Circuito Genético

Simulación gráfica de un algoritmo genético y código evolutivo

Proyecto de Fin de Grado
Grado Superior en Desarrollo de Aplicaciones Multiplataforma
Alberto Pérez Arribas
08/06/2023

Tabla de contenido

1 - Introducción

1.1 - Explicación del proyecto

Sin entrar en detalles del funcionamiento de los algoritmos utilizados todavía, el programa que se va a realizar en este proyecto consiste en una simulación gráfica de un entorno en el que se generan una serie de entidades que se mueven de forma acorde a unas físicas básicas. Estas están rodeadas de una serie de obstáculos que deberán evitar, y con una meta en el espacio gráfico que tendrán que alcanzar por cuenta propia, sin ningún tipo de control por parte del usuario, únicamente a través de una serie de atributos o cualidades que irán evolucionando a lo largo de la ejecución del programa.

De acuerdo a esa descripción del problema, es de suponer que la forma de lograr ese objetivo es desarrollando algún tipo de inteligencia artificial primitiva que sepa ajustar esos parámetros de forma autónoma, aunque en nuestro caso, dicha inteligencia es el producto de haber utilizado lo que se denomina *algoritmo genético*. Dicho algoritmo se basa en la generación aleatoria de valores que cíclicamente se irán evaluando y seleccionando para que progresivamente las entidades realicen su función de manera acorde a lo que exigen los requisitos.

El hecho de que las entidades sean capaces de llegar por sí mismas al objetivo, es por sí sólo algo a tener en cuenta, pero lo que conseguimos con este programa es poder obtener una solución real. Esta consiste en una ruta óptima que tendrían que seguir para llegar lo más rápido posible, de forma que podamos demostrar que el algoritmo en el que se centra este proyecto es aplicable para cualquier caso en el que necesitemos realizar una búsqueda de caminos óptimos, que si bien en este caso es algo literal, puede referirse a conceptos más abstractos y no necesariamente visuales.

Para que el programa no se limite a una visualización de los hechos y podamos interactuar y experimentar con él, se incluirá una pequeña interfaz al margen de la ventana gráfica para poder manipular el proceso evolutivo, de acuerdo a unos parámetros que definiremos más adelante.

1.2 - ¿Por qué este proyecto?

En lo que concierne al desarrollo de software, disponemos de amplios estudios sobre patrones de diseño y algoritmos que nos facilitan la resolución de problemas de forma óptima. Existe un campo en particular llamado *Programación*

genética y evolutiva que implica el uso de unos algoritmos que simulan los principios de la genética y la selección natural Darwiniana del área de la biología, trasladados al de la informática para conseguir que un programa sea capaz de pasar por un proceso evolutivo de forma autónoma y posiblemente con el "feedback" del usuario.

Los algoritmos genéticos son de gran utilidad cuando tratamos problemas que normalmente requerirían un tiempo extremadamente amplio como para resolverlo de una forma tradicional, incluso con la capacidad de cálculo de la que dispone un ordenador moderno. Es por ello que donde más brilla es a la hora de optimizar unos resultados sobre una necesidad específica, ya que son capaces de determinar las mejores soluciones para un tiempo relativamente corto, bajo los criterios que creamos conveniente (rapidez, precisión, o cualquier adecuación al problema).

Pese a ello, se trata de un área poco explorada, especialmente a nivel académico. A día de hoy existen campos de la inteligencia artificial mucho más estudiados, como el aprendizaje automático ("machine learning").

He considerado interesante tratar esta temática, puesto que aporta un enfoque distinto a la hora de abordar diversos problemas en los que contemos con muchas posibilidades y no necesariamente queramos que la solución dependa de analizar datos de forma masiva durante un tiempo indeterminado, sino que el propio programa se encargue de llegar a la solución por sí mismo, siguiendo un proceso que nos es familiar científicamente hablando.

Existe una limitación importante respecto a esta técnica de programación que implica que pese a la efectividad a la hora de encontrar soluciones óptimas, no es la mejor manera de obtener un resultado exacto. Pese a ello, muchos escenarios requieren respuestas aproximadas que sean perfectamente utilizables y cuyo margen de error sea tan mínimo que no influya en la herramienta para la que se esté desarrollando el programa, como puede ser un GPS, que necesita calcular la ruta más óptima lo más rápido posible, siendo esta casi con total certeza la que necesita el consumidor.

1.3 - Objetivos

El objetivo principal del proyecto es demostrar la funcionalidad y la eficacia que tienen este tipo de algoritmos, y como cualquier muestra es perfectamente trasladable a problemas de la vida real. La naturaleza de la programación evolutiva implica que se aplican las mismas mecánicas para cualquier situación, y que lo

único que habría que hacer sería adaptarlo a los requisitos del programa y encajar toda la lógica de este bajo un mismo proceso.

En cuanto a los requisitos que debe cumplir el programa, lo mínimo es que podamos visualizar de forma gráfica el algoritmo en funcionamiento con una simulación casi práctica sobre la optimización y búsqueda de caminos, en forma de unas entidades que deberán desarrollar generación tras generación una mejor aptitud para llevar a cabo el problema y ser capaces de moverse en el espacio en dirección a una meta de la mejor manera posible, cada vez más acertada y rápida a medida que evolucionan. Esto será posible en un tiempo factible gracias al paradigma de la programación genética.

1.4 - Destinatarios

Este proyecto va dirigido a aquellos con interés en expandir su conocimiento sobre una de las ramas de la inteligencia artificial más allá de lo que se suele ver en artículos de primera plana del sector tecnológico, en el cual predominan los modelos basados en el análisis de datos frente a la autonomía de los programas, al margen de disponer o no de una muestra sobre la que entrenar sus conocimientos.

Muchos de los usos más comunes de esta rama giran en torno a la optimización y generación de rutas. En términos de ingeniería se puede aplicar al diseño de componentes industriales, de vehículos o en robótica. Tiene mucho potencial de cara a sectores como el de la bioinformática, el sector sanitario, y cualquier tipo de investigación de la que no se disponga de datos con los que utilizar un modelo orientado al aprendizaje automático, y en la que el proceso deba de ser estudiado y no solo llegar a la solución. Otros casos de uso pueden ser la generación de imágenes digitales, análisis financiero, aplicaciones de planificación, automatización de procesos y simulaciones en videojuegos.

Si eso no fuera motivo suficiente, existen casos en los que los algoritmos genéticos se utilizan en conjunto con otras técnicas de inteligencia artificial, no son mutuamente excluyentes. Existen muchas posibilidades por explorar y cabe la posibilidad de que surjan nuevas aplicaciones en un futuro que exploten las técnicas de la programación genética.

2 - Metodología

2.1 - Herramientas y lenguajes

- **Java:** La elección de este lenguaje de programación se basa en la familiaridad que me supone a la hora de desarrollar y en la comodidad que proporciona para un planteamiento orientado a objetos y con un "tipado" fuerte de datos.
- **Eclipse:** El IDE elegido por facilidad de uso, ya que tiene herramientas de "debug" intuitivas y una estructura de proyectos bien organizada.
- **Processing:** Entorno de desarrollo y lenguaje de programación basado en *Java*, dedicado a facilitar el desarrollo de proyectos multimedia interactivos. Sin embargo, en mi caso utilizo únicamente su librería para *Java* correspondiente para poder disponer de su estructura de clases específica para acelerar la implementación del entorno gráfico, las entidades como elementos visuales y el desarrollo de las físicas, sin prescindir de programar en *Java* nativo con todas las cualidades que tiene, ya que *Processing* por sí solo no es una herramienta para hacer programas complejos.
- **WindowBuilder:** "Plugin" para *Eclipse* que permite desarrollar interfaces gráficas de manera visual y con soluciones sin código, para implementar la UI con la que el usuario interacciona con el programa para modificar el proceso evolutivo.
- **Figma:** Aplicación web utilizada para el diseño del prototipo de la interfaz gráfica del programa, como puede ser la ventana gráfica y su panel de control.
- **Papyrus:** "Plugin" para *Eclipse* que permite crear multitud de diagramas UML.
- **GitHub:** Aplicación web que facilita el control de versiones para el proyecto y permite la creación de ramas de desarrollo para añadir cambios de forma segura. Además almacena en la nube el proyecto para poder acceder a él con facilidad desde cualquier parte. Eclipse tiene un "plugin" por defecto llamado *EGit* que permite conectarse al repositorio remoto de forma local para poder subir código y todo tipo de operaciones *Git* necesarias desde el propio IDE.
- **ObjectAid:** "Plugin" de *Eclipse* que permite obtener diagramas de clase a partir de realizar ingeniería inversa sobre las clases presentes en un proyecto.

2.2 - Cómo se realizará el programa

Lo primero que habrá que desarrollar es la ventana gráfica con todos los obstáculos y la meta dibujados en un espacio en dos dimensiones, así como un sistema de colisiones para se pueda determinar si las entidades chocan con ellas y elaborar una clase que haga de colección de las entidades que tendrán una forma identificativa (una figura humanoide). Todas las entidades aparecerán siempre en

un mismo punto de la ventana al mismo tiempo cada vez que realicen su ciclo de ejecución.

