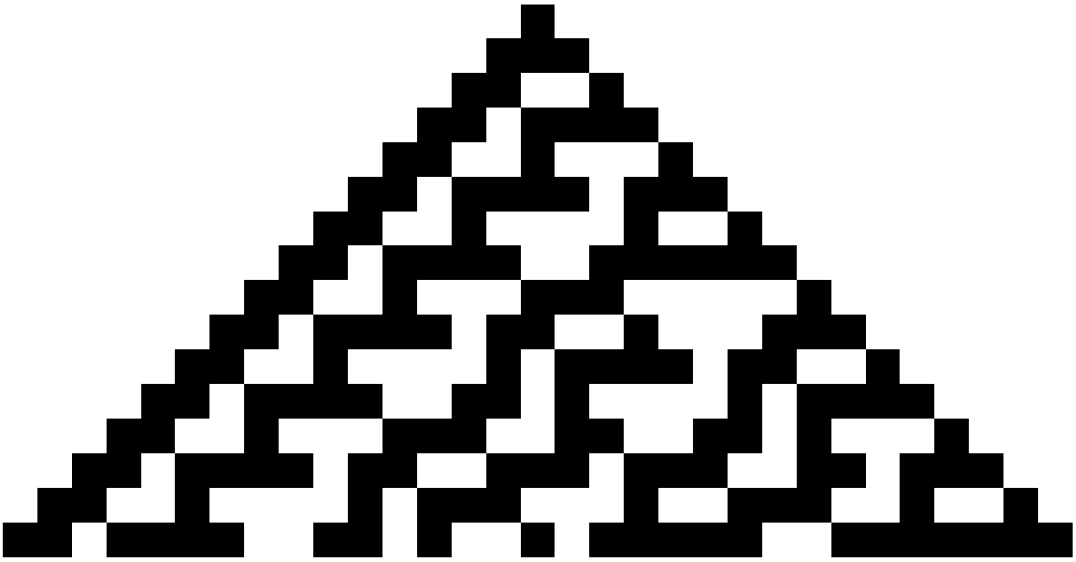


Figura 3: Evolución de una configuración inicial simple con la regla 30. Elaboración propia.

En este ejemplo, cada renglón de la cuadrícula representa un tiempo discreto en la evolución, notemos que solo tenemos una celda blanca al inicio, y, conforme iteramos en tiempo discreto, vamos teniendo más y más celdas blancas, en la parte superior se muestran las reglas que sigue, tomando las celdas adyacentes nos quedan tres celdas, pueden estar de blanco (que representa un 1) o no (un 0), observemos que tomando solo las tres celdas de arriba en cada regla vemos que es un número: 8, 7, 6, 5, 4, 3, 2, 1 y 0, y que la celda de abajo indica si entrega una celda blanca o no (de nuevo, 1 y 0), en la regla 30 tenemos que (B representa blanco, y N, negro):

- BBB → 111 → 7 produce N → 0
- BBN → 110 → 6 produce N → 0
- BNB → 101 → 5 produce N → 0
- BNN → 100 → 4 produce B → 1
- NBB → 011 → 3 produce B → 1
- NBN → 010 → 2 produce B → 1
- NNB → 001 → 1 produce B → 1
- NNN → 000 → 0 produce N → 0

Acomodando lo que entrega: 0001 1110 y al pasarlo a decimal obtenemos 30, por ello el nombre de la regla y que mostrada 256 combinaciones (2^8 es 256), observemos que en el segundo renglón hallamos 3 celdas blancas, la primera se crea a partir de NNB, la segunda a partir de NBN y la tercera a partir de BNN, esto nos indica que se expandirá a lo largo de la cuadrícula, además, en el resto de las iteraciones podemos observar que hay patrones dentro de las blancas.

2.2.1. Vecindad de los autómatas

Retomando lo mencionado en la definición, la vecindad es una región utilizada para evaluar una celda en específico, de forma más concreta, es una región de celdas adyacentes a una posición relativa, esa posición relativa es conocida como célula central o de referencia. Existen diversas formas de vecindad donde podemos encontrar las siguientes:

- Vecindad de von Neumann: dada una cuadrícula de 3x3, la celda central siendo la de referencia, toma en cuenta sus adyacentes e ignora las esquinas, forma una cruz y es usado en autómatas bidimensionales.
- Vecindad de Moore: a diferencia de la von Neumann, esta vecindad utiliza también las esquinas al evaluar.
- Vecindad hexagonal: en lugar de usar una reja cuadricular, se utiliza una hexagonal, se usan los seis hexágonos adyacentes para evaluar.

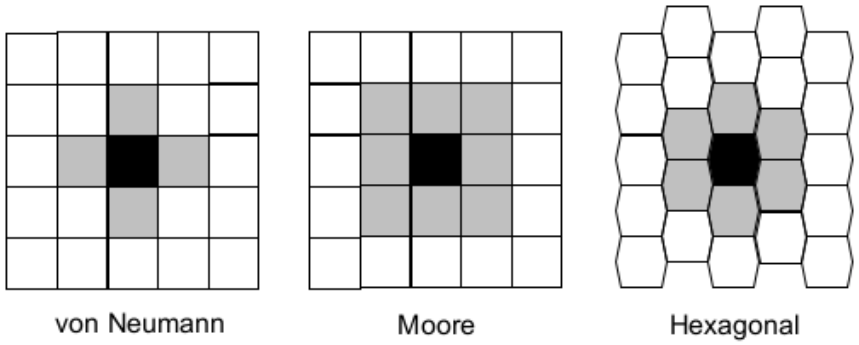


Figura 4: Vecindad de von Neumann, Moore y hexagonal, respectivamente. Elaboración propia.

Dependiendo del problema, y del hecho de que es más fácil usar von Neumann que el resto, las vecindades que presentamos tienen ventajas una sobre otra que abordaremos más tarde.

2.2.2. Entradas y salidas de los autómatas celulares

Existen diversas configuraciones en los autómatas bidimensionales que crean patrones en los ciclos de tiempo discreto, una de las fuentes de ejemplos más citados se encuentra en el Juego de la Vida de Conway, y reciben nombres especiales de acuerdo con lo que parezcan en la vida real, con los pocos pixeles que suelen ocupar, por ejemplo, un deslizador (*glider*, se suele usar el nombre en inglés para las estructuras), se trata de una estructura que tiene un ciclo y avanza por la rejilla a alguna dirección, mientras que una vida estática consiste en una estructura que no cambia sin importar las iteraciones en el tiempo discreto, mientras que los osciladores consisten en estructuras cíclicas que se quedan en su posición global en la rejilla [10].

Además, la rejilla cuenta con valores predefinidos para ver cómo evolucionan al avanzar en el tiempo discreto, esto se le conoce como configuración inicial, y, específicamente en el caso del modelado del Physarum Polycephalum se usa para determinar obstáculos, o si es parte del organismo, si está explorando, si debería de evitar tocar algunas celdas al

expandirse entre muchos otros efectos que se les puede dar, con ello podemos obtener rutas a partir de una réplica en una cuadrícula discreta de algún lugar real en donde queramos que un robot mensajero exista.

2.2.3. Autómata celular de 2 dimensiones: El Juego de la Vida

Los autómatas celulares pueden existir en un espacio de cierta dimensión, como vimos anteriormente, donde los casos comunes son de 1 ó 2 dimensiones, el Juego de la Vida fue desarrollado por John Horton Conway, este es un muy buen ejemplo de un AC de 2 dimensiones mostrado como un tablero de ajedrez empleando la vecindad de Moore definiendo las siguientes reglas, que aunque parezcan reglas simples, estas reglas presentan un comportamiento complejo:

- *Supervivencia*: Para que una célula siga vida, tiene que tener 2 ó 3 vecinas viva.
- **Nacimiento**: Si una célula esta muerta, pero si se encuentran 3 células vecinas vivas esta célula revive.
- **Muerte**: Si una célula esta viva, 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.

En la figura 5 podemos ver la configuración inicial la cual sera usada para aplicar la reglas del juego de la vida y en la figura 6 podemos ver como acabo la configuración inicial después de que se aplicaron las reglas, mencionar que los cuadrados blancos representan una célula viva y los cuadrados en negro representa una célula muerta.

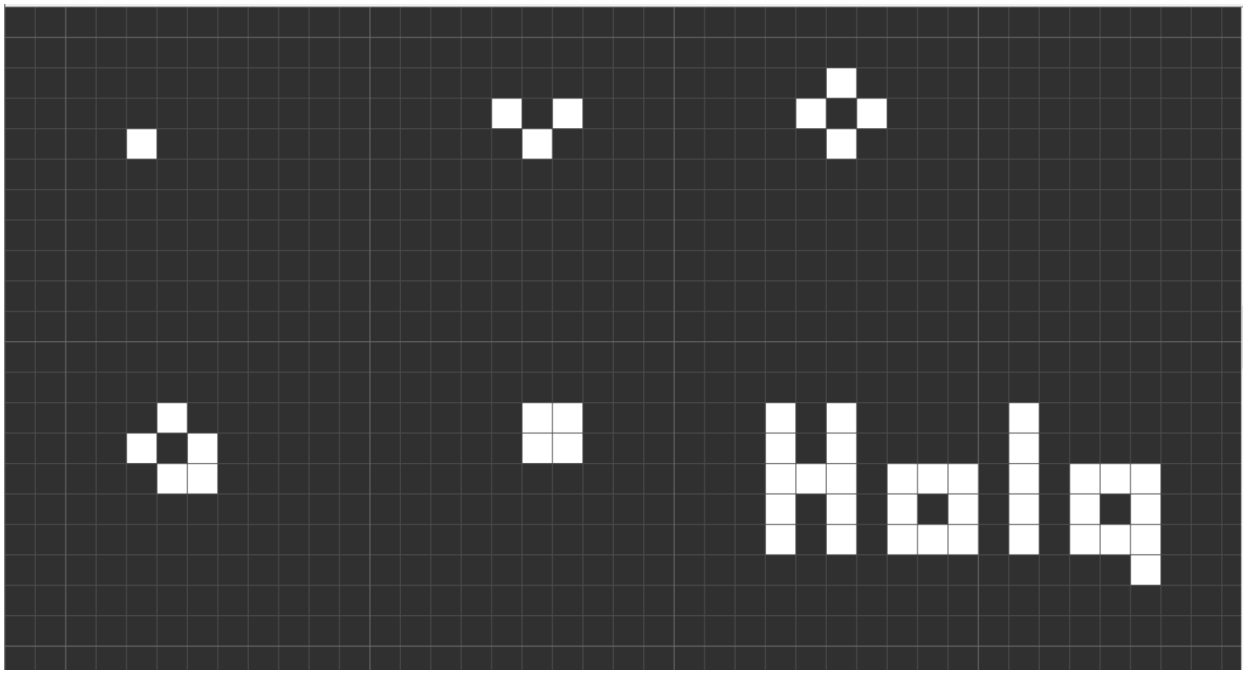


Figura 5: Configuración inicial para usar la regla del juego de la vida. Elaboración propia.

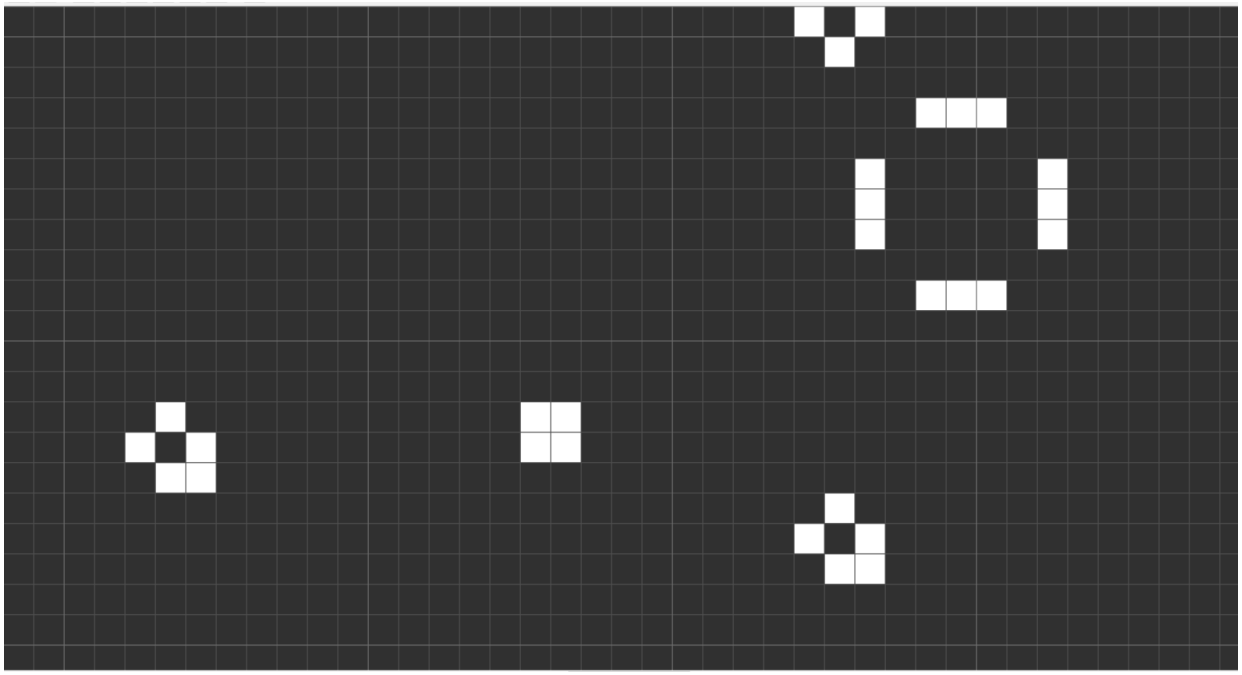


Figura 6: Final de la configuración inicial usando la regla del juego de la vida. Elaboración propia.

2.2.4. Entropía de Shannon

La entropía de Shannon es una de las métricas más importantes en la teoría de la información. La entropía mide la incertidumbre asociada con una variable aleatoria, es decir, el valor esperado de la información en el mensaje (en informática clásica se mide en bits).

El concepto fue introducido por Claude E. Shannon en el artículo *"A Mathematical Theory of Communication"* (1948). La entropía de Shannon permite estimar el número mínimo promedio de bits necesarios para codificar una cadena de símbolos en función del tamaño del alfabeto y la frecuencia de los símbolos y en donde la formula base es la siguiente:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

Formula base de la entropía de Shannon. (1)

En donde b es la base del logaritmo utilizado. Los valores comunes de b son 2, el número de Euler e y 10. La situación de máxima incertidumbre nos genera que sea más difícil predecir el resultado siguiente, esto es debido a que hay probabilidad uniforme. La entropía, entonces, solo puede disminuir a partir del valor asociado con la probabilidad uniforme. El caso extremo es donde un estado tiene probabilidad de 1 ya que ahí no hay incertidumbre, es decir la entropía es cero.

Tomando en cuenta el trabajo desarrollado por Andrew Wuensche en su libro titulado *"Exploring Discrete Dynamics"* [36] en la sección 31.4 él desarrolla una formula para poder calcular la entropía de Shannon de la distribución de frecuencias de entrada en patrones de espacio-tiempo, dicha formula es la siguiente:

$$S^t = -\sum_{i=1}^{2^k} \left(\frac{Q_i^t}{n} \log \left(\frac{Q_i^t}{n} \right) \right)$$

Entropía de Shannon a utilizar. (2)

En donde Q_i^t es la frecuencia de i en tiempo t, n es el tamaño de la red, y k el tamaño de la vecindad.

2.3. Robots mensajeros

Si bien, hemos hablado de las características del Physarum y de la forma en que se modela, consideramos a esta parte el *proveedor*, lo siguiente a tratar, es justamente la parte que consume estas rutas generadas a partir del modelo, un robot mensajero.

Hoy, en nuestro día a día, diversos productos son creados, trasladados, inspeccionados, vendidos, entre muchos otros verbos, mediante la ayuda de hardware y software alrededor de todo el mundo. Cada vez, las empresas decantan por utilizar el tiempo de las personas en algo más valioso que realizar tareas repetitivas. Un ejemplo simple lo podemos observar al momento de pagar por nuestros artículos en el carrito, donde tenemos la elección de tomar la tradicional fila, poner en la banda los elementos a comprar, que nos atienda alguna persona, nos entregue la cuenta y procedamos a pagar, o bien, un autoservicio, donde presentamos nuestros productos, pagamos, y nos marchamos, sin otro humano de por medio.

Una propuesta que se ha tenido es el Scout, un gran reto a resolver por parte de Amazon. Partamos de la idea de que se quiere realizar envío en algún área, el primer problema sería obtener una ruta, un problema que no solo lleva años investigándose, se tratan de décadas de investigación, con este problema ya se vuelve lo suficientemente complejo el sistema. Sin embargo, el Scout debe saber moverse entre personas, mantener íntegro su contenido, actuar conforme el clima, adaptarse a los cambios de suelo, obstáculos de diversas formas como coches, cruces, etcétera [11].

Por ello, el equipo de ciencia e ingeniería de Amazon toma tres pilares para el funcionamiento del Scout: Navegación, Percepción y Simulación, cada uno de una gran difícultad, para ello tres grandes contribuidores compartieron sus opiniones y experiencias en una entrevista [11]:

2.3.1. Navegación: Los detalles son importantes

Paul Reverdy, doctor en Ciencia Aplicada, desde el comienzo determinó que no se podían fíar de algo tan impreciso como el GPS para un robot mensaje como lo es el Scout, debía de implementar algún otro método, aunque el camino no les parecía muy claro al comienzo. *"Simplemente no ofrecen la precisión en detalle que requerimos, y no podemos asegurar que estén disponibles todo el tiempo"*[11].

"Scout debe tomar demasiadas decisiones, algunas son discretas, como ¿puedo cruzar la calle?, y otras son mucho más complicadas, ¿puedo pasar entre esa pared y aquél cubo de basura?", es por eso que Reverdy optó por tener mapas muy detallados de las zonas en las que el robot opera [11]. Justo como los cartógrafos, ciertamente, el suelo cambiará de muchas formas, pero, hay constantes que permanecen por más tiempo, como una casa, una cerca, el resto lo mantiene actualizado el propio robot en sus trayectos. Reverdy sostiene que **al tener un mapa detallado, el robot podrá navegar y crear las rutas con mucha más facilidad**.

2.3.2. Percepción: ¿Qué es manejable y qué no?

Hamed Pirsiavash, el científico encargado de otorgarle independencia al robot, utilizó Aprendizaje Automático y Visión Computacional para que el robot determinara si dado un piso, puede manejar sobre él o no [11]. A primera vista podríamos pensar que es relativamente sencillo determinar si un piso es manejable o no, pero, entran en juego variables como el estado de un semáforo, es decir, el Scout debe poder ver a ambos lados si vienen coches, en caso de que no haya semáforo, si lo hay, determinar si puede cruzar, y entonces, podrá decidir si el camino es o no manejable.

> La idea general es muy similar a la de un automóvil autónomo, con la diferencia que el Scout conduce a una velocidad mucho menor, va en la banqueta y no puede atropellar a la gente, esto tiene ventajas, pero tiene la desventaja de que existen muchos más obstáculos, al final, la vía de coches se diseñó para

> ellos... las banquetas, no tanto para el Scout [...] es gracias a los inmensos conjuntos de datos y la visión computacional que avanza prácticamente día a día, que el Scout puede proceder a avanzar en las calles con obstáculos que incluso no preveíamos, por ejemplo, hay un registro de un aro de baloncesto que calló en la mitad de la calle [11].

2.3.3. Simulación: Medir y entrenar

El entrenamiento del robot no solo fue realizado con la ayuda de los conjuntos de datos mencionados en el punto anterior, sino que también se creó una *'sandbox'* para que el equipo pudiera ver cómo reaccionaría ante escenarios con la capacidad de poder cambiar varios detalles como el clima, poner hojas secas y determinar si estaba preparado el Scout o tendrían que entrenarlo.

De acuerdo con Benjamin Kunsberg, científico encargado de crear el mundo virtual para el Scout, menciona que nivel de detalle que pueden controlar es hasta poner cómo es el pasto [11]. Este método es combinado con fotografías de suelos y datos como el clima para darle mayor precisión al entrenamiento del Scout, ya que fiarse solo de la simulación sería ideal solo si podemos responder a ¿es fiel la versión simulada a la real?, lo cual es muy complicado de responder en ciertos casos, por ello ambos equipo trabajan en colaboración para continuar aumentado la capacidad del Scout.

Figura 7: Amazon Scout [11].

2.4. Sistemas embebidos

Como se ha notado en el punto anterior, una empresa que se ha tomado de forma rigurosa el aprovechar cada vez más la gran utilidad de los robots y la automatización es Amazon, quien ha implementado robots para mantener en orden su inventario, y, hace pocos años, lanzaron al mercado la idea de Amazon Go, un local donde los usuarios se identifican con su cuenta de Amazon (o una tarjeta de crédito ligada a su cuenta, o la palma de su mano), entran, toman los artículos que desean, y se marchan, sin más personas a la vista, al salir, se presenta el saldo exacto de lo que el usuario tomó y se le cobra [16].

Exceptuando a una sola persona que se encarga de prohibir la venta de alcohol a menores, este proceso no tiene otro individuo que el propio consumidor, ahí es donde entra la tecnología, diversas cámaras, sensores, software que corre de forma contínua para identificar si se ha tomado un jugo, si dos personas se cruzan, si se devuelve el artículo, todo esto ocurre tras bambalinas con el fin de cobrar de forma exacta una vez que el cliente decide marcharse [16];

Esta es la idea clave de un sistema embebido: es un conjunto de software o hardware que

se encarga de ciertas tareas en específico, y que, generalmente, se encuentra dentro de otro sistema [32]. Por ejemplo, la tienda en conjunto, se trata de un gran sistema, que contiene y se apoya de diversos sistemas para que funcione, como el sistema de la entrada que escanea el teléfono, determina si puede entrar o no, generalizando, su función es el acceso. Mientras que las cámaras junto con el aprendizaje automático de fondo, determinan si cierta persona tomó un producto y lo mantienen en su cuenta.

Formalizando más la definición, decimos que un sistema embebido es un sistema computacional que cumple una aplicación específica, puede ser un producto final o estar incrustado en un sistema de mayor complejidad, suele estar optimizado para las funciones que cumple y usa los recursos necesarios. Responde a los cambios mediante la ejecución indeterminada de un software y tiene el hardware necesario para poder cumplir sus funciones de forma independiente. Entre las diversas áreas de aplicación, podemos encontrar sistemas embebidos en [32]:

- Computación móvil: usualmente un teléfono se compone de pequeños sistemas conectados mediante el uso de interfaces, por ejemplo, una compañía se encarga de crear las cámaras para varios modelos, y cada dueño del modelo usa interfaces para comunicarse con ese dispositivo, esto permite diversidad, disminuye costos y fomenta la competitividad.
- Internet de las cosas: como aplicaciones o asistentes que nos permiten modificar el estado de ciertos objetos como disminuir la intensidad de la luz, cerrar las cortinas, prender la televisión.
- Ciencia de Datos y Aprendizaje Automático: ya que es común crear o usar sistemas que funcionen como provedores de información para su futuro análisis, por ejemplo, una cámara.

El Scout, en primera instancia, es en realidad un sistema compuesto de sistemas más pequeños (pero aún bastante complejos), donde podemos ver que sigue un patrón de

fragmentar un problema grande, esto nos puede resultar de bastante utilidad, toda la información que hemos presentado aporta de forma considerable y en diversos ámbitos al desarrollo de la propuesta. Los sistemas embebidos de forma general, tienden a ser baratos por su generalización, se diseñan para que estén en múltiples casos de uso, ademas de que los sistemas embebidos modernos contienen las siguientes características:[33]

- Hardware similar a una computadora: uP, memoria, elementos de entrada y salida.
- Software de aplicación.
- Un sistema operativo de tiempo real (RTOS).
- Un sistema operativo basado en Linux.

Estas características nos son de gran utilidad para poder realizar el trabajo que se propone y como veremos en el capitulo dedicado solamente en el robot mensajero MemOso se usará principalmente lo que se conoce como un *System On a Chip* (SoC) que en nuestro caso será la tarjeta Raspberry Pi 4 B+. Esta tarjeta como mencionamos anteriormente tienen un sistema operativo basado en Linux, de forma general la misma empresa creadora de este SoC nos proporciona varias opciones de sistemas operativos todas basadas en Debian GNU/Linux, pero en todas al ser Linux tenemos las mismas opciones de trabajo, en especial lo que se comparte es el Sistema de arranque, el cual nos dedicaremos a explicar un poco ya que se usara para el funcionamiento del robot.

2.4.1. Sistema de arranque en Raspberry Pi 4

De forma general el sistema de arranque de una Raspberry Pi 4 funciona de la siguiente manera, también lo podemos ver ilustrado en la figura 8:

- El procesador BCM2837 inicia.

- Se determina el modo de inicialización (Boot mode). Los modos de inicio son SD1, SD2, NAND (SMI-Secondary Memory Interface), SPI, USB, encontrados en el archivo "bootcode.bin".

- Se comienza el arranque.

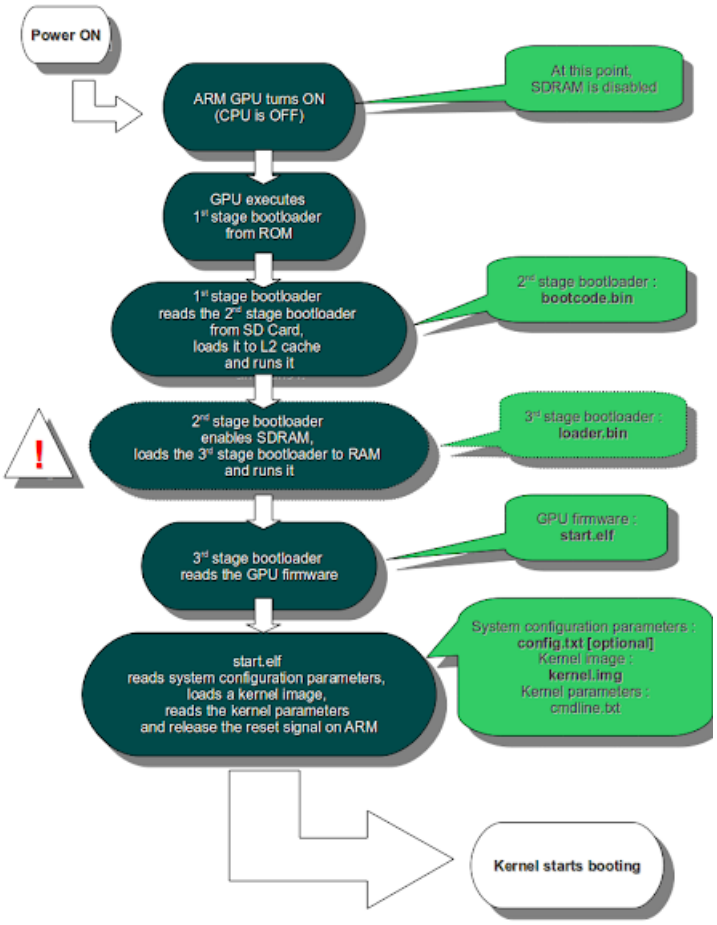


Figura 8: Sistema de arranque en Raspberry[28].

Pero ¿Para que nos interesa saber todo esto? Bueno pues para dar un pequeño contexto del funcionamiento de arranque de nuestro SoC ademas para introducir a nuestro proceso Init, este es siempre el primer proceso en comenzar su ejecución dentro del sistema Linux después de la carga del Kernel, su número de proceso es siempre 1 y se ejecuta como**demonio**. Las funciones principales del proceso Init son las siguientes:

- Es el único proceso que no manda valor de retorno.
- Monta el sistema de archivo raíz. Filesystem/.
- Hereda a los hijos que se quedan sin padre. Estos se encuentran en /proc/a001-.
- Es el encargado de lanzar a todos los demás procesos.
- Lanza los demonios.
- Configura la red. Con ifconfig/ifrouter/Network manager, mediante sus scripts. Se debe colocar el hostname.
- Carga los drivers, incluso los que no vinieron en la distribución original.
- Insertar firmware para alguna tarjeta.
- Apagar el sistema y sus procesos.
- Lanzar ambiente gráfico. X11/Wayland.
- Inicia el sistema de loggeo. Xdm/gdm/system-logind.
- Configura las terminales. tty0 – tty6.
- Borra el contenido del directorio /tmp.

Sin embargo, vemos una palabra algo curiosa y esa es **demonios**, así que nos preguntamos ¿Qué es un demonio en la Raspberry? o más generalmente ¿Qué es un demonio en UNIX?

2.4.2. Demonios en UNIX

Un demonio o servicio es un programa que se ejecuta en segundo plano, fuera del control interactivo de los usuarios del sistema ya que carecen de interfaz con estos. El término demonio se usa fundamentalmente en sistemas UNIX y basados en UNIX, como GNU/Linux o Mac OS X.

La forma de ejecutar y sobre todo de programar los demonios depende del sistema de inicio, en los sistemas de inicio podemos encontrar los siguientes:

- **Systemd**. Es un nuevo sistema para la administración de dispositivos, eventos y servicios en GNU/Linux creado por Lennart Poettering.
- **SysVinit**. Es el sistema tradicional de administración.
- **OpenRC**. OpenRC es un sistema de inicio basado en dependencias que mantiene compatibilidad con el sistema proporcionado por el programa init. Usado por Gentoo.

Los demonios Init de muchos sistemas operativos operan en base a niveles de ejecución. Cada nivel de ejecución define un estado de la máquina después del arranque, estableciendo que demonios deben ser iniciados y/o detenidos para alcanzarlo. Por lo tanto, en cada momento sólo puede haber un nivel de ejecución activo. Un sistema Linux tiene varios niveles de ejecución, numerados de 0 a 6. 0, 1 y 6 son niveles de ejecución especiales realizando funciones específicas, en la siguiente tabla podemos ver un poco mas a detalle los niveles de ejecución así como su nombre y la descripción de los mismos.

ID	Nombre	Descripción
0	Alto (Halt)	Enciende o apaga el sistema.
1	Modo usuario único	Permite reparar problemas o hacer pruebas en el sistema. No configura la interfaz de red o los demonios de inicio, ni permite que ingresen otros usuario que no sean el usuario root.
2	Multiusuario	No configura las interfaces de red ni soporta los servicios de red.
3	Multiusuario con soporte de red	Inicia el sistema normalmente sin interfaz de usuario (GUI).
4	No usado/definible por usuario	Para propósitos especiales.
5	Multiusuario gráfico	Similar el nivel de ejecución 3 con GUI.
6	Reinicio	Se reinicia el sistema.

Tabla 2: Niveles de ejecución en Linux.

Ahora hablemos un poco mas de Systemd la cual corresponde a la distribución que ocupa Raspberry, en esencia Systemd es un conjunto de bloques de construcción básicos para un sistema Linux. Proporciona un administrador de sistema y servicios que se ejecuta como PID 1 y comienza el resto del sistema. Algunas características son:

- Proporciona capacidades de paralelización.
- Utiliza sockets y activación Bus D para iniciar servicios. Bus D, es un sistema de mensajes de bus que proporciona una manera fácil de comunicación inter-proceso.
- Ofrece bajo demanda el inicio de demonios.
- Seguimiento de procesos con grupos de control de control de Linux. Cgroup es una característica del núcleo de Linux para limitar, controlar y contabilizar el uso de recursos para un conjunto de procesos.
- Mantiene puntos de montaje y montaje automático.
- Implementa un elaborado sistema de gestión de dependencias basado en un control lógico de los servicios.
- Soporta scripts de SysVinit y funciona como un reemplazo de SysVinit.
- De forma general los Scripts de arranque, están compuestos de tres secciones:
    1. [Unit] - Elementos de configuración común.
    2. [Service] – Opciones de configuración específica.
    3. [Install] - Elementos de configuración común.
- El script de systemd tiene un nombre con extensión .service
- Se coloca en /etc/systemd/system/

Usamos systemctl para la administración de los demonios inicializados durante el arranque. Las opciones más comunes son:

- sudo systemctl status demonio.service
- sudo systemctl start demonio.service
- sudo systemctl stop demonio.service
- sudo systemctl reload demonio.service
- sudo systemctl enable demonio.service
- journalctl -f permite ver el archivo log

A continuación se relata la forma en que se trabaja la idea y cómo se dividen los capítulos subsecuentes.

3. Estado del arte

3.1. Physarum Polycephalum

En esta sección veremos algunos de los modelos en donde se trata de simular las características del Physarum Polycephalum en donde su principal uso es el de crear rutas entre puntos, nos centraremos en explicar el funcionamiento básico de estos modelos, los modelos de los que hablaremos son principalmente 4, de los autores Adamatzky, Jeff Jones, Gunji y de nuestro compañero Yaír. Mencionar que mas adelante en este documento se encuentra una tabla comparativa de los modelos aquí expuestos con el modelo que se desarrollo, ademas de comparar nuestro modelado mediante la replicación de algunas de las pruebas que los autores previamente mencionados realizaron.

3.1.1. Modelado de Andrew Adamatzky

Para la comparación de este modelo se utilizará como prueba la red de metros de Tokio como se muestra en la figura 9.

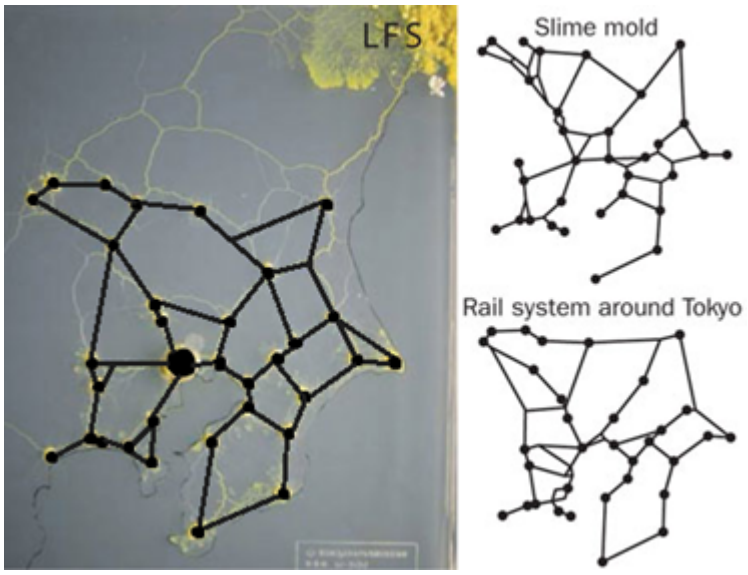


Figura 9: Modelado del sistema de metro de Tokio con el Physarum Polycephalum [4].

El modelo de Andrew Adamatzky [18] se basa en gran parte en los campos de atracción por parámetros que emplea el organismo para trazar sus rutas en función de puntos de interés, la vecindad que este modelo emplea es la de Moore, ademas de que en cada generación donde la célula central extrae valores tipo químico atractor de sus vecinos para saber que tan fuerte o débil es ese punto de interés y con base a ello poder redefinir sus rutas. El resultado de la simulación de la red de Tokio se puede apreciar en la figura 10.

Figura 10: Modelado del sistema de metro de Tokio con el modelo de Adamatzky [18].

3.1.2. Modelado de Jeff Jones

Jeff Jones [17] propone un modelo en la distribución aleatoria de agentes o partículas en el espacio. Este modelo también tiene una idea de atracción donde la idea principal se basa en que los agentes giran y son dirigidos hacia la mayor concentración de atractores por sensores como se muestra en la figura 11, que pueden escanear su entorno para saber cómo posicionarse y con ello empezar a desplazarse hacia los campos que tengan mayor fuerza.

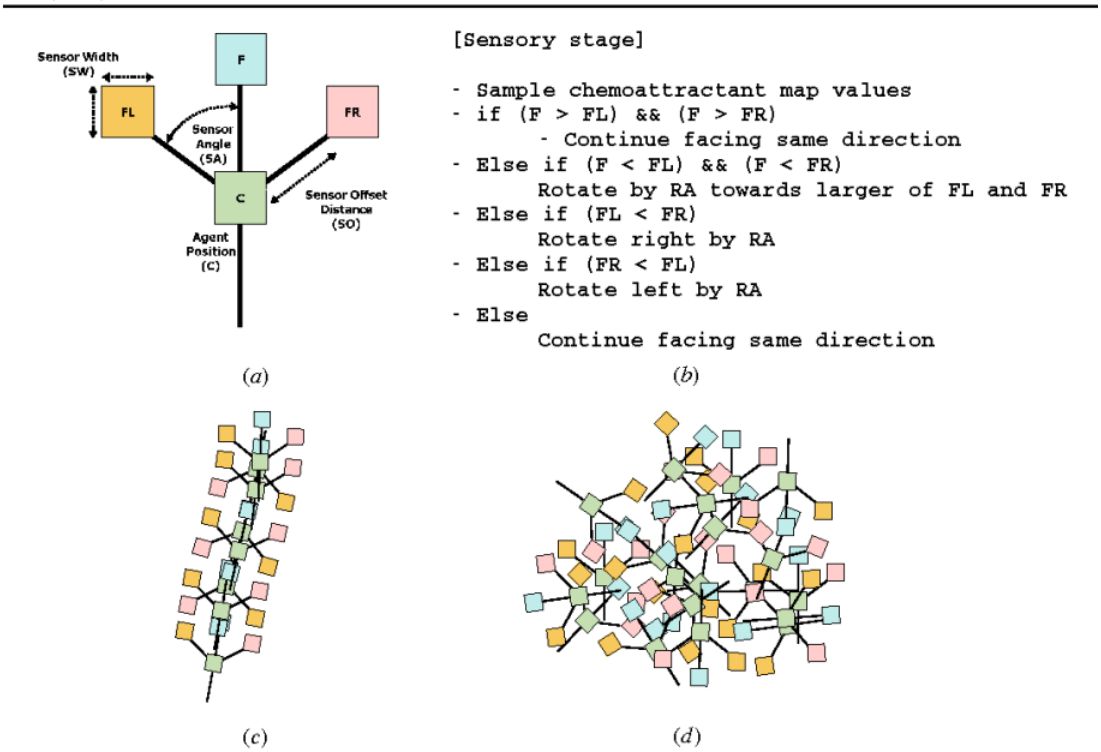


Figura 11: Representación del agente de acuerdo con el modelo de Jeff [18].

El modelo de Jeff Jones lo podemos ver aplicado en la solución de un laberinto (figura 12), 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.

Figura 12: Modelado de Jeff en un laberinto [17].

3.1.3. Modelado de Gunji

Gunji [12] trata el hongo Physarum como un cúmulo de vecinos donde se puede sobrepasar un cierto umbral para pasar a cierto estado, para su crecimiento inicialmente busca regiones de estímulo donde se propaga poco a poco. El componente principal que emplea es llamado "burbuja", podemos verlo como segregaciones que exploran el entorno 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 13.

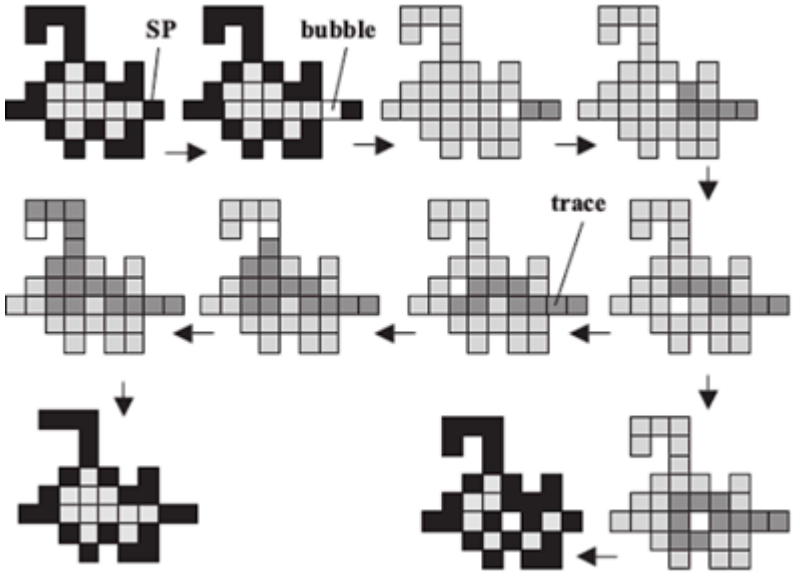


Figura 13: Ejemplo de aplicación del modelo de Gunji [12].

El modelo de Gunji lo podemos ver aplicado en la solución de un laberinto (figura 14).

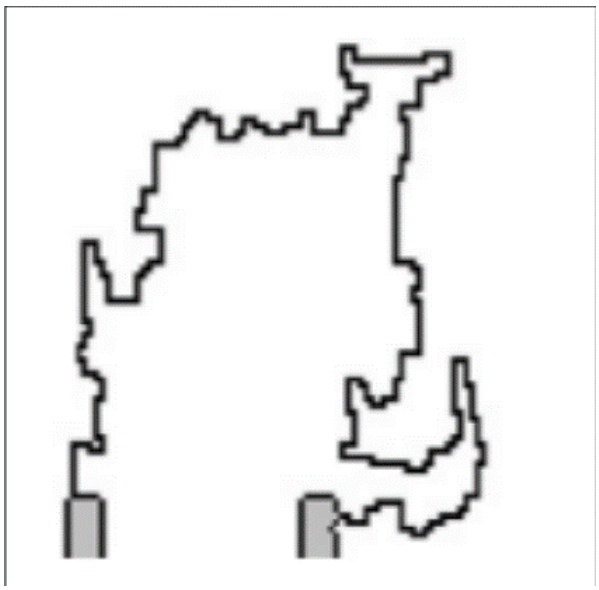


Figura 14: Modelado de Gunji en un laberinto [12].

3.1.4. Modelado de Yair

Como mencionamos, existe un trabajo en el que podemos encontrar una base bastante útil para lo que se pretende en este, se trata del trabajo de nuestro compañero Yaír, en su documento titulado "Modelado del Physarum Polycephalum implementado en robot basado en autómatas celulares", en donde, un apartado menciona las funciones de su simulador [10]:

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

Por otra parte, algunos requerimientos importantes del trabajo de nuestro compañero son que, el Sistema Operativo debe de ser alguna distribución de Linux, esto debido a que el IDE en el que fue desarrollado el sistema, el framework del Bluetooth que proporciona Qt-Creator no trabaja en sistemas Windows 7, el resto de requerimientos se pueden considerar mínimos.

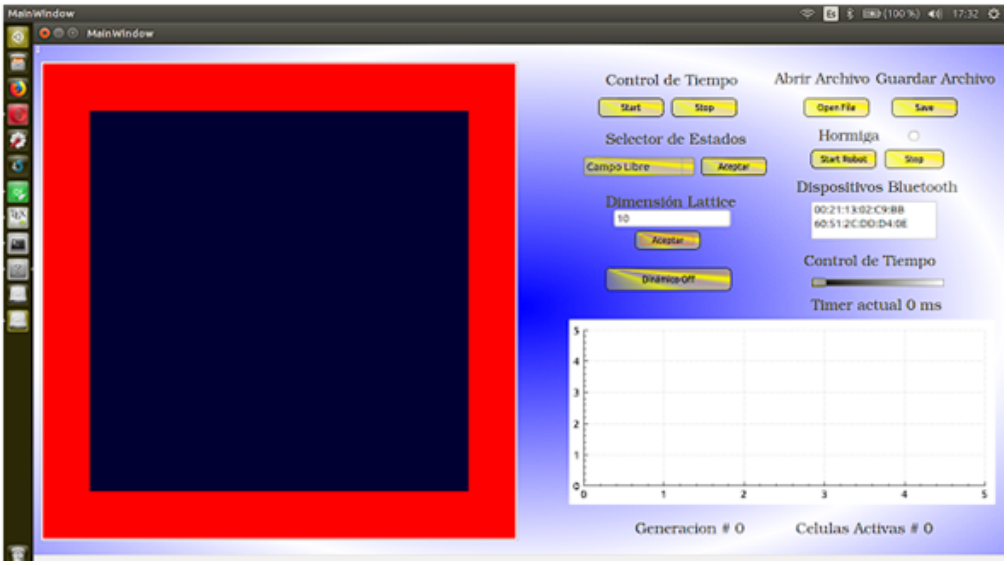


Figura 15: Simulador hecho en C++ [10].

Aunado a lo anterior en la figura 16 podemos ver la prueba del modelo de Yair en un laberinto, una limitante de este modelo es el no poder observar cambios en sus rutas

debido a las reglas que emplea.

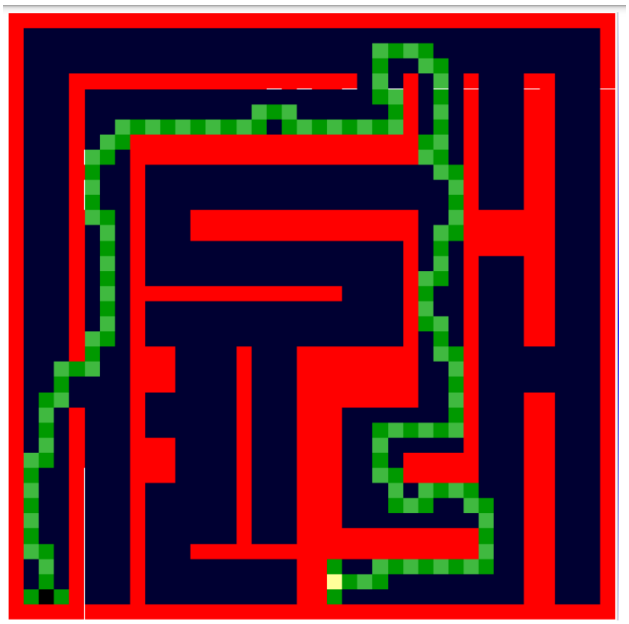


Figura 16: Modelado de Yair en un laberinto [10].

3.2. Autómatas celulares: Aplicación Golly

En este sección solo hablaremos de una aplicación llamada "Golly"la cual consiste en un software que permite visualizar la evolución de autómatas celulares en dos dimensiones, el proyecto inicialmente solo permitía la simulación del Juego de la Vida, afortunadamente ha recibido actualizaciones y mantenimiento por una comunidad ya que es de acceso libre. Además de eso, es multiplataforma, se puede ejecutar en Linux, Mac y Windows. Incluso, en 2020 lanzaron una versión para Android y iOS [30].

El software permite cambiar la regla que se quiere modelar, se pueden hacer cosas complejas mediante los archivos de texto que funcionan como configuración, contiene dentro de sí muchas configuraciones para probar el simulador en general. También le permite al usuario que cambie el aspecto de la interfaz, al igual que varios cosas más sobre la simulación [30], la interfaz de la aplicación Golly la podemos ver en la figura 17. Algunas de estas ideas las veremos en acción en el capítulo donde presentamos el simulador.

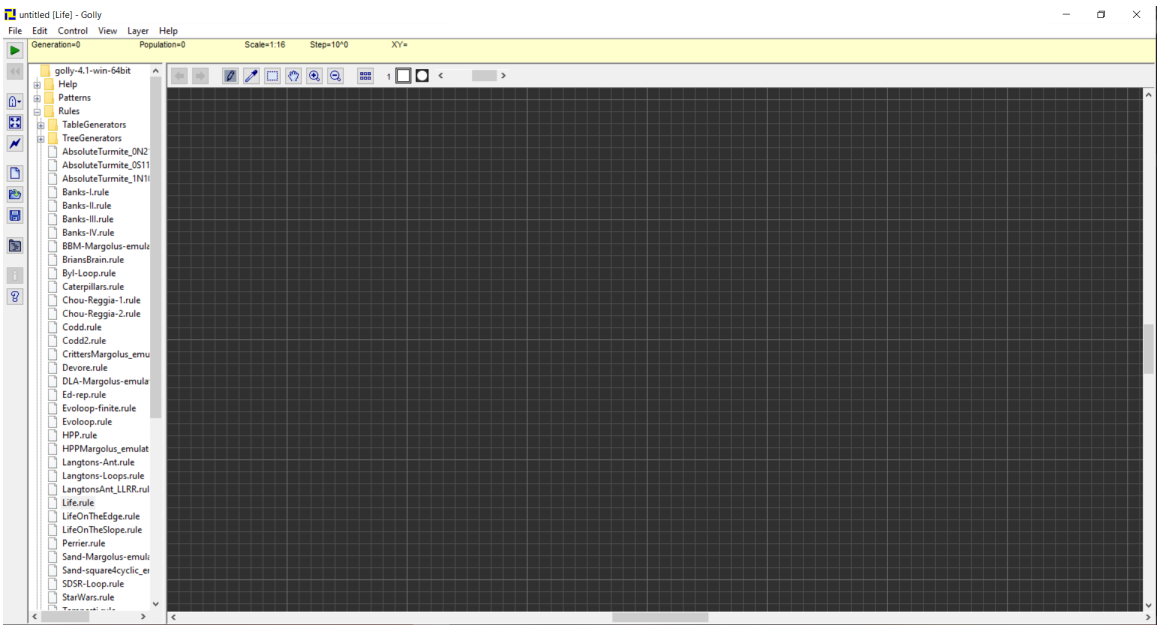


Figura 17: Interfaz de usuario de la aplicación Golly.

3.3. Robots mensajeros

En esta sección veremos algunos de los robots que grandes empresas han creado para la realización de entrega de paquetes, estos robots se suelen conocer robots mensajeros o *delivery robots* en ingles, nos enfocaremos mas en la tecnología que emplean para poder hacer la tarea de entregar paquetes y veremos el como se comunica con los clientes, aunque también hablaremos un poco de la empresa que los creo para tener un contexto de que tipo de paquetes hacen entrega. Nos enfocaremos en 4 robots los cuales son los mas famosos y conocidos.

3.3.1. Prototipo Yape creado por Just Eat

Just Eat Limited es un servicio británico de pedidos y entrega de alimentos en línea fundado en 2001 en Kolding, Dinamarca. La empresa empezó a trabajar en la creación de robots mensajeros desde el año 2016 dando su primera entrega en Greenwich, Inglaterra, esto gracias a la colaboración con una startup, sin embargo, en el año 2020 en el marco de Horeca Profesional Expo (HIP) presento a su prototipo **Yape**.

Este robot tiene la capacidad de soportar hasta 70 kilos de peso y transportarlo en un radio de 80 kilómetros, esto debido a su autonomía, usando una batería eléctrica. Su desplazamiento es totalmente autónomo y hace uso de en un sistema de sensores y cámaras para moverse en las calles y también poder evitar obstáculos [13].

La forma en la que se garantiza que el paquete ha llegado correctamente a su destino es mediante un código de acceso enviado por la aplicación de comida a domicilio, para lograr lo anterior el robot esta equipado con un lector que permite leer el código de acceso, además avisa al destinatario de que el robot se encuentra cerca del destino con 2 minutos de antelación [13]. Una imagen del robot la podemos ver en la figura 18.

Figura 18: Robot autónomo Yape presentado por Just Eat. [1]

[1] Restauración News. (2020, 25 febrero). Just Eat presenta un robot autónomo de reparto. Restauración News. https://restauracionnews.com/2020/02/just-eat-presenta-un-robot-autonomo-de-reparto/

3.3.2. Prototipo Starship creado por Starship Technologies

Starship Technologies es una empresa procedente de Estonia que desarrolla vehículos de reparto autónomos. Su robot insignia **Starship** tiene la capacidad de soportar hasta 10 kilos de peso y transportarlo en un radio de 6 kilómetros. Su desplazamiento es totalmente autónomo y hace uso de en un sistema de sensores, cámaras, GPSs y una unidad de medición inercial para moverse en las calles y también poder evitar obstáculos, mencionar que en el caso de que los sistemas que lo hacen autónomo fallaran puede ser controlado manualmente de manera remota pero esto provocaría que solo pudiera trabajar en un rango mucho menor [29].

La forma en la que se garantiza que el paquete ha llegado correctamente a su destino es igual al como o hace el robot Yape, en donde la comprobación es mediante un código de acceso enviado por la aplicación de comida a domicilio en el celular, para lograr lo anterior el robot esta equipado con un lector que permite leer el código de acceso [29]. Una imagen del robot la podemos ver en la figura 19.

Figura 19: Robot autónomo Starship creado por Starship Technologies.[2]

[2]Goodyear incorpora la tecnología de neumáticos sin aire a los robots Starship. (2022, 11 enero). Posventa. https://www.posventa.info/texto-diario/mostrar/3366463/goodyear-incorpora-tecnologia-neumaticos-aire-robots-starship

3.3.3. Prototipo Kiwibots creado por Kiwi Campus

Kiwi Campus es una startup colombiana que ofrece entregas de alimentos en California, Estados Unidos, utilizando robots en gran parte autónomos llamados Kiwibots. El **Kiwibot** puede recorrer 1.6 kilómetros al rededor de su origen, para lograr su desplazamiento hace uso de una cámara la cual al utilizar aprendizaje profundo (*deep learning*) puede interpretar información y recolectar información con otros sensores para poder realizar decisiones inteligentes asegurando una entrega rápida, segura y con bajo costo, aunado a lo anterior este robot puede identificar la luz de los semáforos para poder atravesar las calles y detectar objetos y obstáculos para prevenir colisiones [29].

De forma muy general el usuario pide algo a tienda de comida por medio de la aplicación de Kiwi, la orden se coloca dentro del robot y este se dirige a la ubicación desde donde se solicito, aun que en ocasiones pude interferir un trabajador de Kiwi para acercar al robot lo mas cerca del destino, finalmente el usuario que solicito el pedido es el único que puede abrir al robot mediante su aplicación [29]. Una imagen del robot la podemos ver en la figura 20.

Figura 20: Robot autónomo Kiwibot creado por Kiwi Campus. [3]

3.3.4. Prototipo Amazon Scout creado por Amazon

Amazon es una compañía estadounidense de comercio electrónico y servicios de computación en la nube con sede en la ciudad de Seattle, Washington. El **Amazon Scout** tiene sensores por todas partes para garantizar que pueda encontrar su camino en las aceras en las cuales puede encontrarse con obstáculos. Alrededor del robot hay una serie de cámaras y sensores ultrasónicos, los cuales luego son evaluados por un conjunto de algoritmos de aprendizaje automático (*machine learning*) que lo ayudan a trazar su camino.[29].

Mencionar que no se pudo encontrar como funciona exactamente la forma de recoger los paquetes que son mandados por medio de este robot, sin embargo, al ver como funciona la entrega en donde el humano interviene siempre avisa con minutos antes de llegar el paquete, por lo que suponemos funciona de manera parecida a los demás robots vistos. Una imagen del robot la podemos ver en la figura 21.

Figura 21: Robot autónomo Amazon Scout creado por Amazon. [4]

[3]EL ROBOT KIWIBOT TE LLEVA LA COMIDA A CASA. (2021, 19 marzo). REVISTA DE ROBOTS. https://revistaderobots.com/robots-y-robotica/el-robot-kiwibot-te-lleva-la-comida-a-casa/

[4]Grabham, D. (2020, 8 septiembre). ¿El robot de entrega Scout de Amazon llegará pronto al Reino Unido y Europa? Pocket-lint. https://www.pocket-lint.com/es-es/gadgets/noticias/amazon/153671-el-robot-de-entrega-scout-de-amazon-llegara-pronto-al-reino-unido-y-europa

4. Propuesta a Desarrollar

Retomando el objetivo de nuestra investigación: Modelar el comportamiento del organismo Physarum Polycephalum usando autómatas celulares para otorgar rutas alternas a un robot mensajero dentro del edificio de gobierno de la Escuela Superior de Cómputo del Instituto Politécnico Nacional; con la finalidad de enviar paquetes menores a 5 kilogramos entre las personas que laboran en ese edificio; es necesario partir este sistema en unos más pequeños para tener un mejor seguimiento y tener más probabilidades de detectar fallas en el diseño y correcciones futuras.

Comencemos por pensar en términos generales cómo va a funcionar el sistema, en su forma mínima permite enviar paquetes pequeños entre dos personas ubicadas en dos puntos A y B contenido en el espacio del edificio de gobierno de la ESCOM, separando esta idea tenemos que, el robot por sí solo, cuenta como un sistema lo suficientemente grande para considerarlo un proyecto, de la misma forma, el modelado del Physarum Polycephalum para proveer de rutas al robot es otro sistema, y, por último, debe existe alguna forma de comunicar a las personas en A y B con el robot, una interfaz como una aplicación web o móvil que debe entenderse por sí sola. Y no solo eso, de fondo debe haber una comunicación entre las partes.

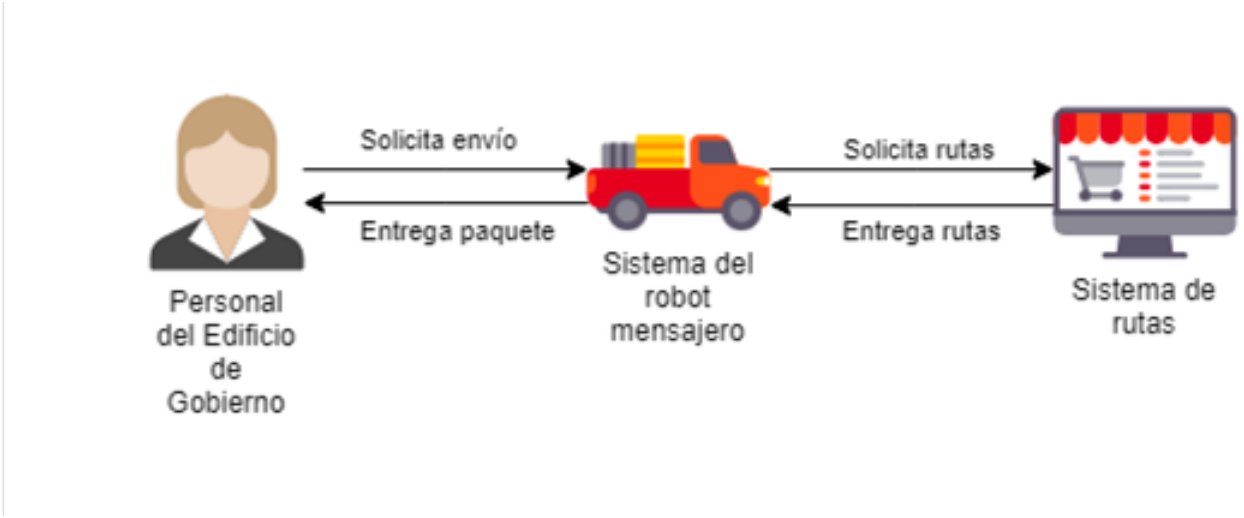


Figura 22: Arquitectura general del sistema. Elaboración propia.

Esta figura es solo con fines ilustrativos y está simplificada, como hemos comentado, se requiere tener sistemas independientes que se comuniquen por interfaces y funcionen como proveedores y consumidores de información, donde la clave está en la palabra independiente, que nos indica que es posible tener un desarrollo en paralelo siempre y cuando queden bien definidos los protocolos de comunicación entre cada parte del sistema final, lo que nos da paso a elegir una metodología orientada a prototipos independientes para el sistema completo, y, usando el modelo V para el desarrollo de cada proyecto que lo compone.

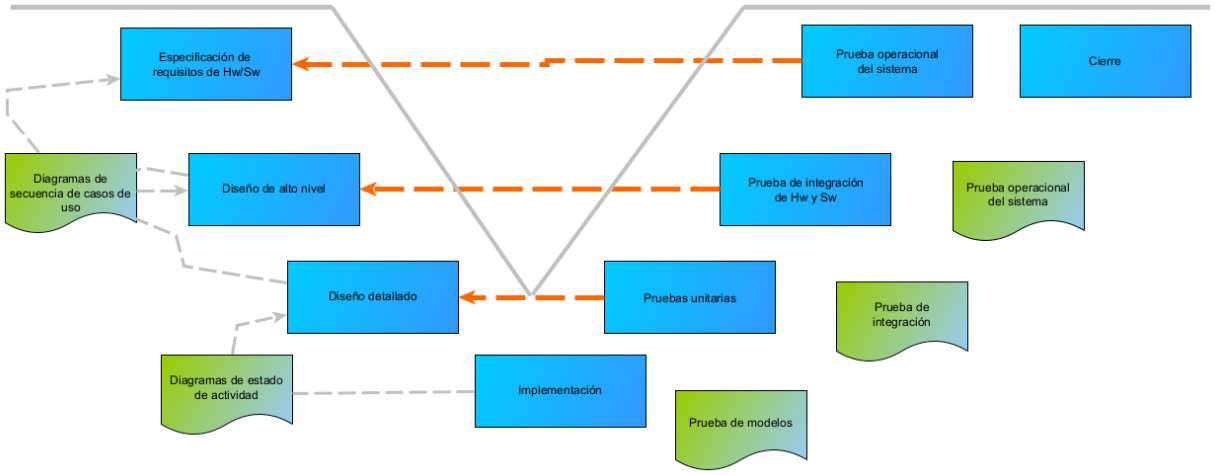


Figura 23: Metodología en V. Elaboración propia.

Observemos que, existe una validación y una verificación en este modelo, además de que permite el retroceso de fases para la corrección de requerimientos o, en su caso, adición o eliminación de estos. Este acercamiento resulta muy útil ya que se espera afrontar diversos problemas que durante el modelo híbrido no podríamos localizar, por lo que debemos estar abiertos a los cambios en cualquiera de las fases del diseño del software. Este modelo que originalmente está basado en el modelo de cascada pretende integrar aseguramiento de calidad con la parte derecha de la V [9].

Esto resulta útil, ya que la validación se asegura de que el sistema se comporte a las necesidades del ambiente, es decir, responde a la pregunta ¿estamos construyendo el sistema correcto?, mientras que la verificación se asegura de que la salida de la fase de desarrollo sea conforme a la especificación prevista como entrada de la V, en otras palabras, responde a la pregunta de ¿está construido bien el sistema? [9]. Sin embargo, el hecho de tener un sistema embebido nos hace tomar una búsqueda más amplia sobre las herramientas que podemos usar durante la fase de requerimientos, una buena herramienta se encuentra en el modelado de procesos de un sistema embebido que propone Ian Sommerville en su libro [27], ya que dedica especialmente un capítulo a ello.

Por otra parte, se requiere un modelado del organismo lo más pronto posible con el fin de poder atender a la mayor cantidad de cambios una vez que el ambiente de trabajo deje de ser remoto, es decir, requerimos de un proveedor de rutas lo más rápido posible, e irlo mejorando conforme a las necesidades y requerimientos que nos encontremos en el camino. Para esto resulta útil utilizar una metodología enfocada en prototipos para el software. Esto permite un enfoque evolutivo, un modelo que se itera con el tiempo y nos permite desarrollar prototipos cada vez más completos y complejos, hasta llegar al objetivo deseado [38]. Los siguientes capítulos, que son parte del desarrollo en el documento, están estructurados de la siguiente manera:

- Simulador. Mediante autómatas celulares, se puede obtener rutas. Principalmente se usa para ver el comportamiento.

- Robot. Se espera que viaje en un espacio abierto.
- Interfaces. Aquí se conecta el Robot con el Simulador, y se esboza la interfaz para los usuarios.
- Robot mensajero. En este capítulo se pulen detalles para las tres partes mediante pruebas.

Después de hablar sobre todo el desarrollo, cerraremos con los capítulos de Resultados y Conclusiones, junto con Trabajo para futuras investigaciones.

5. Análisis de Diseño

5.1. Simulador 2DPyCAM(10)

5.1.1. Requerimientos

5.1.1.1. Funcionales

- RF1: El sistema deberá proveer una interfaz gráfica de usuario donde el usuario pueda interactuar ante la formación de las diferentes configuraciones del AC.
- RF2: Deberá manejar entrada de datos archivo de texto o manualmente, donde este último se le darán las opciones o estados.
- RF3: Se podrá seleccionar el color para cada estado.
- RF4: Se podrá seleccionar si usar la vecindad de Moore o de Neumann
- RF5: Se deberá poder establecer el tamaño de las células.
- RF6: La interfaz deberá mostrar datos importantes tales como generación, número de células en determinado estado o algún otro dato importante.
- RF7: El sistema debe permitir al usuario generar la configuración inicial del AC, donde el robot solo va a esperar las acciones sin necesidad de que el usuario realice alguna configuración previa a este mismo, solamente las posiciones en donde vaya a ser usado.
- RF8: El sistema proveerá de una gráfica de densidad donde mostrará la expansión de Physarum conforme avancen sus generaciones.
- RF9: 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.

- RF10: Al momento de la entrada de archivos este tiene que validar que el archivo sea adecuado para procesar y que si los pixeles no concuerdan con el tamaño del archivo se pueda adaptar los repelentes.
- RF11: Se debe de proveer al usuario poder guardar la configuración actual en un archivo.
- RF12: Se debe poder definir reglas para la conducta del autómata.
- RF13: Una regla debe ser la de dejar una célula de espacio entre los repelentes para evitar colisiones con el robot.

5.1.1.2. No funcionales

- RNF1: 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.
- RNF2: El tiempo de aprendizaje del sistema por el usuario debe ser lo más rápido posible donde por ende no debe de complicarse su uso.
- RNF3: Debe de contar con algunos módulos o explicaciones para su fácil manejo, así como un manual de preferencia, mensajes informativos o de errores.
- RNF4: Debe existir integridad de la información para el manejo adecuado del AC y que no tengan algún problema sobre su evolución.

5.1.1.3. Restricciones

- Existen ciertos aspectos del Physarum que al ser propiedades físicas son difícilmente modelables en aspecto hardware por aspectos de costos, recursos o factibilidad, por lo que se descartaron y pueden ser contempladas para futuras investigaciones.

5.1.2. Diagramas

Para simplificar el funcionamiento del simulador, proporcionamos dos diagramas generales sobre su uso y su composición, en la Figura 24 podemos observar que hay varias funciones que se pueden realizar, algunas son secuenciales y otras se pueden ejecutar en el orden que se desee.

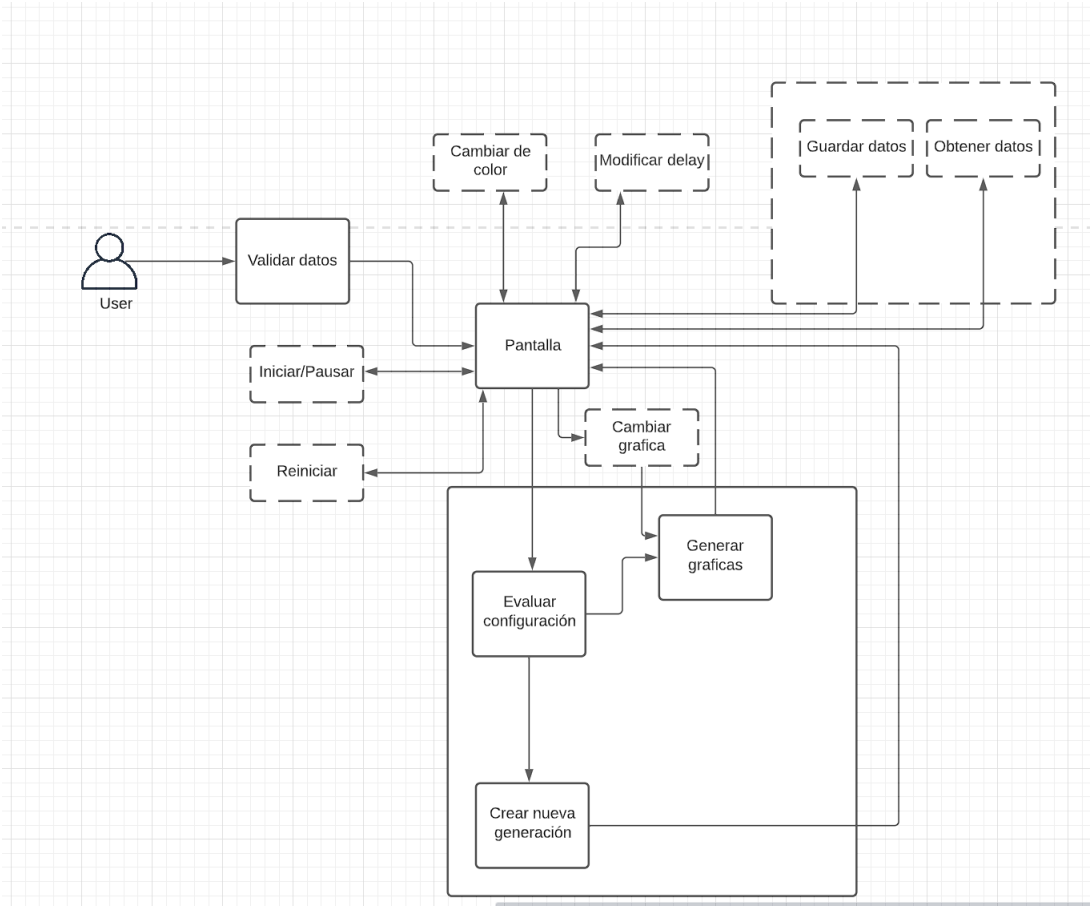


Figura 24: Diagrama general del simulador. Creación propia.

5.1.2.1. Casos de Uso

Para observarlo de una forma más ordenada, podemos ver el diagrama de la Figura 25, donde podemos notar a simple vista las funciones que se pretenden alcanzar con un prototipo. Este prototipo, en la práctica, consistió de 5 versiones, en la que definimos las siguientes características: Una de las primordiales características que el simulador pueda soportar hasta espacios de 100 x 100, reflejado en un mapeo que el usuario podrá

manipular bajo ciertas condiciones tales como pausar, reanudar la simulación, qué regla utilizar, guardar la configuración, abrir la configuración de un archivo y seleccionar los colores de los estados.

A continuación se presentan los caso de uso, después de ello se muestra el diagrama de clases del simulador. Con esto se pretende tener un punto técnico de partida, sin embargo, se ve mucho más claro el recorrido de cómo ha evolucionado el simulador con las cinco versiones que formaron parte de este prototipo.

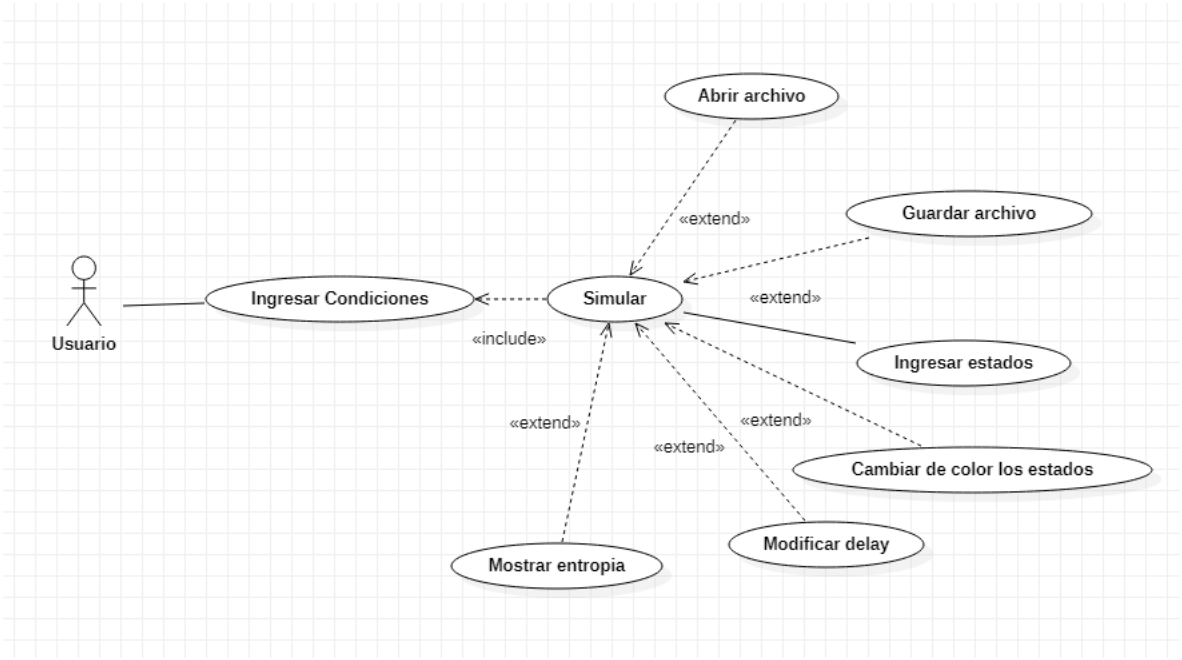


Figura 25: Caso de uso general del simulador. Creación propia.

Caso de uso	Ingresar condiciones
Requisitos Asociados	RF5, RF6, RF12, RF14
Descripción	Permite al usuario ingresar parámetros para iniciar la simulación como son el número de células por lado, la vecindad, la regla a utilizar y si se desea calcular la entropía.
Precondición	Que el ejecutable se encuentre en la misma ruta que las carpetas a usar como son entropía, img y save.
Pasos: Seleccionar la vecindad a ocupar. Ingresar el número de células por lado. Seleccionar la regla a ocupar. Seleccionar si desea calcular la entropía. Dar clic al botón Start.	

Tabla 3: Caso de uso Ingresar condición.

Caso de uso	Simular
Requisitos Asociados	RF1, RF2, RF7, RF9, RF10
Descripción	Permite al usuario ingresar estados en la simulación para evaluar su comportamiento, la simulación le proporciona información al usuario como la generación actual o el crecimiento de estados.
Precondición	Ingresar correctamente las condiciones.
Pasos: Seleccionar la vecindad a ocupar. Ingresar el número de células por lado. Seleccionar la regla a ocupar. Seleccionar si desea calcular la entropía. Dar clic al botón Start.	

Tabla 4: Caso de uso Simular

Caso de uso	Abrir archivo
Requisitos Asociados	RF3
Descripción	Permite al usuario buscar dentro del sistema alguna configuración previamente guardada.
Precondición	Exista un archivo guardado.
Pasos: Se elige la opción de abrir archivo. Se abre el cuadro de diálogo para seleccionar el archivo. Accede a la información que contiene el archivo. Se visualiza la información en el simulador.	

Tabla 5: Caso de Uso Abrir archivo.

Caso de uso	Guardar archivo
Requisitos Asociados	RF11
Descripción	Permite al usuario guardar dentro del sistema alguna configuración con el nombre ingresado por el usuario.
Precondición	Exista las carpetas para almacenar los archivos a guardar.
Pasos: El usuario elige la opción de guardar archivo. Se abre un cuadro de diálogo para poner el nombre del archivo. El archivo se almacena con el nombre con el número de células por lado, en una carpeta con su vecindad seleccionada.	

Tabla 6: Caso de uso Guardar archivo.

Caso de uso	Cambiar de color los estados
Requisitos Asociados	RF4
Descripción	Permite al usuario seleccionar el estado y cambiar el color de este.
Precondición	Ingresar correctamente las condiciones para el correcto funcionamiento del simulador.
Pasos: Elige el botón correspondiente para seleccionar el estado. Se abre el selector de color para que el usuario pueda seleccionarlo. Se visualiza la actualización del color seleccionado en la interfaz.	

Tabla 7: Caso de uso Cambiar de color los estados.

Caso de uso	Modificar el delay
Requisitos Asociados	RF16
Descripción	Permite al usuario aumentar o disminuir el delay de la simulación.
Precondición	Ingresar correctamente las condiciones para el correcto funcionamiento del simulador.
Pasos: El usuario presiona el botón para aumentar o disminuir el delay. Se visualiza el cambio en la ventana del simulador.	

Tabla 8: Caso de uso Modificar el delay

Caso de uso	Mostrar entropía
Requisitos Asociados	RF15
Descripción	En el caso que se deseé calcular la entropía, esta se deberá calcular sin afectar el rendimiento de la simulación.
Precondición	Cuando el usuario ingrese las condiciones, haber seleccionado positivamente calcular la entropía.
Pasos: El usuario ingresa el estado en la simulación. Se inicia la simulación. Cierra la ventana de simulación. Se realizan los cálculos para la entropía. Se muestra la gráfica de la entropía.	

Tabla 9: Caso de uso Mostrar entropía

5.1.2.2. Clases

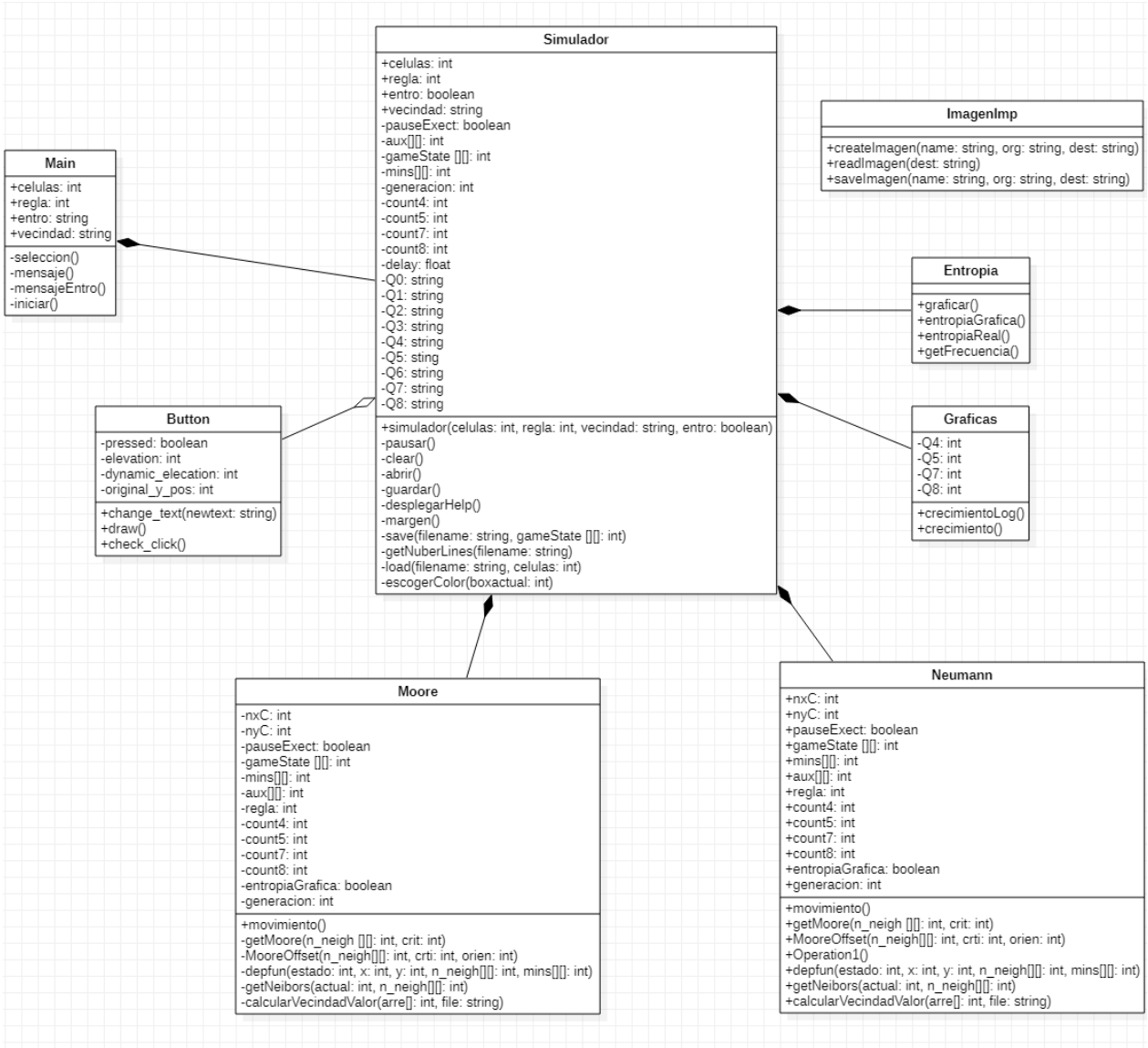


Figura 26: Diagrama de Clases del Simulador.

5.2. Interfaz de usuario usuario - robot (Servicio Web)

5.2.1. Análisis del sistema

En esta sección se define nuestro servicio web en el que se evalúa las herramientas, tecnologías y requerimientos de desarrollo, especificaciones técnicas básicas necesarias para un usuario final, que deberá de cumplir para el correcto funcionamiento del sistema y requerimientos, mencionar que todo este sistema se trabaja en un mismo servidor el cual la misma tarjeta de desarrollo Raspberry Pi 4 que es el controlador del robot.

5.2.1.1. Alcance del sistema

5.2.1.1.1. Físicos y operativos

A modo de programación del servicio web, se desarrolla para que sea accesible desde cualquier computadora hasta en cualquier dispositivo móvil, debido a que nuestro desarrollo web tiene la característica de ser responsivo, es decir, son capaces de adaptarse a cualquier tamaño de pantalla. De igual manera el servicio web se trabajará bajo el Framework de Python llamado Flask el cual nos permite realizar un servicio web completo ya que es una herramienta de desarrollo para crear aplicaciones web rápidamente y con un mínimo número de líneas de código, ademas de que se pueden usar el lenguaje de marcado HTML siendo complementado por JavaScript y Bulma que es un Framework de CSS, de esta manera se logra tener todo un servicio web completo y con una estética muy amigable con los usuarios finales.

5.2.1.1.2. Usuario básico

De forma muy general témenos solo dos tipos de usuarios, el usuario básico y el administrador. El usuario básico puede registrarse en el servicio web, sin embargo, el usuario básico tendrá que ser validado por el administrador de forma general este usuario será capaz de hacer uso del robot para envió de paquetes ademas de tener unos cuantos módulos

adicionales como modificar cuenta, bitácora de lo que suceda en el sistema en donde el usuario se vea afectado entre otros módulos que veremos un poco mas adelante.

5.2.1.1.3. Administrador(es)

Los usuarios administradores tienen los mismos módulos que los usuarios básicos, sin embargo, tiene la facultad de aceptar o eliminar la solicitud de registro de los usuarios básicos, ademas de que estos son capaces de ver una cámara instalada en el robot y que la bitácora que mencionamos en el punto anterior en lugar de ver lo que sucede en el sistema lo que afectan al mismo usuario, los administradores pueden ver todo lo que sucede en el sistema.

5.2.1.2. Usuarios del sistema

En este apartado se definen todos los usuarios bajo el contexto en el que se desarrolla la aplicación y los requerimientos para cada uno.

5.2.1.2.1. Usuario básico

Un usuario básico es aquella persona que su lugar principal de trabajo es en el primer piso del edificio de gobierno de la Escuela Superior de Cómputo y que ademas de que tenga la necesidad de hacer envíos entre oficinas en la ubicación anteriormente mencionada.

Perfil:

- Laborar en una oficina que se ubique en el primer piso del edificio de gobierno de la ESCOM.
- Agilidad en uso básico de plataformas web.
- Interés en hacer envíos entre oficinas para optimización de su tiempo.

5.2.1.2.2. Administrador(es)

Un administrador será todo aquel personal que labora en el edificio de gobierno de la ESCOM el cual sea jefe del departamento (oficina), por ejemplo, el director de la ESCOM, el subdirector, etc.

Puede entenderse que el rol de administrador será el que puede validar o no el registro de los usuarios básicos con base en su plantilla de trabajadores, ademas de poder ver todos los movimientos que suceden en el sistema, ademas de que también pueden realizar envíos.

Perfil:

- Ser jefe de un departamento que la oficina se ubique en el primer piso del edificio de gobierno de la ESCOM.
- Interés en hacer envíos entre oficinas para optimización de su tiempo.
- Agilidad en uso básico de plataformas web.

5.2.2. Modelo de dominio

El modelo de dominio se crea con el fin de representar el vocabulario y los conceptos clave del dominio del problema. El modelo de dominio también identifica las relaciones entre todas las entidades comprendidas en el ámbito del dominio del problema, y comúnmente identifica sus atributos.

Nombre	**Tipo**	**Descripción**
Correo electrónico	Correo, identificador	Correo electrónico del usuario, identificador dentro del sistema.
Nombre	Palabra corta	Nombre o nombres del usuario.
Apellido paterno	Palabra corta	Primer apellido del usuario.
Apellido materno	Palabra corta	Segundo apellido del usuario.
Teléfono celular	Teléfono	Numero telefónico de contacto.
Contraseña	Contraseña	Contraseña para acceder a este servicio web.
Identificador tipo usuario	Número	Numero para tipo de usuario (0 para usuario básico, 1 para administrador).
Validado	Booleano	Booleano de validación

Tabla 10: Entidad usuario.

Nombre	Tipo	Descripción
Identificador tipo usuario	Número	Numero para tipo de usuario (0 para usuario básico, 1 para administrador).
Descripción	Palabra corta	Nombre para los tipos de usuario

Tabla 11: Entidad tipos usuarios.

Nombre	Tipo	Descripción
Identificador	Número, identificador	Numero para identificar el reporte.
Actividad	Palabra larga	Actividad que se llevo acabo en el servicio web.
Identificador	Número	Numero para identificar el reporte.
Correo responsable	Correo	Correo del responsable de la actividad.
Correo afectado	Correo	Correo del afectado de la actividad.
Fecha	Fecha	Fecha en que se realiza la actividad.

Tabla 12: Entidad reportes del sistema.

5.2.3. Reglas de negocio

Las reglas de negocio sirven para definir o restringir acciones que serán implementadas en funcionalidad. En la presente sección, se muestran las reglas de negocio del sistema

- RN1: El usuario básico y el administrador, deberán estas registrados y validados según sea el caso e iniciar sesión en el servicio web para poder usarlo.
- RN2: Al registrar un usuario, la contraseña registrada debe de tener mínimo 6 caracteres alfanuméricos, no se aceptan acentos, comas ni diéresis.
- RN3: Las contraseñas de usuario para el acceso se deberán de almacenar de manera cifrada.
- RN4: Los administradores son los únicos capaces de aceptar o rechazar una solicitud de registros.
- RN5: Tanto un usuario básico como un administrador pueden hacer el uso de robot para envió de paquetes.

- RN6: Tanto un usuario básico como un administrador pueden ver las actividades que se realizan en el servicio web, por un lado el usuario básico solo puede ver las que lo afectan directamente y por el lado del administrador puede ver todo.
- RN7: Ambos tipos de usuario tiene la facultada de poder actualizar su contraseña así como su numero telefónico.
- RN8: Ambos tipos de usuario también tiene la facultad de eliminar su cuenta si así lo desean.
- RN9: Tanto el usuario básico como el administrador tienen la posibilidad de recuperar su contraseña.
- RN10: La recuperación de contraseña se hace de manera cifrada y con tokens con un tiempo de vida de hasta 5 minutos.
- RN11 Los usuarios administradores tiene la facultad de poder ver una cámara instalada en el robot la cual les permitirá ver el avance de este.
- RN12: El robot solo puede ser usado por una persona al mismo tiempo.
- RN13: Tanto un usuario básico como un administrador pueden hacer el uso de robot para modificar su ubicación actual para iniciar un envió de paquetes.

5.2.4. Requerimientos

5.2.4.1. Funcionales

Con los requisitos funcionales definiremos las capacidades que deberá tener el sistema, especificando los comportamientos específicos de las actividades y servicios que el sistema proveerá, en pocas palabras, las funcionalidades que el sistema debe realizar.

- RF1: El sistema deberá proveer una interfaz gráfica de usuario donde el usuario pueda interactuar con el robot.

- RF2: El sistema permite al usuario ingresar al servicio web, debe capturar su correo electrónico y contraseña, una vez autenticado se desplegarán las funcionalidades dependiendo del rol; usuario básico o administrador.
- RF3: El sistema permite registrar usuarios de tipo usuario básico.
- RF4: El sistema permite visualizar las actividades que afectan al usuario básico o todas las actividades para el administrador.
- RF5: El sistema permite ver la cámara del robot solo para administradores.
- RF6: El sistema permite la actualización de información de los dos tipos de usuarios.
- RF7: El sistema permite la eliminación de la cuenta para los dos tipos de usuario .
- RF8: El sistema proveerá del estatus del robot en todo momento y se mostrara solamente cuando sea solicitado por cualquiera de los dos tipos de usuarios.
- RF9: El sistema es capaz de mandar correo para poder hacer la recuperación de contraseña de cualquiera de los dos tipos de usuarios de manera cifrada.
- RF10: El sistema permite a los dos tipos de usuarios poder hacer envíos con el robot así como cambiar su ubicación para poder iniciar un envió.
- RF11: El sistema permite ver las cuentas nuevas, la validación o eliminación de estas por parte de los administradores.
- RF12: El sistema es capaz de mandar correo para avisar a cualquiera de los dos tipos de usuario que se esta enviando un paquete para el.
- RF13: El sistema es capaz de mandar correo para avisar a cualquiera de los dos tipos de usuario que el robot llego a la ubicación final y que puede recogerlo.
- RF14: El sistema es capaz de abrir una caja para que cualquiera de los dos tipos de usuario pueda recoger el paquete entregado.

- RF15: El sistema es capaz de cerrar la caja en donde se colocan los paquetes.
- RF16: El sistema es capaz de mandar correo para avisar a cualquiera de los dos tipos de usuario que el paquete fue recogido de manera exitosa.

5.2.4.2. No funcionales

- RNF1: El sistema debe de soportar un cierto número máximo conexiones simultaneas.
- RNF2: El tiempo de aprendizaje del sistema por el usuario debe ser lo más rápido posible donde por ende no debe de complicarse su uso.
- RNF3: Debe de ser utilizable desde cualquier tipo de dispositivo.
- RNF4: Debe de ser capaz de poder guardar la información del robot al momento.
- RNF5: En caso de haber un error del sistema tanto interno como externo debe de ser capaz de recuperar la ultima información almacenada.
- RNF6: En caso de un error en el servicio web el sistema sera capaz de mostrare una pagina en donde diga que paso un error.

5.2.5. Diagramas

5.2.5.1. Arquitectura del sistema

El sistema completo se basa en la arquitectura cliente servidor, podemos identificar 1 cliente, el servicio web para los usuarios básicos y los administradores. El servicio web se realiza la lógica de negocio principal, para el caso del servidor sera para almacenamiento de datos exclusivamente, para proveer imagen en directo del robot y hacer uso de este.

Flask funcionará como lenguaje de frontend y backend para el desarrollo del único cliente, todos los servicios que consumirá nuestro cliente son servicios que contiene el mismo sistema.

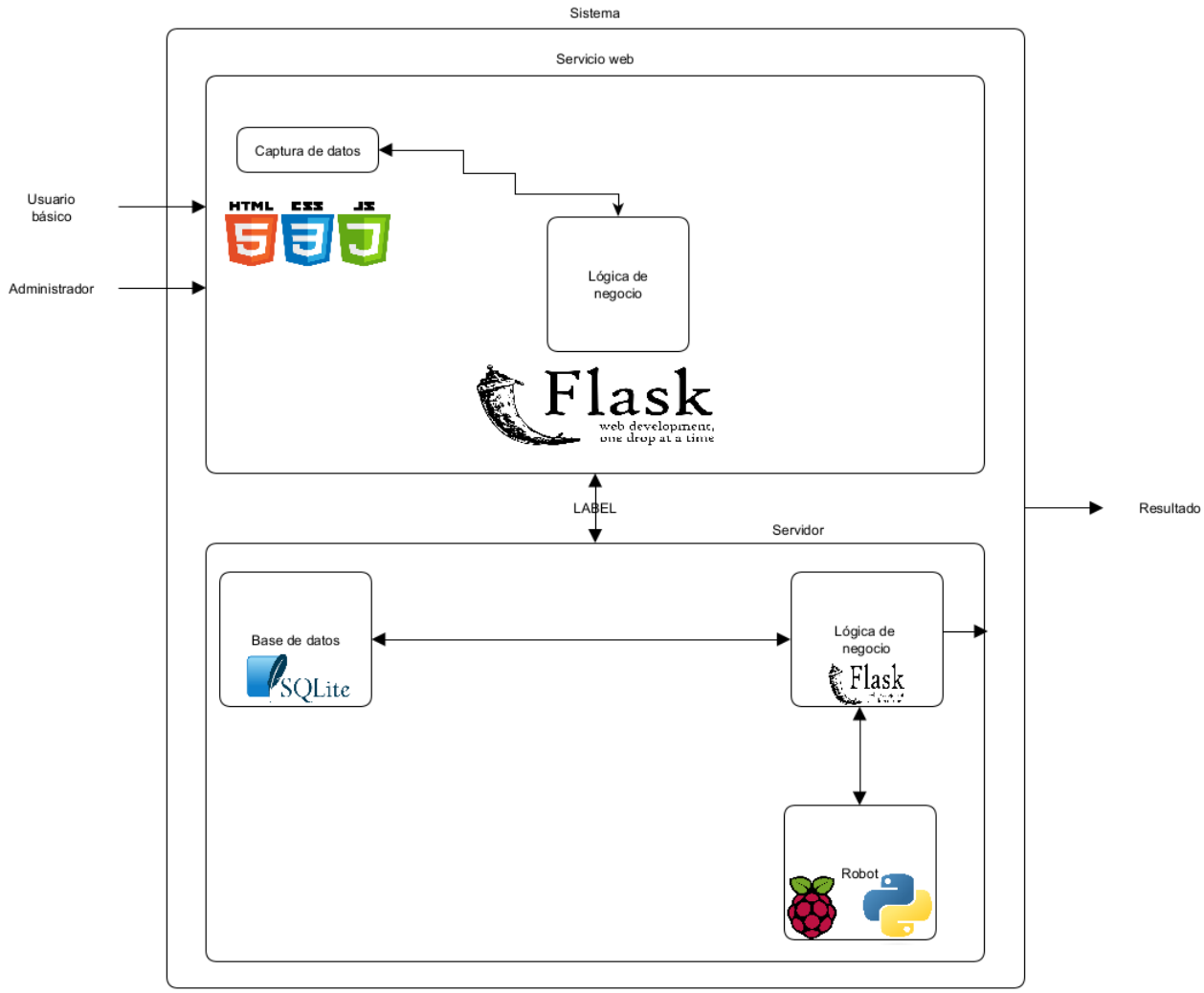


Figura 27: Diagrama a bloques del sistema.

5.2.5.2. Modelado de la informacion

Base de datos SQL, la cual se trabaja desde el mismo servidor.