Habrá que programar una clase Entidad de forma que realice un ciclo de vida que consista en moverse dentro del circuito de acuerdo a unos parámetros únicos para cada objeto hasta que colisionen con algún elemento, sea la meta o un obstáculo, momento en el que no necesitan moverse más. De llegar a la meta, habrá obtenido un tiempo que podrá comparar con el objetivo, y en caso de chocar con algún obstáculo, no seguirán mostrándose y se quedarán sin un tiempo de llegada. Es por esto que las entidades deberán evitar los obstáculos (de hacerlo significa que han fracasado a la hora de intentar cumplir lo que se les pide).

A su vez deberá existir una clase Población que agrupe a todas las entidades del sistema para que pueda acceder y manipular cada una de ellas en el procedimiento que vamos a realizar. Dicha clase se encargará de darles también unas características comunes y llamar a que se ejecuten por separado.

Lo siguiente que habría que hacer una vez tenemos todos los elementos visuales y nuestro modelo de objetos, sería implementar unas físicas de movimiento para las entidades, tal que por cada “frame” (fotograma que se dibujará en la pantalla con una frecuencia muy alta para dar la sensación de movimiento) de la ejecución de la ventana, se desplacen en el espacio de en una dirección y a una velocidad variable de acuerdo a unos parámetros que definen qué fuerzas se le aplicarán para modificar su aceleración (la magnitud, dirección y sentido de ésta define hacia qué punto se desplaza en ese instante y cómo de rápido lo hará).

Los parámetros que definen el comportamiento de la entidad se generarán de manera semialeatoria. La primera vez que arranque el proceso y se genere una primera población de entidades, esas fuerzas (que son realmente vectores) serán totalmente aleatorias, pero lo que queremos lograr es que evolucionen hasta adaptarse al escenario en el que están, para que en todo momento sepan cómo llegar hasta la meta lo más rápido posible evitando los obstáculos.

¿Cómo es esto posible? Aquí es donde entra en juego el algoritmo genético que vamos a implementar en el programa. Suponiendo que las entidades tienen un tiempo de vida determinado para poder llegar a la meta, se les deberá aplicar una cantidad determinada de vectores acorde a los frames que tiene para realizar su tarea, para ir alterando su movimiento en ese intervalo de tiempo hasta que por una causa u otra todas las entidades terminen su ciclo de ejecución.

Dichos vectores son los que irán evolucionando a lo largo del programa para conseguir lo que nos proponemos. A partir de ahora los llamaremos “genes”, ya que son una representación del “genotipo” del objeto, lo que se traduce en términos

informáticos como los datos, las propiedades del objeto que definen lo que puede hacer (qué vectores se le aplicarán en su ciclo de vida). Por otro lado, tenemos su "fenotipo", que es el cómo expresará esos datos en forma de funciones (cómo va a utilizar esos vectores para moverse a donde debería).

Sabiendo qué es lo que deben cumplir los genes (los encapsularemos en una clase que denominaremos el "ADN" del objeto), tendremos que evaluarlos de alguna forma para aplicar el principio de la selección natural, que implica que la probabilidad de reproducirse es mayor si tiene las aptitudes que buscamos para la resolución del problema. En esta fase es donde utilizaremos la "función de aptitud", que se encargará de calificar el objeto asignando un valor al atributo "Aptitud".

La aptitud será inversamente proporcional a lo cerca que se ha quedado de la meta en algún momento de su ciclo y cuánto ha tardado en llegar (o cuánto tiempo se ha mantenido en ejecución si no ha llegado), y directamente proporcional a lo cerca que está el tiempo obtenido sobre el tiempo que hemos indicado como el objetivo a cumplir. Dicho valor se verá recompensado si ha llegado a la meta. La parte de la selección del algoritmo genético es la que más varía de acuerdo al problema, ya que es la que define qué deben cumplir los objetos implicados y por tanto deberemos programar a nuestra necesidad.

Una vez conocemos la aptitud de cada objeto o entidad, podremos pasar a determinar cuáles deberán "reproducirse" o no utilizando algún método probabilístico. Los objetos tendrán una probabilidad directamente proporcional a su aptitud de ser escogidos en una colección que llamamos "pool genético", para que si el objeto tiene cualidades que buscamos tenga más opciones de cruzarse con otro y pasar sus genes a la siguiente generación.

Para poder crear una nueva población de entidades, habrá que generar unos objetos "hijos" de la anterior generación, cuyos genes sean el producto de haber mezclado los de dos objetos "parientes" aleatorios, utilizando el algoritmo que creamos conveniente para determinar qué cantidad y/o partes de los genes de cada uno de los dos parientes se deberán seleccionar. La forma de hacerlo será obteniendo dos parientes al azar del "pool genético", teniendo estos más probabilidades de ser escogidos según su aptitud, y manipular los genes de ambos para crear un "ADN" nuevo para el nuevo objeto.

Además de obtener los genes a partir de los anteriores, existe una probabilidad pequeña (que siempre podremos reajustar) de que esos genes muten, y por tanto, se introduzca alguna variación aleatoria entre los genes heredados. Este paso es necesario para que, independientemente de qué atributos tiene la población inicial, siempre exista la posibilidad de que las generaciones posteriores

tengan genes nuevos que no pudieran ser obtenidos mediante los posibles cruces hereditarios, ya que lo contrario limitaría el proceso evolutivo y no garantizaría lograr el objetivo, y en ese caso, la solución pasaría a manos de la suerte de los datos generados al inicio. Una tasa de mutación demasiado alta, por otro lado, provocaría tanta aleatoriedad que el proceso de selección de los genes aptos quedaría anulado, lo cual queremos evitar.

Una vez tengamos nuestra nueva generación de entidades, esta reemplazará a la anterior y repetirán el mismo ciclo de vida utilizando sus genes para expresarlos en forma de movimiento (el fenotipo). El proceso evolutivo se volverá a llevar a cabo repitiendo todos los pasos anteriores y se repetirá sucesivamente en un número indeterminado de generaciones. De esa manera nos iremos quedando con los genes que nos interesan y cada iteración será mejor que la anterior.

Según dejemos el programa en ejecución y vayamos viendo los resultados de la evolución, podremos observar que cada vez las entidades se quedan más cerca de cumplir la solución que buscamos. Podemos incluir alguna condición para que el proceso pare en el momento que la encuentre y muestre el resultado final. Esta solución será la ruta más óptima para llegar lo antes posible (el tiempo que consideremos aceptable) hacia a la meta desde el punto inicial de acuerdo a las físicas que hemos considerado. Es conveniente mostrar la solución de forma visual.

Al tratarse de un proceso cíclico, se vería un número altamente variable de generaciones hasta que demos con los requisitos, puesto que hasta cierto punto sigue en manos del azar, pero sigue siendo mucho más eficaz y rápido que utilizar cualquier otro método convencional. Además, tiene la ventaja de que podemos utilizar el mismo mecanismo para diferentes escenarios sin que tengamos que cambiar nada de la programación de las entidades. Podríamos cambiar de sitio la meta y los obstáculos y el flujo seguiría funcionando igual. Es por eso que el programa incluirá también la opción de elegir varios circuitos creados previamente desde código y serializados como objetos en forma de ficheros que se pueden cargar.

Si intentáramos lo mismo usando la fuerza bruta a base de obtener valores completamente aleatorios, podríamos tardar una cantidad de tiempo tan alta que no sería viable, puesto que tendríamos miles de variables que calcular en todo momento.

Un factor a tener en cuenta es que el número de elementos que tiene una población será muy relevante para el tiempo que llevará llegar a la solución, puesto que cuantas más entidades existan, más cruces se producirán y más probabilidad

de obtener los genes deseados habrá. Sin embargo, una cantidad demasiado elevada puede ser contraproducente, ya que el rendimiento del ordenador se puede ver afectado al no poder procesar bien tantas entidades al mismo tiempo y verse ralentizado.

Para tener un mayor control sobre el proceso evolutivo, se incluirá un panel de control como interfaz gráfica que permita manejar los parámetros como la tasa de mutación, el número de entidades, el tiempo objetivo y el tiempo de vida, así como poder monitorizar en cierta medida la evolución que se está produciendo, los mejores resultados obtenidos, y todo tipo de información sobre las generaciones que se producen y las entidades que deseemos elegir observar individualmente.

Por último, esta interfaz deberá dar al usuario control directo sobre el flujo de ejecución del programa, para que pueda comenzar y reiniciar un proceso evolutivo desde cero (creando una población completamente nueva), dar la orden de pasar a la siguiente generación según termina de actuar y se reproduce cada una (o de activar un modo automático que lo haga por nosotros inmediatamente), o de pausar la ventana y el proceso hasta que decidamos reanudarlo. Podremos también seleccionar un circuito si no hemos iniciado todavía una población.