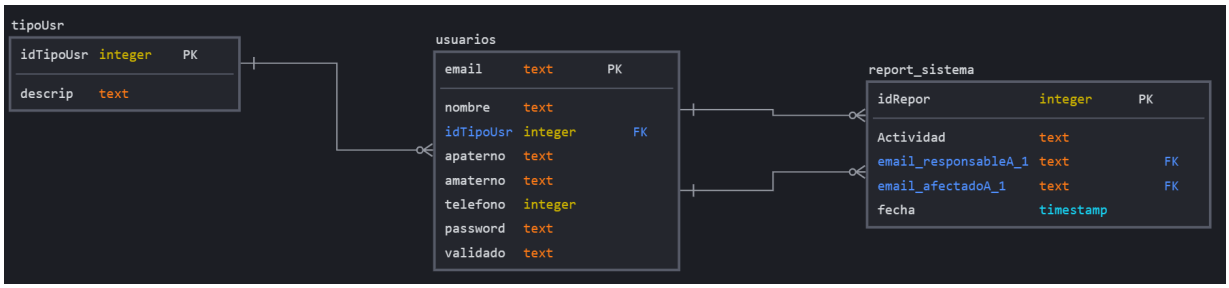


Figura 28: Diagrama de base de datos SQL.

5.2.5.3. Diagrama de proceso general

En el siguiente diagrama BPMN, se representa el conjunto de actividades que permite describir las funcionalidad principal del sistema, considerando que un usuario básico ha sido validado por los administradores, a partir de ahí se realiza el proceso de uso del servicio web junto con el robot.

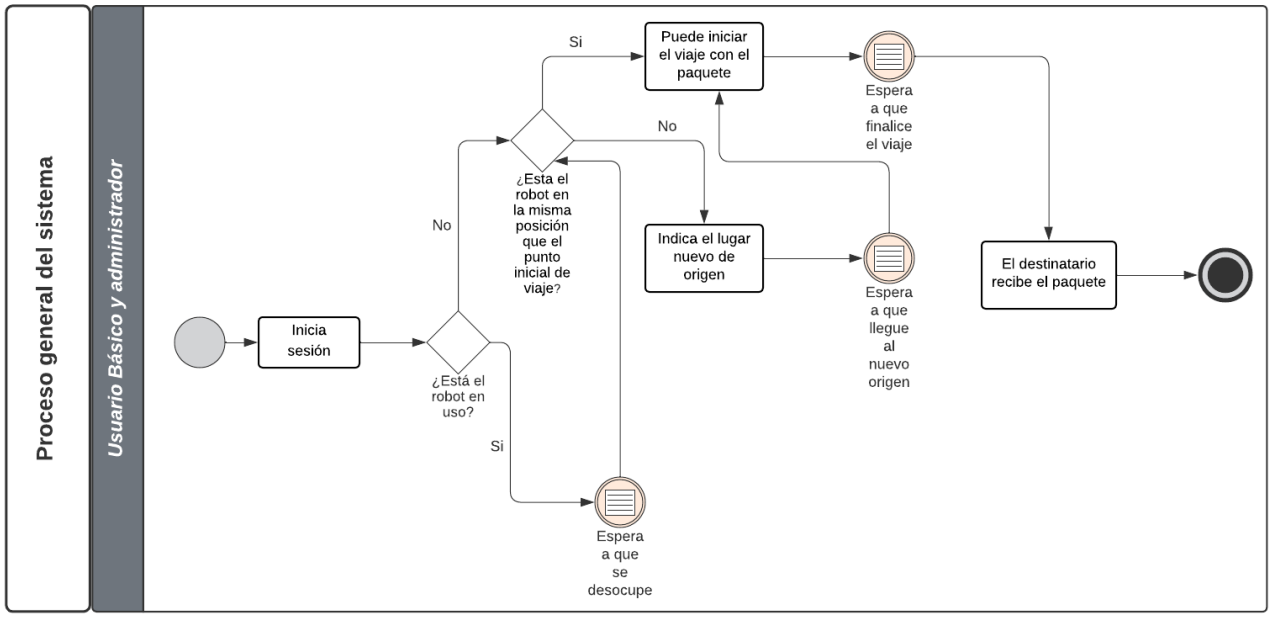


Figura 29: Diagrama BPMN de proceso general.

5.2.5.4. Caso de Uso

Se muestra el diagrama de casos de uso general, es decir, representa los casos de uso o funcionalidades de todo el sistema.

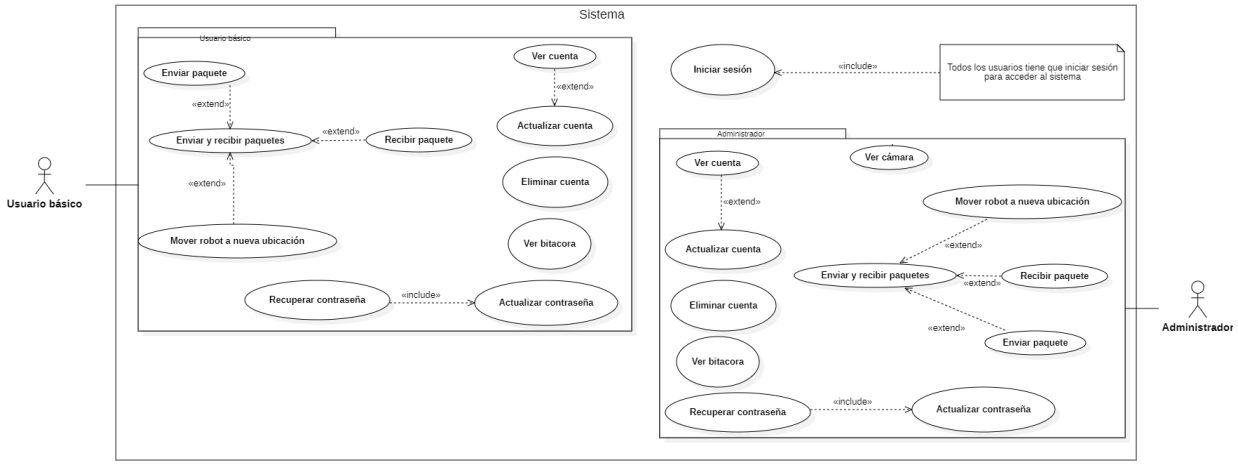


Figura 30: Diagrama general de casos de uso

Caso de uso	Inicio sesión
Requisitos Asociados	RF2
Descripción	Permite al usuario ingresar su correo y contraseña para poder usar el servicio web dependiendo de su rol.
Precondición	Haber entrado a la sección de iniciar sesión e ingresar correctamente sus datos
Pasos: Ingresar a la sección de iniciar sesión. Ingresar su correo. Ingresar su contraseña. Dar clic al botón Ingresar. Confirmar acción.	

Tabla 13: Caso de uso Inicio sesión.

Caso de uso	Registro
Requisitos Asociados	RF3
Descripción	Permite al usuario ingresar sus datos para el registro en el servicio web para poder usarlo, los datos a ingresar son nombre(s), apellido paterno, apellido materno, número de celular, contraseña, validación de contraseña y correo-e.
Precondición	Haber entrado a la sección de registro e ingresar correctamente sus datos
Pasos: Ingresar a la sección de registro. Ingresar su(s) nombre(s). Ingresar su apellido paterno. Ingresar su apellido materno. Ingresar su número de celular. Ingresar su contraseña. Validar su contraseña. Ingresar su correo-e. Dar clic al botón Crear cuenta	

Tabla 14: Caso de uso Registro

Caso de uso	Enviar o recibir paquetes(Cambiar ubicación actual del robot)
Requisitos Asociados	RF1,RF8,RF10
Descripción	Permite al usuario enviar y recibir paquete, ademas de cambiar la ubicación actual del robot.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: Se elige la opción de enviar o recibir paquetes. Elegir la ubicación nueva para el robot. Dar clic al botón Mover robot a la nueva ubicación inicial.	

Tabla 15: Caso de Uso Enviar o recibir paquetes(Cambiar ubicación actual del robot).

Caso de uso	Enviar o recibir paquetes(Enviar paquete)
Requisitos Asociados	RF1,RF8,RF10,RF13
Descripción	Permite al usuario enviar y recibir paquete, ademas de cambiar la ubicación actual del robot.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: Se elige la opción de enviar o recibir paquetes. Elegir la ubicación para la entrega. Elegir el correo de la persona que recibirá el paquete. Colocar paquete. Dar clic al botón Enviar paquete.	

Tabla 16: Caso de Uso Enviar o recibir paquetes(Enviar paquete).

Caso de uso	Enviar o recibir paquetes(Recibir paquete)
Requisitos Asociados	RF14,RF15,RF16
Descripción	Permite al usuario enviar y recibir paquete, ademas de cambiar la ubicación actual del robot.
Precondición	Haber iniciado sesión.
Pasos: El usuario ingresa al enlace mandado a su correo. Dar clic al botón Recoger paquete. Dar clic al botón Confirmar paquete entregado.	

Tabla 17: Caso de Uso Enviar o recibir paquetes(Recibir paquete).

Caso de uso	Bitácora del sistema
Requisitos Asociados	RF4
Descripción	Permite al usuario básico ver todo aquel evento en el sistema que lo involucra, por otro lado, al administrador le permite ver todo evento en el sistema.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: El usuario elige la opción de Bitácora del sistema.	

Tabla 18: Caso de uso Bitácora del sistema.

Caso de uso	Vídeo en directo del robot
Requisitos Asociados	RF5
Descripción	Permite solamente al administrador ver la cámara incorporada al robot.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: El usuario administrador elige la opción de Vídeo en directo del robot.	

Tabla 19: Caso de uso Vídeo en directo del robot.

Caso de uso	Modificar cuenta(ver información)
Requisitos Asociados	RF6
Descripción	Permite al usuario ver información de su cuenta.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: El usuario elige la opción de Modificar cuenta.	

Tabla 20: Caso de uso Modificar cuenta(ver información).

Caso de uso	Modificar cuenta(modificar cuenta)
Requisitos Asociados	RF6
Descripción	Permite al usuario ver modificar cierta información de su cuenta.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: El usuario elige la opción de Modificar cuenta. Ingresa nuevo número celular, si así lo requiere. Ingresa nueva contraseña y confirma nueva contraseña, si así lo requiere. Dar clic al botón Enviar. Confirmar acción.	

Tabla 21: Caso de uso Modificar cuenta(modificar cuenta).

Caso de uso	Eliminar cuenta
Requisitos Asociados	RF7
Descripción	Permite al usuario eliminar su cuenta.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: El usuario elige la opción de Eliminar cuenta. Lee el texto en negritas. Dar clic al botón Eliminar cuenta. Confirmar acción.	

Tabla 22: Caso de uso Eliminar cuenta.

Caso de uso	Recuperar contraseña(solicitud de enlace)
Requisitos Asociados	RF10
Descripción	Permite al usuario recuperar su contraseña.
Precondición	Estar en la opción Iniciar sesión y elegir la opción correspondiente .
Pasos: El usuario elige la opción de Recuperar contraseña. Ingresar su correo-e. Dar clic al botón Recuperar contraseña.	

Tabla 23: Caso de uso Recuperar contraseña(solicitud de enlace).

Caso de uso	Recuperar contraseña(formulario cambio de contraseña)
Requisitos Asociados	RF10
Descripción	Permite al usuario cambiar su contraseña con un tiempo de vida del enlace de 5 minutos.
Precondición	Ingresar al enlace mandado por correo.
Pasos: El usuario ingresa al enlace mandado a su correo. Ingresar su nueva contraseña. Validar su nueva contraseña. Dar clic al botón Cambiar contraseña.	

Tabla 24: Caso de uso Recuperar contraseña(formulario cambio de contraseña).

Caso de uso	Ver cuentas no validadas (ver cuentas)
Requisitos Asociados	RF11
Descripción	Permite al usuario administrador ver las cuentas no validadas .
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: El usuario administrador ingresa a la opción Ver cuentas no validadas.	

Tabla 25: Caso de uso Ver cuentas no validadas (ver cuentas).

Caso de uso	Ver cuentas no validadas (validar cuenta)
Requisitos Asociados	RF11
Descripción	Permite al usuario administrador ver las cuentas no validadas y validar las cuentas según sea el caso.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: El usuario administrador ingresa a la opción Ver cuentas no validadas. En caso de querer validar una cuenta dar clic en el botón de color verde. Confirmar acción.	

Tabla 26: Caso de uso Ver cuentas no validadas (validar cuenta).

Caso de uso	Ver cuentas no validadas (eliminar cuenta)
Requisitos Asociados	RF11
Descripción	Permite al usuario administrador ver las cuentas no validadas y eliminar las cuentas según sea el caso.
Precondición	Haber iniciado sesión y elegir la opción correspondiente .
Pasos: El usuario administrador ingresa a la opción Ver cuentas no validadas. En caso de querer eliminar una cuenta dar clic en el botón de color rojo.	

Tabla 27: Caso de uso Ver cuentas no validadas (eliminar cuenta).

6. Simulador 2DPyCAM(10)

6.1. Physarum Polycephalum

Siguiendo la estructura planteada en el capítulo 3, comenzaremos por el simulador, teniendo en cuenta el avance mencionado en la introducción por nuestro compañero Yair. Partamos del hecho que debe contar una interfaz gráfica y permitir la modificación de ciertos aspectos, su objetivo es que se logre visualizar el resultado de una simulación con una configuración inicial, la definición de ciertos estados, lo que simbolizan los colores entre muchos otros ámbitos. Concentrándolo tenemos lo siguiente:

6.1.1. Especificaciones sobre el modelo

Continuando con la línea de delimitar aún más los parámetros, se ha decidido usar la vecindad de Moore, la decisión de tomar esta vecindad es debido a que el robot va a emplear las rutas trazadas por el AC, por lo cual esta vecindad es adecuada para obtener rutas en diagonal. A partir de esto, definimos el autómata celular de radio 1 de la siguiente manera:

- $d = 2$
- $S = \{0; 1; 2; 3; 4; 5; 6; 7; 8\}$
- $N = \{(-1, 0), (0, 1), (0, 0), (1, 0), (0, -1), (1, 1), (-1, -1), (-1, 1), (1, -1)\}$
- $f : S^9 \to S$

A partir de ahora, usaremos la notación $c\left(\overrightarrow{n}\right)$, refiriéndonos a la configuración de c tal que $c\left(\overrightarrow{n}\right)$ es una transición adecuada de c para algún 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.

Color	Estado	Evolución	Significado	¿Es Physarum?
	q0	q7	Campo libre	No
	q1	q6	Nutriente no encontrado	No
	q2	q2	Repelente	No
	q3	q3	Punto inicial	Sí
	q4	q5	Gel en contracción	Sí
	q5	q0, q8	Gel en compuesto	Sí
	q6	q6	Nutriente hallado	No
	q7	q4	Expansión del Physarum	Sí
	q8	q5	Gel sin compuesto	Sí

Tabla 28: Simbología del Simulador. Creación propia.

6.1.2. 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.

$$c\left(\overrightarrow{n}\right)_{t+1} = 7 \text{ sí } c\left(\overrightarrow{n}\right)_t = 0 \text{ y } q_j > 0,\ j = 3,4,6 \tag{6.1}$$

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

$$c(\overrightarrow{n})_{t+1} = 6 \text{ sí } c(\overrightarrow{n})_t = 1 \text{ y } q_j > 0,\ j = 5,6 \tag{6.2}$$

Repelente (estado 2): Estado periódico.

$$c(\overrightarrow{n})_{t+1} = 2 \text{ sí } c(\overrightarrow{n})_t = 2 \tag{6.3}$$

Punto inicial (estado 3): Estado periódico.

$$c(\overrightarrow{n})_{t+1} = 3 \text{ sí } c(\overrightarrow{n})_t = 3 \tag{6.4}$$

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.

$$c(\overrightarrow{n})_{t+1} = 5 \text{ sí } c(\overrightarrow{n})_t = 4 \text{ y } q_j > 0,\ j = 3,4,6;\ connect = true \tag{6.5}$$

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.

$$c(\overrightarrow{n})_{t+1} = 0 \text{ sí } c(\overrightarrow{n})_t = 5 q_j < 2,\ j = 5,\ dep = 0; \text{ si no } 8 \tag{6.6}$$

Nutriente hallado (estado 6): Sin transición.

$$c(\overrightarrow{n})_{t+1} = 6 \text{ sí } c(\overrightarrow{n})_t = 6 \tag{6.7}$$

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.

$$c(\overrightarrow{n})_{t+1} = 4 \text{ sí } c(\overrightarrow{n})_t = 7 \text{ y } q_j > 0,\ j = 3,4,6 \tag{6.8}$$

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

$$c(\overrightarrow{n})_{t+1} = 5 \text{ sí } c(\overrightarrow{n})_t = 8q_j > 0, j = 0, 4, 1, 6 \tag{6.9}$$

6.1.3. Primera versión

Partimos de un primer avance del simulador con dimensiones de 20 células x 20 células, en este prototipo solamente se buscó simular correctamente el comportamiento del Physarum con la vecindad de Moore, en donde al correr el simulador automáticamente se pone un margen con repelentes como se aprecia en la Figura 31.

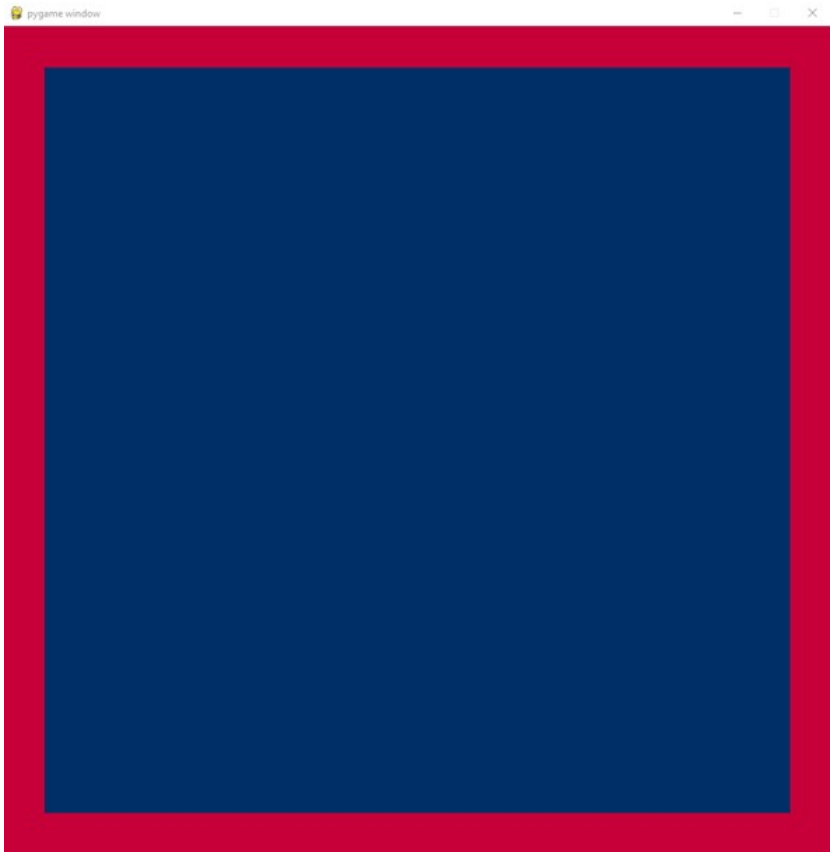


Figura 31: T=0.

En este prototipo se puede ingresar un punto inicial con el click izquierdo, un repelente con el click derecho, un nutriente con el click del botón de la rueda del mouse y puede pausar o reanudar la simulación presionando cualquier tecla.

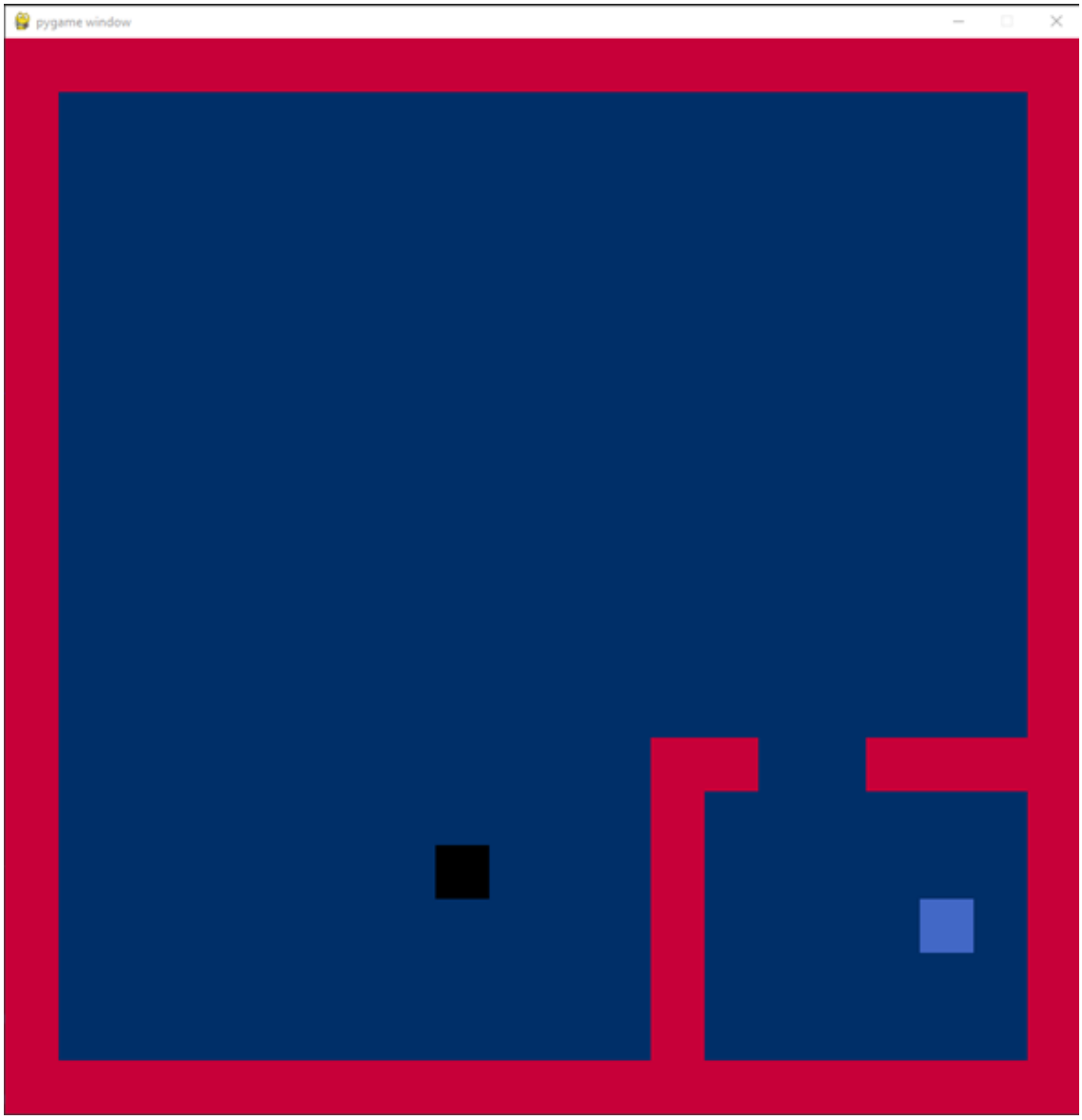


Figura 32: T=1.

Para la primera evolución se aplica la regla 1, donde el campo libre va a detectar el punto inicial, sin embargo, dentro de cada celda que es candidato a nacer o bien para pasar al estado q7, 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 32.

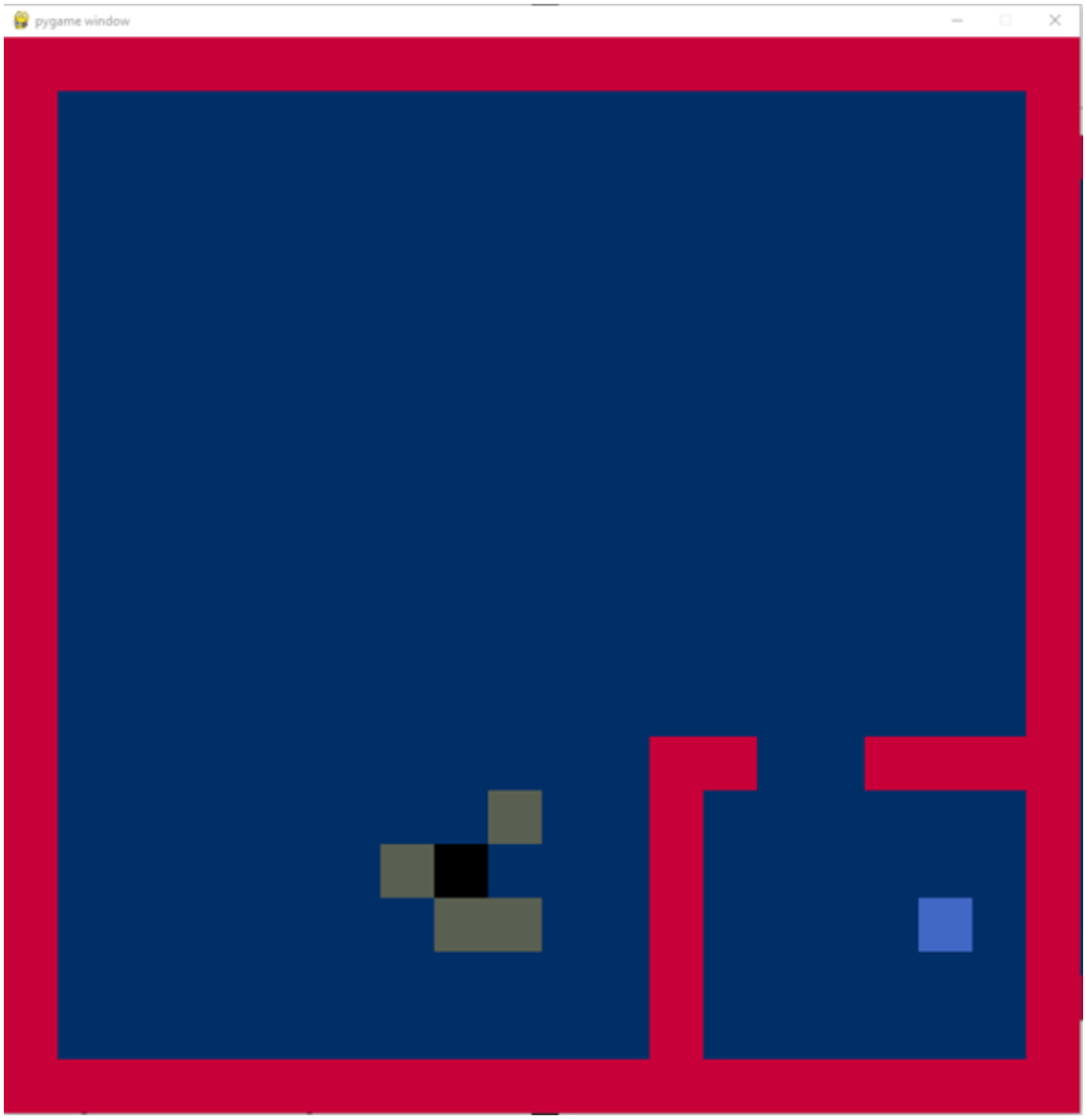


Figura 33: T=2.

En la segunda evolución nuevamente aplica la regla 1 para el campo libre que evolucionó, sin embargo, se observa que la célula en estado q7, pasa a un estado q4, por criterio de la regla 8, mostrado en la Figura 33.

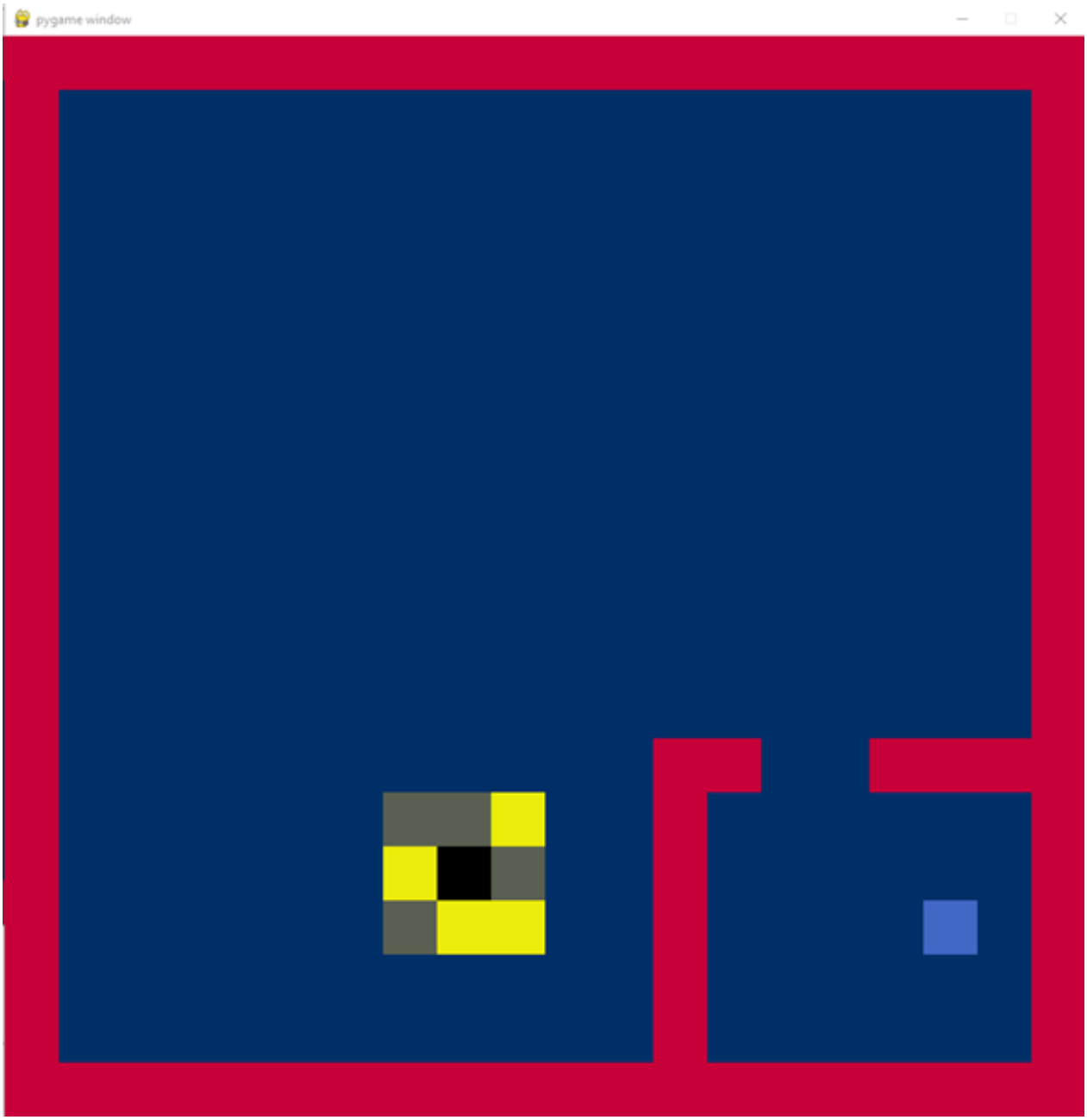


Figura 34: T=3.

Observamos que nace una nueva célula con q7, por la misma regla 1 y anteriormente la célula que estaba en estado q7, cambio a q4, por la regla 2.

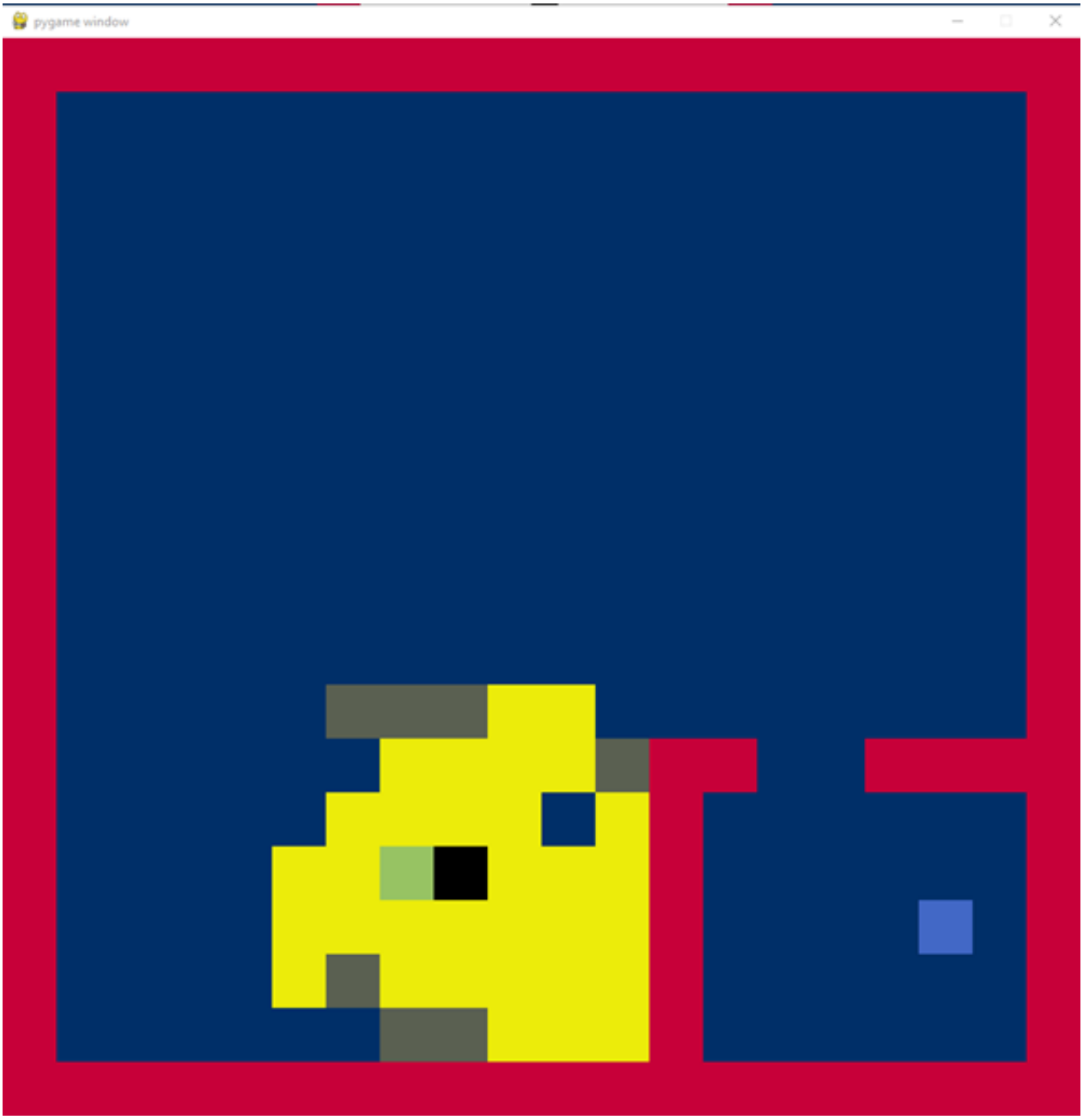


Figura 35: T=7.

Seguimos bajo las mimas reglas, hasta que por criterio de la regla 5, una célula en q4 pasa a q5, esta 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 para después pasar a q8, véase la Figura 35.

Figura 36: T=8.

En la siguiente evolución, la célula en q8, pasa a q5 por la regla 5, Ecuación 14, este caso solamente representa como un transporte o movimiento dentro de la estructura del plasmodio, además al tener dependencia, no puede morir, véase la Figura 36.

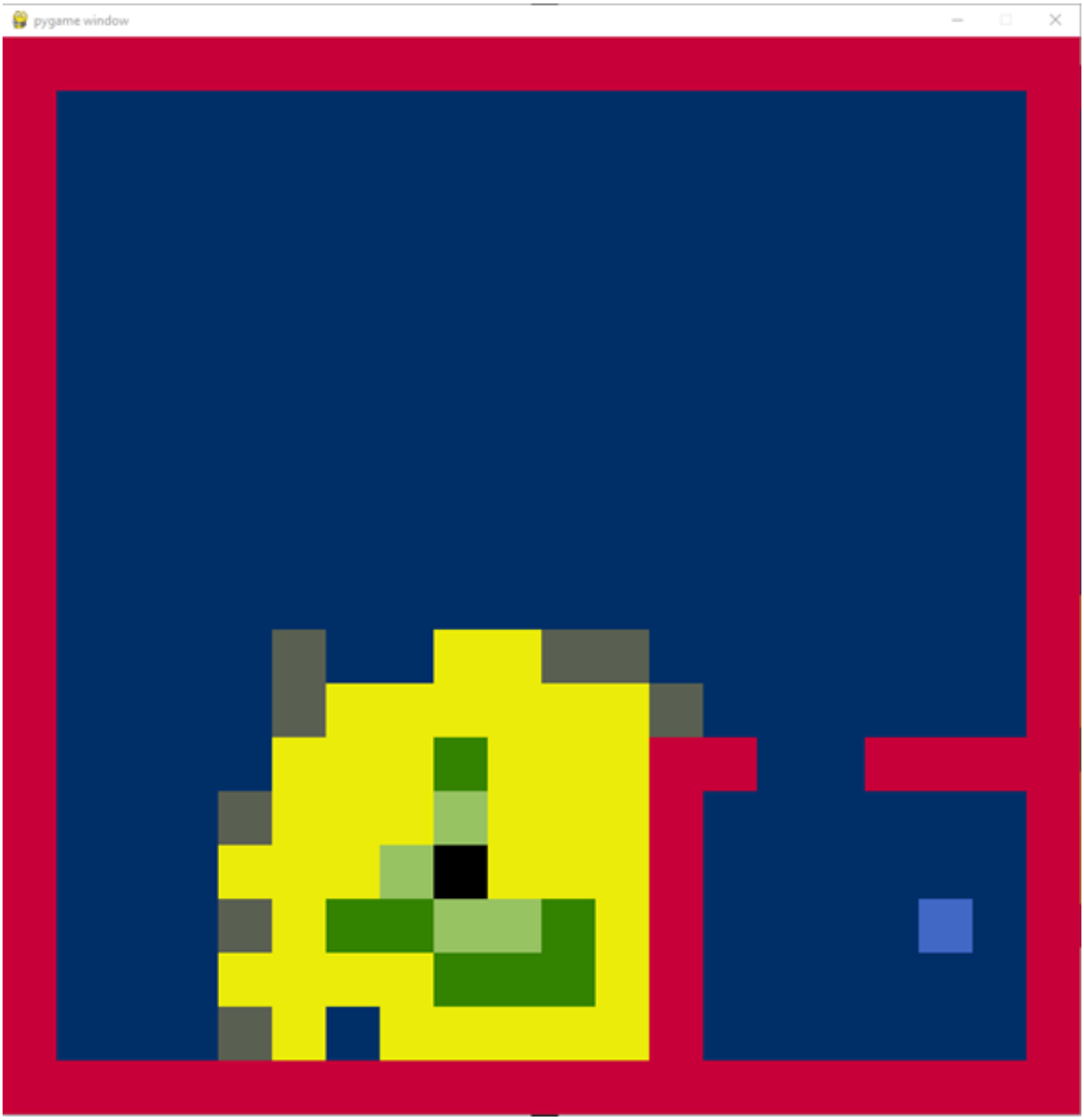


Figura 37: T=9.

En la siguiente transición la célula en estado q5, regresa a q8, por la regla 6, el resto de los elementos siguen teniendo su evolución a consecuencia de reglas anteriores.

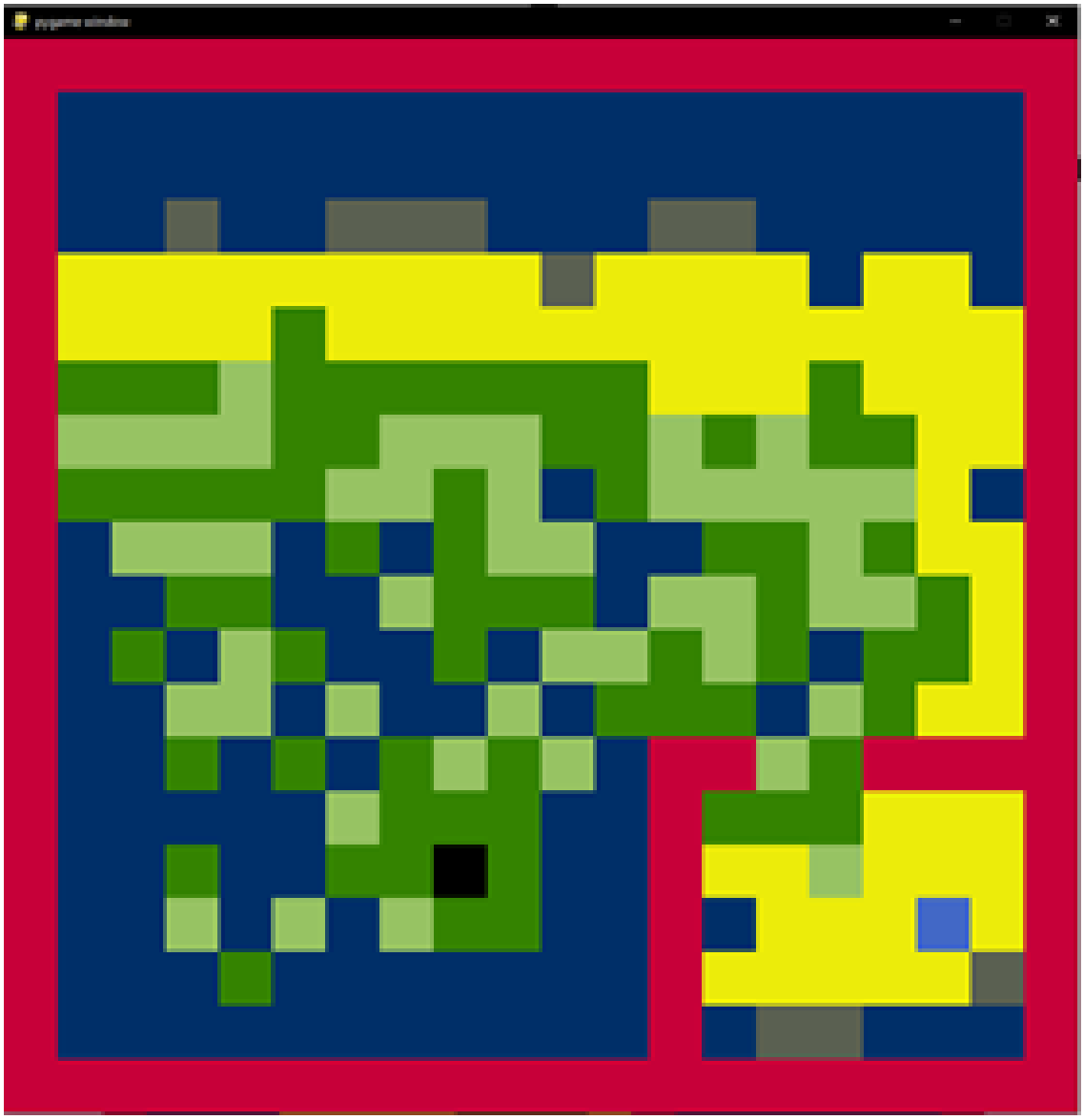


Figura 38: T=20.

Tras una serie de evoluciones empiezan a morir células, con base a la regla 5 (Figura 38) donde la célula marcada, (Figura 39) morirá debido a que no puede extenderse o encontrar alguna dependencia con algún 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.

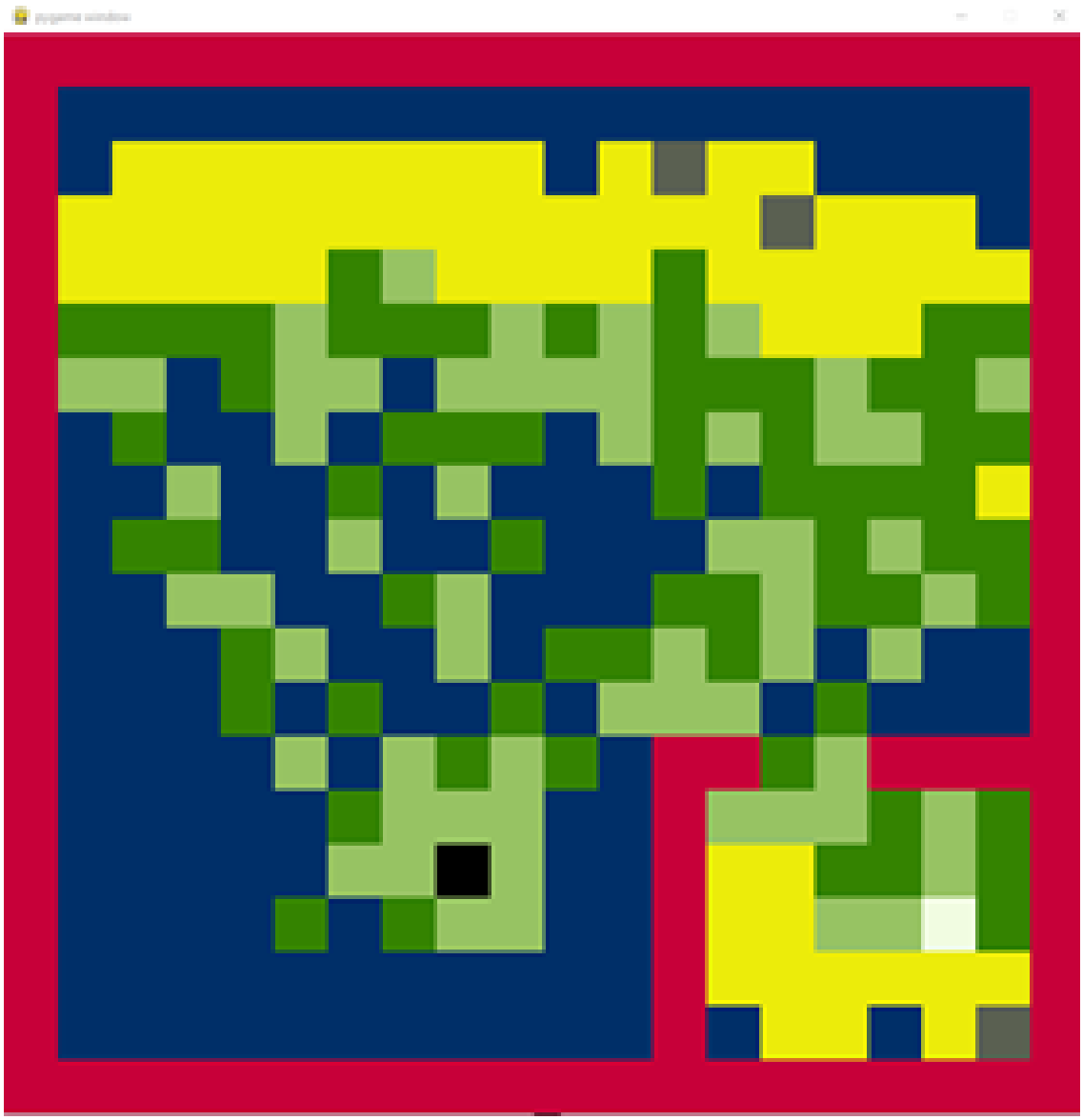


Figura 39: T=25.

Traducido al Physarum, recordemos que el organismo puede saber dónde ya ha estado, entonces para el ya no es necesario volver a explorar la zona donde desapareció puesto que sabe que ahí al menos en su primera exploración habrá nada, esto se puede ver de una mejor manera prestando atención al cuadrado morado en las figuras anteriores.

Toma hasta la generación 20, para que el Physarum se extienda al nutriente donde este cambia de estado q1 a q6, por la regla 2. Para la generación 25, 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 organismos logra cubrir y para este caso, puede pasar sobre él para seguir explorando, entonces la zona libre pasa a estado q7 por la regla 1.

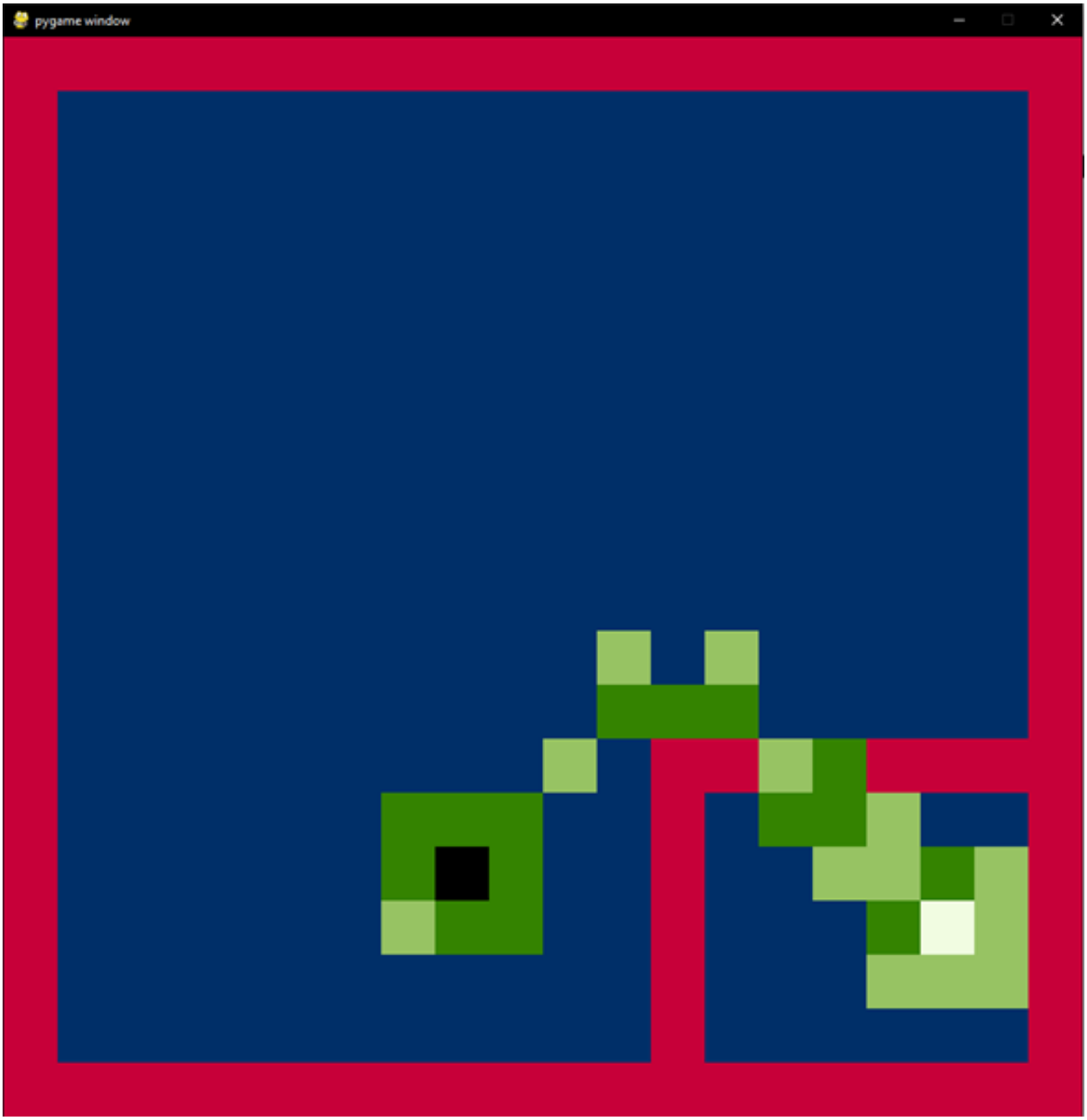


Figura 40: T=43.

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.

6.1.4. Segunda versión

Para este avance se agregaron nuevos comandos con teclas en el simulador, un contador de las generaciones y una gráfica para las generaciones. Los comandos con las teclas son:

- Tecla espaciadora: Pausar / reanudar simulación
- Tecla C: Reiniciar simulación
- Tecla A: Abrir archivos para cargar simulación
- Tecla G: Guardar la simulación en un archivo
- Tecla P: Graficar el crecimiento de estados hasta el punto actual.

Figura 41: Pantalla inicial.

En la Figura 41, se puede apreciar el simulador al correrlo con la generación en 0.

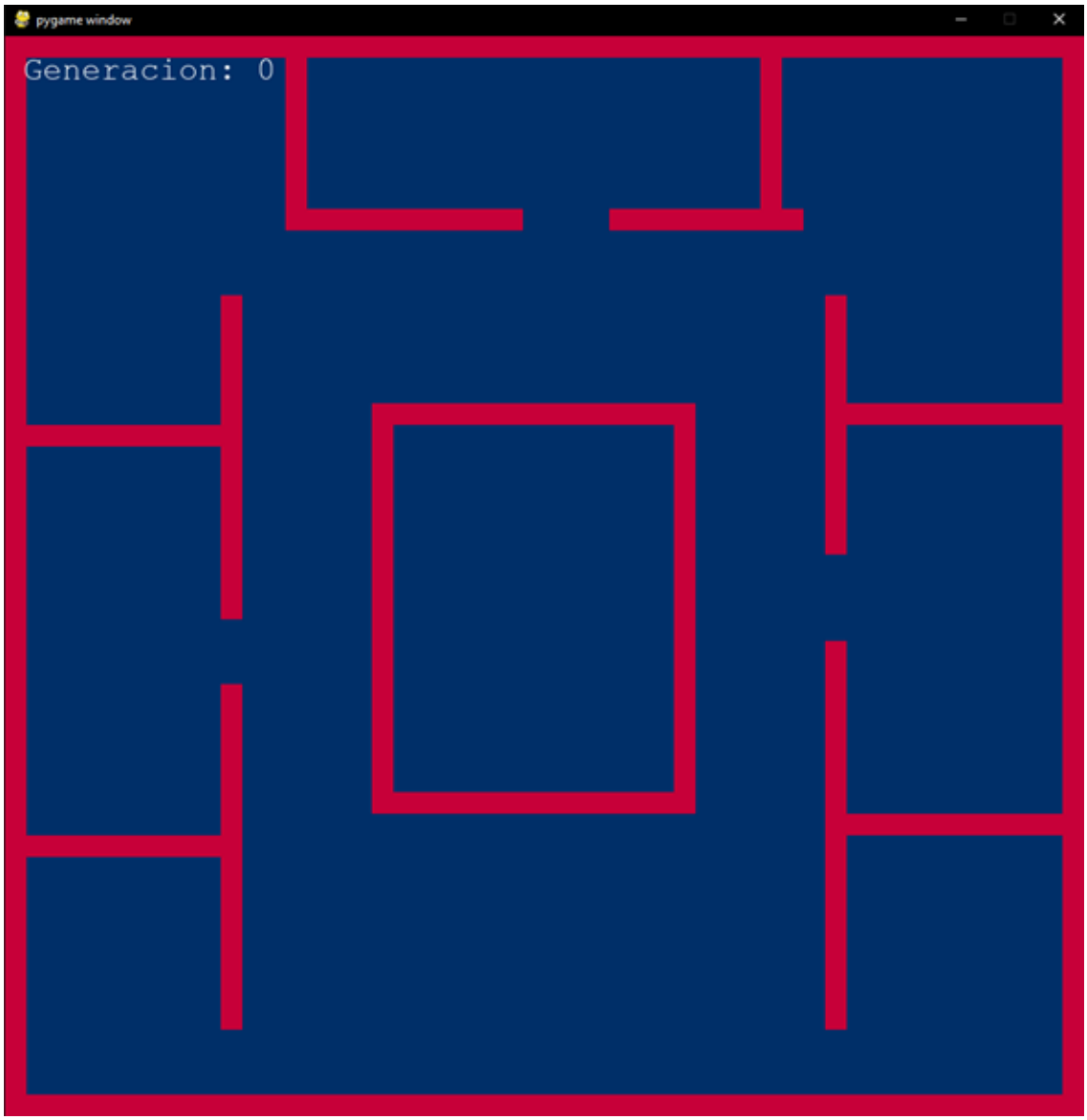


Figura 42: Mapa para realizar las pruebas.

En la Figura 42 se dibujó una representación de un piso con diferentes habitaciones, en este caso las paredes son representadas por el estado 2 que es el repelente.

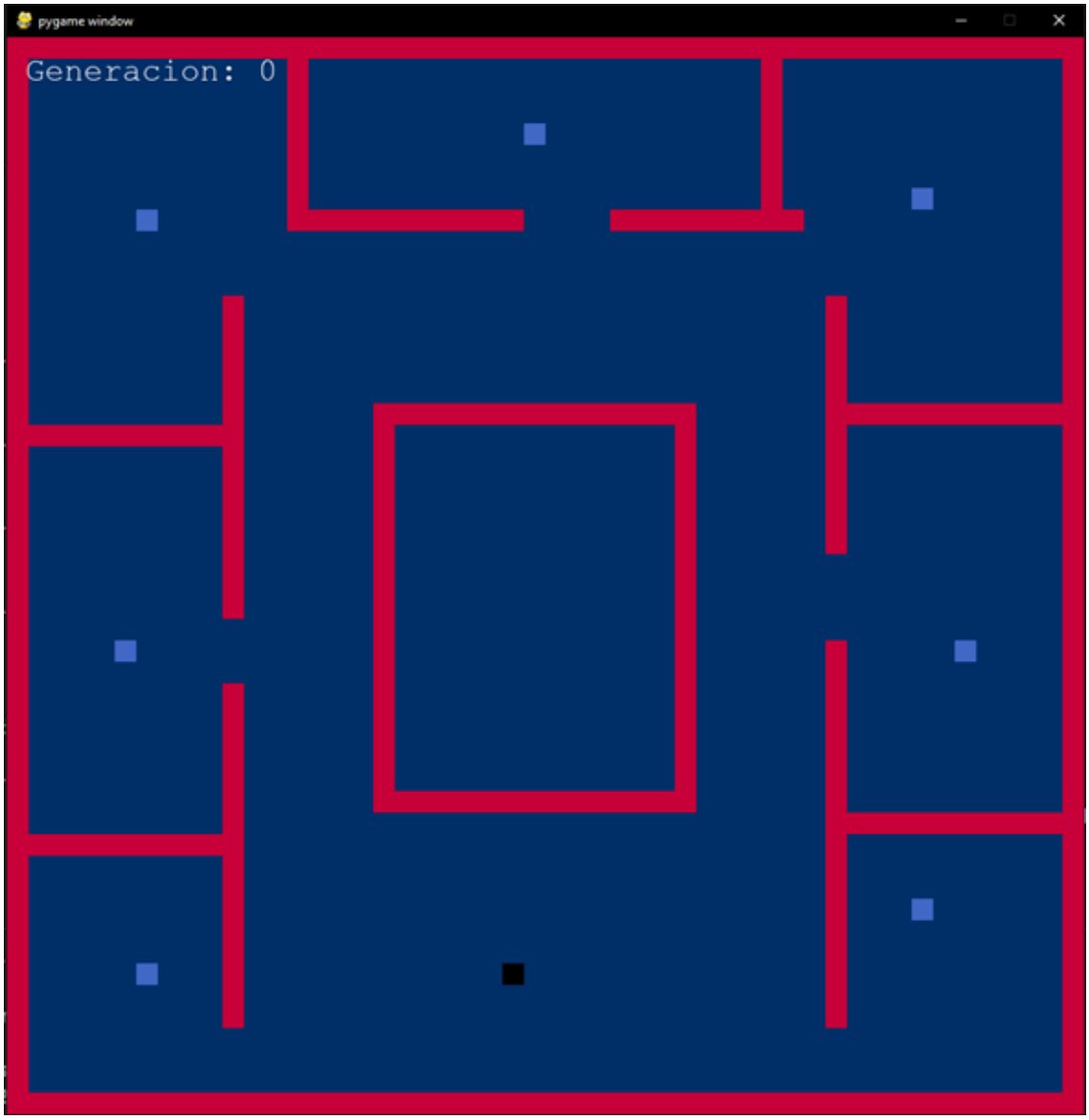


Figura 43: T=0.

En la Figura 43 se muestra una configuración inicial en donde se encuentran 7 nutrientes repartidos por todo el mapa y un punto inicial.

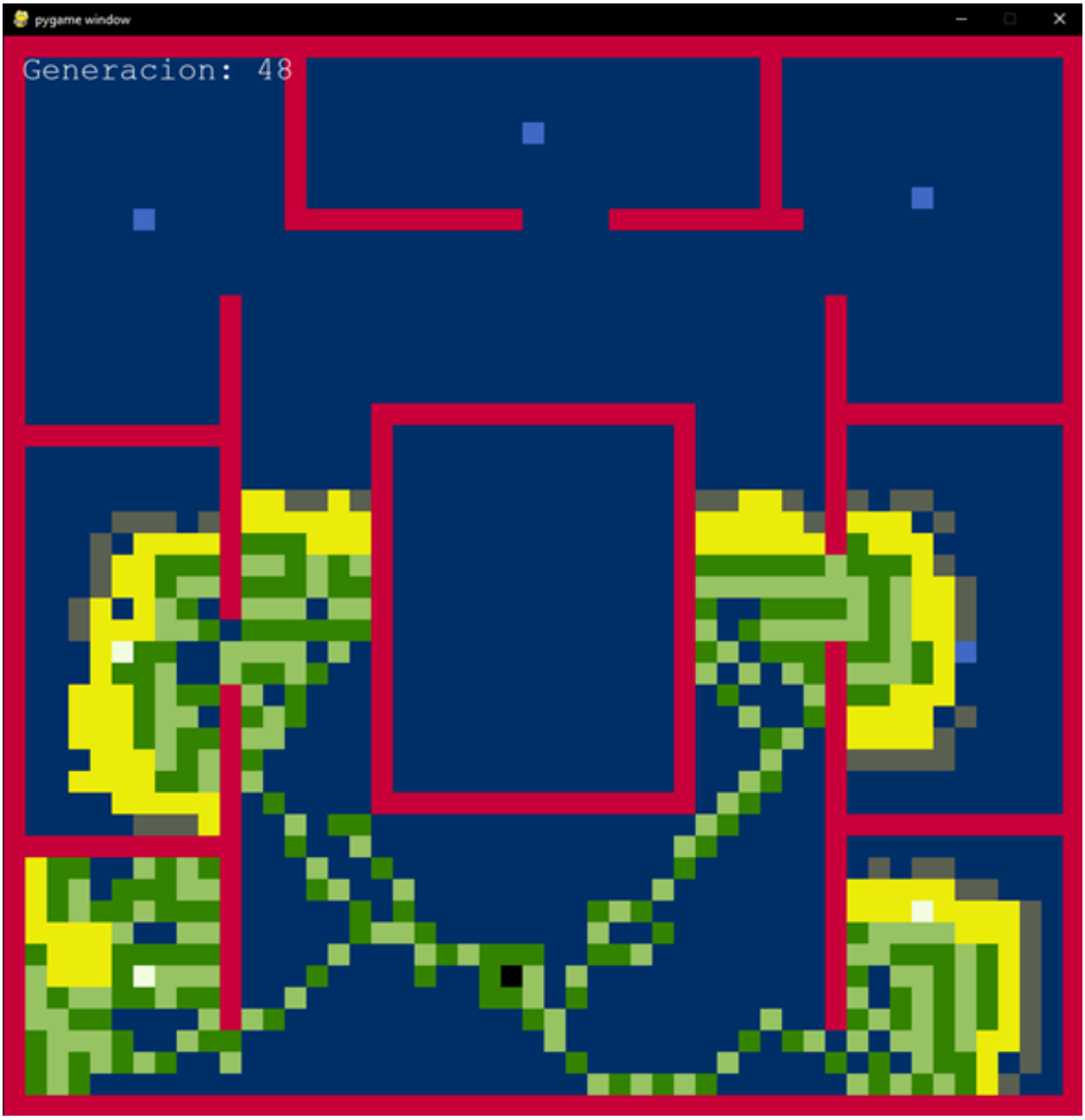


Figura 44: T=43.

En la Figura 44 podemos ver como se hace contacto con los primeros nutrientes formando ya algunas rutas.

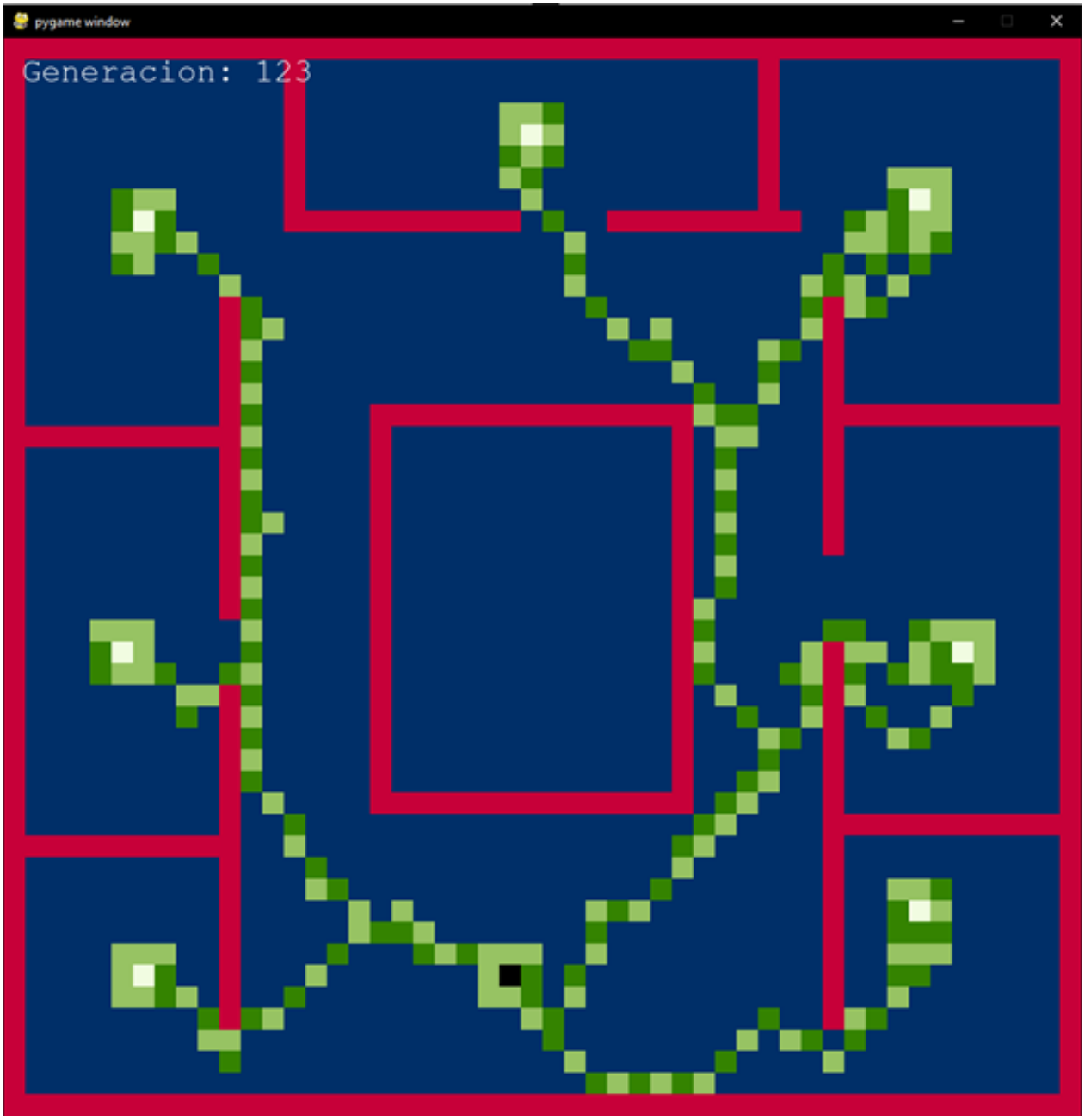


Figura 45: T=123.

En la Figura 45 podemos apreciar que el sistema se estabiliza y ya se formaron las rutas desde el punto inicial hasta los nutrientes.

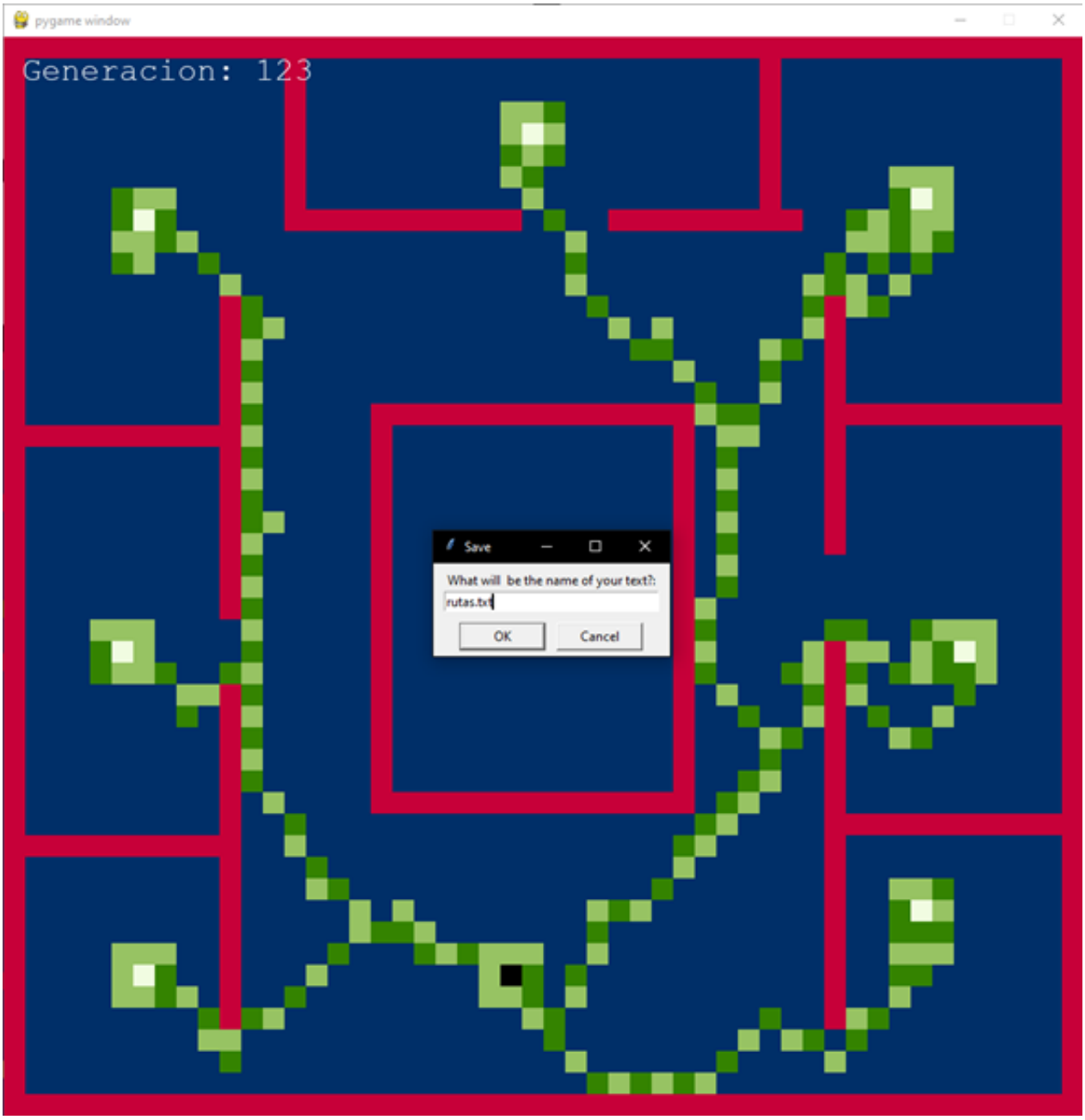


Figura 46: Guardado del archivo

En la Figura 46 podemos apreciar como guardamos el archivo donde el usuario escoge el nombre del archivo, este archivo se guarda por default en una carpeta llamada “save”.

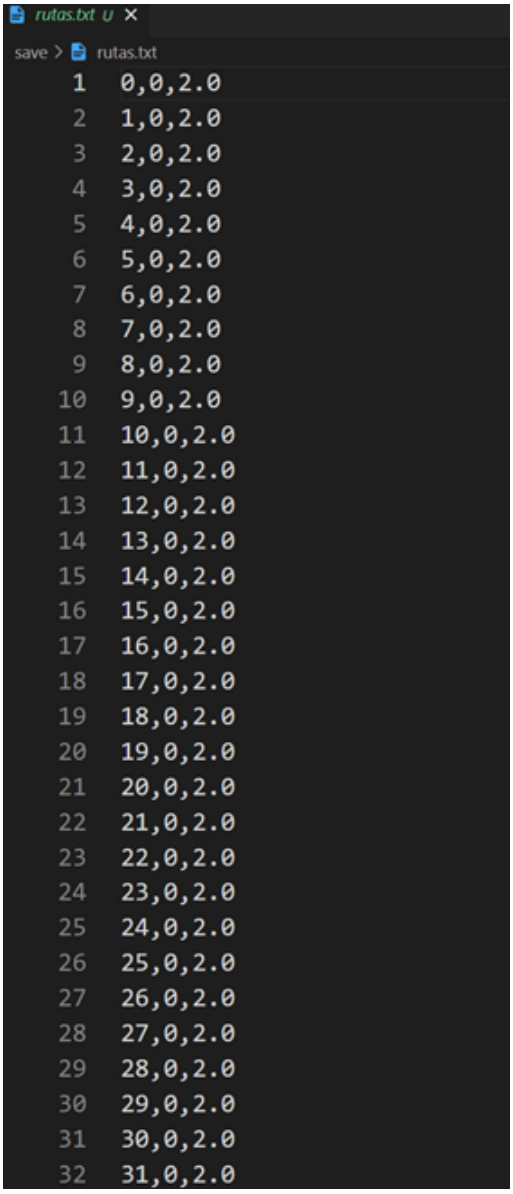


Figura 47: Formato del archivo guardado.

Para el guardado de los archivos, guardamos su posición y su estado, en la figura 47 se muestra cómo se guardan los estados del margen, que se encuentran en el estado 2.

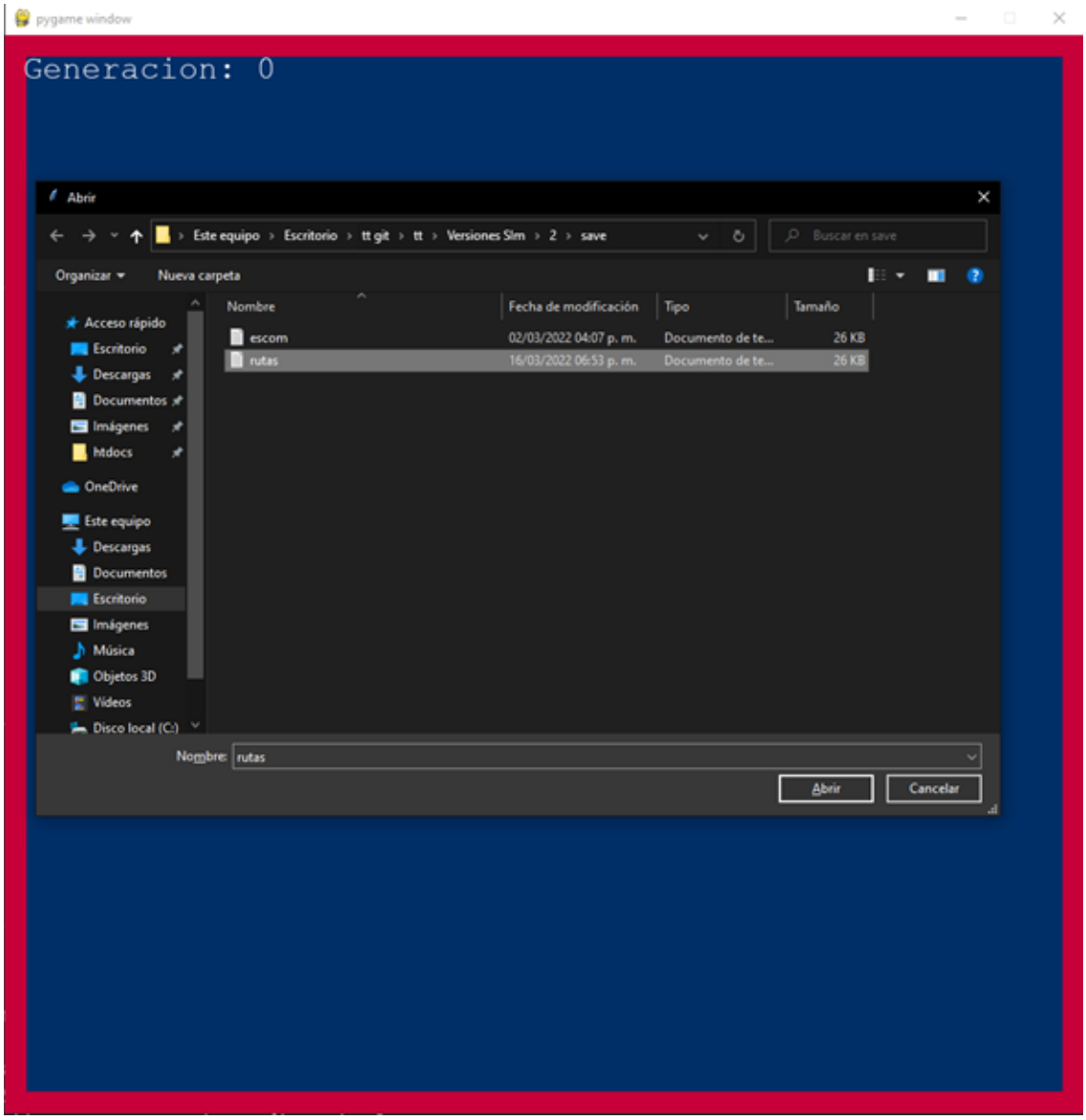


Figura 48: Menú para buscar archivo.

En la figura 48 podemos apreciar cómo después de presionar la tecla A, nos abre un menú para buscar el archivo a abrir.

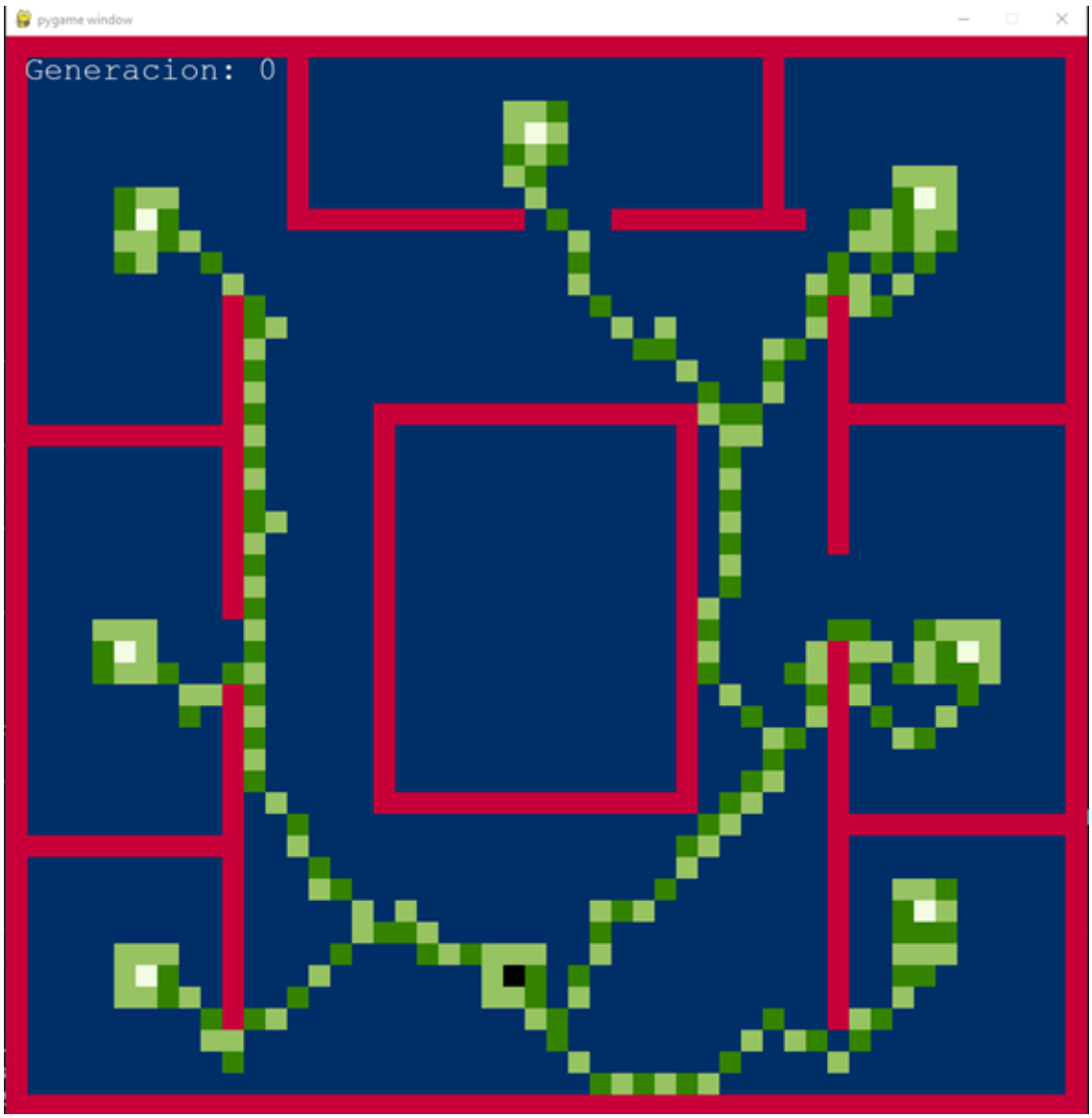


Figura 49: Carga de la configuración desde el archivo.

En la Figura 49 es la configuración que se guardó en la Figura 46, un área de oportunidad en este caso es almacenar también la generación en la que se guardo para mantener la integridad de los datos.

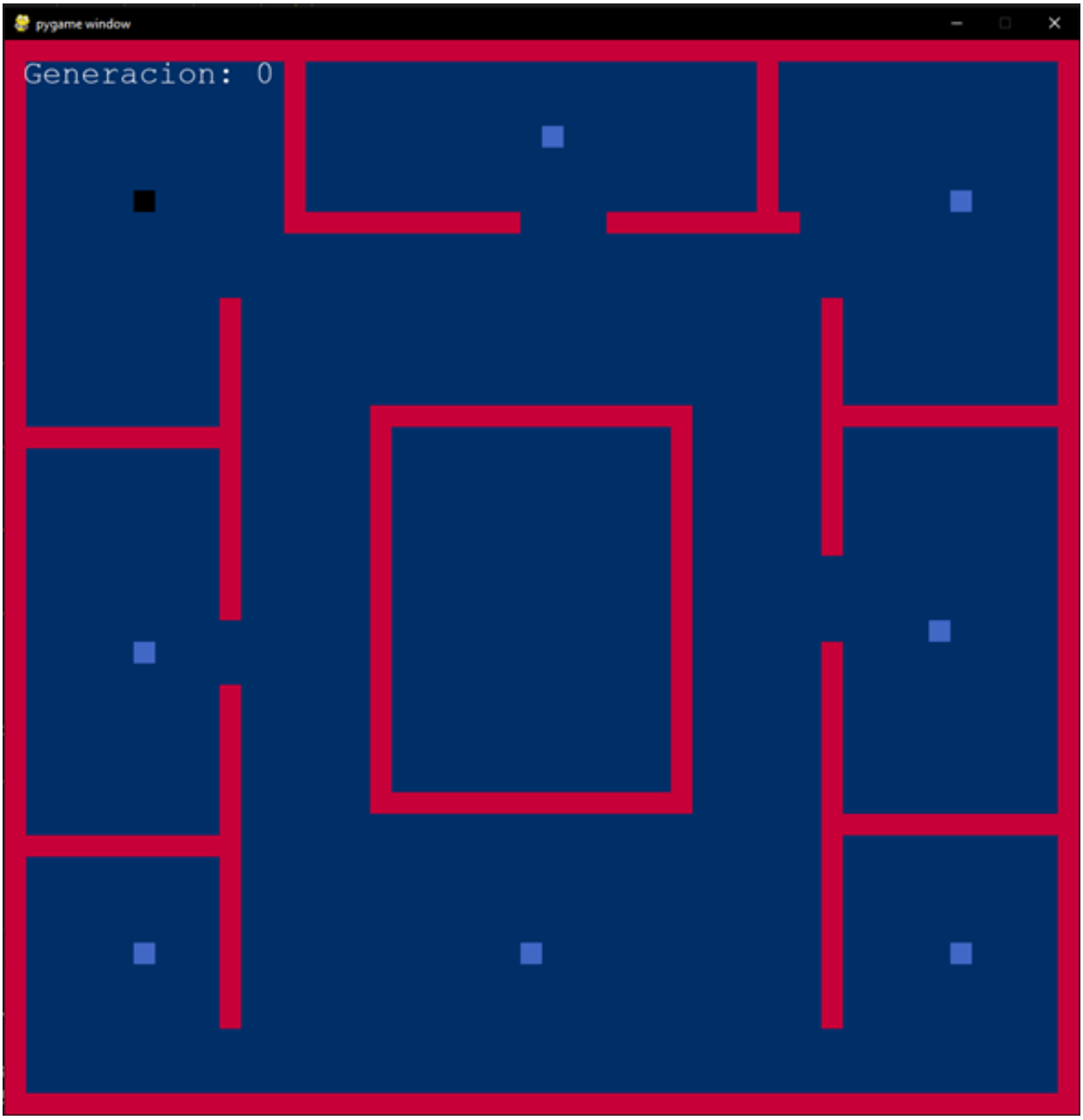


Figura 50: Configuración inicial para la gráfica.

Para mostrar la grafica del crecimiento de los estados se utilizará la configuración inicial de la Figura 50.

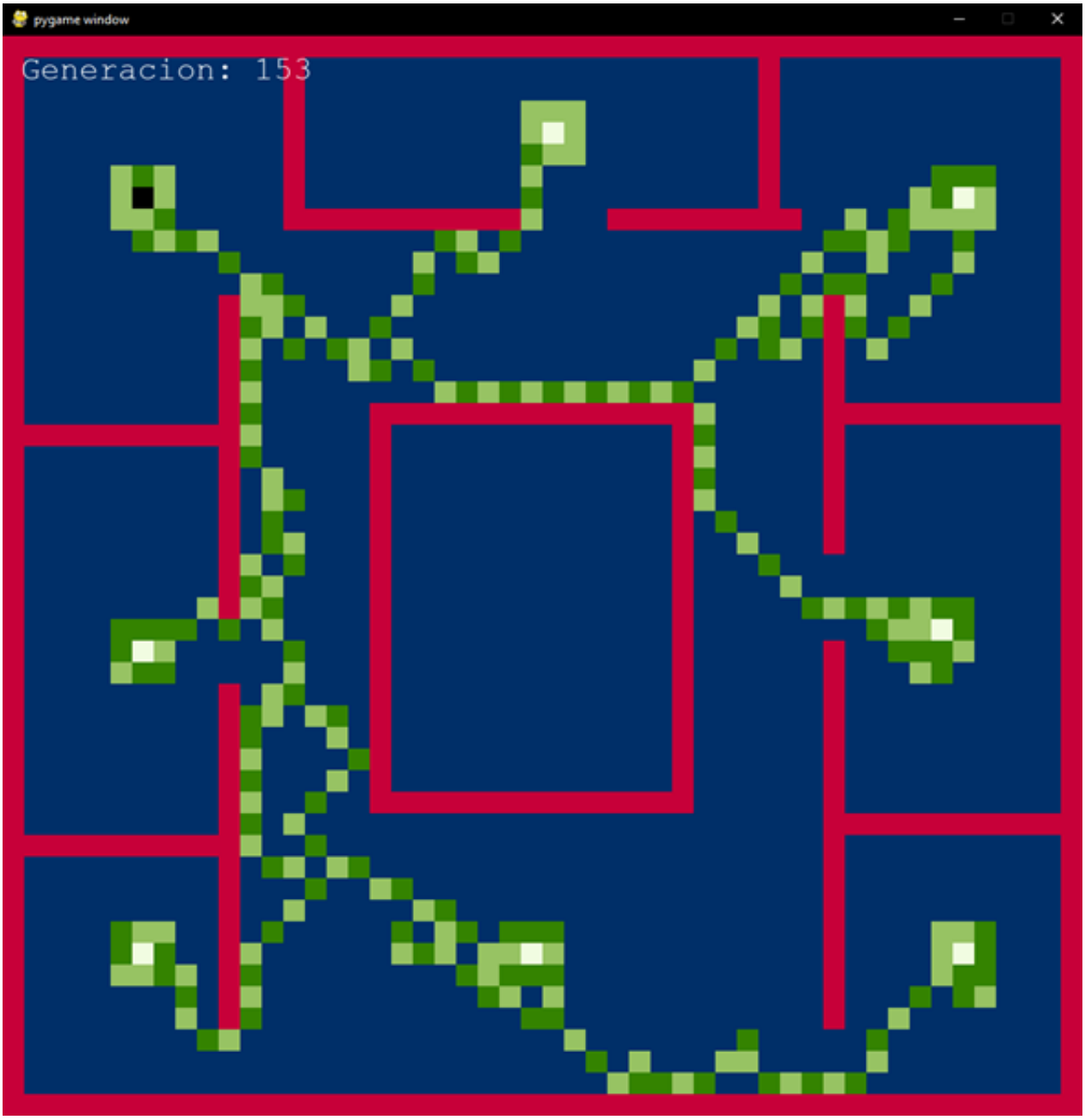


Figura 51: Prueba de gráfica, T=153.

Fue hasta la generación 153 cuando el sistema se pudo estabilizar.

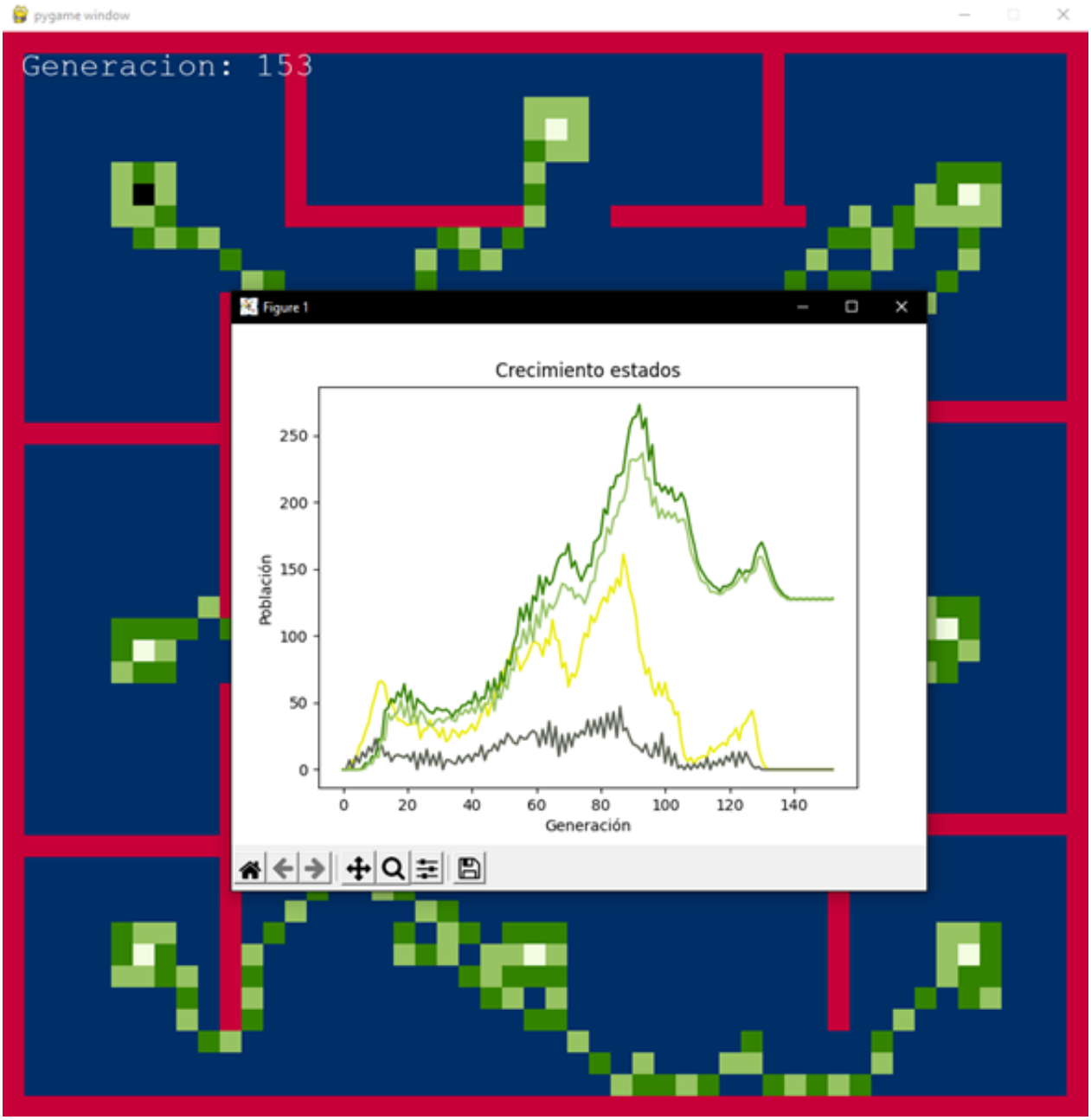


Figura 52: Gráfica de crecimiento.

En la Figura 52 se aprecia que después de presionar la tecla P, se muestra una grafica con los principales 4 estados que conforman el modelo Physarum.

6.1.5. Tercera versión

Para este avance se incluye un menú antes de correr el simulador donde le usuario puede seleccionar los colores de los 4 principales estados, puede escoger la longitud del lado de las células y qué regla utilizar.

Para este prototipo se manejan 2 reglas:

- Regla 1: Comportamiento normal.
- Regla 2: Distancia de una célula con respecto a los repelentes para evitar colisiones.

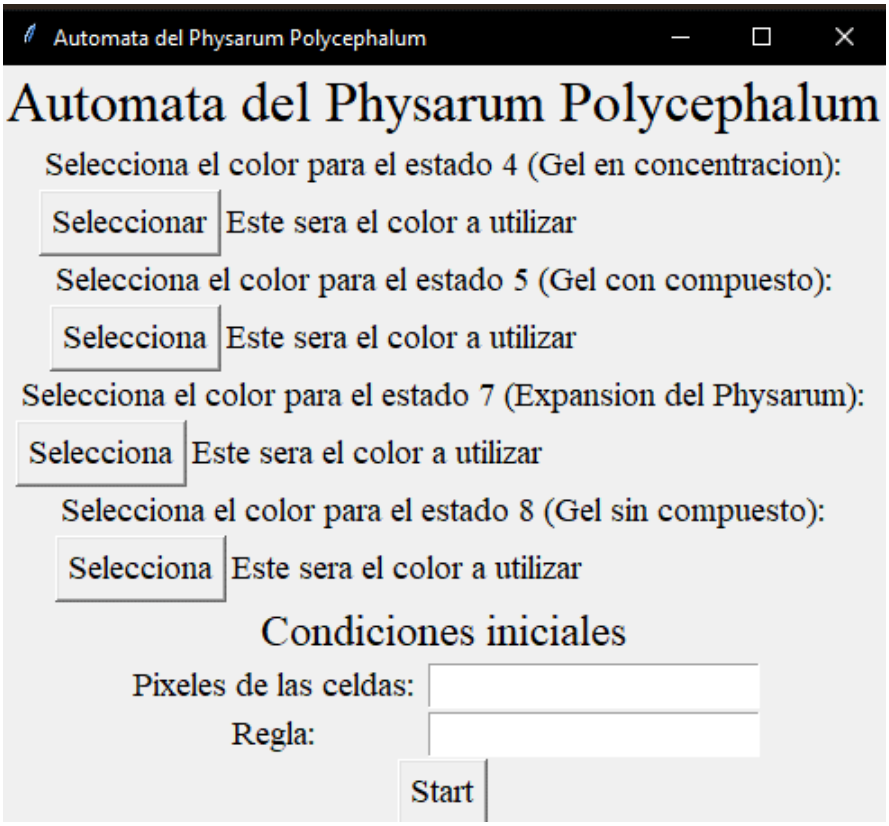


Figura 53: Menú inicial de la versión tres.

En la Figura 53 se muestra el menú mencionado con anterioridad, cabe mencionar que, si no se selecciona algún color, se mantendrán los colores por defecto de las versiones anteriores.

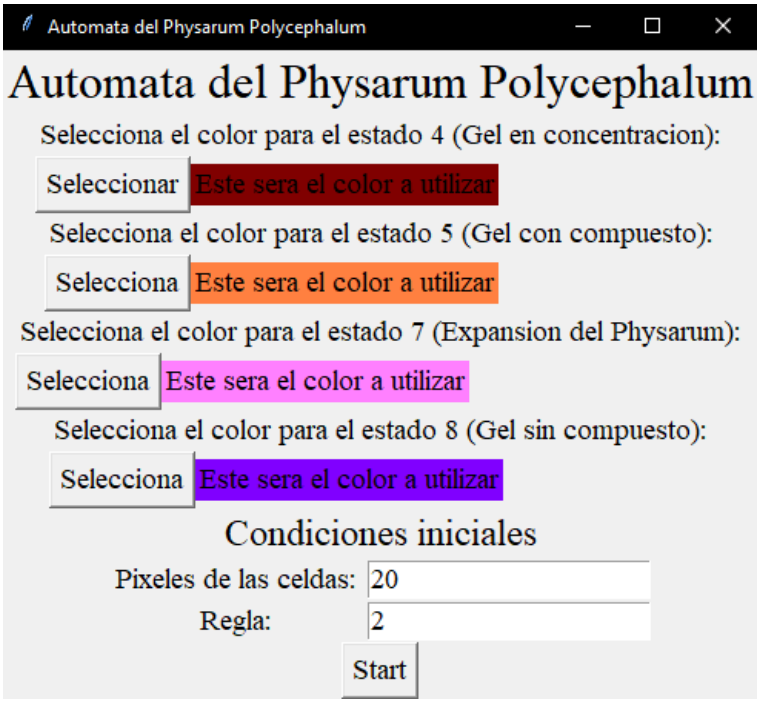


Figura 54: Menú con colores seleccionados.

En la Figura 54 se muestra el menú seleccionando los colores a utilizar, también se elige la regla 2 en la cual se trata de dejar una célula de espacio entre el physarum y los repelentes.

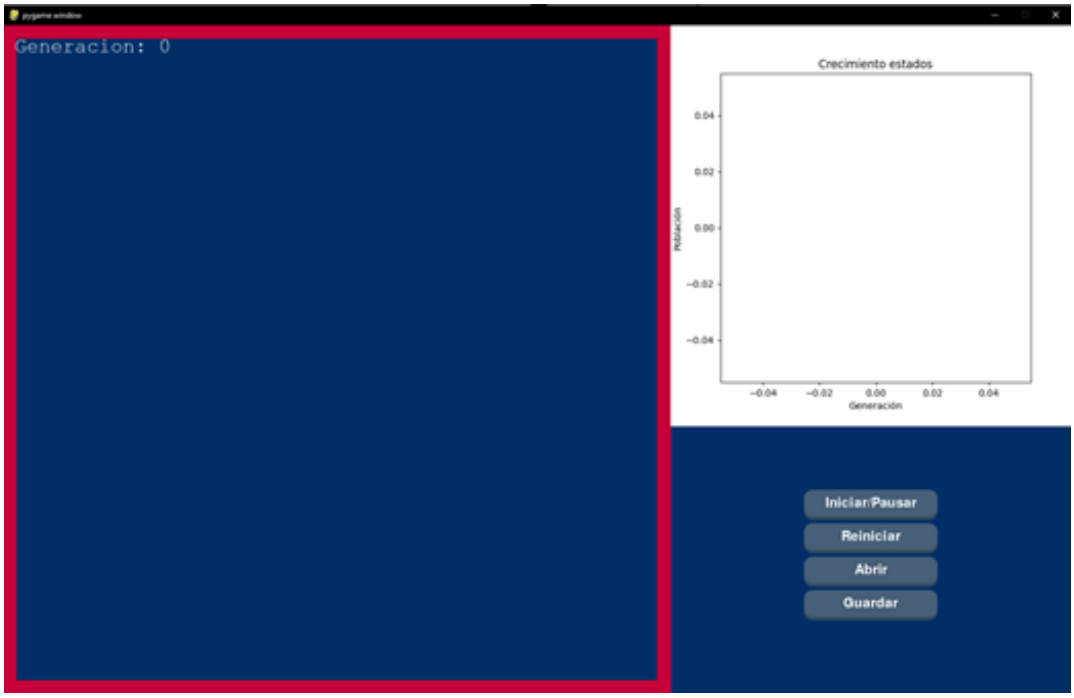


Figura 55: Nueva interfaz para el simulador.

En la Figura 55 podemos apreciar el nuevo entorno grafico del simulador en donde se remplazaron los comandos de las teclas por botones para que sea más amigable para el usuario, también la gráfica se actualiza en tiempo real.

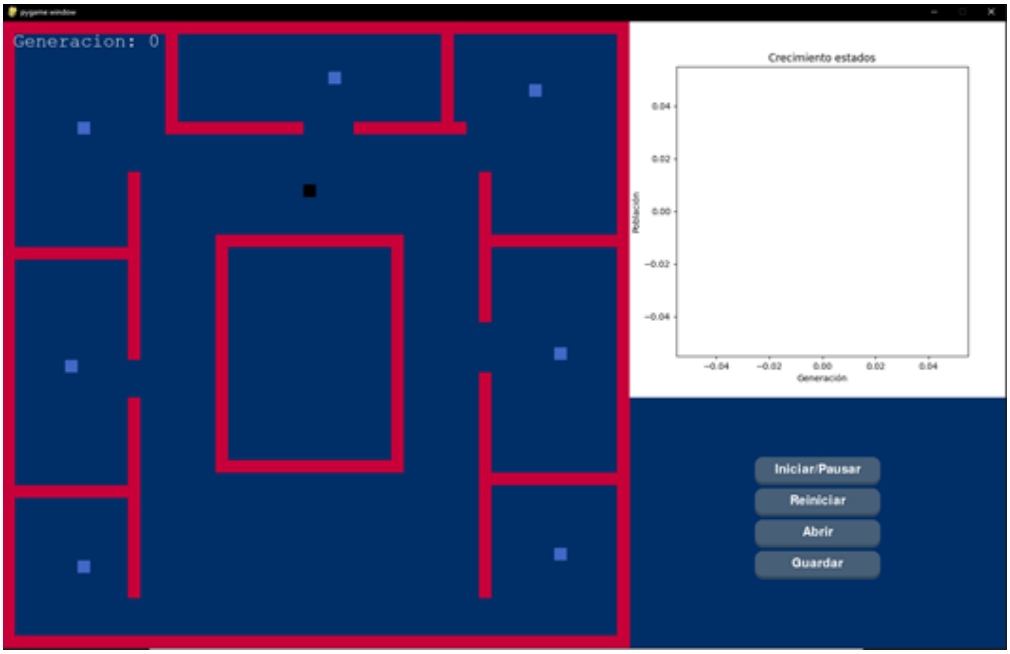


Figura 56: Interfaz con configuración cargada.

En la Figura 56 se abre el archivo con nuestro piso construido de las versiones anteriores y se establecen algunos nutrientes con el punto inicial en la parte superior.

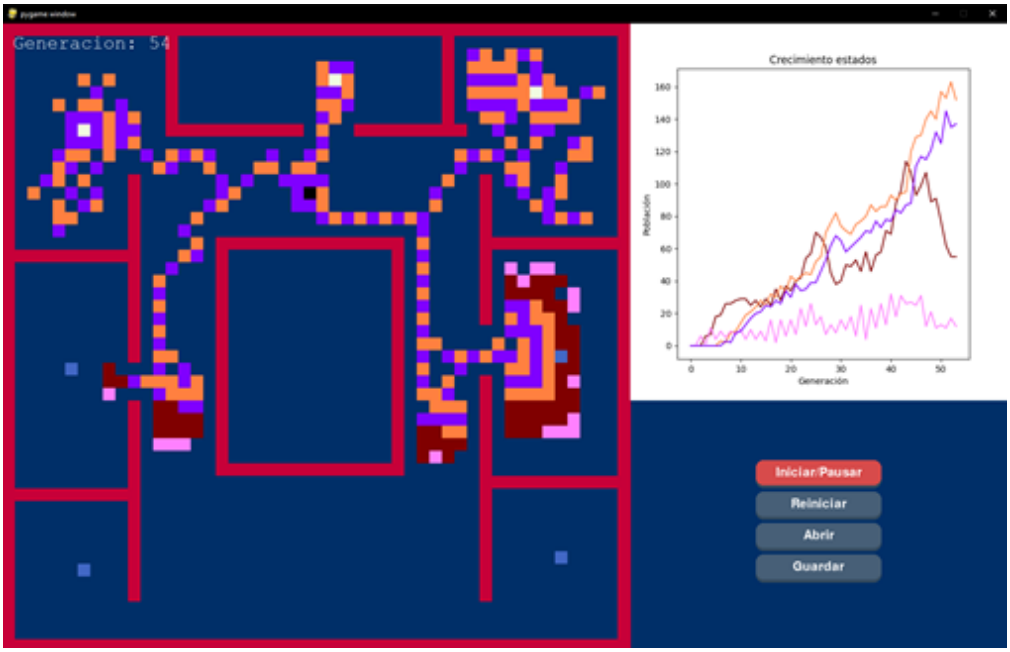


Figura 57: Evolución del modelo y gráfica en progreso.

En la Figura 57 se ve el progreso de nuestra configuración inicial, se puede apreciar el cambio de colores en las células y en la gráfica.

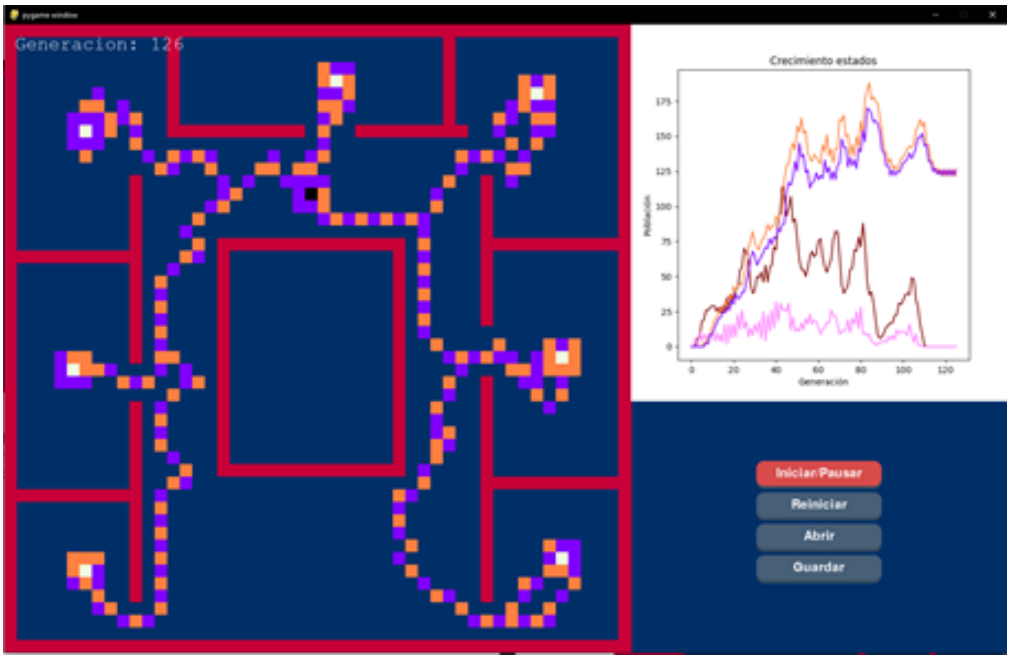


Figura 58: Rutas del modelo.

En la figura 58 la simulación se estabiliza y se puede apreciar que el autómata siempre

mantiene una distancia de al menos un bloque con respecto a los repelentes debido a que se estableció la regla 2.

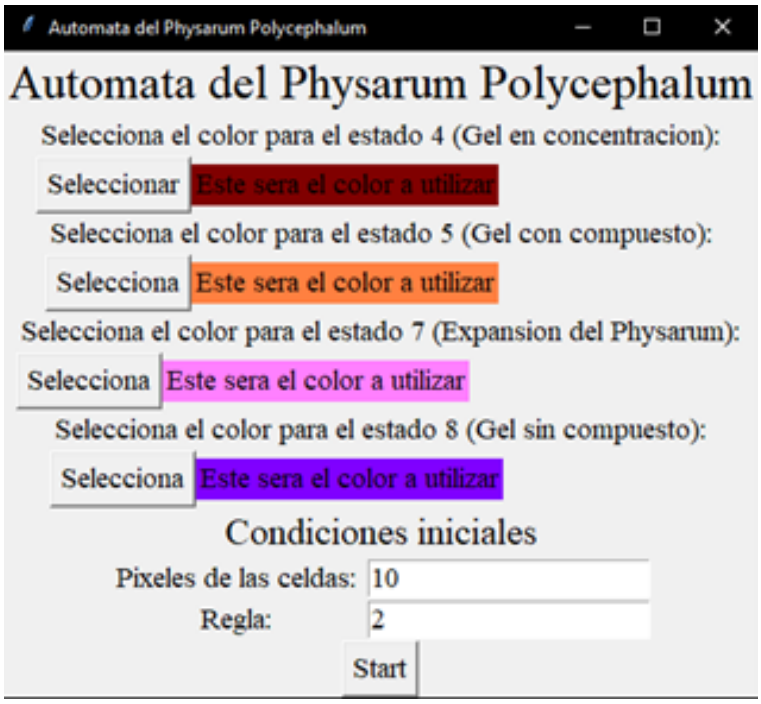


Figura 59: Menú con tamaño de 10.

En la Figura 59 se aprecia que ahora trabajaremos con células de tamaño 10, esto es con el propósito de usar la configuración del piso ya hecha en otros tamaños para la célula, especialmente porque puede ser un proceso tedioso hacerlo célula por célula.

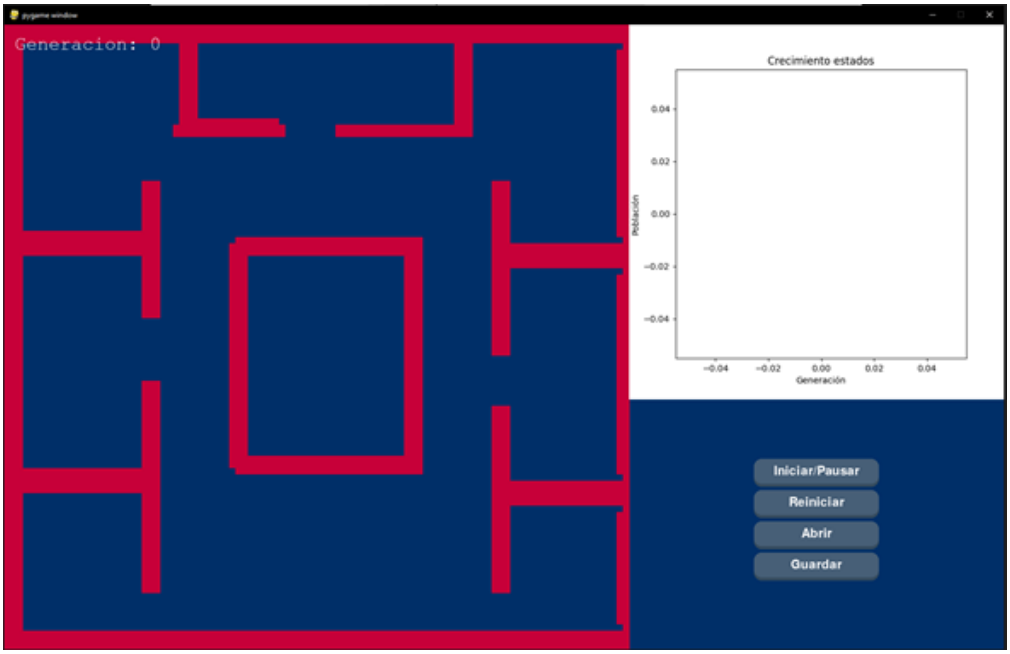


Figura 60: T=0.

En la Figura 60 se muestra cómo está la configuración del piso en donde originalmente se guardó para una configuración de 20 pixeles por lado, como en este caso es de 10 pixeles, tenemos una cuadrícula de 100x100, por el momento solo funciona para los repelentes. Para lograr esto se crea una imagen con la configuración original para luego estirarla.

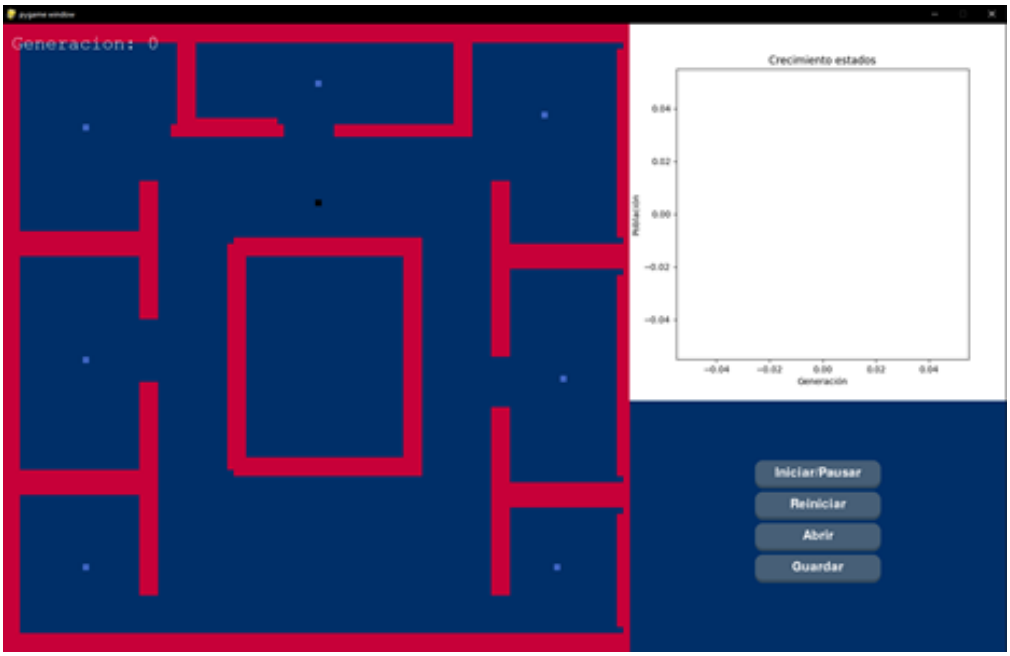


Figura 61: Carga del archivo en distinto tamaño.

En la Figura 61 se intenta recrear el estado inicial de la Figura 56.

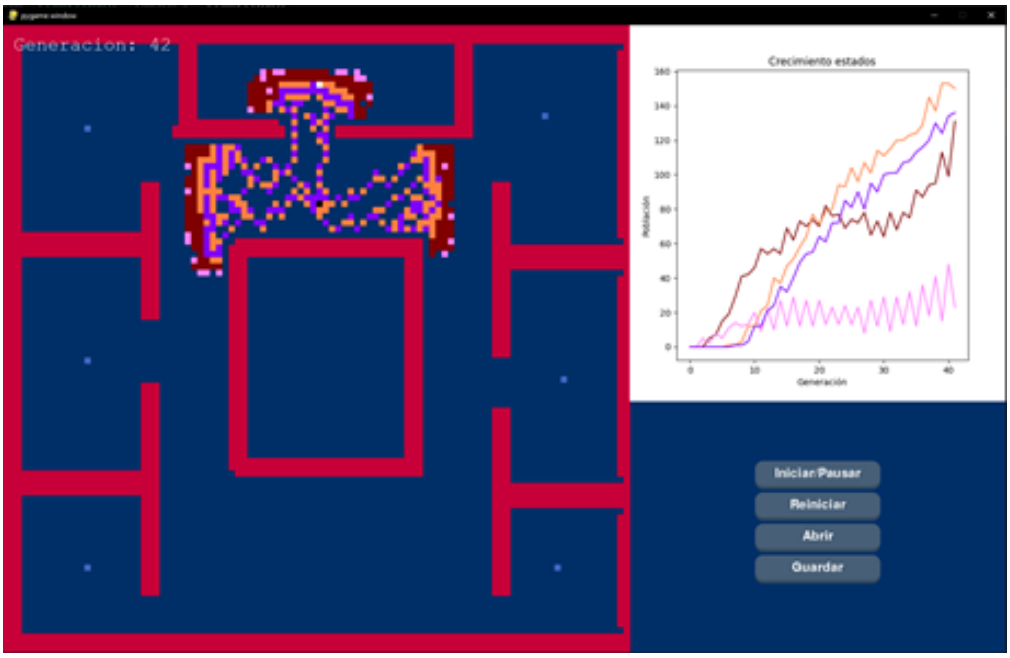


Figura 62: Evolución del modelo en distinto tamaño.

En la Figura 62 se muestra el avance en la generación 42 del automata.

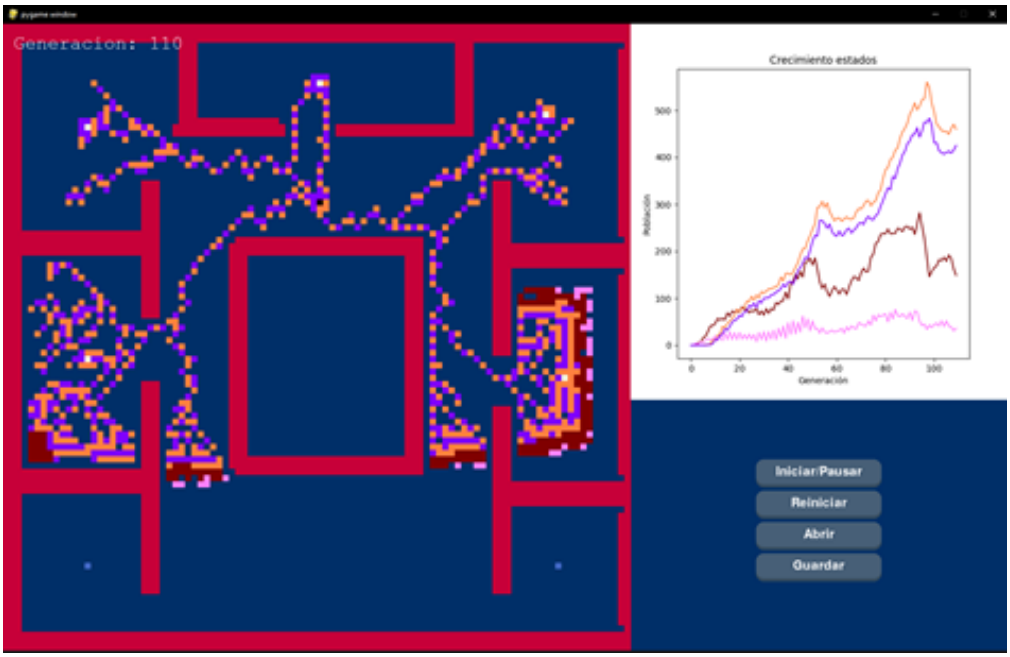


Figura 63: T=110.

En la Figura 63 se muestra el avance en la generación 110 del automata. Es destacable que

el cuadro blanco como se respeta la regla establecida de dejar por lo menos 1 bloque de distancia de los repelentes.

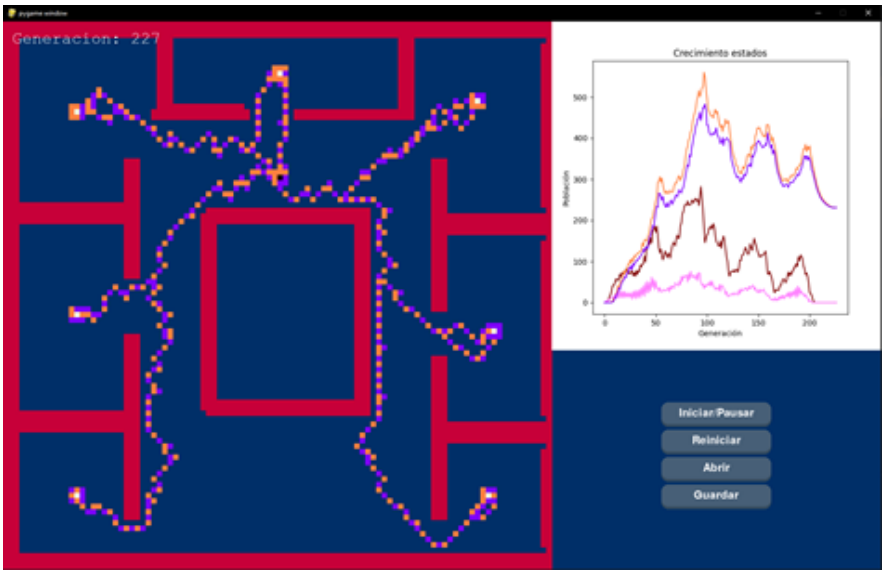


Figura 64: Modelo con rutas estables en distinto tamaño.

En la Figura 64 se muestra el sistema ya estabilizado en la generación 227. Debido a que que el área de exploración es mucho mayor que en la Figura 58, donde se logró la estabilidad del sistema en la generación 126.

6.1.6. Cuarta versión

Para este avance se modificó la interfaz inicial del usuario como se muestra en la Figura 65. Se agregó un ComboBox en donde el usuario pueda seleccionar qué vecindad usar, ya sea la de Moore o la de Neumann, también puede escoger la cantidad de células que tendrá por cada lado, qué regla utilizar con un botón informativo sobre las reglas existentes y si desea mostrar la entropía, que para este avance solamente es una prueba ya que no está implementada y ocupa la información de las generaciones.

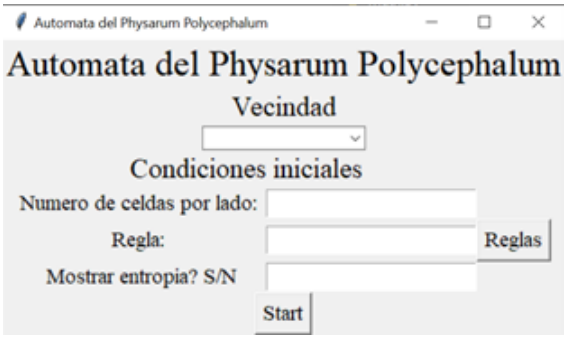


Figura 65: Nueva interfaz del menú.

En la figura 66 se muestra una configuración inicial en la cual el usuario escogió la vecindad de Moore, donde va a tener 80 células por lado, se ocupa la regla 1 y se decide mostrar la entropía.

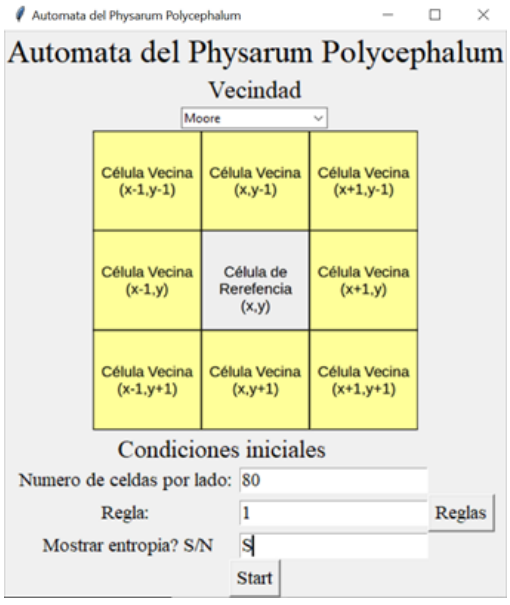


Figura 66: Selección de vecindad.

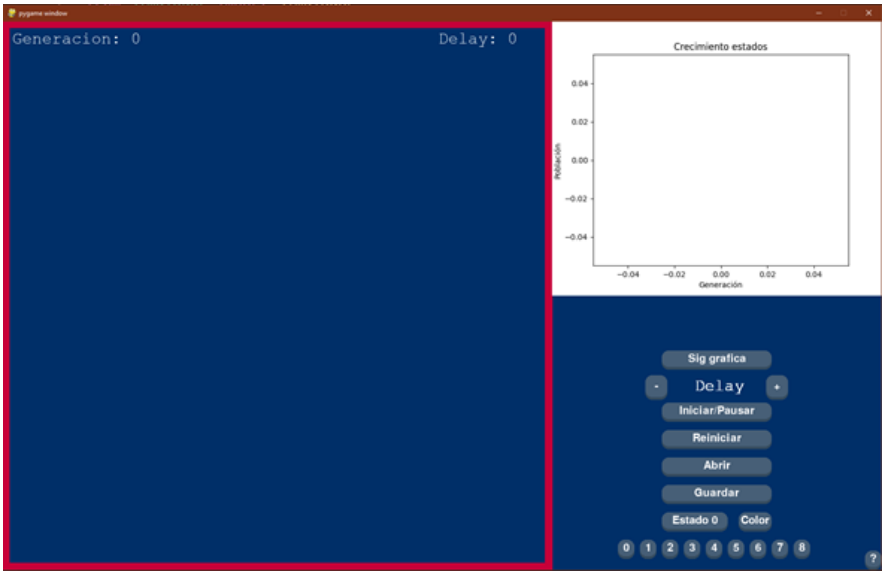


Figura 67: Nueva interfaz del simulador.

En la figura 67 se muestra cómo también se modificó la interfaz grafica del simulador, en donde se agregaron más elementos como un botón para cambiar la gráfica que se está mostrando, dos botones para controlar el delay del simulador, un recuadro el cual muestra el estado actual seleccionado, un botón para cambiar el color del estado seleccionado y un botón de ayuda para obtener información de los botones.

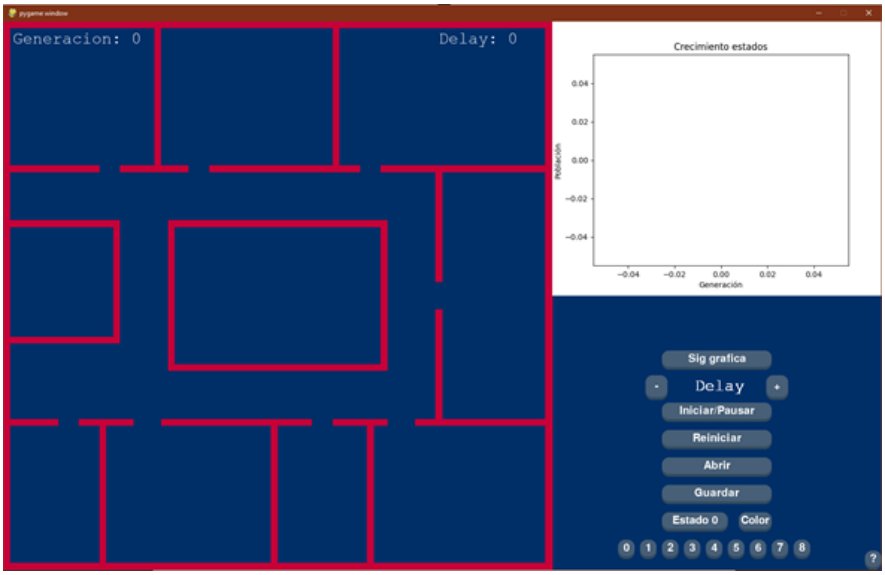


Figura 68: Nuevo mapa para hacer pruebas.

En la Figura 68 se abrió un nuevo mapa del edificio de gobierno siendo este más acercado

a las dimensiones reales.

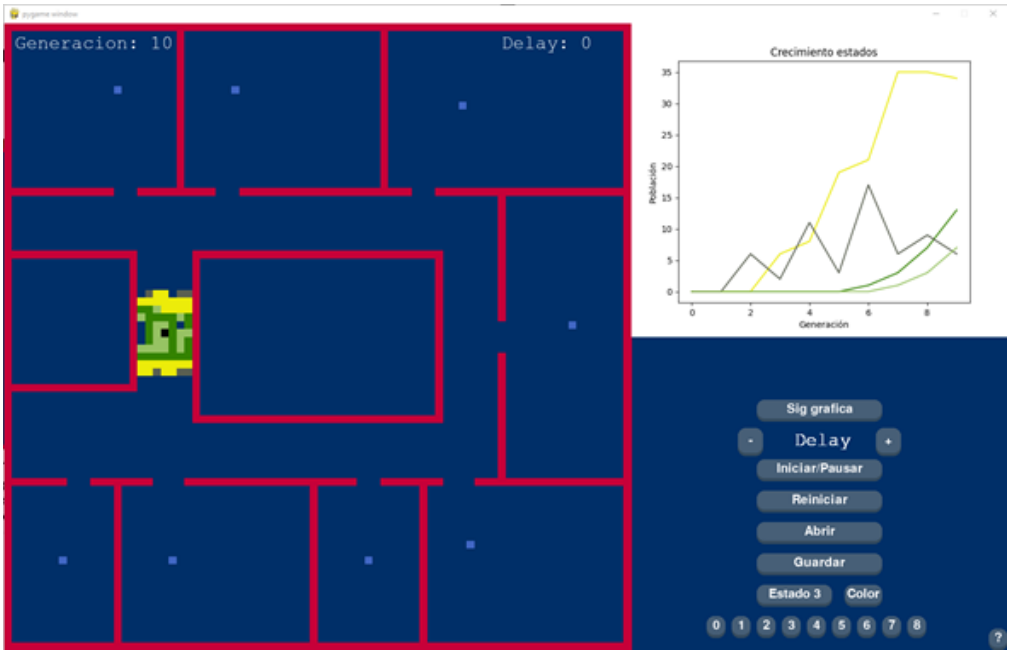


Figura 69: Prueba del simulador.

En la Figura 69 se muestra una configuración con 8 nutrientes y la simulación avanzado 10 generaciones, por parte de la grafica se muestra los crecimientos de estados.

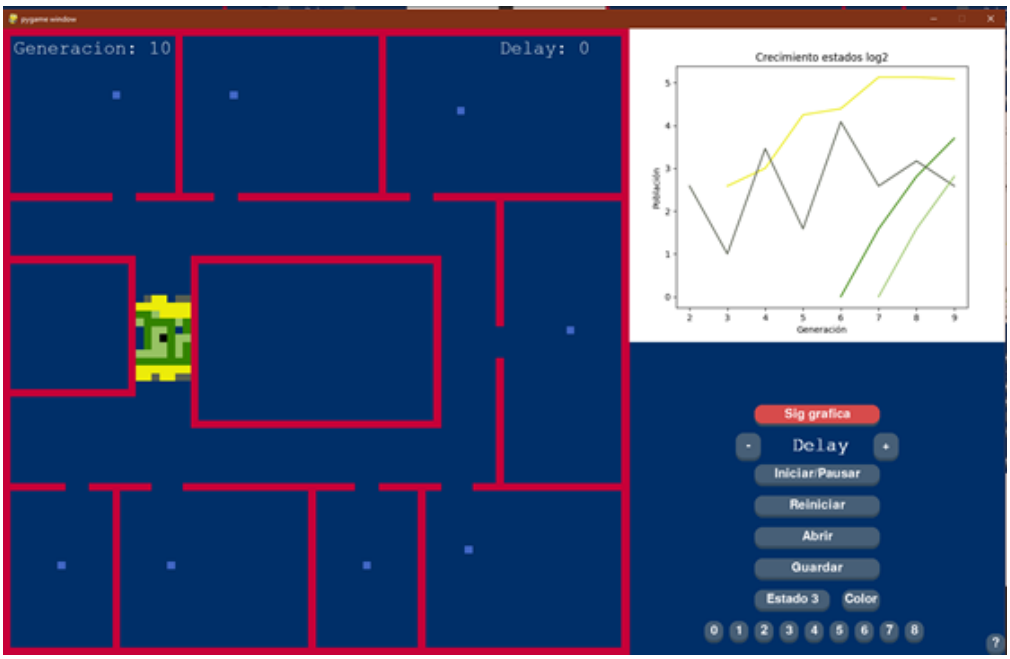


Figura 70: Se oprime el botón de siguiente gráfica.

En la Figura 70 se oprime el botón de Sig grafica, se cambia la grafica mostrada a la de crecimiento de estados con log 2, para poder ver de una mejor manera la información.

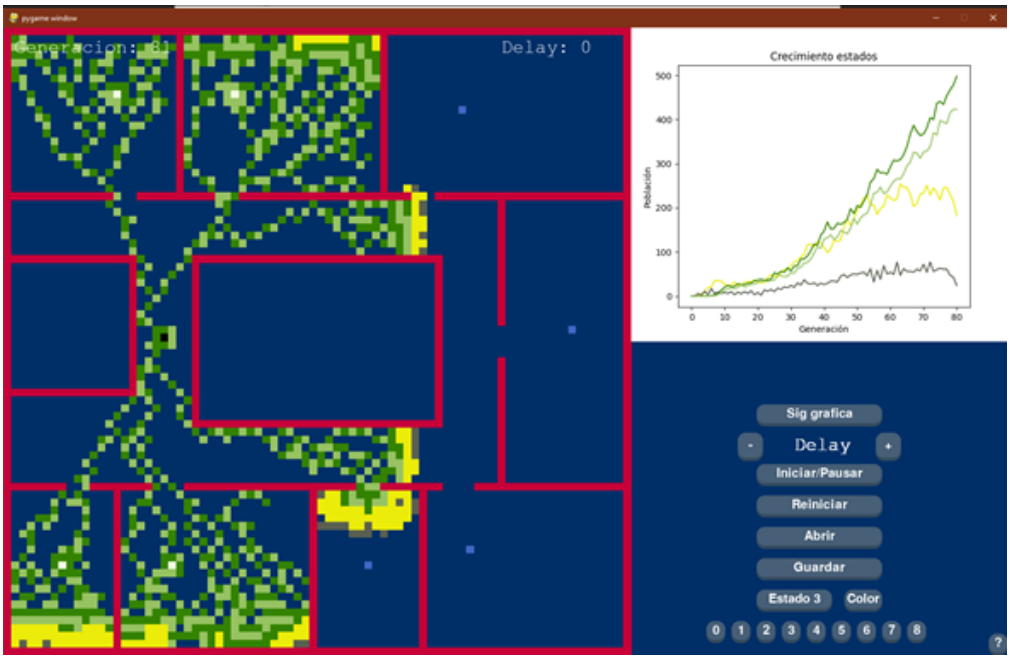


Figura 71: T=81.

En la Figura 71 nuestra simulación se encuentra en la generación 81, y esta continúa explorando el espacio para encontrar los nutrientes faltantes.

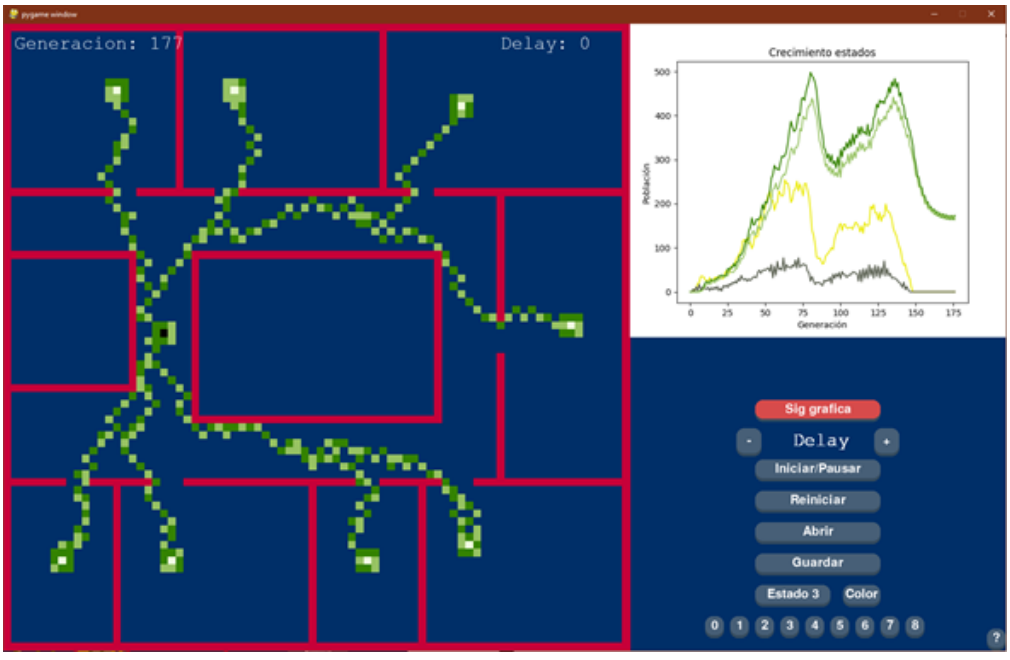


Figura 72: Rutas estables del simulador.

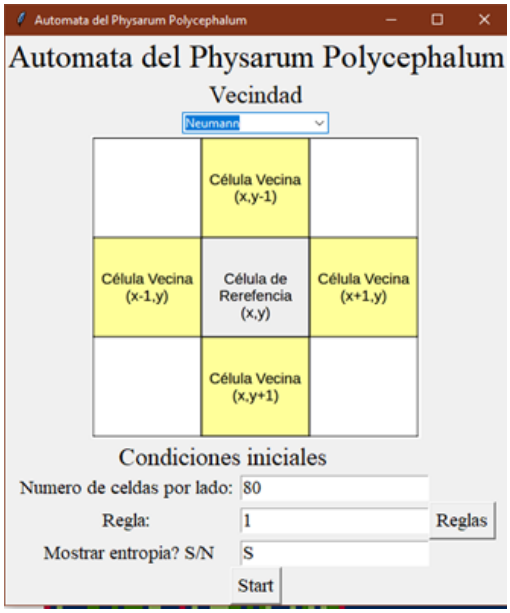


Figura 73: Seleccionamos la vecindad de Neumann.

Ahora probemos con la vecindad de Neumann donde las demás características se encuentran igual, como se muestra en la figura 73.

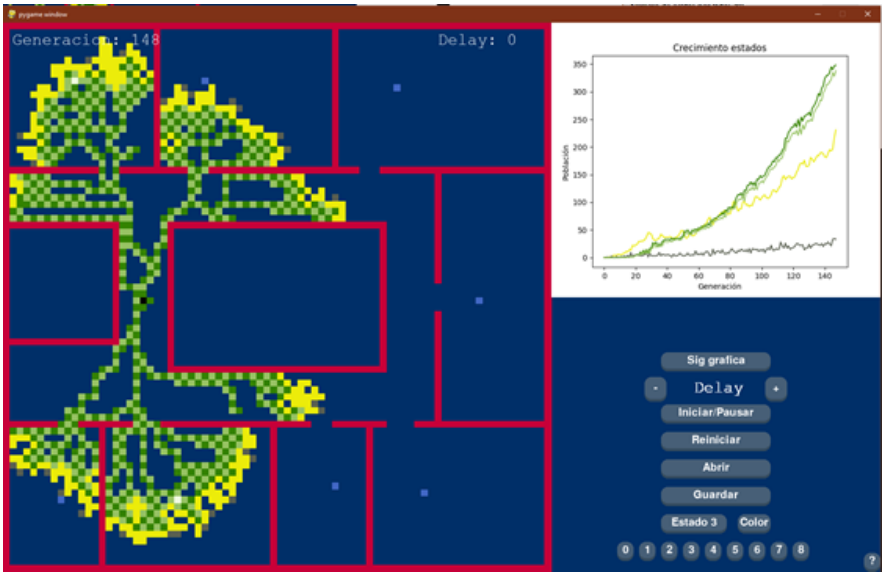


Figura 74: Primeros encuentros con los nutrientes.

En la Figura 74 se muestra cómo el simulador encuentra sus primeros nutrientes en la generación 148.

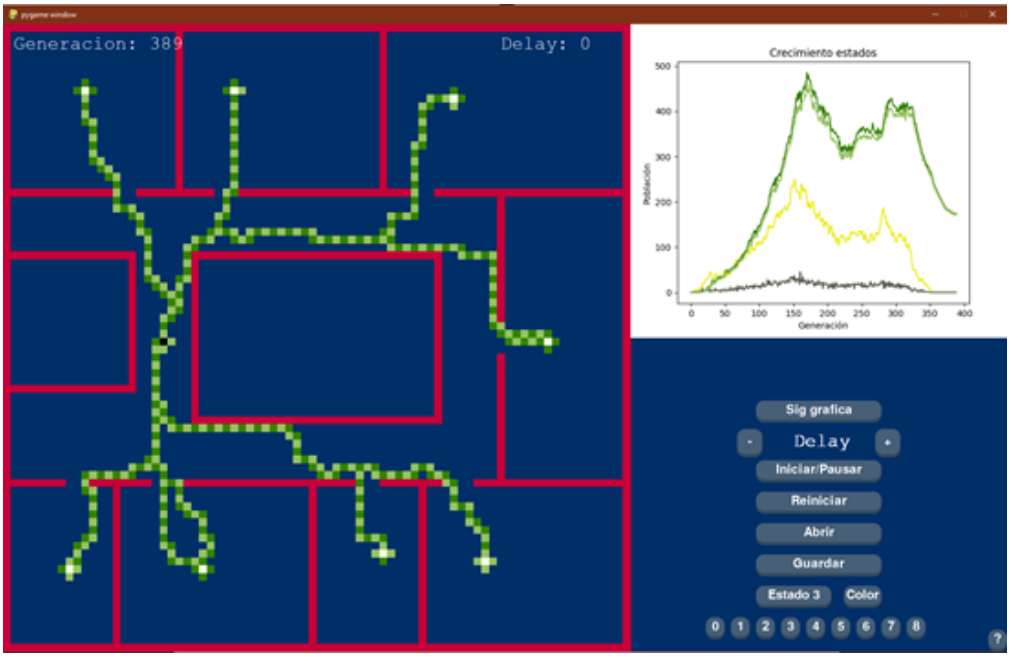


Figura 75: Rutas encontradas por el modelo.

En la Figura 75 se muestra la simulación en un estado estable hasta la generación 389 mostrando que la simulación es significativamente mas lenta con la vecindad de Neumann contra la vecindad de Moore.

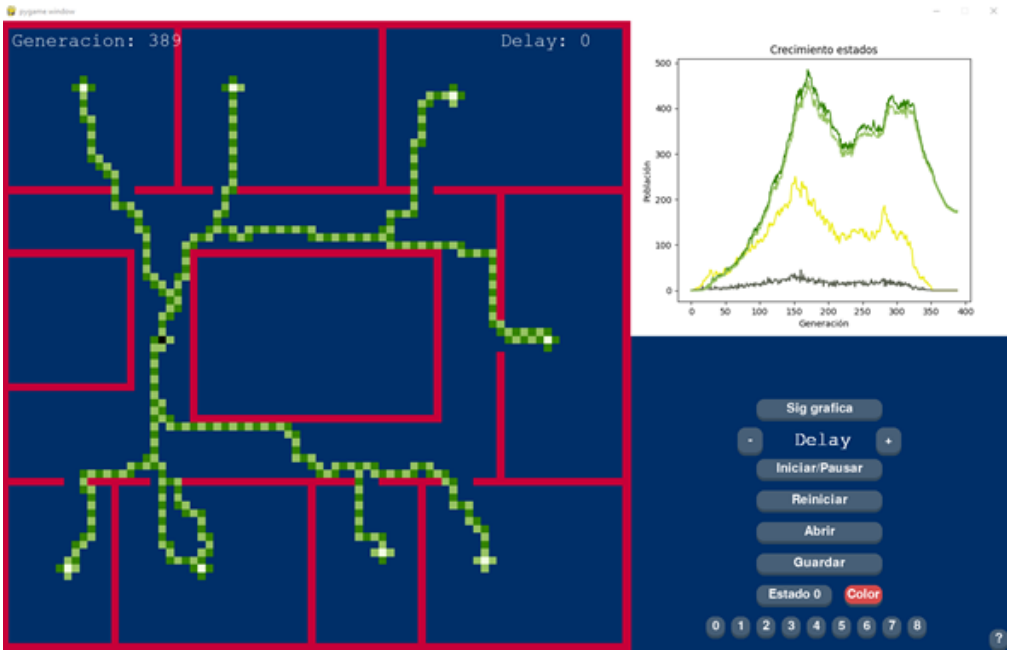


Figura 76: Se presiona el botón de Color.

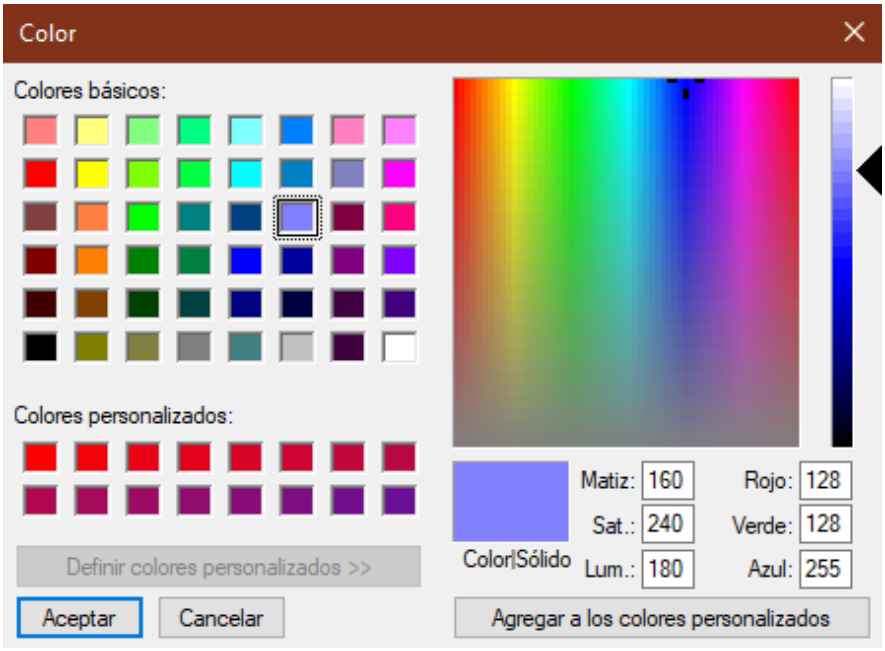


Figura 77: Selector de colores.

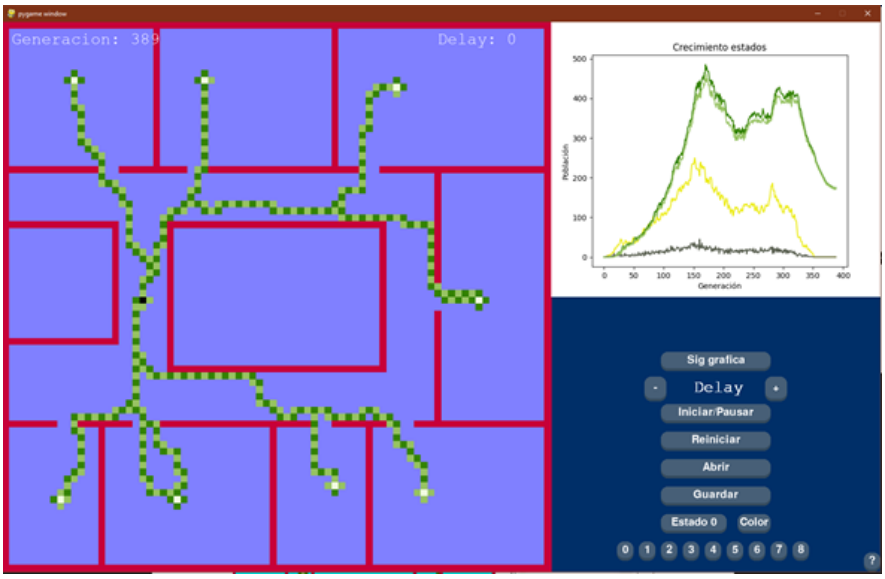


Figura 78: Cambio de color de fondo.

Si presionamos al botón de color, se nos abrirá un menú como se muestra en la Figura 77, el cual nos permitirá cambiar el color de nuestro estado, si se llegase a cambiar el color de los estados 4,5,7 u 8; estos cambios también se verían reflejados en las graficas de crecimiento como se muestra en la Figura 81.

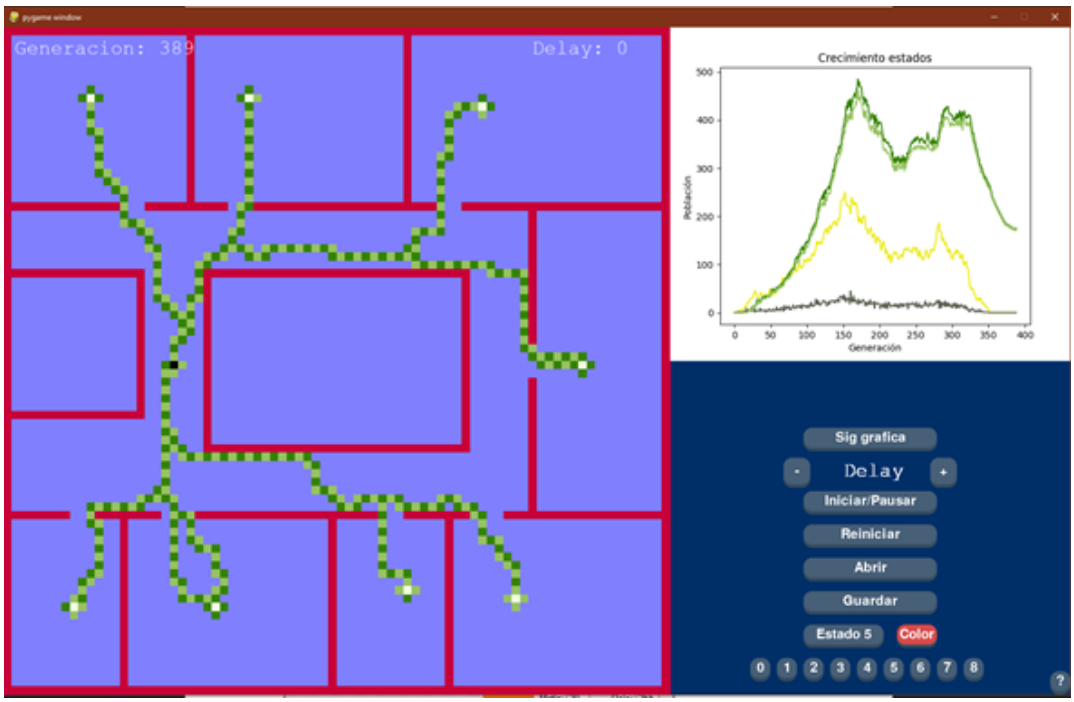


Figura 79: Se presiona de nuevo el botón de Color.

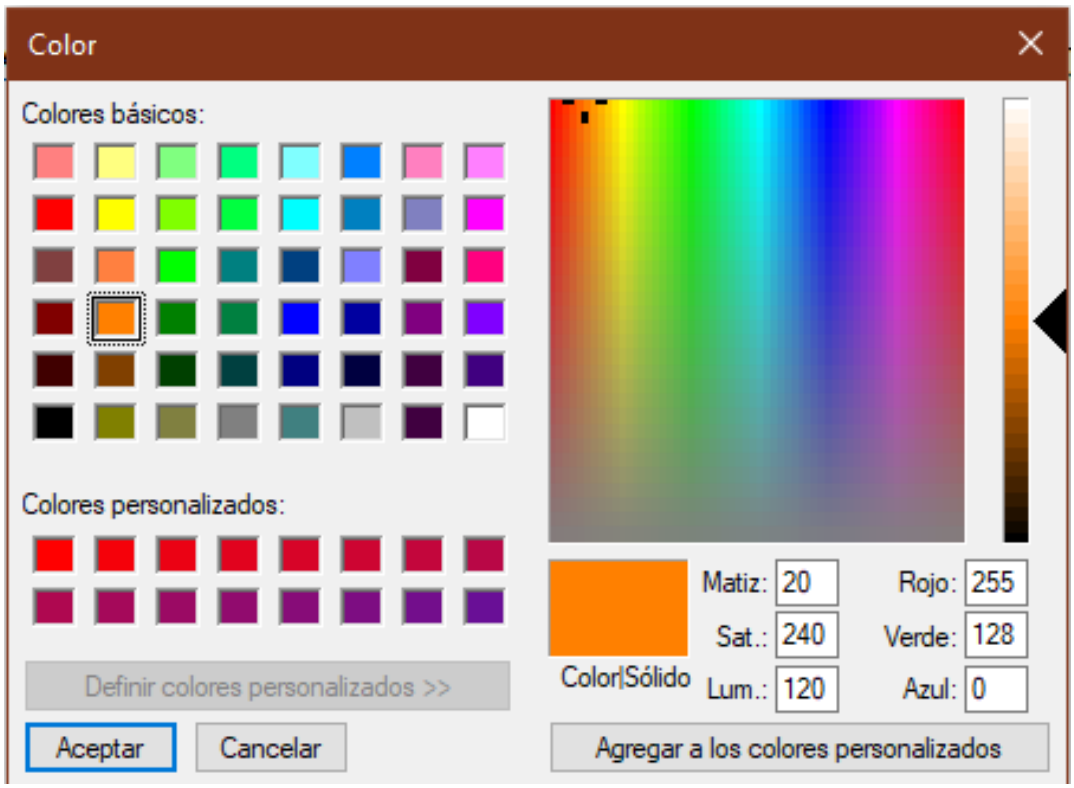


Figura 80: Seleccionamos un color.

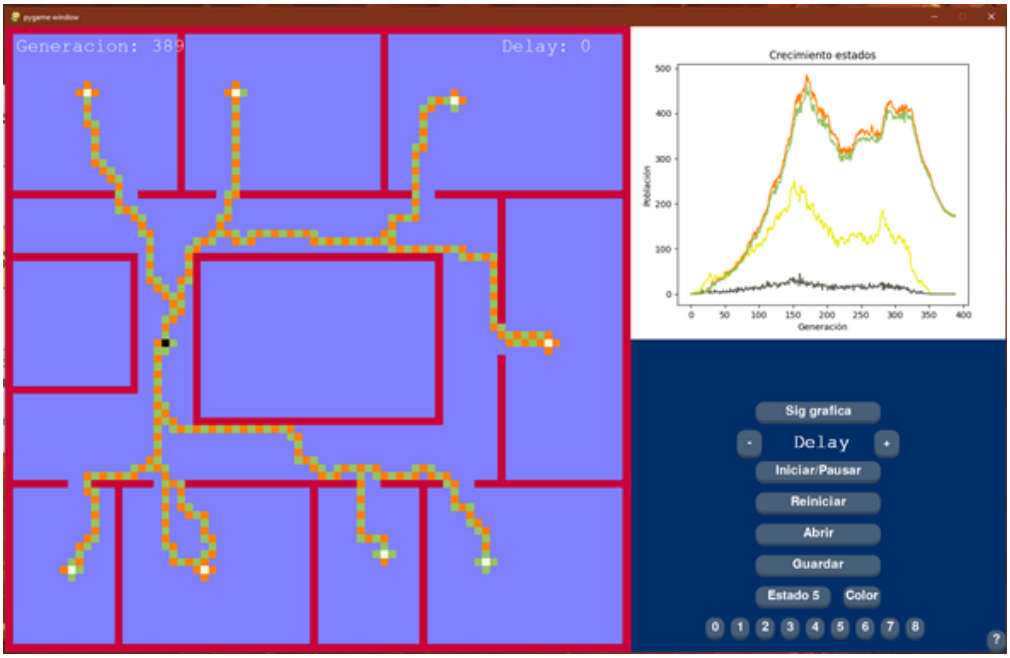


Figura 81: Se reflejan los cambios en la interfaz.

Si se presiona el botón con el símbolo ?, se mostrará el mensaje de la Figura 81, el cual nos da una breve descripción de cuáles son los estados que podemos manipular.

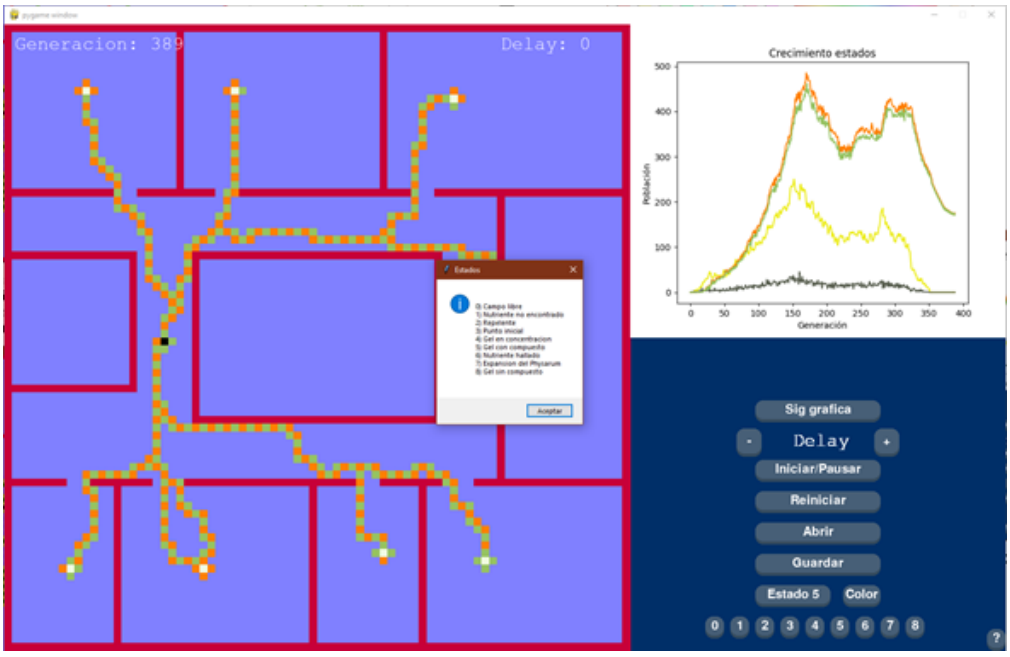


Figura 82: Ventana de información.

6.1.7. Versión final del prototipo 2DPyCAM(10)

En la Figura 83 se puede apreciar unos cambios en la interfaz gráfica cuando se corre el programa debido que se agregaron ComboBox para seleccionar la regla a utilizar y si el usuario desea mostrar la entropía.

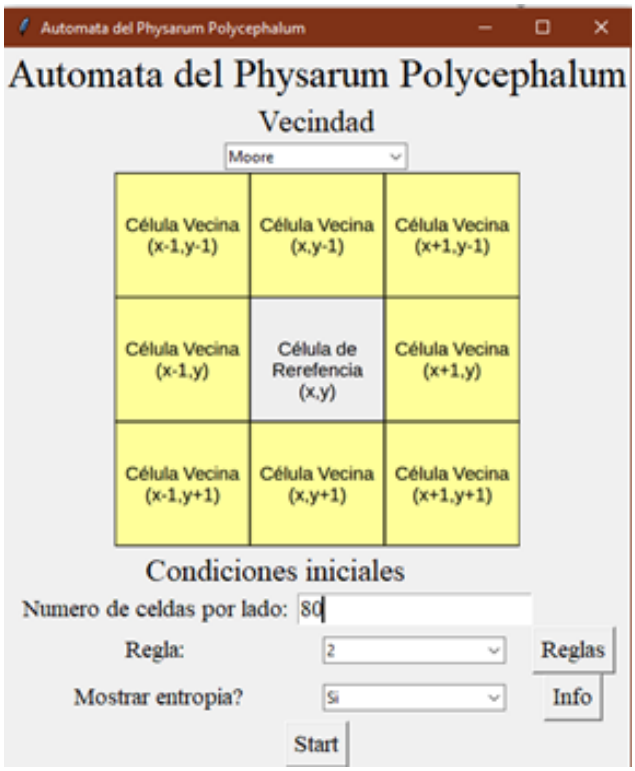


Figura 83: Menú inicial con pequeños cambios.

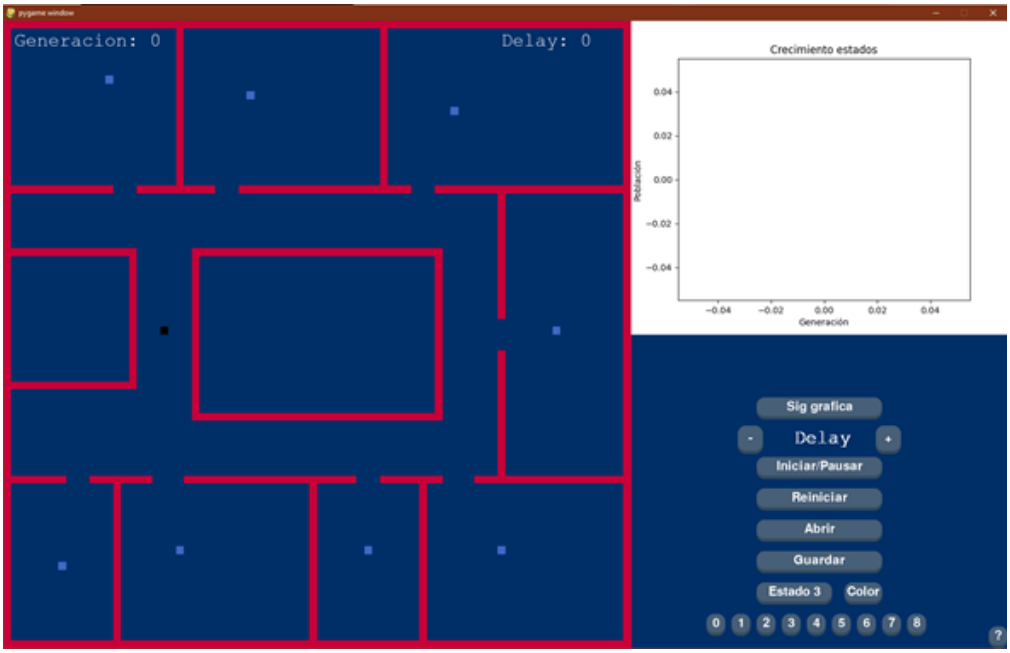


Figura 84: Cargado del mapa.

.

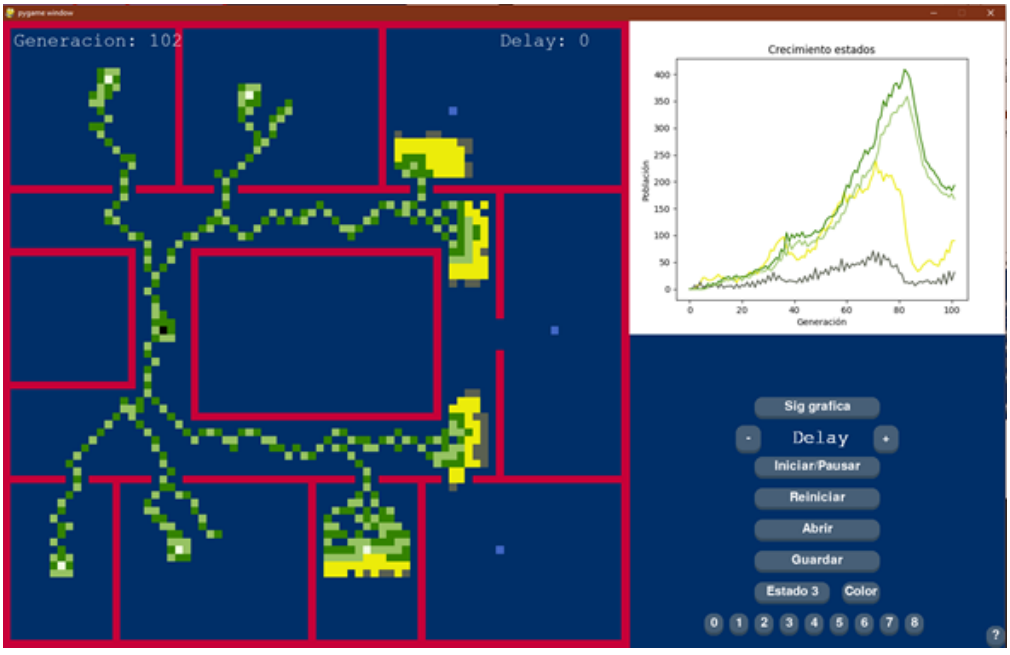


Figura 85: Generación 102.

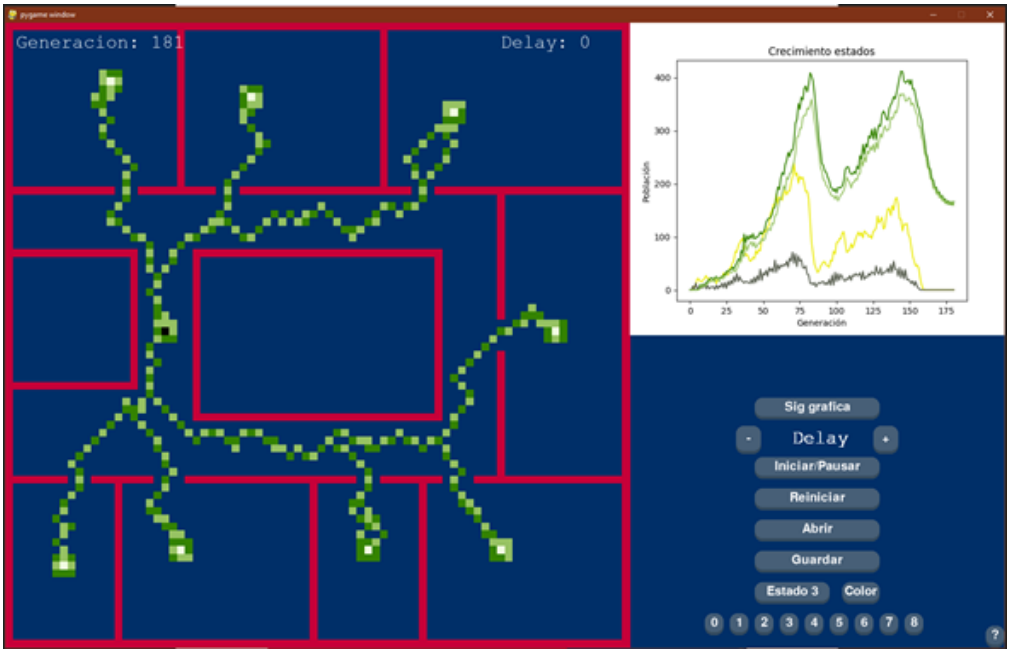


Figura 86: Rutas establecidas por el modelo.

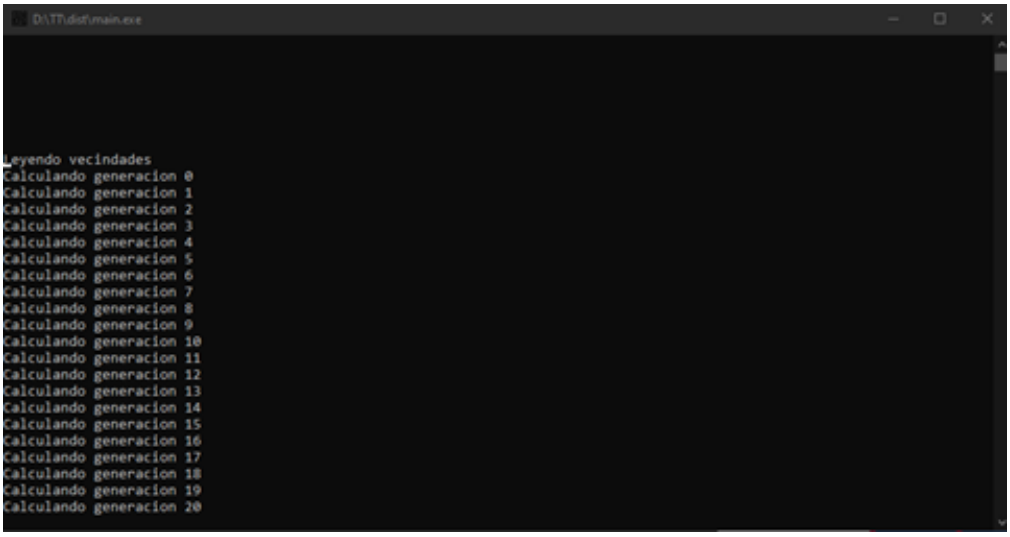


Figura 87: Registro de acciones.

Como el usuario seleccionó mostrar la entropía, al finalizar la simulación se calculará esta para no afectar el funcionamiento de la simulación, debido a que los cálculos realizados alentarían mucho su funcionamiento.

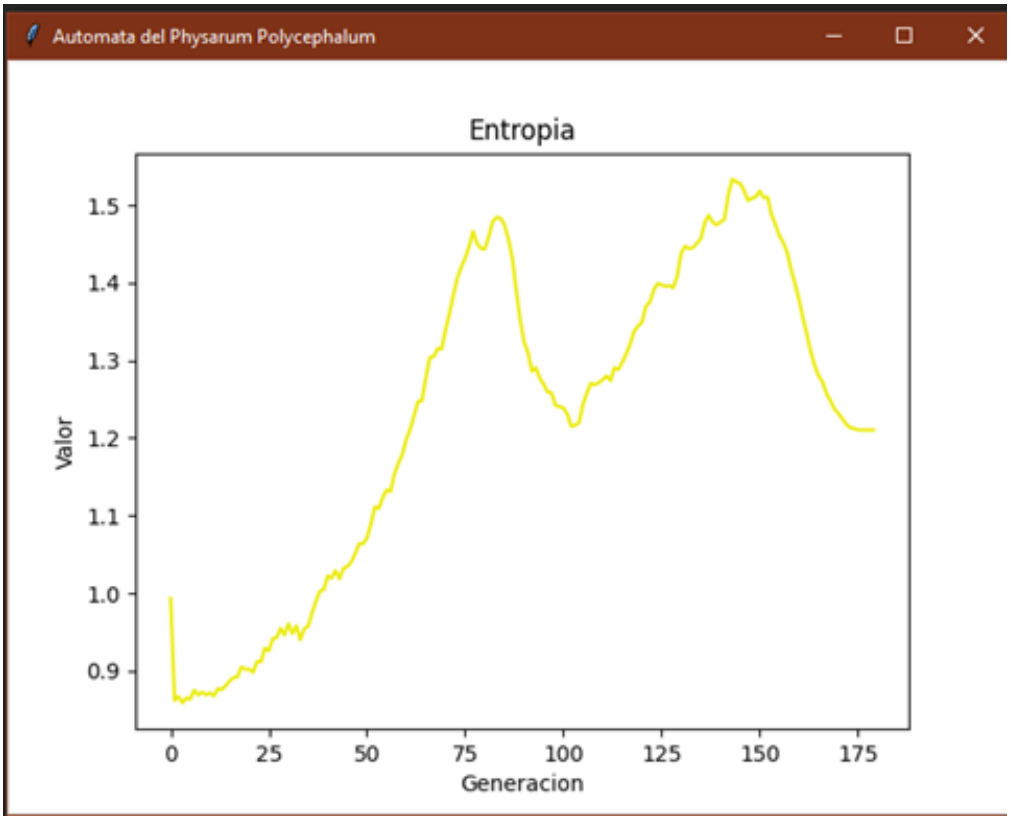


Figura 88: Gráfica de Entropía.

En la Figura 88 se muestra la entropía de la simulación de la Figura 86.

7. Interfaz usuario - robot (Servicio Web)

7.1. Servicio Web

Ahora veremos el desarrollo de la interfaz de usuario - robot, esta sección la dividiremos en dos grandes secciones la primera será el desarrollo de la interfaz de los módulos planteados en la sección de Análisis de Diseño, mencionar que los módulos correspondientes a la interacción con el robot son meramente de prueba, por lo que, la segunda parte será ya la incorporación con el robot en sí que sería prácticamente la ultima parte de este trabajo.

Recordar que esta interfaz usuario - robot partir de este momento llamaremos como servicio web para mayor facilidad de lectura y comprensión, ademas de que, nuestra interfaz como ya vimos anteriormente esta desarrollada con tecnologías orientadas a la web.

7.1.1. Primer prototipo interfaz usuario - robot (Servicio Web)

En este primer prototipo se creo un sistema web cuya intención es montarlo en la misma Raspberry pi 4 para que funcionara como un servidor móvil, ademas de que seguiremos toda la sección de análisis y diseño para crear esta primera versión.

Para poder ver como es el nuestro servicio web pondremos unas capturas de pantalla junto con un texto explicativo si es necesario, el orden de estas capturas y de esta sección se seguirá el orden de los casos de uso mostrados en la sección de Análisis de Diseño.

7.1.1.1. Página principal

Para este servicio web se tiene una página principal o también conocida como página inicial, esta página contiene las opciones de Inicio que hace referencia a la mima pagina inicial, Registro que nos da la opción de realizar un nuevo registro para usar el servicio

web como usuario básico a esperar la validación por parte de un administrador y de la opción de Iniciar sesión que es la opción para poder acceder al sistema.

Ademas de los componentes anteriormente mencionados tiene un titulo de Envió de paquetes y un subtítulo que nos comenta el fin de este servicio web, también contiene un GIF de muestra delo funcionamiento del algoritmo desarrollado siendo usado en el mapeo del primer del edificio de gobierno de la ESCOM con un pie de imagen que nos comenta que podemos hacer desde esta página.

Finalmente, tenemos un pie de página que será compartido a lo largo del servicio web en todas la paginas el cual contiene el título, así como el numero del Trabajo Terminal, escuela en donde se realizo y finalmente nuestros nombres, es decir, los autores del TT.

En las siguientes dos figuras podemos ver la página principal, mencionar que la segunda figura solo se agrego debido a que queríamos que se pudiera ver el pie de página, pero a partir de este punto la mayor parte de las capturas no lo tendrá ya que no es un elemento imprescindible ademas de que en todos los casos tiene el mismo contenido.

Figura 89: Pagina principal. Parte 1.

Figura 90: Pagina principal. Parte 2.

7.1.1.2. Iniciar sesión

Ahora en la página de iniciar sesión, nos pide nuestro correo y contraseña para poder acceder al servicio web, ahora bien, también vemos la opción de Recuperar contraseña, pero esto lo veremos mas adelante, ahora llenamos los campos con una cuenta de administrador pre creada como lo vemos en la figura 92 y le damos clic al botón Ingresar, como vemos en la figura 93 nos pide confirmar o cancelar la acción si queremos acceder con ese correo, si damos cancelar simplemente cierra la ventana emergente y no accedemos al servicio web, por otro lado si confirmamos la acción y nuestra contraseña, así como el usuario es correcto entonces nos dará un mensaje de Bienvenido administrador o simplemente Bienvenid@ en caso de un usuario básico, como lo vemos en la figura 94 y 95 respectivamente.

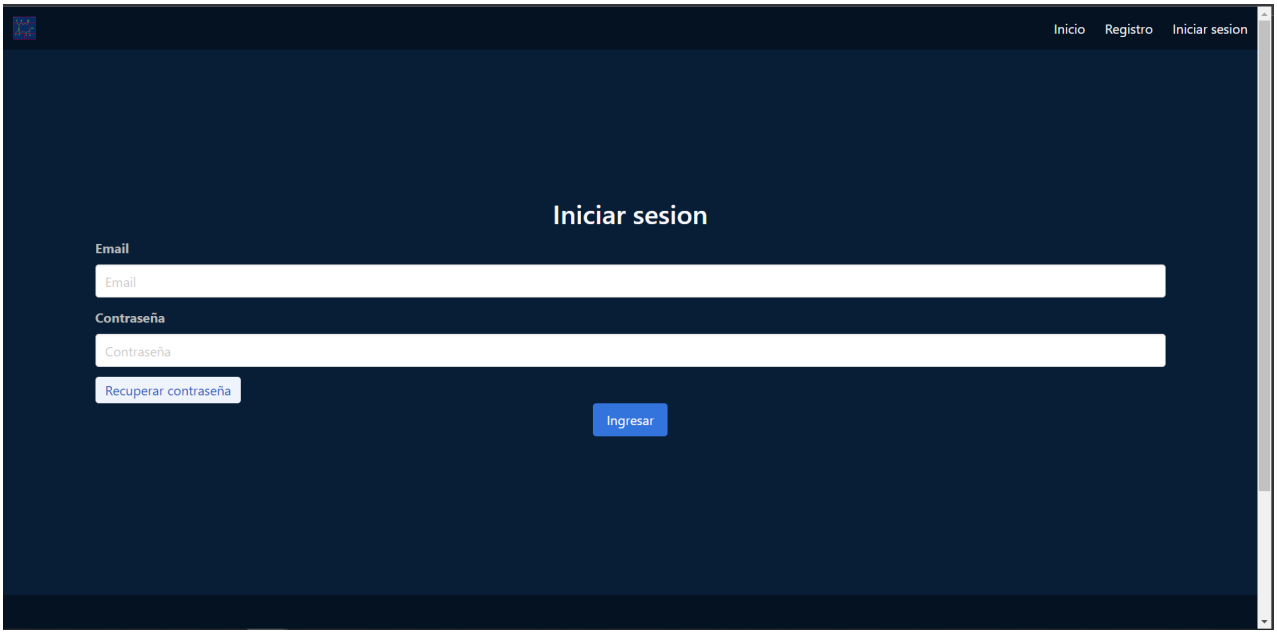


Figura 91: Formulario inicio sesión.

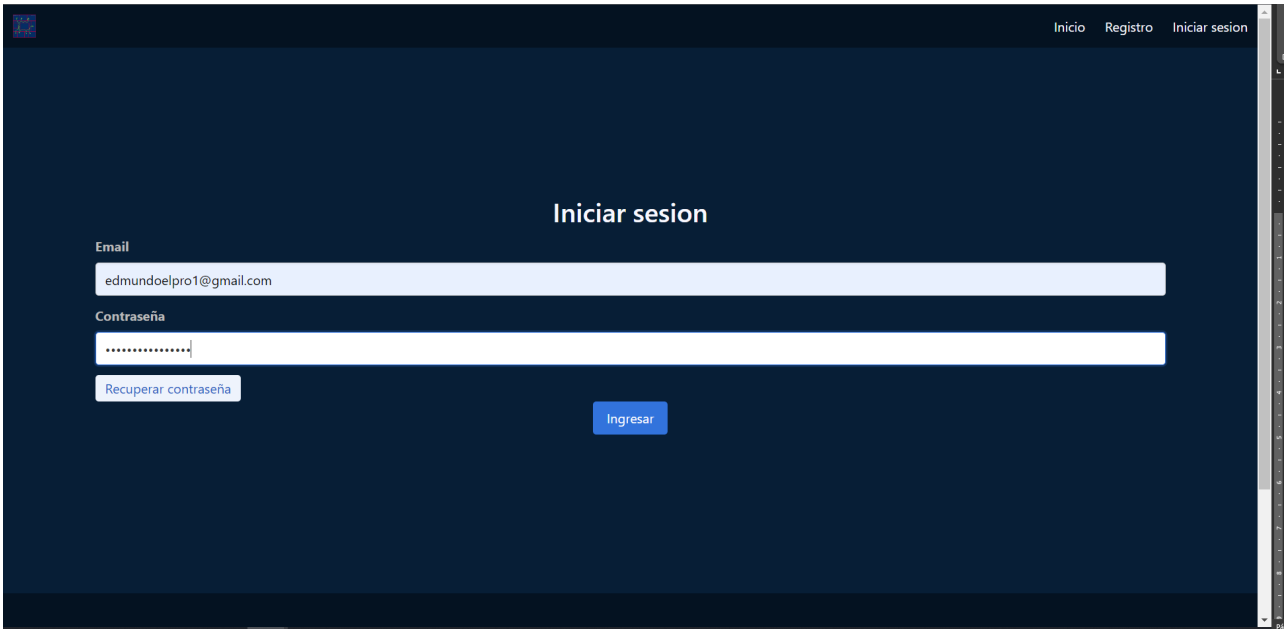


Figura 92: Formulario inicio sesión llenado con datos de prueba.

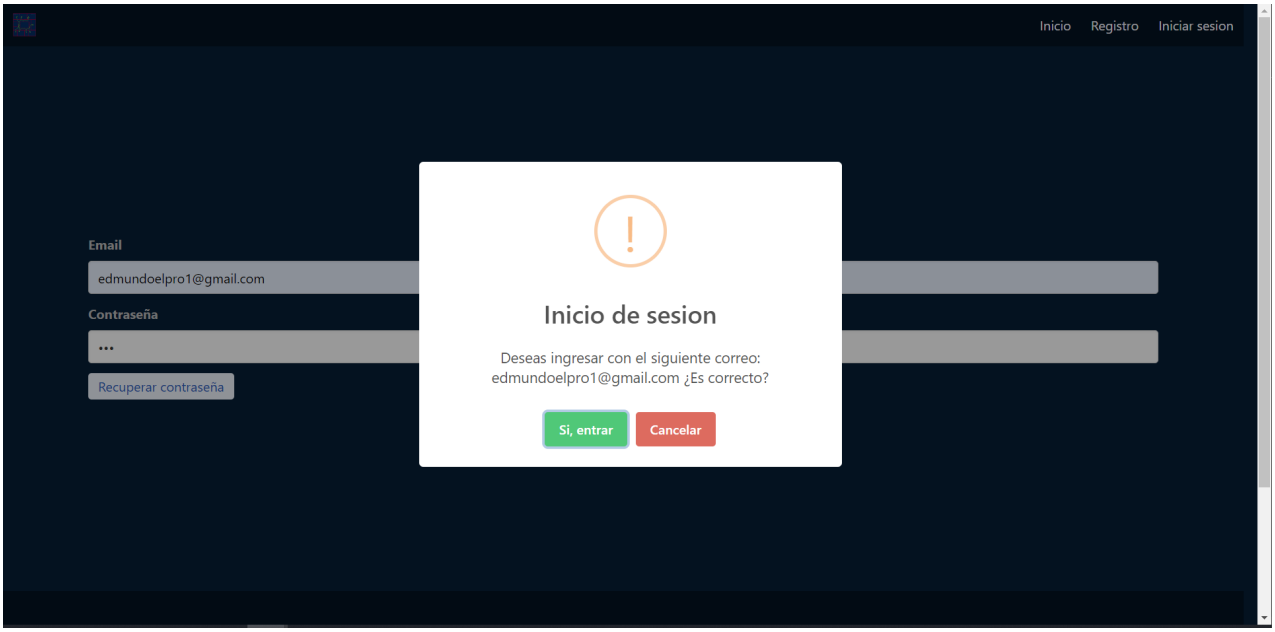


Figura 93: Mensaje de confirmación.

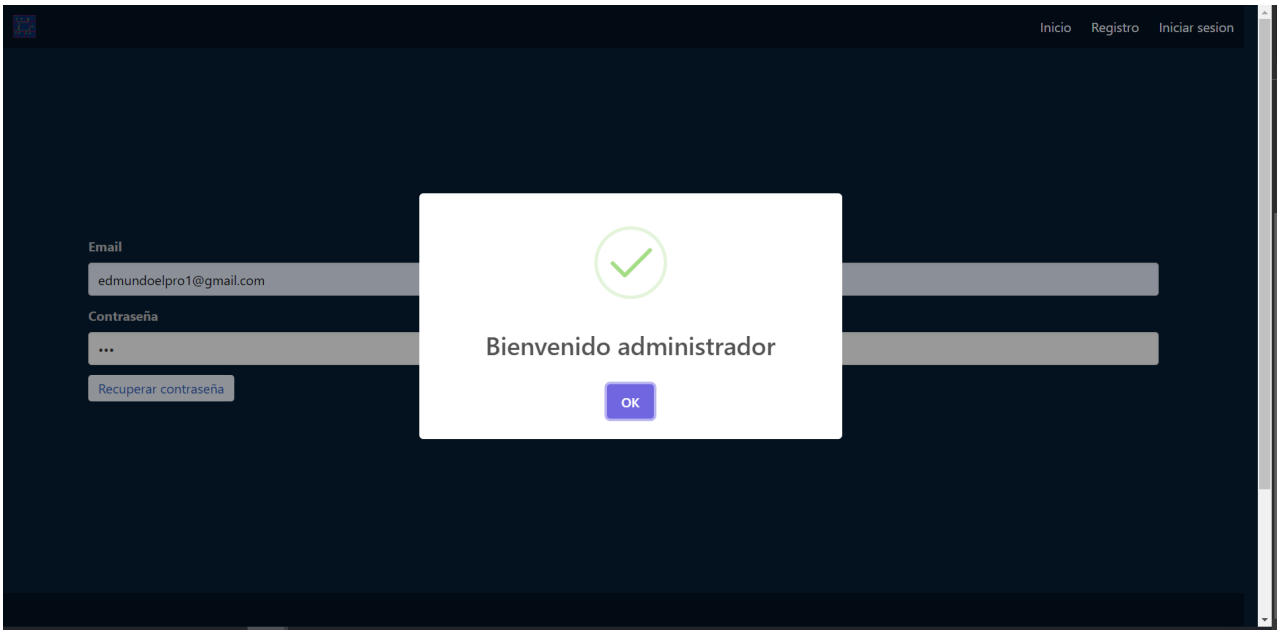


Figura 94: Mensaje antes de ingresar cuando las claves de administrador son correctas.

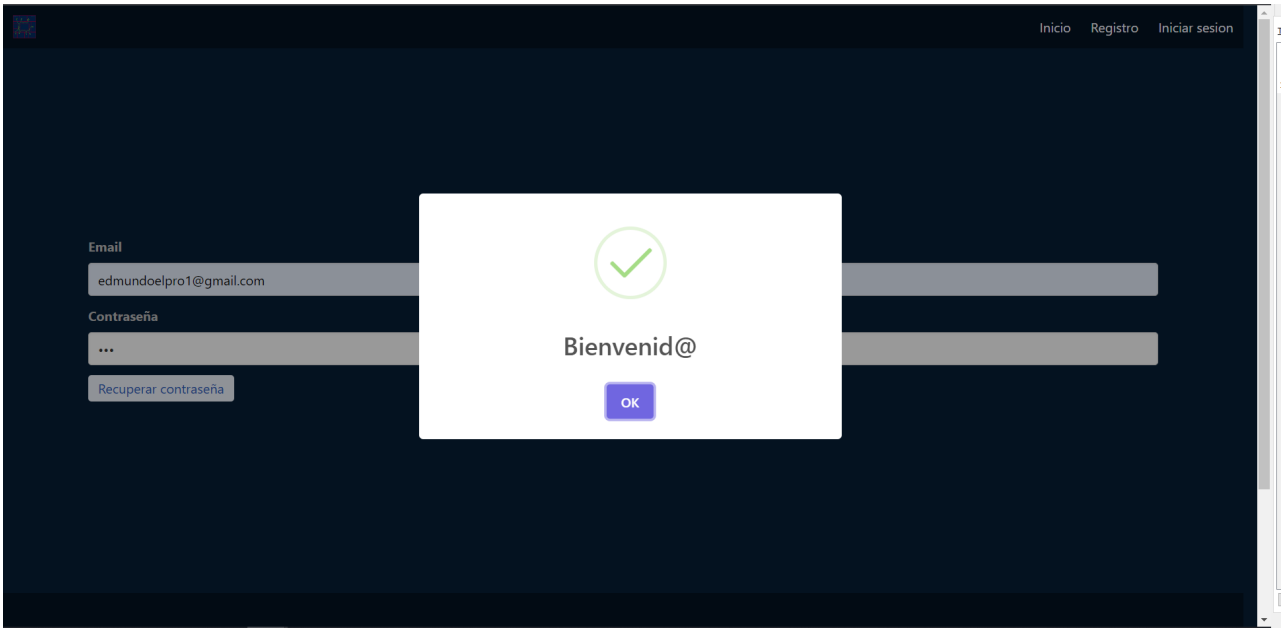


Figura 95: Mensaje antes de ingresar cuando las claves de usuario básico son correctas.