2.3 - Tareas a emprender

- Estudiar y entender en profundidad los conceptos teóricos y prácticos sobre los algoritmos genéticos.
- Desarrollar un entorno gráfico donde se visualicen las formas y entidades.
- Implementar unas físicas de movimiento para las entidades basadas en los principios básicos de la mecánica clásica.
- Poner en práctica conocimientos de probabilidad y estadística para aleatorizar el comportamiento de las entidades.
- Desarrollar una lógica fundamental para el flujo del programa, las interacciones entre las entidades y las colecciones de datos que mantiene el sistema.
- Aplicar los fundamentos de los algoritmos genéticos a la base del programa para proporcionar una inteligencia artificial a las entidades.
- Adaptar los algoritmos a los requisitos, de forma que la evolución del comportamiento de las entidades tenga un proceso y objetivo concreto.
- Elegir correctamente los algoritmos y métodos estadísticos que se utilizarán para la selección de atributos (genes) y reproducción de las entidades.
- Hacer visible en la interfaz los resultados de los objetivos cumplidos para demostrar la utilidad y la eficacia de la metodología.

- Proporcionar una interfaz de usuario que permita manipular la ejecución del programa y el proceso evolutivo mediante diferentes opciones que modifiquen los atributos necesarios.
- Implementar un mecanismo de creación, almacenamiento y selección de circuitos utilizando “serialización” para poder cargarlos desde ficheros.
- Refinar y ampliar todo lo anterior en la medida de lo posible.

2.4 - Cronograma

1) Estudio y familiarización con la librería de *Processing* para el desarrollo gráfico, así como los conceptos de la programación de gráficos (2 semanas)
2) Estudio de las físicas a implementar y los conceptos matemáticos de probabilidad y estadística necesarios para el algoritmo genético (1 semana)
3) Estudio y análisis de los algoritmos genéticos en la teoría y práctica (2 semanas)
4) Conceptualización y diseño del programa a realizar (1 semana)
5) Desarrollo del programa aplicando los conocimientos necesarios (2 semanas)
6) Ampliación y pulido del programa (1 semana)

3 - Arquitectura del programa

3.1 - Diseño de la interfaz gráfica

3.1.1 - Ventana gráfica

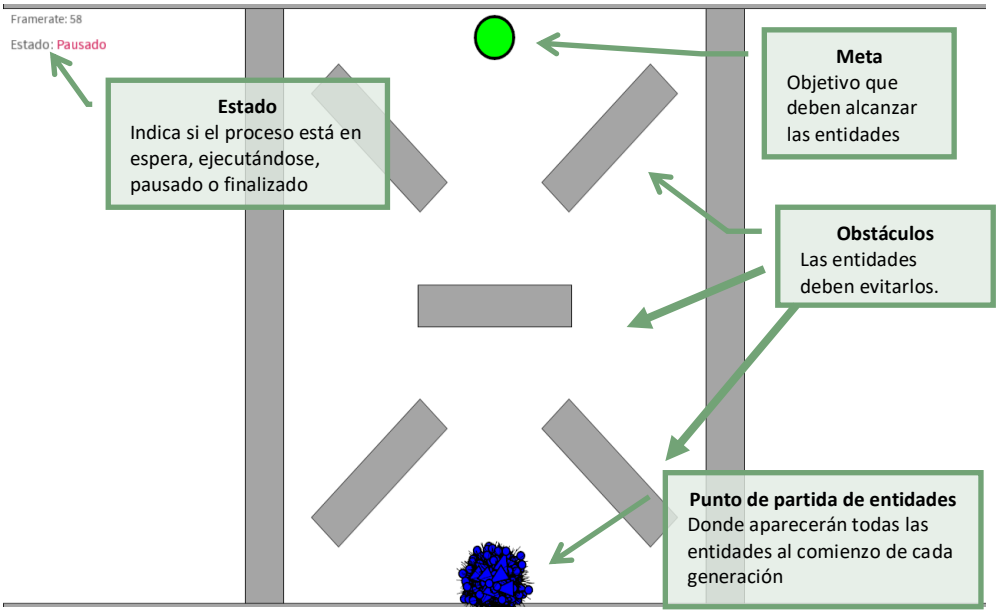
La interfaz principal consistirá en una ventana generada utilizando la librería de *Processing*. Dicha ventana “dibujará” cada frame una serie de elementos gráficos en la pantalla, representados por formas “renderizadas” (procesadas para visualizarse) en tiempo real, teniendo en cuenta que actualizará 60 frames por segundo.

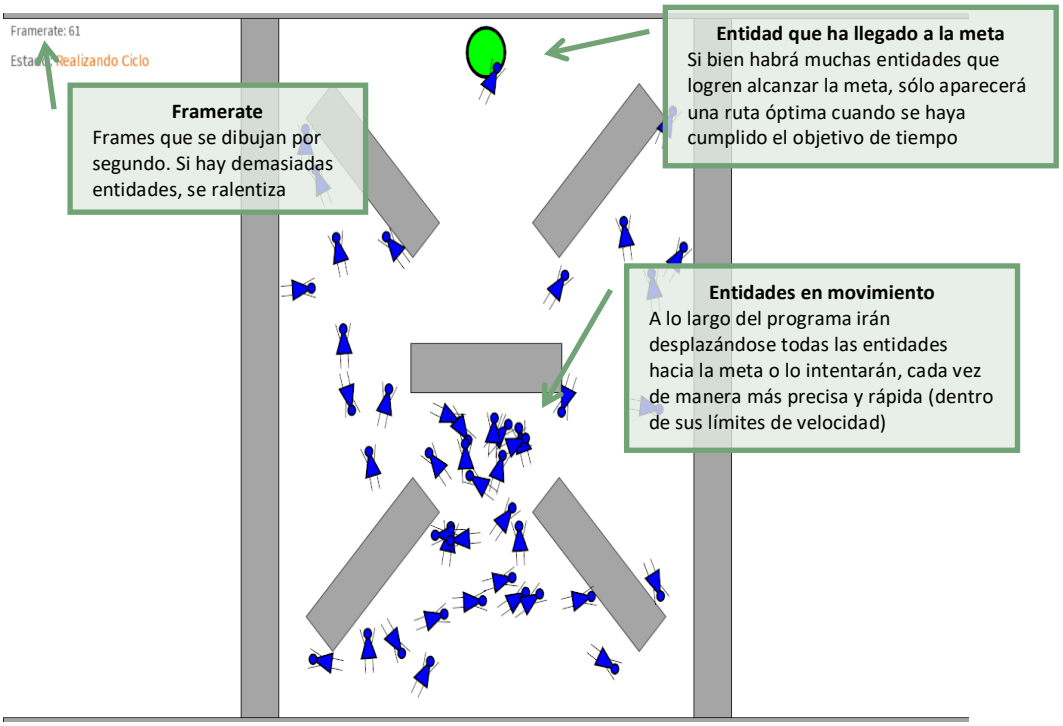
Dicha ventana contiene un conjunto de entidades en forma de figuras similares a personas que son las que deberán desplazarse hacia la meta, de peor a mejor manera según evolucionen. Al principio saldrán todos desde el mismo punto de partida y, según los genes de cada uno, se desplazarán de una manera distinta individualmente.

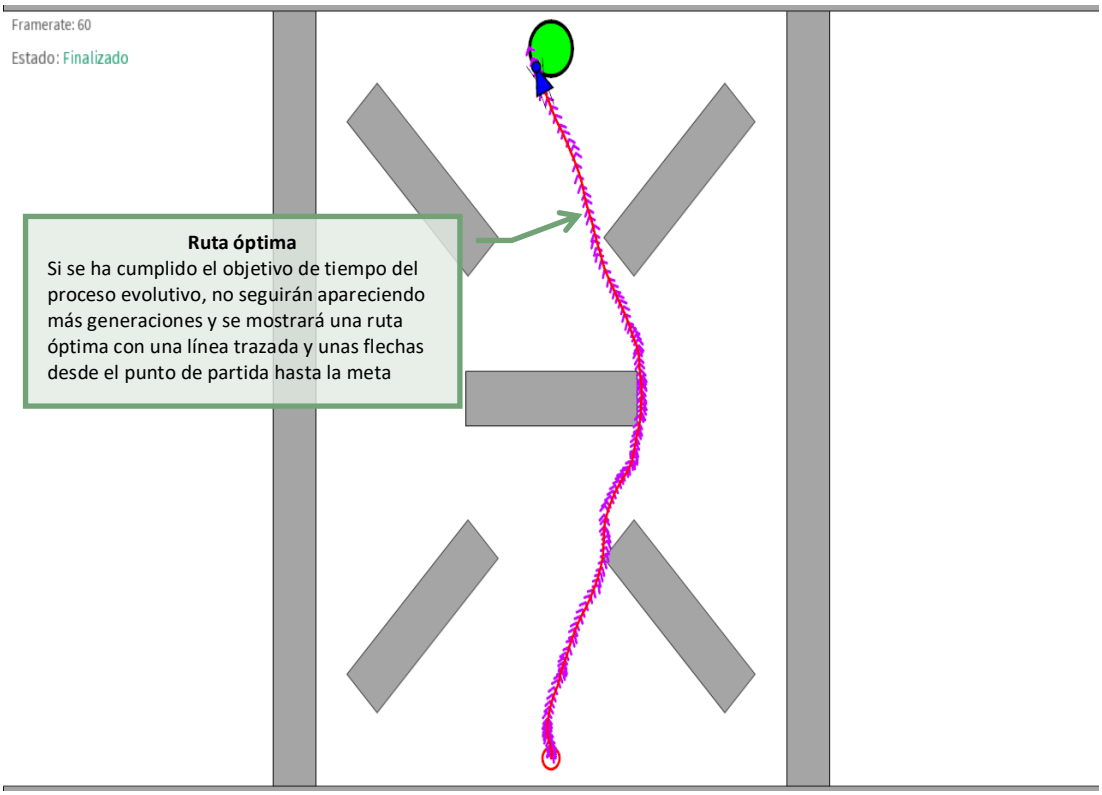
En su camino habrá una serie de obstáculos que aparecen como rectángulos en diferentes ángulos de rotación y que dificultarán su movimiento hacia la meta. Tienen una posición prefijada como si se tratara de un circuito.