En caso de acceder con un correo no registrado en que la contraseña sea errónea se mostraran los siguientes mensajes.

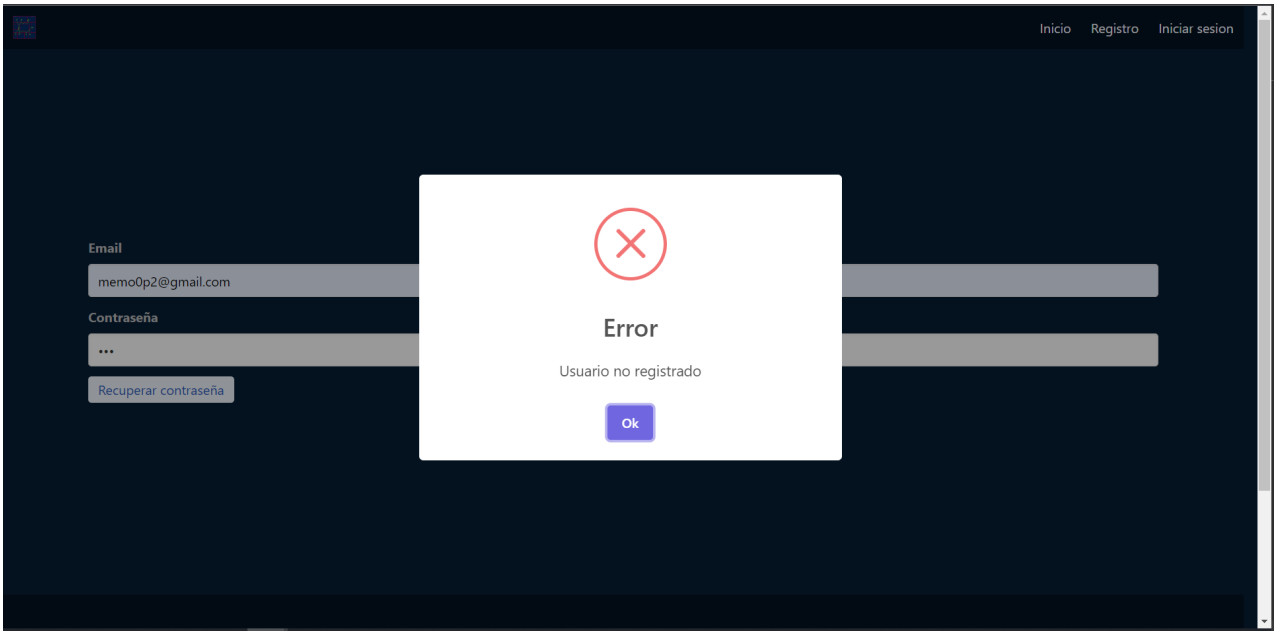


Figura 96: Mensaje de error cuando un correo no esta registrado.

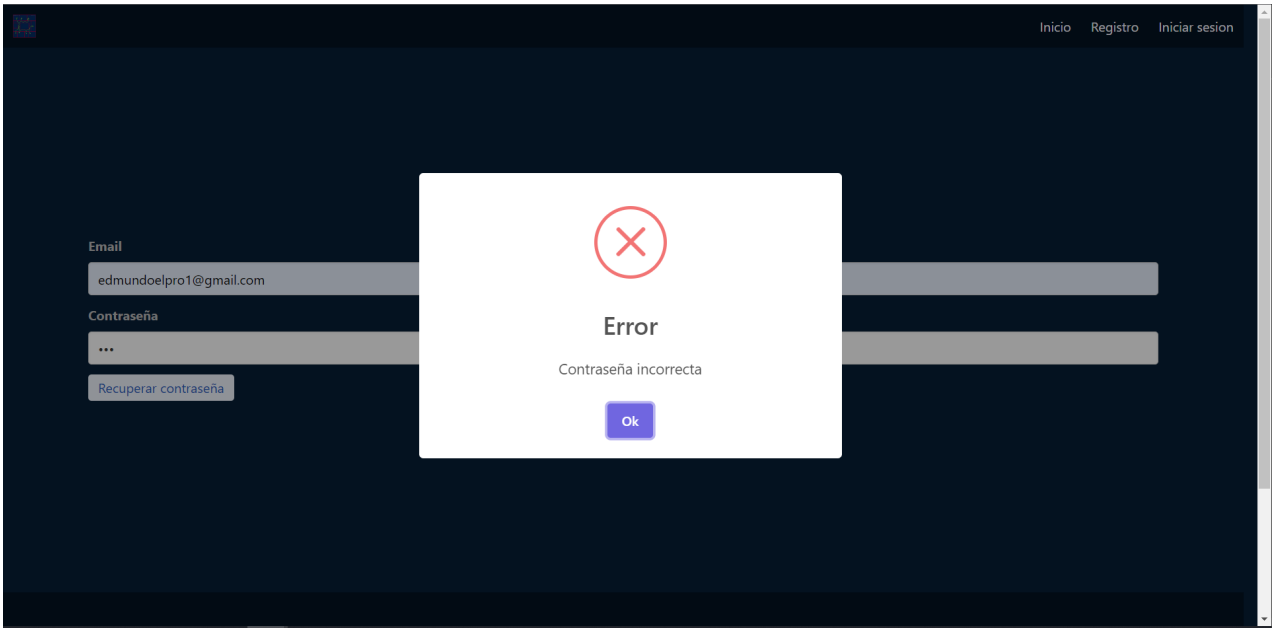


Figura 97: Mensaje de error cuando la contraseña es incorrecta.

7.1.1.3. Registro

En el caso de la sección de registro, tenemos un formulario con varios campos que tendrán que ser llenados debidamente para crear una nueva contraseña a nivel de usuario básico y este tendrá que esperar a que sea validad, así, tenemos un cierto control en los usuarios que quieran registrarse en nuestro sistema. Finalmente vemos en la figura 99 como llenamos el formulario con datos reales, mencionar que el formulario no nos dejara continuar con el envió del formulario, si es que las contraseñas no coinciden, así como si no cumple con las características planteadas en secciones anteriores. Finalmente, en la figura 100 podemos ver como el nuevo usuario fue registrado de manera correcta.

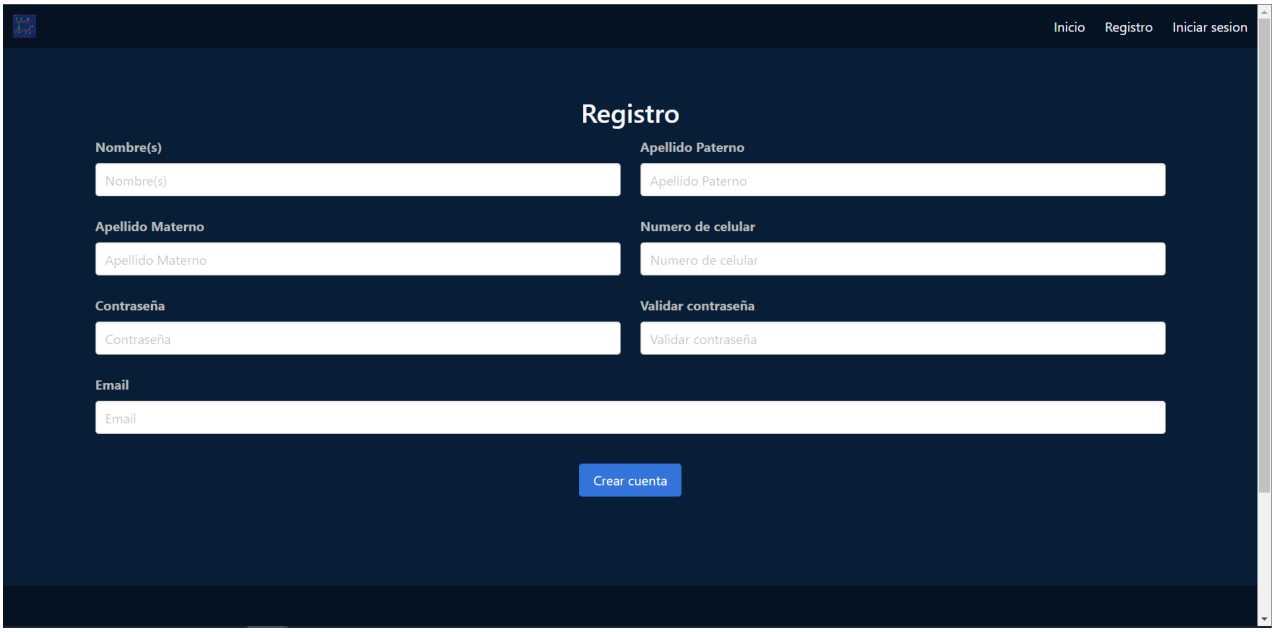


Figura 98: Formulario registro.

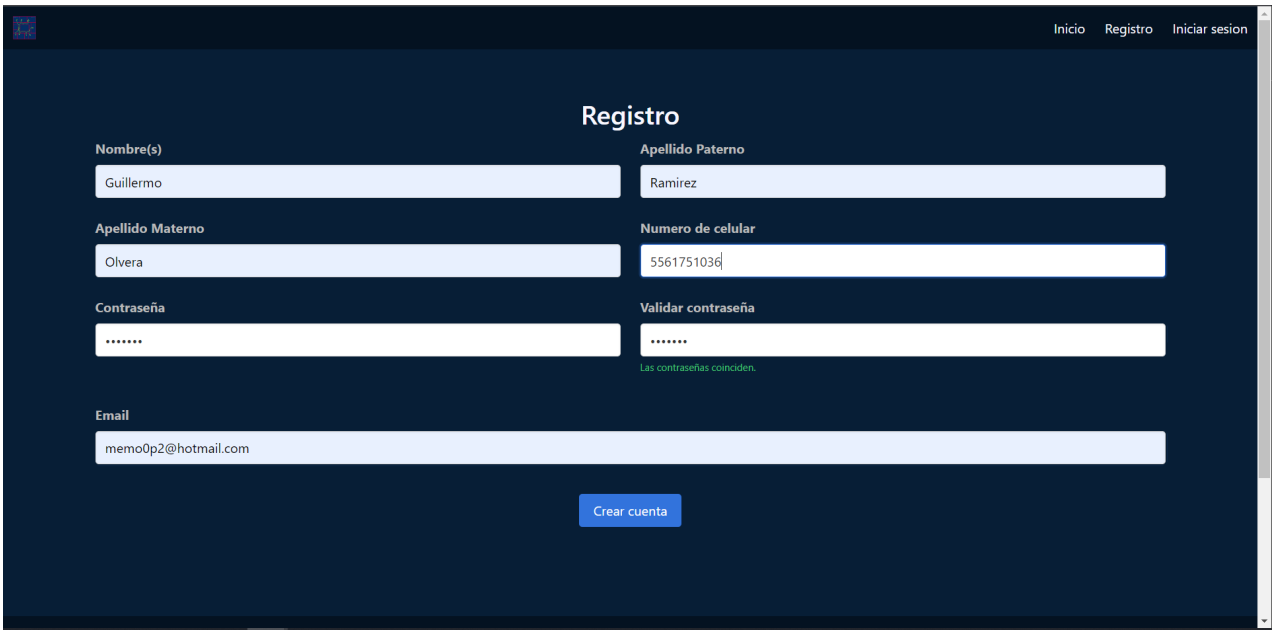


Figura 99: Formulario registro llenado.

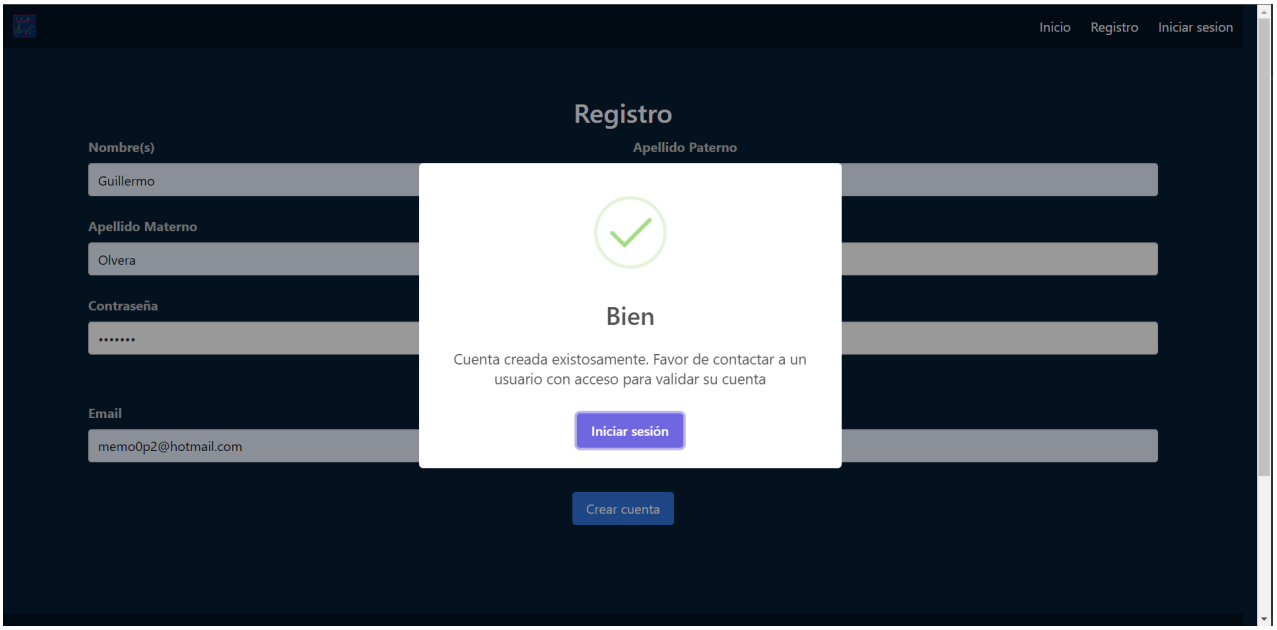


Figura 100: Registro realizado con éxito.

En caso de que ya haya un registro con ese correo el servicio web mandara un mensaje que nos dice que ya existe un usuario con ese correo.

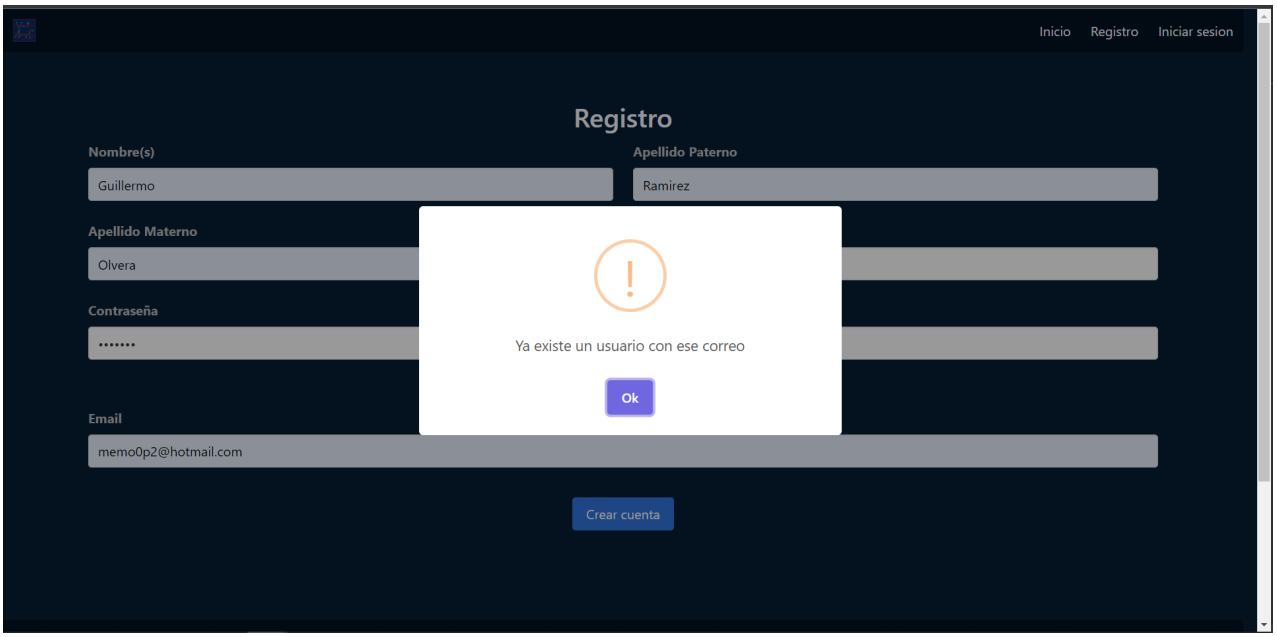


Figura 101: Mensaje de que ya existe un usuario con ese correo.

7.1.1.4. Enviar o recibir paquetes(Cambiar ubicación actual del robot)

Antes de iniciar debemos comentar que todo este apartado y el siguiente apartado que está relacionado con el robot para esta versión será simulado. Ahora vayamos a lo más importante de nuestro servicio web, lo cual es el enviar o recibir paquetes, en este caso nos enfocaremos en el cambio de ubicación del robot, esto solo es aplicable si el robot será usado en un lugar diferente de donde está ubicado el robot, como vemos en la figura 102 para esta parte solo usaremos la primera sección, en este caso moveremos el robot a Capital Humano, para esto debemos seleccionar la ubicación deseada como vemos en la figura 103 donde nos muestra todas las ubicaciones posibles a excepción de la ubicación actual por obviedad, seleccionamos una ubicación como en la figura 104, ahora le damos clic al botón Mover robot a la nueva ubicación inicial y ahora tendremos dos casos, en la figura 105 podemos ver como se ve esta pagina desde el punto de vista del usuario que realizo el movimiento y en la figura 106 podemos ver como lo ve un usuario quien no fue el que efectuó el movimiento, mencionar que en el caso de que el robot este en movimiento no se podrá hacer uso de esta parte del servicio web.

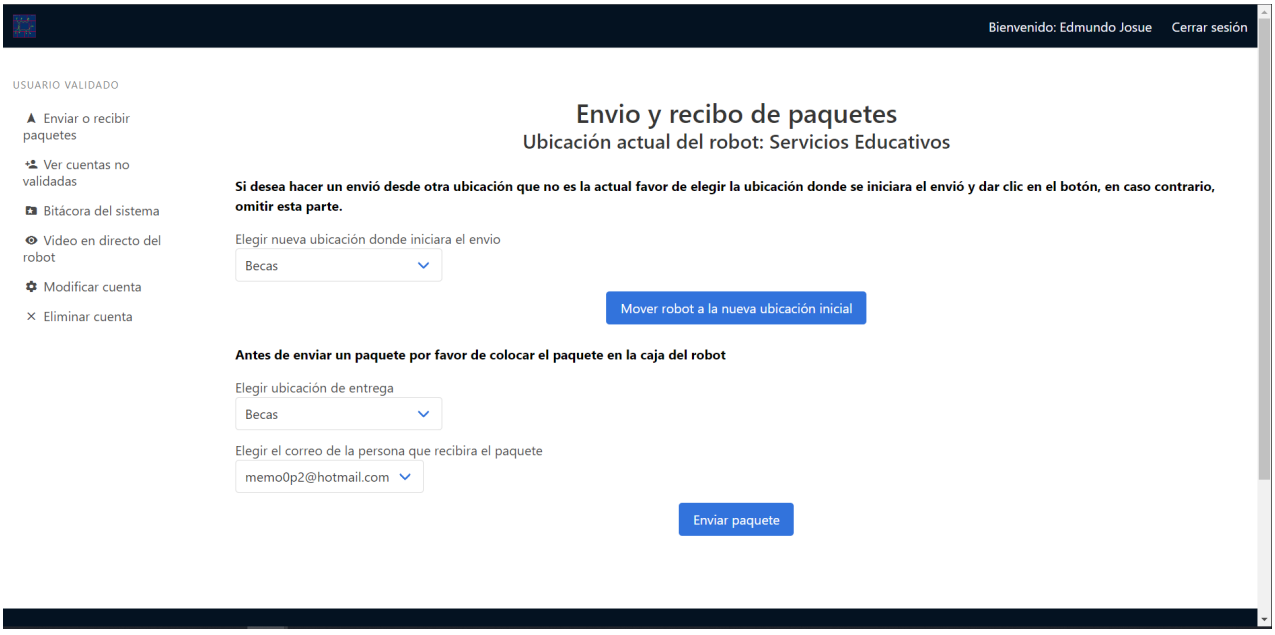


Figura 102: Opción de envió y recibo de paquetes.

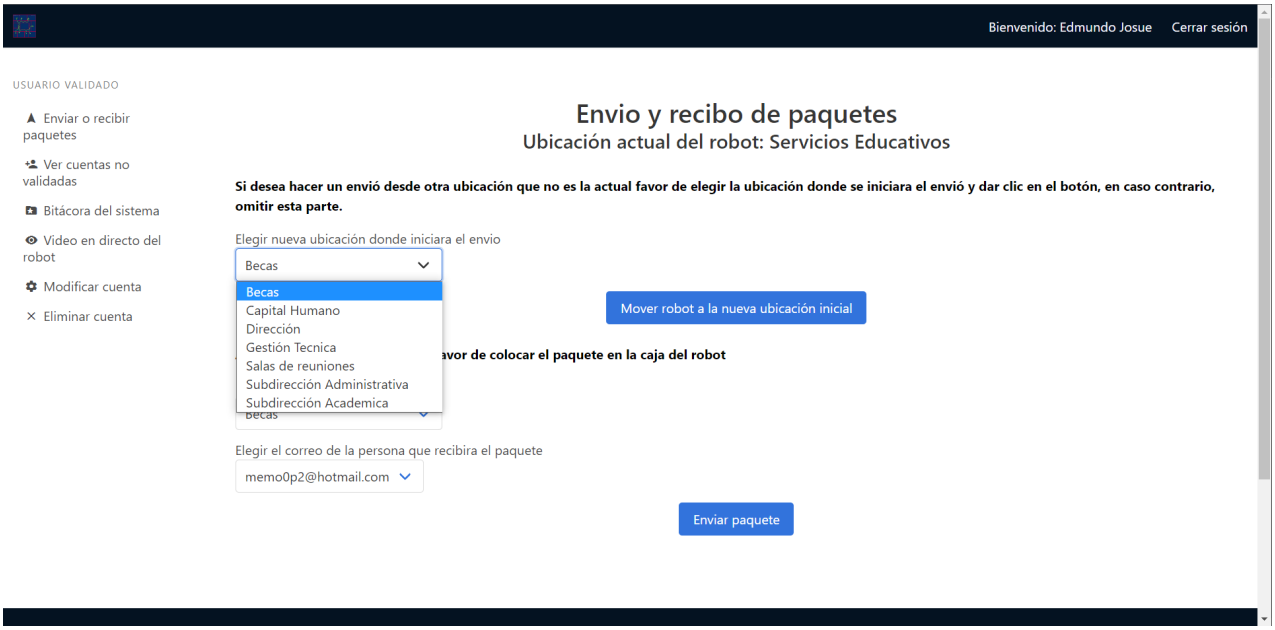


Figura 103: Muestra de ubicaciones disponibles.

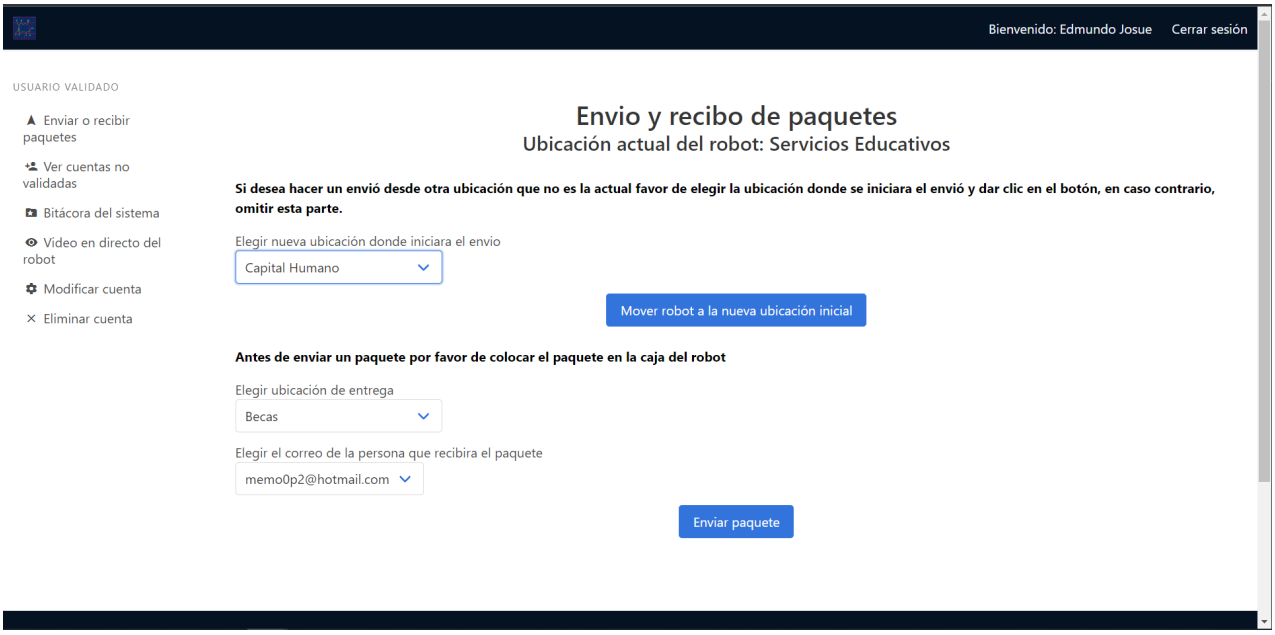


Figura 104: Seleccionado nueva ubicación.

Figura 105: Robot en movimiento. Punto de vista del usuario que realizo el movimiento.

Figura 106: Robot en movimiento. Punto de vista del usuario que no realizo el movimiento.

Finalmente podemos ver en la figura 107 como se nos avisa de que el robot ya llego al destino seleccionado y en la figura 108 podemos ver como el otro usuario puede ya hacer uso del robot.

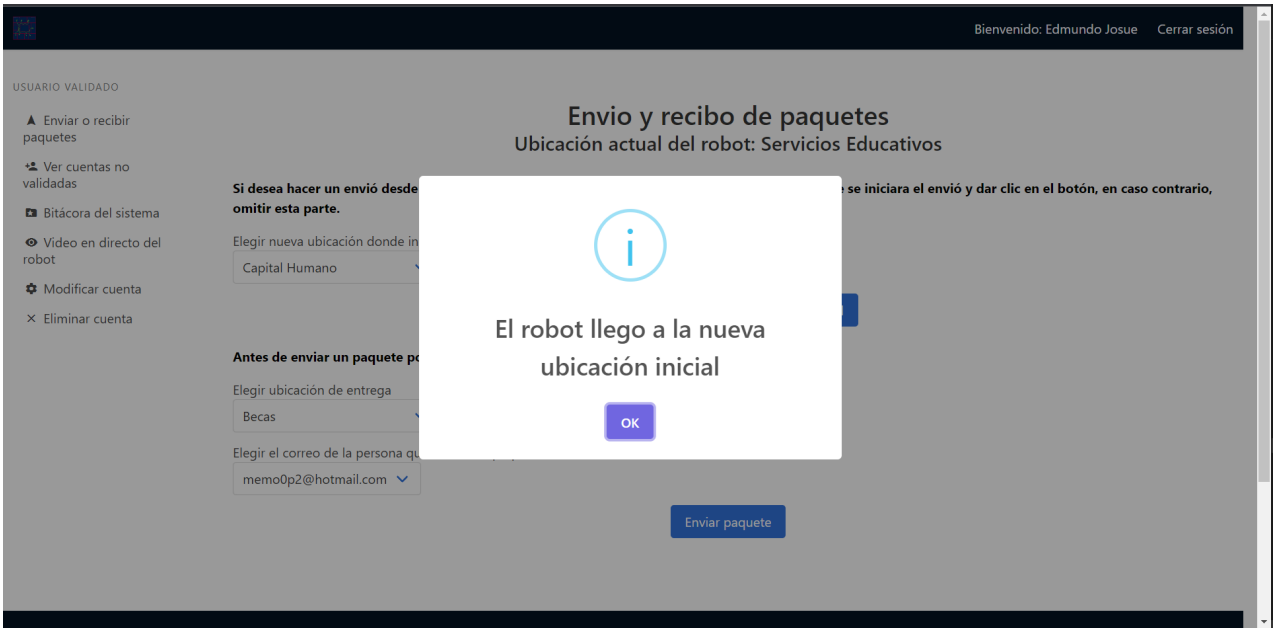


Figura 107: Robot finalizo su reubicación. Mensaje al usuario que realizo el movimiento.

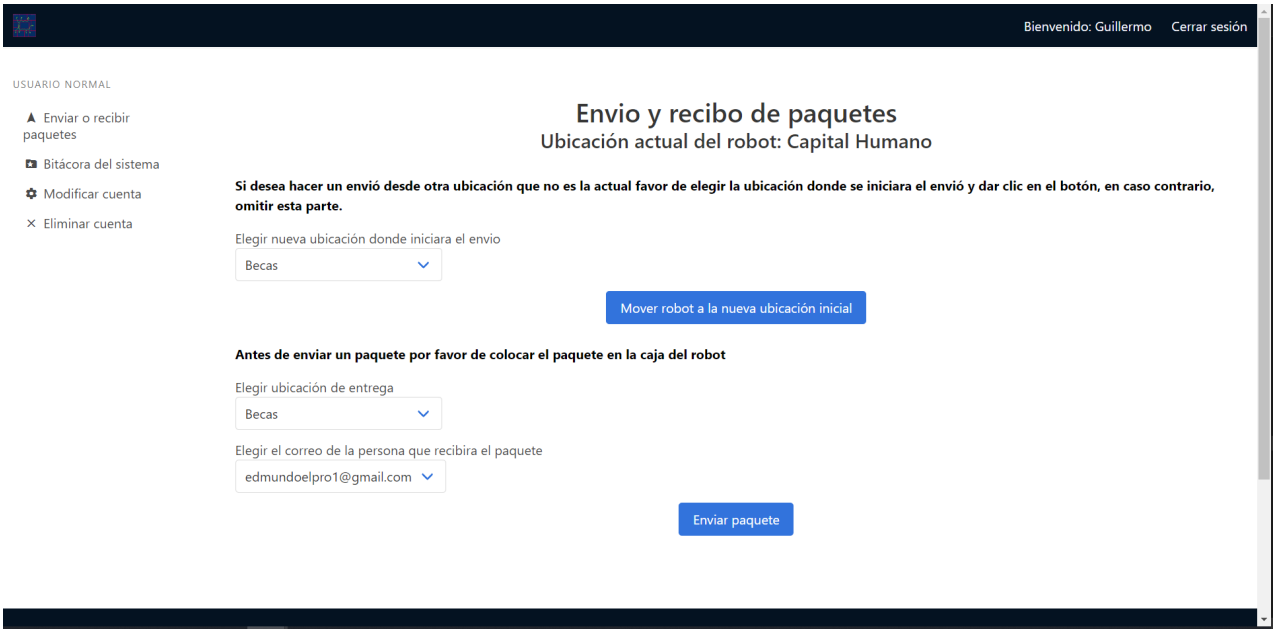


Figura 108: Página nuevamente utilizable por cualquier usuario.

7.1.1.5. Enviar o recibir paquetes(Enviar paquete)

Continuando con el planteamiento anterior, veamos ahora como seria el funcionamiento de enviar paquetes, el recibo de paquete como vemos aun no se tiene un caso de uso ya que aun en este punto de escribir este documento aun no tenemos bien definido como

será el funcionamiento de este, lo único que tenemos bien definido es que necesitaremos implementar algún tipo de criptografía, como la híbrida o realizar una forma de firma digital.

Ahora bien, vayamos con el funcionamiento de este módulo. Es muy sencillo, por el momento, solo necesitamos elegir el correo de la persona que recibirá el paquete así como la ubicación de entrega, como lo podemos ver en la figura 102, de igual forma vemos en la figura 109 que en las ubicaciones de entrega disponibles no se encuentra la ubicación actual del robot, ahora como vemos en la figura 110 nos aparecen los usuarios que pueden recibir paquetes, es decir, administradores y usuarios básicos que ya hayan sido validados, por el momento el sistema cuenta con solo dos usuarios registrados, uno básico y otro administrador y ahora solo nos queda dar clic en el botón Enviar paquete.

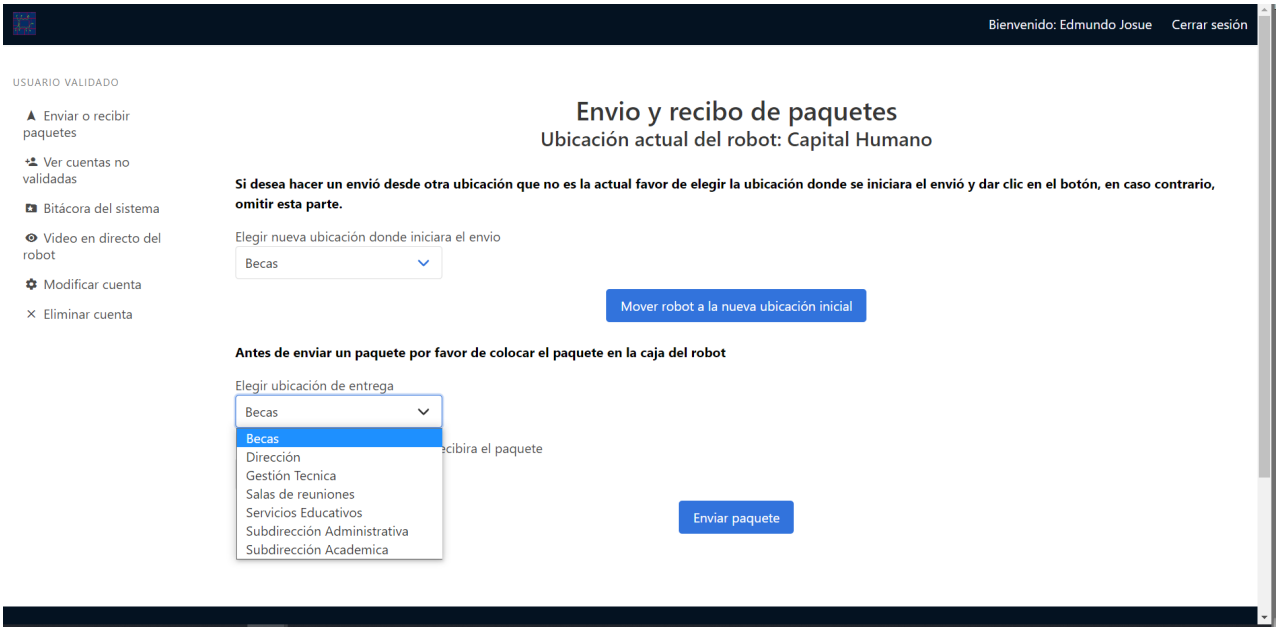


Figura 109: Muestra de ubicaciones disponibles.

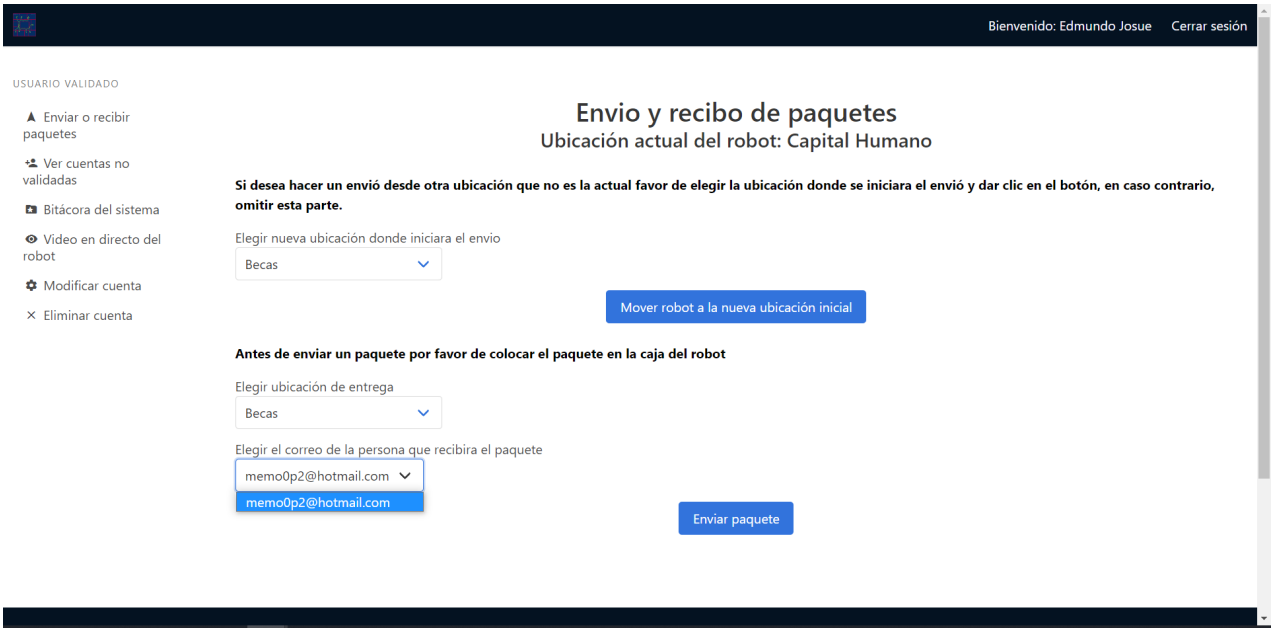


Figura 110: Muestra de usuarios que pueden recibir paquetes.

Finalmente vemos nuevamente como se ve la página cuando el robot estaría en movimiento tanto para la persona que envía el paquete como para los demás usuarios, esto lo podemos ver en la figura 111 y 112, ahora bien, cuando el robot finaliza el entrega del paquete podemos ver en la figura 113 volvemos a ver como esta pagina es nuevamente usable y que ademas la ubicación del robot es nuevamente actualizada, esto lo podemos ver en la figura 114.

Figura 111: Robot en movimiento. Punto de vista del usuario que realizo el envió de paquete.

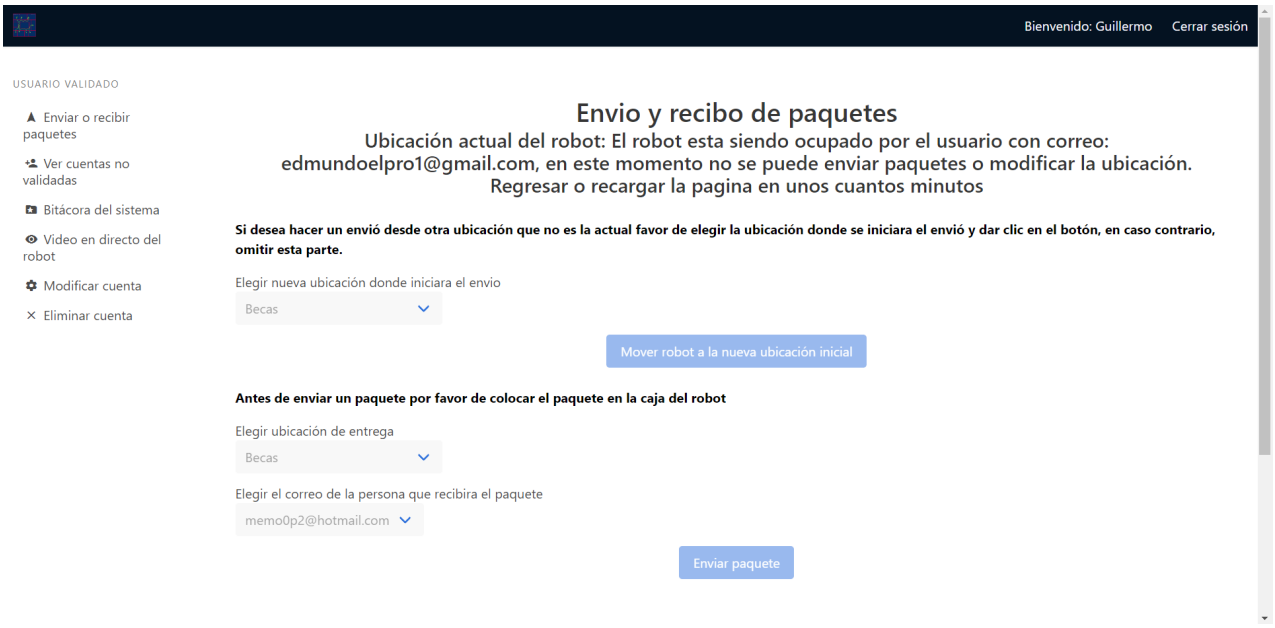


Figura 112: Robot en movimiento. Punto de vista del usuario que no realizo el envió de paquete.

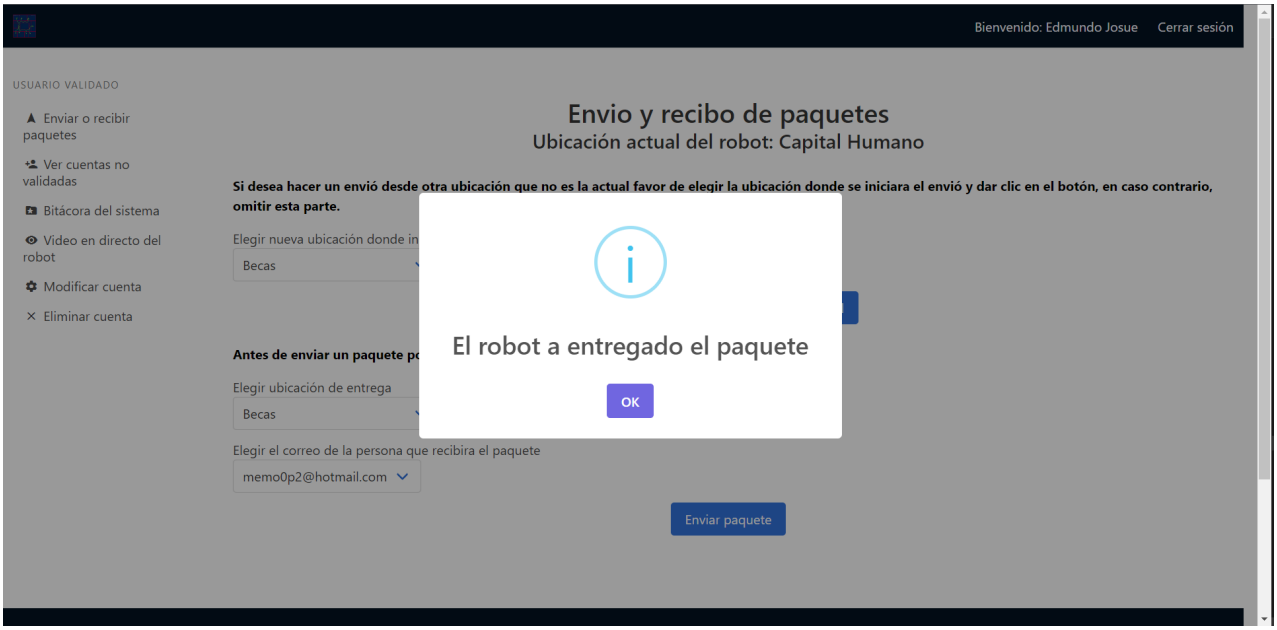


Figura 113: Robot finalizo el envió del paquete. Mensaje al usuario que realizo el envió de paquete.

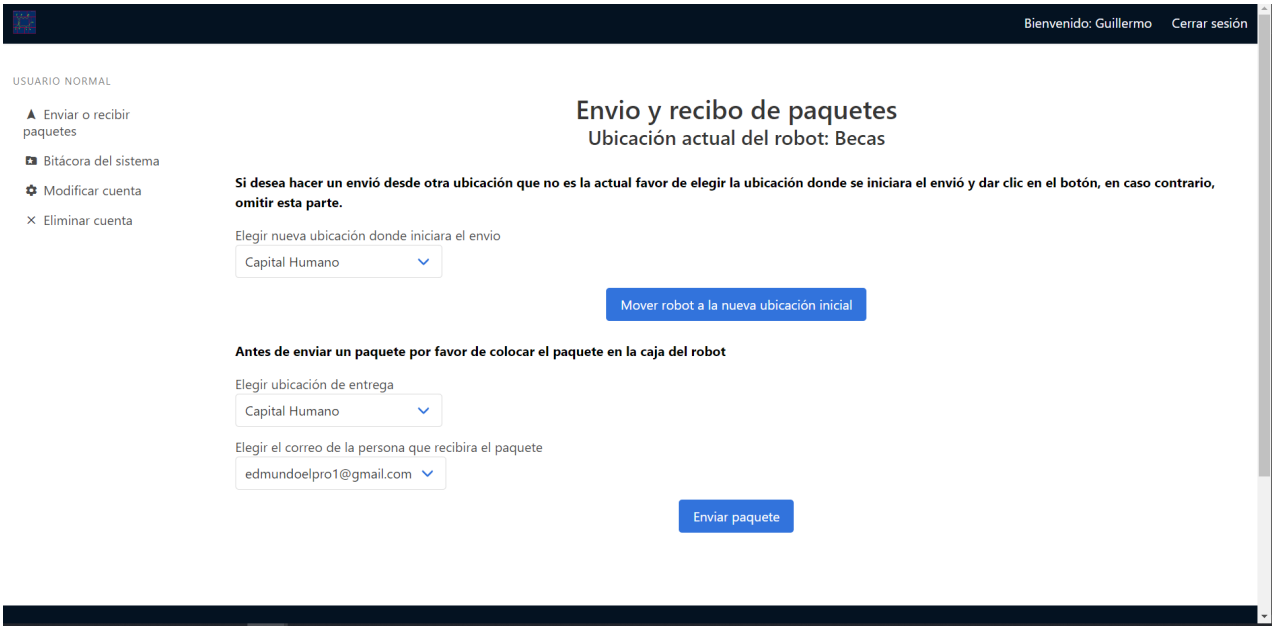


Figura 114: Página nuevamente utilizable por cualquier usuario.

7.1.1.6. Bitácora del sistema

Ahora vayamos a la parte de la bitácora del sistema, en donde cualquier tipo de usuario puede acceder, sin embargo, se les muestra diferente información, en la figura 115 podemos ver el punto de vista de un usuario administrador en donde podemos ver todo lo que ocurra

en el sistema, por el contrario, para los usuarios básicos como vemos en la figura 116 solo nos muestra aquellos eventos que afecten a este usuario.

Figura 115: Bitácora del sistema desde el punto de vista de un administrador.

Figura 116: Bitácora del sistema desde el punto de vista de un usuario básico.

7.1.1.7. Vídeo en directo del robot

En esta sección veremos el funcionamiento de unos de los módulos exclusivos para el usuario tipo administrador, el cual es ver el vídeo en directo del robot, el fin de este modulo es para poder ver la cámara que tiene incorporada nuestro robot la cual hace uso de un programa conocido como *Motion* el cual esta desarrollado para monitorear señales de vídeo de varios tipos de cámara, en nuestro caso en una cámara de 360° de la marca Steren, la salida del vídeo es en directo y el programa *Motion* nos da la posibilidad para poder detectar solo cuando hay movimiento y mandar el vídeo en directo. En la figura 117 podemos ver como se ve el vídeo.

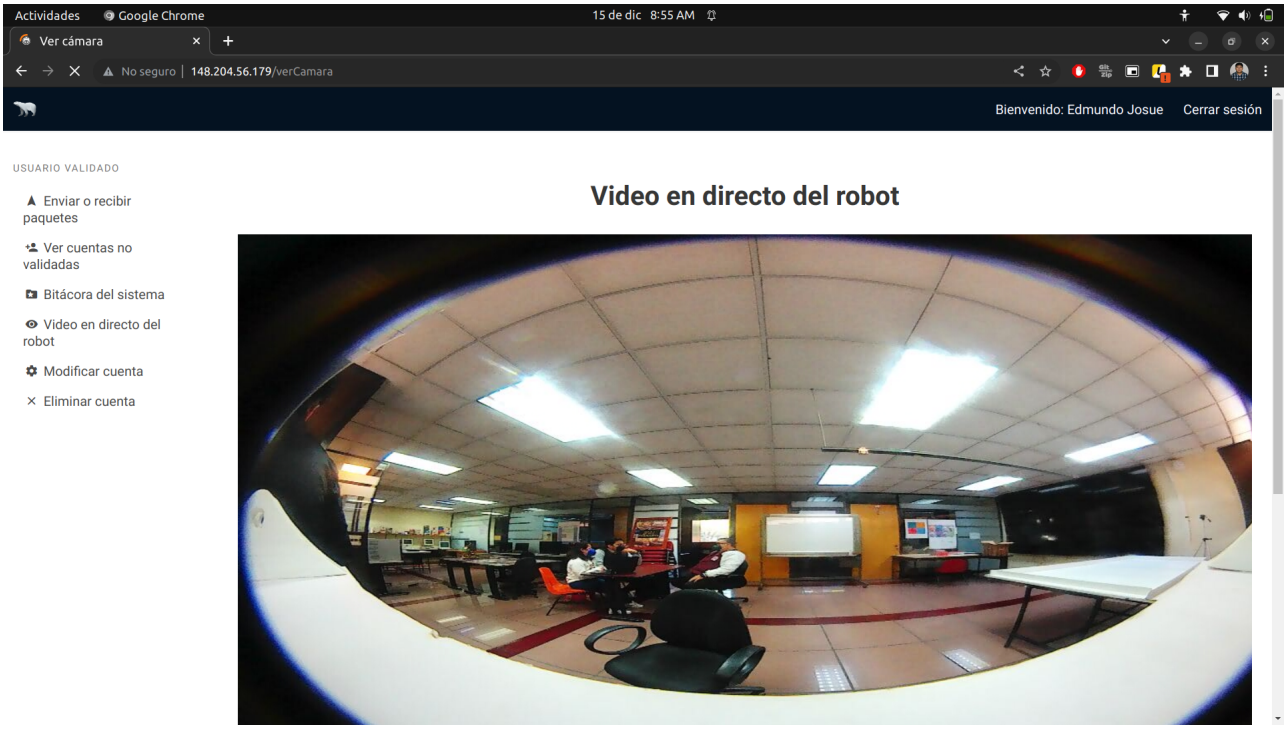


Figura 117: Vídeo en directo del robot.

7.1.1.8. Modificar cuenta(ver información)

Ahora volvamos con un módulo, que es compartido en funcionamiento por los dos tipos de usuarios, este modulo está dividido en dos partes la primera es donde nos permite ver la información del usuario, a excepción de la contraseña como lo podemos ver en la figura 118.

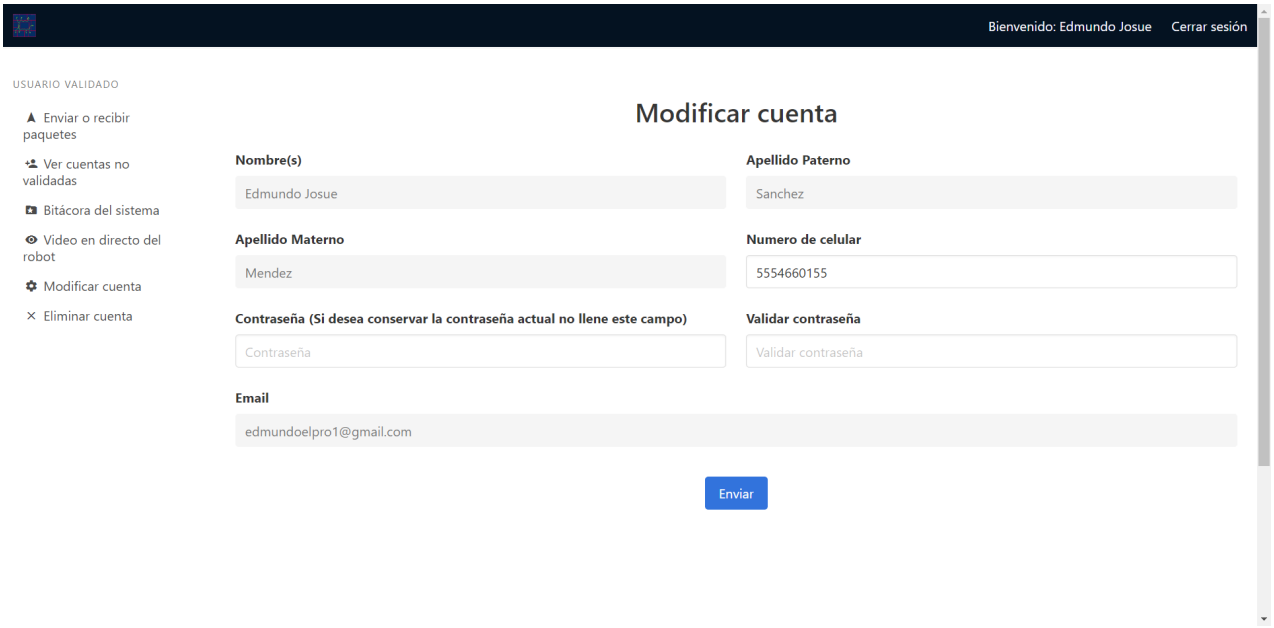


Figura 118: Modificar cuenta(ver información).

7.1.1.9. Modificar cuenta(modificar cuenta)

Continuando con la parte anterior ahora veamos la parte de modificar la cuenta, los datos que podemos modificar son el numero celular y modificar la contraseña, en el caso de que solo se desea modificar el numero celular entonces los campos de contraseña y validar contraseña tendrá que ser dejados vacíos, en caso contrario tendrá que ser llenados, después de llenar los campos que se quieran modificar se tendrá que dar clic en el botón Enviar, confirmar la acción y finalmente dar Ok en el mensaje resultante.

Un ejemplo de uso seria modificar tanto la contraseña como el número de celular en la figura 119 vemos como modificamos el numero celular comparando con la figura 118 y tenemos una nueva contraseña, damos clic en el botón Enviar, confirmamos la acción como lo vemos en la figura 120 y finalmente damos Ok en el mensaje mostrando como en la figura 121. Finalmente en la figura 122 podemos ver como el numero de celular también fue modificado así como la contraseña de del usuario .

Figura 119: Nuevos datos a modificar.

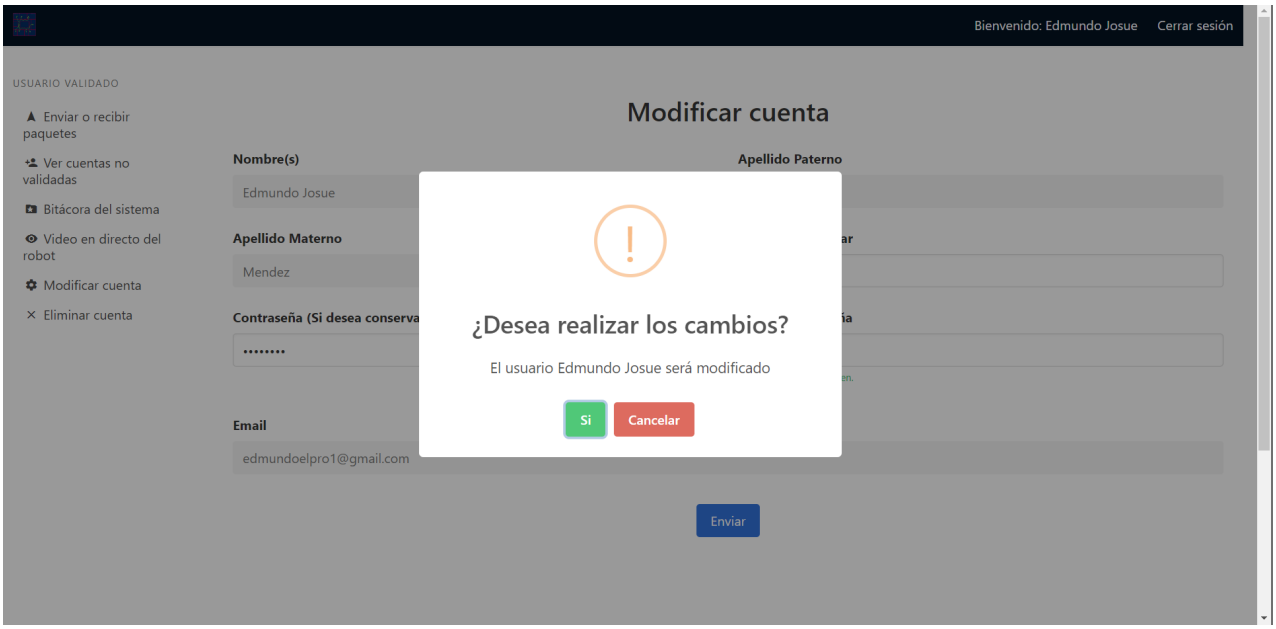


Figura 120: Confirmar acción de modificar.

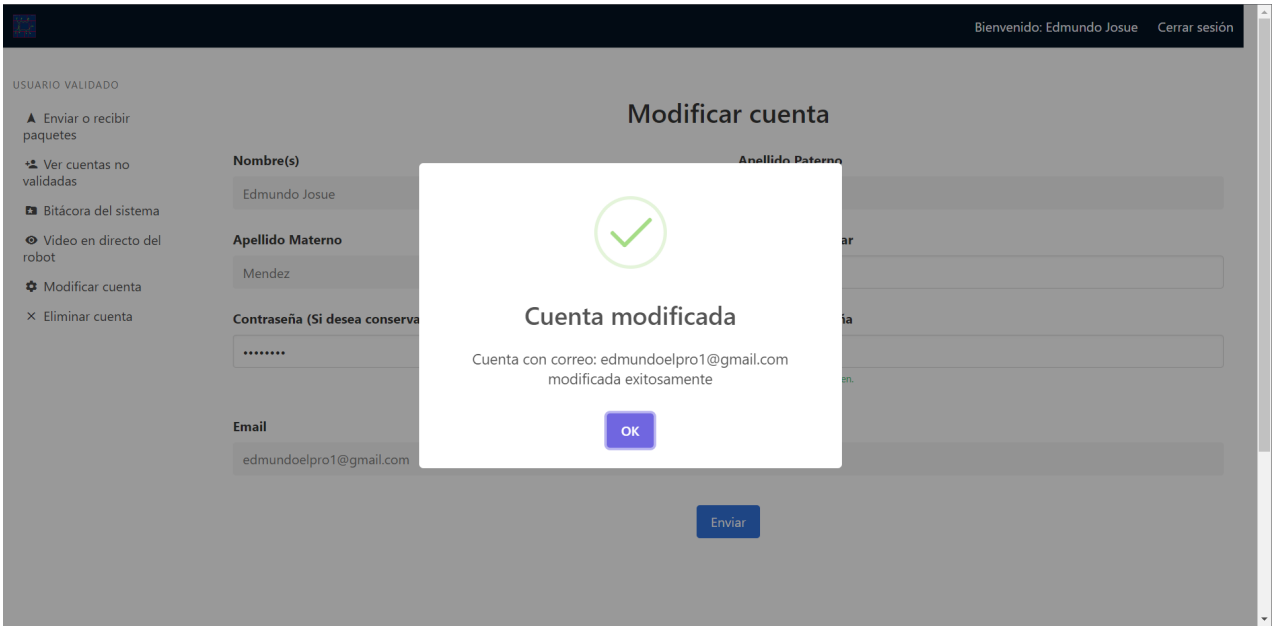


Figura 121: Cuenta modificada exitosamente.

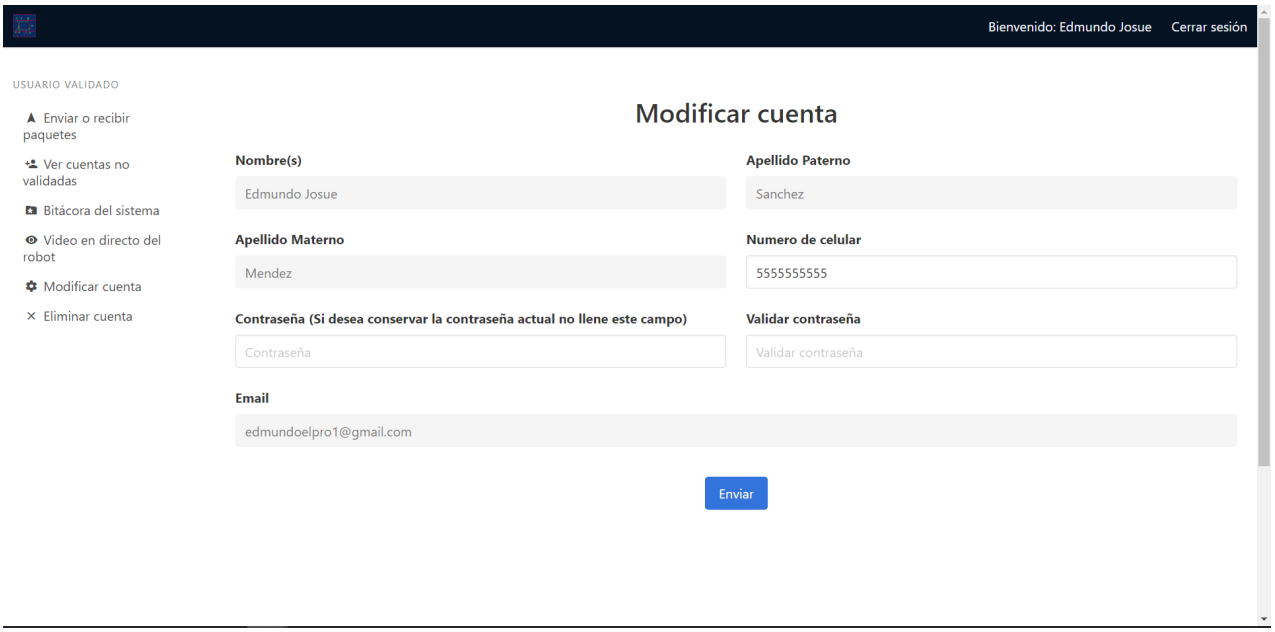


Figura 122: Nuevos datos de usuario.

7.1.1.10. Eliminar cuenta

Continuando con el penúltimo modulo que es similar entre los dos usuarios es el de eliminar, para hacer uso de este modulo solo se tiene que dar clic Eliminar cuenta, confirmar la acción y la cuenta es eliminada, mencionar que esta acción se ve reflejada en el módulo

de Bitácora del sistema para los administradores y los usuarios que hayan eliminado su cuentan pueden volver a registrarse si así lo desean, pero nuevamente tendrán que ser validados.

Un ejemplo de su uso lo vemos a continuación como en la figura 123 vemos la pantalla del módulo, en donde tenemos una pequeña leyenda en negritas para llamar algo la atención, damos clic en el botón rojo y como vemos en la figura 124 nos pide confirmar la acción, damos clic en el botón verde y finalmente en la figura 125 nos dice que la cuenta fue eliminada con éxito cerramos damos clic en Ok y nos saca del sistema, hacemos una prueba de intenta ingresar con el usuario eliminado y vemos que no nos deja ingresar como vemos en la figura 126.

Figura 123: Pagina para eliminar cuenta.

Figura 124: Confirmar acción de eliminar cuenta.

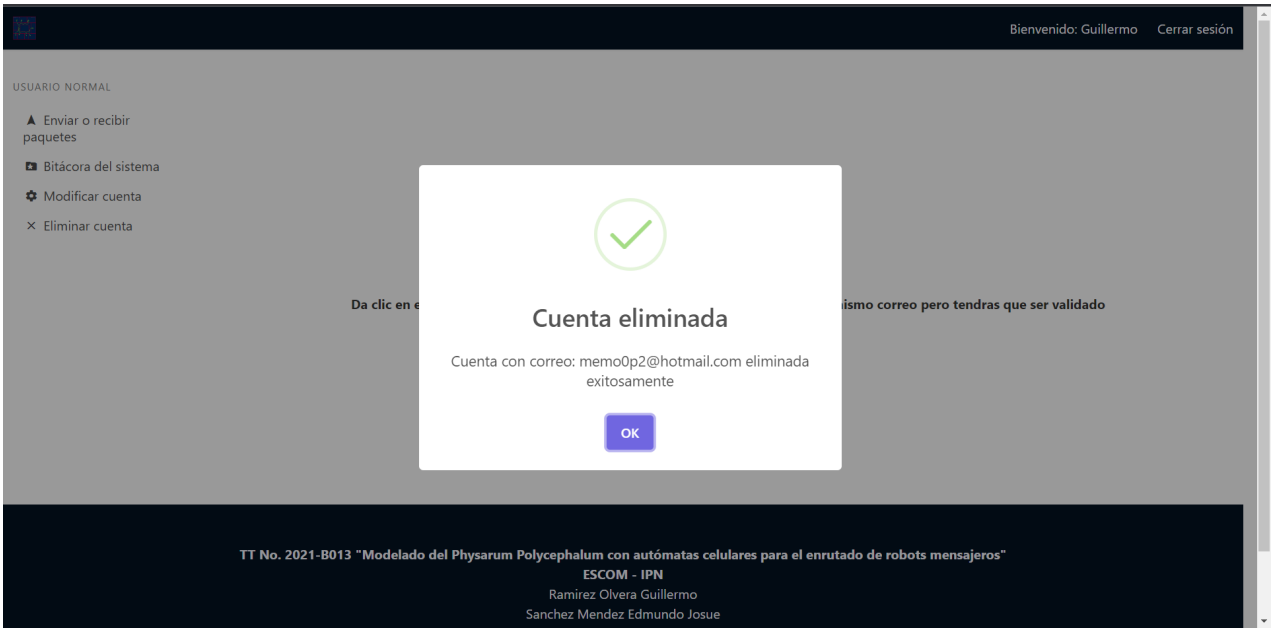


Figura 125: Cuenta eliminada exitosamente.

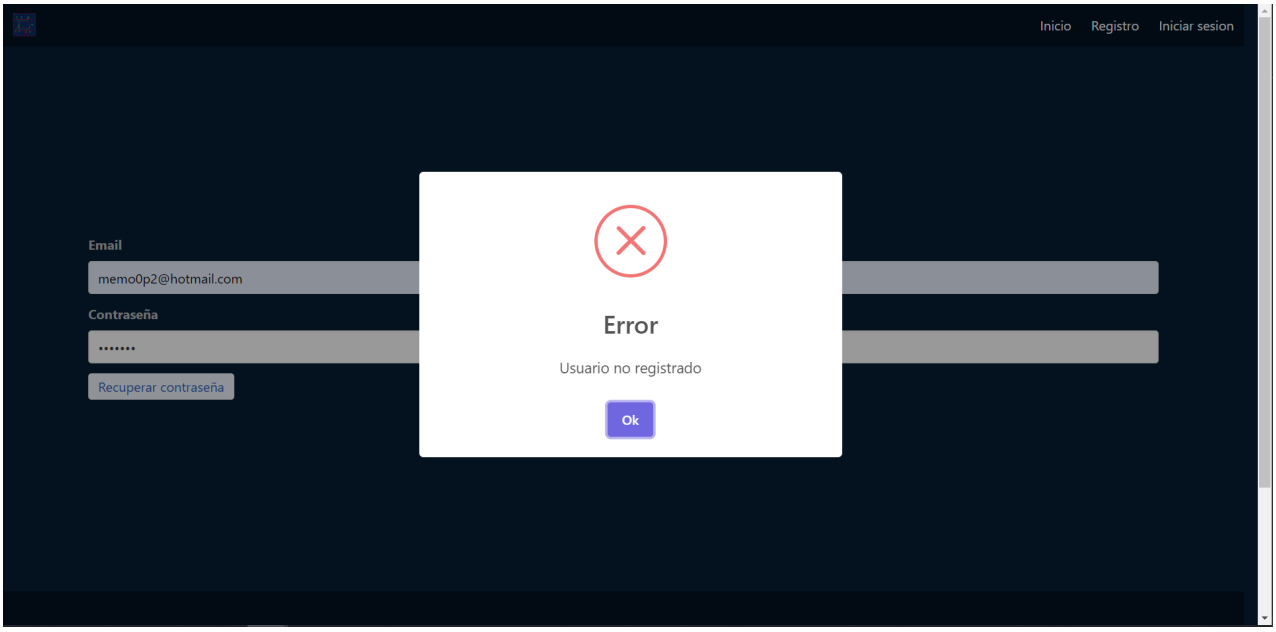


Figura 126: Prueba de ingresar nuevamente con la cuenta eliminada.

7.1.1.11. Recuperar contraseña(solicitud de enlace)

Ahora vayamos con el ultimo modulo compartido por ambos usuarios el cual es el recuperar contraseña, en esta parte veremos la recuperación de contraseña, para ello debemos ir al módulo de iniciar sesión como lo pudimos ver en la figura 91, después de estar ahí vamos a la opción de Recuperar contraseña y viendo la figura 127 vemos que nos solicita solamente el correo electrónico del cual se desea recuperar la contraseña, en caso de que no haya registrado algún usuario con el correo ingresado no será mandado ningún correo, en otro caso, el correo será mandado y vemos en la figura 128 como la página cambia a un círculo en movimiento indicando que se está enviando el correo y como al final en la figura 129 nos indica que el correo ya fue mandado de manera exitosa.

Figura 127: Pagina para recuperar contraseña (solicitud de enlace).

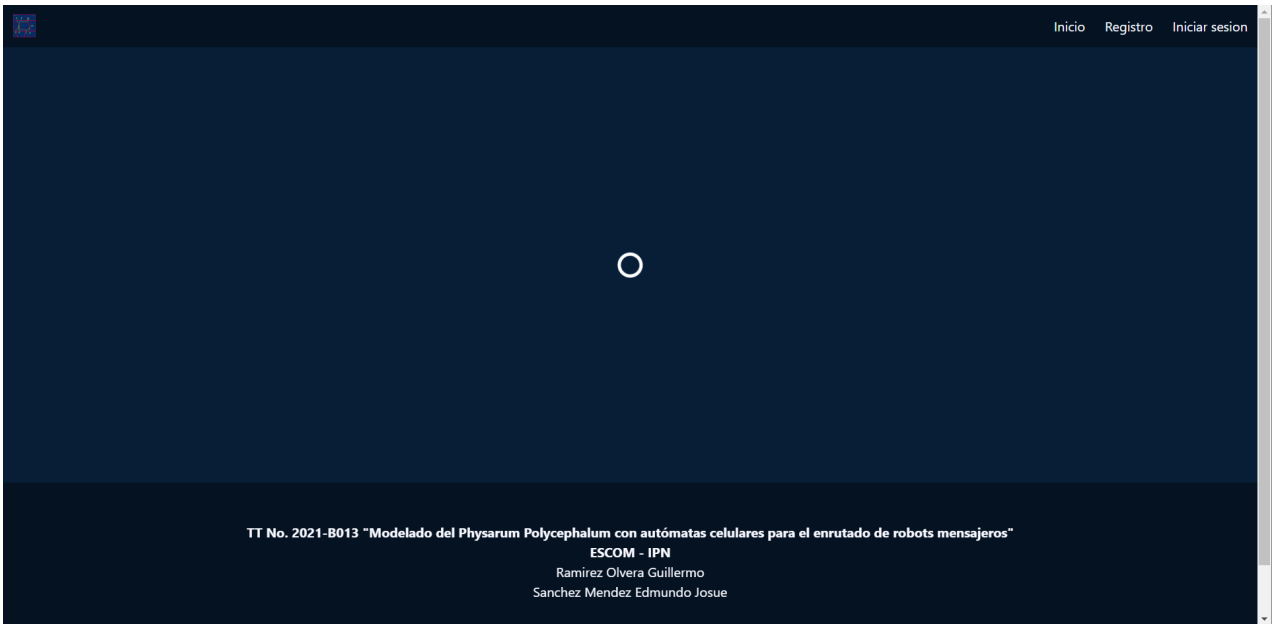


Figura 128: Enviando correo.

Figura 129: Correo enviado exitosamente.

7.1.1.12. Recuperar contraseña(formulario cambio de contraseña)

Continuando con lo anterior, el formato del correo es el que podemos ver en la figura 130, en donde nos indica que queremos hacer recuperación de contraseña junto con el nombre completo del usuario, así mismo vemos como al final del correo se encuentra un enlace que dice Recuperar contraseña, es importante mencionar que este enlace tiene un tiempo de vida de 5 minutos y que después de este tiempo el enlace quedara inutilizable.

Ahora vemos en la figura 131 como nos pide ingresar una nueva contraseña y confirmar esta nueva contraseña, damos clic a Cambiar contraseña y vemos en la figura 132 como nos indica que la contraseña fue cambiada con éxito, damos clic en el botón Iniciar sesión y nos lleva a la página de iniciar sesión.

Figura 130: Correo de recuperación de contraseña.

Figura 131: Formulario para la actualización de la contraseña.

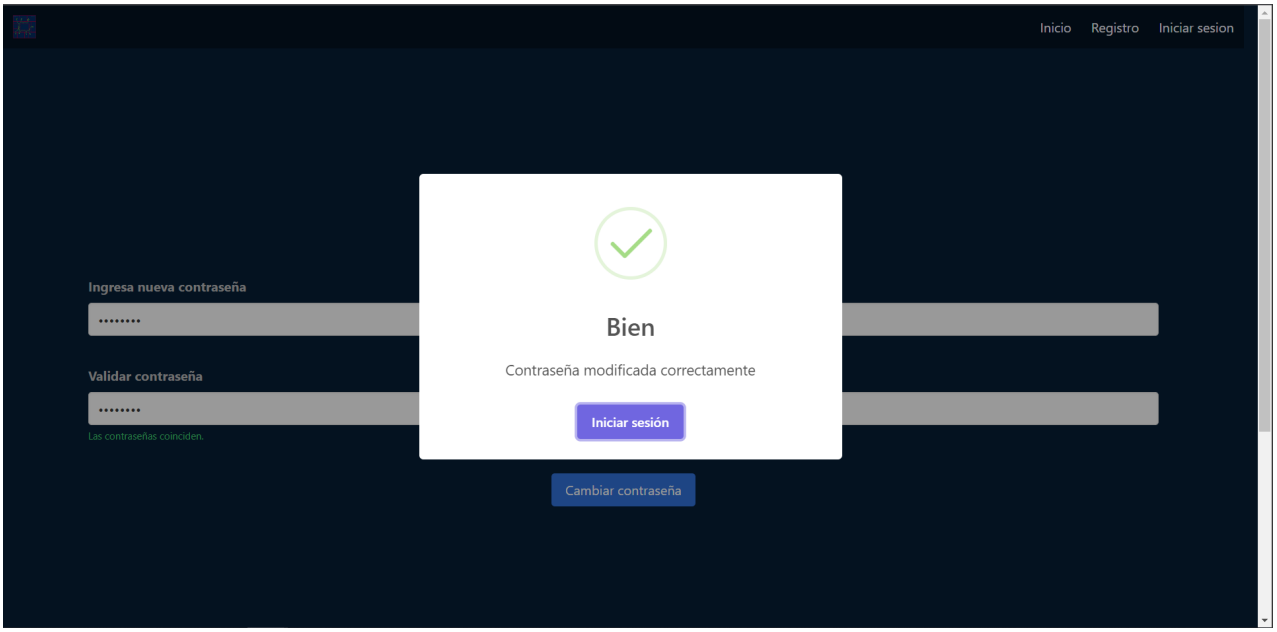


Figura 132: Contraseña actualizada de manera correcta.

7.1.1.13. Ver cuentas no validadas (ver cuentas)

Ahora vayamos con el ultimo modulo del sistema, el cual es exclusivo para el usuario tipo administrador, este modulo esta dividido en 3 partes, ver cuentas a validar, validar cuenta o eliminar cuenta, vayamos con el primero el cual es ver cuentas a validar, como vemos en la figura 133 nos muestra una cuenta que solicito entrar al sistema y vemos su información básica a excepción del correo, así como la contraseña y las opciones de eliminar o validar cuenta.

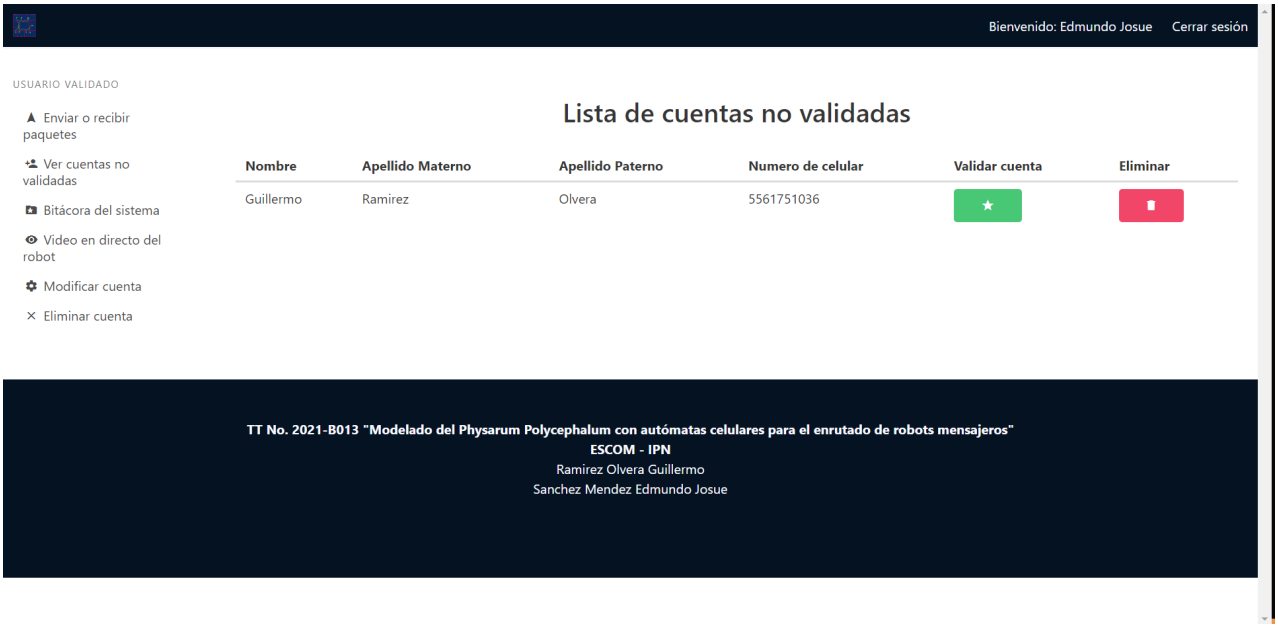


Figura 133: Listado de cuentas por validar en el sistema.

7.1.1.14. Ver cuentas no validadas (validar cuenta)

Continuando con el punto anterior veamos cómo se hace la validación de una cuenta, solo entramos al módulo correspondiente, buscamos al usuario que queramos validar y damos clic en el botón de color verde con una estrella en medio, nos sale un mensaje de confirmación como lo podemos ver en la figura 134, confirmamos la acción y finalmente nos vuelve a salir un mensaje que nos indica que el usuario fue validado correctamente como lo vemos en la figura 135.

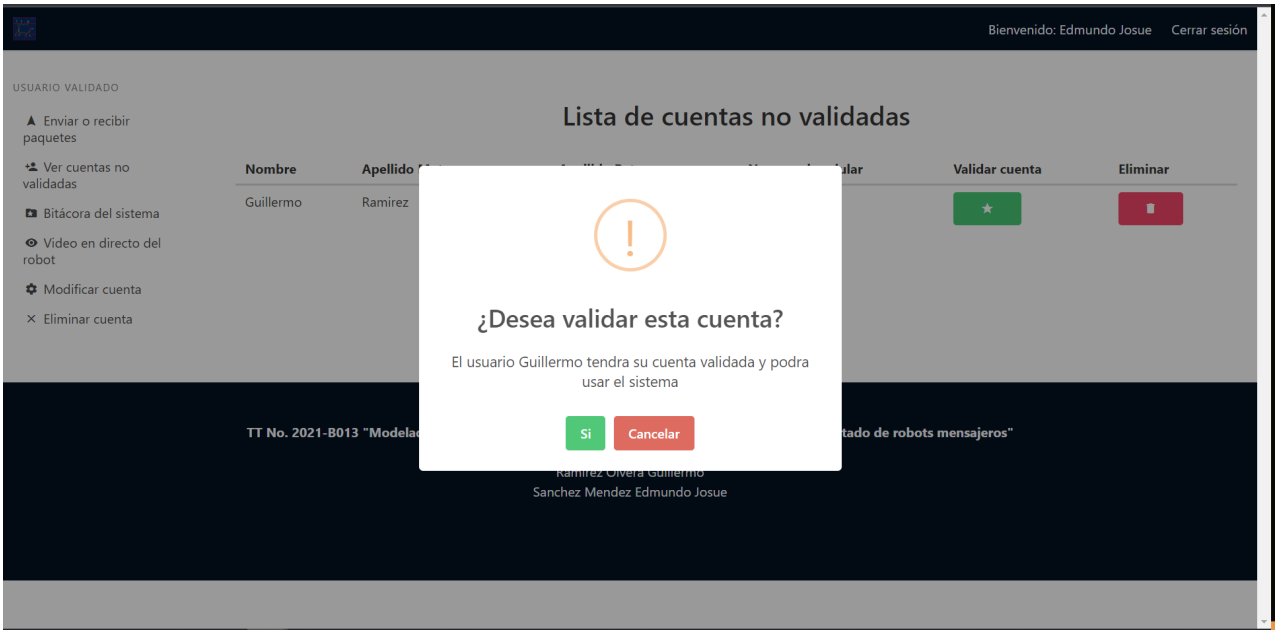


Figura 134: Validando cuenta.

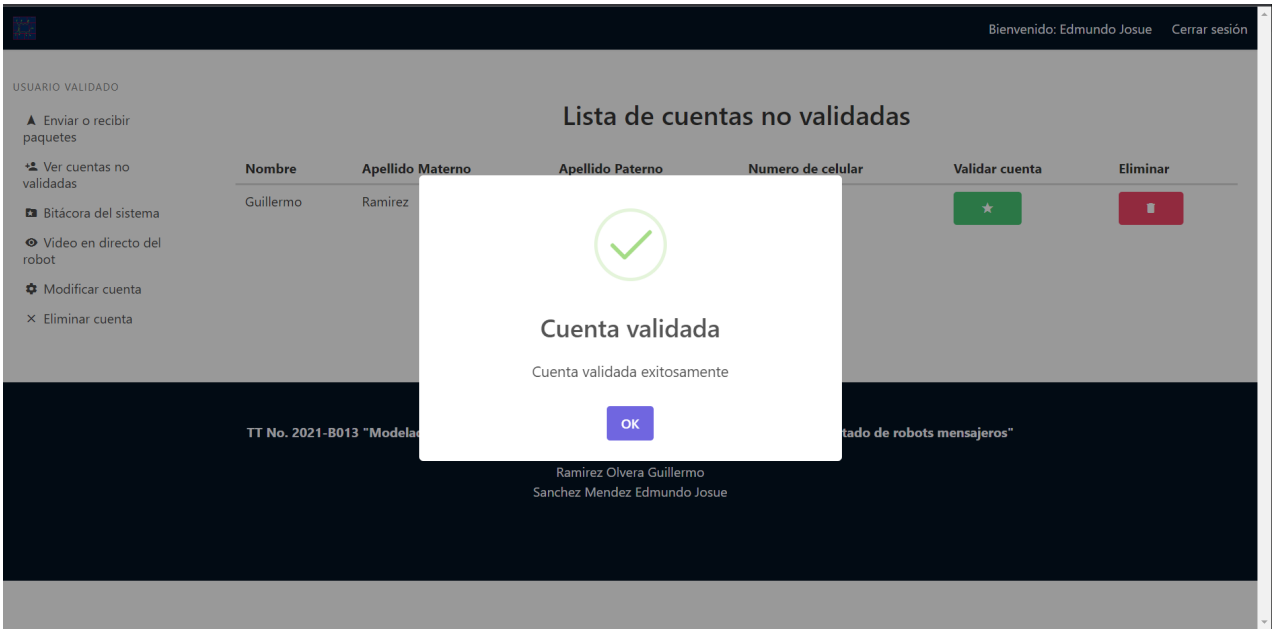


Figura 135: Cuenta validada exitosamente.

7.1.1.15. Ver cuentas no validadas (eliminar cuenta)

Finalmente veamos cómo se hace la eliminación de solicitud de una cuenta, solo entramos al módulo correspondiente, buscamos al usuario que queramos eliminar y damos clic en el botón de color rojo, nos sale un mensaje de confirmación como lo podemos ver en la figura

136, confirmamos la acción y finalmente nos vuelve a salir un mensaje que nos indica que la solicitud del usuario fue eliminada correctamente como lo vemos en la figura 137.

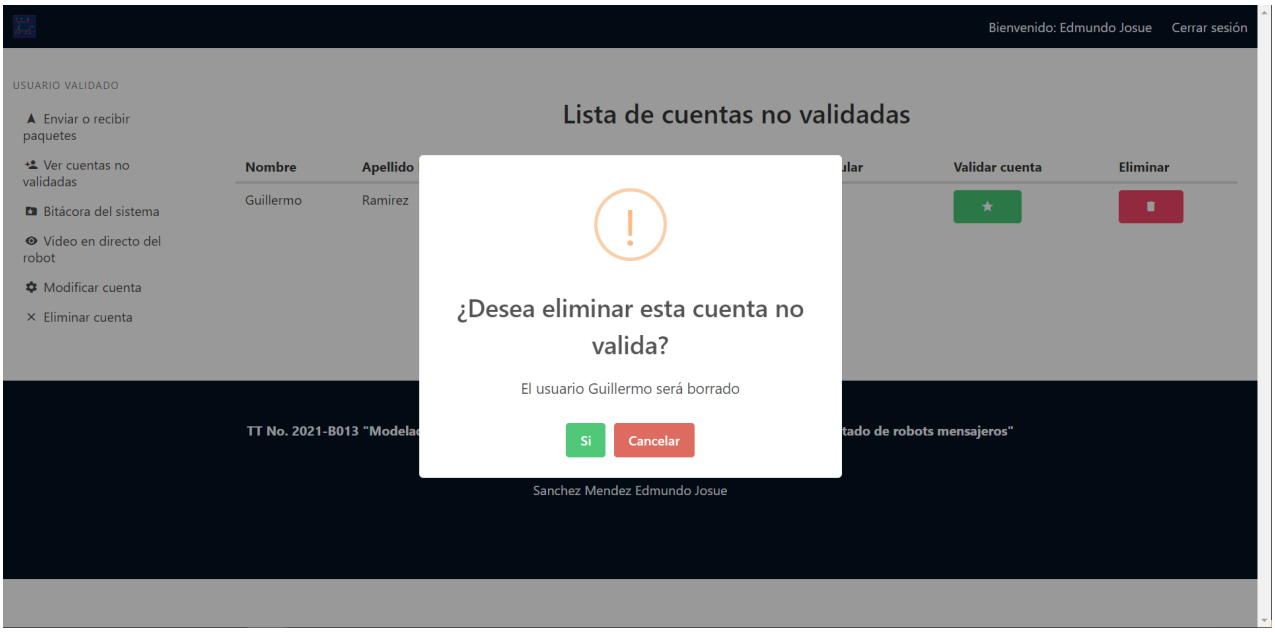


Figura 136: Eliminando cuenta.

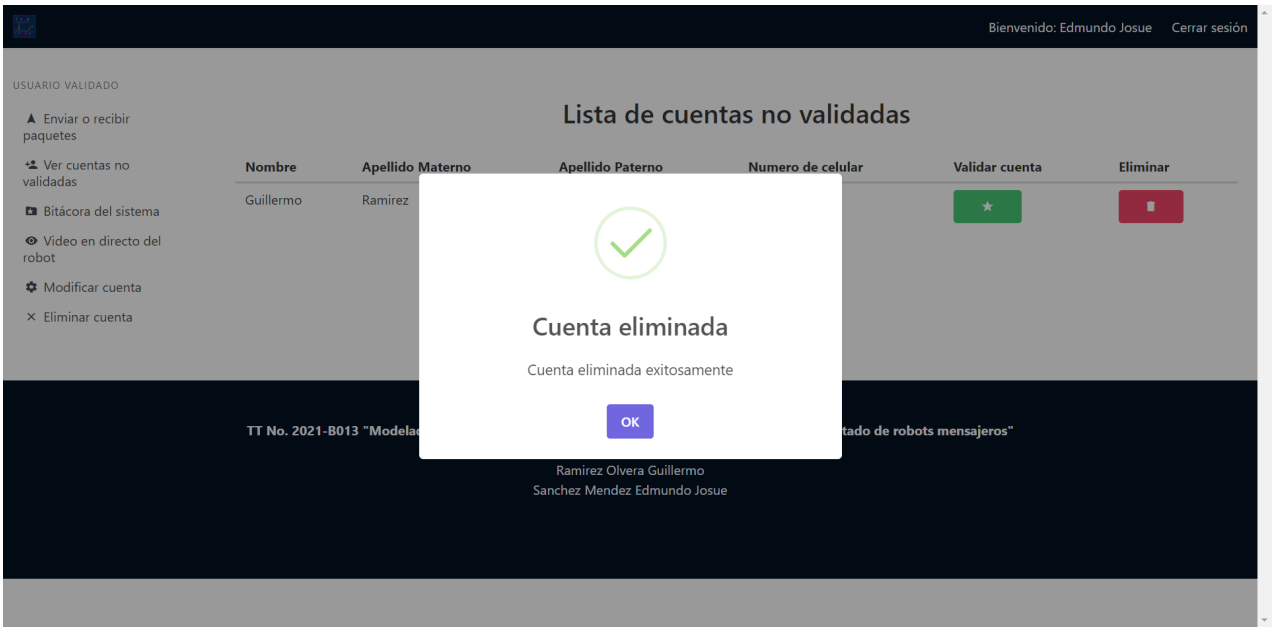


Figura 137: Cuenta eliminada exitosamente.

7.1.2. Versión final de la interfaz usuario - robot (Servicio Web)

En esta versión final solo añadiremos el funcionamiento de un nuevo módulo, así como una nueva característica de un modulo anteriormente explicado, el nuevo modulo es el de recibir un paquete enviado y el modulo con una nueva característica es el modulo de enviar paquete, así que tomando como base el primer prototipo veamos entonces el funcionamiento del nuevo modulo.

7.1.2.1. Enviar o recibir paquetes(Enviar paquete)

Ahora veamos las nuevas características de este módulo, ahora al momento de enviar un paquete y que este sea colocado en la caja se enviará un correo al destinatario informando que se esta enviando un paquete para él y la ubicación final que tendrá el robot, como vemos en la figura 138, también al momento de que el robot llega al destino se vuelve a enviar un correo al destinatario avisando que el robot ya esta en el destino y que reciba el paquete, como vemos en la figura 139, el funcionamiento de esto se explica más adelante.

Figura 138: Correo avisando de que hay un paquete en camino.

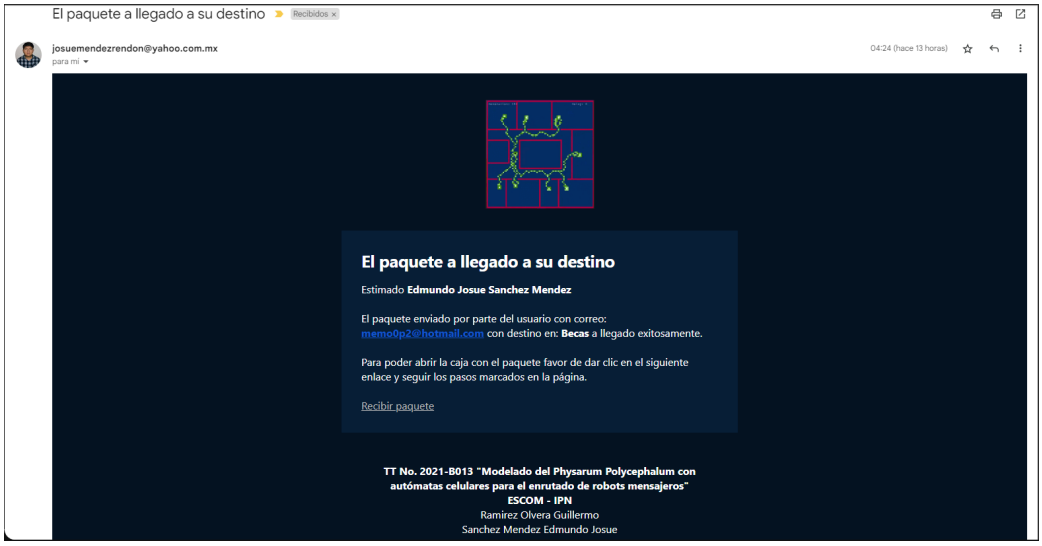


Figura 139: Correo avisando de que el paquete ya llego al lugar destino.

Finalmente hay que comentar que todo lo explicado anteriormente en el primer prototipo del servicio web sigue funcionando exactamente igual.

7.1.2.2. Enviar o recibir paquetes(Recibir paquete)

Ahora veamos el funcionamiento del nuevo módulo, este módulo es solo sobre recoger el paquete y trabaja de la siguiente manera, el robot al llegar a la ubicación final mandara un correo a la persona que recibirá el paquete con un enlace (como lo podemos ver en la figura 139), al entrar en el enlace vemos una página simple que podemos ver en la figura 140, en donde al dar clic al botón con la leyenda “Recoger paquete” se desbloqueara la caja permitiendo así poder abrirla y sacar las cosas de adentro , después se le solicitara al usuario dar clic a un botón con la leyenda “Confirmar paquete entregado” como se ve en la figura 141 y finalmente se le mandara un correo al usuario que mando el paquete informándole que el paquete fue entregado con éxito como lo vemos en la figura 142.

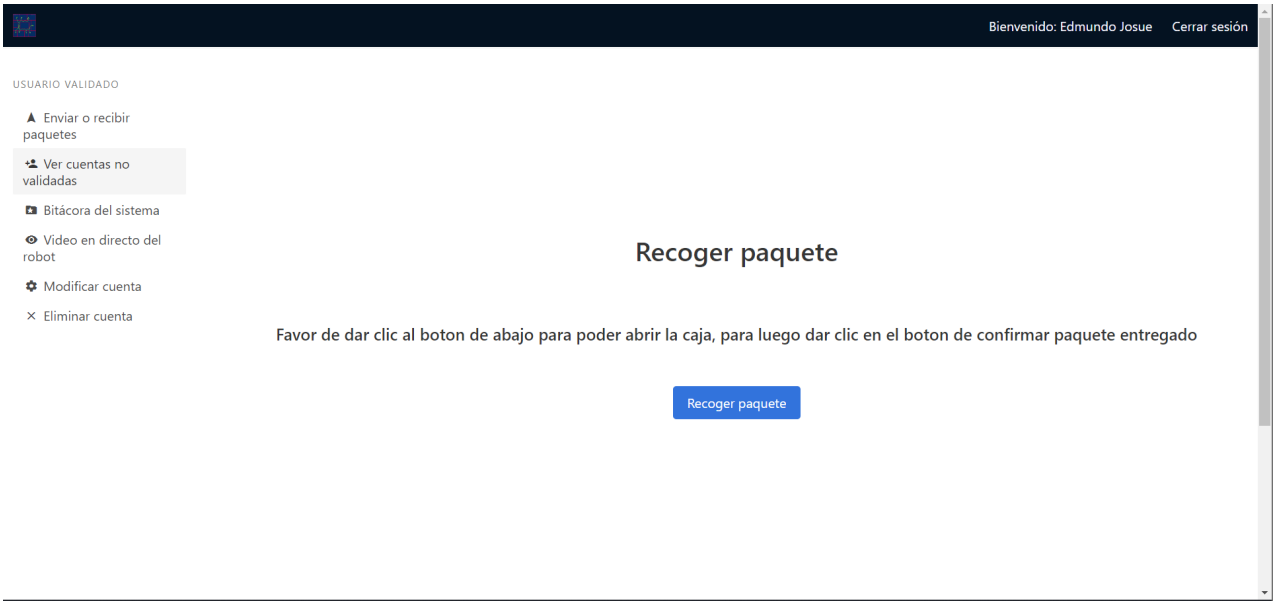


Figura 140: Pagina recoger paquete.

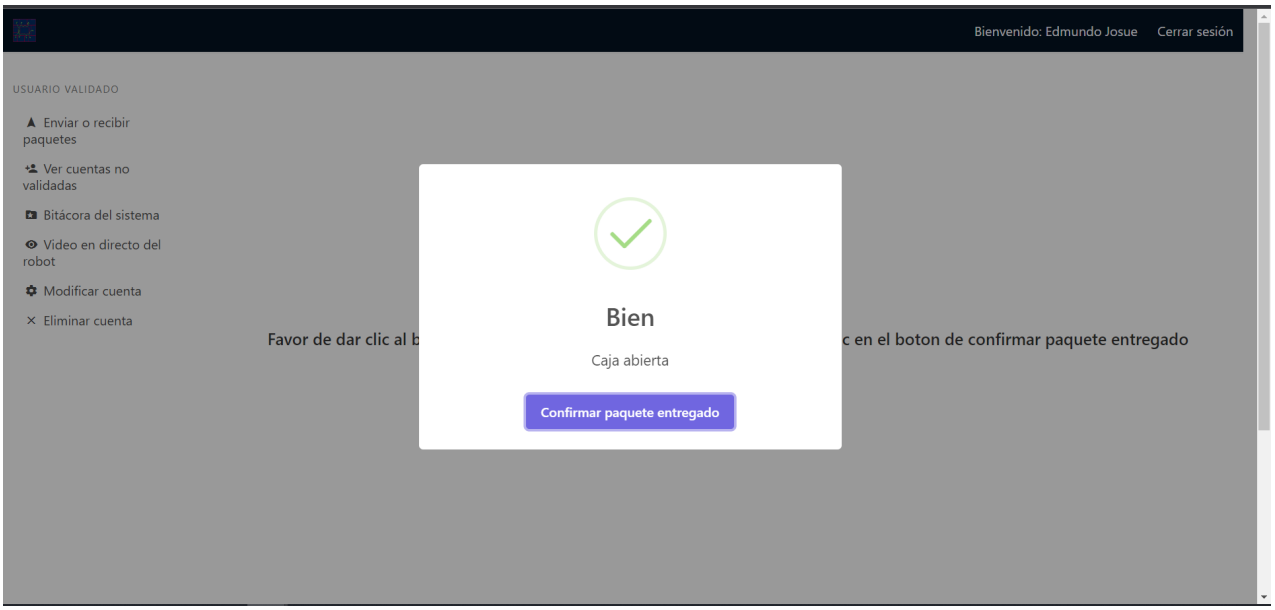


Figura 141: Confirmar entrega del paquete.

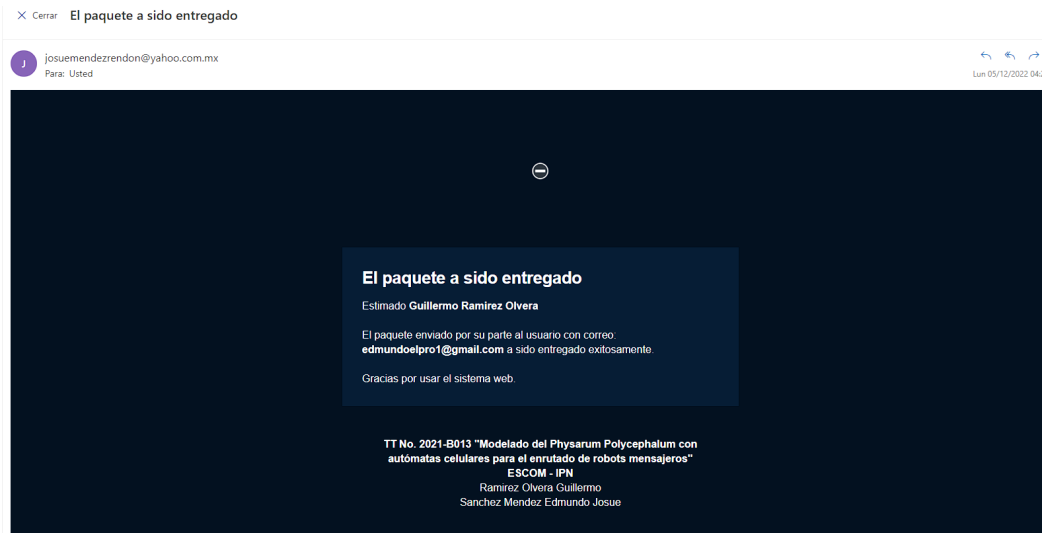


Figura 142: Correo avisando de paquete entregado con éxito.

Hay que mencionar que el servicio de envió se detiene ya que necesitamos que el paquete sea recogido.

7.2. Implementación del Servicio Web

En esta sección veremos cómo se realizó la implementación del servicio web en la Raspberry Pi 4. Para hacer esto hacemos uso de Apache el cual nos permite levantar un servidor web de una manera muy sencillo, dejamos la configuración por defecto y hacemos uso de WSGI, WSGI son las siglas de Web Server Gateway Interface el cual es una especificación que permite la comunicación de un servidor web con una aplicación web, WSGI fue creado con la finalidad de ejecutar código Python. Una vez implementado lo anterior era necesario tener una IP publica para que todos los usuarios puedan hacer uso de este sistema web desde cualquier dispositivo, ya sea móvil o de escritorio, para ello se solicito a la Unidad De Informática (UDI) de la Escuela Superior de Computo, por medio de nuestro director el Dr. Genaro Juárez Martínez, se nos asignara una IP en el servidor de la escuela, la IP asignada es la siguiente 148.204.56.179, es aquí en donde se alojara el servicio. Para poder hacer todo el trabajo anterior se tuvo que crear una estructura de red en la cual se compró un rúter y se configuro con la IP anteriormente asignada y se creó una red Wifi con contra-

seña WPA2, ademas de ello se agrego un repetidor para que la red tenga un mayor alcance en el primer piso del edificio de gobierno de la ESCOM. Después de esto se realizaron dos configuraciones extra. La primera que podemos ver en la figura 143 que trata sobre la redirección de puertos, teniendo que configurar el puerto 80 para el servicio web y el puerto 8081 para el video en vivo de la cámara, especificando la IP de la Raspberry Pi 4 la cual ya es una IP fija en la subred creada, así como el protocolo de comunicación a usar, la segunda configuración realizada la podemos ver en la figura 144 en donde configuramos una zona desmilitarizada (DMZ) en palabras simples exponemos la IP fija de la subred a internet solamente con los puertos redireccionados. Finalmente, en la figura 145 podemos ver como el servicio web ya está en internet.

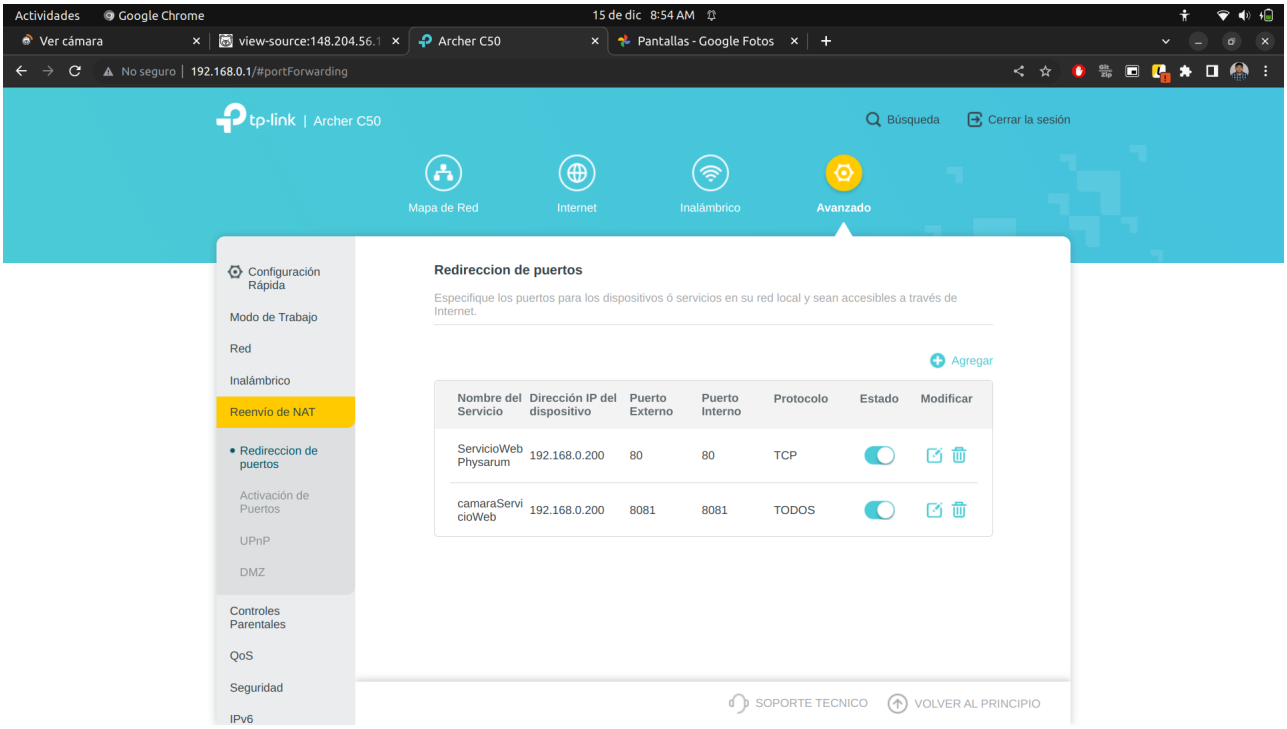


Figura 143: Redirección de puertos.

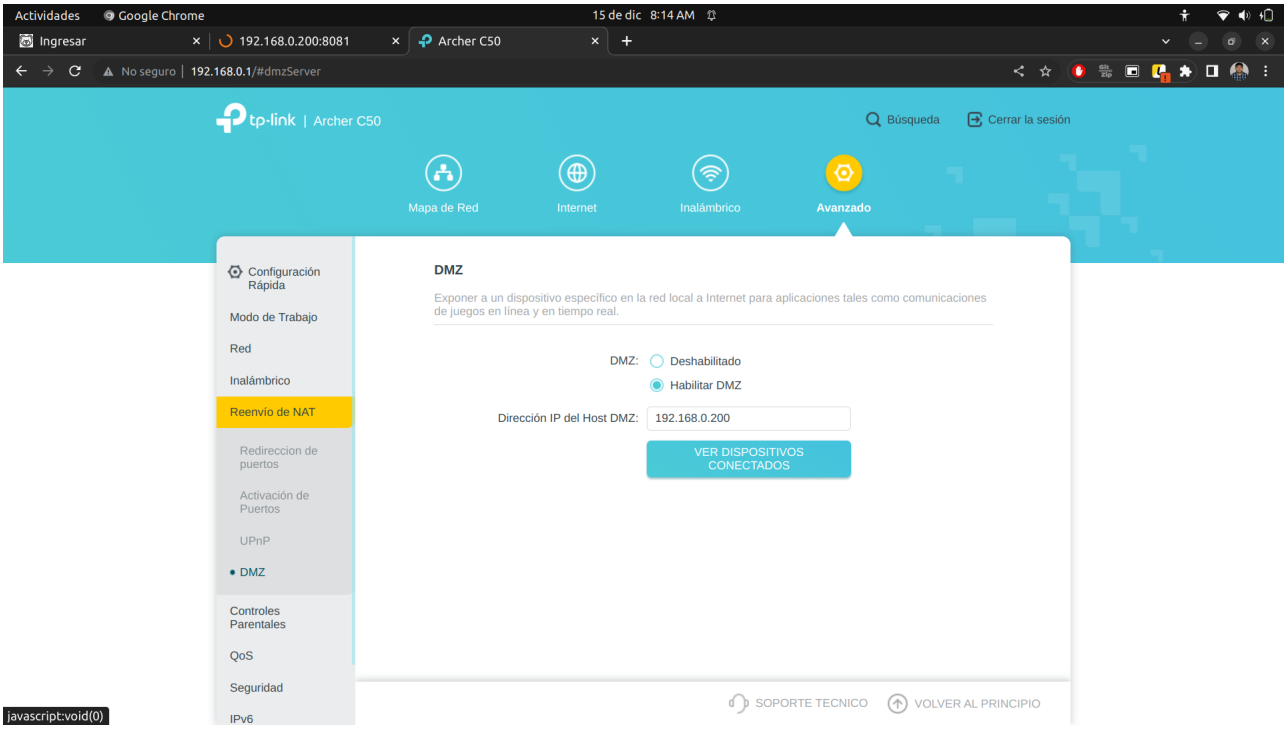


Figura 144: Configuración de DMZ.

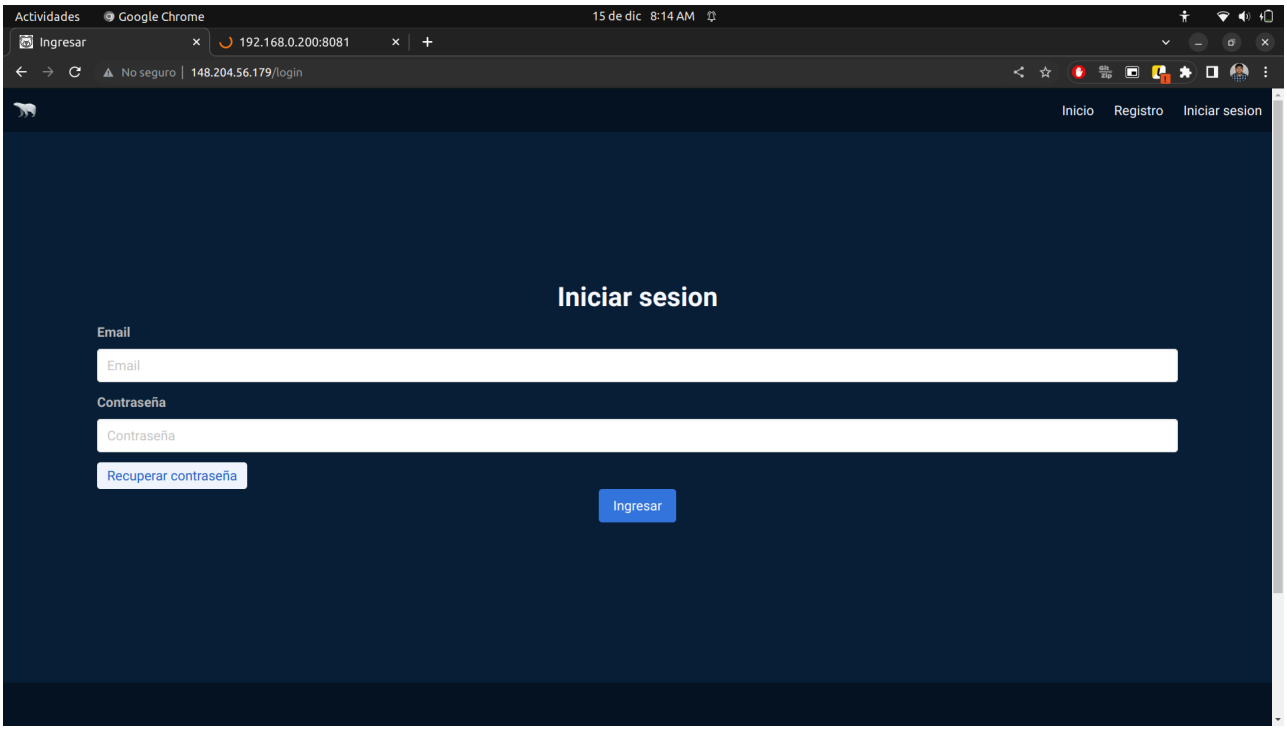


Figura 145: Servicio web en internet.

Hay que mencionar que el servicio sera visible si y solo si la Raspberry Pi 4 se encuentra encendida y conectada a la red.

8. Robot mensajero MemOso

Para poder realizar el envió de paquetes en el primer piso del edificio de gobierno de la Escuela Superior de Cómputo se empleó una arquitectura que la denominamos MemOso desarrollada en la Escuela Superior de Cómputo (ESCOM) bajo la asesoría del Dr. Luz Noé Oliva Moreno y el Dr. Juárez Martínez Genaro (Figura 146), mencionar que el esqueleto base del robot fue desarrollado en la Ciudad de las ideas en la Unidad Profesional Interdisciplinaria de Ingeniería Campus Hidalgo (UPIIH) bajo la asesoría del Dr. Luz Noé Oliva Moreno este esqueleto para el robot tiene una dimensión aproximada de 56 cm de ancho por 56 cm de largo por 30 cm de altura, ademas de que cuanta con llantas de hule en la parte trasera y llantas de polipropileno giratorias en la parte delantera, la parte de la conexión física de las llantas con los motores se realizo con una impresora 3D y finalmente comentar que este robot hacia uso de un microcontrolador MSP430G23 el cual fue cambiado como veremos mas adelante por un *System On a Chip* (SoC) debido a las necesidades que se presentan.

Figura 146: Robot MemOso, parte frontal.

8.1. Primera versión del robot. (Robot viajero)

8.1.1. Circuito eléctrico del robot

Con la finalidad de hacer uso del Physarum Polycephalum se ensamblara un robot, dicho robot cuenta con un *System On a Chip* (SoC), una etapa de sensado de proximidad de objetos y hace uso de 2 motores paso a paso bipolares que le permitirán el desplazamiento hacia enfrente, atrás, derecha e izquierda.

El diagrama esquemático lo podemos ver en la Figura 147 adicionalmente en la Figura 148 podemos ver un diagrama pictórico del circuito con los nombres de los componentes y valores de resistencias y capacitores para el funcionamiento del robot para poder hacer aún más entendible en la figura 149 ponemos ver los GPIO Pinout de la tarjeta Raspberry Pi 4 modelo B.

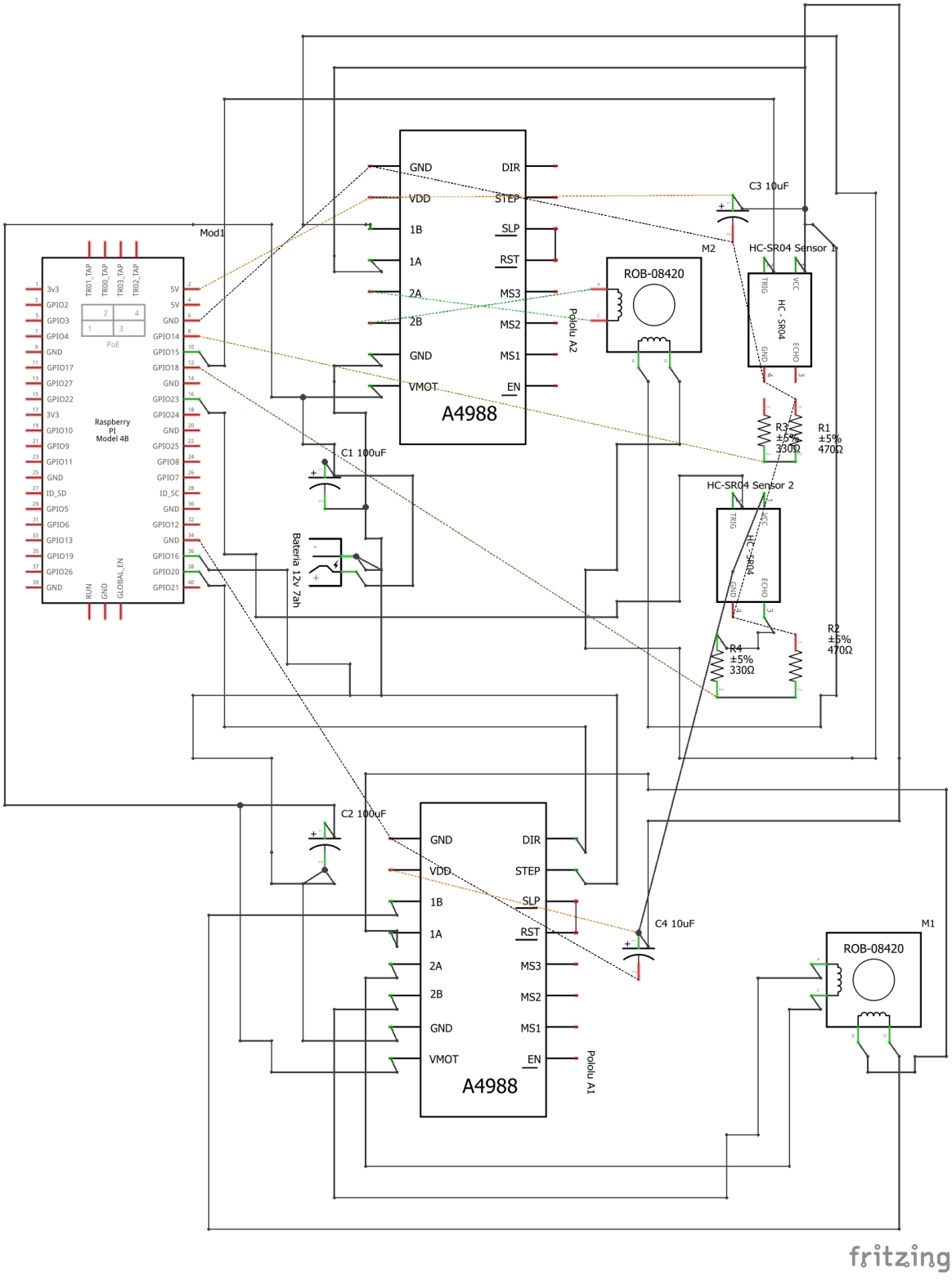


Figura 147: Diagrama esquemático del robot MemOso.

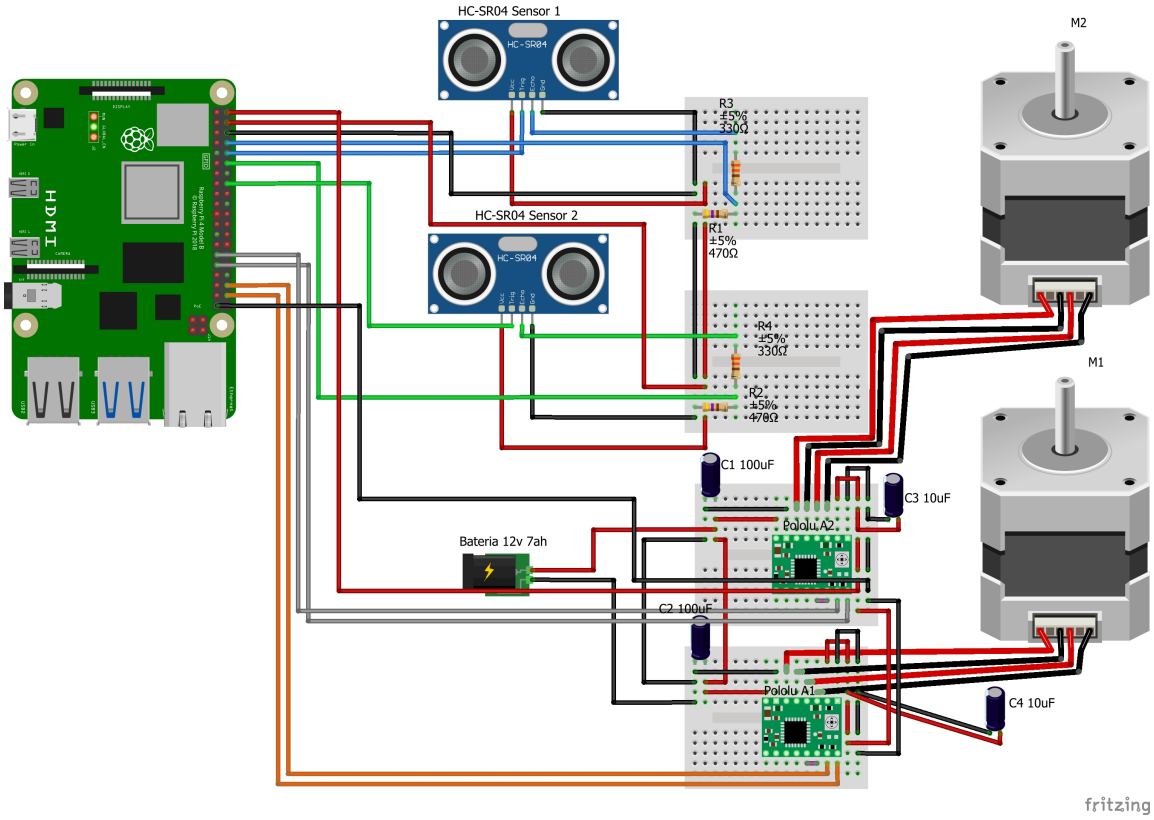


Figura 148: Diagrama pictórico del robot MemOso.

PIN	NAME		NAME	PIN
01	3.3V DC Power		5V DC Power	02
03	GPIO02 (SDA1,I²C)		5V DC Power	04
05	GPIO03 (SDL1,I²C)		Ground	06
07	GPIO04 (GPCLK0)		GPIO14 (TXD0, UART)	08
09	Ground		GPIO15 (RXD0, UART)	10
11	GPIO17		GPIO18(PWM0)	12
13	GPIO27		Ground	14
15	GPIO22		GPIO23	16
17	3.3V DC Power		GPIO24	18
19	GPIO10 (SP10_MOSI)		Ground	20
21	GPIO09 (SP10_MISO)		GPIO25	22
23	GPIO11 (SP10_CLK)		GPIO08 (SPI0_CE0_N)	24
25	Ground		GPIO07 (SPI0_CE1_N)	26
27	GPIO00 (SDA0, I²C)		GPIO07 (SCL0, I²C)	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12 (PWM0)	32
33	GPIO13 (PWM1)		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Figura 149: GPIO Pinout de la tarjeta Raspberry Pi 4 [1]

[1] Khan, B. A. (s. f.). Raspberry Pi 4 GPIO Pinout. https://linuxhint.com/gpio-pinout-raspberry-pi/.

8.1.2. Descripción de los componentes

8.1.2.1. System On a Chip

Figura 150: Raspberry Pi 4 [2]

Esta parte es la más importante ya que sera el encargado de realizar todo los cálculos, controlar todo lo que el robot utilice y otorgar servicios a usuarios, para este caso se utiliza una tarjeta Raspberry Pi 4 B de 4GB de RAM, esto debido a las grandes posibilidades que nos da usar un SoC ya que prácticamente es una computadora personal, ademas de que por medio de sus GPIO's nos permite comunicarnos y usar componentes digitales, ademas de que se tienen GPIO's exclusivas para alimentación y tierra de los componentes, pero sobre todo nos permite hacer desarrollo en cualquier tipo de lenguaje soportado por GNU/Linux, ya que este es la base de su Sistema Operativo.

8.1.2.2. Movimiento (motores)

Debido a la carga que debe de llevar el robot se opto por usar dos motores paso a paso y bipolares, esto ultimo útil para poder retroceder, para ello se opto por la familia NEMA 17 los cuales tienen un ángulo de paso de 1.8º (200 pasos por vuelta) y cada bobinado es de 1.7A, capaz de desarrollar un torque de hasta 4Kg/cm.

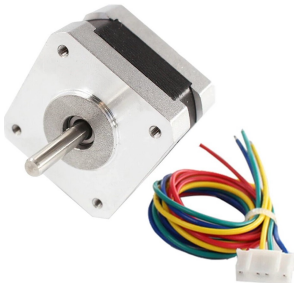


Figura 151: Motor NEMA 17 [3]

[2] 330ohms. (s. f.). Raspberry Pi 4 modelo B - 4GB. https://www.330ohms.com/es-us/products/raspberry-pi-4-modelo-b-4gb.

8.1.2.3. Controlador para los motores

Este motor puede ser utilizado con los controladores A4988 “Pololu” o DRV8825 la propuesta es usar el controlador A4988 ya que resulta ideal para manejar motores a pasos bipolares de hasta 2 amperes por bobina de una manera sencilla mediante los pines digitales, siendo su principal aplicación maquinas CNC, impresoras 3D y robots.

Figura 152: Controlador A4988 “Pololu” [4]

8.1.2.4. Alimentación

Para el funcionamiento del robot, la energía que debe de suministrarse tanto al SoC como para el funcionamiento de los motores es de 5v y 12v respectivamente, por lo que el uso de una batería ácido-plomo de 12v 7Ah es una de las mejores opciones para el funcionamiento del robot ademas de que no son tan pesadas comparando otras opciones.

Figura 153: Batería plomo 12v 7Ah [5]

[3]Geek Factory. (2022, 15 diciembre). Motor a pasos NEMA 17 17HS4401. https://www.geekfactory.mx/tienda/robotica/actuadores/motor-a-pasos-nema-17-17hs4401/

[4]Geek Factory. (2022, diciembre 9). A4988 Driver para motor a pasos. https://www.geekfactory.mx/tienda/modulos/motores/a4988-driver-para-motor-a-pasos/

8.1.2.5. Sensores de distancia

Para que el robot pueda detectar un posible obstáculo en la ruta que este siguiendo se hace uso de 2 sensores ultrasónico HC-SR04 con la capacidad de medir desde los 2cm hasta los 450cm de distancia con un angulo optimo de visión de 15° y sobre todo con una precisión de +- 3mm.

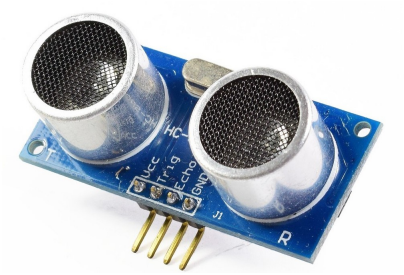


Figura 154: Sensor ultrasónico HC-SR04 [6]

[5] Batterie Yuasa plomb 12v 7ah - NP7-12 -. (s. f.). https://www.vlad.fr/es/estacionaria/971-bater3fa-12v-7ah-plomo-28151x65x97-529-yuasa.html

[6] Sensor Ultrasonico Hc-sr04 Arduino Sensor De Nivel Sr04. (s. f.). MercadoLibre. https://articulo.mercadolibre.com.mx/MLM-609041149-sensor-ultrasonico-hc-sr04-arduino-sensor-de-nivel-sr04-_JM

8.2. Segunda versión del robot. (Robot viajero)

8.2.1. Problemas encontrados con la primera versión y nuevos requerimientos

Después de hacer pruebas con la primera versión y considerando los consejos proporcionados en la primera evaluación de este trabajo por parte de los sinodales así como por parte de los directores nos dimos cuenta de un gran problema y es que los motores paso a paso NEMA 17, así como sus controladores y la forma de administrar energía a los circuitos no eran las mejores opciones, ya que existía la posibilidad de no cumplir con uno de los requisitos más importantes del sistema que es de mover paquetes menores a 5 kilogramos, adicionalmente a lo anterior mencionado nos planteamos el uso de un giroscopio esto debido a que surgió una preocupación al momento de que el robot diera vueltas en el edificio de gobierno y el uso de este nos puede indicar que tan desviado o recto se encuentra el robot después de los movimientos que este hace. De hecho en el transcurso de elaboración de esta segunda versión del robot nos dimos cuenta de que al implementar los nuevos motores y soldar los nuevos ejes adecuados para estos empezamos a obtener una desviación mayor a la esperada en el movimiento de las llantas por lo que uso de un giroscopio se volvió una necesidad.

A continuación, dedicaremos una sección para hablar de los diseños de alto y bajo nivel para dar soluciones a los problemas que encontramos en la primera versión del robot MemOso, así como implementar estos para así dar paso a una segunda versión del robot el cual tiene como objetivo usar el simulador 2DPyCAM(10) para la generación de rutas con el fin de poder moverse en el primer piso del edificio de gobierno de la ESCOM.

8.2.2. Diseños alto y bajo nivel e implementación en el robot MemOso

Solucionando primeramente el tema de los motores nos dimos cuenta de que los motores NEMA 17 no nos darían la suficiente potencia para poder mover un paquete de 5 kilogramos más lo que pesa el robot, así como sus componentes por lo que tomando en cuenta el

blog desarrollado por Radek Jarema con titulo "10 Steps to Choosing the Right Motors for Your Robotic Project"[15] en este blog Radek nos da 10 pasos para elegir el mejor motor para nuestro proyecto, para empezar necesitamos obtener algunos datos iniciales necesarios como lo son el peso del robot, el peso máximo de carga, la velocidad nominal, también el tiempo de aceleración y finalmente la inclinación máximo. En nuestro caso esos datos los podemos ver en la tabla 29:

Peso del robot	$m_R = 8kg$
Peso de carga máxima	$m_L = 5kg$
Velocidad nominal	$v_N = 30cm/s$
Tiempo de aceleración	$t_A = 0.5s$
Inclinación máxima	$K = 5\%$

Tabla 29: Datos iniciales para elegir motor. Creación propia.

Una vez teniendo los datos de la tabla 29 calculemos entonces la velocidad de rotación de la rueda:

$$N_T = \frac{60 v_N}{\pi D_r}$$

$$N_T = \frac{60}{\pi} * \frac{0.3}{0.127}$$

$$N_T \approx \frac{18}{0.3989} RPM$$

$$N_T \approx 45.12 RPM \approx 45 RPM$$

Donde:

$v_N = 0.3m/s$
$D_r = 5in \approx 0.127m$

Ahora procedemos a calcular la tracción que necesitamos en el motor o también conocido como el torque del motor, en este caso omitiremos la fricción que pueda haber con el piso. El empuje que necesitamos para mover el robot en el primer piso del edificio de gobierno

de la ESCOM se calcula de la siguiente manera:

$$F_T = gk(m_R + m_L)$$

$$F_T \approx 9.81 * 0.05(8 + 5)$$

$$F_T \approx 6.3765N$$

$$F_T \approx 6.38N$$

Donde:

$g \approx 9.81 \frac{m}{s^2}$	- Gravedad
$K = 0.05$	- Inclinación máxima
$m_R = 8kg$	- Peso del robot
$m_L = 5kg$	- Peso de carga máxima

Ahora calculemos la potencia mecánica que necesitamos para mover el robot con la velocidad nominal, para ello lo haremos de la siguiente manera:

$$P_U = F_T + v_N$$

$$P_U = 6.38 * 0.3$$

$$P_U \approx 1.914W$$

$$P_U \approx 1.91W$$

Donde:

$v_N = 0.3m/s$
$F_T = 6.38N$

Esta potencia mecánica calculada es por todo el robot, por lo que al usar dos motores tenemos que cada uno nos tiene que proporcionar 0.955 W. El torque correspondiente

para la tracción de cada una de las dos ruedas viene dada de la siguiente manera:

$$T_T = \frac{1}{2} * \frac{D_r}{2} F_T$$

$$T_T = 0.5 * 0.5 * 0.127 * 6.38$$

$$T_T \approx 0.203 Nm$$

Donde:

$D_r = 5in \approx 0.127m$
$F_T = 6.38N$

Finalmente y recapitulando todo lo calculado tenemos que los requerimientos para la tracción por cada motor son los siguientes:

Velocidad nominal de rotación:	$N_T = 45RPM$
Torque:	$T_T = 0.203Nm$
Potencia:	$P_T = 0.955W$

Tabla 30: Requerimientos para elegir motor. Creación propia.

Una vez realizado lo anterior llegamos a la conclusión de que los mejores motores que podemos usar los motores DC RS-550 esto tomando como base la figura 155 en donde podemos ver una gráfica que nos proporciona toda la información de eficiencia de un motor RS-550 pero con la característica de entregar 19300 RPM, como mencionamos un poco más adelante este tipo de motores pueden entregar diferentes cantidades de RPM, sin embargo, esta gráfica nos sirve de guía para ver que la elección de motores es la correcta ademas de que abre la posibilidad de usar el robot en otros espacios o soportar pesos mayores, pero eso es algo que no nos corresponde explorar.

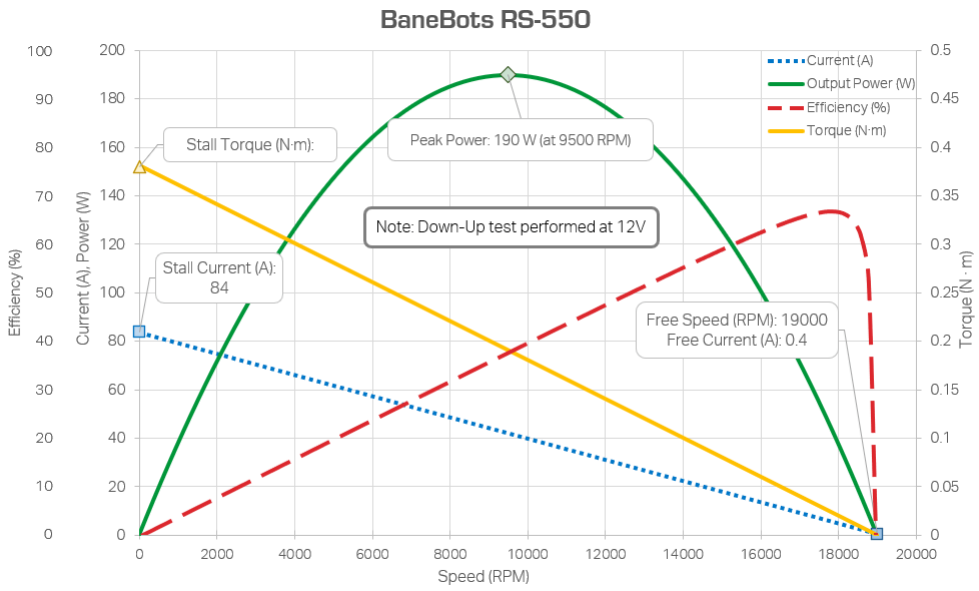


Figura 155: Gráfica de eficiencia de un motor RS-550 con 19300 RPM de salida [7]

Este tipos de motores son muy comúnmente encontrados en los famosos coches eléctricos para niños, este tipo de motores funcionan con 12v este tipo de motores tiene variación en la cantidad de RPM que nos puede proporcionar en nuestro caso y debido a los cálculos obtenidos se eligió la versión que proporciona 15000RPM. Este motor viene usualmente acompañado con una caja de cambios que básicamente se compone por un sistema de engranajes en donde la salida se nos proporciona un mayor torque para soportar mover grandes cargas, el interior de la caja de cambios, así como el motor junto con la caja de cambios lo podemos ver en las Figuras 156 y 157 respectivamente.

[7] VEX Robotics Motor Data - Banebots RS-550. (s. f.). https://motors.vex.com/other-motors/bb-550

Figura 156: Engranajes que componen la caja de cambios para el motor RS-550. [8]

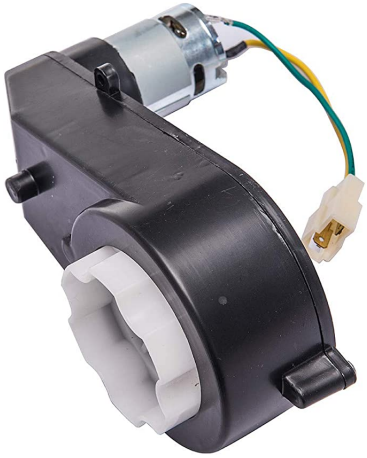


Figura 157: Caja de cambios con el motor RS-550. [9]

Ahora nos encontramos con una problemática los módulos A4988 que son los controladores que tenemos para la primera versión del robot nos resultan inútiles principalmente por el hecho de que estos están diseñados exclusivamente para funcionar con motores paso a paso, ahora necesitamos hacer uso de otro tipo de controladores, por defecto sabemos que necesitamos hacer uso de un puente H y que además de que el controlador que lleguemos a usar nos sirva para comunicarnos con algún microcontrolador o en nuestro caso un siste-

[8] Motor de coche eléctrico para niños, caja de cambios con alto par y alta velocidad, 6V, 12V, 24V, RS550, RS570, RS555. (s. f.). AliExpress. https://es.aliexpress.com/i/1005002193536143.html

[9] Caja de cambios de 550 40000 RPM con motor de 12 V, Amazon.com.mx. (s. f.). https://cutt.ly/82yP6Lg

ma en chip, podríamos llegar a pensar que el controlador L298N que es el más típico para controlar motores DC sería buena opción, sin embargo, este controlador solo soporta una corriente máxima de 3A y aunque puede soportar una tención de 3v a 35v no nos resulta útil debido a que los motores DC RS-550 tienen un pico de corriente de 35A, es decir, poco más de 10 veces más la corriente máxima soportada por el controlador L298N, tomando en cuenta todo lo anterior mencionado se encontró un controlador diseñado para su uso con Arduino, pero cualquier otro microcontrolador podría hacer uso de este, el controlador es un BTS74960 el cual está diseñado para controlar motores de corriente continua de alta potencia, siendo capaz de proporcionar hasta 43A de corriente a una tensión de alimentación de 5v hasta 27v cumpliendo así las características de funcionamiento de nuestros motores, adicionalmente a ello la lógica de este controlador funciona con voltajes de 3.3v a 5v, además de que admite realizar el control de la velocidad del motor mediante PWM con una frecuencia máxima de 25 kHz, es decir, este controlador es ideal para nuestro proyecto.

Una vez solucionado el problema de los motores vayamos a atacar el problema de la suministración de energía del sistema, inicialmente contábamos con una sola batería de 12v 7Ah, sin embargo, no es lo más óptimo ni lo mejor, ya que suministrar energía con una sola batería a los componentes digitales, el sistema en chip y a los motores que aunque sea posible el consumo de tanto los motores como el sistema en chip es alto, así que separar estos sistemas es lo mejor, por lo que se decidió agregar una batería extra con las mismas características que la batería de la primera versión separando así los circuitos, por una parte tenemos los motores así como los controladores de estos y del otro lado tenemos al sistema en chip junto con los circuitos digitales.

Antes de empezar a ver el nuevo circuito eléctrico del robot comentaremos un paso que se tuvo que hacer para que el robot funcionara adecuadamente, como podemos ver en la figura 158 nuestra caja de cambios tiene una forma en la salida que nos proporciona todo el movimiento del motor, podemos decir que la forma es un círculo con 6 semicírculos cortándolo, dando como si fuera un hexágono con vértices curvos. Esto nos propone un

nuevo reto y es que necesitamos crear una pieza mecánica para poder hacer uso de la caja de cambios y comunicar la salida de la caja de cambios a una llanta del robot para así aprovechar toda la potencia que nos puede proporcionar la caja de cambios. Para

Figura 158: Salida de la caja de cambios. [10]

solucionar el reto anteriormente planteado nos vimos en la necesidad de diseñar una pieza 3D e imprimirla dos veces porque hacemos uso de dos motores, para diseñar esta pieza 3D hicimos uso del programa Blender el cual nos permitió realizar el diseño de la pieza necesaria, en la figura 159 podemos ver las dimensiones que tiene nuestra pieza, mencionar que la medida está en milímetros. Finalmente, necesitamos hacer realidad esta pieza, por lo que fue necesario imprimirla en 3D, por lo que necesitamos elegir el tipo de material con la que imprimiremos la pieza, existen diversos materiales para este fin, sin embargo, los materiales más usados son el plástico PLA y el plástico ABS, a continuación mencionaremos unas características generales de estos.

- *Plástico PLA*: Es un plástico biodegradable ideal para piezas con lados finos y de alto detalle, ideal para piezas pequeñas y grandes, es más estable y fácil de imprimir.
- *Plástico ABS*: Es un plástico muy común y resistente, este material es usado para las famosas piezas LEGO, es ideal para la impresión de piezas mecánicas y que sean de uso rudo ya que se puede lijar con una lija de madera y pulir aplicando ligeramente un trapo con acetona, ademas de que también se puede taladrar o pintar.

[10] Motor de coche eléctrico para niños, caja de cambios con alto par y alta velocidad, 6V, 12V, 24V, RS550, RS570, RS555. (s. f.). AliExpress. https://es.aliexpress.com/i/1005002193536143.html

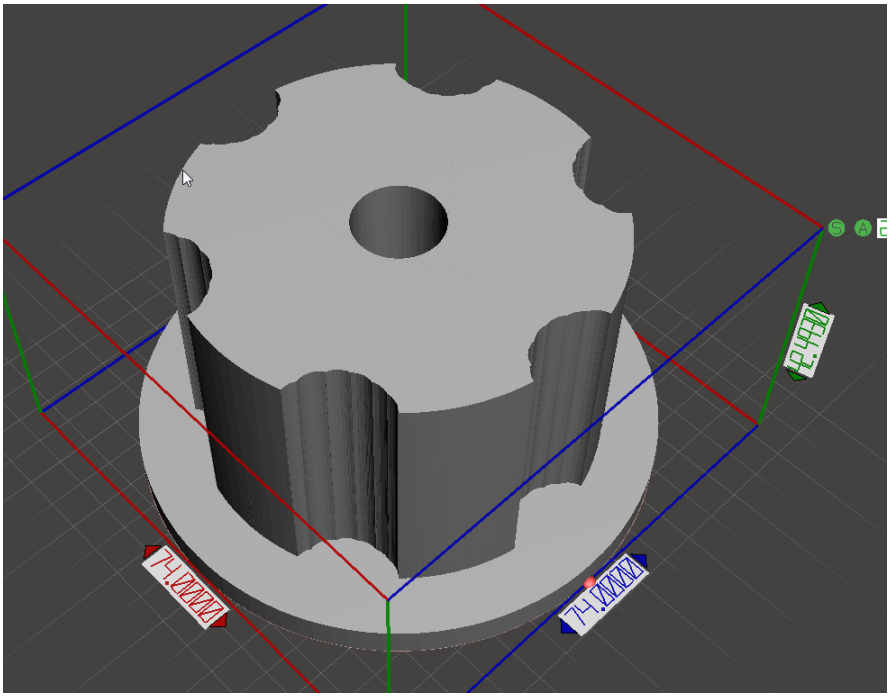


Figura 159: Medidas de la pieza 3D creada.

Con base en lo anterior hemos elegido imprimir la pieza con el plástico ABS debido a la gran resistencia de este material, ya que en el entorno en donde se usará nos importa que el material tenga una gran dureza, las piezas ya impresas la podemos ver en la figura 160 y en la figura 161 podemos ver como esta embona perfectamente con nuestra caja de cambios, finalmente en la figura 162 y 164 podemos ver la conexión del motor con la caja de cambios, la pieza 3D y la llanta.

Figura 160: Piezas 3D impresas.

Figura 161: Pieza 3D con la caja de cambios unida.

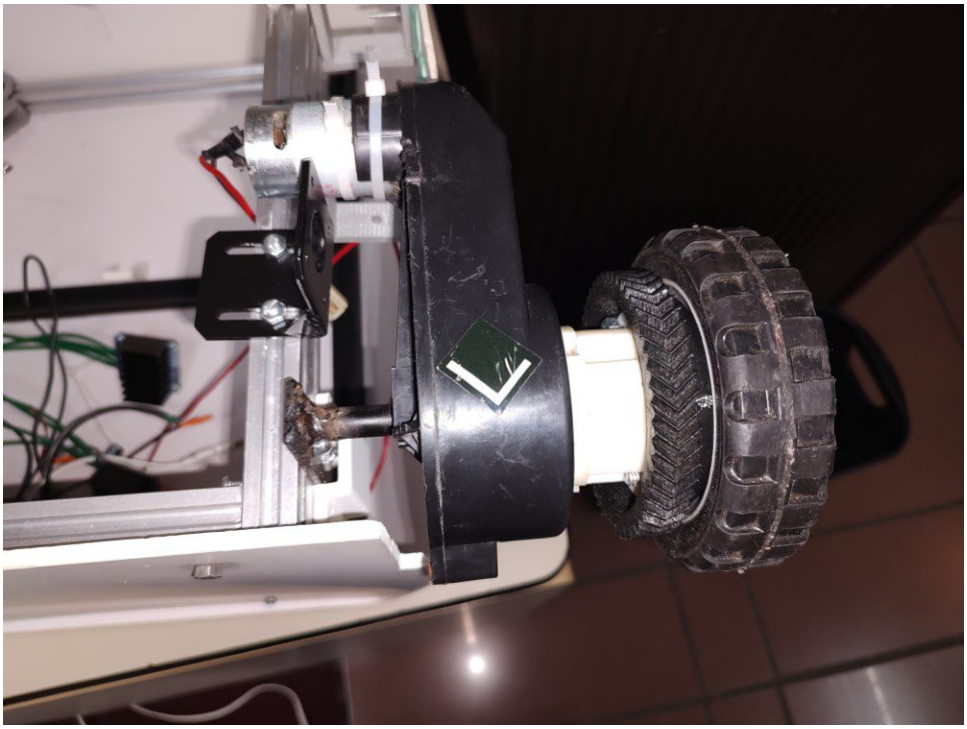


Figura 162: Conexión caja de cambios, llanta y pieza 3D (lado izquierdo).

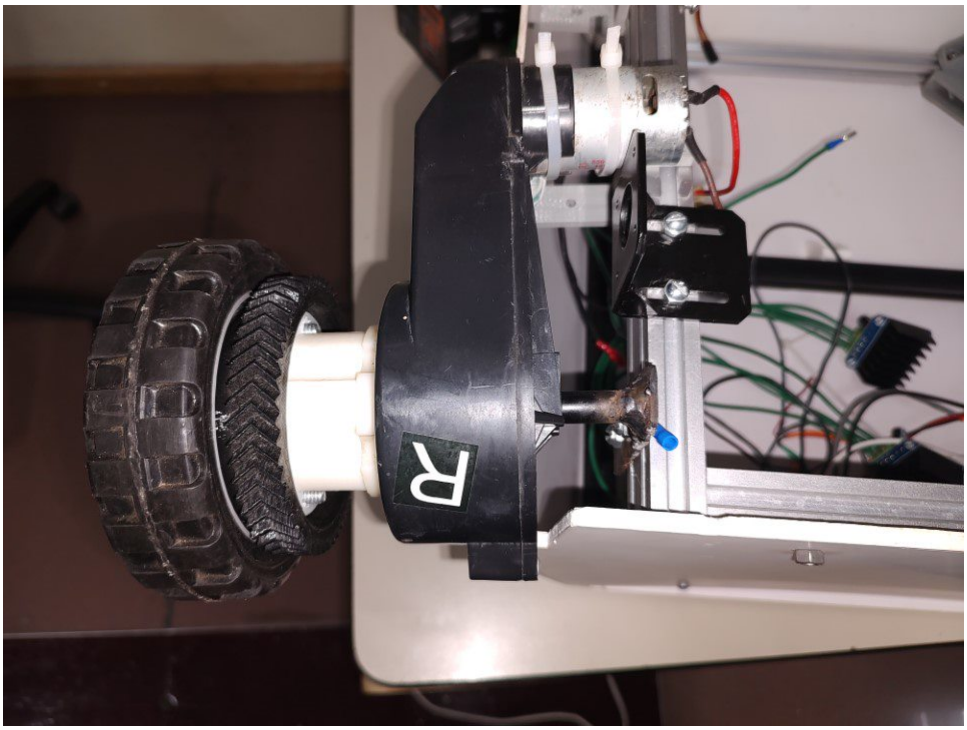


Figura 163: Conexión caja de cambios, llanta y pieza 3D (lado derecho).

Una vez solucionado el problema de los motores vayamos a la implementación del giroscopio para poder tener una mayor exactitud en el movimiento del robot. El módulo

sensor MPU6050 es un dispositivo completo de seguimiento de movimiento de 6 ejes, ya que combina un giroscopio de 3 ejes, un acelerómetro de 3 ejes y un procesador de movimiento digital, además tiene una función adicional de sensor de temperatura en el chip. Este módulo se comunica a través de la interfaz de bus I2C para comunicarse con los microcontroladores.

Tiene un bus I2C auxiliar para comunicarse con otros dispositivos sensores como magnetómetro de 3 ejes, sensor de presión, etc. Si el magnetómetro de 3 ejes está conectado al bus I2C auxiliar, entonces el MPU6050 puede proporcionar una salida completa de 9 ejes.

A continuación veamos con más detalle al sensor que nos importa que es el del giroscopio. El MPU6050 consiste en un giroscopio de 3 ejes con tecnología de sistemas microelectromecánicos (por sus siglas en ingles MEMS). Se utiliza para detectar la velocidad de rotación a lo largo de los ejes X, Y, Z, también conocidos por los ejes de aviación Pitch, Roll y Yaw respectivamente, como se muestra en la siguiente figura.

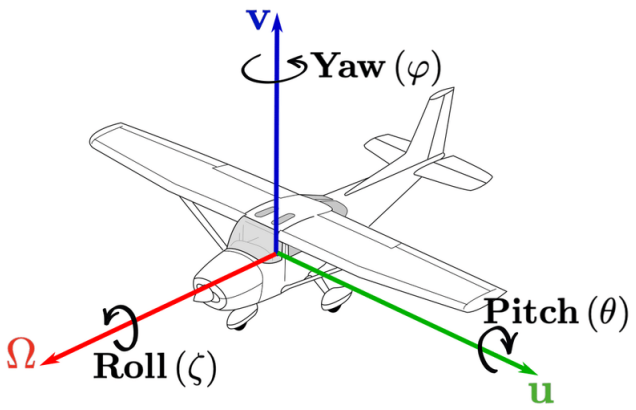


Figura 164: Ejes de rotación. [11]

- Cuando los giroscopios giran sobre cualquiera de los ejes de detección, el efecto Coriolis provoca una vibración que es detectada por un MEM dentro del MPU6050.

[11] Degond, P. (2021, 22 enero). Bulk topological states in a new collective dynamics model. arXiv.org. https://arxiv.org/abs/2101.10864

- La señal resultante se amplifica, demodula y filtra para producir un voltaje proporcional a la velocidad angular.
- Este voltaje se digitaliza usando ADC de 16 bits para muestrear cada eje.
- El rango de escala completa de salida es +/- 250, +/- 500, +/- 1000, +/- 2000.
- Mide la velocidad angular a lo largo de cada eje en grados por unidad de segundo.

Para poder procesar los datos de los sensores se hace uso de un procesador de movimiento digital (DMP) integrado, este se utiliza para calcular algoritmos de procesamiento de movimiento, para esto se toman datos del giroscopio, el acelerómetro y un sensor adicional de terceros, como un magnetómetro, y procesa los datos, proporciona datos de movimiento como balanceo, cabeceo, ángulos de guiñada, sentido de paisaje y retrato, etc. Minimiza los procesos del host en la computación de datos de movimiento. Los datos resultantes se pueden leer desde los registros del DMP.

Ahora veamos como hacer los cálculos finales para poder hacer uso de los datos generados por lo sensores pero en especifico del giroscopio. Para esto hay que tener en cuenta que los datos del sensor de giroscopio y acelerómetro del módulo MPU6050 consisten en datos sin procesar de 16 bits en forma de complemento a 2. Los datos del sensor de temperatura del módulo MPU6050 consisten en datos de 16 bits (no en forma de complemento a 2).

Ahora supongamos que hemos seleccionado un rango de escala completa del giroscopio de +/- 250 °/s con factor de escala de sensibilidad de 131 LSB (recuento)/°/s, después, para obtener datos sin procesar del sensor, primero debemos realizar el complemento de 2 en los datos del sensor del giroscopio. Después de obtener los datos sin procesar del sensor, podemos calcular la aceleración y la velocidad angular dividiendo los datos sin procesar del sensor por su factor de escala de sensibilidad de la siguiente manera:

Valores del giroscopio en °/s (grados por segundo):

- Velocidad angular a lo largo del eje X = (datos sin procesar del eje X del giroscopio/131) °/s.
- Velocidad angular a lo largo del eje Y = (datos brutos del eje Y del giroscopio/131) °/s.
- Velocidad angular a lo largo del eje Z = (datos brutos del eje Z del giroscopio/131) °/s.

8.2.3. Circuito eléctrico del robot

Una vez solucionado los problemas encontrados con la primera versión de nuestro robot MemOso veamos nuestro nuevo diagrama esquemático el cual lo podemos ver en la Figura 165 adicionalmente en la Figura 166 podemos ver un diagrama pictórico del circuito con los nombres de los componentes y valores de resistencias para el funcionamiento del robot para poder hacer aún más entendible en la figura 149 ponemos ver los GPIO Pinout de la tarjeta Raspberry Pi 4 modelo B.

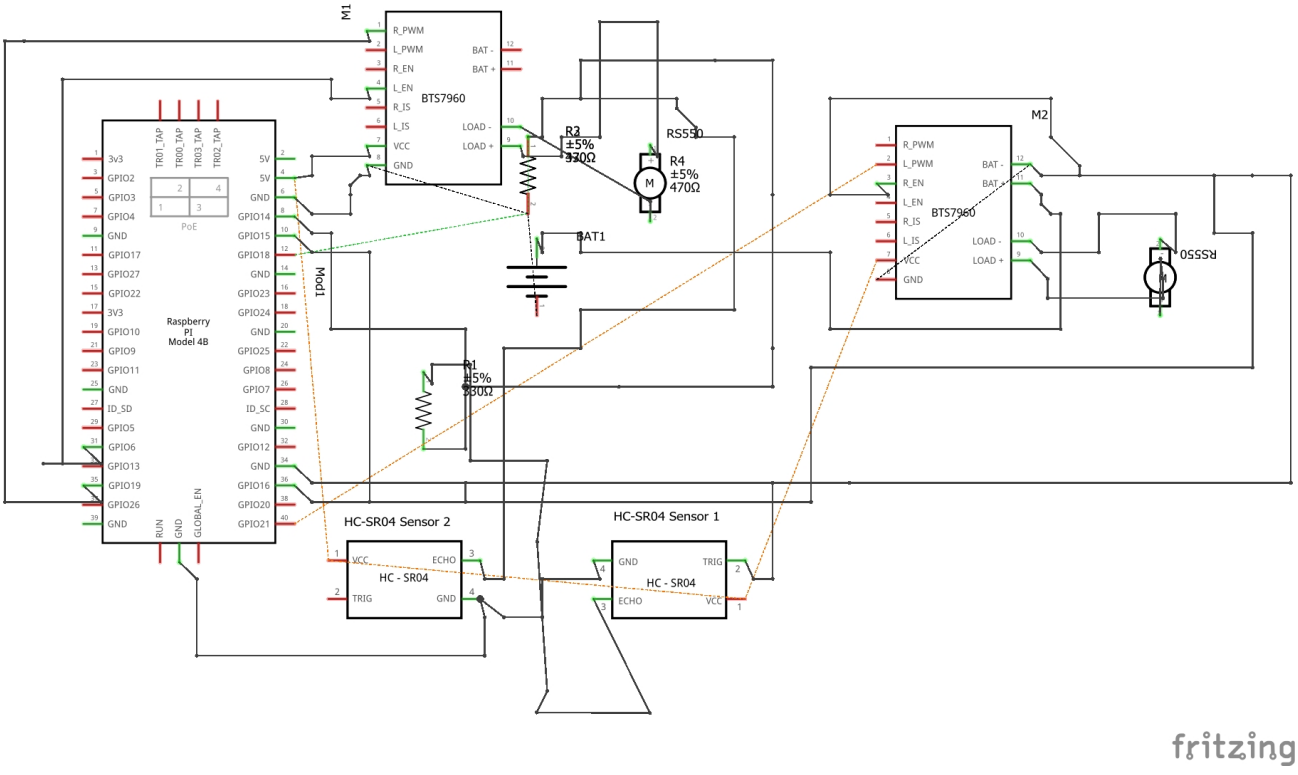


Figura 165: Diagrama esquemático del robot MemOso segunda versión.

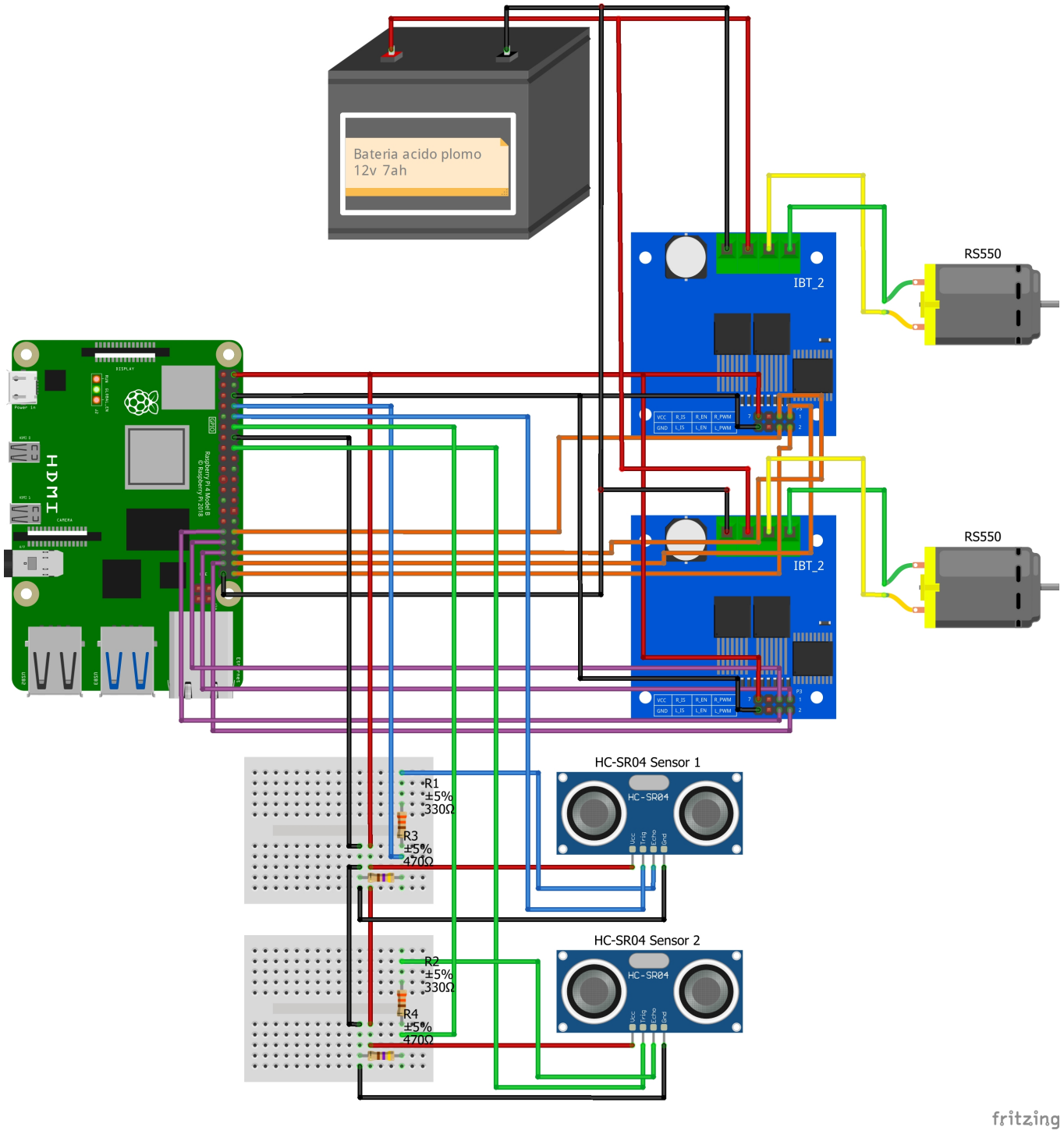


Figura 166: Diagrama pictórico del robot MemOso segunda versión.

8.2.4. Calibración de los sensores del robot

Ahora vayamos con la calibración de los sensores que usaremos en el robot para poder detectar algún obstáculo, esto se hace ya que los sensores no vienen calibrados con base a

las condiciones ambientales del lugar en donde se empleara, sin embargo, esto no quiere decir que, si no se calibran estos sensores arrojara información errónea, si no, datos con pequeños errores ya sea de mas un centímetro o de mas dos centímetros.

Como podemos ver en la figura 167 ese es nuestro pequeño entorno de calibración de los sensores, compuesto por el sensor a calibrar y un patrón el cual es un flexómetro, con todo esto y un objeto recto con el cual podamos realizar las medidas sera lo único que necesitemos.

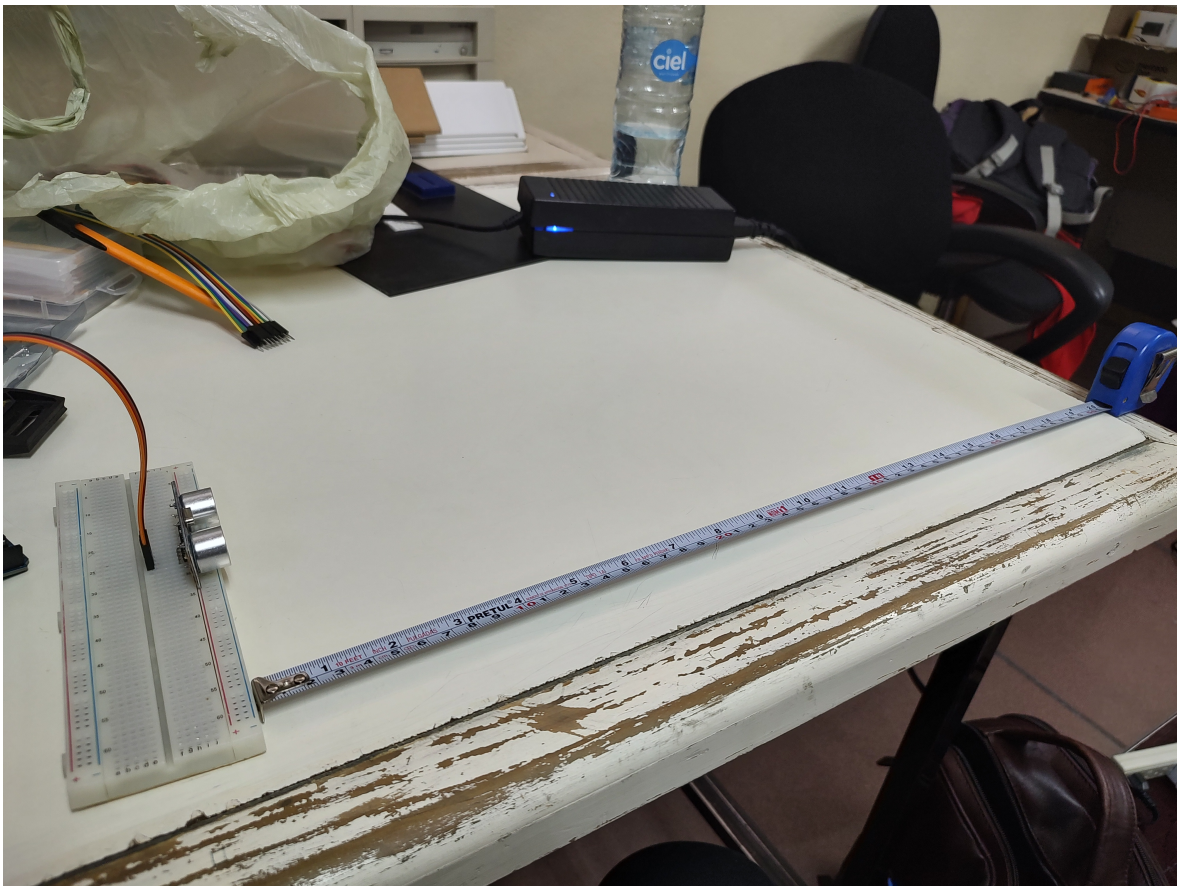


Figura 167: Entorno de calibración para los sensores.

Ahora como vemos en las figuras 168 y 169 vemos como se están haciendo las pruebas, en la primera vemos como tenemos nuestro objeto plano el cual es una tabla de madera delgada de color negro, ademas de ponerle un peso a la protoboard para que esta no pueda moverse y tener una cierta constante en las mediciones, finalmente en la segunda imagen vemos como vamos a notando todo en una hoja de datos para posteriormente hacer todos

los cálculos correspondientes.

Figura 168: Haciendo mediciones.

Figura 169: Guardando información obtenida en una hoja de datos.

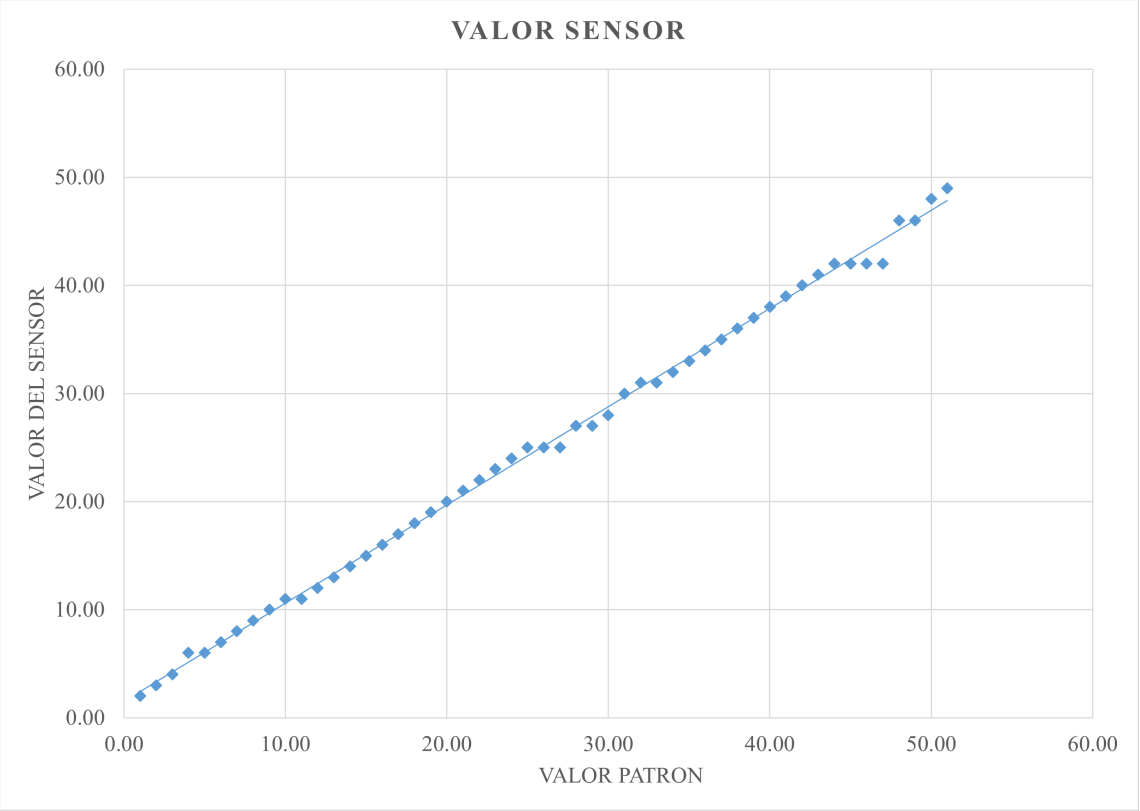


Figura 170: Gráfica de relación entre la medida del sensor y del patrón.

Para calibrar estos sensores es tan sencillo como comparar las distancias reales (del patrón) con las dadas por el módulo del sensor y hacer con ellas una regresión lineal. Lo que se pretende con esta regresión lineal es corregir el valor de la medida del módulo por otro que sea más próximo a la realidad. Para esto necesitamos al menos veinte medidas distintas, en nuestro caso hicimos 50 medidas. Como podemos ver en la figura 170 podemos ver que la relación entre la medida de distancia patrón (tomada con el flexómetro) y la medida dada por el módulo del sensor es una función lineal de la siguiente forma:

*medida del sonar = medida de la regla * pendiente + ordenada*

En donde la pendiente se calcula con de la siguiente manera:

$$\hat{\beta}_1 = \frac{\sum x \sum y - n \sum x\, y}{(\sum x)^2 - n \sum x^2}$$

Formula de la pendiente

Donde:

x - Valor de medida del patrón
y - Valor de medida obtenida del sensor
n - Numero de valores tomados

Y la ordenada de la siguiente manera:

$$\hat{\beta}_0 = \frac{\sum y - \hat{\beta}_1 \sum x}{n}$$

Formula de la ordenada

Todo lo anterior con base a la regresión lineal entre los pares de datos.

Tras obtener los valores solo hay que modificar el código. Se toma el valor que nos daba inicialmente y se modifica así:

valor corregido = (valor inicial – ordenada) / pendiente

Con esto ya se a calibrado un sensor, así que necesitamos repetir el proceso una vez mas ya que tenemos dos sensores.

8.2.5. Descripción de los nuevos componentes

8.2.5.1. Movimiento (motores)

Los motores RS-550 son motores DC muy utilizados en diseños de ingeniería debido a las características torque-velocidad que poseen con diferentes configuraciones eléctricas o mecánicas que puede variar la cantidad de RPM, estos motores son típicos de ver y encontrar en aparatos en donde se requiere tener una gran potencia como puede ser los taladros eléctricos, prensas y también en carros eléctricos de juguete para niños. El modelo elegido proporciona 15000RPM teniendo un voltaje de operación de 12v y con un amperaje de 0.8A para cuando no tiene carga o de 35A para cuando tiene carga, así como un torque de 331.2 mN-m, teniendo una constante de velocidad Kv = 1250 RPM/v y una constante de torque del motor de Kt = 7.64 N-m/A

Figura 171: Motor RS-550. [12]

8.2.5.2. Controlador para los motores

Como se menciona en un apartado anterior hay varios controladores para los motores DC, pero el controlador que estaremos usando para el proyecto es el BTS7960, estos son controladores para motores de corriente continua de alta potencia, capaces de proporcionar hasta 43A de corriente a una tensión de alimentación de entre 6 a 27V, ideal para la batería que estamos usando.

La lógica del controlador funciona con voltajes de 3.3V a 5V por lo cual es compatible con la mayoría de microprocesadores y evidentemente manejable con la Raspberry Pi. Además, admiten realizar el control de la velocidad del motor mediante PWM con una frecuencia máxima de 25 kHz, adicionalmente a ello el BTS7960 incorpora mecanismos de protección contra cortocircuito, sobre o infra voltaje, y sobre temperatura. También dispone de dos pines que permiten medir la corriente entregada por el controlador.

[12] Motor de coche eléctrico para niños, caja de cambios con alto par y alta velocidad, 6V, 12V, 24V, RS550, RS570, RS555. (s. f.). AliExpress. https://es.aliexpress.com/i/1005002193536143.html

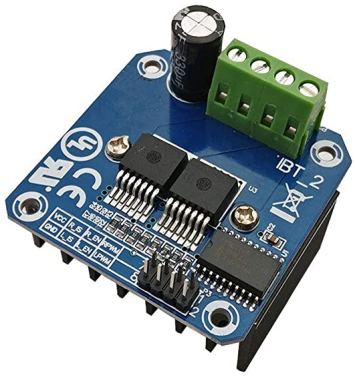


Figura 172: Controlador BTS7960. [13]

8.2.5.3. Alimentación

Como se comentó anteriormente usaremos dos baterías ácido-plomo una exclusivamente para el SoC y la otra para los controladores de los motores, recordaremos nuevamente que son baterías de 12v 7Ah.

Figura 173: Batería plomo 12v 7Ah. [14]

8.2.5.4. Giroscopio

Como se comentó anteriormente usaremos un giroscopio para mejorar el rendimiento de nuestro robot, tener un mayor control sobre el robot y sobre todo solucionar los problemas físicos que tiene nuestro robot.

[13] Modulo Puente H Bts7960 43a, Arduino, Pic. (s. f.). Meses sin intereses. https://articulo.mercadolibre.com.mx/MLM-1510163267-modulo-puente-h-bts7960-43a-arduino-pic-_JM?matt_tool=91188883

[14] Batterie Yuasa plomb 12v 7ah - NP7-12 -. (s. f.). https://www.vlad.fr/es/estacionaria/971-bater3fa-12v-7ah-plomo-28151x65x97-529-yuasa.html

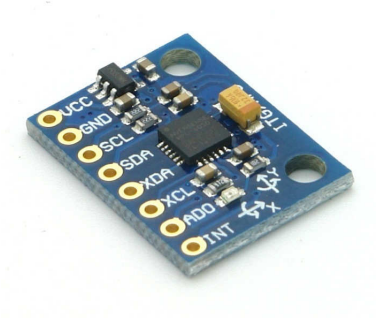


Figura 174: Giroscopio MPU6050 [15]

El módulo MPU-6050 tiene 8 pines:

- INT: pin de salida digital de interrupción.
- AD0: Pin LSB de dirección de esclavo I2C. Este es el bit 0 en la dirección esclava de 7 bits del dispositivo. Si está conectado a VCC, se lee como uno lógico y cambia la dirección del esclavo.
- XCL: pin de reloj serie auxiliar. Este pin se utiliza para conectar el pin SCL de otros sensores habilitados con interfaz I2C al MPU-6050.
- XDA: pin de datos seriales auxiliares. Este pin se utiliza para conectar el pin SDA de otros sensores habilitados con interfaz I2C al MPU-6050.
- SCL: pin de reloj serie. Conecte este pin al pin SCL de los microcontroladores.
- SDA: pin de datos en serie. Conecte este pin al pin SDA de los microcontroladores.
- GND: Pin de tierra
- VCC: pin de fuente de alimentación.

El módulo MPU-6050 tiene una dirección de esclavo (cuando AD0 = 0, es decir, no está conectado a Vcc) como:

[15] MPU6050 (Gyroscope + Accelerometer + Temperature) Sensor Module. (s. f.). ElectronicWings. https://www.electronicwings.com/sensors-modules/mpu6050-gyroscope-accelerometer-temperature-sensor-module

Dirección de escritura del esclavo (SLA+W): 0xD0

Dirección de lectura del esclavo (SLA+R): 0xD1

El objetivo de esta segunda versión es que el robot sea capaz de generar una ruta entre dos puntos mediante nuestro autómata, seguir la ruta generada, detectar un posible obstáculo al momento de seguir la ruta y en caso de que se encontrara con un obstáculo ser capaz de detener el seguimiento de la ruta, generar una nueva ruta añadiendo el obstáculo encontrado y seguir a la nueva ruta generada hasta llegar al punto inicial.

Hay que mencionar que en esta segunda versión la definición del punto inicial y final vienen seleccionados por defecto, pero es modificable a través del código y tomando como base un mapa a escala del primer piso del edificio de gobierno de la ESCOM en donde ya tenemos por defecto las coordenadas de diferentes áreas que laboran en el piso.

8.2.6. Manejo de mapa base y rutas del Robot

El robot tiene la capacidad de generar rutas tomando como base un archivo que posee el mapa del edificio de gobierno de la Escuela Superior de Cómputo el cual funciona como un arreglo de 2 dimensiones siendo conformado de 80 elementos en ambas dimensiones, mencionar que al ser un mapa a escala se adaptó las medidas al tamaño del robot el cual como mencionamos anteriormente es de 56 cm de largo y ancho ignorando la altura, el azulejo que tiene el piso en donde el robot trabajara posee unas medidas de 30 cm de largo y ancho, por lo que, lo más sencillo es tomar 4 azulejos para poder obtener prácticamente la misma medida que tiene el robot, de forma visual lo podemos ver en la figura 175. Esto nos simplifica bastante el funcionamiento del robot ya que podemos trabajar con coordenadas fijas, así como la orientación preferente a tomar para que el robot se pueda mover para las distintas áreas de trabajo que tiene el edificio como lo pude ser dirección, subdirección académica, becas entre otras.

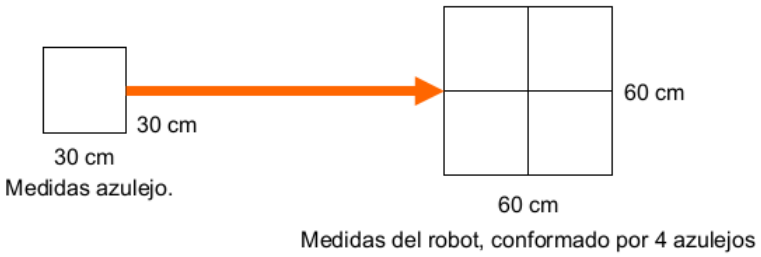


Figura 175: Conversión de azulejo a medida robot.

El manejo del punto inicial y final se hacen de manera lógica, es decir, solo tomamos la información del mapa base y lo modificamos con un nutriente no encontrado y un punto inicial (esto no es directo en el archivo de nuestro mapa base), al ser calculada la ruta se guarda el autómata completo en un archivo con extensión csv y será leído para que el robot pueda seguir la ruta, mientras esto sucede el robot ira guardando su ubicación en coordenadas, esto con el objetivo de que como podemos ver en el diagrama de flujo del robot figura 176 en caso de encontrar un obstáculo se pueda recalcular la ruta con facilidad, lo anterior posee el siguiente formato y es guardado en un archivo de texto:

```
{
    "fin_x": 56,
    "fin_y": 61,
    "orientacion": "E",
    "pos_x": 56,
    "pos_y": 61
}
```

Como podemos ver este archivo contiene las coordenadas finales, la orientación del robot que tiene en ese momento, así como las coordenadas actuales del robot, en este ejemplo el robot ya llego al lugar objetivo, mencionar que tanto el archivo que contiene la ruta, así como las coordenadas de la ubicación son sobrescritos para evitar la saturación de nuestro SoC.

Finalmente mencionamos que en caso de encontrar un obstáculo se toma como base las ultimas coordenadas guardadas del robot, se calcula la distancia del objeto, se aproxima las coordenadas en donde podría estar el obstáculo y se coloca un repelente en el mapa base y vuelve a ver un calculo de ruta como se menciona con anterioridad.

8.2.7. Comportamiento del Robot

Para el presente apartado nos basamos en el funcionamiento de que el robot deberá de ser ejecutado un archivo principal en donde se tengan cargados los nombres de los lugares de inicio y fin en donde se realizará la entrega del paquete, después el robot se encargará de realizar el cálculo de la ruta por medio del autómata y procederá a moverse para ello tendremos 3 tipos de movimientos.

1. Rotar a la izquierda 90 grados (orientar a la izquierda).
2. Rotar a la derecha 90 grados (orientar a la derecha).
3. Avanzar (mover hacia adelante).
4. Retroceder (mover hacia atrás).

8.2.7.1. Funcionamiento del movimiento del robot antes de implementar el giroscopio

Por el momento se maneja una cierta distancia sobre el avance, donde el motor se activará n segundos para realizar dicha acción. Los pasos que sigue el robot para poder seguir la ruta generada hasta llegar al punto final son los siguientes y pueden ser observados en la Figura 176.

1. Se ejecuta el archivo principal del robot donde se tiene previamente escrito el nombre del lugar a donde enviará el paquete y el nombre del lugar donde se encuentra ubicado actualmente.
2. El robot espera la generación de la ruta a seguir donde una vez que haya sida calculada entra al seguimiento de la ruta.
3. El robot irá leyendo la ruta mientras se mueve e ira usando los sensores para poder detectar algún obstáculo en el seguimiento de la ruta y también ira almacenando su ubicación, la orientación y el punto final a llegar por si hubiera alguna interrupción

el seguimiento de la ruta será interrumpido y se tendrá que recalcular la ruta con base en la ubicación en donde está el robot y el punto final.

4. Al llegar al final se considerarán las coordenadas finales como las nuevas iniciales del robot para la siguiente ruta a seguir.

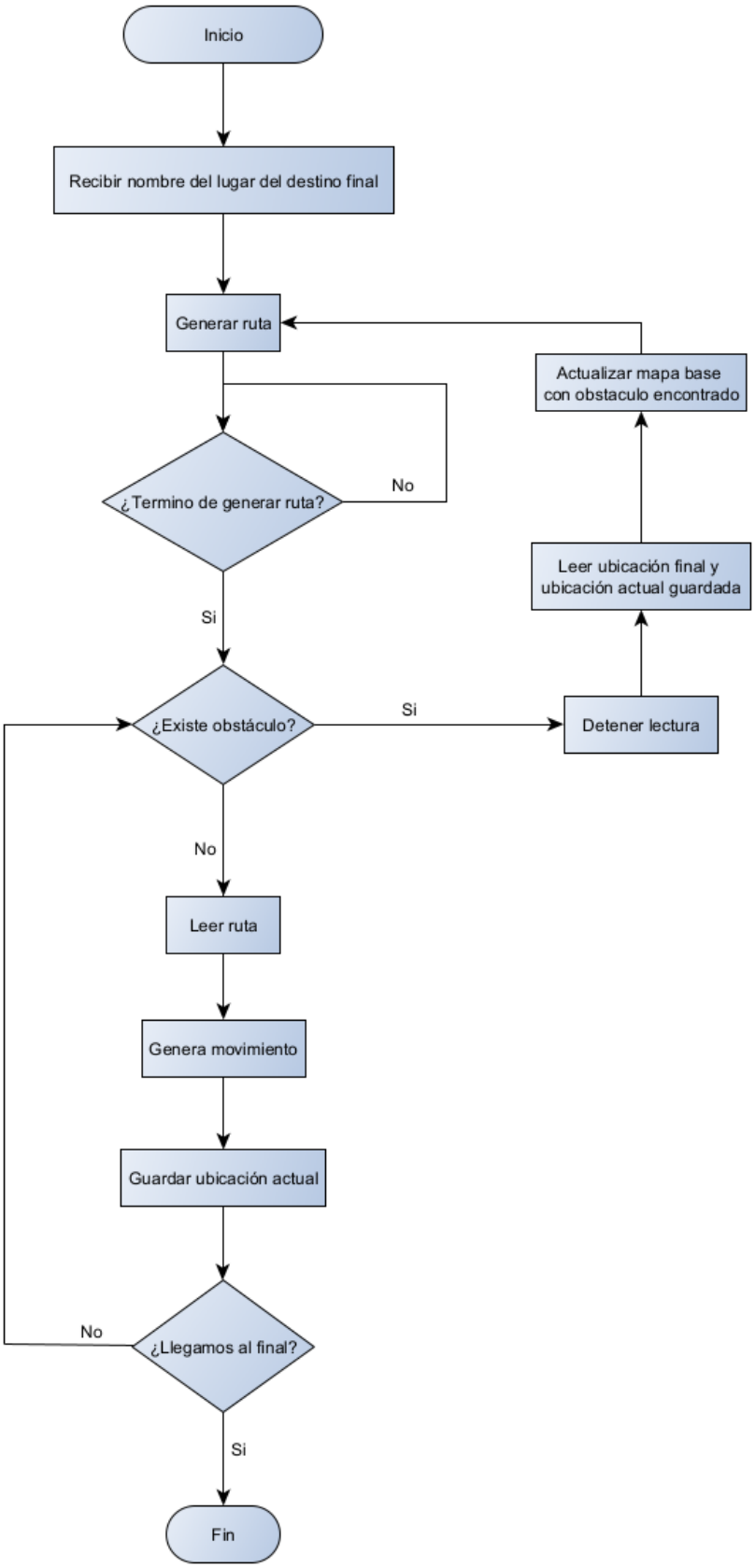


Figura 176: Diagrama de flujo del robot.

8.2.7.2. Funcionamiento del movimiento del robot después de implementar el giroscopio

Los pasos que sigue el robot para poder seguir la ruta generada hasta llegar al punto final son los siguientes y pueden ser observados en la Figura 177 como vemos se añadieron dos pasos mas en los que el robot lee el giroscopio y si hubo algún tipo de desviación el robot lo corregirá automáticamente.

1. Se ejecuta el archivo principal del robot donde se tiene previamente escrito el nombre del lugar a donde enviará el paquete y el nombre del lugar donde se encuentra ubicado actualmente.

2. El robot espera la generación de la ruta a seguir donde una vez que haya sida calculada entra al seguimiento de la ruta.

3. El robot irá leyendo la ruta mientras se mueve e ira usando los sensores para poder detectar algún obstáculo en el seguimiento de la ruta, en el caso de que haya una desviación el robot corregirá esta automáticamente y también ira almacenando su ubicación, la orientación y el punto final por si hubiera alguna interrupción el seguimiento de la ruta será interrumpido y se tendrá que recalcular la ruta con base en la ubicación en donde está el robot y el punto final.

4. Al llegar al final se considerarán las coordenadas finales como las nuevas iniciales del robot para la siguiente ruta a seguir.

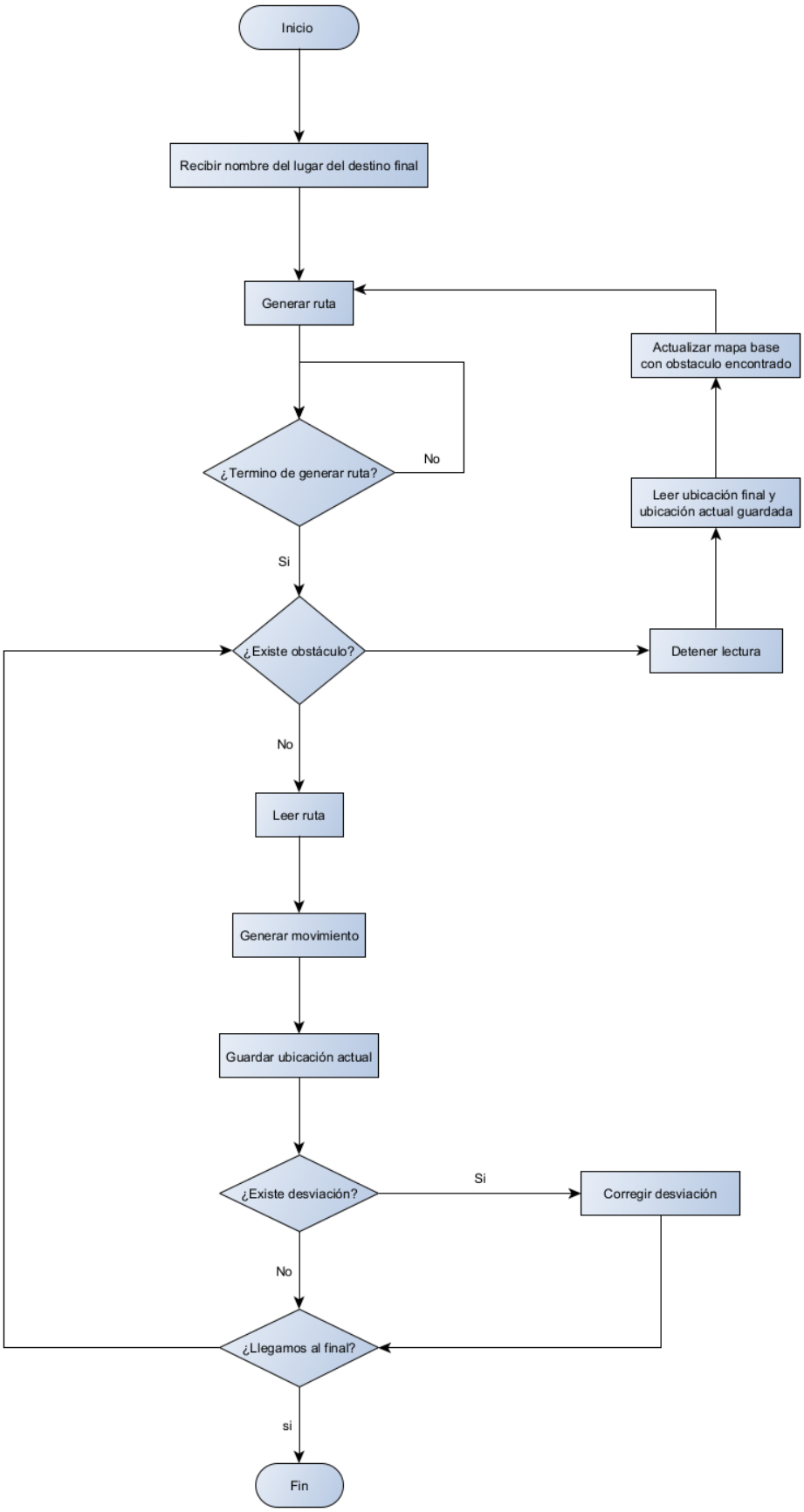


Figura 177: Diagrama de flujo del robot con el giroscopio implementado.