Finalmente, aparece una elipse en un punto fijo que actuará como la meta que deberán alcanzar las entidades al desplazarse. Cuando se alcance una solución óptima, aparecerá la ruta que ha tomado la entidad más rápida desde el punto de partida hacia la meta. Esta ruta aparecerá marcada en rojo y con flechas mostrando cada movimiento.

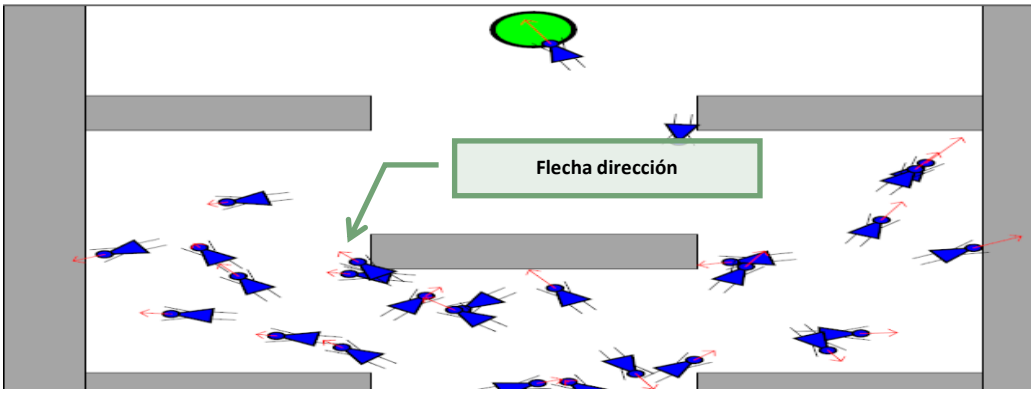
También se incluyen dos textos que informan del número de fotogramas por segundo y del estado del flujo de ejecución de la población de entidades.







Si pulsamos la tecla “d” en cualquier momento, se mostrarán unas flechas de dirección sobre las entidades para visualizar hacia donde se mueven y qué velocidad, como una especie de “modo debug”. Hay que tener en cuenta que esto ralentizará severamente el programa, por lo que no es aconsejable tenerlo siempre activado (podemos desactivarlo con la misma tecla).



3.1.2 - Panel de Control

Además de la ventana gráfica, dispondremos de otra interfaz desarrollada en *Java Swing* con apoyo del “plugin” *WindowBuilder* de *Eclipse.* Dicha interfaz actuará como un panel de control que nos permitirá manipular y monitorizar el proceso evolutivo.

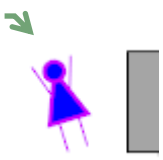
En la primera sección podremos visualizar en qué generación se encuentra actualmente la simulación. Al principio del programa, aparecerá un botón de “Empezar” habilitado para iniciar el proceso. Tras comenzarlo, ese botón cambiará a “Siguiente” y sólo podremos pulsarlo cuando acabe el ciclo de vida de todas las entidades de la generación actual, para generar una nueva generación a partir de la existente. El botón de “Reiniciar” solo aparecerá una vez comenzado el proceso para reiniciar todos los elementos del sistema a su estado inicial. El botón de pausa permite paralizar la ejecución de las entidades sin dejar de mostrarlas en pantalla, que después podremos pulsar de nuevo para reanudarlas.

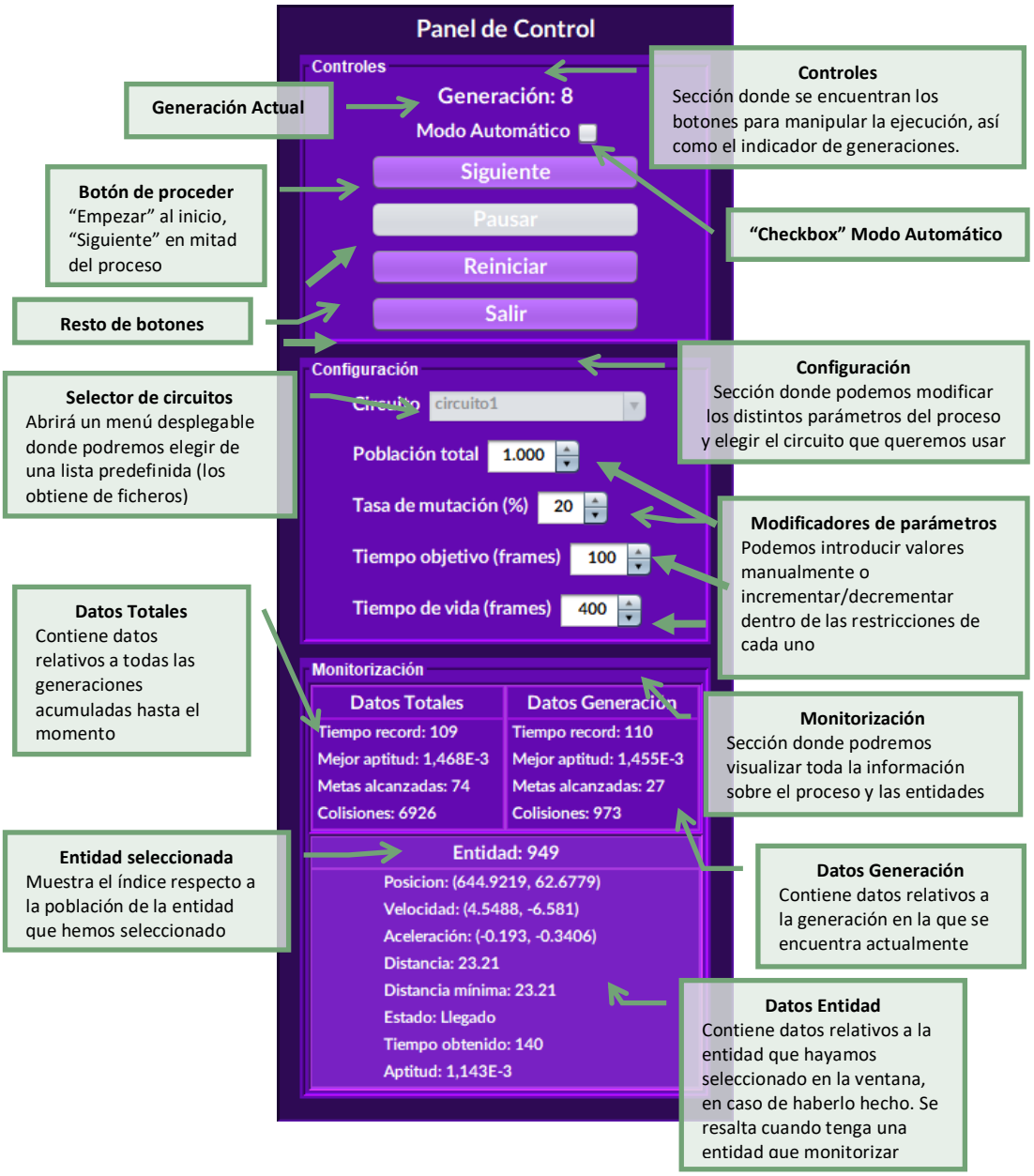
El modo automático se puede activar para que las generaciones se sustituyan sin necesidad de indicárselo explícitamente y que el proceso evolutivo continúe sólo, sin necesidad de nuestro “input”. Todos estos controles pueden realizarse en la ventana gráfica con atajos de teclado (“espacio” para empezar, proceder, pausar y reanudar, “a” para activar/desactivar el modo automático y “r” para reiniciar).

La siguiente sección nos permite manipular algunos parámetros del proceso, como la población total de entidades que aparecerá cada generación, la tasa de mutación (porcentaje), el tiempo objetivo (en frames) que deberán alcanzar, y el tiempo de vida (frames) del que dispondrán las entidades antes de reproducirse. Podrán alterarse incluso en mitad del proceso. Además, dispone de un selector de circuitos, que nos da la posibilidad de cambiar a otro circuito sólo antes de que comience el proceso.

Por último, tendremos una sección donde consultar información sobre el proceso evolutivo, tanto a lo largo de todas las generaciones, como de la actual (el mejor tiempo obtenido, la mejor aptitud, cuántas entidades han alcanzado la meta, cuántas han chocado con algún obstáculo). Si pinchamos sobre una de las entidades, podremos monitorizar información específica sobre ésta en tiempo real, como sus vectores posición y movimiento, la distancia actual y mínima a la meta, el tiempo obtenido, aptitud y su estado. La información sobre la generación actual y la de la entidad se limpian cada vez que se cambia de generación.

Entidad monitorizada
Si pinchamos en una de las entidades de la ventana, se resaltará bordeada en morado y comenzará a monitorizarla en tiempo real en el panel de control en su sección correspondiente, hasta que seleccionemos otra o deje de existir





3.2 - Estructura de clases

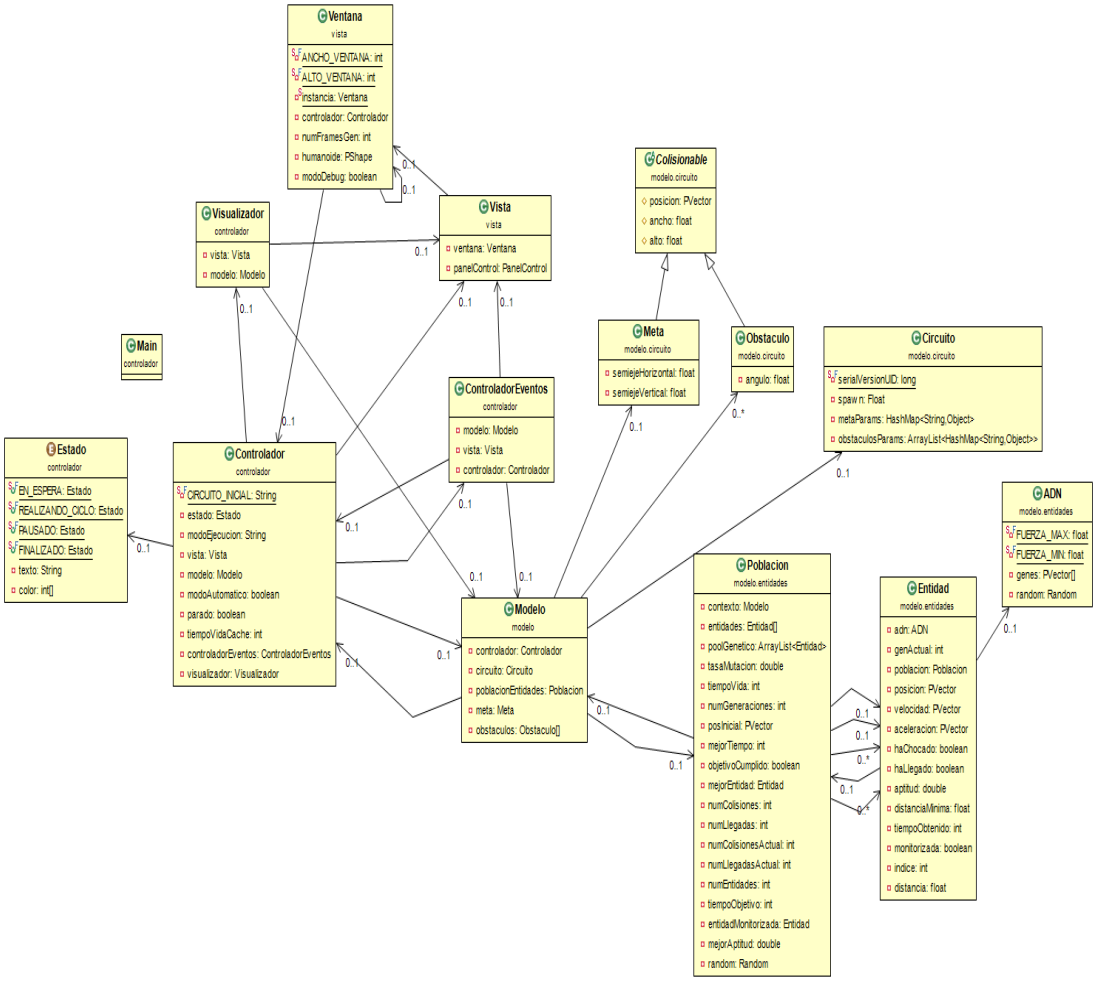
El programa está implementado utilizando una arquitectura basada en el "Modelo Vista Controlador". Esta estructura ayudará a separar las clases de datos y la lógica interna del programa, de las interfaces gráficas y la interacción con el usuario. De esta manera, si la vista requiere consultar datos lo hará a través del controlador y si el modelo necesita actualizar la interfaz deberá hacer una llamada al controlador. Se podría decir que el controlador hace de nexo entre ambas partes y mantiene el código mucho más organizado.

El "Controlador" principal se encarga de llevar a cabo la lógica principal del proceso evolutivo bajo demanda de las actualizaciones de la "Ventana", pero tiene otros sub-controladores a los que delega tareas. El "Visualizador" permite al

"Modelo" modificar la interfaz y el "ControladorEventos" a la "Vista" manipular los datos.

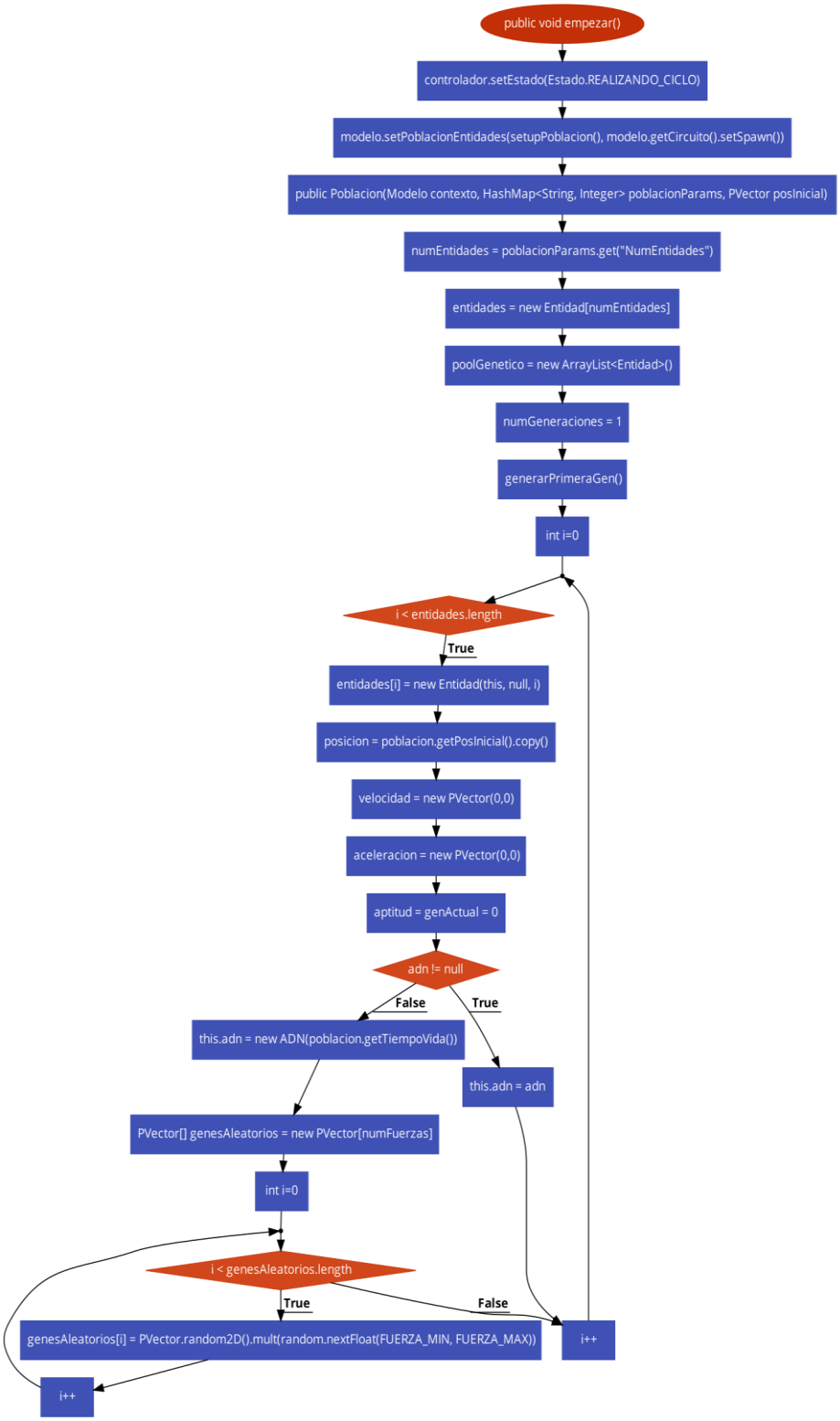
La "Vista" contiene el "PanelControl" y la "Ventana", cuyo funcionamiento ya hemos explicado anteriormente. El "Modelo" contiene el "Circuito", que permite la creación de la "Meta" y la colección de "Obstáculos" a partir de sus datos (estas dos clases heredan de la clase abstracta "Colisionable", ya que comparten características y deben implementar colisiones). A su vez, contiene la "Población", que tiene una colección de "Entidades" cuyo comportamiento viene definido en su "ADN".