Es importante mencionar que las rutas generadas no son rectas completamente, de hecho, existe la posibilidad de que se nos presenten rutas en las cuales tengamos caminos escalonados o zigzagueantes como lo podemos ver en un ejemplo práctico con la Figura 178, para ello se programó que el robot priorice el movimiento recto esto debido a que es sin duda alguna el más fácil de llevar a cabo debido a las dimensiones del robot, pero en el caso de que no exista un movimiento recto disponible prioriza buscar una diagonal, sin embargo, cabe la posibilidad que debido a la orientación que en ese momento tenga el robot no encuentre ninguno de los dos casos anteriormente mencionados procederá a tomar una orientación basada principalmente en una pregunta ¿Hacia qué lado está el punto final?, es decir, el robot se preguntará ¿Se encuentra a la derecha o izquierda? o ¿Está hacia enfrente o detrás?, en caso de que todo lo anterior mencionado no sea así buscará una orientación que le permita seguir avanzando independientemente de la ubicación del punto final.

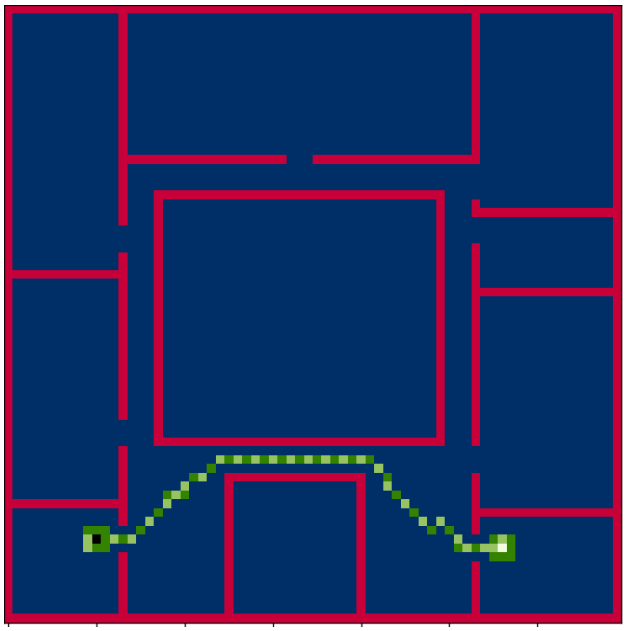


Figura 178: Ruta en donde se presentan camino recto, camino zigzagueante y camino escalonado.

8.2.8. Ejemplo de seguimiento de rutas del robot de manera lógica

En esta sección veremos un pequeño ejemplo del funcionamiento del algoritmo creado para el seguimiento de la ruta creada por nuestro modelado del Physarum Polycephalum, para que el algoritmo funcione es necesario que se conozcan las coordenadas finales e iniciales de la ruta, el cual debido a como se programo el algoritmo siempre la conoce, una vez iniciado el seguimiento de la ruta el robot sigue las indicaciones que el algoritmo le dé. Mencionar que algoritmo siempre prioriza el camino recto ya que es el mas sencillo de seguir, así mismo, en caso de que no encuentre camino recto busca las esquinas de la vecindad tomando como base la orientación que tenga el robot en ese momento, en caso de no encontrar alguna célula para avanzar en la misma orientación busca una nueva orientación que le permita continuar el camino, también se implementó un tipo de memoria para que vaya recordando en donde ya ha pasado para que no vuelva a pasar por ahí.

Ahora vemos el ejemplo, como vemos en la figura 179 se creo una pequeña ruta que el algoritmo del seguimiento de rutas tendrá que seguir, a continuación de la figura 179 vemos la salida que el algoritmo nos está dando, el cual es como el robot se tendría que mover (mencionar que el robot sigue este algoritmo sigue en tiempo real) este algoritmo se detiene hasta que este en el punto final de la ruta. La parte de la salida que tiene la palabra "Lógico" es que se tuvo la necesidad de que de buscar otra orientación para poder seguir la ruta que se creó, se muestra la vecindad, la orientación actual del robot, el movimiento a seguir del robot y el formato de guardado de la información del robot.

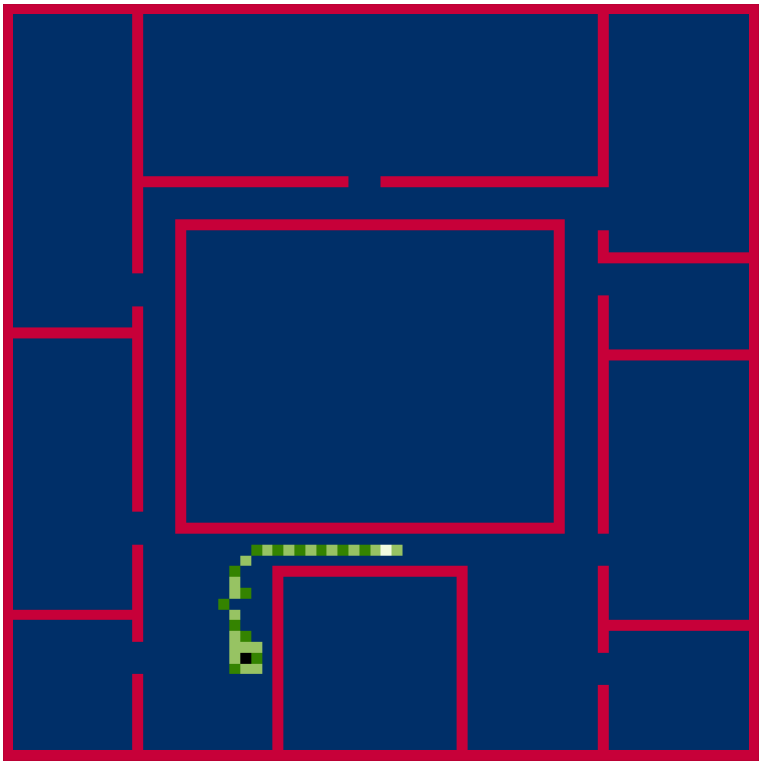


Figura 179: Ruta a seguir (Ejemplo del funcionamiento del algoritmo seguidor de rutas).

Seguimiento de la ruta generada por nuestro algoritmo (dividido en dos columnas, leer la columna de la izquierda primero y luego la columna derecha).

```
8.0 8.0 8.0
8.0 3.0 5.0
5.0 8.0 8.0
N
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 22,
    "pos_y": 59
}
#####################################
8.0 5.0 0.0
8.0 8.0 8.0
8.0 3.0 5.0
N
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 22,
    "pos_y": 58
}
#####################################
5.0 0.0 0.0
8.0 5.0 0.0
8.0 8.0 8.0
N
Horientando hacia la izquierda
Moviendo hacia delante
Horientando hacia la derecha
Moviendo hacia delante
diagonal directa
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 21,
    "pos_y": 57
}
#####################################
0.0 8.0 0.0
0.0 5.0 0.0
0.0 8.0 5.0
N
Moviendo hacia delante
{
    "fin_x": 35,
```

```
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 21,
    "pos_y": 56
}
#####################################
5.0 0.0 0.0
0.0 8.0 0.0
0.0 5.0 0.0
N
Horientando hacia la izquierda
Moviendo hacia delante
Horientando hacia la derecha
Moviendo hacia delante
diagonal directa
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 20,
    "pos_y": 55
}
#####################################
0.0 0.0 8.0
0.0 5.0 0.0
0.0 0.0 8.0
N
Horientando hacia la derecha
Moviendo hacia delante
Horientando hacia la izquierda
Moviendo hacia delante
diagonal directa
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 21,
    "pos_y": 54
}
#####################################
0.0 8.0 0.0
0.0 8.0 5.0
5.0 0.0 0.0
N
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 21,
    "pos_y": 53
}
#####################################
0.0 5.0 0.0
0.0 8.0 0.0
0.0 8.0 5.0
N
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 21,
    "pos_y": 52
}
#####################################
0.0 0.0 8.0
0.0 5.0 0.0
0.0 8.0 0.0
N
Horientando hacia la derecha
Moviendo hacia delante
Horientando hacia la izquierda
Moviendo hacia delante
diagonal directa
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 22,
    "pos_y": 51
}
#####################################
0.0 0.0 5.0
0.0 8.0 0.0
5.0 0.0 0.0
N
Horientando hacia la derecha
Moviendo hacia delante
Horientando hacia la izquierda
Moviendo hacia delante
diagonal directa
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "N",
    "pos_x": 23,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
0.0 5.0 8.0
8.0 0.0 9.0
N
Logico
E
Horientando hacia la derecha
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 23,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
0.0 5.0 8.0
8.0 0.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 24,
    "pos_y": 50
```

```
}
#####################################
9.0 9.0 9.0
5.0 8.0 5.0
0.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 25,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
8.0 5.0 8.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 26,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
5.0 8.0 5.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 27,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
8.0 5.0 8.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 28,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
5.0 8.0 5.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 29,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
8.0 5.0 8.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 30,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
5.0 8.0 5.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 31,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
8.0 5.0 8.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 32,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
5.0 8.0 5.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "E",
    "pos_x": 33,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
8.0 5.0 8.0
9.0 9.0 9.0
E
Moviendo hacia delante
{
    "fin_x": 35,
    "fin_y": 50,
```

```
    "orientacion": "E",
    "pos_x": 34,
    "pos_y": 50
}
#####################################
9.0 9.0 9.0
5.0 8.0 6.0
9.0 9.0 9.0
E
W
Moviendo hacia delante
Horientando hacia la izquierda
Horientando hacia la izquierda
{
    "fin_x": 35,
    "fin_y": 50,
    "orientacion": "W",
    "pos_x": 35,
    "pos_y": 50
}
9.0 9.0 9.0
8.0 6.0 8.0
9.0 9.0 9.0
Fin del caminante:
Datos del caminante
Orientacion final W
Coordenada x final 35
Coordenada y final 50
```

Para ver algunas pruebas realizadas con esta versión del robot por favor dirigirse al apartado de pruebas y en el subapartado dedicado a esta segunda versión del robot (robot viajero).

8.3. Tercera versión del robot. (Robot mensajero)

8.3.1. Nuevos requerimientos

Como última versión de nuestro robot necesitamos añadir una caja diseñada para que puedan meter paquetes en ella, ademas de que necesitamos alguna forma de seguridad para que esta caja no pueda ser abierta por cualquier persona, si no, que únicamente sea abierta por la persona que envía y recibe el paquete, la primera persona para poder guardar el paquete y la segunda para que pueda tomar el paquete, ademas de esto necesitamos añadir dos centros de carga para que puedan cargar las dos baterías que el robot ocupa, esto para que el mismo robot pueda trabajar adecuadamente.

Como lo hicimos en el subcapítulo anterior, dedicaremos una sección para hablar de los diseños de alto y bajo nivel para dar soluciones a los nuevos requerimientos que encontramos para poder completar el robot MemOso, así como implementar estos para así dar paso a la tercera y última versión del robot el cual tiene como objetivo usar el simulador 2DPyCAM(10) y una interfaz web para la generación de rutas con el fin de poder moverse en el primer piso del edificio de gobierno de la ESCOM y así poder hacer entregas de

paquetes con un peso no mayor a 5kg.

8.3.2. Diseños alto y bajo nivel e implementación en el robot MemOso

Como vemos uno de los requerimientos nuevos para esta tercera y prácticamente ultima versión del robot necesitamos una caja de acrílico, para ello se mando a hacer una caja de acrílico, se escogió este material debido a su relación resistencia peso que tiene, ya que necesitábamos algo ligero pero resistente, esta caja tiene las siguientes características:

1. Dimensiones (exteriores): 30 cm de alto, 30 cm de ancho y 50 cm de largo.
2. Tapa con tres bisagras para fácil apertura.
3. Cantos pulidos.
4. Acrílico transparente de 4 mm para soportar cargas pesadas..

Como podemos ver en la figura 180 esta la representación aproximada de la caja junto con las características anteriormente mencionadas.

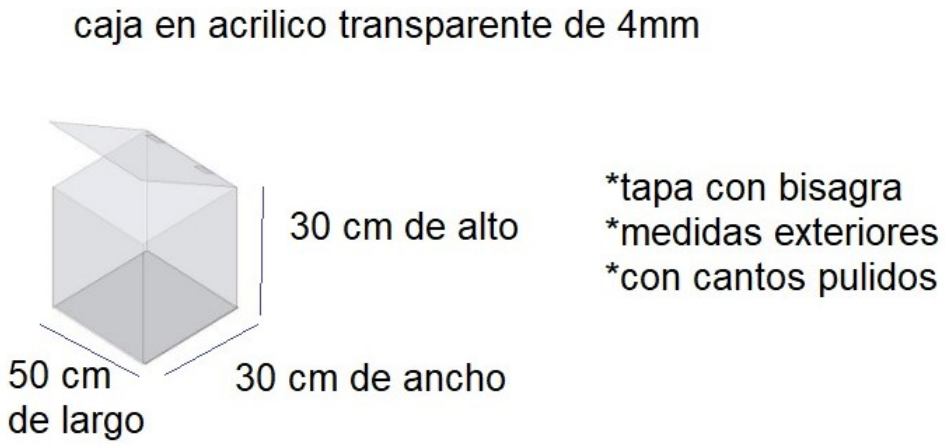


Figura 180: Representación de la caja a usar con características.

Ahora veamos la parte de seguridad de la caja, para esto hacemos uso de un servomotor Futaba s3003 (hablaremos un poco mas adelante sobre el), así como una de sus hélices

que vienen incluidas regularmente cuando se compran este tipo de servomotor, también hacemos uso de palitos de madera, un poco de pegamento blanco y Kola Loka. El sistema es bastante simple y funciona como un pasador, básicamente con nuestro servomotor y la hélice que se le coloco será nuestro pasador automático, la finalidad de los palitos de madera es juntarlos para crear una especie de pared y que la hélice en cierta posición choque con dicha pared para así evitar la abertura no deseada de la caja, todo lo anterior lo podemos ver en la figura 181, en donde podemos ver la hélice del motor un poco debajo de donde se encuentra la pared de madera.

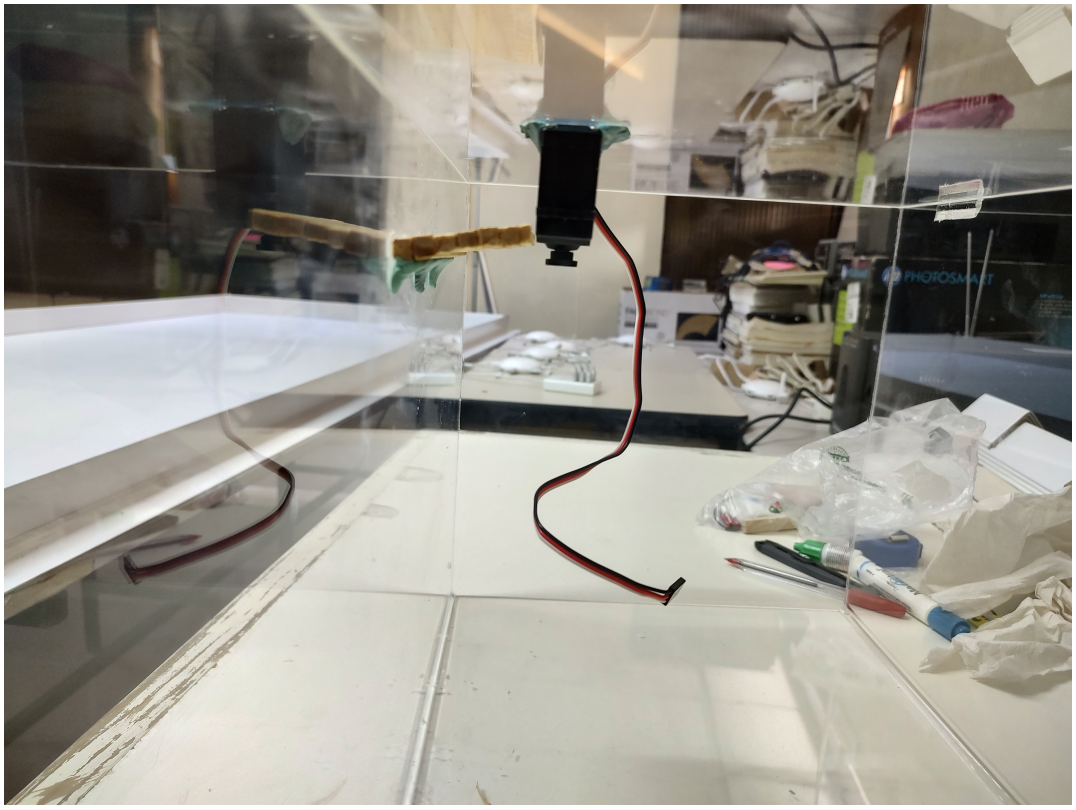


Figura 181: Sistema de apertura y bloqueo de la caja.

Ahora veamos unas figuras en donde podemos ver ejemplificado el funcionamiento de este sistema, en la figura 182 podemos ver como la hélice está en una posición en la cual esta justo debajo de la pared de madera y esto provoca, como lo podemos ver en la figura 183 que la caja no se pueda abrir completamente, solo permite abrirla un poco. Es importante mencionar que este sistema aun no este acoplado en su totalidad en el robot, sin embargo,

ya esta siendo controlado por este.

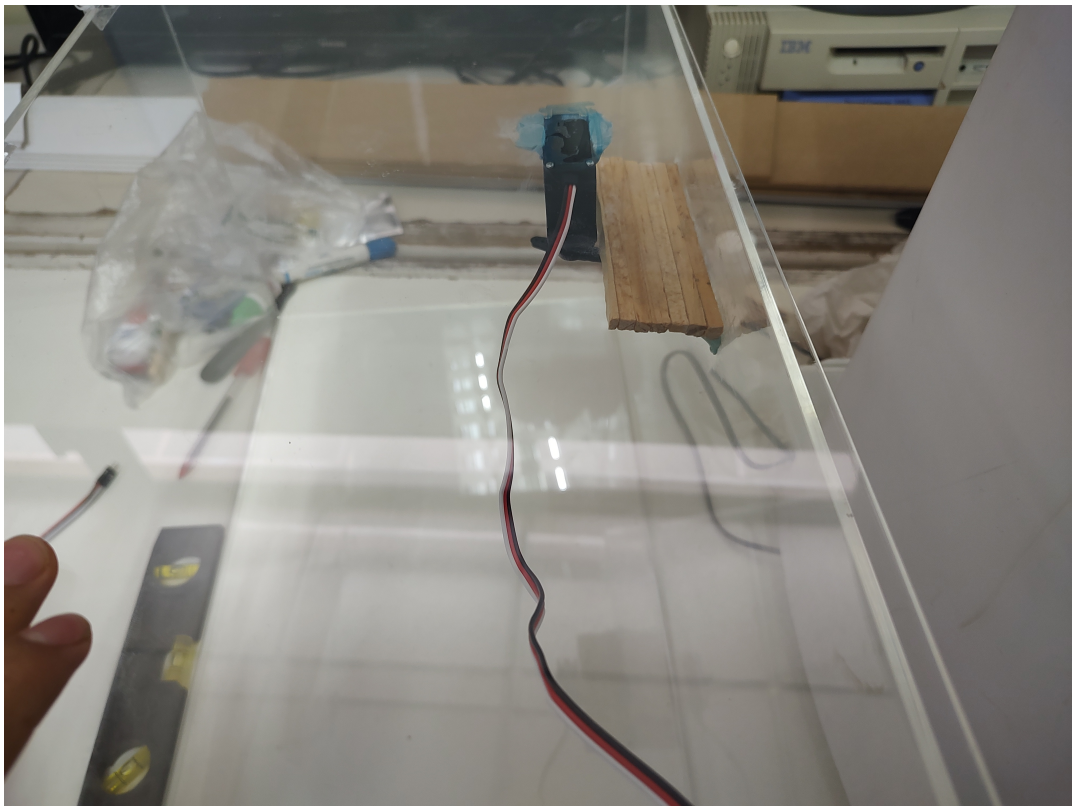


Figura 182: Caja bloqueada por el sistema.

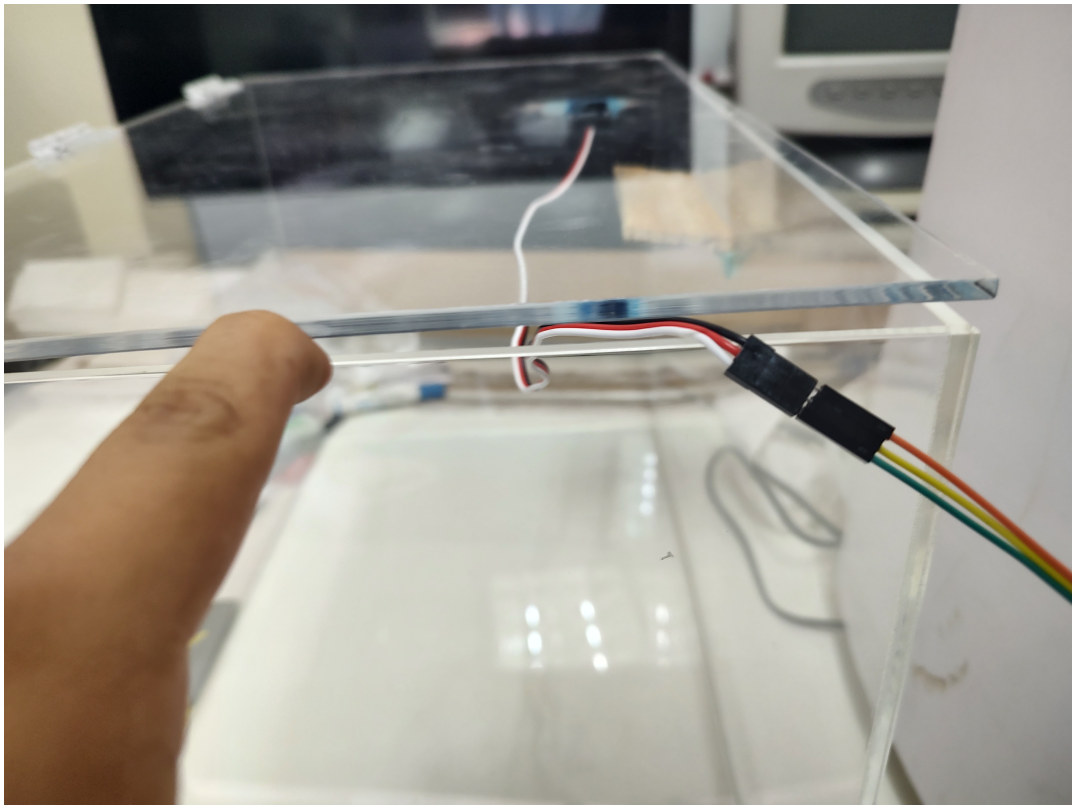


Figura 183: Caja bloqueada por el sistema (intento de apertura).

Por otro lado, podemos ver en la figura 184 cuando la hélice está en paralelo a la pared de madera podemos abrir la caja sin problema alguno como se ve en la figura 185.

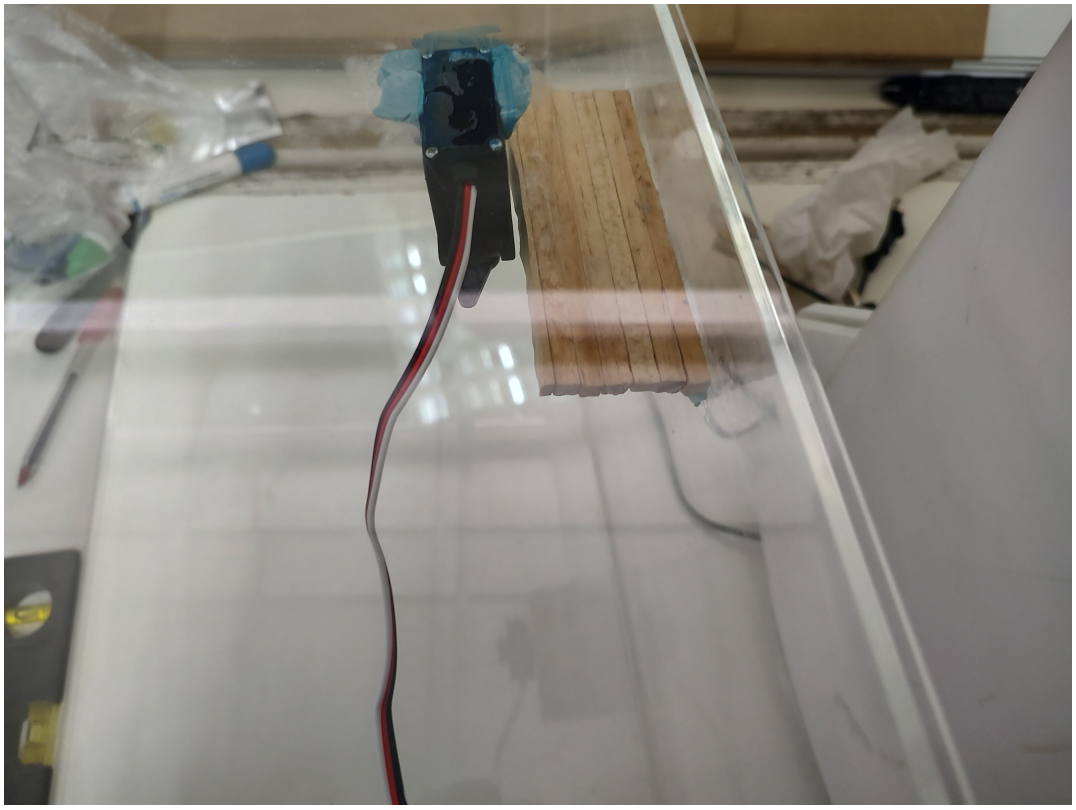


Figura 184: Caja no bloqueada por el sistema.

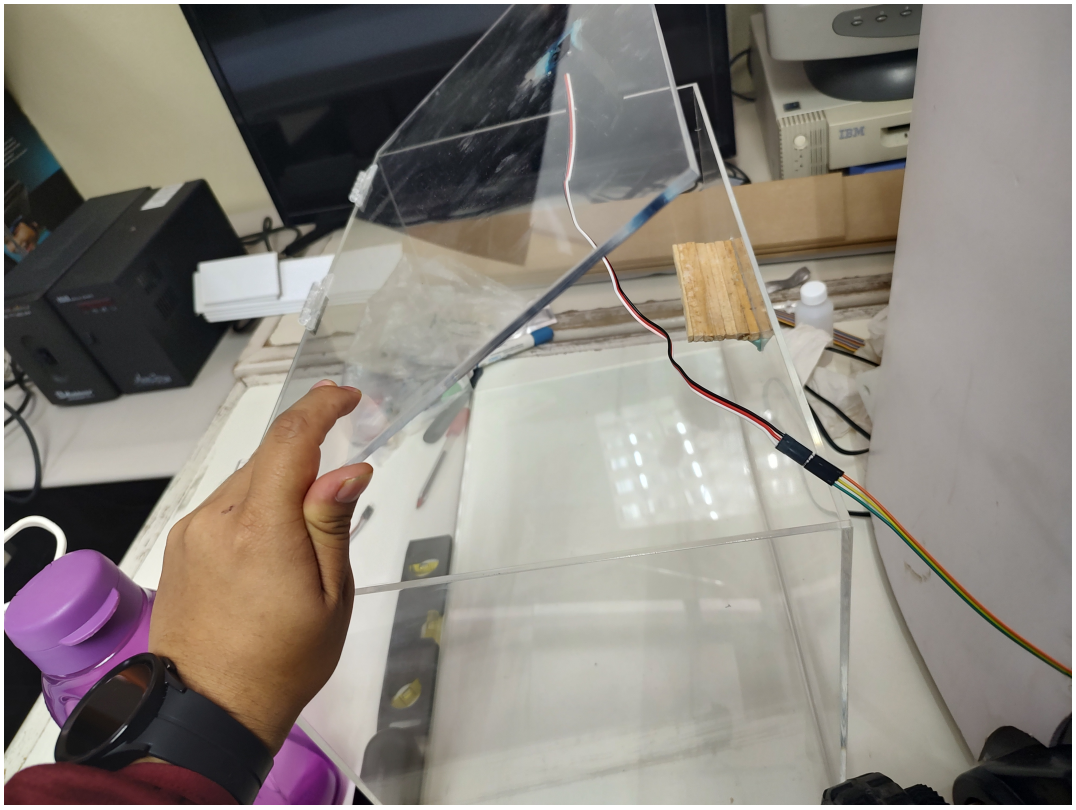


Figura 185: Caja no bloqueada por el sistema (apertura).

Ahora veamos la implementación de los centros de carga para las baterías del robot, esta parte fue muy sencilla de realizar ya que al hacer uso de un cargador de baterías de 12 v solo necesitamos agregar las entradas macho correspondientes, ya que el cargador tiene entrada hembra. En la figura 186 podemos ver cómo es físicamente estos centros de carga, como vemos son dos entradas en cada uno, el color naranja es la entrada positiva y el color amarillo la entrada negativa, con base en la figura el lado izquierdo corresponde a la batería de los motores y del lado derecho corresponde a la batería de la Raspberry Pi 4 que es la que maneja todos los sistemas que se tienen en el robot, mencionar que para poder hacer uso de estos centros de carga es necesario que todo el robot este apagado por medio de los interruptores que este tiene.

Figura 186: Centro de carga para las baterías del robot.

8.3.3. Circuito eléctrico del robot

Una vez implementados los requisitos para esta tercera version de nuestro robot MemOso veamos nuestro nuevo y prácticamente ultimo diagrama esquemático el cual lo podemos

ver en la Figura 187 adicionalmente en la Figura 188 podemos ver un diagrama pictórico del circuito con los nombres de los componentes y valores de resistencias para el funcionamiento del robot para poder hacer aún más entendible en la figura 149 ponemos ver los GPIO Pinout de la tarjeta Raspberry Pi 4 modelo B.

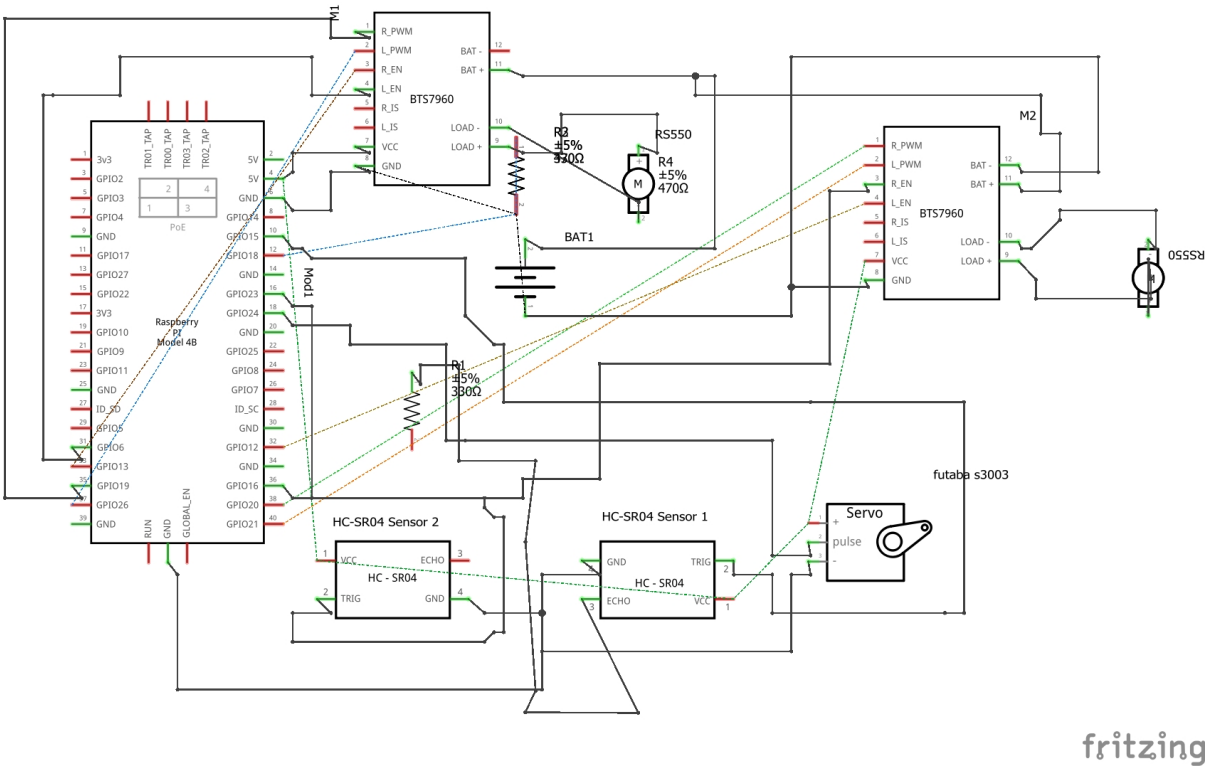


Figura 187: Diagrama esquemático del robot MemOso tercera versión.

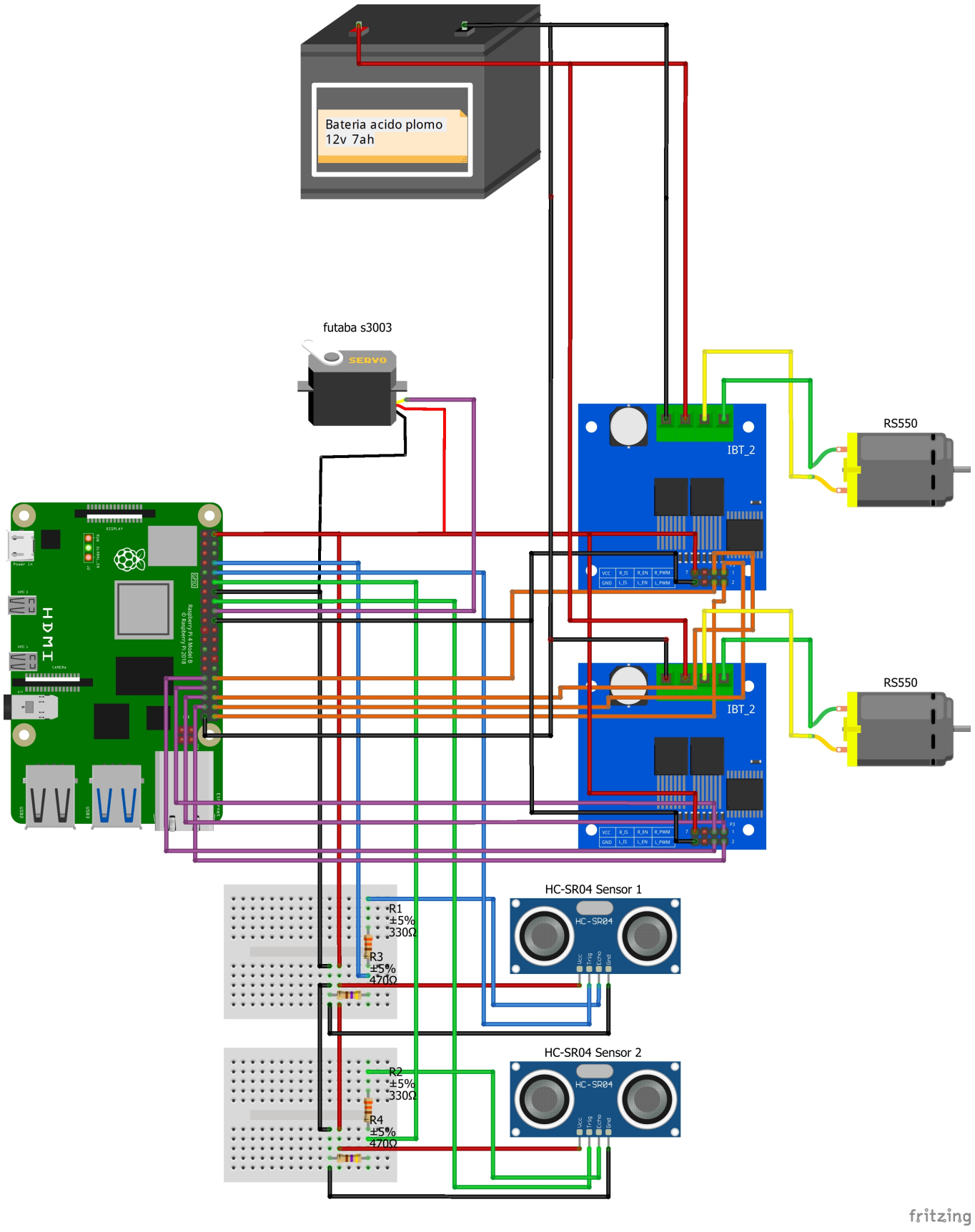


Figura 188: Diagrama pictórico del robot MemOso tercera versión.

8.3.4. Descripción de los nuevos componentes

8.3.4.1. Servomotor Futaba S3003

El servomotor por usar es un servomotor Futaba S3003 típicamente usado para el desarrollo de aplicaciones de robótica, ya que este tipo de motores de corriente continúan permiten dar giros o torques para movimientos de alguna articulación de robots de entro los más famosos son los robots tipo brazo manipulador, robot arácnido, etc. Este servomotor consta de tres terminales de las cuales sobre sale un cable de 3 colores, el color rojo representa la entrada de voltaje que para el caso de este robot puede trabar de un rango de 4.5v a 6 v, el color negro es la tierra (GND) y finalmente el blanco que representa la señal de control del Futaba.

Figura 189: Servomotor Futaba S3003. [16]

[16] Futaba Servo estándar S3003: Hobbico Inc: Amazon.com.mx: Juguetes y Juegos. (s. f.). https://www.amazon.com.mx/Futaba-FUTM0031-S3003-Standard-Servo/dp/B0015H2V72

8.4. Versión final del robot. (Robot mensajero - Integración de todo el sistema)

Como última versión solo tenemos unos pequeños añadidos al robot, que más que nada son indicativos de algunas cosas del robot, como a que sistema corresponde cada interruptor, así como los centros de carga e indicando su polaridad, aunado a esto la caja en donde se colocaran los paquetes ya está colocada en su posición definitiva, así como las baterías ya que encuentran pegadas dentro del robot para evitar que estas se despeguen y también la cámara del robot. Todo esto lo podemos ver en las siguientes figuras.

Figura 190: Etiqueta interruptor motores.

Figura 191: Etiqueta interruptor Raspberry Pi.

Figura 192: Etiquetas centros de carga indicando polaridad.

Para ver algunas pruebas realizadas con esta versión del robot por favor dirigirse al apartado de pruebas y en el subapartado dedicado a esta tercera versión del robot (robot mensajero).

9. Pruebas

9.1. Pruebas del simulador 2DPyCAM(10)

9.1.1. Requerimientos

Paras las simulaciones realizadas en donde se incluye la interfaz de usuario, las características del equipo en donde se realizaron son las siguientes.

- **Sistema Operativo**: Windows 10 Pro.
- **Procesador**: Intel Core i7-7700HQ, 2.80GHz.
- **Memoria RAM**: 16 Gb.

El requerimiento mas importante que podemos tener es que se tiene que tener la version de Python 3.9 o mayor instalada en el Sistema Operativo, ya que el programa puede ser perfectamente portado a cualquier tipo de sistema operativo si y solo si cumple la condición mencionada anteriormente.

9.1.2. Entorno gráfico

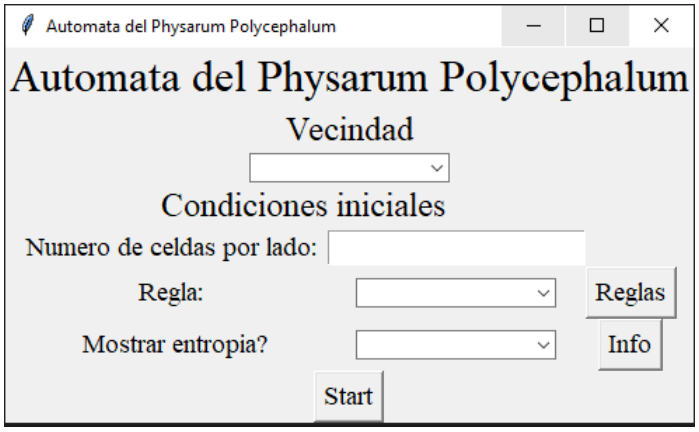


Figura 193: Pantalla de opciones para simulador.

En la figura 193 podemos ver la primera interfaz gráfica de nuestro simulador y en la figura 194 podemos ver un ejemplo de llenado, teniendo las siguientes opciones:

- **Vecindad**: Empleado para poder elegir entre la vecindad de Moore o von Neumann.
- **Numero de celdas por lado** : 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 Start.
- **Regla**: Se emplea para elegir entre dos tipos de reglas, la primera es la regla base de nuestro autómata celular, la segunda distancia de una célula con respecto a los repelentes para evitar colisiones.
- **Botón Reglas**: Este botón explica las reglas anteriormente mencionadas.
- **Mostrar entropia?**: Se emplea para elegir entre dos opciones, "Si" esta opción genera una gráfica con el calculo de la entropia y la opción "No" que ejecuta el simulador sin calcula la entropia.
- **Botón Info**: Este botón explica las condiciones para el calculo de la entropia.
- **Start**: Este botón inicia el simulador con las condiciones que hayamos introducido.

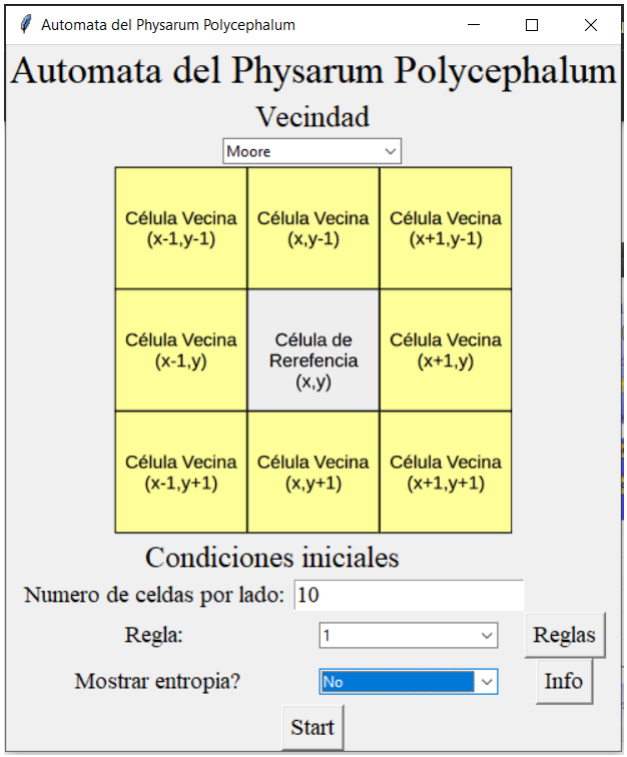


Figura 194: Ejemplo de llenado de las opciones para simulador.

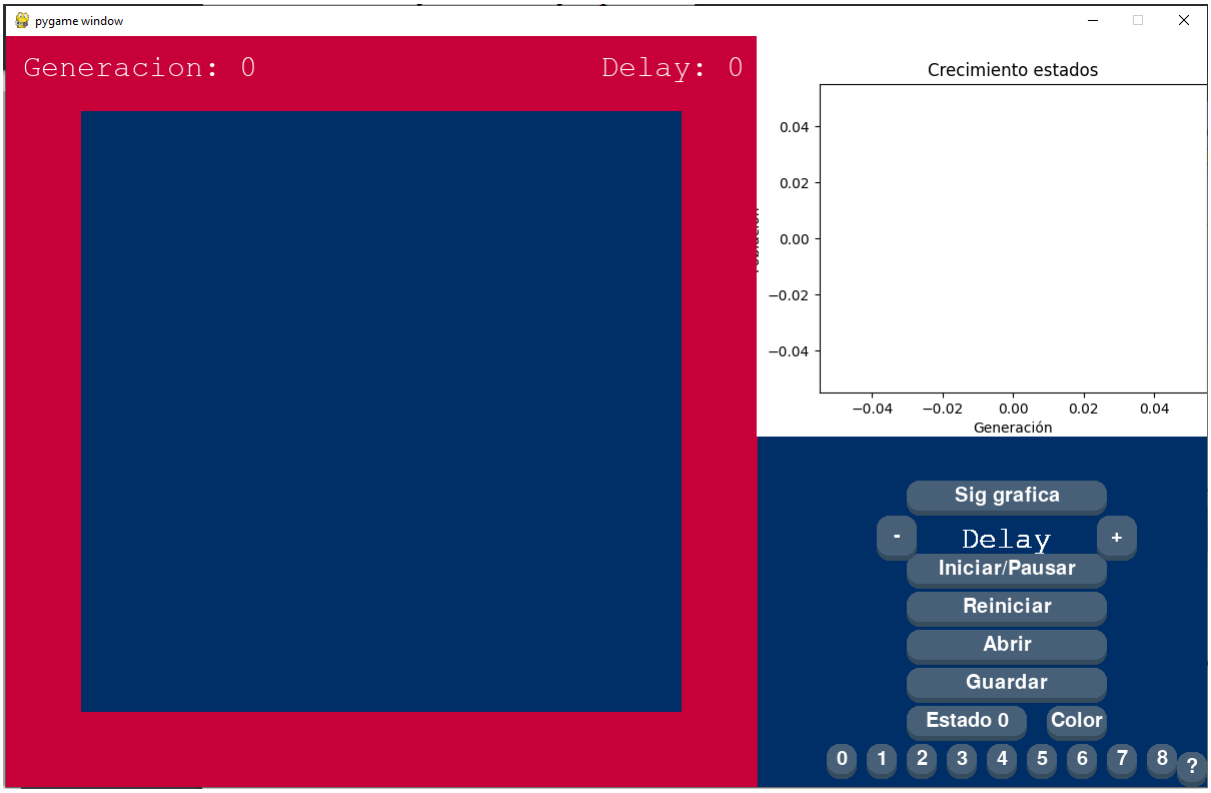


Figura 195: Simulador iniciado.

En la figura 195 podemos ver la ultima interfaz gráfica de nuestro simulador, teniendo las siguientes opciones:

- **Botón Sig gráfica**: Cambia entre la gráficas disponibles, las gráficas disponibles son "Crecimiento de estados" y "Crecimiento de estados log2".
- **Delay**: En este caso se cuentan con dos botones uno con signo "-" y "+", el primero provoca que haya menos delay en la ejecución de la simulación y por otro lado el segundo provoca que vaya con mas delay.
- **Botón Iniciar/Pausar**: Este botón nos permite iniciar o pausar el simulador.
- **Botón Reiniciar**: Este botón borra todo el contenido del simulador a excepción de los repelentes exteriores.
- **Botón Abrir**: Permite abrir un abrir un archivo con una configuración previamente creada.
- **Botón Guardar**: Guarda la configuración actual del simulador.
- **Botón Estado "n"**: Indica al usuario que estado es el con el que se esta trabajando.
- **Botón Color**: Cambia el color del estado "n" según la elección del usuario.
- **Botones 0-8**: Representan a los estados en el autómata,al seleccionar cualquiera de estos botones el valor "n" toma el valor del estado seleccionado.
- **Botón ?**: Informa a usuario que representa cada estado en el autómata celular.

9.1.3. Prueba del simulador con las principales carreteras de México

En esta prueba se recreara el análisis de [5], donde se busca representar las principales carreteras de México con el Physarum Polycephalum, en la figura 196 podemos ver el mapa de México con las principales carreteras.

Figura 196: Mapa de México con las principales carreteras.

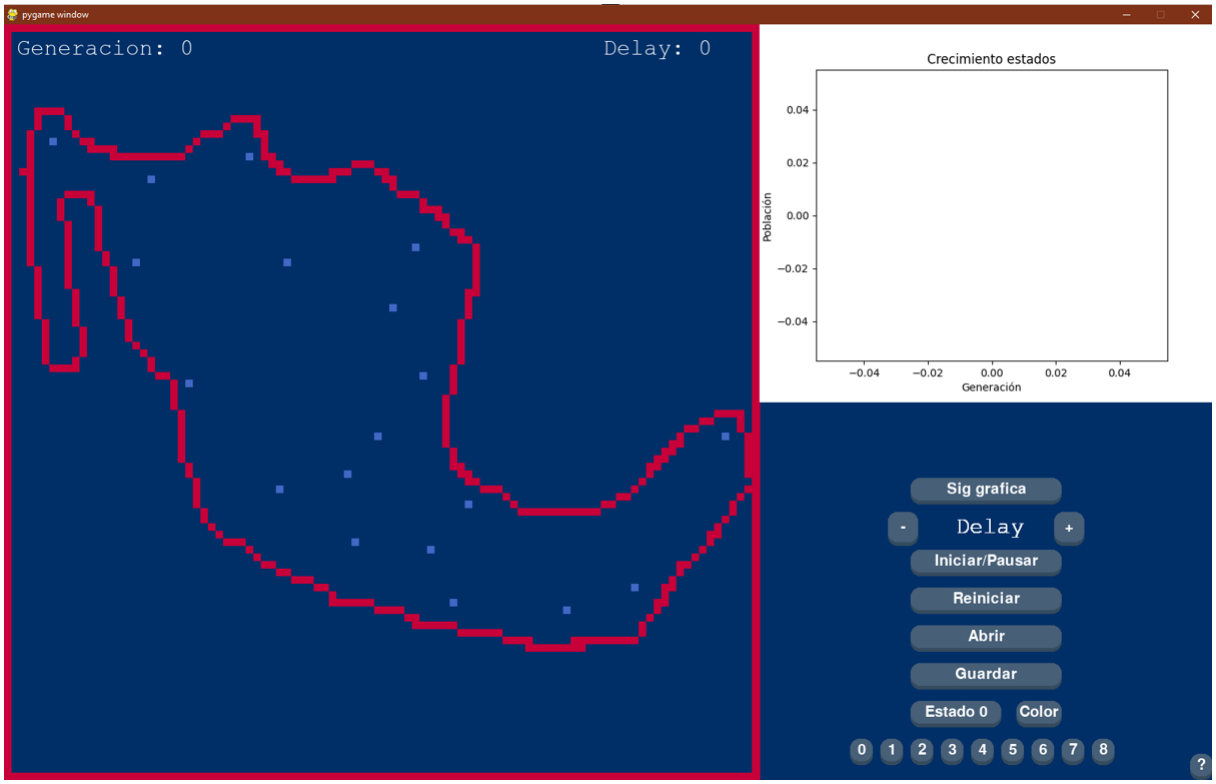


Figura 197: Mapa de México en el simulador.

En la figura 197 se muestra la representación de México, donde los nutrientes representan las principales ciudades del país.

9.1.3.1. Prueba usando la vecindad de Moore

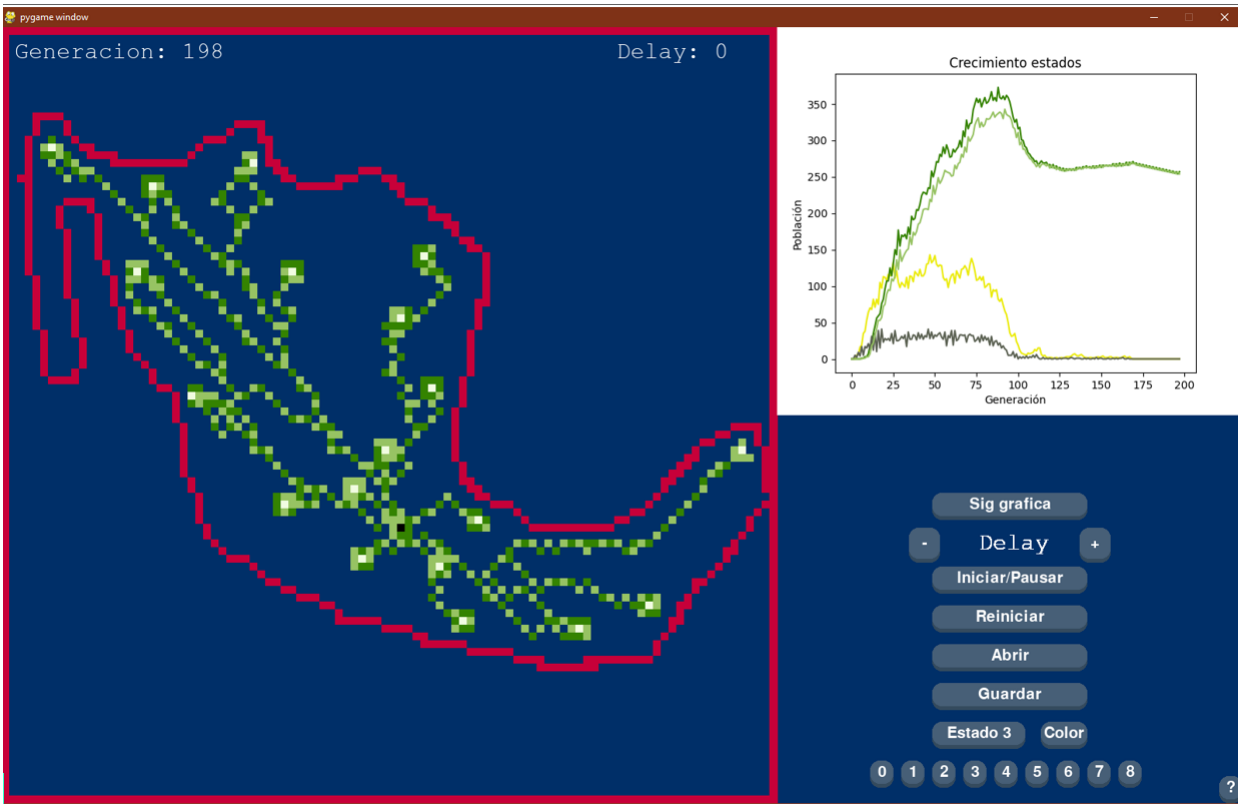


Figura 198: Prueba del mapa de México en el simulador usando Moore.

Prueba realizada con la vecindad de Moore, hasta la generación 198, el autómata entra en un estado estable, donde ya se encontraron todos los nutrientes, a comparación de la figura 199.

9.1.3.2. Prueba usando la vecindad de von Neumann

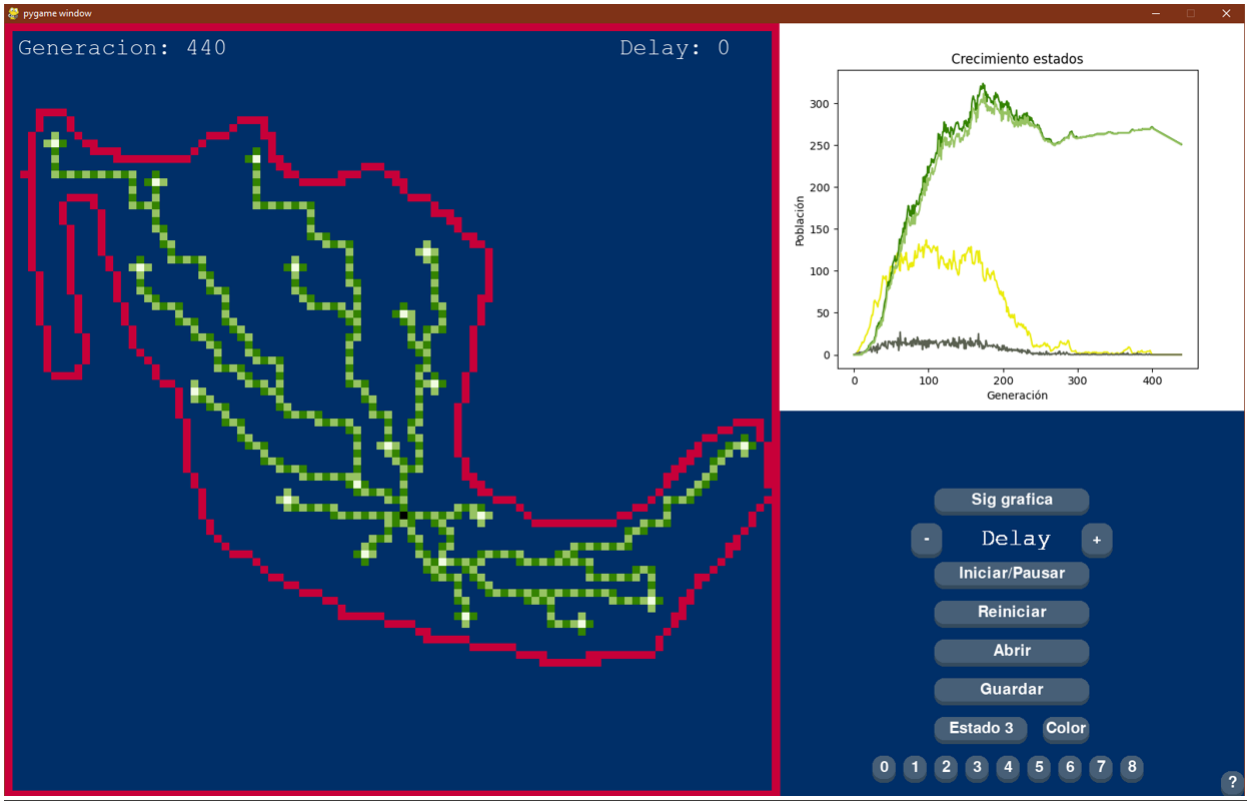


Figura 199: Prueba del mapa de México en el simulador usando von Neumann.

Prueba realizada con von Neumann, hasta la generación 440, el autómata entra en un estado estable, donde ya se encontraron todos los nutrientes, se puede apreciar la gran diferencia de generaciones entre vecindades. En este caso se puede apreciar una gran diferencia en las conexiones de los nutrientes en la parte norte del país a comparación de la simulación con la vecindad de Moore.

9.1.4. Prueba del simulador en el circuito de la Unidad Zacatenco-IPN

En esta prueba se crea una simulación en donde se busca ver el comportamiento del Physarum Polycephalum en el circuito de la unidad Zacatenco como lo podemos ver en la figura 200, así mismo podemos ver en la figura 201 el circuito de la Unidad Zacatenco ya en el simulador, mencionar que esta simulación tiene una dimension de 500 x 500 células.

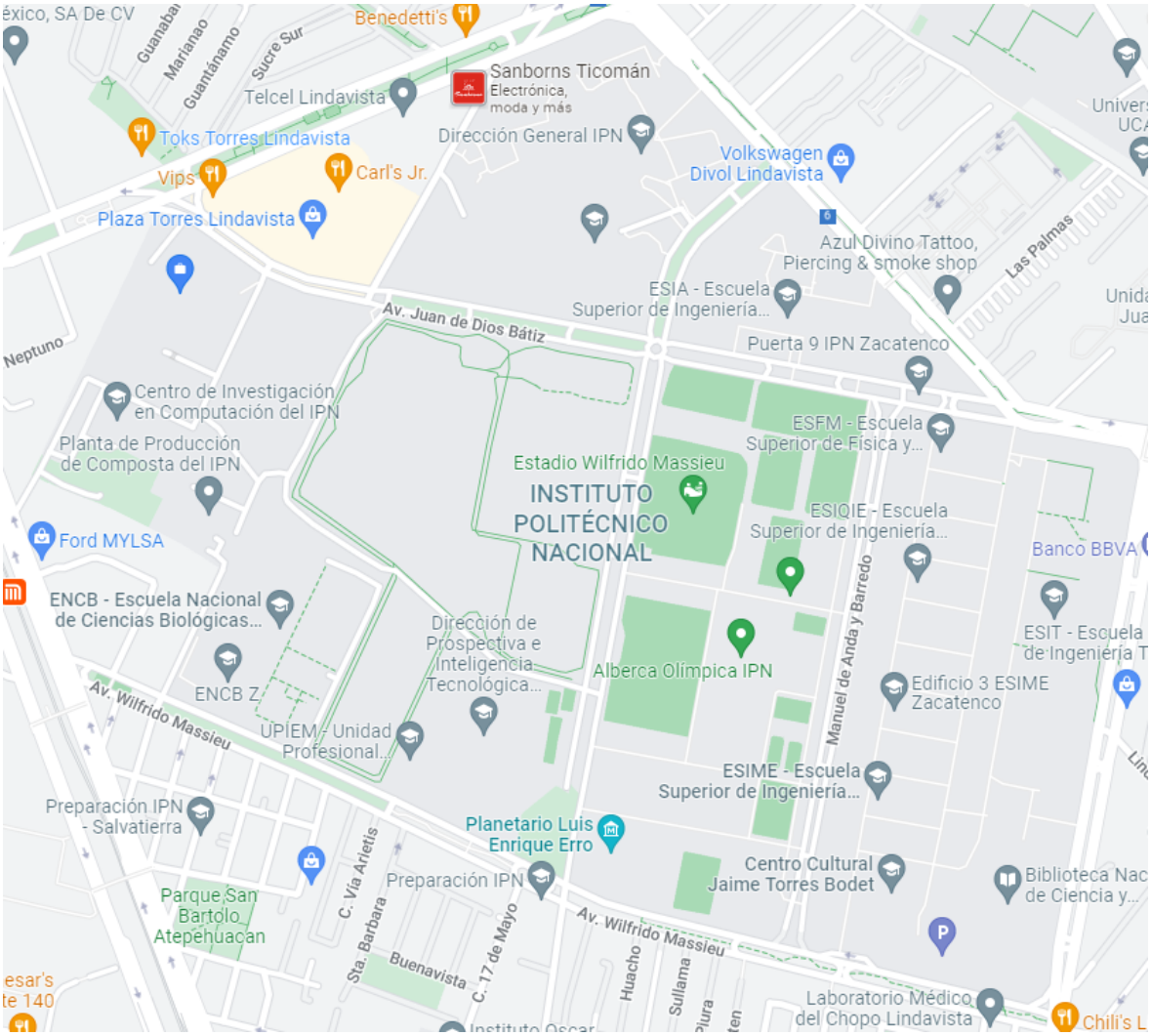


Figura 200: Mapa del circuito de la Unidad Zacatenco del IPN.

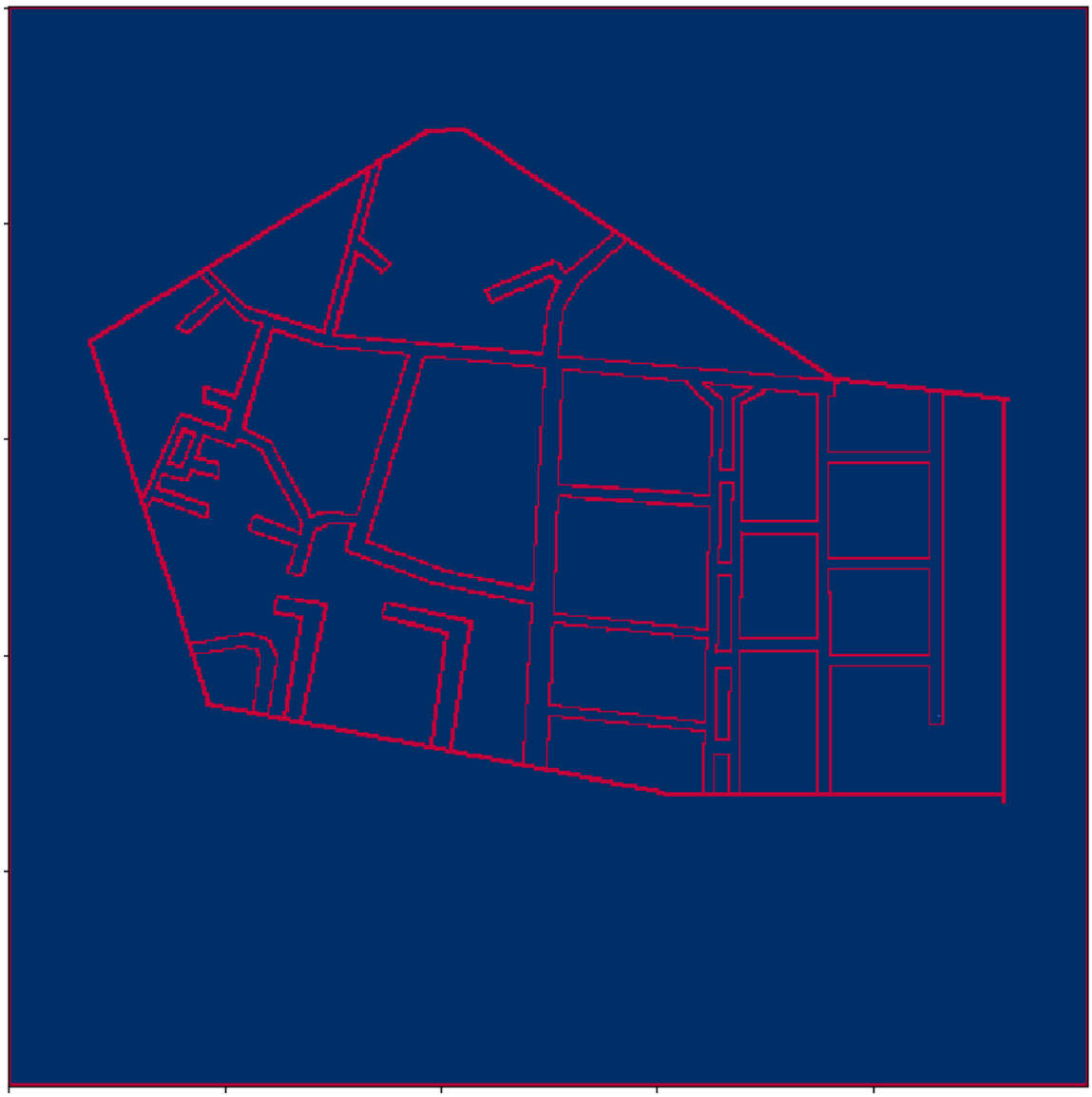


Figura 201: Mapa del circuito de la Unidad Zacatenco del IPN en nuestro simulador (Tamaño 500x500 células).

9.1.4.1. Prueba comparando con la tecnología GPS de Google Maps

En la figura 202 podemos ver una ruta generada por Google Maps para ir de un punto A a un punto B dentro de la Unidad de Zacatenco.

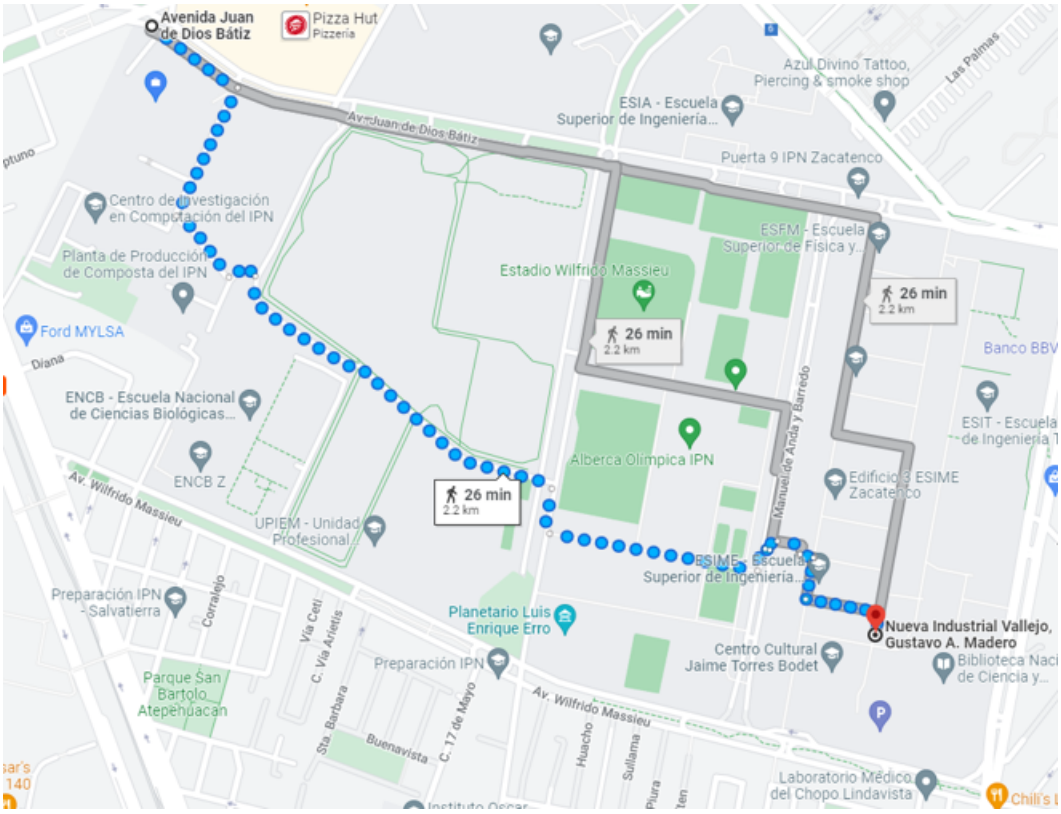


Figura 202: Ruta generada por Google Maps en la Unidad Zacatenco. [1]

Ahora en la figura 203 podemos ver una ruta generada por nuestro simulador para ir a los mismos puntos que en la figura pasada. Podemos ver como obtuvimos resultados similares entre ambas tecnologías, mencionar que nuestro simulador se tardó aproximadamente 1 hora completa para crear esta ruta, esto debido a que también hace el procesamiento de la animación la cual permite ver el avance de nuestro Physarum Polycephalum esta simulación tomo 2023 generaciones para completar la ruta, sin embargo, sin la parte visual la cual permite ver como avanza nuestro modelo nos tomo aproximadamente 20 minutos con la misma cantidad de generaciones en ambas pruebas. Finalmente en la figura 204 podemos ver otra ruta generada la cual tomo 1310 generaciones para generar la ruta, esta ruta la hicimos con una version del simulador en donde se omite por completo la parte gráfica y solo queda la generación de ruta, comparando con la figura 203 los cambios en la ruta son muy pocos, en realidad podríamos estar hablando de una misma ruta pero con

[1] Google (s.f.). [Indicaciones de Google Maps para caminar de Avenida Juan de Dios Bátiz, a Centro Cultural Jaime Torres Bodet]. Recuperado el 01 de octubre de 2022.

ligeras variaciones.

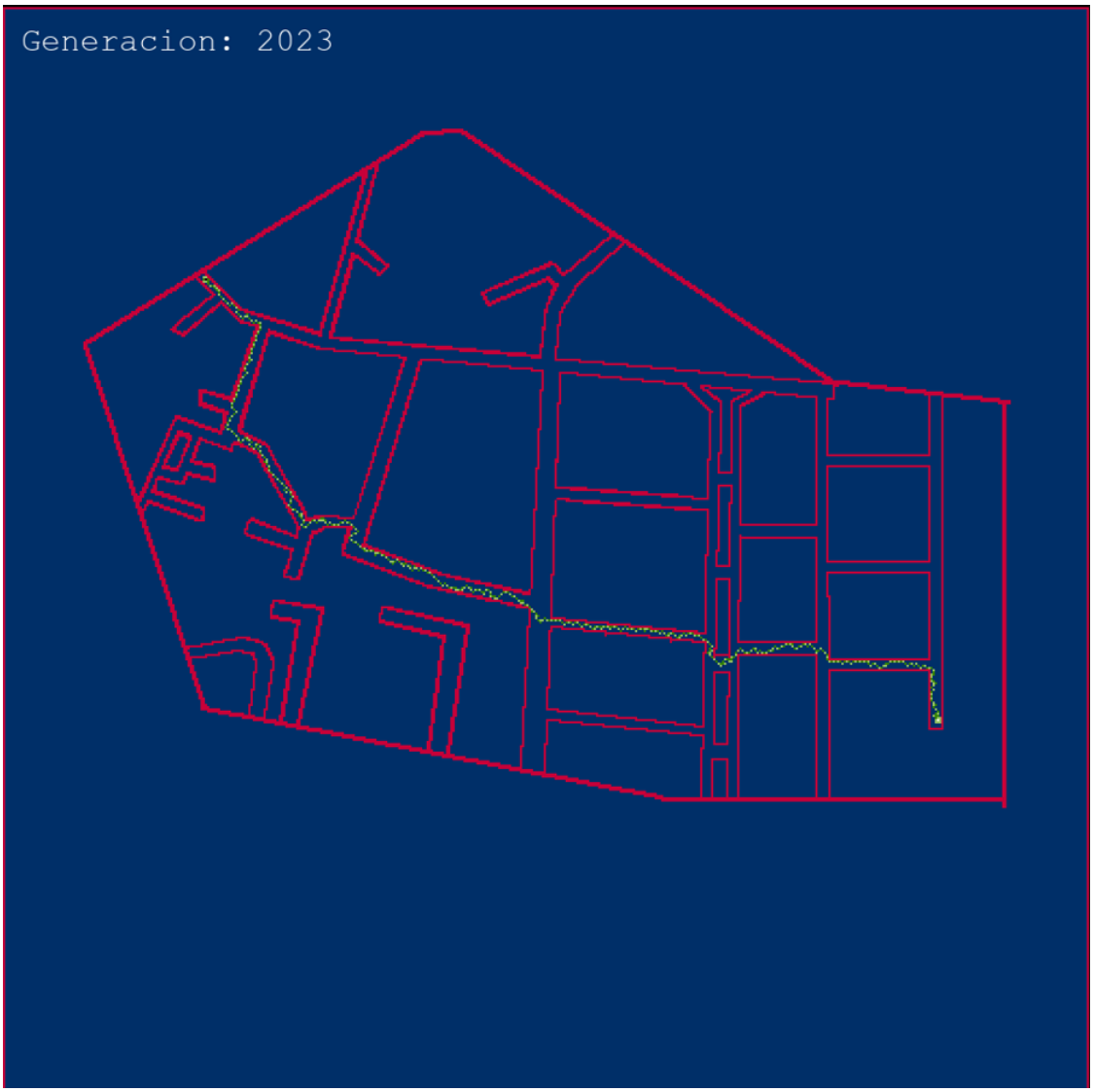


Figura 203: Ruta generada por nuestro simulador en la Unidad Zacatenco. Simulación 1.

Figura 204: Ruta generada por nuestro simulador en la Unidad Zacatenco. Simulación 2.

9.1.5. Prueba del simulador en la Escuela Superior de Computo

En esta prueba se crea una simulación en donde se busca ver el comportamiento del Physarum Polycephalum en la Escuela Superior de Computo como lo podemos ver en la figura 205, así mismo podemos ver en la figura 206 el mapa de la ESCOM ya en el simulador, mencionar que esta simulación tiene una dimension de 200 x 200 células.

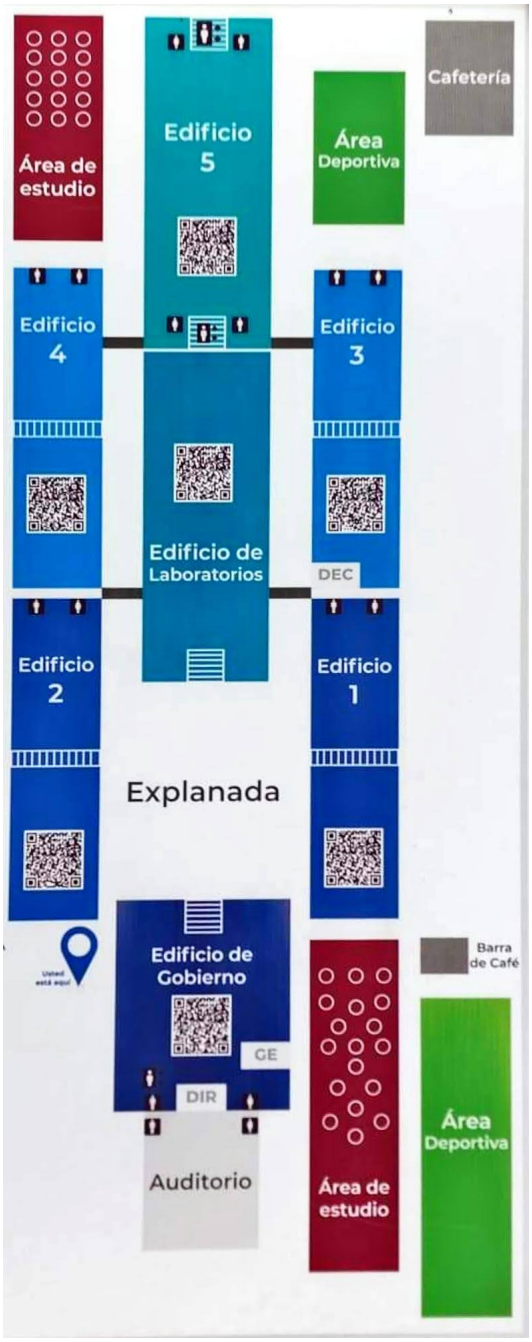


Figura 205: Mapa de la ESCOM del IPN. [2]

[2]Escuela Superior de Cómputo (s.f.). [Mapa 2D a escala de la Escuela Superior de Cómputo]. Recuperado el 18 de diciembre de 2022.

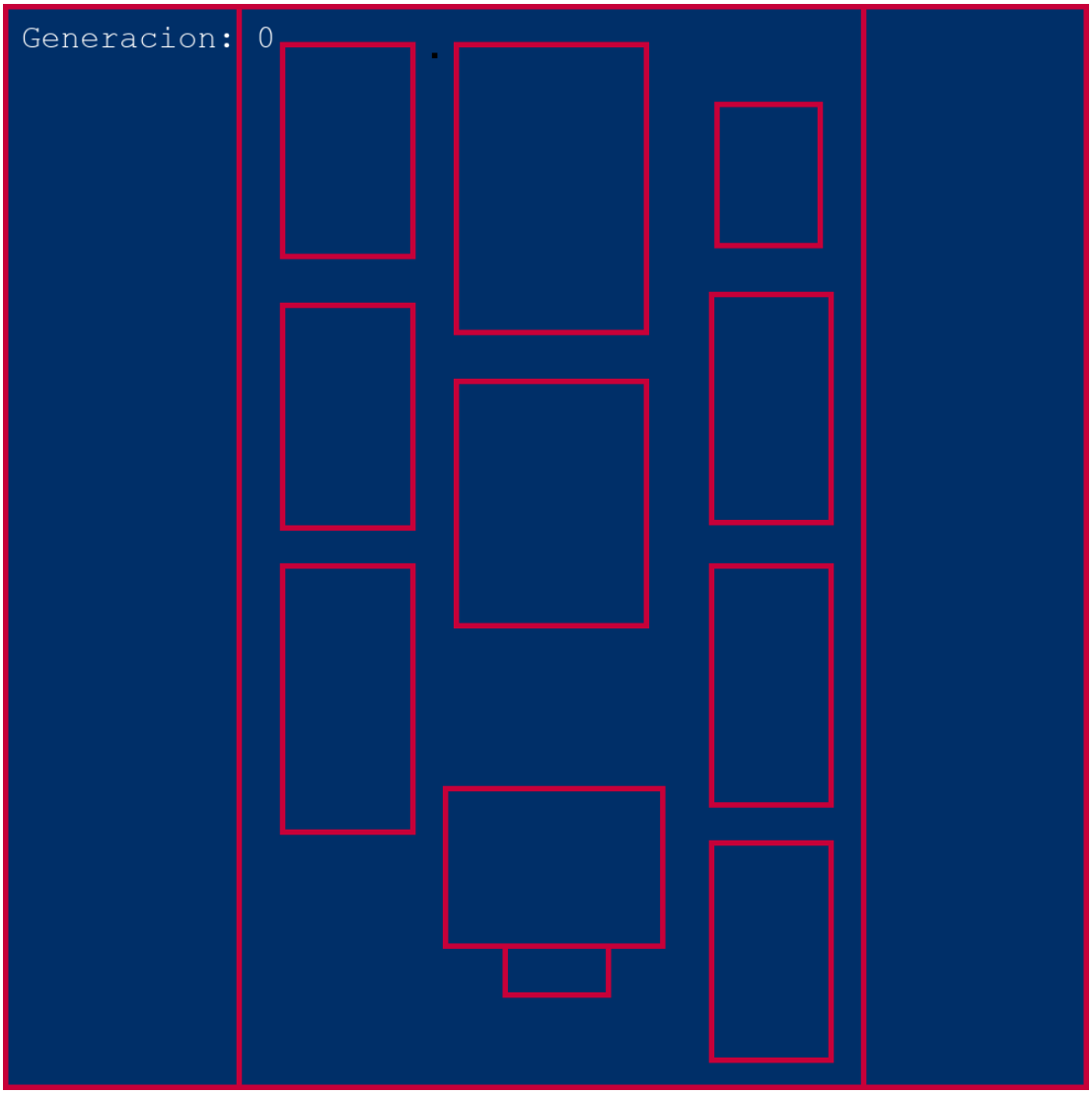


Figura 206: Mapa de la ESCOM del IPN en nuestro simulador (Tamaño 200x200 células).

Ahora en la figura 207 podemos ver una ruta generada por nuestro simulador, mencionar que nuestro simulador se tardó aproximadamente 3 minutos y medio para crear esta ruta, esto debido a que también hace el procesamiento de la animación la cual permite ver el avance de nuestro Physarum Polycephalum esta simulación tomo 583 generaciones para completar la ruta.

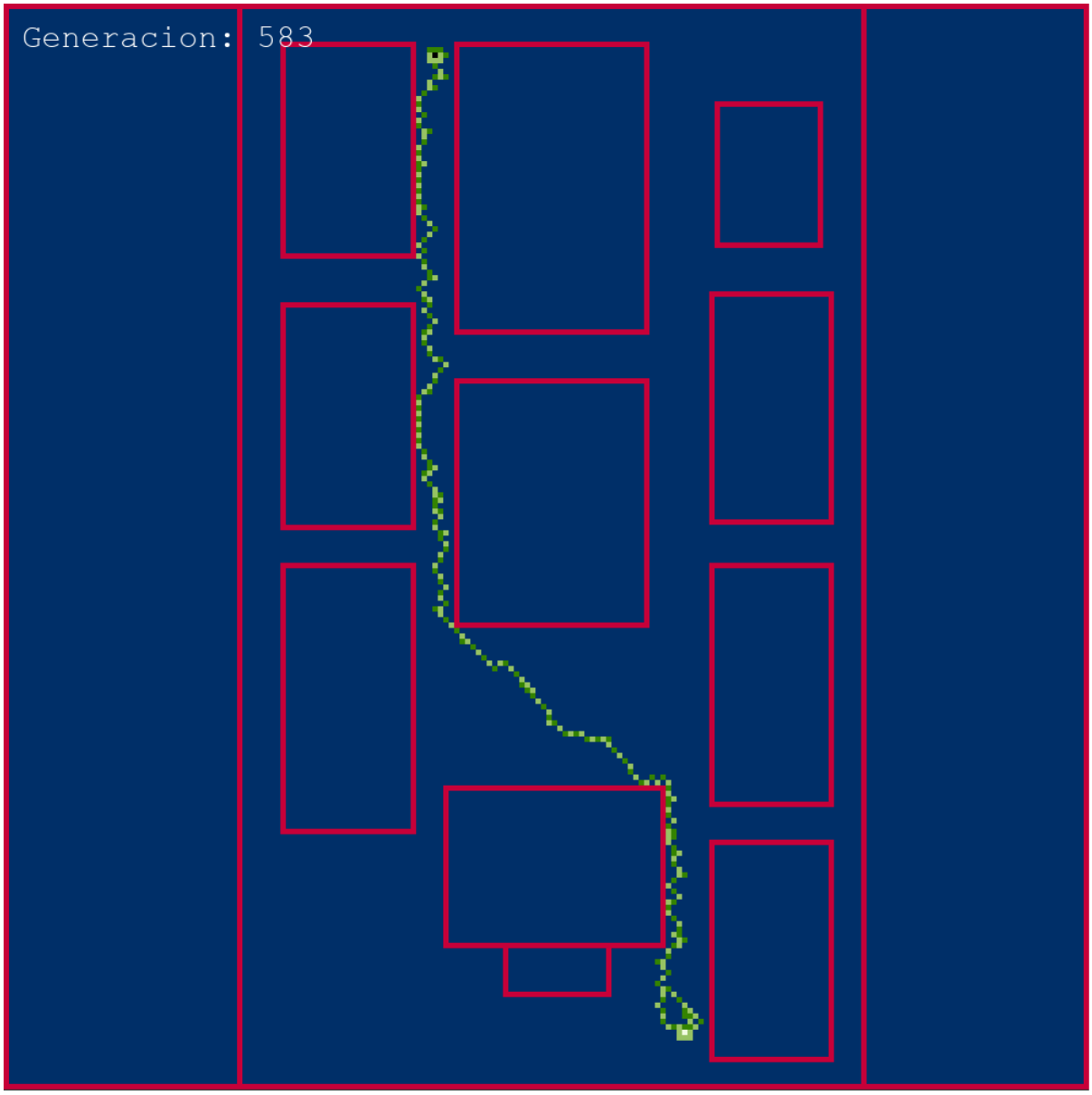


Figura 207: Ruta generada por nuestro simulador en ESCOM.

9.1.6. Comparación con otros modelos y el organismo real

En la sección estado del arte se hizo mención de algunos modelos inspirados en el Physarum Polycephalum, en el presente punto vamos a comparar algunos resultados o caracteristicas de los modelos mencionados con el nuestro, no necesariamente habláramos del tiempo de ejecución, aunado a lo anterior veremos al organismo real con el fin de poder ver que tan fiel es la propuesta del autómata celular.

9.1.6.1. Comparación con el modelo de Andrew Adamatzky

En la prueba realizada con nuestro autómata celular se utilizó de la vecindad de Moore, fue hasta la generación 183 en la que el autómata entra en un estado estable, donde ya se encontraron todos los nutrientes. Se puede apreciar que en el modelo de Adamatzky no crea huecos entre las rutas, tampoco se puede observar que un nutriente tengas mas de una ruta para llegar o salir de el, por lo que si existe variación entre los modelos, sin embargo, si comparamos nuestro modelo con el organismo real vemos como hay nutrientes que si tienen mas de una ruta, ademas de que si crean huecos entre las rutas, es decir,en nuestro autómata se obtuvo un comportamiento simular al organismo real, solamente cambio en el trazado de rutas.

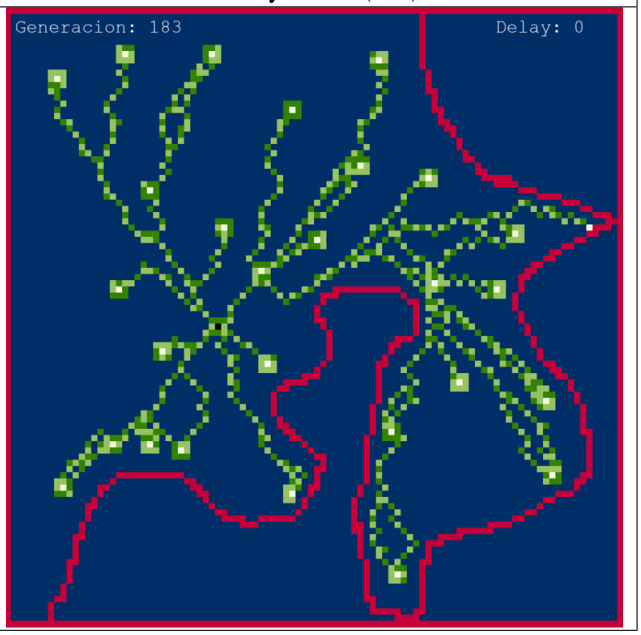
Red de Tokyo	
Organismo real [4]	**2DPyCAM(10)**
	

Tabla 31: Comparación de la Red de Tokyo con el organismo real.

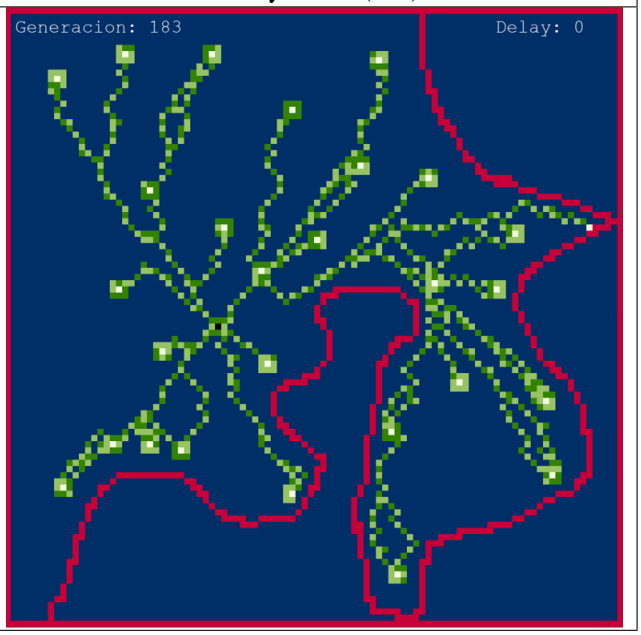
Red de Tokyo	
Modelo de Adamatzky [18]	**2DPyCAM(10)**
	

Tabla 32: Comparación de la Red de Tokyo con el modelo de Adamazky.

9.1.6.2. Comparación con el modelo de Jeff Jones

Como se puede apreciar el resultado de nuestro autómata celular y el de Jeff difieren debido a que este último tiene conexiones de las soluciones posibles debido a los propios sensores de los agentes que provocan que se sitúen dentro de una unión de soluciones.

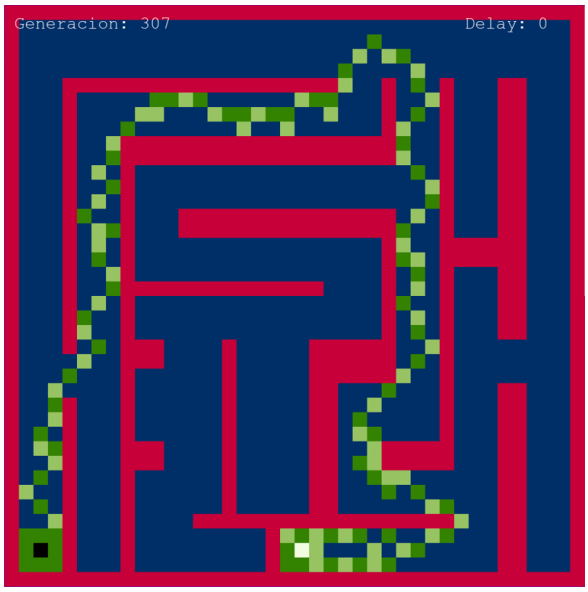
Laberinto	
Modelo de Jeff [17]	**2DPyCAM(10)**
	

Tabla 33: Comparación de solución de laberinto Jeff.

El cúmulo de los agentes con los que el modelo de Jeff trabaja proboca que se vea una línea de solución aparentemente uniforme, por el contrario el autómata celular propuesto la rute es diferente pero que asemeja a la misma solución de Jeff siendo no tan uniforme.

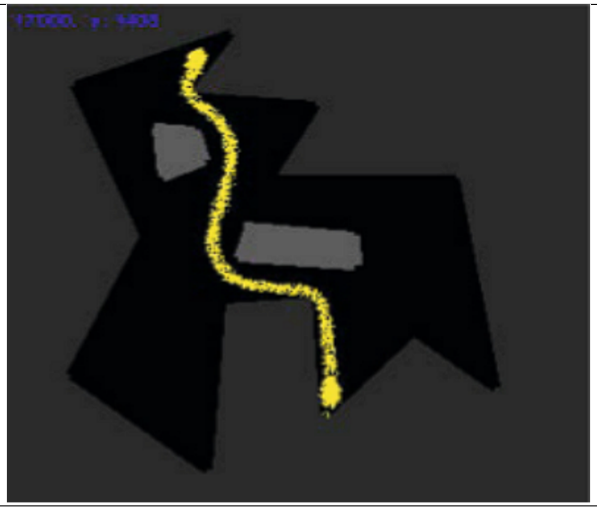
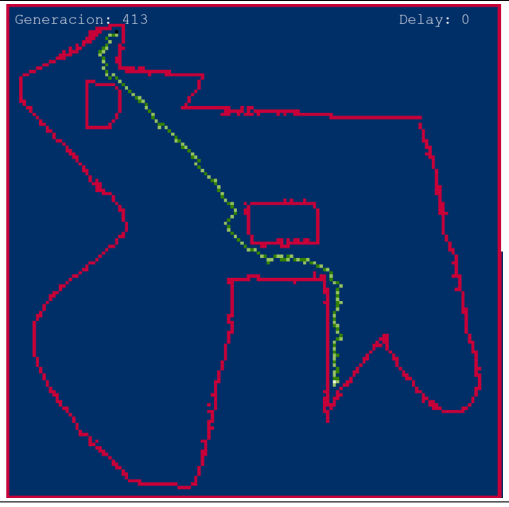
Red de Tokyo	
Modelo de Jeff [17]	**2DPyCAM(10)**
	

Tabla 34: Experimento de Jeff.

9.1.6.3. Comparación con el modelo de Gunji

Para Gunji a pesar de que de igual modo existe variación con respecto a las decisiones de generar rutas 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, provocando así que tenga un comportamiento similar al nuestro.