El "Controlador" llama a la "Población" a lo largo del proceso para que ejecute todas sus entidades y realice las distintas fases del algoritmo genético, mientras que cada "Entidad" individual tiene su lógica interna para poder actuar de acuerdo a su "ADN", e interactuando con "Meta" y "Obstáculo", que después se reflejará en la "Vista".

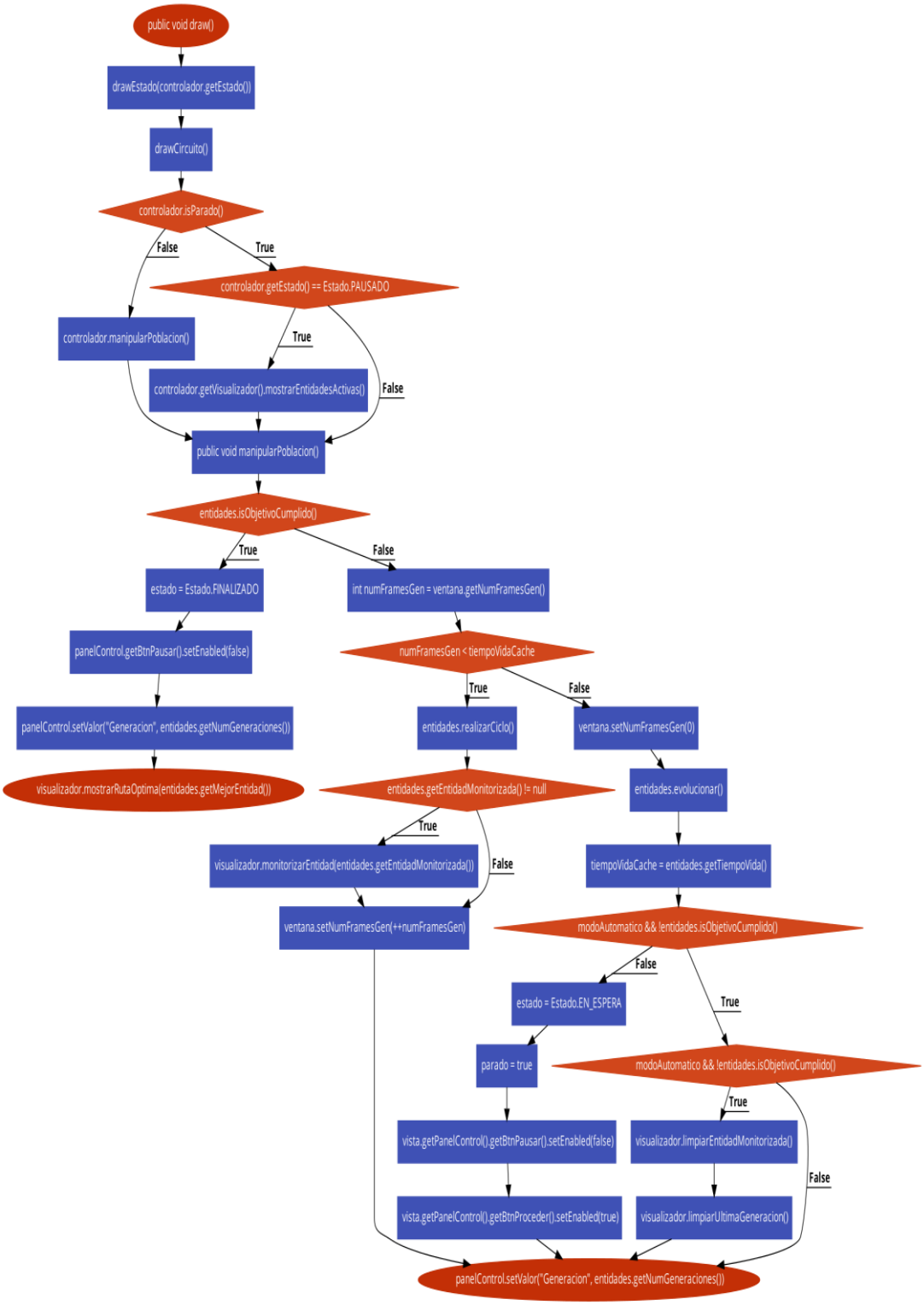


3.3 - Diagramas de flujo

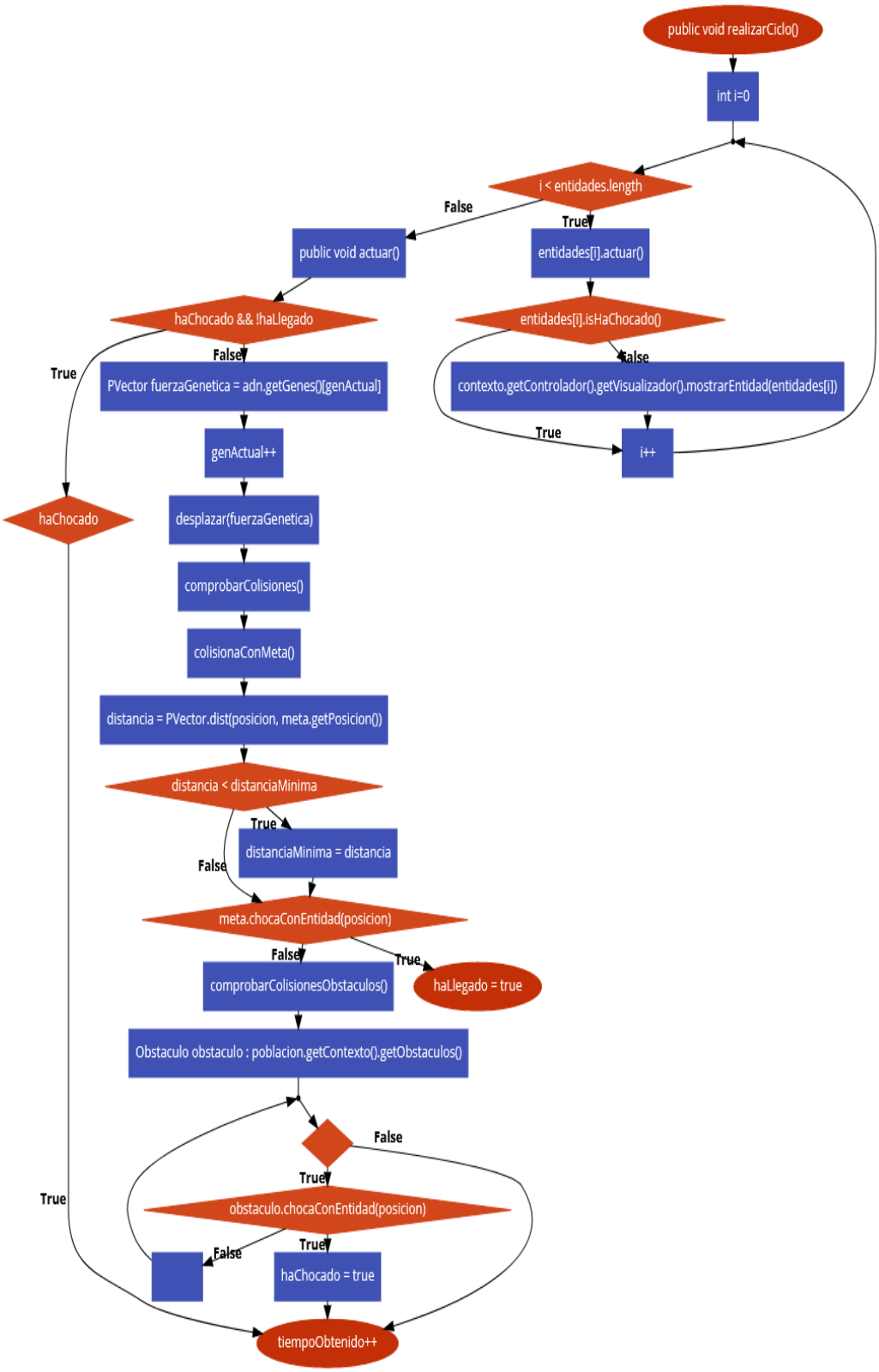
Flujo de iniciación de la población, las entidades y sus genes:



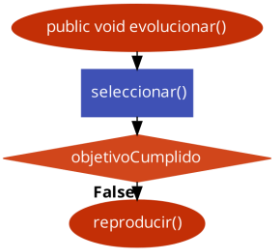
Flujo general del proceso evolutivo:



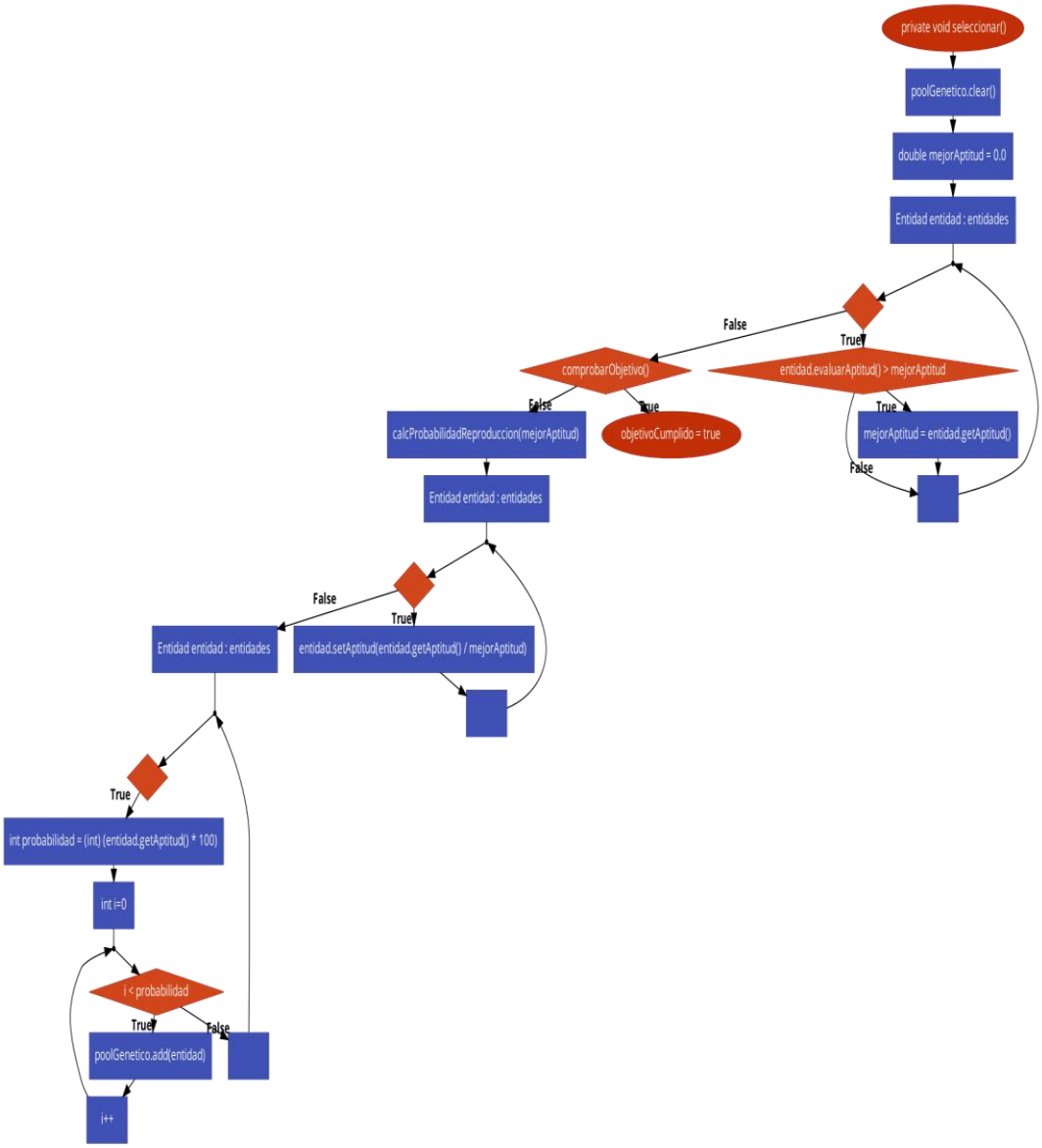
Flujo de actuación de las entidades:



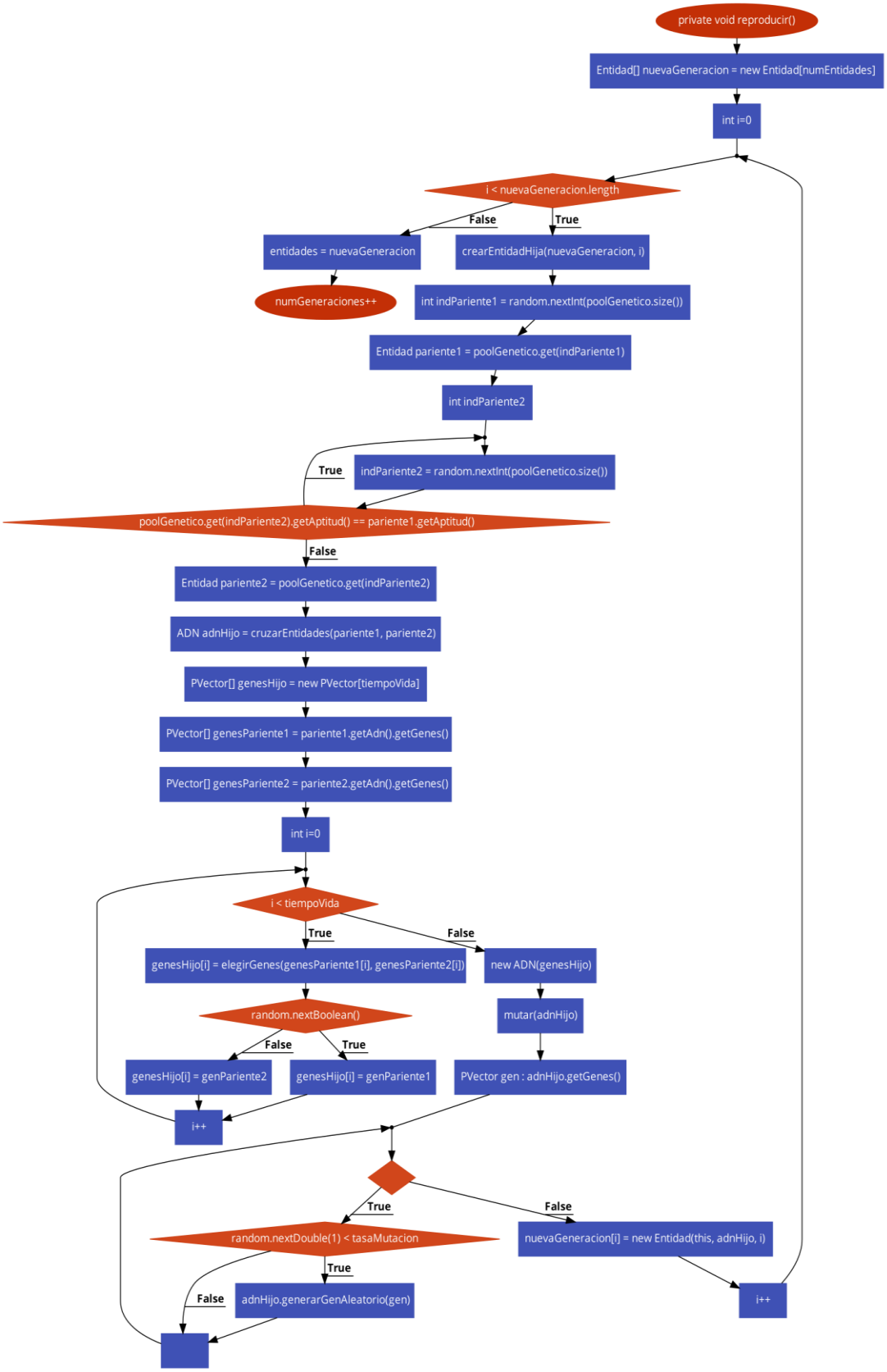
Flujo simplificado de evolución de la población:



Flujo de evaluación y selección de las entidades:



Flujo de reproducción de las entidades:



3.4 - Diagrama de estados

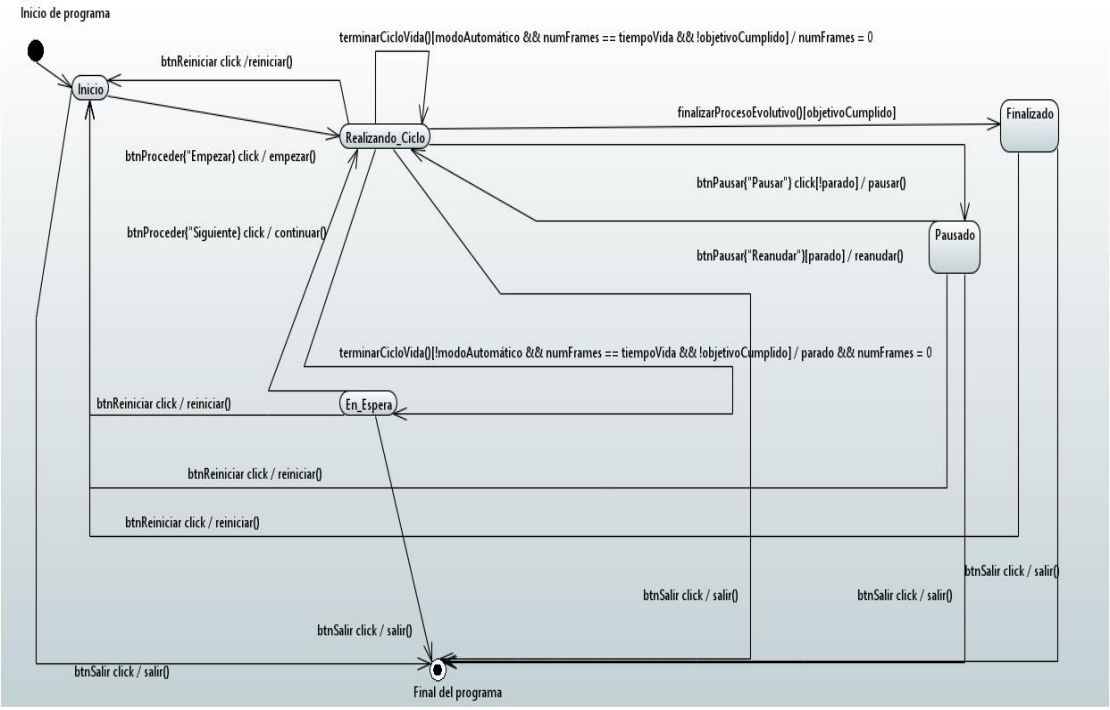
El programa pasa por diferentes estados que se desencadenan según los eventos, ya sean por parte de la interacción del usuario con los controles de la interfaz o por el flujo natural del programa. Los cinco estados principales en los que se puede encontrar son “Inicio”, “Realizando Ciclo”, “Pausado”, “En Espera” y “Finalizado”.

Al principio del programa siempre está en el “Inicio” y es el momento en el que el usuario todavía no ha comenzado el proceso. Aquí tiene la posibilidad de cambiar de circuito y de cambiar parámetros (aunque esto último lo puede hacer en cualquier momento del proceso) antes de iniciar una población.

Una vez comenzado, pasará a “Realizando Ciclo”, momento en el que las entidades empezarán a actuar hasta que se les acabe el tiempo de vida. A partir de aquí, si el modo automático está activado, pasará directamente a ejecutar la siguiente generación sin necesidad de interacción con el usuario y, por tanto, se mantiene en el mismo estado; pero si no tenemos el modo automático activado, pasa al estado "En Espera" y no pasará a ejecutar la siguiente generación hasta que el usuario active el evento de continuar y vuelva al mismo estado de antes. Toda la monitorización del proceso se llevará a cabo aquí también.

Siempre que las entidades estén actuando, es decir, el estado sea “Realizando Ciclo”, podemos desencadenar el estado “Pausado” si llamamos al evento de pausar, lo que detendrá la ejecución de las entidades hasta que llamemos al evento de reanudar (que se hace desde el mismo control una vez esté ya parado), tras lo cual volverá a “Realizando Ciclo” y volverán a moverse.

Cuando alguna de las generaciones haya logrado el objetivo, se llamará un evento de finalizar y pasará de “Realizando Ciclo” a “Finalizado”, momento en el que dejará de ejecutar el proceso evolutivo y se mostrará la ruta óptima obtenida.
Podemos llamar al evento de reiniciar correspondiente en cualquiera de estos estados para volver al estado de “Inicio”, que provocará que desaparezca la población y regresemos a la misma situación que al empezar el programa. Si lo deseamos, podemos terminar el programa en cualquier momento sin importar en qué estado se encuentre.



4 – Epílogo

4.1 - Futuras mejoras

- Realizar el programa adaptado para 3 dimensiones utilizando algún motor gráfico con una cámara trasladable con el input del espectador.
- Trasladar el programa a una librería de *Java* que utilice *OpenGL* (se trata de una API multiplataforma para desarrollar gráficos en 2D y 3D) para poder desarrollar un programa con gráficos más optimizado a nivel de rendimiento.
- Implementar físicas y comportamientos de direccionamiento realistas para las entidades, que también evolucionarían.
- Desarrollar una red neuronal que se compenetre con el algoritmo genético.
- Mejorar los gráficos con texturas, efectos especiales y otras ideas estéticas.
- Almacenar "IAs" para poder cargarlas y que reproduzcan los comportamientos desarrollados con el proceso evolutivo para cumplir siempre el objetivo.
- Realizar la simulación de un circuito de carreras con coches evolutivos.

4.2 Conclusiones

Tras haber finalizado el proyecto, puedo afirmar con certeza que los objetivos propuestos para el programa han sido cumplidos. Además, he sido capaz de pulir la implementación para hacer la experiencia de usuario mucho más satisfactoria, con información intuitiva, y aumentando considerablemente la interactividad a través de la interfaz, manteniendo un nivel estético agradable en el proceso.

Debido a la naturaleza experimental del programa, he podido observar numerosas situaciones que me han llevado a plantearme muchas preguntas sobre el funcionamiento de los algoritmos genéticos. A la hora de buscar unos resultados óptimos, he tenido que modificar el planteamiento de los algoritmos utilizados en cada fase del proceso en varias ocasiones y de realizar ajustes en la función de aptitud. De esta manera he sido capaz de encontrar el equilibrio ideal entre un sistema en evolución que aprenda el curso de acción que tiene que tomar relativamente rápido, pero que tenga suficiente variabilidad como para que no se estanque y continúe mejorando sus cualidades. Esto garantiza que, tarde o temprano, logre alcanzar el objetivo propuesto. Este logro ha sido posible gracias a un periodo de pruebas exhaustivo en el que analicé los resultados y estadísticas que demuestran la eficacia del planteamiento.

Durante el desarrollo, he introducido varias ideas nuevas que he podido implementar y que hacen el proyecto más interesante. Por ejemplo, he añadido un selector de circuitos que permite cambiarlos durante la ejecución para visualizar cómo pueden afrontar las entidades un escenario distinto. Otras mejoras destacables son los sistemas de colisiones con las entidades o la posibilidad de modificar parámetros del algoritmo genético en mitad del proceso y que la población se adapte automáticamente a los cambios.

Después de haber tenido mi primera experiencia práctica con este paradigma de programación, me ha quedado claro el potencial que puede tener. Contemplo la posibilidad de continuar expandiendo el proyecto o incluso de crear otros nuevos, aplicando algoritmos genéticos para buscar soluciones a problemas reales donde se les pueda sacar provecho.

Es importante destacar que la programación genética y evolutiva es un campo con muchas más capas de complejidad, especialmente en proyectos de mayor envergadura. Lo expuesto aquí solo representa una base para explorar más a fondo este tema.

Para obtener más información sobre el programa y sobre los detalles específicos de la implementación, recomiendo revisar el código comentado y su correspondiente *JavaDoc*, donde se podrá acceder a todos los atributos y métodos de las clases, descritos en mayor detalle.

5 – Webgrafía

- Shiffman, D. (s.f.). Learning Processing. Recuperado de http://learningprocessing.com/
- Processing Foundation. (s.f.). Processing Tutorials. Recuperado de https://processing.org/tutorials
- Processing Foundation. (s.f.). Processing Reference. Recuperado de https://processing.org/reference
- Shiffman, D. (s.f.). Calling Processing Functions from Non-Sketch Classes. Recuperado de https://happycoding.io/tutorials/java/processing-in-java#calling-processing-functions-from-non-sketch-classes
- Tutorials Point. (s.f.). Genetic Algorithms. Recuperado de https://www.tutorialspoint.com/genetic_algorithms/
- Jitkrittum, W. (2018, 27 de julio). An Illustrated Guide to Genetic Algorithm. Towards Data Science. Recuperado de https://towardsdatascience.com/an-illustrated-guide-to-genetic-algorithm-ec5615c9ebe
- Shiffman, D. (s.f.). The Nature of Code. Recuperado de https://natureofcode.com/book/
- FloatyMonkey [Canal de YouTube]. (s.f.). Recuperado de https://www.youtube.com/@FloatyMonkey/
- Quora. (s.f.). What are some real-world applications of genetic algorithms? Recuperado de https://www.quora.com/What-are-some-real-world-applications-of-genetic-algorithms
- Quora. (s.f.). Are genetic algorithms currently used much in the applications of AI? If yes, where? Recuperado de https://www.quora.com/Are-genetic-algorithms-currently-used-much-in-the-applications-of-AI-If-yes-where
- Wikipedia. (s.f.). Genetic algorithm. Recuperado de https://en.wikipedia.org/wiki/Genetic_algorithm
- Stack Overflow. (s.f.). Recuperado de https://stackoverflow.com/