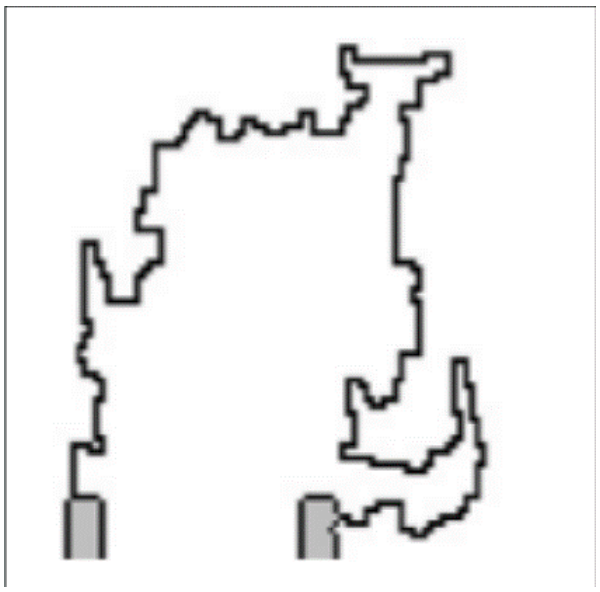
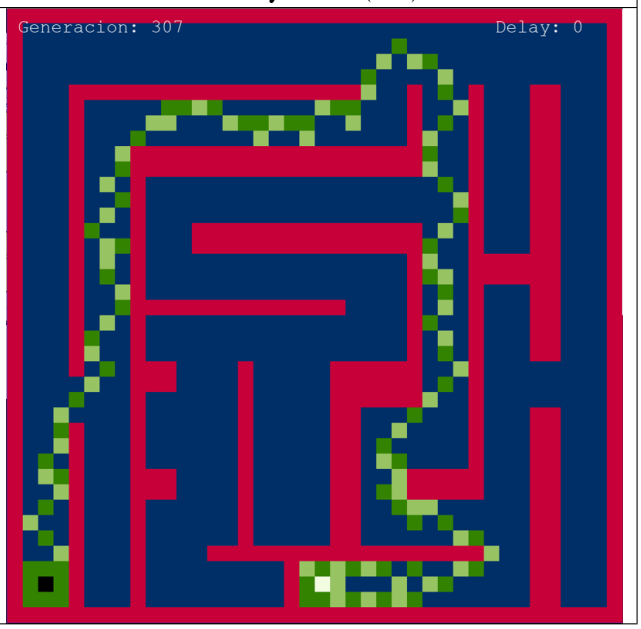
Laberinto	
Modelo de Gunji [12]	**2DPyCAM(10)**
	

Tabla 35: Comparación de solución de laberinto Gunji.

9.1.6.4. 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.

Modelo	Tipo de función	Vecindad o conectividad	Estados	Zd	Número de transiciones
Andrew 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
2DPyCAM(10)	Autómata celular	Vecindad de Moore y von Neumann	10	2D	10

Tabla 36: Resumen de comparaciones de modelos.

9.1.6.5. Comparación con el organismo real

En esta parte, vamos a emplear los experimentos comunes que se aplican al Physarum Polycephalum, es importante recordar el hecho de que el organismo real se extiende 1 cm cada hora.

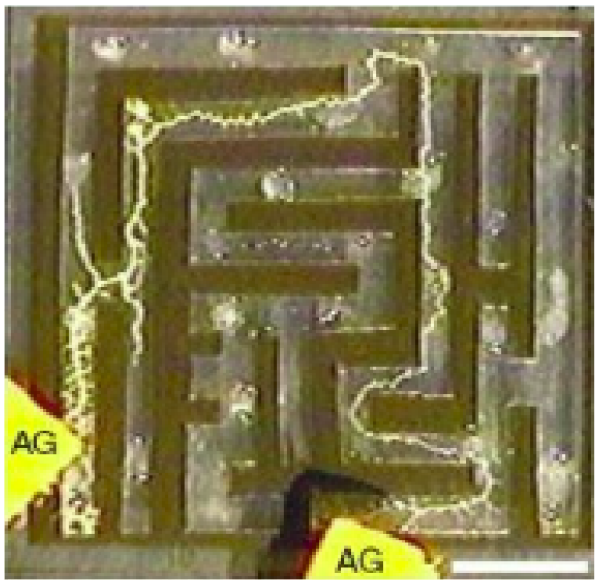
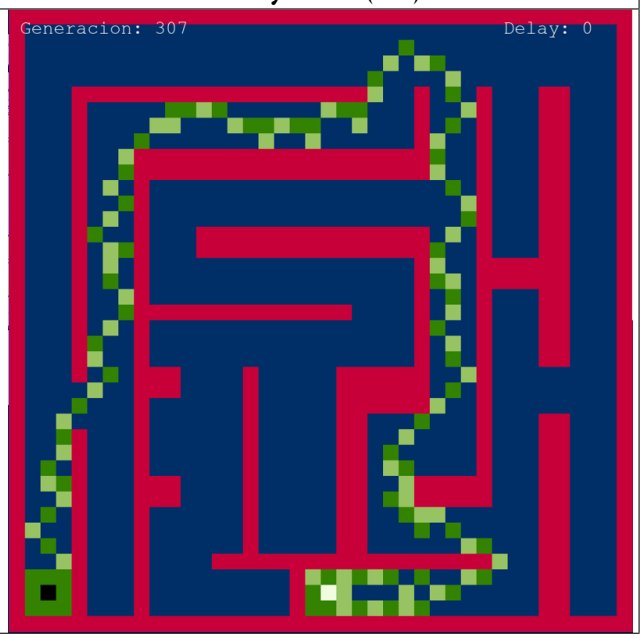
Laberinto	
Organismo real [4]	**2DPyCAM(10)**
	

Tabla 37: Comparación de solución de laberinto real.

Como dato a resaltar al Physarum le tomó aproximadamente 92 horas en realizar el trazo de la ruta para este experimento, la ruta creada de ambos hace ver la similitud que se

tiene con el modelo propuesto y el organismo real, también mencionar que no siempre el organismo real elegirá las misma solución, del mismo modo nuestro modelo no repetirá la misma solución, sin embargo, la ruta sera muy similar.

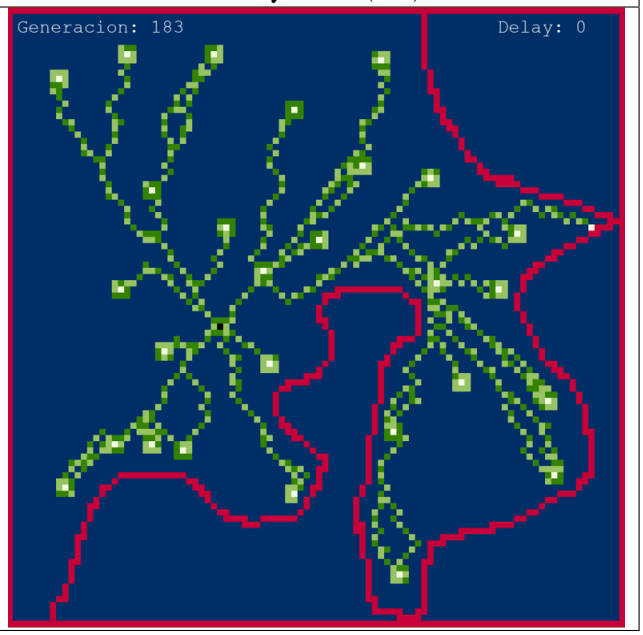
Red de Tokyo	
Organismo real [4]	**2DPyCAM(10)**
	

Tabla 38: Comparación de la Red de Tokyo con el organismo real.

Volviendo a retomar un experimento que se realizo 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.

9.2. Pruebas de la segunda versión del robot MemOso (Robot viajero)

9.2.1. Pruebas de los 4 movimientos básicos del robot

En esta sección de pruebas vamos a poder ver el comportamiento de nuestro robot MemOso en los cuatro movimientos básicos programados, los cuales son ir hacia adelante, ir hacia atrás, orientar a la derecha y orientar a la izquierda, es importante mencionar que estos movimientos fueron ajustados y programados con base a nuestra necesidad de que el robot se moviera 60 cm exactos al momento de que las instrucciones de mover hacia adelante y hacia atrás fueran ejecutadas, así mismo las instrucciones de orientación fueron programadas y ejecutadas para que se mantuviera en su posición inicial ajustando solo la orientación.

La intención de estos dos capítulos escritos a continuación es ver la diferencia del funcionamiento de los cuatro movimientos básicos usando y no usando un giroscopio, mencionar que al ser algo físico lo que se pondrá como evidencia son links directos a vídeos en la plataforma de YouTube para su consulta en cualquier momento.

En el caso de que se vea este documento en algún navegador web como Google Chrome o Microsoft Edge se recomienda dar clic derecho encima del link y dar a la opción abrir enlace(vinculo) en otra pestaña ya que si solo se da clic el vídeo sera reproducido en la pestaña actual, es decir, en la pestaña en donde se esta viendo el documento.

9.2.1.1. Pruebas sin el giroscopio implementado

9.2.1.1.1. Movimiento hacia adelante

En el siguiente enlace: `https://youtu.be/h2hYHFwTpR0` vamos a poder ver un pequeño vídeo en donde se demuestra el movimiento hacia adelante del robot MemOso sin usar el giroscopio. Si vemos detalladamente el vídeo podemos ver como el robot tiene una pequeña desviación hacia la derecha, que aunque es muy ligera, al seguir la ruta nuestro robot

debe de hacer varios movimientos y esto puede llegar a provocar grandes desviaciones provocando así un mal funcionamiento del robot y por lo tanto un mal seguimiento de la ruta.

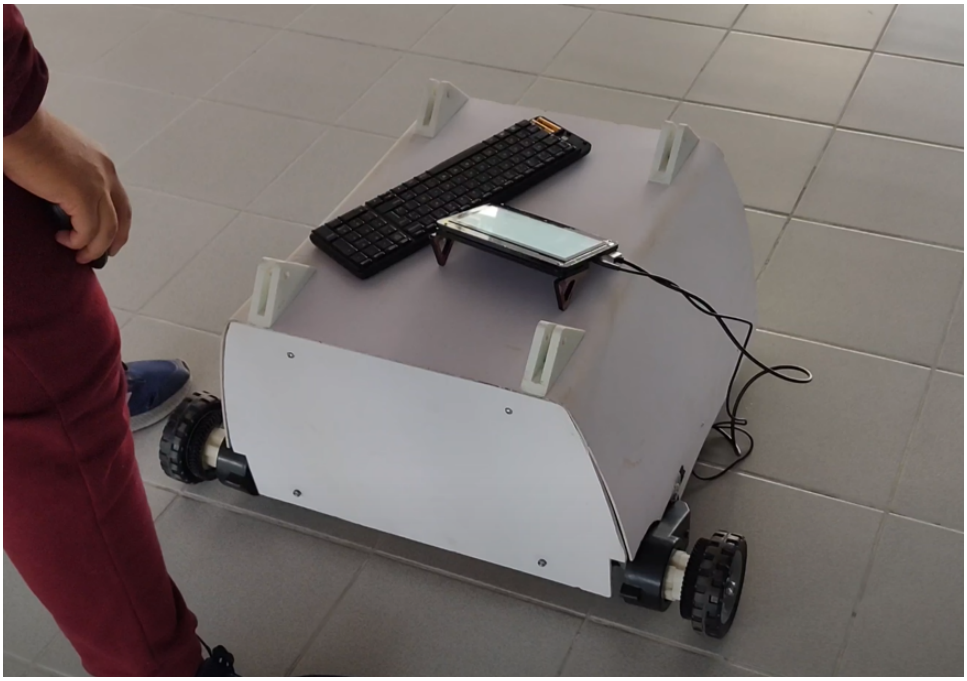


Figura 208: Inicio mover hacia adelante sin giroscopio.

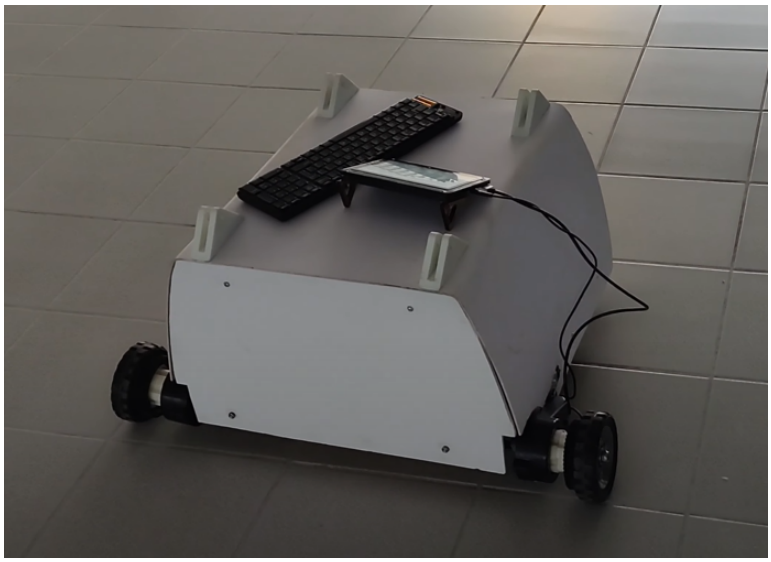


Figura 209: Fin mover hacia adelante sin giroscopio.

9.2.1.1.2. Movimiento hacia atrás

En el siguiente enlace: `https://youtu.be/saF9c7OMKW4` vamos a poder ver un pequeño

vídeo en donde se demuestra el movimiento hacia atrás del robot MemOso sin usar el giroscopio. Si vemos detalladamente el vídeo podemos ver como el robot tiene una pequeña desviación hacia la izquierda, que aunque es muy ligera, al seguir la ruta nuestro robot debe de hacer varios movimientos y esto puede llegar a provocar grandes desviaciones provocando así un mal funcionamiento del robot y por lo tanto un mal seguimiento de la ruta.

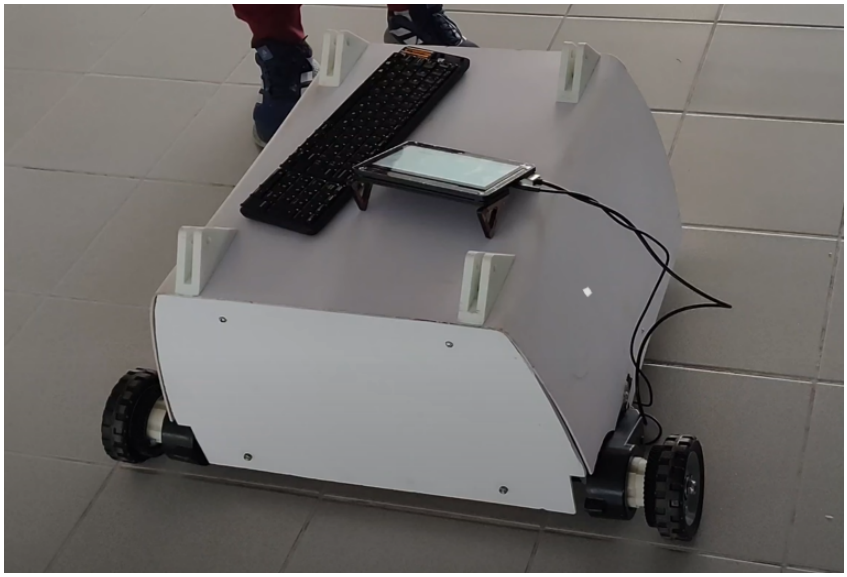


Figura 210: Inicio mover hacia atrás sin giroscopio.

Figura 211: Fin mover hacia atrás sin giroscopio.

9.2.1.1.3. Movimiento orientar hacia la derecha

En el siguiente enlace: `https://youtu.be/nkr0LcrPoDQ` vamos a poder ver un pequeño vídeo en donde se demuestra el movimiento de orientar hacia la derecha del robot MemOso sin usar el giroscopio (el robot esta mirando hacia nosotros y aunque parezca que esta yendo hacia la izquierda en realidad lo estamos viendo al revés). Si vemos detalladamente el vídeo podemos ver como el robot tiene una pequeña desviación hacia la izquierda, que aunque es muy ligera, al seguir la ruta nuestro robot debe de hacer varios movimientos y esto puede llegar a provocar grandes desviaciones provocando así un mal funcionamiento del robot y por lo tanto un mal seguimiento de la ruta.

Figura 212: Inicio orientar hacia la derecha sin giroscopio.

Figura 213: Fin orientar hacia la derecha sin giroscopio.

9.2.1.1.4. Movimiento orientar hacia la izquierda

En el siguiente enlace: `https://youtu.be/rcwxsyoY1XI` vamos a poder ver un pequeño vídeo en donde se demuestra el movimiento de orientar hacia la izquierda del robot MemOso sin usar el giroscopio, en este caso el robot esta viendo hacia enfrente. Si vemos detalladamente el vídeo podemos ver como el robot tiene una pequeña desviación hacia la

izquierda, que aunque es muy ligera, al seguir la ruta nuestro robot debe de hacer varios movimientos y esto puede llegar a provocar grandes desviaciones provocando así un mal funcionamiento del robot y por lo tanto un mal seguimiento de la ruta.

Figura 214: Inicio orientar hacia la izquierda sin giroscopio.

Figura 215: Fin orientar hacia la izquierda sin giroscopio.

9.2.1.2. Pruebas con el giroscopio implementado

9.2.1.2.1. Movimiento hacia adelante

En el siguiente enlace: https://youtu.be/bzqDK0DKxXg vamos a poder ver un pequeño vídeo en donde se demuestra el movimiento hacia adelante del robot MemOso ahora usando el giroscopio. Si vemos detalladamente el vídeo podemos ver como el robot tiene una pequeña desviación hacia la derecha, sin embargo, rápidamente el robot lo detecto y arreglo esa pequeña desviación evitando así los problemas anteriormente mencionados.

Figura 216: Inicio mover hacia adelante con giroscopio.

Figura 217: Fin mover hacia adelante con giroscopio.

9.2.1.2.2. Movimiento hacia atrás

En el siguiente enlace: `https://youtu.be/hfCTADtfuWE` vamos a poder ver un pequeño vídeo en donde se demuestra el movimiento hacia atrás del robot MemOso ahora usando el giroscopio. Si vemos detalladamente el vídeo podemos ver como el robot tiene una pequeña desviación hacia la izquierda, sin embargo, rápidamente el robot lo detecto y arreglo esa pequeña desviación evitando así los problemas anteriormente mencionados.

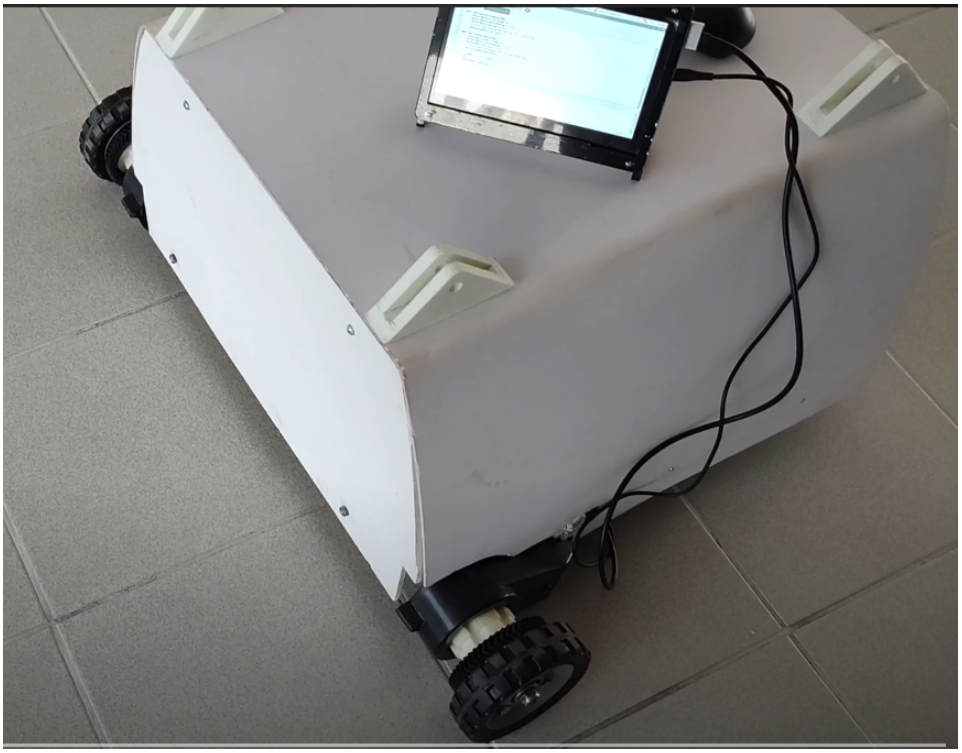


Figura 218: Inicio mover hacia atrás con giroscopio.

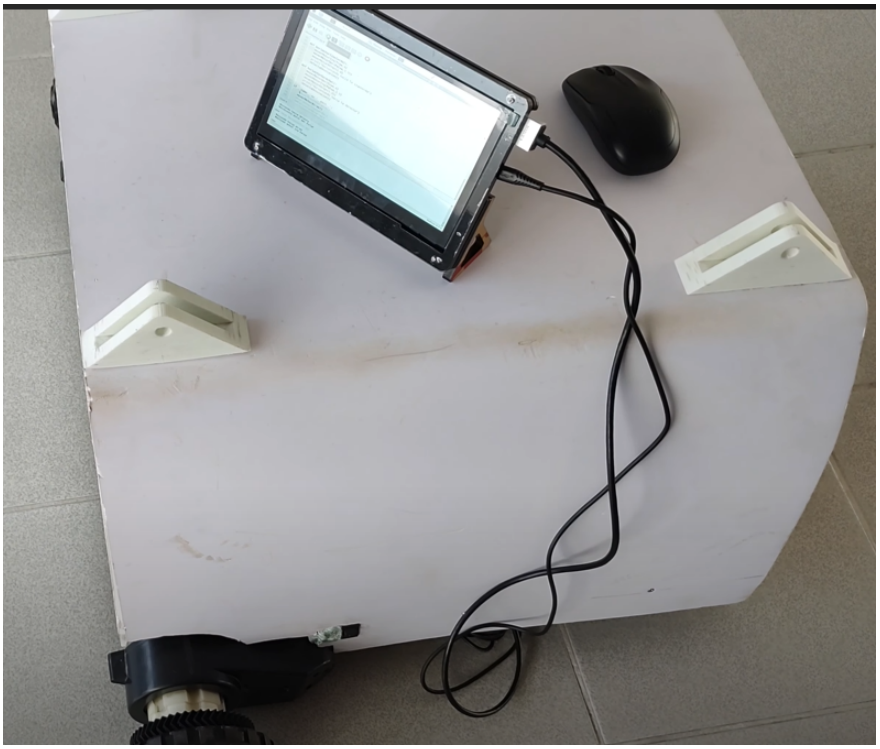


Figura 219: Fin mover hacia atrás con giroscopio.

9.2.1.2.3. Movimiento orientar hacia la derecha

En el siguiente enlace: https://youtu.be/S_8sQHBuDhc vamos a poder ver un pequeño vídeo en donde se demuestra el movimiento de orientar hacia la derecha del robot MemOso ahora usando el giroscopio. Si vemos detalladamente el vídeo podemos ver como el robot tiene una pequeña desviación hacia la derecha, sin embargo, rápidamente el robot lo detecto y arreglo esa pequeña desviación evitando así los problemas anteriormente mencionados.

Figura 220: Inicio orientar hacia la derecha con giroscopio.

Figura 221: Fin orientar hacia la derecha con giroscopio.

9.2.1.2.4. Movimiento orientar hacia la izquierda

En el siguiente enlace: https://youtu.be/gGvSmhtAx10 vamos a poder ver un pequeño vídeo en donde se demuestra el movimiento de orientar hacia la izquierda del robot MemOso ahora usando el giroscopio. Si vemos detalladamente el vídeo podemos ver como el robot tiene una pequeña desviación hacia la izquierda, sin embargo, rápidamente el robot lo detecto y arreglo esa pequeña desviación evitando así los problemas anteriormente mencionados.

Figura 222: Inicio orientar hacia la izquierda con giroscopio.

Figura 223: Fin orientar hacia la izquierda con giroscopio.

9.2.2. Pruebas del seguimiento de rutas

Como pudimos ver en la sección anterior el uso del giroscopio es de vital importancia debido a que esto permite a nuestro robot tener movimientos más precisos, es por eso que en esta sección de pruebas vamos a poder ver el comportamiento del robot MemOso siguiendo la ruta generada por el simulador del Physarum Polycephalum, en las siguientes dos secciones veremos el seguimiento de una ruta en donde no hay ningún obstáculo y en la otra veremos el seguimiento de una ruta en donde nuestro robot se encuentra con obstáculos en el camino.

9.2.2.1. Prueba 1

En el siguiente enlace: `https://youtu.be/UtnzcIzbCt8` vamos a poder ver un vídeo en donde se demuestra el seguimiento de una ruta creada por el simulador del Physarum Polycephalum, en donde el punto inicial es a la altura de la salida de la oficina de Becas y como punto final es la entrada de la oficina de capital humano, esto debido a que no queremos obstruir el paso en estas oficinas. En el vídeo podemos ver como el robot MemOso va avanzando con base en la ruta que podemos ver en la parte derecha inferior del vídeo y como un circulo de color gris nos va indicando como va avanzando el robot físicamente con respecto a la ruta, al final del vídeo vemos como el robot toma la posición de salida predefinida con anterioridad y es hasta ese punto que concluyo el viaje.

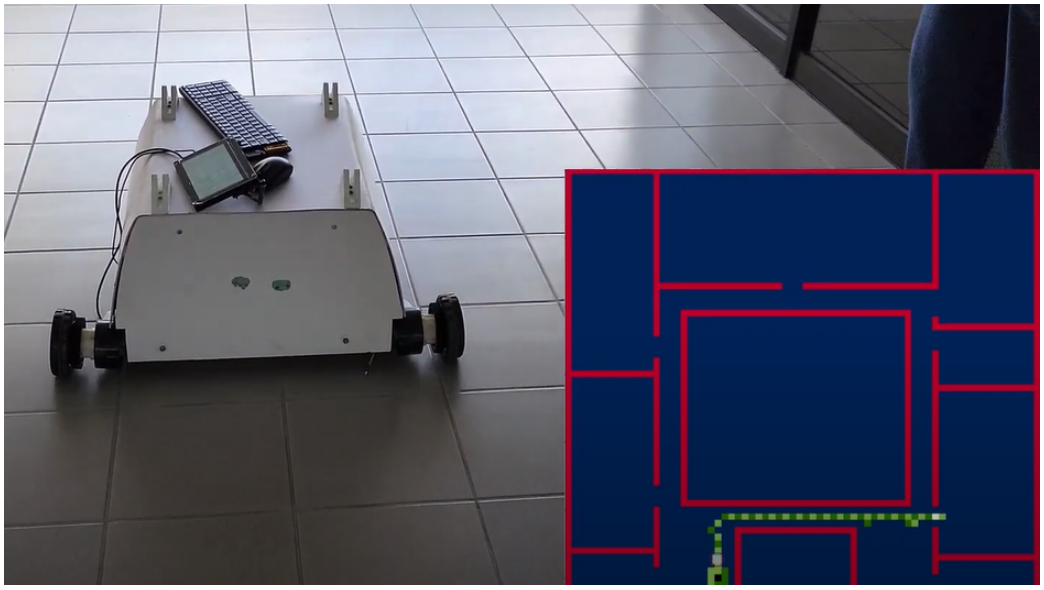


Figura 224: Inicio de la prueba 1 sin sensores.

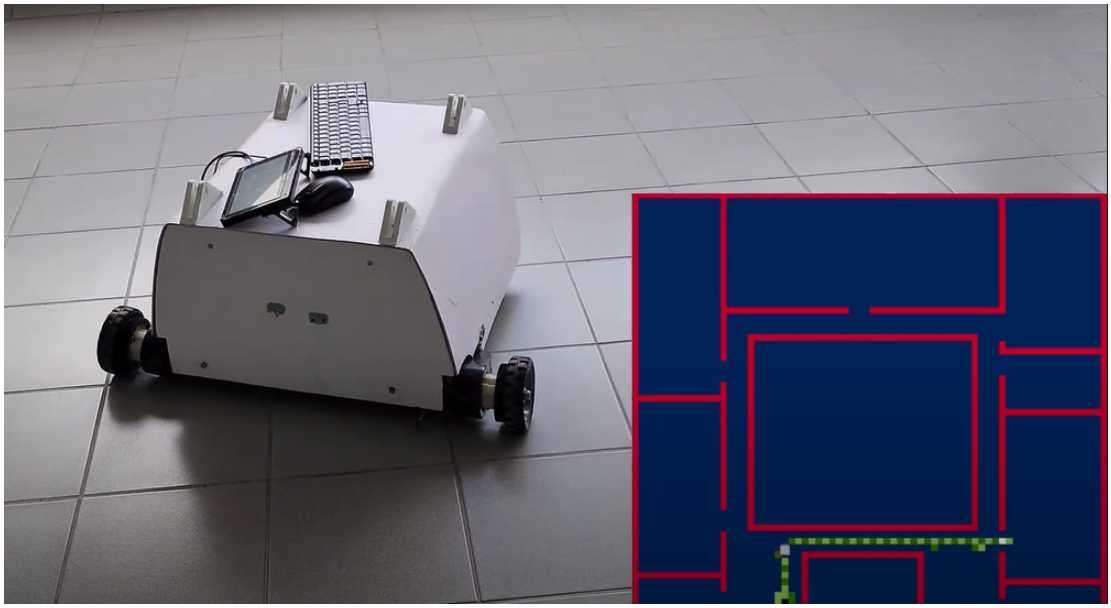


Figura 225: Transcurso de la prueba 1 sin sensores. Parte 1.

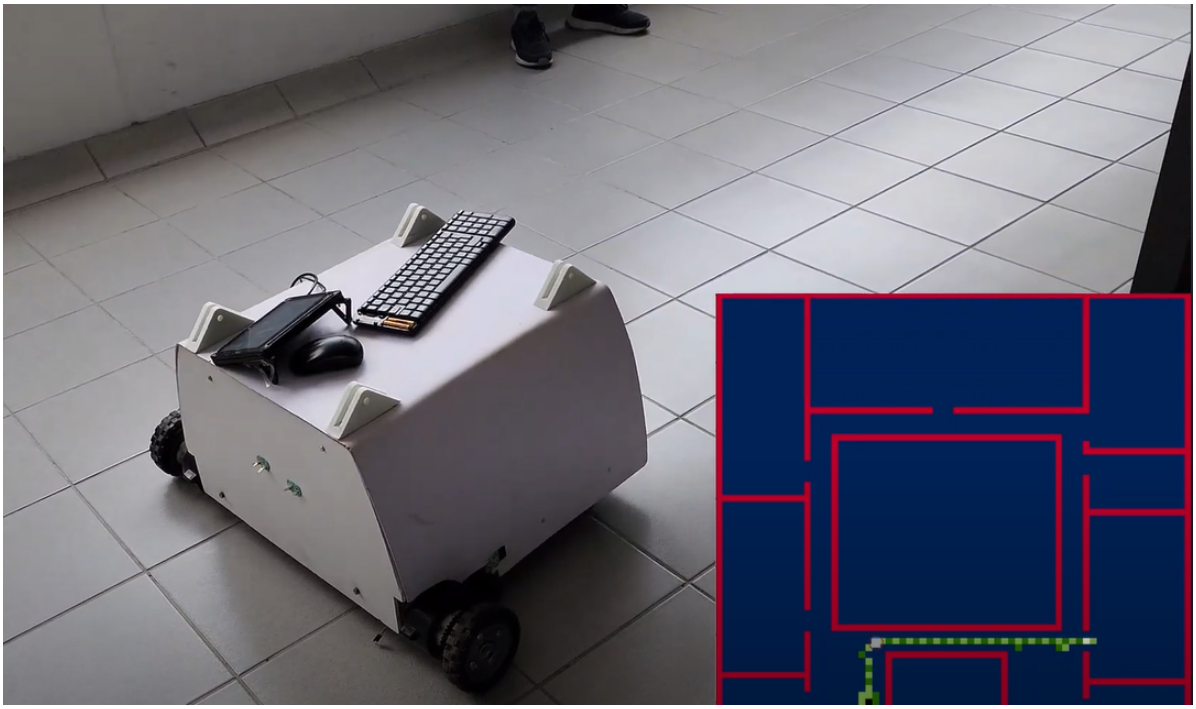


Figura 226: Transcurso de la prueba 1 sin sensores. Parte 2.

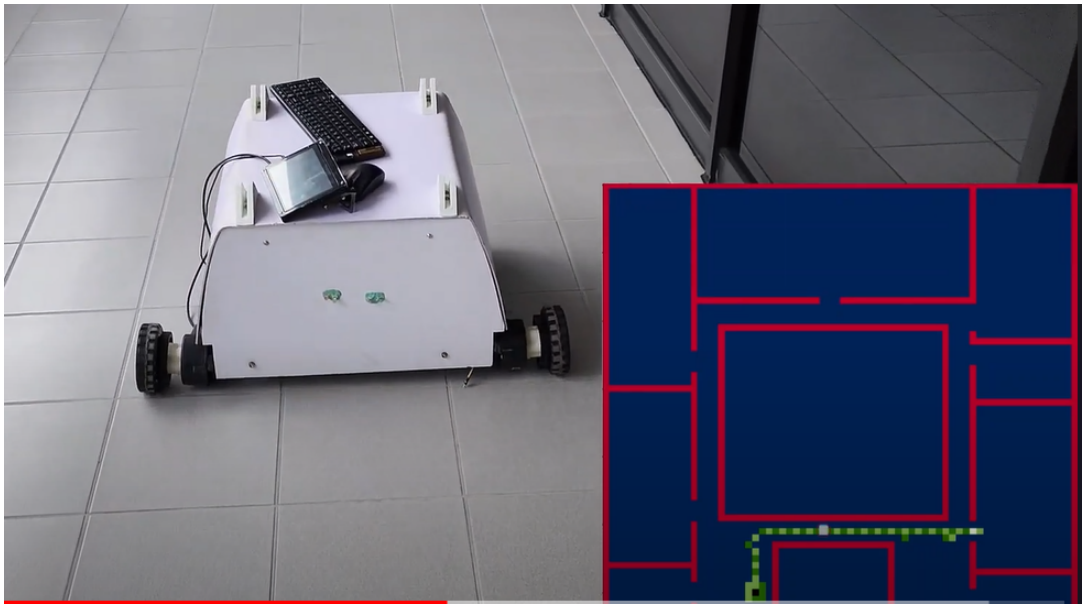


Figura 227: Transcurso de la prueba 1 sin sensores. Parte 3.

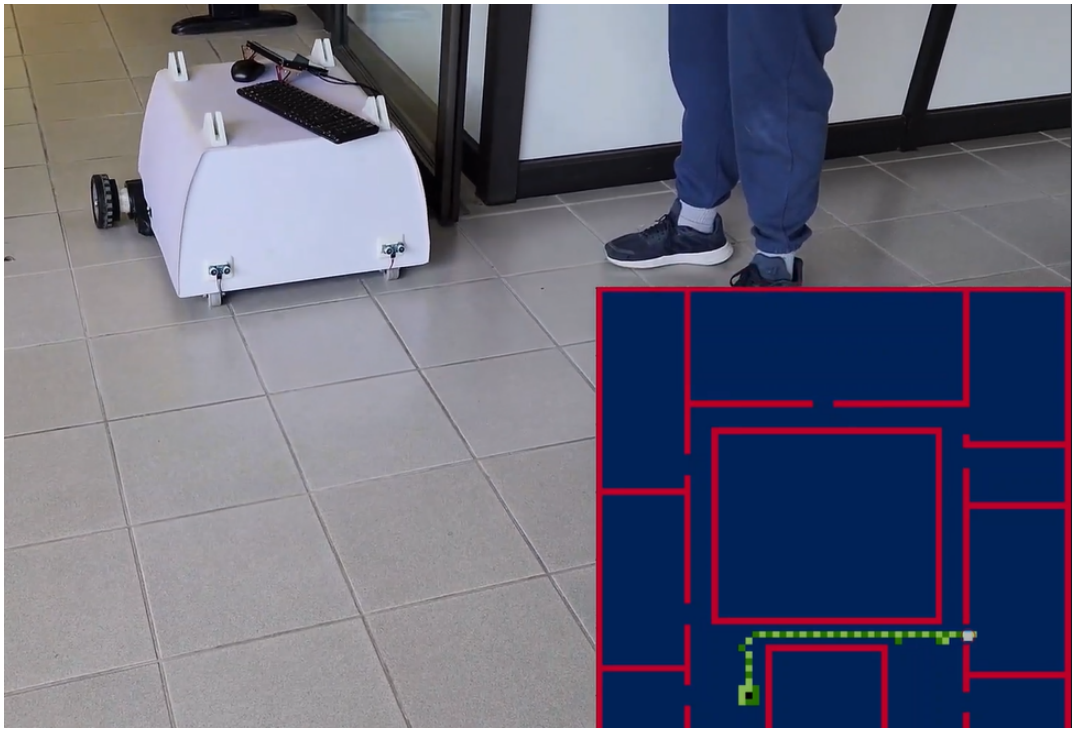


Figura 228: Fin de la prueba 1 sin sensores.

9.2.2.2. Prueba 2

En el siguiente enlace: `https://youtu.be/kbuw55FTBEU` vamos a poder ver un segundo vídeo en donde se demuestra el seguimiento de una ruta creada por el simulador del Physarum Polycephalum, en donde el punto inicial es a la altura de la salida de la oficina de Becas y como punto final tenemos que ahora es la altura de la entrada de la oficina de dirección, esto debido a que, igual que en la prueba anterior, no queremos obstruir el paso en estas oficinas. En el vídeo podemos ver como el robot MemOso va avanzando con base en la ruta que podemos ver en la parte derecha inferior del vídeo y como un circulo de color gris nos va indicando como va avanzando el robot físicamente con respecto a la ruta, al final del vídeo vemos como el robot toma la posición de salida predefinida con anterioridad y es hasta ese punto que concluyo el viaje.

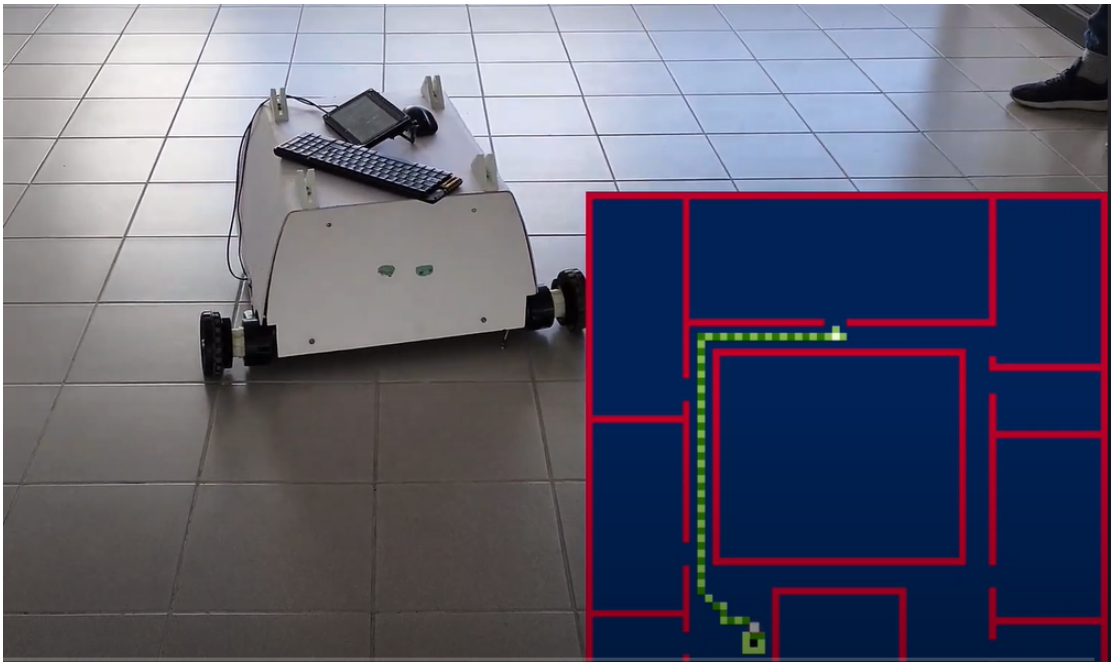


Figura 229: Inicio de la prueba 2 sin sensores.

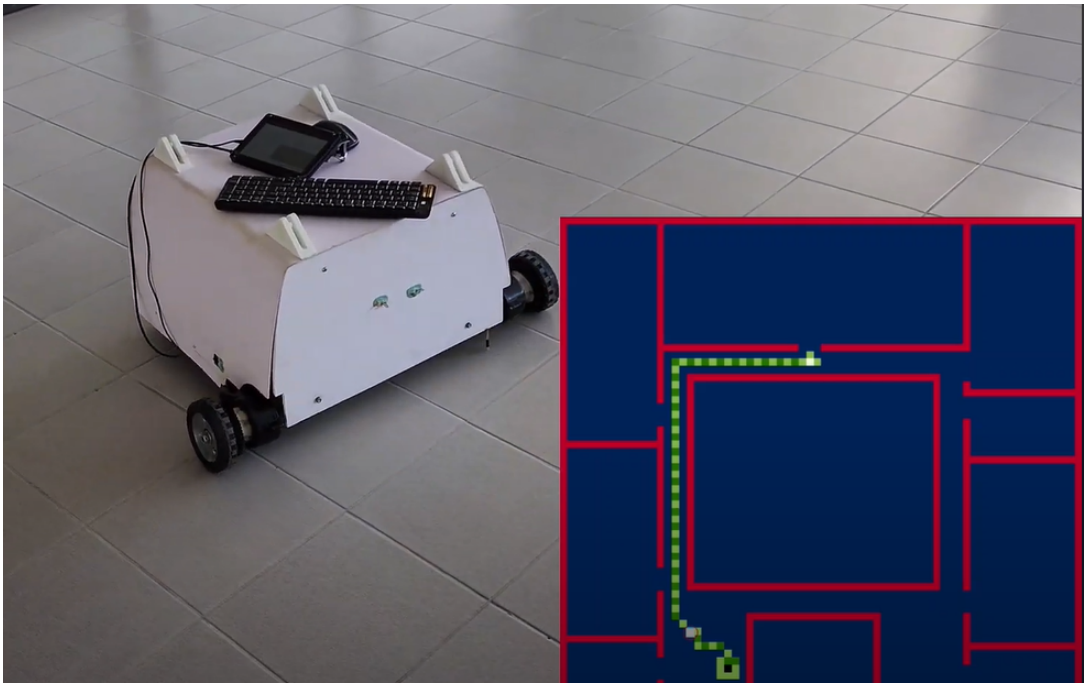


Figura 230: Transcurso de la prueba 2 sin sensores. Parte 1.

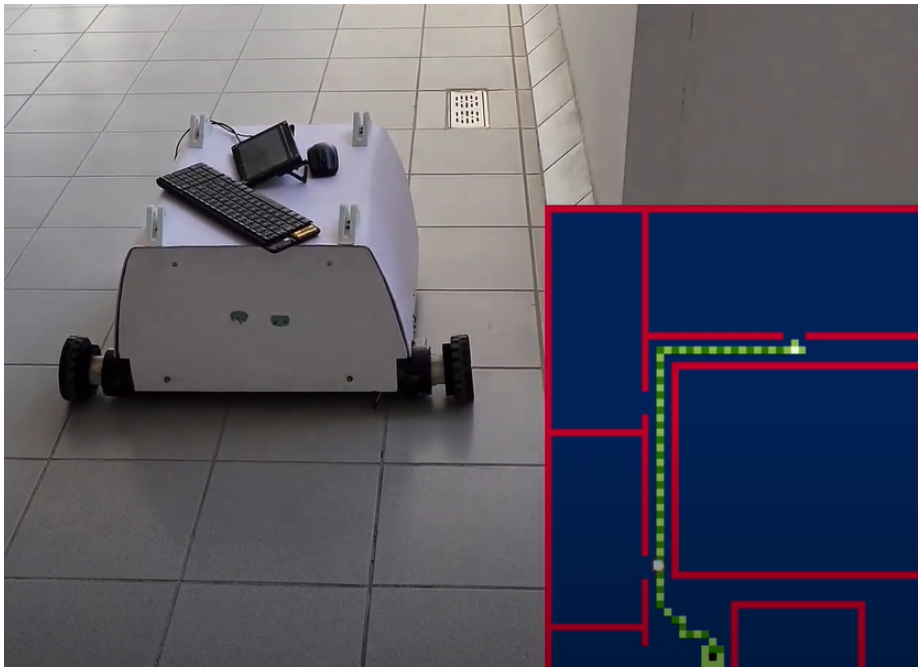


Figura 231: Transcurso de la prueba 2 sin sensores. Parte 2.

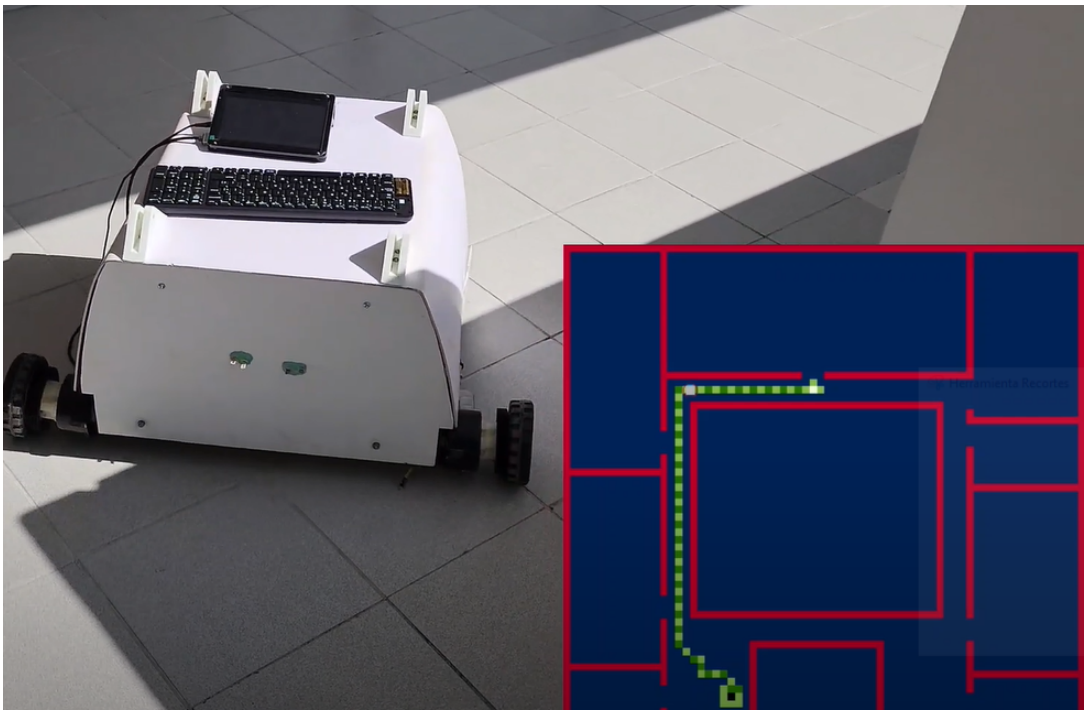


Figura 232: Transcurso de la prueba 2 sin sensores. Parte 3.

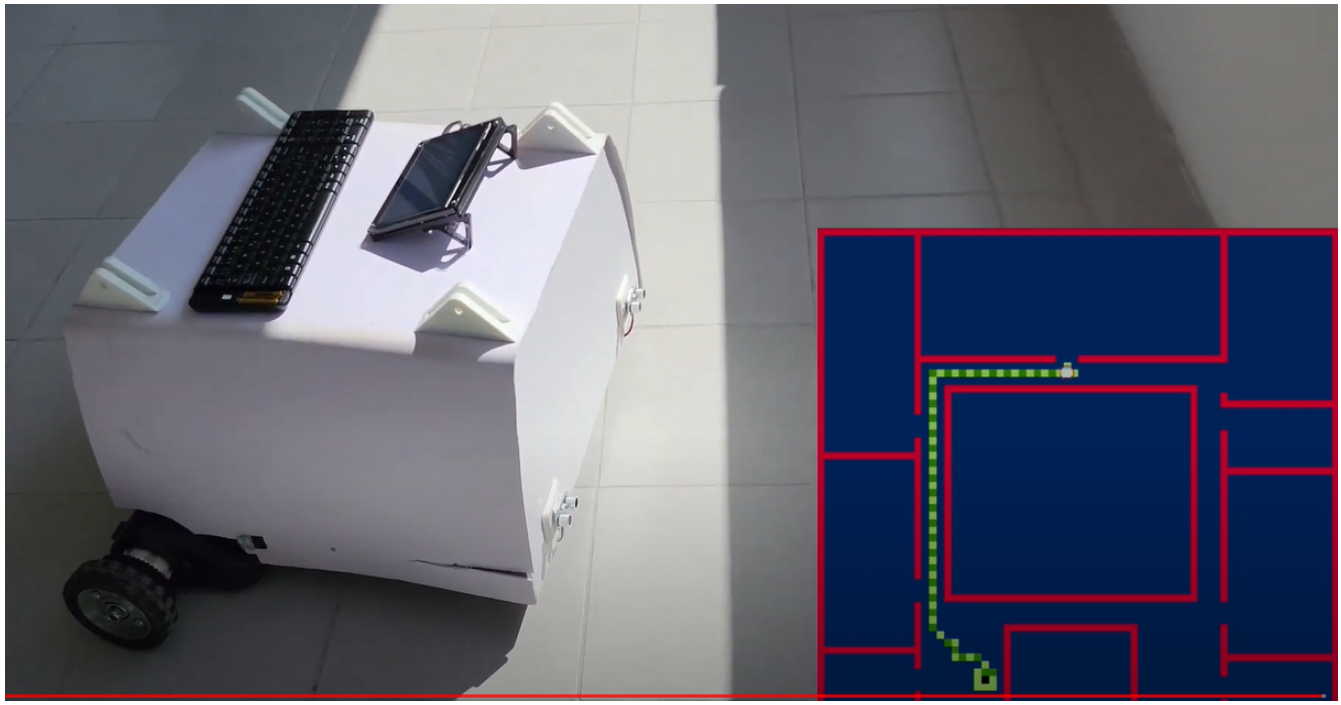


Figura 233: Fin de la prueba 2 sin sensores.

9.2.3. Prueba del seguimiento de rutas y de reconfiguración al encontrar un obstáculo

En el siguiente enlace: `https://youtu.be/reyo0fs854s` vamos a poder ver un vídeo en donde se demuestra el seguimiento de una ruta creada por el simulador del Physarum Polycephalum mientras funcionan los sensores, en donde el punto inicial es a la altura de la salida de la oficina de Becas y como punto final tenemos que ahora es la altura de la entrada de la oficina de dirección, esto debido a que, igual que en la prueba anterior, no queremos obstruir el paso en estas oficinas. En el vídeo podemos ver como el robot MemOso va avanzando con base en la ruta que podemos ver en la parte derecha inferior del vídeo y como un circulo de color gris nos va indicando como va avanzando el robot físicamente con respecto a la ruta, sin embargo, al usar sensores vemos como el robot tiene la capacidad de reconfigurar su ruta en tiempo real para evitar posibles colisiones, al final del vídeo vemos como el robot queda dos células antes de terminar la ruta como tal ya que como se menciono anterior no queríamos interrumpir a las personas que laboran en las oficinas.

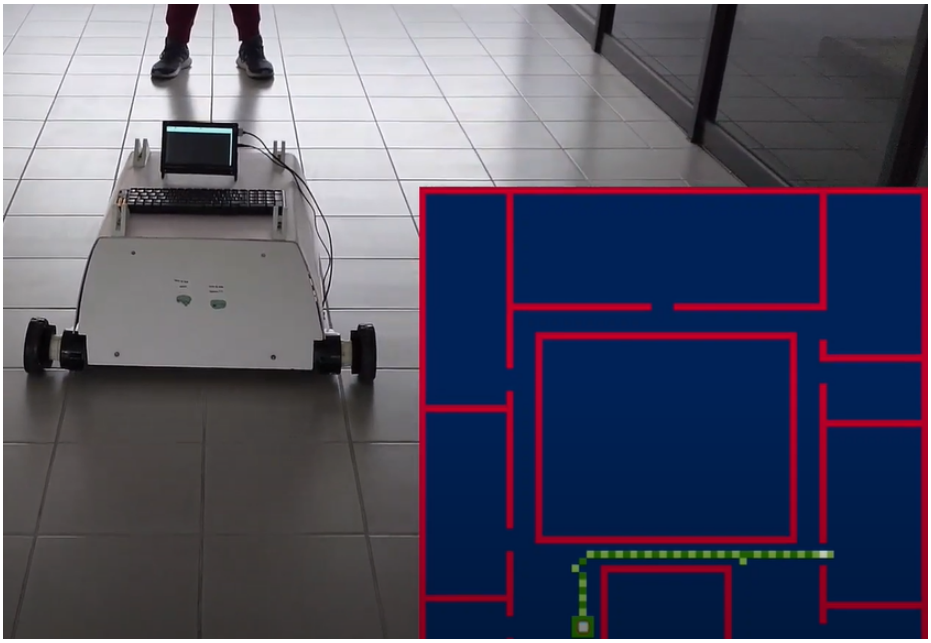


Figura 234: Inicio de la prueba con sensores.

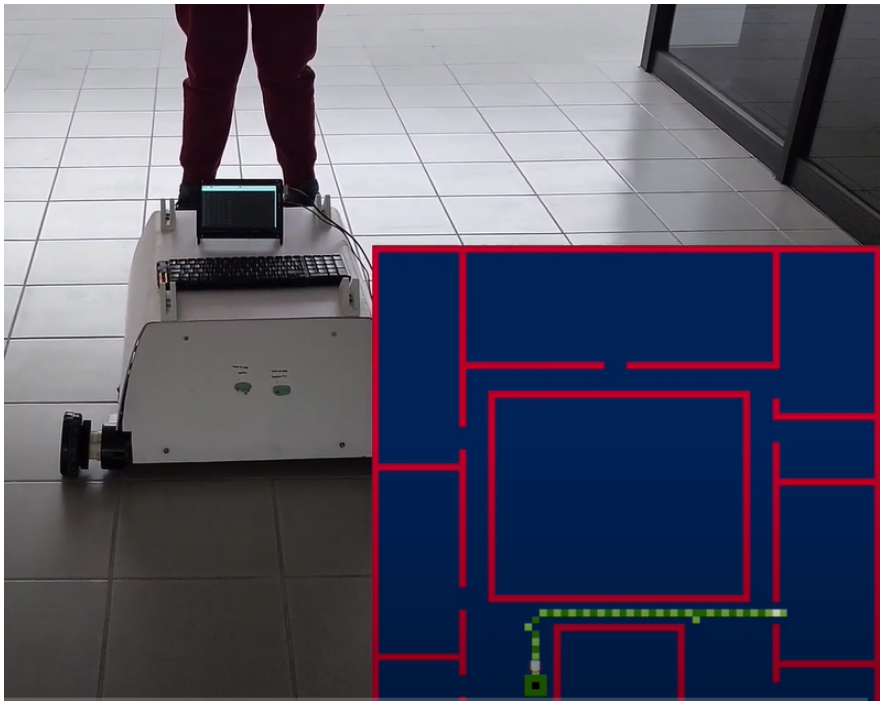


Figura 235: Transcurso de la prueba con sensores (Objeto encontrado). Parte 1.

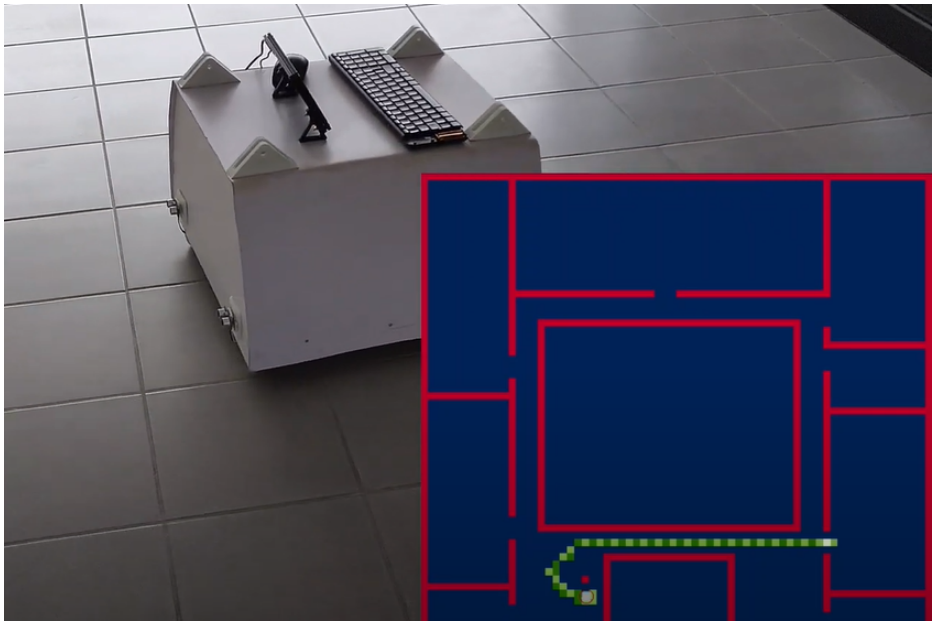


Figura 236: Transcurso de la prueba con sensores. Parte 2.

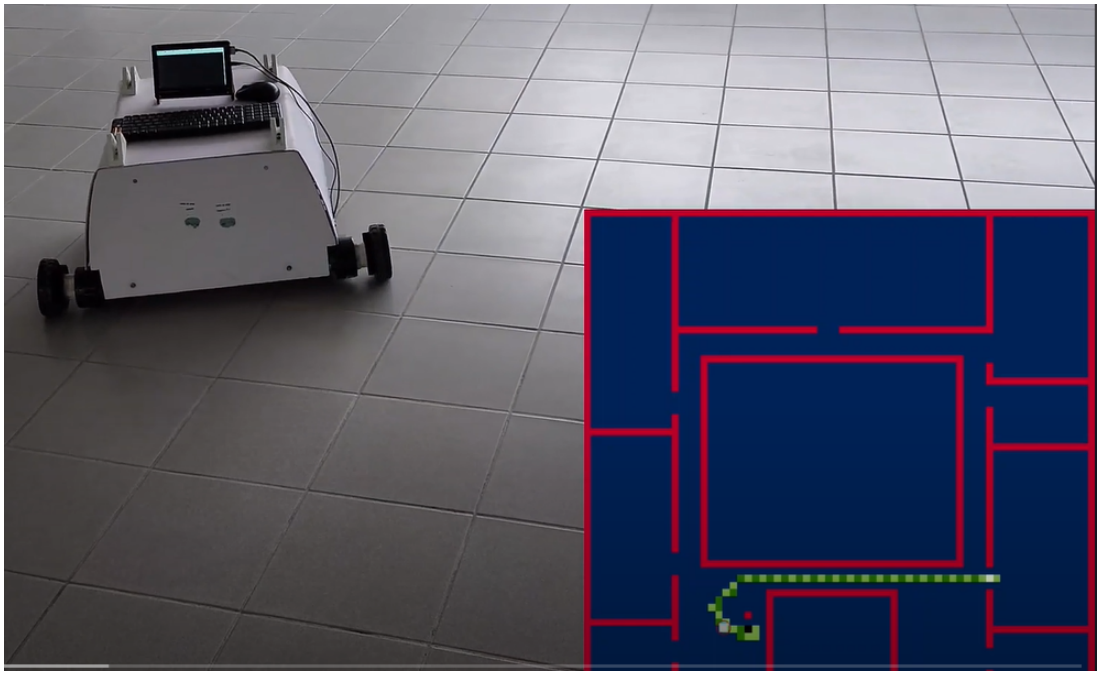


Figura 237: Transcurso de la prueba con sensores. Parte 3.

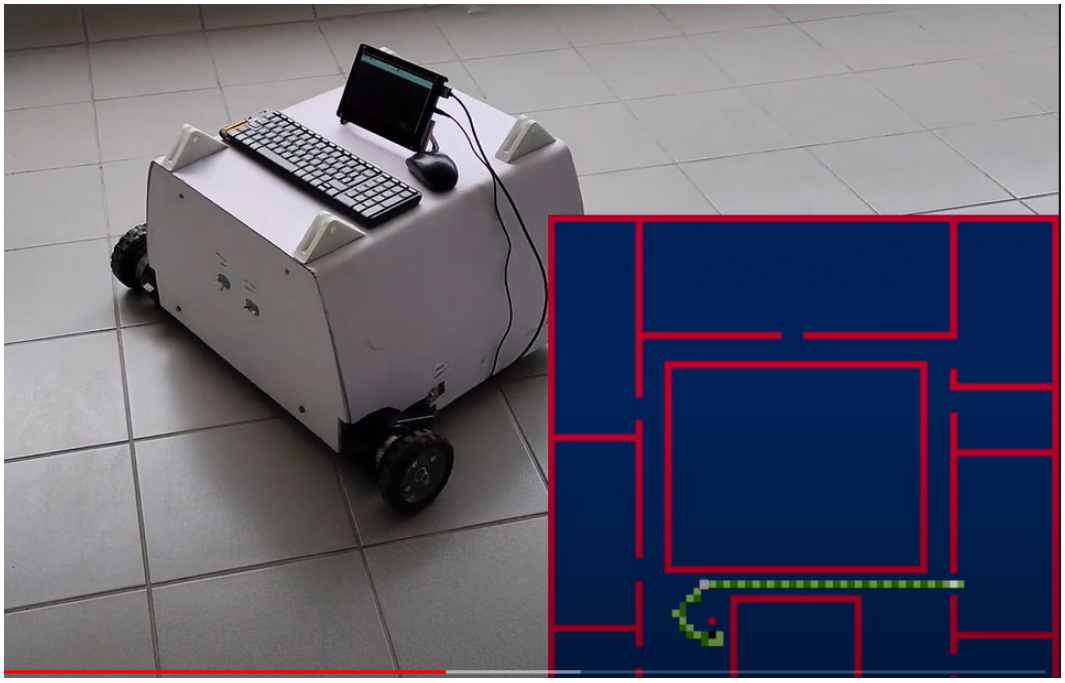


Figura 238: Transcurso de la prueba con sensores. Parte 4.

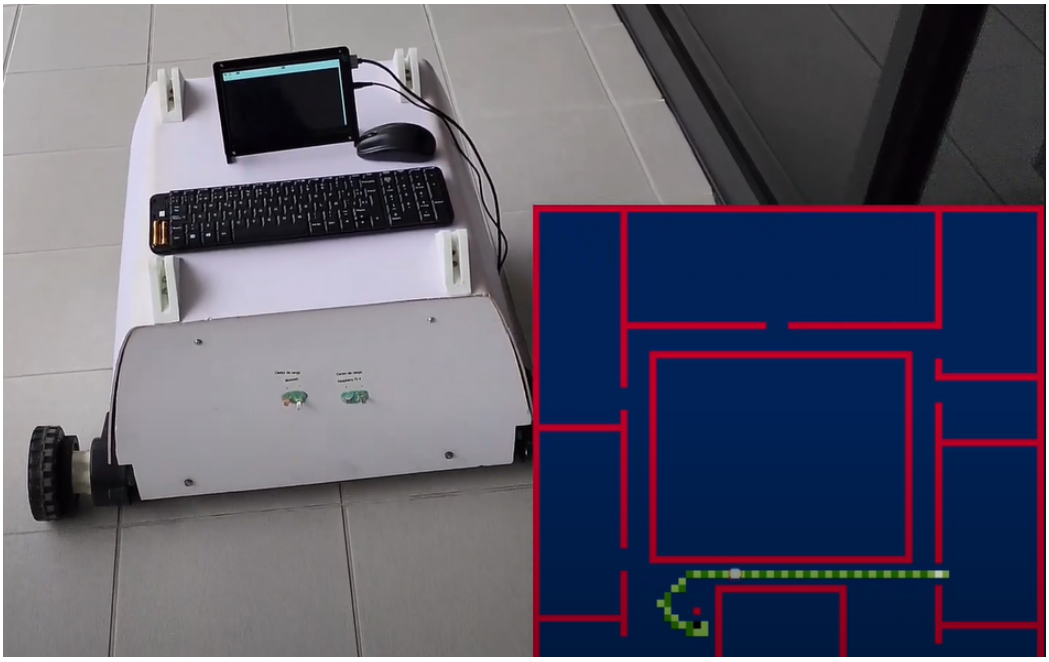


Figura 239: Transcurso de la prueba con sensores. Parte 5.

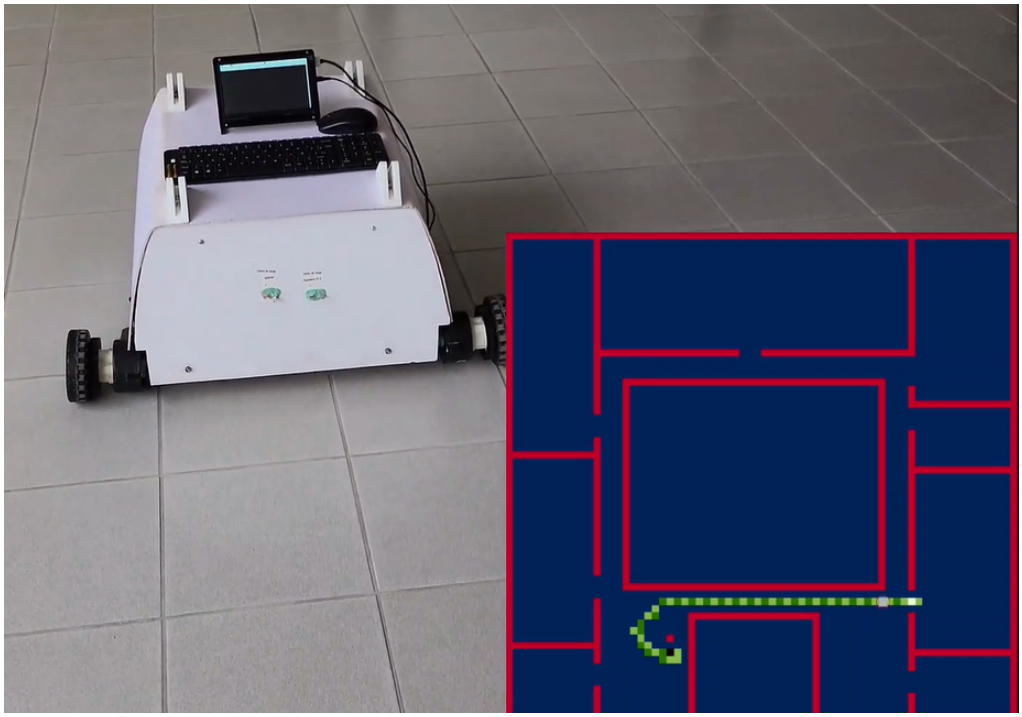


Figura 240: Fin de la prueba con sensores.

10. Conclusiones

Al modelar estos tipos de organismos necesitamos primeramente saber que propiedades tiene estos y cuales son las propiedades que deseamos obtener de estos, ya que, dependiendo de las propiedades que queramos rescatar el sistema trabajara de una u otra manera, provocando así diferentes resultados. En el caso del organismo Physarum Polycephalum al tener características que podríamos decirles simples, resulta interesante ya que podemos obtener comportamientos que pueden llegar a ser caóticos, como otros tipos de autómatas celulares como podría ser la Hormiga de Langton o el Juego de la Vida, que, con reglas simples, se han obtenido resultados no tan triviales o poco predecibles.

Como vimos a lo largo de la elaboración de este trabajo el organismo siempre es capaz de obtener al menos una solución y dicha solución difieren de otras, esto ayuda para poder ver como el organismo se concentra en vivir dependiendo de lo explorado, dichas soluciones aplicadas podemos verlas como rutas a seguir en un terreno, entonces al aplicar el modelado del Physarum Polycephalum a un robot, estenos puede dar una opción sobre un empleo de rutas poco predecibles o simplemente trayectorias aplicadas a este tipo de entidades, también al implementarse con sensores ultrasónicos para detectar obstáculos nos permite que el organismo pueda adaptarse en tiempo real y generar nuevas rutas, teniendo así un gran potencial de aplicación para este tipo de entidades.

Finalmente comentar que parte de este trabajo fue presentado en el evento Complexity frontiers organizado por la Red de Expertos en Sistemas Complejos (ROSC) del Instituto Politécnico Nacional, dicho evento tomo lugar en la Escuela Superior de Cómputo (ESCOM) del Instituto Politécnico Nacional, nuestra participación se puede ver en el siguiente enlace `https://www.facebook.com/escomipnmx/videos/527893502491053` a partir de las 2 horas con 2 minutos con 54 segundos, en dicho evento se expone principalmente el funcionamiento de nuestro modelado del Physarum Polycephalum junto a algunas prue-

bas realizadas en grandes tamaños en donde vemos el potencial de este modelo ya que como hemos mencionado en reiteradas ocasiones nos encuentra al menos una solución en un universo de soluciones, a continuación se agregan unas imágenes de la conferencia que se dio.

Figura 241: Foto de nuestra participación (1).

Figura 242: Foto de nuestra participación (2).

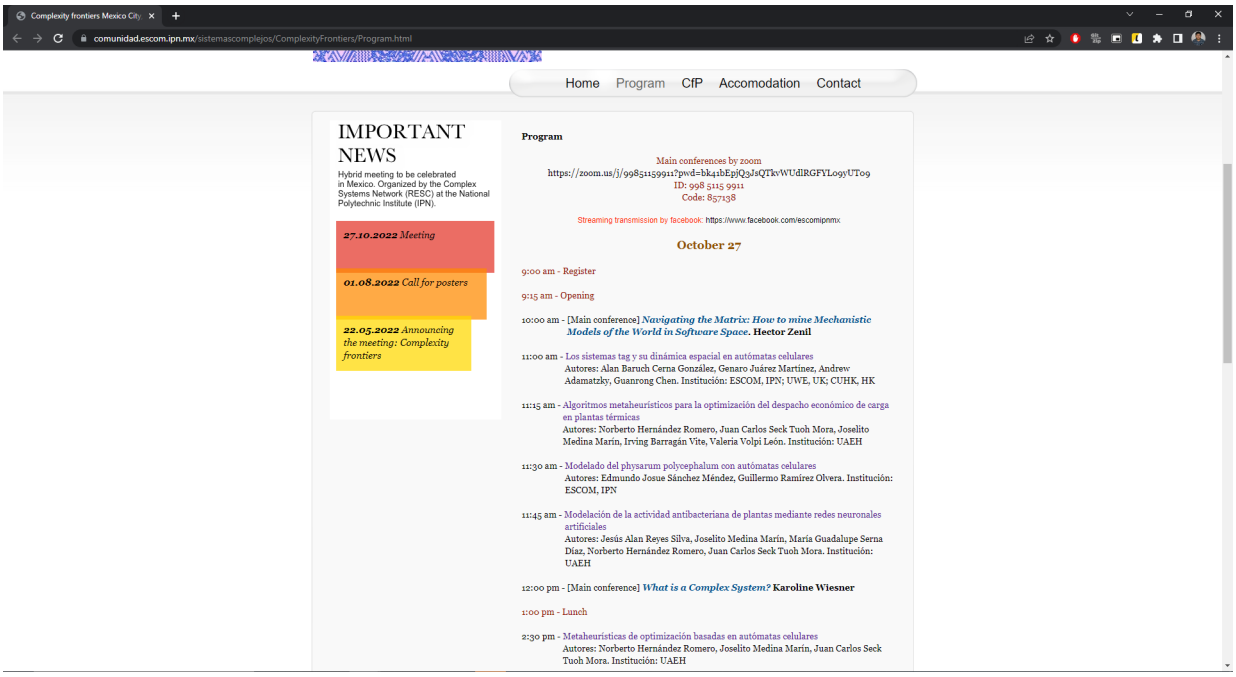


Figura 243: Horario de algunas de las conferencias que se impartieron, entre ellas la nuestra.

11. Trabajo a futuro

El modelo presentado se concentra en características de supervivencia del organismo, para trabajo a futuro se propone lo siguiente:

- Incorporar nuevos tipos de repelentes, como podrían ser sustancias, para así ver si el organismo puede adaptarse o no para saber el grado de rechazo del Physarum Polycephalum.
- Explorar la vecindad faltante, que es la vecindad hexagonal y ver que tan variable en cuanto a velocidad de generación de ruta o de densidad de estados difiere con la principal investigada en este documento.
- Usar métodos de paralelismo para una mejor optimización en el tiempo de generación de rutas.

Ahora veamos el posible trabajo futuro en el robot MemOso:

- Usar o implementar otro tipo de sensores para lograr una mayor precisión o en su defecto detectar todo tipo de materiales.
- Añadir un voltímetro para saber la cantidad de carga restante de las baterías.
- Usar un eje completo para las llantas traseras, para que ambas sean exactamente iguales.
- Cambiar el tipo de batería, usar baterías de Litio puede ser mejor a largo tiempo que usar baterías de ácido-plomo.
- Adaptar otro tipo de sistema de llantas para abrir la posibilidad de usar el robot a diferentes niveles de un edificio.

11.1. Repositorio

El presente trabajo terminal cuenta con un repositorio en la plataforma de GitHub, el enlace a se repositorio es el siguiente: `https://github.com/EdmundoSanchezM/TT2021-B013`, en este repositorio se encuentra todo el código utilizado y de cosas adicionales como los modelos 3D utilizados.

El repositorio esta dividido en 5 partes, el simulador del modelado, el código del robot el cual permite seguir las rutas, código del servicio web utilizado, vídeos de algunas pruebas realizadas y finalmente de la documentación del trabajo terminal incluyendo el manual de usuario del sistema.

12. Apéndice A

12.1. Código del simulador

En esta sección se muestra tres archivos de código principales de nuestro simulador, esto debido a que, si añadiéramos el código completo del simulador serian demasiadas paginas.

12.1.1. Main

```
1 ||
2 ||from pickle import TRUE
3 ||from simulador import Simulador
4 ||from Entropia import entropiaGrafica
5 ||import tkinter
6 ||from tkinter.ttk import Combobox
7 ||from tkinter import Tk, Frame, Label,messagebox
8 ||from PIL import ImageTk, Image
9 ||import threading
10 ||
11 ||def start():
12 ||    #------------------------------CREAR VENTANA
  ||       ---------------------------------
13 ||    root = tkinter.Tk()
14 ||    root.wm_title("Automata del Physarum Polycephalum")
15 ||    framet = tkinter.Frame(root)
16 ||    frameVecindad = tkinter.Frame(root)
17 ||    frameMainInformacion = tkinter.Frame(root)
18 ||    frameStart = tkinter.Frame(root)
19 ||    #------------------------------FUNCIONES
  ||       ---------------------------------
20 ||    global arregloInicial
21 ||    tamEspacio = tkinter.StringVar()
22 ||    arregloInicial = []
23 ||
24 ||    def seleccion(event):
25 ||        img_png = tkinter.PhotoImage(file = 'img/'+str(combo.get
  ||           ())+'.png')
26 ||        label_img = tkinter.Label(frameVecindad, image = img_png)
27 ||        label_img.grid(row = 2, column = 0)
28 ||        tkinter.mainloop()
29 ||
30 ||
31 ||
32 ||    def mensaje():
33 ||        mensaje="""
34 ||1) Comportamiento normal
35 ||2) Separacion de una celula en los repelentes
```

```
36 ||          """
37 ||          tkinter.messagebox.showinfo(message=mensaje, title="
   ||             Reglas")
38 ||
39 ||      def mensajeEntro():
40 ||          mensaje="""La grafica de la entropia puede tardar, no
   ||             cerrar el programa """
41 ||          tkinter.messagebox.showinfo(message=mensaje, title="
   ||             Reglas")
42 ||
43 ||      def iniciar():
44 ||          global arregloInicial
45 ||
46 ||          entro=''
47 ||          celulas=0
48 ||          try:
49 ||              celulas=int(entradaEspacio.get())
50 ||              regla=int(combo2.get())
51 ||              entro=combo3.get()
52 ||          except:
53 ||              tkinter.messagebox.showinfo(message="Ingrese
   ||                 correctamente los datos", title="Validacion")
54 ||              return
55 ||
56 ||
57 ||
58 ||
59 ||          if entro=='Si':
60 ||              entro=True
61 ||          elif entro=='No':
62 ||              entro=False
63 ||          pasa=0
64 ||
65 ||          if entro!='':
66 ||              if celulas >1:
67 ||                  if(regla>0 and regla <3):
68 ||                      pasa=1
69 ||                      if combo.get()=="Neumann":
70 ||                          Simulador(celulas,regla,'neumann',entro)
71 ||                      elif combo.get()=="Moore":
72 ||                          Simulador(celulas,regla,'moore',entro)
73 ||                      else:
74 ||                          tkinter.messagebox.showinfo(message="
   ||                             Escoga una vecinadad", title="
   ||                             Validacion")
75 ||                  else:
76 ||                      tkinter.messagebox.showinfo(message="No
   ||                         existe esa regla", title="Validacion")
77 ||              else:
78 ||                      tkinter.messagebox.showinfo(message="No se
   ||                         puede trabajar con esa cantidad de celulas
   ||                         ", title="Validacion")
79 ||          else:
80 ||              tkinter.messagebox.showinfo(message="Seleccionar si
   ||                 desea entropia o no", title="Validacion")
```

```
81 ||
82 ||          if pasa and entro:
83 ||              print(entro)
84 ||              if combo.get()=="Neumann":
85 ||                  entropiaGrafica('neumann',celulas,celulas)
86 ||              elif combo.get()=="Moore":
87 ||                  entropiaGrafica('moore',celulas,celulas)
88 ||
89 ||      #------------------------------CREAR INTERFAZ
   ||         ---------------------------------
90 ||
91 ||      titulo = tkinter.Label(framet, text="Automata del Physarum
   ||         Polycephalum",font=("times new roman", 24))
92 ||      titulo.pack(side="top")
93 ||
94 ||       #------------------------------VECINDAD
   ||          ---------------------------------
95 ||      tittleCondicionUsuario = tkinter.Label(frameVecindad, text =
   ||         "Vecindad" ,font=("times new roman", 18))
96 ||      tittleCondicionUsuario.grid(row = 0, column = 0,columnspan=2)
97 ||
98 ||      combo = tkinter.ttk.Combobox(frameVecindad,state="readonly",
   ||         values=["Moore", "Neumann"])
99 ||      combo.bind("<<ComboboxSelected>>", seleccion)
100||      combo.grid(row = 1, column = 0)
101||
102||
103||
104||#------------------------------CONDICION INICIAL USUARIO
   ||    ---------------------------------
105||      tittleCondicionUsuario = tkinter.Label(frameMainInformacion,
   ||         text = "Condiciones iniciales" ,font=("times new roman",
   ||         18))
106||      tittleCondicionUsuario.grid(row = 0, column = 0,columnspan=2)
107||
108||      tamanioEspacio = tkinter.Label(frameMainInformacion, text = "
   ||         Numero de celdas por lado: " , font=("times new roman",
   ||         14))
109||      tamanioEspacio.grid(row=1, column=0)
110||      entradaEspacio = tkinter.Entry(frameMainInformacion,font=("
   ||         times new roman", 14), textvariable = tamEspacio)
111||      entradaEspacio.grid(row=1, column=1)
112||
113||      tamanioRegla = tkinter.Label(frameMainInformacion, text = "
   ||         Regla: " , font=("times new roman", 14))
114||      tamanioRegla.grid(row=2, column=0)
115||      combo2 = tkinter.ttk.Combobox(frameMainInformacion,state="
   ||         readonly",values=["1", "2"])
116||      combo2.bind("<<ComboboxSelected>>")
117||      combo2.grid(row = 2, column = 1)
118||      buttonHelp = tkinter.Button(frameMainInformacion, text = "
   ||         Reglas", command=mensaje ,font=("times new roman", 14))
119||      buttonHelp.grid(row=2, column=2)
120||
121||      pregunta = tkinter.Label(frameMainInformacion, text = "
   ||         Mostrar entropia?" , font=("times new roman", 14))
```

```
122 ||     pregunta.grid(row=3, column=0)
123 ||
124 ||
125 ||     combo3 = tkinter.ttk.Combobox(frameMainInformacion,state="
    ||        readonly",values=["Si", "No"])
126 ||     combo3.bind("<<ComboboxSelected>>")
127 ||     combo3.grid(row = 3, column = 1)
128 ||     buttonEntro = tkinter.Button(frameMainInformacion, text = "
    ||        Info", command=mensajeEntro ,font=("times new roman", 14))
129 ||     buttonEntro.grid(row=3, column=2)
130 ||
131 ||
132 ||     #------------------------------BOTON MAESTRO
    ||        ---------------------------------
133 ||
134 ||     buttonStart = tkinter.Button(frameStart, text = "Start",
    ||        command=iniciar ,font=("times new roman", 14))
135 ||     buttonStart.pack(side="top")
136 ||
137 ||     framet.grid(row=0,column=0,columnspan=1)
138 ||     frameVecindad.grid(row=1,column=0)
139 ||     frameMainInformacion.grid(row=2,column=0)
140 ||     frameStart.grid(row=3,column=0)
141 ||
142 ||     tkinter.mainloop()
143 ||
144 ||
145 ||if __name__ == '__main__':
146 ||     start()
```

12.1.2. Moore

```
  1 ||from random import randint
  2 ||import numpy as np
  3 ||import pygame
  4 ||import math
  5 ||import Entropia
  6 ||import threading
  7 ||class Moore:
  8 ||     def __init__(self,nxC,nyC,pauseExect,gameState,mins,aux,regla
    ||        ,val,cont,count4,count5,count7,count8,dimCW,dimCH,screen,
    ||        Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,entropiaGrafica,generacion):
  9 ||          self.nxC=nxC
 10 ||          self.nyC=nyC
 11 ||          self.pauseExect=pauseExect
 12 ||          self.gameState=gameState
 13 ||          self.mins=mins
 14 ||          self.aux=aux
 15 ||          self.regla=regla
 16 ||          self.val=val
 17 ||          self.cont=cont
 18 ||          self.count4=count4
 19 ||          self.count5=count5
 20 ||          self.count7=count7
 21 ||          self.count8=count8
```

```
22 ||         self.dimCW=dimCW
23 ||         self.dimCH=dimCH
24 ||         self.screen=screen
25 ||         self.entropiaGrafica=entropiaGrafica
26 ||         self.generacion=generacion
27 ||
28 ||
29 ||         self.Q0 = Q0     # Campo libre
30 ||         self.Q1 = Q1     # Nutriente no encontrado
31 ||         self.Q2 = Q2     # Repelente
32 ||         self.Q3 = Q3     # Punto inicial
33 ||         self.Q4 = Q4     # Gel en concentracion
34 ||         self.Q5 = Q5     # Gel con compuesto
35 ||         self.Q6 = Q6     # Nutriente hallado
36 ||         self.Q7 = Q7     # Expansion del Physarum
37 ||         self.Q8 = Q8     # Gel sin compuesto
38 ||         self.Q9 = Q0     #q0+algo para la regla 2
39 ||
40 ||
41 ||         pass
42 ||
43 ||     def randomOffset(self):
44 ||         return randint(0, ((7 + 1) - 0)) + 0
45 ||
46 ||
47 ||
48 ||     def getMoore(self,n_neigh, crit):
49 ||         x = 0
50 ||         if(n_neigh[0] == crit):
51 ||             x +=1
52 ||         if(n_neigh[1] == crit):
53 ||             x +=1
54 ||         if(n_neigh[2] == crit):
55 ||             x +=1
56 ||         if(n_neigh[3] == crit):
57 ||             x +=1
58 ||         if(n_neigh[4] == crit):
59 ||             x +=1
60 ||         if(n_neigh[5] == crit):
61 ||             x +=1
62 ||         if(n_neigh[6] == crit):
63 ||             x +=1
64 ||         if(n_neigh[7] == crit):
65 ||             x +=1
66 ||         return x
67 ||
68 ||     def MooreOffset(self,n_neigh, crit, orien):
69 ||         if(n_neigh[6] == crit and orien ==1):
70 ||             return True
71 ||         if(n_neigh[5] == crit and orien ==2):
72 ||             return True
73 ||         if(n_neigh[3] == crit and orien ==3):
74 ||             return True
75 ||         if(n_neigh[0] == crit and orien ==4):
76 ||             return True
```

```
77 ||          if(n_neigh[1] == crit and orien ==5):
78 ||              return True
79 ||          if(n_neigh[7] == crit and orien ==8):
80 ||              return True
81 ||          if(n_neigh[4] == crit and orien ==7):
82 ||              return True
83 ||          if(n_neigh[2] == crit and orien ==6):
84 ||              return True
85 ||          return False
86 ||
87 ||      def depfun(self,estado, x, y, n_neigh,mins):
88 ||          if(estado == n_neigh[6] and mins[x][y+1] == 5):
89 ||              return True
90 ||          if(estado == n_neigh[5] and mins[x-1][y+1] == 6):
91 ||              return True
92 ||          if(estado == n_neigh[3]and mins[x-1][y] == 7):
93 ||              return True
94 ||          if(estado == n_neigh[0] and mins[x-1][y-1] == 8):
95 ||              return True
96 ||          if(estado == n_neigh[1] and mins[x][y-1] == 1):
97 ||              return True
98 ||          if(estado == n_neigh[2]and mins[x+1][y-1] == 2):
99 ||              return True
100 ||         if(estado == n_neigh[4] and mins[x+1][y] == 3):
101 ||             return True
102 ||         if(estado == n_neigh[7] and mins[x+1][y+1] == 4):
103 ||             return True
104 ||         return False
105 ||
106 ||     def getNeibors(self,actual,n_neigh):
107 ||         auxi=[]
108 ||         for val in n_neigh:
109 ||             auxi.append(val)
110 ||         auxi.insert(4,actual)
111 ||         return auxi
112 ||
113 ||     def calcularVecindadValor(self,arre,file):
114 ||         total=0
115 ||         x=0
116 ||         for x in range(len(arre)):
117 ||             if arre[x]!=0 and arre[x]!=2:
118 ||                 total+=2**x
119 ||         file.write(str(total)+'\n')
120 ||
121 ||     def movimiento(self):
122 ||         file= open('entropia/EntroInfo.txt','a')
123 ||         for x in range(0,self.nxC):
124 ||                 for y in range(0,self.nyC):
125 ||                     if(self.gameState[x][y] == 3):
126 ||                         self.val[x][y] = self.generacion+1
127 ||         for x in range(0, self.nxC):
128 ||             for y in range(0, self.nyC):
129 ||                 if not self.pauseExect:
130 ||                     n_neigh = np.array([self.gameState[(x-1) %
    ||                        self.nxC, (y-1) % self.nyC],
```

```
131                                         self.gameState[(x) % self
                                               .nxC, (y-1) % self.nyC
                                               ],
132                                         self.gameState[(x+1) %
                                               self.nxC, (y-1) % self
                                               .nyC],
133                                         self.gameState[(x-1) %
                                               self.nxC, (y) % self.
                                               nyC],
134                                         self.gameState[(x+1) %
                                               self.nxC, (y) % self.
                                               nyC],
135                                         self.gameState[(x-1) %
                                               self.nxC, (y+1) % self
                                               .nyC],
136                                         self.gameState[(x) % self
                                               .nxC, (y+1) % self.nyC
                                               ],
137                                         self.gameState[(x+1) %
                                               self.nxC, (y+1) % self
                                               .nyC]])
138
139                         if self.gameState[x, y] == 4:
140                             self.count4 +=1
141                         if self.gameState[x, y] == 5:
142                             self.count5 +=1
143                         if self.gameState[x, y] == 7:
144                             self.count7 +=1
145                         if self.gameState[x, y] == 8:
146                             self.count8 +=1
147                         if self.entropiaGrafica:
148                             self.calcularVecindadValor(self.
                                   getNeibors(self.gameState[x, y],
                                   n_neigh),file)
149
150
151                         # Q0
152                         if self.gameState[x, y] == 0:
153                             if( self.mins[x,y] == 0 and ((self.
                                   randomOffset()+1)<7) and (self.
                                   getMoore(n_neigh,4) != 0 or self.
                                   getMoore(n_neigh,3) != 0 or self.
                                   getMoore(n_neigh,6) != 0) ):
154                                 self.aux[x, y] = 7
155                                 self.cont+=1
156                                 self.val[x, y] = self.cont
157                             if(int(self.regla)==2):
158                                 if(self.getMoore(n_neigh,2) != 0):
159                                     self.aux[x, y] = 9
160                         # Q1
161                         elif self.gameState[x, y] == 1:
162                             if(self.getMoore(n_neigh,5) != 0 or self.
                                   getMoore(n_neigh,6)!=0):
163                                 self.aux[x, y] = 6
164                         # Q2 periodico, Q3 periodico, Q4
```

```
165 ||                      elif self.gameState[x, y] == 4: #Caso 0
166 ||                          if (self.getMoore(n_neigh,0) == 0 and
    ||                             self.getMoore(n_neigh,7) == 0):
167 ||                              d1 = self.val[x,y+1]
168 ||                              d2 = self.val[x-1,y+1]
169 ||                              d3 = self.val[x-1,y]
170 ||                              d4 = self.val[x-1,y-1]
171 ||                              d5 = self.val[x,y-1]
172 ||                              d6 = self.val[x+1,y-1]
173 ||                              d7 = self.val[x+1,y]
174 ||                              d8 = self.val[x+1,y+1]
175 ||                              tempo = []
176 ||                              if( 3 == n_neigh[6] or 5 == n_neigh
    ||                                 [6] or 6 == n_neigh[6]):
177 ||                                  tempo.insert(0,d1)
178 ||                              if( 3 == n_neigh[5] or 5 == n_neigh
    ||                                 [5] or 6 == n_neigh[5]):
179 ||                                  tempo.insert(0,d2)
180 ||                              if( 3 == n_neigh[3] or 5 == n_neigh
    ||                                 [3] or 6 == n_neigh[3]):
181 ||                                  tempo.insert(0,d3)
182 ||                              if( 3 == n_neigh[0] or 5 == n_neigh
    ||                                 [0] or 6 == n_neigh[0] ):
183 ||                                  tempo.insert(0,d4)
184 ||                              if( 3 == n_neigh[1] or 5 == n_neigh
    ||                                 [1] or 6 == n_neigh[1]):
185 ||                                  tempo.insert(0,d5)
186 ||                              if( 3 == n_neigh[2] or 5 == n_neigh
    ||                                 [2] or 6 == n_neigh[2]):
187 ||                                  tempo.insert(0,d6)
188 ||                              if( 3 == n_neigh[4] or 5 == n_neigh
    ||                                 [4] or 6 == n_neigh[4]):
189 ||                                  tempo.insert(0,d7)
190 ||                              if( 3 == n_neigh[7] or 5 == n_neigh
    ||                                 [7] or 6 == n_neigh[7] ):
191 ||                                  tempo.insert(0,d8)
192 ||                              linp = np.array(tempo)
193 ||                              peq = 0
194 ||                              if linp.size != 0:
195 ||                                  peq = np.min(linp)
196 ||                              if(d1 == peq and ( self.MooreOffset(
    ||                                 n_neigh,3,1) or self.MooreOffset(
    ||                                 n_neigh,5,1) or self.MooreOffset(
    ||                                 n_neigh,6,1))):
197 ||                                  self.aux[x,y] = 5
198 ||                                  self.mins[x,y] = 1
199 ||                              elif(d2 == peq and ( self.MooreOffset
    ||                                 (n_neigh,3,2) or self.MooreOffset(
    ||                                 n_neigh,5,2) or self.MooreOffset(
    ||                                 n_neigh,6,2))):
200 ||                                  self.aux[x,y] = 5
201 ||                                  self.mins[x,y] = 2
202 ||                              elif(d3 == peq and ( self.MooreOffset
    ||                                 (n_neigh,3,3) or self.MooreOffset(
    ||                                 n_neigh,5,3) or self.MooreOffset(
```

```
                                    n_neigh,6,3))):
203                                   self.aux[x,y] = 5
204                                   self.mins[x,y] = 3
205                                elif(d4 == peq and ( self.MooreOffset
                                      (n_neigh,3,4) or self.MooreOffset(
                                      n_neigh,5,4) or self.MooreOffset(
                                      n_neigh,6,4))):
206                                   self.aux[x,y] = 5
207                                   self.mins[x,y] = 4
208                                elif(d5 == peq and ( self.MooreOffset
                                      (n_neigh,3,5) or self.MooreOffset(
                                      n_neigh,5,5) or self.MooreOffset(
                                      n_neigh,6,5))):
209                                   self.aux[x,y] = 5
210                                   self.mins[x,y] = 5
211                                elif(d6 == peq and ( self.MooreOffset
                                      (n_neigh,3,6) or self.MooreOffset(
                                      n_neigh,5,6) or self.MooreOffset(
                                      n_neigh,6,6))):
212                                   self.aux[x,y] = 5
213                                   self.mins[x,y] = 6
214                                elif(d7 == peq and ( self.MooreOffset
                                      (n_neigh,3,7) or self.MooreOffset(
                                      n_neigh,5,7) or self.MooreOffset(
                                      n_neigh,6,7))):
215                                   self.aux[x,y] = 5
216                                   self.mins[x,y] = 7
217                                elif(d8 == peq and ( self.MooreOffset
                                      (n_neigh,3,8) or self.MooreOffset(
                                      n_neigh,5,8) or self.MooreOffset(
                                      n_neigh,6,8))):
218                                   self.aux[x,y] = 5
219                                   self.mins[x,y] = 8
220                           # Q5
221                           elif self.gameState[x, y] == 5:
222                               if(not self.depfun(8,x,y,n_neigh,self.
                                      mins) and  not self.depfun(5,x,y,
                                      n_neigh,self.mins) and self.getMoore(
                                      n_neigh,6) == 0 and self.getMoore(
                                      n_neigh,4) == 0 and self.getMoore(
                                      n_neigh,1) == 0 and self.getMoore(
                                      n_neigh,3) == 0 ):
223                                    self.aux[x, y] = 0
224                                    self.mins[x, y] = 0
225                               else:
226                                    self.aux[x, y] = 8
227                           # Q6 periodico, Q7
228                           elif self.gameState[x, y] == 7:
229                               if( self.getMoore(n_neigh,5) > 0 or self.
                                      getMoore(n_neigh,6) > 0 or self.
                                      getMoore(n_neigh,3) > 0 or self.
                                      getMoore(n_neigh,4) > 0):
230                                    self.aux[x, y] = 4
231
232                           elif self.gameState[x, y] == 8:
```

```
233                         self.aux[x, y] = 5
234
235                 poly = [((x)*self.dimCW, y*self.dimCH),
236                         ((x+1)*self.dimCW, y*self.dimCH),
237                         ((x+1)*self.dimCW, (y+1)*self.dimCH),
238                         ((x)*self.dimCW, (y+1)*self.dimCH)]
239
240
241                 if(self.aux[x, y] == 0):
242                     pygame.draw.polygon(self.screen, self.Q0,
                            poly, 0)
243                 elif(self.aux[x, y] == 1):
244                     pygame.draw.polygon(self.screen, self.Q1,
                            poly, 0)
245                 elif(self.aux[x, y] == 2):
246                     pygame.draw.polygon(self.screen, self.Q2,
                            poly, 0)
247                 elif(self.aux[x, y] == 3):
248                     pygame.draw.polygon(self.screen, self.Q3,
                            poly, 0)
249                 elif(self.aux[x, y] == 4):
250                     pygame.draw.polygon(self.screen, self.Q4,
                            poly, 0)
251                 elif(self.aux[x, y] == 5):
252                     pygame.draw.polygon(self.screen, self.Q5,
                            poly, 0)
253                 elif(self.aux[x, y] == 6):
254                     pygame.draw.polygon(self.screen, self.Q6,
                            poly, 0)
255                 elif(self.aux[x, y] == 7):
256                     pygame.draw.polygon(self.screen, self.Q7,
                            poly, 0)
257                 elif(self.aux[x, y] == 8):
258                     pygame.draw.polygon(self.screen, self.Q8,
                            poly, 0)
259                 elif(self.aux[x, y] == 9):
260                     pygame.draw.polygon(self.screen, self.Q9,
                            poly, 0)
261
262
263         if(self.generacion>1 and not self.pauseExect and self.
               entropiaGrafica):
264             file.write('---------\n')
265             file.close()
266
267         return self.aux,self.count4,self.count5,self.count7,self.
               count8,self.cont,self.mins,self.val
```

12.1.3. Neumann

```
1 from random import randint
2 import numpy as np
3 import pygame
4 class Neumann:
```

```
5 ||     def __init__(self,nxC,nyC,pauseExect,gameState,mins,aux,regla
  ||        ,val,cont,count4,count5,count7,count8,dimCW,dimCH,screen,
  ||        Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,entropiaGrafica,generacion):
6 ||         self.nxC=nxC
7 ||         self.nyC=nyC
8 ||         self.pauseExect=pauseExect
9 ||         self.gameState=gameState
10||         self.mins=mins
11||         self.aux=aux
12||         self.regla=regla
13||         self.val=val
14||         self.cont=cont
15||         self.count4=count4
16||         self.count5=count5
17||         self.count7=count7
18||         self.count8=count8
19||         self.dimCW=dimCW
20||         self.dimCH=dimCH
21||         self.screen=screen
22||         self.entropiaGrafica=entropiaGrafica
23||         self.generacion=generacion
24||
25||         self.Q0 = Q0    # Campo libre
26||         self.Q1 = Q1    # Nutriente no encontrado
27||         self.Q2 = Q2    # Repelente
28||         self.Q3 = Q3    # Punto inicial
29||         self.Q4 = Q4    # Gel en concentracion
30||         self.Q5 = Q5    # Gel con compuesto
31||         self.Q6 = Q6    # Nutriente hallado
32||         self.Q7 = Q7    # Expansion del Physarum
33||         self.Q8 = Q8    # Gel sin compuesto
34||         self.Q9 = Q0    #q0+algo para la regla 2
35||
36||         pass
37||
38||     def randomOffset(self):
39||         return randint(0, ((7 + 1) - 0)) + 0
40||
41||     def getNewman(self,n_neigh, crit): #Cambiar a moore despues
  ||        xd
42||         x = 0
43||         if(n_neigh[3] == crit):
44||             x +=1
45||         if(n_neigh[1] == crit):
46||             x +=1
47||         if(n_neigh[0] == crit):
48||             x +=1
49||         if(n_neigh[2] == crit):
50||             x +=1
51||         return x
52||
53||     def NewmanOffset(self,n_neigh, crit, orien):
54||         if(n_neigh[3] == crit and orien ==1):
55||             return True
56||         elif(n_neigh[1] == crit and orien ==2):
```

```
57 ||             return True
58 ||         elif(n_neigh[0] == crit and orien ==3):
59 ||             return True
60 ||         elif(n_neigh[2] == crit and orien ==4):
61 ||             return True
62 ||         return False
63 ||
64 ||
65 ||
66 ||     def depfun(self,estado, x, y, n_neigh,mins):  # mide las
   ||        dependecias
67 ||     # abajo,izq,derecha,arriba
68 ||         if(estado == n_neigh[3] and mins[x][y+1] == 3):
69 ||             return True
70 ||         elif(estado == n_neigh[1] and mins[x-1][y] == 4):
71 ||             return True
72 ||         elif(estado == n_neigh[0]and mins[x][y-1] == 1):
73 ||             return True
74 ||         elif(estado == n_neigh[2] and mins[x+1][y] == 2):
75 ||             return True
76 ||         return False
77 ||
78 ||     def getNeibors(self,actual,n_neigh):
79 ||         auxi=[]
80 ||         for val in n_neigh:
81 ||             auxi.append(val)
82 ||         auxi.insert(2,actual)
83 ||         return auxi
84 ||
85 ||     def calcularVecindadValor(self,arre,file):
86 ||         total=0
87 ||         x=0
88 ||         for x in range(len(arre)):
89 ||             if arre[x]!=0 and arre[x]!=2:
90 ||                 total+=2**x
91 ||         file.write(str(total)+'\n')
92 ||
93 ||     def movimiento(self):
94 ||         file= open('entropia/EntroInfo.txt','a')
95 ||         for x in range(0, self.nxC):
96 ||             for y in range(0, self.nyC):
97 ||                 if not self.pauseExect:
98 ||                     n_neigh = np.array([self.gameState[(x) % self
   ||                        .nxC, (y-1) % self.nyC],
99 ||                                         self.gameState[(x-1) %
   ||                                            self.nxC, (y) %self.
   ||                                            nyC],
100||                                         self.gameState[(x+1) %
   ||                                            self.nxC, (y) % self.
   ||                                            nyC],
101||                                         self.gameState[(x) % self
   ||                                            .nxC, (y+1) % self.nyC
   ||                                            ]])
102||
103||                     if self.gameState[x, y] == 4:
```

```
104                         self.count4 +=1
105                     if self.gameState[x, y] == 5:
106                         self.count5 +=1
107                     if self.gameState[x, y] == 7:
108                         self.count7 +=1
109                     if self.gameState[x, y] == 8:
110                         self.count8 +=1
111                     if self.entropiaGrafica:
112                         self.calcularVecindadValor(self.
                                getNeibors(self.gameState[x, y],
                                n_neigh),file)
113
114                     # Q0
115                     if self.gameState[x, y] == 0:
116                         if( self.mins[x,y] == 0 and ((self.
                                randomOffset()+1)<3) and (self.
                                getNewman(n_neigh,4) != 0 or self.
                                getNewman(n_neigh,3) != 0 or self.
                                getNewman(n_neigh,6) != 0) ):
117                             self.aux[x, y] = 7
118                             self.cont+=1
119                             self.val[x, y] = self.cont
120                         if(int(self.regla)==2):
121                             if(self.getNewman(n_neigh,2) != 0):
122                                 self.aux[x, y] = 9
123                     # Q1
124                     elif self.gameState[x, y] == 1:
125                         if(self.getNewman(n_neigh,5) != 0 or self
                                .getNewman(n_neigh,6)!=0):
126                             self.aux[x, y] = 6
127                     # Q2 periodico, Q3 periodico, Q4
128                     elif self.gameState[x, y] == 4: #Caso 0
129                         if (self.getNewman(n_neigh,0) == 0 and
                                self.getNewman(n_neigh,7) == 0):
130                             d1 = self.val[x,y+1]
131                             d2 = self.val[x-1,y]
132                             d3 = self.val[x,y-1]
133                             d4 = self.val[x+1,y]
134                             tempo = []
135                             if( 3 == n_neigh[3] or 5 == n_neigh
                                    [3] or 6 == n_neigh[3]):
136                                 tempo.insert(0,d1)
137                             if( 3 == n_neigh[1] or 5 == n_neigh
                                    [1] or 6 == n_neigh[1]):
138                                 tempo.insert(0,d2)
139                             if( 3 == n_neigh[0] or 5 == n_neigh
                                    [0] or 6 == n_neigh[0]):
140                                 tempo.insert(0,d3)
141                             if( 3 == n_neigh[2] or 5 == n_neigh
                                    [2] or 6 == n_neigh[2] ):
142                                 tempo.insert(0,d4)
143                             linp = np.array(tempo)
144                             peq = 0
145                             if linp.size != 0:
146                                 peq = np.min(linp)
```

```
147                             if(d1 == peq and ( self.NewmanOffset(
                                   n_neigh,3,1) or self.NewmanOffset(
                                   n_neigh,5,1) or self.NewmanOffset(
                                   n_neigh,6,1))):
148                                 self.aux[x,y] = 5
149                                 self.mins[x,y] = 1
150                             elif(d2 == peq and ( self.
                                   NewmanOffset(n_neigh,3,2) or self.
                                   NewmanOffset(n_neigh,5,2) or self.
                                   NewmanOffset(n_neigh,6,2))):
151                                 self.aux[x,y] = 5
152                                 self.mins[x,y] = 2
153                             elif(d3 == peq and ( self.
                                   NewmanOffset(n_neigh,3,3) or self.
                                   NewmanOffset(n_neigh,5,3) or self.
                                   NewmanOffset(n_neigh,6,3))):
154                                 self.aux[x,y] = 5
155                                 self.mins[x,y] = 3
156                             elif(d4 == peq and ( self.
                                   NewmanOffset(n_neigh,3,4) or self.
                                   NewmanOffset(n_neigh,5,4) or self.
                                   NewmanOffset(n_neigh,6,4))):
157                                 self.aux[x,y] = 5
158                                 self.mins[x,y] = 4
159                     # Q5
160                     elif self.gameState[x, y] == 5:
161                         if(not self.depfun(8,x,y,n_neigh,self.
                               mins) and  not self.depfun(5,x,y,
                               n_neigh,self.mins) and self.getNewman(
                               n_neigh,6) == 0 and self.getNewman(
                               n_neigh,4) == 0 and self.getNewman(
                               n_neigh,1) == 0 and self.getNewman(
                               n_neigh,3) == 0 ):
162                             self.aux[x, y] = 0
163                             self.mins[x, y] = 0
164                         else:
165                             self.aux[x, y] = 8
166                     # Q6 periodico, Q7
167                     elif self.gameState[x, y] == 7:
168                         if( self.getNewman(n_neigh,5) > 0 or self
                               .getNewman(n_neigh,6) > 0 or self.
                               getNewman(n_neigh,3) > 0 or self.
                               getNewman(n_neigh,4) > 0):
169                             self.aux[x, y] = 4
170                     elif self.gameState[x, y] == 8:
171                         self.aux[x, y] = 5
172 
173                 poly = [((x)*self.dimCW, y*self.dimCH),
174                         ((x+1)*self.dimCW, y*self.dimCH),
175                         ((x+1)*self.dimCW, (y+1)*self.dimCH),
176                         ((x)*self.dimCW, (y+1)*self.dimCH)]
177 
178 
179                 if(self.aux[x, y] == 0):
180                     pygame.draw.polygon(self.screen, self.Q0,
```

```
                        poly, 0)
                elif(self.aux[x, y] == 1):
                    pygame.draw.polygon(self.screen, self.Q1,
                        poly, 0)
                elif(self.aux[x, y] == 2):
                    pygame.draw.polygon(self.screen, self.Q2,
                        poly, 0)
                elif(self.aux[x, y] == 3):
                    pygame.draw.polygon(self.screen, self.Q3,
                        poly, 0)
                elif(self.aux[x, y] == 4):
                    pygame.draw.polygon(self.screen, self.Q4,
                        poly, 0)
                elif(self.aux[x, y] == 5):
                    pygame.draw.polygon(self.screen, self.Q5,
                        poly, 0)
                elif(self.aux[x, y] == 6):
                    pygame.draw.polygon(self.screen, self.Q6,
                        poly, 0)
                elif(self.aux[x, y] == 7):
                    pygame.draw.polygon(self.screen, self.Q7,
                        poly, 0)
                elif(self.aux[x, y] == 8):
                    pygame.draw.polygon(self.screen, self.Q8,
                        poly, 0)
                elif(self.aux[x, y] == 9):
                    pygame.draw.polygon(self.screen, self.Q9,
                        poly, 0)

        if(self.generacion>1 and not self.pauseExect and self.
            entropiaGrafica):
             file.write('---------\n')
             file.close()

        return self.aux,self.count4,self.count5,self.count7,self.
            count8,self.cont,self.mins,self.val
```

12.1.4. Simulador

```
import pygame, sys
import numpy as np
import time
import pylab
import math
import tkinter
import tkinter.filedialog
import tkinter.simpledialog
from tkinter import colorchooser
import matplotlib
matplotlib.use("Agg")
import matplotlib.animation as animation
import matplotlib.backends.backend_agg as agg
from random import randint
from tkinter import *
from PIL import ImageColor
```

```
17 from boton import Button
18 from boton import setTexto
19 from boton import buttons_draw
20 from imagenImp import saveImagen
21 from graficas import Graficas
22 from screeninfo import get_monitors
23 from Moore import Moore
24 from Neumann import Neumann
25 import tkinter as tk
26 global pauseExect,aux,gameState,mins,val,generacion,count4,count5
   ,count7,count8,textoBoton,delay,boxactual,graficaActual,Q0,Q1,
   Q2,Q3,Q4,Q5,Q6,Q7,Q8
27 textoBoton =''
28 def Simulador (celulas,regla,vecindad,entro):
29     global pauseExect,aux,gameState,mins,val,generacion,count4,
          count5,count7,count8,textoBoton,delay,boxactual,cont,
          graficaActual,Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8
30     screenHeigt=[]
31     screenWidth=[]
32     for m in get_monitors():
33         screenHeigt.append(int(m.height))
34         screenWidth.append(int(m.width))
35 
36     print("Usando la vecindad de ",vecindad)
37 
38     TotalLargo=math.ceil(screenHeigt[0]*0.6944)
39     grafica=math.ceil(TotalLargo*0.60)
40     ladoGrafica=math.ceil(grafica/100)
41 
42     Q0 = "#002f68"  # Campo libre
43     Q1 = "#4268c6"  # Nutriente no encontrado
44     Q2 = "#c70039" # Repelente
45     Q3 = "#000000"  # Punto inicial
46     Q4 = "#ECEC0A"  # Gel en concentracion
47     Q5 = "#348300" # Gel con compuesto
48     Q6 = "#f1fce1" # Nutriente hallado
49     Q7 = "#5A6051"  # Expansion del Physarum
50     Q8 = "#97C363"  # Gel sin compuesto
51     Q9 = "#002f68" #q0+algo para la regla 2
52 
53     fig = pylab.figure(figsize=[ladoGrafica, ladoGrafica-1],dpi
          =100,)
54     ax = fig.add_subplot()
55     ax.clear()
56     #Grafica de crecimiento de estados
57     ani = animation.FuncAnimation(fig, Graficas(ax,Q4,Q5,Q7,Q8).
          crecimiento,interval=1)
58     canvas = agg.FigureCanvasAgg(fig)
59     canvas.draw()
60     renderer = canvas.get_renderer()
61     raw_data = renderer.tostring_rgb()
62 
63     window = Tk()
64     PixelesXlado=TotalLargo
65     height,width = PixelesXlado, PixelesXlado
```

```
66 ||
67 ||     nxC, nyC = celulas, celulas
68 ||     dimCW = width/celulas
69 ||     dimCH = height/celulas
70 ||
71 ||     pygame.init()
72 ||
73 ||     screen = pygame.display.set_mode([height+grafica, width],
   ||        pygame.DOUBLEBUF)
74 ||
75 ||     size = canvas.get_width_height()
76 ||     gui_font = pygame.font.Font(None,30)
77 ||
78 ||     ladoExtra=int(math.ceil(PixelesXlado*0.2))
79 ||     otroExtra=int(math.ceil(PixelesXlado*0.05))
80 ||
81 ||     button0 = Button('Sig grafica',200,30,(PixelesXlado+ladoExtra
   ||        ,PixelesXlado-(otroExtra*7.9)),5,gui_font,screen)
82 ||
83 ||     button1 = Button('Iniciar/Pausar',200,30,(PixelesXlado+
   ||        ladoExtra,PixelesXlado-(otroExtra*6)),5,gui_font,screen)
84 ||     button2 = Button('Reiniciar',200,30,(PixelesXlado+ladoExtra,
   ||        PixelesXlado-(otroExtra*5)),5,gui_font,screen)
85 ||     button3 = Button('Abrir',200,30,(PixelesXlado+ladoExtra,
   ||        PixelesXlado-(otroExtra*4)),5,gui_font,screen)
86 ||     button4 = Button('Guardar',200,30,(PixelesXlado+ladoExtra,
   ||        PixelesXlado-(otroExtra*3)),5,gui_font,screen)
87 ||     buttonColor = Button('Color',60,30,(PixelesXlado+ladoExtra
   ||        +140,PixelesXlado-(otroExtra*2)),5,gui_font,screen)
88 ||     button5 = Button('+',40,40,(PixelesXlado+ladoExtra +190,
   ||        PixelesXlado-(otroExtra*7)),5,gui_font,screen)
89 ||     button6 = Button('-',40,40,(PixelesXlado+ladoExtra -30,
   ||        PixelesXlado-(otroExtra*7)),5,gui_font,screen)
90 ||
91 ||     buttonEstado0 = Button('0',30,30,(PixelesXlado+ladoExtra -80,
   ||        PixelesXlado-(otroExtra*1)),5,gui_font,screen)
92 ||     buttonEstado1 = Button('1',30,30,(PixelesXlado+ladoExtra -40,
   ||        PixelesXlado-(otroExtra*1)),5,gui_font,screen)
93 ||     buttonEstado2 = Button('2',30,30,(PixelesXlado+ladoExtra +00,
   ||        PixelesXlado-(otroExtra*1)),5,gui_font,screen)
94 ||     buttonEstado3 = Button('3',30,30,(PixelesXlado+ladoExtra +40,
   ||        PixelesXlado-(otroExtra*1)),5,gui_font,screen)
95 ||     buttonEstado4 = Button('4',30,30,(PixelesXlado+ladoExtra +80,
   ||        PixelesXlado-(otroExtra*1)),5,gui_font,screen)
96 ||     buttonEstado5 = Button('5',30,30,(PixelesXlado+ladoExtra
   ||        +120,PixelesXlado-(otroExtra*1)),5,gui_font,screen)
97 ||     buttonEstado6 = Button('6',30,30,(PixelesXlado+ladoExtra
   ||        +160,PixelesXlado-(otroExtra*1)),5,gui_font,screen)
98 ||     buttonEstado7 = Button('7',30,30,(PixelesXlado+ladoExtra
   ||        +200,PixelesXlado-(otroExtra*1)),5,gui_font,screen)
99 ||     buttonEstado8 = Button('8',30,30,(PixelesXlado+ladoExtra
   ||        +240,PixelesXlado-(otroExtra*1)),5,gui_font,screen)
100 ||
101 ||    buttonHelp = Button('?',30,30,(height+grafica-30, width-30)
   ||        ,5,gui_font,screen)
```

```
102     buttons= [button0,button1,button2,button3,button4,button5,
            button6,buttonColor,buttonEstado0,buttonEstado1,
            buttonEstado2,buttonEstado3,buttonEstado4,buttonEstado5,
            buttonEstado6,buttonEstado7,buttonEstado8,buttonHelp]
103
104     myfont = pygame.font.SysFont("monospace", 30)
105
106     bg = 0, 47, 104
107     screen.fill(bg)
108     gameState = np.zeros((nxC, nyC))
109
110     pauseExect = True
111     running = True
112
113     mins = np.zeros((nxC, nyC))
114     cont = 0
115     val = np.zeros((nxC, nyC))
116     generacion = 0
117     aux = np.copy(gameState)
118     delay=0
119     graficaActual=0
120     boxactual=0
121     #contadores
122
123     count4 = 0 #Q4
124     count5 = 0 #Q5
125     count7 = 0 #Q7
126     count8 = 0 #Q8
127
128     listgene= []
129     list4=[]
130     list5=[]
131     list7=[]
132     list8=[]
133     buttons_draw(buttons)
134     pygame.display.update()
135
136     file= open('data.txt','w')
137     file.write(str(generacion)+','+str(count4)+','+str(count5)+',
            '+str(count7)+','+str(count8)+'\n')
138     file.close()
139
140     file= open('entropia/EntroInfo.txt','w')
141     file.write('')
142     file.close()
143
144     def pausar():
145         global pauseExect
146         pauseExect = not pauseExect
147
148     def clear():
149         global pauseExect,aux,gameState,mins,val,generacion,
                count4,count5,count7,count8,delay,boxactual,
                graficaActual,cont
150         aux = np.zeros((nxC, nyC))
```

```
151 ||          gameState = np.zeros((nxC, nyC))
152 ||          mins = np.zeros((nxC, nyC))
153 ||          val = np.zeros((nxC, nyC))
154 ||          aux=margen()
155 ||          generacion = 0
156 ||          delay=0
157 ||          cont = 0
158 ||          boxactual=0
159 ||          graficaActual=0
160 ||          file= open('data.txt','w')
161 ||          file.write('')
162 ||          file.close()
163 ||          file= open('entropia/EntroInfo.txt','w')
164 ||          file.write('')
165 ||          file.close()
166 ||          count4 = 0 #Q4
167 ||          count5 = 0 #Q5
168 ||          count7 = 0 #Q7
169 ||          count8 = 0 #Q8
170 ||
171 ||      def abrir():
172 ||          global aux,mins
173 ||          window.withdraw()
174 ||          filename = tkinter.filedialog.askopenfilename()
175 ||          try:
176 ||              aux,mins=load(filename,celulas)
177 ||          except:
178 ||              print('No se selecciono un archivo')
179 ||
180 ||      def guardar():
181 ||          window.withdraw()
182 ||          txt = tkinter.simpledialog.askstring(title="Save",prompt=
    ||             "What will  be the name of your text?:")
183 ||          save(txt,gameState)
184 ||
185 ||      def desplegarHelp():
186 ||          window.withdraw()
187 ||          mensaje="""
188 ||0) Campo libre
189 ||1) Nutriente no encontrado
190 ||2) Repelente
191 ||3) Punto inicial
192 ||4) Gel en concentracion
193 ||5) Gel con compuesto
194 ||6) Nutriente hallado
195 ||7) Expansion del Physarum
196 ||8) Gel sin compuesto
197 ||          """
198 ||          tkinter.messagebox.showinfo(message=mensaje, title="
    ||             Estados")
199 ||
200 ||      def margen():
201 ||          aux = np.copy(gameState)
202 ||          for y in range(0, nyC):
203 ||              for x in range(0, nxC):
```

```
204                     if(y==0 or y==nyC-1 or x==0 or x==nxC-1):
205                         aux[x, y] = 2
206             return aux
207 
208         def save(filename,gameState):
209             f = open('save/moore/'+str(filename)+'_'+str(celulas)+'.
                    txt','w')
210             aux = np.copy(gameState)
211             for y in range(0, nxC):
212                 for x in range(0, nyC):
213                     f.write(str(x)+","+str(y)+","+str(aux[x, y])+","+
                            str(mins[x, y])+"\n")
214             f.close
215 
216         def getNuberLines(filename):
217             count = 0
218             with open(filename,encoding="utf8") as f:
219                 lines = f.readlines()
220                 for line in lines:
221                     count +=1
222             return int(math.sqrt(count))
223 
224 
225         def load(filename,celulas):
226             clon = np.zeros((nxC, nyC))
227             clonMins = np.zeros((nxC, nyC))
228             numauxi=getNuberLines(filename)
229             with open(filename) as f:
230                 lines = f.readlines()
231 
232                 if numauxi==celulas:
233                     for line in lines:
234                         auxi=line.split(',')
235                         x = int(auxi[0])
236                         y = int(auxi[1])
237                         tipo= auxi[2]
238                         mini= auxi[3]
239                         clon[x][y]=tipo
240                         clonMins[x][y]=mini
241                 else:
242                     clon=saveImagen(filename,int(numauxi),int(celulas
                            ))
243             return clon,clonMins
244 
245         def escogerColor(boxactual):
246             global Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8
247             window.withdraw()
248             if boxactual==0:
249                 coloraux=Q0
250                 Q0 = colorchooser.askcolor()[1]
251                 if(Q0==None):
252                     Q0=coloraux
253             if boxactual==1:
254                 coloraux=Q1
255                 Q1 = colorchooser.askcolor()[1]
```

```
256 ||              if(Q1==None):
257 ||                  Q1=coloraux
258 ||          if boxactual==2:
259 ||              coloraux=Q2
260 ||              Q2 = colorchooser.askcolor()[1]
261 ||              if(Q2==None):
262 ||                  Q2=coloraux
263 ||          if boxactual==3:
264 ||              coloraux=Q3
265 ||              Q3 = colorchooser.askcolor()[1]
266 ||              if(Q3==None):
267 ||                  Q3=coloraux
268 ||          if boxactual==4:
269 ||              coloraux=Q4
270 ||              Q4 = colorchooser.askcolor()[1]
271 ||              if(Q4==None):
272 ||                  Q4=coloraux
273 ||          if boxactual==5:
274 ||              coloraux=Q5
275 ||              Q5 = colorchooser.askcolor()[1]
276 ||              if(Q5==None):
277 ||                  Q5=coloraux
278 ||          if boxactual==6:
279 ||              coloraux=Q6
280 ||              Q6 = colorchooser.askcolor()[1]
281 ||              if(Q6==None):
282 ||                  Q6=coloraux
283 ||          if boxactual==7:
284 ||              coloraux=Q7
285 ||              Q7 = colorchooser.askcolor()[1]
286 ||              if(Q7==None):
287 ||                  Q7=coloraux
288 ||          if boxactual==8:
289 ||              coloraux=Q8
290 ||              Q8 = colorchooser.askcolor()[1]
291 ||              if(Q8==None):
292 ||                  Q8=coloraux
293 ||
294 ||      while running:
295 ||          aux=margen()#Pone el margen de repelentes al rededor
296 ||          button7 = Button('Estado ' + str(boxactual),120,30,(
    ||             PixelesXlado+ladoExtra,PixelesXlado-(otroExtra*2)),5,
    ||             gui_font,screen)
297 ||          buttons.append(button7)
298 ||          #Indica que acciones tomar con base a los botones
299 ||          textoBoton=setTexto()
300 ||          if(textoBoton=='Iniciar/Pausar'):
301 ||              pausar()
302 ||              textoBoton=''
303 ||          elif(textoBoton=='Reiniciar'):
304 ||              clear()
305 ||              textoBoton=''
306 ||          elif(textoBoton=='Abrir'):
307 ||              clear()
308 ||              abrir()
```

```
309 ||              textoBoton=''
310 ||          elif(textoBoton=='Guardar'):
311 ||              guardar()
312 ||              textoBoton=''
313 ||          elif(textoBoton=='Color'):
314 ||              escogerColor(boxactual)
315 ||              textoBoton=''
316 ||          elif(textoBoton=='+'):
317 ||              if(delay>=0 and delay<=9):
318 ||                  delay+=1
319 ||              textoBoton=''
320 ||          elif(textoBoton=='-'):
321 ||              if(delay>0 and delay<=10):
322 ||                  delay-=1
323 ||              textoBoton=''
324 ||          elif(textoBoton=='0'):
325 ||              boxactual=0
326 ||              textoBoton=''
327 ||          elif(textoBoton=='1'):
328 ||              boxactual=1
329 ||              textoBoton=''
330 ||          elif(textoBoton=='2'):
331 ||              boxactual=2
332 ||              textoBoton=''
333 ||          elif(textoBoton=='3'):
334 ||              boxactual=3
335 ||              textoBoton=''
336 ||          elif(textoBoton=='4'):
337 ||              boxactual=4
338 ||              textoBoton=''
339 ||          elif(textoBoton=='5'):
340 ||              boxactual=5
341 ||              textoBoton=''
342 ||          elif(textoBoton=='6'):
343 ||              boxactual=6
344 ||              textoBoton=''
345 ||          elif(textoBoton=='7'):
346 ||              boxactual=7
347 ||              textoBoton=''
348 ||          elif(textoBoton=='8'):
349 ||              boxactual=8
350 ||              textoBoton=''
351 ||          elif(textoBoton=='?'):
352 ||              desplegarHelp()
353 ||              textoBoton=''
354 ||          elif(textoBoton=='Sig grafica'):
355 ||              graficaActual+=1
356 ||              graficaActual=graficaActual%2
357 ||              textoBoton=''
358 ||          textoBoton=''
359 ||
360 ||
361 ||          ev = pygame.event.get()
362 ||          for event in ev:
```

```
363 ||              if event.type == pygame.QUIT:
364 ||                  window.withdraw()
365 ||                  running = False
366 ||                  file= open('data.txt','w')
367 ||                  file.write('')
368 ||                  file.close()
369 ||              if event.type == pygame.KEYDOWN:
370 ||                  if event.key == pygame.K_SPACE:
371 ||                      pauseExect = not pauseExect
372 ||                  if event.key == pygame.K_c:
373 ||                      aux = np.zeros((nxC, nyC))
374 ||                      gameState = np.zeros((nxC, nyC))
375 ||                      mins = np.zeros((nxC, nyC))
376 ||                      val = np.zeros((nxC, nyC))
377 ||                      aux=margen()
378 ||                      generacion = 0
379 ||                      file= open('data.txt','w')
380 ||                      file.write('')
381 ||                      file.close()
382 ||                      file= open('entropia/EntroInfo.txt','w')
383 ||                      file.write('')
384 ||                      file.close()
385 ||                      count4 = 0 #Q4
386 ||                      count5 = 0 #Q5
387 ||                      count7 = 0 #Q7
388 ||                      count8 = 0 #Q8
389 ||
390 ||                  if event.key == pygame.K_a:
391 ||                      window.withdraw()
392 ||                      filename = tkinter.filedialog.askopenfilename
    ||                         ()
393 ||                      aux=load(filename,celulas)
394 ||
395 ||                  if event.key == pygame.K_g:
396 ||                      window.withdraw()
397 ||                      txt = tkinter.simpledialog.askstring(title="
    ||                         Save",prompt="Ingresa el nombre de tu
    ||                         archivo, se le agregara la extencion y las
    ||                          celulas por lado")
398 ||                      save(txt,gameState)
399 ||
400 ||
401 ||              mouseClick = pygame.mouse.get_pressed()
402 ||              if sum(mouseClick) > 0:
403 ||                  posX, posY = pygame.mouse.get_pos()
404 ||                  celX, celY = int(np.floor(posX/dimCW)), int(np.
    ||                     floor(posY/dimCH))
405 ||                  try:
406 ||                      aux[celX, celY]=boxactual
407 ||                  except:
408 ||                      print("")
409 ||
410 ||          if not pauseExect:
411 ||              listgene.append(generacion)
412 ||              list4.append(count4)
```

```
413                 list5.append(count5)
414                 list7.append(count7)
415                 list8.append(count8)
416 
417                 file= open('data.txt','a')
418                 file.write(str(generacion)+','+str(count4)+','+str(
                        count5)+','+str(count7)+','+str(count8)+'\n')
419                 file.close()
420 
421                 generacion +=1
422                 count4 = 0 #Q4
423                 count5 = 0 #Q5
424                 count7 = 0 #Q7
425                 count8 = 0 #Q8
426             #Creacion de la instancia para las vecindades
427             if vecindad == 'neumann':
428                 comportamiento = Neumann(nxC,nyC,pauseExect,gameState
                        ,mins,aux,regla,val,cont,count4,count5,count7,
                        count8, dimCW,dimCH,screen,Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7
                        ,Q8,entro,generacion)
429             elif vecindad == 'moore':
430                 comportamiento = Moore(nxC,nyC,pauseExect,gameState,
                        mins,aux,regla,val,cont,count4,count5,count7,
                        count8, dimCW,dimCH,screen,Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7
                        ,Q8,entro,generacion)
431 
432 
433             aux,count4,count5,count7,count8,cont,mins,val=
                    comportamiento.movimiento()
434 
435             gameState = np.copy(aux)
436 
437             time.sleep(delay/100)
438             #textos
439             texto= "Generacion: "+str(generacion)
440             label = myfont.render(texto, 1, (255,255,255))
441             screen.blit(label, (17, 15))
442 
443             textoDelay= "Delay: "+str(delay)
444             labelDelay = myfont.render(textoDelay, 1, (255,255,255))
445             screen.blit(labelDelay, (math.ceil(screenHeigt[0]*0.55)
                    ,15))
446 
447             textoDelay= "  Delay  "
448             labelDelay = myfont.render(textoDelay, 1, (255,255,255))
449             screen.blit(labelDelay, (math.ceil(screenHeigt[0]*0.85),
                    math.ceil(screenWidth[0]*0.2530)))
450 
451             #checkbox
452 
453             #actualizar animacion
454             if graficaActual==0:
455                 ani = animation.FuncAnimation(fig, Graficas(ax,Q4,Q5,
                        Q7,Q8).crecimiento,interval=1)
```

```
456 ||          elif graficaActual==1:
457 ||              ani = animation.FuncAnimation(fig, Graficas(ax,Q4,Q5,
    ||                 Q7,Q8).crecimientoLog,interval=1)
458 ||          canvas = agg.FigureCanvasAgg(fig)
459 ||          canvas.draw()
460 ||          renderer = canvas.get_renderer()
461 ||          raw_data = renderer.tostring_rgb()
462 ||          surf = pygame.image.fromstring(raw_data, size, "RGB")
463 ||          screen.blit(surf, (PixelesXlado,0))
464 ||
465 ||          buttons_draw(buttons)
466 ||          pygame.display.update()
467 ||
468 ||      pygame.quit()
469 ||      file= open('data.txt','w')
470 ||      file.write('')
471 ||      file.close()
472 ||
473 ||if __name__ == "__main__":
474 ||      Simulador(30,1,'moore','S')
```

12.2. Código del robot

De igual forma, el código del robot es el siguiente, mencionar que solo se agregaron los archivos principales del funcionamiento del robot, omitiendo completamente el servicio web.

12.2.1. Archivo

```
import numpy as np
import os
import json
import pandas as pd
from matplotlib import pyplot as plt
import signal
from matplotlib.colors import ListedColormap
from Caminante import Caminante
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path
    .dirname(__file__)))

def getPIDFile():#Obtenemos el pid del proceso principal
    with open(os.path.join(__location__, "pidMain.txt")) as f:
        pid = f.read()
        return pid

def getGyroYaw():#Obtenemos el valor yaw del giroscopio con tipo
    string
    f = open(os.path.join(__location__, "GyroYawdata.txt"))
    lines = f.readlines()
    return lines[0].strip(),lines[1]

def deletePIDFile():
    os.remove(os.path.join(__location__, 'pidMain.txt'))

def setPIDFile(pidN):#Obtenemos es pid del proceso principal
    f = open(os.path.join(__location__, 'pidMain.txt'), "w")
    f.write(str(pidN))
    f.close()

def setCoorPas(x,y):
    arrA = ["coorPas1.txt","coorPas2.txt","coorPas3.txt","coorPas4
        .txt"]
    xpasA = ""
    ypasA = ""
    for i,nombre in enumerate(arrA):
        with open(os.path.join(__location__, nombre), 'r') as f:
            l = f.readlines()
            xs = l[0].replace('x=', '')
            xactA = int(xs.replace('\n', ''))
            yactA = int(l[1].replace('y=', ''))
        if(i==0):
```

```
40 ||           f = open(os.path.join(__location__, nombre), "w")
41 ||           f.write("x="+str(x)+"\n"+"y="+str(y))
42 ||           f.close()
43 ||           xpasA = xactA
44 ||           ypasA = yactA
45 ||       else:
46 ||           f = open(os.path.join(__location__, nombre), "w")
47 ||           f.write("x="+str(xpasA)+"\n"+"y="+str(ypasA))
48 ||           f.close()
49 ||           xpasA = xactA
50 ||           ypasA = yactA
51 ||
52 ||def compareCoorPasCoor(y,x):
53 ||    arrA = ["coorPas1.txt","coorPas2.txt","coorPas3.txt","coorPas4
   ||       .txt"]
54 ||    xpasC = ""
55 ||    ypasC = ""
56 ||    for nombre in arrA:
57 ||        with open(os.path.join(__location__, nombre), 'r') as f:
58 ||           l = f.readlines()
59 ||           xs = l[0].replace('x=', '')
60 ||           xpasC = int(xs.replace('\n', ''))
61 ||           ypasC = int(l[1].replace('y=', ''))
62 ||        if( x == xpasC and y == ypasC ):
63 ||           return True
64 ||    return False
65 ||
66 ||def getPosFile(name):
67 ||    filename = ""
68 ||    if(name == "Becas"):
69 ||        filename = "CoordLugares/Becas.txt"
70 ||    elif(name == "SubDireccion"):
71 ||        filename = "CoordLugares/SubDireccion.txt"
72 ||    elif(name == "Direccion"):
73 ||        filename = "CoordLugares/Direccion.txt"
74 ||    elif(name == "SEducativos"):
75 ||        filename = "CoordLugares/SEducativos.txt"
76 ||    elif(name == "SalaReunion"):
77 ||        filename = "CoordLugares/SalaReunion.txt"
78 ||    elif(name == "SAdministrativa"):
79 ||        filename = "CoordLugares/SubAdministrativa.txt"
80 ||    elif(name == "CHumano"):
81 ||        filename = "CoordLugares/CapitalHumano.txt"
82 ||    elif(name == "GTecnica"):
83 ||        filename = "CoordLugares/GestionTecnica.txt"
84 ||    elif(name == "TestA"):
85 ||        filename = "CoordLugares/TestA.txt"
86 ||    elif(name == "TestB"):
87 ||        filename = "CoordLugares/TestB.txt"
88 ||    elif(name == "TestC"):
89 ||        filename = "CoordLugares/TestC.txt"
90 ||    elif(name == "TestAS"):
91 ||        filename = "CoordLugares/TestAS.txt"
92 ||    elif(name == "Robot"):
93 ||        objCaminante = getFilePosCaminante()
```

```
94 ||        return objCaminante.pos_x,objCaminante.pos_y,objCaminante.
   ||           orientacion
95 ||    with open(os.path.join(__location__, filename), 'r') as f:
96 ||        l = f.readlines()
97 ||        xs = l[0].replace('x=', '')
98 ||        x = int(xs.replace('\n', ''))
99 ||        y = int(l[1].replace('y=', ''))
100||        salida = l[2].replace('salida=', '')
101||        return x,y,salida
102||
103||def setUbicFinal(ubic):
104||    f = open(os.path.join(__location__, "UbicFinal.txt"), "w")
105||    f.write(ubic)
106||    f.close()
107||
108||def getUbicFinal():
109||    with open(os.path.join(__location__, "UbicFinal.txt"), 'r') as
   ||        f:
110||        l = f.readlines()
111||        ubicacion = l[0].replace('\n', '')
112||        return str(ubicacion)
113||
114||def getPosition(array, value):
115||    position = np.where(array==value)
116||    y = position[0][0]
117||    x = position[1][0]
118||    return x,y
119||
120||
121||def getMapPrueba(nprueba):
122||    data_file = None
123||    if(nprueba==0):
124||        data_file = np.genfromtxt(os.path.join(__location__, '
   ||           ESCOM_Prueba1.csv'), delimiter=',').T
125||    elif(nprueba==1):
126||        data_file = np.genfromtxt(os.path.join(__location__, 'xd.
   ||           csv'), delimiter=',').T
127||    elif(nprueba==2):
128||        data_file = np.genfromtxt(os.path.join(__location__, '
   ||           ESCOM_Obstaculo_P1.csv'), delimiter=',').T
129||    colors = ["#002f68", "#4268c6", "#c70039", "#000000",
130||              "#ECEC0A", "#348300", "#f1fce1", "#5A6051", "#97C363","
   ||                #002f68"]  # use hex colors here, if desired.
131||    cmap = ListedColormap(colors)
132||    plt.imshow(data_file, vmin=0, vmax=len(cmap.colors), cmap=cmap
   ||        , interpolation='nearest')#Forma visual, sera eliminado
   ||       posteriormente
133||    plt.show()
134||    return data_file
135||def getMapRuta():
136||    data_file = np.genfromtxt(os.path.join(__location__, '
   ||       ESCOM_Ruta_creada.csv'), delimiter=',')
137||    return data_file.T #arreglo ordenado
138||
139||def getMapESCOM():
```

```
140     data_file = np.genfromtxt(os.path.join(__location__, '
        ESCOM_base.csv'), delimiter=',')
141     return data_file.T #arreglo ordenado
142 
143 def getMapESCOMObstaculo():
144     data_file = np.genfromtxt(os.path.join(__location__, '
        ESCOM_Obstaculo.csv'), delimiter=',')
145     return data_file.T #arreglo ordenado
146 
147 def getMapIPN():
148     data_file = np.genfromtxt(os.path.join(__location__, 'IPN.csv'
        ), delimiter=',')
149     return data_file.T
150 
151 def saveMap(data):#Salvando mapa despues de colocar puntos rojos
152     os.remove(os.path.join(__location__, 'ESCOM_Obstaculo.csv'))
153     aux = np.copy(data.T)
154     DF = pd.DataFrame(aux)
155     DF.to_csv(os.path.join(__location__, 'ESCOM_Obstaculo.csv'),
        index=False,  header=False )
156     colors = ["#002f68", "#4268c6", "#c70039", "#000000",
157             "#ECEC0A", "#348300", "#f1fce1", "#5A6051", "#97C363","
                #002f68"]  # use hex colors here, if desired.
158     cmap = ListedColormap(colors)
159 
160     plt.imshow(data, vmin=0, vmax=len(cmap.colors), cmap=cmap,
        interpolation='nearest')#Forma visual, sera eliminado
        posteriormente
161     plt.show()
162 
163 def saveRoute(data):#Salvando mapa despues de calcular ruta
164     aux = np.copy(data.T)
165     DF = pd.DataFrame(aux)
166     DF.to_csv(os.path.join(__location__, 'ESCOM_Ruta_creada.csv'),
         index=False,  header=False )
167     colors = ["#002f68", "#4268c6", "#c70039", "#000000",
168             "#ECEC0A", "#348300", "#f1fce1", "#5A6051", "#97C363","
                #002f68"]  # use hex colors here, if desired.
169     cmap = ListedColormap(colors)
170 
171     plt.imshow(data, vmin=0, vmax=len(cmap.colors), cmap=cmap,
        interpolation='nearest')#Forma visual, sera eliminado
        posteriormente
172     plt.show()
173 
174 def getFilePosCaminante():
175     with open(os.path.join(__location__, "caminante.txt")) as f:
176         data = json.dumps(json.loads(f.read()))
177         j = json.loads(data)
178         u = Caminante(**j)
179         if(u.orientacion == None):
180             u.orientacion = getFilePosCaminantePass().orientacion
181         return u
182 
183 def getFilePosCaminantePass():
```

```
184     with open(os.path.join(__location__, "caminantePass.txt")) as
            f:
185         data = json.dumps(json.loads(f.read()))
186         j = json.loads(data)
187         u = Caminante(**j)
188         return u
189 
190 def caminanteEnd():
191     x,y,salida=getPosFile(getUbicFinal())
192     xR,yR,salidaR=getPosFile("Robot")
193     if(x==xR and y==yR and salida==salidaR):
194         return True
195     return False
196 
197 if __name__ == '__main__':#Sirve 100% para una sola ruta, i.e, de
        punto a a punto b, no de punto a a varios puntos
198     #logicaPonerROjo(11)
199     getPosFile("Robot")#corremos Archivo.py como el principal
200 #   simularmatar()
```

12.2.2. Caminante

```
  1 import json
  2 import os
  3 import shutil
  4 
  5 __location__ = os.path.realpath(os.path.join(os.getcwd(), os.path
        .dirname(__file__)))
  6 
  7 class Caminante:
  8     def toJSON(self):
  9         if(self.orientacion != None):
 10             shutil.copy(os.path.join(__location__, 'caminante.txt'),
                    os.path.join(__location__, 'caminantePass.txt'))
 11         with open(os.path.join(__location__, 'caminante.txt'), 'w',
                encoding='utf-8') as f:
 12             f.write(json.dumps(self, default=lambda o: o.__dict__,
                    sort_keys=True, indent=4))
 13         return json.dumps(self, default=lambda o: o.__dict__,
                sort_keys=True, indent=4)
 14     def __init__(self, orientacion, pos_x, pos_y,fin_x,fin_y):
 15         self.orientacion = orientacion #N,S,E,W
 16         self.pos_x = pos_x
 17         self.pos_y = pos_y
 18         self.fin_x = fin_x
 19         self.fin_y = fin_y
 20 
 21     def __repr__(self):
 22         return "<Caminante Orientacion: %s Pos_x: %s Pos_y :%s
                fin_x: %s fin_y:%s>" % (self.orientacion, self.pos_x,
                self.pos_y, self.fin_x, self.fin_y)
 23 
 24     def __str__(self):
 25         return "<Caminante Orientacion: %s Pos_x: %s Pos_y: %s
                fin_x: %s fin_y:%s>" % (self.orientacion, self.pos_x,
```

```
        self.pos_y, self.fin_x, self.fin_y)
```

12.2.3. Lanzador

```
from main import lanzadorR
import sys

print('cmd entry:', sys.argv)
lanzadorR("Robot", str(sys.argv[1]), int(sys.argv[2]))#Inicio,
    final,tipo
```

12.2.4. main

```
#MAIN
import numpy as np
from random import randint
import os
import Caminante
from Archivo import setPIDFile, saveRoute,getGyroYaw, getMapESCOM
    ,getPosFile,getFilePosCaminante,getMapRuta,
    getMapESCOMObstaculo,setUbicFinal
from MoverCaminante import iniciarCaminata
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path
    .dirname(__file__)))

def getPID():
    pid = os.getpid()
    print("PID main", pid)
    return pid

def Simulador (gameState,origen,destino):
    nxC, nyC =70, 70
    running = True
    mins = np.zeros((nxC, nyC))
    cont = 0
    val = np.zeros((nxC, nyC))
    incre = 0
    aux = np.copy(gameState)
    count5 = 0 #Q5
    count8 = 0 #Q8
    testuwu = np.copy(gameState)
    def randomOffset():
        return randint(0, ((7 + 1) - 0)) + 0

    def getMoore(n_neigh, crit):
        x = 0
        if(n_neigh[0] == crit):
            x +=1
        if(n_neigh[1] == crit):
            x +=1
        if(n_neigh[2] == crit):
            x +=1
        if(n_neigh[3] == crit):
            x +=1
```

```
39 ||         if(n_neigh[4] == crit):
40 ||             x +=1
41 ||         if(n_neigh[5] == crit):
42 ||             x +=1
43 ||         if(n_neigh[6] == crit):
44 ||             x +=1
45 ||         if(n_neigh[7] == crit):
46 ||             x +=1
47 ||         return x
48 ||
49 ||     def MooreOffset(n_neigh, crit, orien):
50 ||         if(n_neigh[6] == crit and orien ==1):
51 ||             return True
52 ||         if(n_neigh[5] == crit and orien ==2):
53 ||             return True
54 ||         if(n_neigh[3] == crit and orien ==3):
55 ||             return True
56 ||         if(n_neigh[0] == crit and orien ==4):
57 ||             return True
58 ||         if(n_neigh[1] == crit and orien ==5):
59 ||             return True
60 ||         if(n_neigh[7] == crit and orien ==8):
61 ||             return True
62 ||         if(n_neigh[4] == crit and orien ==7):
63 ||             return True
64 ||         if(n_neigh[2] == crit and orien ==6):
65 ||             return True
66 ||         return False
67 ||
68 ||     def depfun(estado, x, y, n_neigh):
69 ||         if(estado == n_neigh[6] and mins[x][y+1] == 5):
70 ||             return True
71 ||         if(estado == n_neigh[5] and mins[x-1][y+1] == 6):
72 ||             return True
73 ||         if(estado == n_neigh[3]and mins[x-1][y] == 7):
74 ||             return True
75 ||         if(estado == n_neigh[0] and mins[x-1][y-1] == 8):
76 ||             return True
77 ||         if(estado == n_neigh[1] and mins[x][y-1] == 1):
78 ||             return True
79 ||         if(estado == n_neigh[2]and mins[x+1][y-1] == 2):
80 ||             return True
81 ||         if(estado == n_neigh[4] and mins[x+1][y] == 3):
82 ||             return True
83 ||         if(estado == n_neigh[7] and mins[x+1][y+1] == 4):
84 ||             return True
85 ||         return False
86 ||
87 ||     while running:
88 ||         testuwu = np.copy(gameState)
89 ||         incre +=1
90 ||         count5 = 0 #Q5
91 ||         count8 = 0 #Q8
92 ||         for x in range(0,nxC):
93 ||             for y in range(0,nyC):
```

```
94                  if(gameState[x][y] == 3):
95                      val[x][y] = incre+1
96          for x in range(0, nxC):
97              for y in range(0, nyC):
98                  n_neigh = np.array([gameState[(x-1) % nxC, (y-1)
                        % nyC],
99                                      gameState[(x) % nxC, (y-1) %
                                          nyC],
100                                     gameState[(x+1) % nxC, (y-1)
                                          % nyC],
101                                     gameState[(x-1) % nxC, (y) %
                                          nyC],
102                                     gameState[(x+1) % nxC, (y) %
                                          nyC],
103                                     gameState[(x-1) % nxC, (y+1)
                                          % nyC],
104                                     gameState[(x) % nxC, (y+1) %
                                          nyC],
105                                     gameState[(x+1) % nxC, (y+1)
                                          % nyC]])
106
107                 if gameState[x, y] == 5:
108                     count5 +=1
109                 if gameState[x, y] == 8:
110                     count8 +=1
111                 # Q0
112                 if gameState[x, y] == 0:
113                     if( mins[x,y] == 0 and ((randomOffset()+1)<7)
                            and (getMoore(n_neigh,4) != 0 or getMoore
                            (n_neigh,3) != 0 or getMoore(n_neigh,6) !=
                            0) ):
114                         aux[x, y] = 7
115                         cont+=1
116                         val[x, y] = cont
117
118                     if(getMoore(n_neigh,2) != 0):
119                         aux[x, y] = 9
120                 # Q1
121                 elif gameState[x, y] == 1:
122                     if(getMoore(n_neigh,5) != 0 or getMoore(
                            n_neigh,6)!=0):
123                         aux[x, y] = 6
124                 # Q2 periodico, Q3 periodico, Q4
125                 elif gameState[x, y] == 4: #Caso 0
126                     if (getMoore(n_neigh,0) == 0 and getMoore(
                            n_neigh,7) == 0):
127                         d1 = val[x,y+1]
128                         d2 = val[x-1,y+1]
129                         d3 = val[x-1,y]
130                         d4 = val[x-1,y-1]
131                         d5 = val[x,y-1]
132                         d6 = val[x+1,y-1]
133                         d7 = val[x+1,y]
134                         d8 = val[x+1,y+1]
135                         tempo = []
```

```
136 ||                          if( 3 == n_neigh[6] or 5 == n_neigh[6] or
    ||                              6 == n_neigh[6]):
137 ||                              tempo.insert(0,d1)
138 ||                          if( 3 == n_neigh[5] or 5 == n_neigh[5] or
    ||                              6 == n_neigh[5]):
139 ||                              tempo.insert(0,d2)
140 ||                          if( 3 == n_neigh[3] or 5 == n_neigh[3] or
    ||                              6 == n_neigh[3]):
141 ||                              tempo.insert(0,d3)
142 ||                          if( 3 == n_neigh[0] or 5 == n_neigh[0] or
    ||                              6 == n_neigh[0] ):
143 ||                              tempo.insert(0,d4)
144 ||                          if( 3 == n_neigh[1] or 5 == n_neigh[1] or
    ||                              6 == n_neigh[1]):
145 ||                                  tempo.insert(0,d5)
146 ||                          if( 3 == n_neigh[2] or 5 == n_neigh[2] or
    ||                              6 == n_neigh[2]):
147 ||                              tempo.insert(0,d6)
148 ||                          if( 3 == n_neigh[4] or 5 == n_neigh[4] or
    ||                              6 == n_neigh[4]):
149 ||                                  tempo.insert(0,d7)
150 ||                          if( 3 == n_neigh[7] or 5 == n_neigh[7] or
    ||                              6 == n_neigh[7] ):
151 ||                              tempo.insert(0,d8)
152 ||                          linp = np.array(tempo)
153 ||                          peq = 0
154 ||                          if linp.size != 0:
155 ||                              peq = np.min(linp)
156 ||                          if(d1 == peq and ( MooreOffset(n_neigh
    ||                             ,3,1) or MooreOffset(n_neigh,5,1) or
    ||                             MooreOffset(n_neigh,6,1))):
157 ||                              aux[x,y] = 5
158 ||                              mins[x,y] = 1
159 ||                          elif(d2 == peq and ( MooreOffset(n_neigh
    ||                             ,3,2) or MooreOffset(n_neigh,5,2) or
    ||                             MooreOffset(n_neigh,6,2))):
160 ||                              aux[x,y] = 5
161 ||                              mins[x,y] = 2
162 ||                          elif(d3 == peq and ( MooreOffset(n_neigh
    ||                             ,3,3) or MooreOffset(n_neigh,5,3) or
    ||                             MooreOffset(n_neigh,6,3))):
163 ||                              aux[x,y] = 5
164 ||                              mins[x,y] = 3
165 ||                          elif(d4 == peq and ( MooreOffset(n_neigh
    ||                             ,3,4) or MooreOffset(n_neigh,5,4) or
    ||                             MooreOffset(n_neigh,6,4))):
166 ||                              aux[x,y] = 5
167 ||                              mins[x,y] = 4
168 ||                          elif(d5 == peq and ( MooreOffset(n_neigh
    ||                             ,3,5) or MooreOffset(n_neigh,5,5) or
    ||                             MooreOffset(n_neigh,6,5))):
169 ||                              aux[x,y] = 5
170 ||                              mins[x,y] = 5
171 ||                          elif(d6 == peq and ( MooreOffset(n_neigh
    ||                             ,3,6) or MooreOffset(n_neigh,5,6) or
```

```
                                MooreOffset(n_neigh,6,6))):
172                              aux[x,y] = 5
173                              mins[x,y] = 6
174                          elif(d7 == peq and ( MooreOffset(n_neigh
                                ,3,7) or MooreOffset(n_neigh,5,7) or
                                MooreOffset(n_neigh,6,7))):
175                              aux[x,y] = 5
176                              mins[x,y] = 7
177                          elif(d8 == peq and ( MooreOffset(n_neigh
                                ,3,8) or MooreOffset(n_neigh,5,8) or
                                MooreOffset(n_neigh,6,8))):
178                              aux[x,y] = 5
179                              mins[x,y] = 8
180                  # Q5
181                  elif gameState[x, y] == 5:
182                      if(not depfun(8,x,y,n_neigh) and  not depfun
                            (5,x,y,n_neigh) and getMoore(n_neigh,6) ==
                             0 and getMoore(n_neigh,4) == 0 and
                            getMoore(n_neigh,1) == 0 and getMoore(
                            n_neigh,3) == 0 ):
183                              aux[x, y] = 0
184                              mins[x, y] = 0
185                      else:
186                              aux[x, y] = 8
187                  # Q6 periodico, Q7
188                  elif gameState[x, y] == 7:
189                      if( getMoore(n_neigh,5) > 0 or getMoore(
                            n_neigh,6) > 0 or getMoore(n_neigh,3) > 0
                            or getMoore(n_neigh,4) > 0):
190                              aux[x, y] = 4
191                  elif gameState[x, y] == 8:
192                      aux[x, y] = 5
193          gameState = np.copy(aux)
194          if(len(np.where(gameState==6)[0]) != 0 and len(np.where(
                gameState==4)[0]) == 0 and len(np.where(gameState==7)
                [0]) == 0 ):
195              temporalgs = np.copy(gameState)
196              temporalgs[temporalgs == 5] = 77
197              temporalgs[temporalgs == 8] = 5
198              temporalgs[temporalgs == 77] = 8
199              comparison = testuwu == temporalgs
200              equal_arrays = comparison.all()
201              if(equal_arrays):
202                  saveRoute(gameState)
203                  data = getMapRuta()
204                  iniPos_x, iniPos_y, salida = getPosFile(origen)
205                  finPos_x, finPos_y, entrada = getPosFile(destino)
206                  caminador = Caminante.Caminante(salida, iniPos_x,
                        iniPos_y,finPos_x,finPos_y)
207                  print(data[finPos_y][finPos_x])
208                  print(caminador.orientacion)
209                  if(iniciarCaminata(caminador, data, entrada)):
210                      print("Fin del caminante: \nDatos del
                            caminante")
211                      print("Orientacion final",caminador.
```

```
                        orientacion)
                    print("Coordenada x final",caminador.pos_x)
                    print("Coordenada y final",caminador.pos_y)
                break

def lanzadorR(lugarI, lugarF, tipo):
    setPIDFile(getPID())
    setUbicFinal(lugarF)
    data = None
    print(tipo)
    if(tipo == 0):#Usa el mapa basico
        data = getMapESCOM()
    elif(tipo == 1):#Usa el mapa con obstaculo
        data = getMapESCOMObstaculo()
    origen = lugarI
    destino = lugarF
    iniPos_x, iniPos_y, salida = getPosFile(origen)
    print(destino)
    finPos_x, finPos_y, entrada = getPosFile(destino)
    data[finPos_y][finPos_x] = 1 #fin
    data[iniPos_y][iniPos_x] = 3 #inicio
    gameState = data
    saveRoute(gameState)
    Simulador(gameState,origen,destino)
```

12.2.5. MoverCaminante

```
from Movimientos import *
import time
import copy
import os,signal
import Archivo
import subprocess
__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path
    .dirname(__file__)))

def KillProcessCaminante(pid):#Funcionando
    try:
        os.kill(pid, signal.SIGKILL)
        Archivo.deletePIDFile()
        print("Process Successfully terminated")
    except:
        print("Error Encountered while running script")

def moore(data,pos_y, pos_x): #data[y][x]
    nw = data[pos_y-1][pos_x-1]
    n = data[pos_y-1][pos_x]
    ne = data[pos_y-1][pos_x+1]
    w = data[pos_y][pos_x-1]
    e = data[pos_y][pos_x+1]
    sw = data[pos_y+1][pos_x-1]
    s = data[pos_y+1][pos_x]
    se = data[pos_y+1][pos_x+1]
    print(nw,n,ne)
    print(w,data[pos_y][pos_x],e)
```

```
28 ||    print(sw,s,se)
29 ||    return [[nw,n,ne],[w,data[pos_y][pos_x],e],[sw,s,se]]
30 ||
31 ||def testQuitarNull():
32 ||    objCaminantePass = Archivo.getFilePosCaminantePass()
33 ||    return objCaminantePass.orientacion
34 ||
35 ||def test(objT,fin):
36 ||    valX = objT.pos_x
37 ||    valY = objT.pos_y
38 ||    if(fin == "W"):
39 ||       boolCoorPassNU = Archivo.compareCoorPasCoor(valY-1,valX-1)
40 ||       boolCoorPassD = Archivo.compareCoorPasCoor(valY,valX-1)
41 ||       boolCoorPassND = Archivo.compareCoorPasCoor(valY+1,valX-1)
42 ||       if(boolCoorPassNU or boolCoorPassD or boolCoorPassND):
43 ||          return True#Ya pasamos por ahi
44 ||       return False#No pasamos por ahi
45 ||    elif(fin == "E"):
46 ||       boolCoorPassNU = Archivo.compareCoorPasCoor(valY-1,valX+1)
47 ||       boolCoorPassD = Archivo.compareCoorPasCoor(valY,valX+1)
48 ||       boolCoorPassND = Archivo.compareCoorPasCoor(valY+1,valX+1)
49 ||       if(boolCoorPassNU or boolCoorPassD or boolCoorPassND):
50 ||          return True#Ya pasamos por ahi
51 ||       return False#No pasamos por ahi
52 ||    elif(fin == "N"):
53 ||       boolCoorPassNW = Archivo.compareCoorPasCoor(valY-1,valX-1)
54 ||       boolCoorPassD = Archivo.compareCoorPasCoor(valY-1,valX)
55 ||       boolCoorPassNE = Archivo.compareCoorPasCoor(valY-1,valX+1)
56 ||       if(boolCoorPassNW or boolCoorPassD or boolCoorPassNE):
57 ||          return True#Ya pasamos por ahi
58 ||       return False#No pasamos por ahi
59 ||    else:
60 ||       boolCoorPassNW = Archivo.compareCoorPasCoor(valY+1,valX-1)
61 ||       boolCoorPassD = Archivo.compareCoorPasCoor(valY+1,valX)
62 ||       boolCoorPassNE = Archivo.compareCoorPasCoor(valY+1,valX+1)
63 ||       if(boolCoorPassNW or boolCoorPassD or boolCoorPassNE):
64 ||          return True#Ya pasamos por ahi
65 ||       return False#No pasamos por ahi
66 ||
67 ||def orientarRobot(obj, fin):
68 ||    if(fin == "W"):
69 ||       if(obj.orientacion == "N"):
70 ||          horientarIzquierda()
71 ||       elif(obj.orientacion == "S"):
72 ||          horientarDerecha()
73 ||       elif(obj.orientacion == "E"):
74 ||          horientarIzquierda()
75 ||          horientarIzquierda()
76 ||    elif(fin == "E"):
77 ||       if(obj.orientacion == "N"):
78 ||          horientarDerecha()
79 ||       elif(obj.orientacion == "S"):
80 ||          horientarIzquierda()
81 ||       elif(obj.orientacion == "W"):
82 ||          horientarIzquierda()
```

```
83 ||           horientarIzquierda()
84 ||    elif(fin == "N"):
85 ||       if(obj.orientacion == "E"):
86 ||           horientarIzquierda()
87 ||       elif(obj.orientacion == "W"):
88 ||           horientarDerecha()
89 ||       elif(obj.orientacion == "S"):
90 ||           horientarIzquierda()
91 ||           horientarIzquierda()
92 ||    else:
93 ||       if(obj.orientacion == "E"):
94 ||           horientarDerecha()
95 ||       elif(obj.orientacion == "W"):
96 ||           horientarIzquierda()
97 ||       elif(obj.orientacion == "N"):
98 ||           horientarIzquierda()
99 ||           horientarIzquierda()
100||    obj.orientacion = fin
101||
102||def orientarRobotLogico(obj,vecindad):
103||    newObj = copy.copy(obj)
104||    print("Logico")
105||    if(obj.orientacion == "N"):
106||       if( obj.fin_x <= obj.pos_x ): #Fin a la izq
107||           newObj.orientacion = "W"
108||           straight = goStraight(vecindad, newObj)
109||           diagonal = goDiagonal(vecindad, newObj)
110||           if((straight or diagonal) and (not test(newObj,newObj.
    ||              orientacion))):
111||               return "W"
112||       if(obj.fin_x > obj.pos_x):#Fin a la der
113||           newObj.orientacion = "E"
114||           straight = goStraight(vecindad, newObj)
115||           diagonal = goDiagonal(vecindad, newObj)
116||           if((straight or diagonal) and (not test(newObj,newObj.
    ||              orientacion))):
117||               return "E"
118||       newObj.orientacion = "E"
119||       straight = goStraight(vecindad, newObj)
120||       diagonal = goDiagonal(vecindad, newObj)
121||       if((straight or diagonal) and (not test(newObj,newObj.
    ||          orientacion))):
122||           return "E"
123||       newObj.orientacion = "W"
124||       straight = goStraight(vecindad, newObj)
125||       diagonal = goDiagonal(vecindad, newObj)
126||       if((straight or diagonal) and (not test(newObj,newObj.
    ||          orientacion))):
127||           return "W"
128||       return "S"
129||    elif(obj.orientacion == "S"):
130||       if( obj.fin_x <= obj.pos_x ): #Fin a la izq
131||           newObj.orientacion = "W"
132||           straight = goStraight(vecindad, newObj)
133||           diagonal = goDiagonal(vecindad, newObj)
```

```
134 ||          if((straight or diagonal) and (not test(newObj,newObj.
    ||             orientacion))):
135 ||             return "W"
136 ||       if(obj.fin_x > obj.pos_x):#Fin a la der
137 ||          newObj.orientacion = "E"
138 ||          straight = goStraight(vecindad, newObj)
139 ||          diagonal = goDiagonal(vecindad, newObj)
140 ||          if((straight or diagonal) and (not test(newObj,newObj.
    ||             orientacion))):
141 ||             return "E"
142 ||       newObj.orientacion = "E"
143 ||       straight = goStraight(vecindad, newObj)
144 ||       diagonal = goDiagonal(vecindad, newObj)
145 ||       if((straight or diagonal) and (not test(newObj,newObj.
    ||          orientacion))):
146 ||          return "E"
147 ||       newObj.orientacion = "W"
148 ||       straight = goStraight(vecindad, newObj)
149 ||       diagonal = goDiagonal(vecindad, newObj)
150 ||       if((straight or diagonal) and (not test(newObj,newObj.
    ||          orientacion))):
151 ||          return "W"
152 ||       return "N"
153 ||    elif(obj.orientacion == "E"):
154 ||       if( obj.fin_y <= obj.pos_y ): #Fin a la izq
155 ||          newObj.orientacion = "N"
156 ||          straight = goStraight(vecindad, newObj)
157 ||          diagonal = goDiagonal(vecindad, newObj)
158 ||          if((straight or diagonal) and (not test(newObj,newObj.
    ||             orientacion))):
159 ||             return "N"
160 ||       if( obj.fin_y > obj.pos_y ):#Fin a la der
161 ||          newObj.orientacion = "S"
162 ||          straight = goStraight(vecindad, newObj)
163 ||          diagonal = goDiagonal(vecindad, newObj)
164 ||          if((straight or diagonal) and (not test(newObj,newObj.
    ||             orientacion))):
165 ||             return "S"
166 ||       newObj.orientacion = "N"
167 ||       straight = goStraight(vecindad, newObj)
168 ||       diagonal = goDiagonal(vecindad, newObj)
169 ||       if((straight or diagonal) and (not test(newObj,newObj.
    ||          orientacion))):
170 ||          return "N"
171 ||       newObj.orientacion = "S"
172 ||       straight = goStraight(vecindad, newObj)
173 ||       diagonal = goDiagonal(vecindad, newObj)
174 ||       if((straight or diagonal) and (not test(newObj,newObj.
    ||          orientacion))):
175 ||          return "S"
176 ||       return "W"
177 ||    elif(obj.orientacion == "W"):
178 ||       if( obj.fin_y <= obj.pos_y ): #Fin a la izq
179 ||          newObj.orientacion = "N"
180 ||          straight = goStraight(vecindad, newObj)
```

```
181 ||          diagonal = goDiagonal(vecindad, newObj)
182 ||          if((straight or diagonal) and (not test(newObj,newObj.
    ||             orientacion))):
183 ||             return "N"
184 ||       if( obj.fin_y > obj.pos_y ):#Fin a la der
185 ||          newObj.orientacion = "S"
186 ||          straight = goStraight(vecindad, newObj)
187 ||          diagonal = goDiagonal(vecindad, newObj)
188 ||          if((straight or diagonal) and (not test(newObj,newObj.
    ||             orientacion))):
189 ||             return "S"
190 ||       newObj.orientacion = "N"
191 ||       straight = goStraight(vecindad, newObj)
192 ||       diagonal = goDiagonal(vecindad, newObj)
193 ||       if((straight or diagonal) and (not test(newObj,newObj.
    ||          orientacion))):
194 ||          return "N"
195 ||       newObj.orientacion = "S"
196 ||       straight = goStraight(vecindad, newObj)
197 ||       diagonal = goDiagonal(vecindad, newObj)
198 ||       if((straight or diagonal) and (not test(newObj,newObj.
    ||          orientacion))):
199 ||          return "S"
200 ||       return "E"
201 ||
202 ||
203 ||def goStraight(vecindad, obj):
204 ||    if(obj.orientacion == "N"):
205 ||       if(vecindad[0][1] == 5 or vecindad[0][1] == 8):
206 ||          return True
207 ||       else:
208 ||          return False
209 ||    elif(obj.orientacion == "S"):
210 ||       if(vecindad[2][1] == 5 or vecindad[2][1] == 8):
211 ||          return True
212 ||       else:
213 ||          return False
214 ||    elif(obj.orientacion == "W"):
215 ||       if(vecindad[1][0] == 5 or vecindad[1][0] == 8):
216 ||          return True
217 ||       else:
218 ||          return False
219 ||    elif(obj.orientacion == "E"):
220 ||       if(vecindad[1][2] == 5 or vecindad[1][2] == 8):
221 ||          return True
222 ||       else:
223 ||          return False
224 ||
225 ||def cambiarXYCaminante(obj,direc):
226 ||    if(obj.orientacion == "N"):
227 ||       obj.pos_y = obj.pos_y - 1
228 ||       if(direc == "Izq"):
229 ||          obj.pos_x = obj.pos_x - 1
230 ||          orientarRobot(obj,"W")
231 ||          moverMotorAdelante()
```

```
232 ||          orientarRobot(obj,"N")
233 ||          moverMotorAdelante()
234 ||       elif(direc == "Der"):
235 ||          obj.pos_x = obj.pos_x + 1
236 ||          orientarRobot(obj,"E")
237 ||          moverMotorAdelante()
238 ||          orientarRobot(obj,"N")
239 ||          moverMotorAdelante()
240 ||    elif(obj.orientacion == "S"):
241 ||       obj.pos_y = obj.pos_y + 1
242 ||       if(direc == "Izq"):
243 ||          obj.pos_x = obj.pos_x + 1
244 ||          orientarRobot(obj,"E")
245 ||          moverMotorAdelante()
246 ||          orientarRobot(obj,"S")
247 ||          moverMotorAdelante()
248 ||       elif(direc == "Der"):
249 ||          obj.pos_x = obj.pos_x - 1
250 ||          orientarRobot(obj,"W")
251 ||          moverMotorAdelante()
252 ||          orientarRobot(obj,"S")
253 ||          moverMotorAdelante()
254 ||    elif(obj.orientacion == "W"):
255 ||       obj.pos_x = obj.pos_x - 1
256 ||       if(direc == "Izq"):
257 ||          obj.pos_y = obj.pos_y + 1
258 ||          orientarRobot(obj,"S")
259 ||          moverMotorAdelante()
260 ||          orientarRobot(obj,"W")
261 ||          moverMotorAdelante()
262 ||       elif(direc == "Der"):
263 ||          obj.pos_y = obj.pos_y - 1
264 ||          orientarRobot(obj,"N")
265 ||          moverMotorAdelante()
266 ||          orientarRobot(obj,"W")
267 ||          moverMotorAdelante()
268 ||    elif(obj.orientacion == "E"):
269 ||       obj.pos_x = obj.pos_x + 1
270 ||       if(direc == "Izq"):
271 ||          obj.pos_y = obj.pos_y - 1
272 ||          orientarRobot(obj,"N")
273 ||          moverMotorAdelante()
274 ||          orientarRobot(obj,"E")
275 ||          moverMotorAdelante()
276 ||       elif(direc == "Der"):
277 ||          obj.pos_y = obj.pos_y + 1
278 ||          orientarRobot(obj,"S")
279 ||          moverMotorAdelante()
280 ||          orientarRobot(obj,"E")
281 ||          moverMotorAdelante()
282 ||    Archivo.setCoorPas(obj.pos_x,obj.pos_y)
283 ||
284 ||def goDiagonal(vecindad, obj):#Daremos prioridad a la esquina mas
    ||     cerca del punto final o con mas dependecias
285 ||    existe = False
```

```
286 ||    if(obj.orientacion == "N"):
287 ||       if((vecindad[0][0] == 5 or vecindad[0][0] == 8)):
288 ||          if(obj.fin_x <= obj.pos_x):
289 ||             return "Izq"
290 ||          else:
291 ||             existe = "Izq"
292 ||       if((vecindad[0][2] == 5 or vecindad[0][2] == 8)):
293 ||          if(obj.fin_x >= obj.pos_x):
294 ||             return "Der"
295 ||          else:
296 ||             existe = "Der"
297 ||       return existe
298 ||    elif(obj.orientacion == "S"):
299 ||       if((vecindad[2][0] == 5 or vecindad[2][0] == 8)):
300 ||          if(obj.fin_x <= obj.pos_x):
301 ||             return "Der"
302 ||          else:
303 ||             existe = "Der"
304 ||       if((vecindad[2][2] == 5 or vecindad[2][2] == 8)):
305 ||          if(obj.fin_x >= obj.pos_x):
306 ||             return "Izq"
307 ||          else:
308 ||             existe = "Izq"
309 ||       return existe
310 ||    elif(obj.orientacion == "W"):
311 ||       if((vecindad[0][0] == 5 or vecindad[0][0] == 8)):
312 ||          if(obj.fin_y <= obj.pos_y):
313 ||             return "Der"
314 ||          else:
315 ||             existe = "Der"
316 ||       if((vecindad[2][0] == 5 or vecindad[2][0] == 8)):
317 ||          if(obj.fin_y >= obj.pos_y):
318 ||             return "Izq"
319 ||          else:
320 ||             existe = "Izq"
321 ||       return existe
322 ||    elif(obj.orientacion == "E"):
323 ||       if((vecindad[0][2] == 5 or vecindad[0][2] == 8)):
324 ||          if(obj.fin_y <= obj.pos_y):
325 ||             return "Izq"
326 ||          else:
327 ||             existe = "Izq"
328 ||       if((vecindad[2][2] == 5 or vecindad[2][2] == 8)):
329 ||          if(obj.fin_y >= obj.pos_y):
330 ||             return "Der"
331 ||          else:
332 ||             existe = "Der"
333 ||       return existe
334 ||
335 ||def finalizarCaminante(vecindad, obj,orientacionSalFinal):#
    ||    Optimizable
336 ||    if(obj.orientacion == "N"):
337 ||       if(vecindad[0][0] == 6):
338 ||          cambiarXYCaminante(obj, "Izq")
339 ||       elif(vecindad[0][1] == 6):
```

```
340 ||          moverMotorAdelante()
341 ||          cambiarXYCaminante(obj, "straight")
342 ||       elif(vecindad[0][2] == 6):
343 ||          cambiarXYCaminante(obj, "Der")
344 ||       else:
345 ||          orientarRobot(obj,"S")
346 ||          finalizarCaminante(vecindad, obj,orientacionSalFinal)
347 ||    elif(obj.orientacion == "S"):
348 ||       if(vecindad[2][0] == 6):
349 ||          cambiarXYCaminante(obj, "Der")
350 ||       elif(vecindad[2][1] == 6):
351 ||          moverMotorAdelante()
352 ||          cambiarXYCaminante(obj, "straight")
353 ||       elif(vecindad[2][2] == 6):
354 ||          cambiarXYCaminante(obj, "Izq")
355 ||       else:
356 ||          orientarRobot(obj,"W")
357 ||          finalizarCaminante(vecindad, obj,orientacionSalFinal)
358 ||    elif(obj.orientacion == "W"):
359 ||       if(vecindad[0][0] == 6):
360 ||          cambiarXYCaminante(obj, "Der")
361 ||       elif(vecindad[1][0] == 6):
362 ||          moverMotorAdelante()
363 ||          cambiarXYCaminante(obj, "straight")
364 ||       elif(vecindad[2][0] == 6):
365 ||          cambiarXYCaminante(obj, "Izq")
366 ||       else:
367 ||          orientarRobot(obj,"E")
368 ||          finalizarCaminante(vecindad, obj,orientacionSalFinal)
369 ||    elif(obj.orientacion == "E"):
370 ||       if(vecindad[0][2] == 6):
371 ||          cambiarXYCaminante(obj, "Izq")
372 ||       elif(vecindad[1][2] == 6):
373 ||          moverMotorAdelante()
374 ||          cambiarXYCaminante(obj, "straight")
375 ||       elif(vecindad[2][2] == 6):
376 ||          cambiarXYCaminante(obj, "Der")
377 ||       else:
378 ||          orientarRobot(obj,"W")
379 ||          finalizarCaminante(vecindad, obj,orientacionSalFinal)
380 ||    orientarRobot(obj,orientacionSalFinal)
381 ||
382 ||def iniciarCaminata(obj, mapa, orientacionSalFinal,xd):
383 ||    pasos = 0
384 ||    print(orientacionSalFinal)
385 ||    while True:
386 ||       vecindad = moore(mapa,obj.pos_y,obj.pos_x)
387 ||       if(obj.orientacion == None):
388 ||          obj.orientacion = testQuitarNull()
389 ||       print(obj.orientacion)
390 ||       if(6 in vecindad[0] or 6 in vecindad[1] or 6 in vecindad
    ||          [2]):
391 ||          print(orientacionSalFinal)
392 ||          finalizarCaminante(vecindad, obj, orientacionSalFinal)
393 ||          print(obj.toJSON())
```

```
394 ||          moore(mapa,obj.pos_y,obj.pos_x)
395 ||          break
396 ||       straight = goStraight(vecindad, obj)
397 ||       diagonal = goDiagonal(vecindad, obj)
398 ||       if(straight):
399 ||          moverMotorAdelante()
400 ||          cambiarXYCaminante(obj,"straight")
401 ||       else:
402 ||          if(diagonal):
403 ||             cambiarXYCaminante(obj,diagonal)
404 ||             print("diagonal directa")
405 ||          else:
406 ||             orientacion = orientarRobotLogico(obj,vecindad)
407 ||             print(orientacion)
408 ||             orientarRobot(obj,orientacion)
409 ||       print(obj.toJSON())
410 ||       time.sleep(0.5)#dar tiempo a sensores. Modificar por mas
    ||          tiempo si es necesario 5s
411 ||       print("#######################################")
412 ||
413 ||       if(pasos == 2 and xd):
414 ||          KillProcessCaminante(int(Archivo.getPIDFile()))
415 ||          robotObstaculo = subprocess.Popen(['python3', os.path.
    ||             join(__location__, "Pruebas.py")] + [Archivo.
    ||             getUbicFinal(),"1"])
416 ||          break
417 ||       pasos += 1
418 ||    return True
```

12.2.6. Movimientos

```
 1 ||from multiprocessing import Process
 2 ||from MotoresDC import *
 3 ||import os
 4 ||__location__ = os.path.realpath(os.path.join(os.getcwd(), os.path
   ||    .dirname(__file__)))
 5 ||
 6 ||def getGyroYaw():#Obtenemos el valor yaw del giroscopio con tipo
   ||    string
 7 ||    f = open(os.path.join(__location__, "GyroYawdata.txt"))
 8 ||    lines = f.readlines()
 9 ||    return lines[0].strip(),lines[1]
10 ||
11 ||def detectaDesviacion():
12 ||    direc, yawValueFile = getGyroYaw()
13 ||    if(direc != 'Ok'):
14 ||       yawValue = float(yawValueFile)
15 ||       print(direc)
16 ||       print(yawValue)
17 ||       if(direc == "N"):
18 ||          if(yawValue > 0):
19 ||             moverMotorDer(False,(yawValue*0.3)/2.8459 )
20 ||          else:
21 ||             yawValue = yawValue * -1
```

```
22 ||             moverMotorIzq(False,(yawValue*0.3)/ 1.6578 )
23 ||       elif(direc == "W"):
24 ||          if(yawValue > 0):
25 ||             moverMotorDer(False,(yawValue*0.3)/ 4.4 )
26 ||          else:
27 ||             yawValue = yawValue * -1
28 ||             moverMotorIzq(False,(yawValue*0.3)/ 1.6578 )
29 ||       if(direc == "E"):
30 ||          if(yawValue > 0):
31 ||             moverMotorDer(False,(yawValue*0.3)/2.86 )
32 ||          else:
33 ||             yawValue = yawValue * -1
34 ||             moverMotorIzq(False,(yawValue*0.3)/ 1.6578 )
35 ||       if(direc == "SP"):
36 ||          if(yawValue < 0):
37 ||             yawValue = yawValue * -1
38 ||             moverMotorIzq(False,(yawValue*0.3)/ 1.8 )
39 ||          else:
40 ||             moverMotorIzq(False,(yawValue*0.3)/ 1.8 )
41 ||       if(direc == "SN"):
42 ||          if(yawValue < 0):
43 ||             yawValue = yawValue * -1
44 ||             moverMotorDer(False,(yawValue*0.3)/ 1.95 )
45 ||          else:
46 ||             moverMotorDer(False,(yawValue*0.3)/ 1.95 )
   ||
47 ||def moverMotorAdelante(tmp=4,ext=False):
48 ||    print("Moviendo hacia delante")
49 ||    p1 = Process(target=motorIzq, args=(True,tmp,))
50 ||    p2 = Process(target=motorDer, args=(True,tmp,))
51 ||    p1.start()
52 ||    p2.start()
53 ||    p1.join()
54 ||    p2.join()
55 ||    if(not ext):
56 ||       detectaDesviacion()
57 ||
58 ||def moverMotorAtras(tmp=4,ext=False):
59 ||    print("Moviendo hacia atras")
60 ||    p1 = Process(target=motorIzq, args=(False,tmp,))
61 ||    p2 = Process(target=motorDer, args=(False,tmp,))
62 ||    p1.start()
63 ||    p2.start()
64 ||    p1.join()
65 ||    p2.join()
66 ||    if(not ext):
67 ||       detectaDesviacion()
68 ||
69 ||def moverMotorIzq(sentido, tmp=4):
70 ||    print("Moviendo motor izq", sentido)
71 ||    motorIzq(sentido, tmp)
72 ||
73 ||def moverMotorDer(sentido, tmp=4):
74 ||    print("Movimiento motor der", sentido)
75 ||    motorDer(sentido, tmp)
```

```
76 ||
77 ||def horientarIzquierda():
78 ||    moverMotorIzq(True,4)
79 ||    moverMotorDer(False,3.55)
80 ||    moverMotorAtras(1.4,True)
81 ||    print("Horientando hacia la izquierda")
82 ||    detectaDesviacion()
83 ||
84 ||def horientarDerecha():
85 ||    moverMotorDer(True,4)
86 ||    moverMotorIzq(False,3.3)
87 ||    moverMotorAtras(1.35,True)
88 ||    print("Horientando hacia la derecha")
89 ||    detectaDesviacion()
90 ||
91 ||if __name__ == '__main__':
92 ||    #moverMotorAdelante()
93 ||    #moverMotorAtras()
94 ||    #horientarIzquierda()
95 ||    #horientarDerecha()
96 ||    #horientarDerecha()
97 ||    detectaDesviacion()
```

12.2.7. MotoresDC

```
 1 ||import RPi.GPIO as GPIO
 2 ||import time
 3 ||
 4 ||def motorIzq(sentido, tiempo=4):
 5 ||    GPIO.setmode(GPIO.BCM)
 6 ||    GPIO.setwarnings(False)
 7 ||    #Right motor pines
 8 ||    R_RPWM = 19;  # GPIO pin 19 to the RPWM on the BTS7960 (right
   ||        controller)
 9 ||    R_LPWM = 26;  # GPIO pin 26 to the LPWM on the BTS7960 (right
   ||        controller)
10 ||    # For enabling "Left" and "Right" movement
11 ||    R_L_EN = 6;  # connect GPIO pin 6 to L_EN on the BTS7960 (
   ||       right controller)
12 ||    R_R_EN = 13;  # connect GPIO pin 13 to R_EN on the BTS7960 (
   ||       right controller)
13 ||
14 ||    # Set all of our PINS to output Right
15 ||    GPIO.setup(R_RPWM, GPIO.OUT)
16 ||    GPIO.setup(R_LPWM, GPIO.OUT)
17 ||    GPIO.setup(R_L_EN, GPIO.OUT)
18 ||    GPIO.setup(R_R_EN, GPIO.OUT)
19 ||
20 ||    # Enable "Left" and "Right" movement on the HBRidge R
21 ||    GPIO.output(R_R_EN, True)
22 ||    GPIO.output(R_L_EN, True)
23 ||
24 ||    rpwm= GPIO.PWM(R_RPWM, 500)
25 ||    lpwm= GPIO.PWM(R_LPWM, 500)
26 ||
```

```
27 ||     if(not sentido): #sentido R
28 ||         rpwm.start(0)
29 ||         rpwm.ChangeDutyCycle(20)
30 ||         time.sleep(tiempo)
31 ||     else:#sentido 1
32 ||         lpwm.start(0)
33 ||         lpwm.ChangeDutyCycle(20)
34 ||         time.sleep(tiempo)
35 ||
36 ||     GPIO.output(R_RPWM, GPIO.LOW)
37 ||     GPIO.output(R_LPWM, GPIO.LOW)
38 ||     GPIO.output(R_R_EN, GPIO.LOW)
39 ||     GPIO.output(R_L_EN, GPIO.LOW)
40 ||     GPIO.cleanup()
41 ||
42 ||def motorDer(sentido, tiempo=4):
43 ||     GPIO.setmode(GPIO.BCM)
44 ||     GPIO.setwarnings(False)
45 ||     #Left motor pines
46 ||     L_RPWM = 20;  # GPIO pin 19 to the RPWM on the BTS7960 (left
   ||        controller)
47 ||     L_LPWM = 21;  # GPIO pin 26 to the LPWM on the BTS7960 (left
   ||        controller)
48 ||     # For enabling "Left" and "Right" movement
49 ||     L_L_EN = 12;  # connect GPIO pin 20 to L_EN on the BTS7960 (
   ||        left controller)
50 ||     L_R_EN = 16;  # connect GPIO pin 21 to R_EN on the BTS7960 (
   ||        left controller)
51 ||
52 ||
53 ||     # Set all of our PINS to output left
54 ||     GPIO.setup(L_RPWM, GPIO.OUT)
55 ||     GPIO.setup(L_LPWM, GPIO.OUT)
56 ||     GPIO.setup(L_L_EN, GPIO.OUT)
57 ||     GPIO.setup(L_R_EN, GPIO.OUT)
58 ||     # Enable "Left" and "Right" movement on the HBRidge L
59 ||     GPIO.output(L_R_EN, True)
60 ||     GPIO.output(L_L_EN, True)
61 ||     rpwm= GPIO.PWM(L_RPWM, 500)
62 ||     lpwm= GPIO.PWM(L_LPWM, 500)
63 ||
64 ||     if(not sentido): #sentido R
65 ||         rpwm.start(0)
66 ||         rpwm.ChangeDutyCycle(20)
67 ||         time.sleep(tiempo)
68 ||     else:#sentido 1
69 ||         lpwm.start(0)
70 ||         lpwm.ChangeDutyCycle(20)
71 ||         time.sleep(tiempo)
72 ||
73 ||     GPIO.output(L_RPWM, GPIO.LOW)
74 ||     GPIO.output(L_LPWM, GPIO.LOW)
75 ||     GPIO.output(L_R_EN, GPIO.LOW)
76 ||     GPIO.output(L_L_EN, GPIO.LOW)
77 ||     GPIO.cleanup()
```

```
78 ||
79 ||
80 ||if __name__ == '__main__':
81 ||    #moverMotorAdelante()
82 ||    #GPIO.cleanup()
83 ||    motorDer(False)
```

[1] Adamatzky, A. (2007). Physarum machine: implementation of a Kolmogorov-Uspensky machine on a biological substrate. Parallel Processing Letters, 17(04), 455-467.

[2] Adamatzky, A. (2010). Physarum machines: computers from slime mould (Vol. 74). World Scientific. Chicago

[3] Adamatzky, A. (2012). Bioevaluation of World Transport Networks. World Scientific Publishing Company.

[4] Adamatzky, A. (2016). Advances in Physarum Machines: Sensing and Computing with Slime Mould (Emergence, Complexity and Computation, 21) (1st ed. 2016). Springer.

[5] Adamatzky, A., Martínez, G.J., Chapa-Vergara, S.V. et al. Approximating Mexican highways with slime mould. Nat Comput 10, 1195–1214 (2011). https://doi.org/10.1007/s11047-011-9255-z

[6] Adamatzky, A., & Martinez, G. J. (2016). Recolonisation of USA: Slime Mould on 3D Terrains. Advances in Physarum Machines, págs. 337–348. https://doi.org/10.1007/978-3-319-26662-6_17

[7] Boccara, N. (2010). Modeling complex systems. Springer Science & Business Media.

[8] Chazelle, B. (2012). Natural algorithms and influence systems. Communications of the ACM, 55(12), 101–110. https://doi.org/10.1145/2380656.2380679

[9] D. Wuttke A. Mitschele-Thiel. Digital Systems Design. DesIRE. https://ec.europa.eu/programmes/erasmus-plus/project-result-content/9f367412-e981-4a64-b01a-1157cbc933f5/Digital%20Systems%20Design.pdf

[10] E. Y. Marín Alavez. Modelado del Physarum Polycephalum implementado en robot basado en autómatas celulares. ESCOM, IPN: B.S. Thesis, Escuela Superior de Cómputo,IPN, 2018. https://www.comunidad.escom.ipn.mx/genaro/Papers/Thesis_files/ThesisYairMarinAlavez2018.pdf

[11] Gantenbein, D. (2020, noviembre 02). How Amazon scientists are helping the Scout delivery device find a path to success. Amazon Science. https://www.amazon.science/latest-news/how-amazon-scientists-are-helping-the-scout-delivery-device-find-a-path-to-success

[12] Gunji, Y. P., Shirakawa, T., Niizato, T., & Haruna, T. (2008). Minimal model of a cell connecting amoebic motion and adaptive transport networks. Journal of Theoretical Biology, 253(4), págs. 659–667. https://doi.org/10.1016/j.jtbi.2008.04.017

[13] iProUP. (2021, abril 18). Robots para delivery: mirá como esta realidad que ya llegó. iProUP. https://www.iproup.com/innovacion/22173-robots-para-delivery-mira-como-esta-realidad-que-ya-llego

[14] J. L. Rosas Trigueros. Class Lecture, Topic: "Evolutionary Computing: Cellular Automata." ESCOM, IPN, 2021.

[15] Jarema, R. (2018, August 23). 10 Steps to Choosing the Right Motors for Your Robotic Project. Medium. https://medium.com/husarion-blog/10-steps-to-choosing-the-right-motors-for-your-robotic-project-bf5c4b997407

[16] Jiménez Cano, R. (2018, enero 23). Amazon Go, la tienda sin dependientes donde no se pasa por caja. El País. https://elpais.com/tecnologia/2018/01/22/actualidad/1516601138_966659.html

[17] Jones, J. (2015). From Pattern Formation to Material Computation: Multi-agent Modelling of Physarum Polycephalum. Springer Publishing. págs. 33-59, 111, 261.

[18] Jones, J., & Adamatzky, A. (2012). Emergence of self-organized amoeboid movement in a multi-agent approximation of Physarum polycephalum. ArXiv E-Prints, arXiv:1212.0023. http://arxiv.org/abs/1212.0023

[19] Mainzer, K., & Chua, L. (2011). The universe as automaton: From simplicity and symmetry to complexity (Vol. 1). Springer Science & Business Media.

[20] Martínez, G. J., Adamatzky, A., Figueroa, R. Q., Schweikardt, E., Zaitsev, D. A., Zelinka, I., & Oliva-Moreno, L. N. (2022). Computing with Modular Robots. International Journal of Unconventional Computing, 17.

[21] Martínez, G. J., Adamatzky, A., & Alonso-Sanz, R. (2013). Designing complex dynamics in cellular automata with memory. International Journal of Bifurcation and Chaos, 23(10), 1330035.

[22] Martínez, G. J., Zenil, H., & Stephens, C. R. S. (Eds.). (2011). Sistemas Complejos como modelos de Computación. Luniver Press.

[23] Martínez, G. J., & Moreno, L. N. O. (2013). ¿Desarrollo de Robótica en México?/Robotics in Mexico?. ESCOM, National Polytechnic Institute of Mexico.

[24] Mitchell, M. (2009). Complexity: A guided tour. Oxford university press.

[25] Nakagaki, T. (2000, septiembre 28). Maze-solving by an amoeboid organism. Nature. https://www.nature.com/articles/35035159?error=cookies_not_supported&code=d4aaf852-0709-469c-a6c2-b0528a35f1a1

[26] Renner, B. (2006). Slime Mold Reproduction. http://bioweb.uwlax.edu/bio203/2010/renner_brad/reproduction.htm

[27] Sommerville, I. (2011). Ingeniería de software (Spanish Edition). Pearson (México).

[28] Soyding. (n.d.). Raspberry Pi boot sequence. http://myembeddedlinux.blogspot.com/2013/05/raspberry-pi-boot-sequence.html

[29] TerkRecoms - Tech TV. (2019, marzo 01). Top 6 Delivery Robots - Self Driving (Autonomous) Delivery Robots [Video]. YouTube. https://www.youtube.com/watch?v=dagjQW_jgtE

[30] Trevorrow, A., & Rokicki, T. (n.d.). Golly Game of Life Home Page. SourceForge. https://golly.sourceforge.net/

[31] Tsuda, S., Artmann, S., & Zauner, K. P. (2009). The phi-bot: a robot controlled by a slime mould. In Artificial Life models in hardware (pp. 213-232). Springer, London.

[32] V. H. García Ortega. Class Lecture, Topic: "Embedded Systems: Tecnologías y Aplicaciones." ESCOM, IPN, 2021.

[33] V. H. García Ortega. Class Lecture, Topic: "Sistema de arranque." ESCOM, IPN, 2021.

[34] Waggoner, B. M., & Poinar, G. O. (1992). A Fossil Myxomycete Plasmodium from Eocene-Oligocene Amber of the Dominican Republic. The Journal of Protozoology, 39(5), págs. 639–642. https://doi.org/10.1111/j.1550-7408.1992.tb04864.x

[35] Weisstein, E. (n.d.). Cellular Automaton – from Wolfram MathWorld. https://mathworld.wolfram.com/CellularAutomaton.html

[36] Wuensche, A. (2011). Exploring Discrete Dynamics (Revised). Luniver Press. págs. 361–363.

[37] Yong, E. (2010, enero 21). Slime mould attacks simulates Tokyo rail network. Science. https://www.nationalgeographic.com/science/article/slime-mould-attacks-simulates-tokyo-rail-network

[38] Zumba Gamboa, J. P., & León Arreaga, C. A. (2018). Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software. INNOVA Research Journal, 3(10), págs. 20–33. https://doi.org/10.33890/innova.v3.n10.2018