|  |
| --- |
| **Algoritmos genéticos** |
| Representación gráfica |
|  |
| **Proyecto de Fin de Grado** |
| Grado Superior en Desarrollo de Aplicaciones Multiplataforma |
| **Alberto Pérez Arribas** |
| **08/06/2023** |
|  |

Tabla de contenido

**1 - Introducción**3

1.1 - Explicación del proyecto3

1.2 - ¿Por qué este proyecto?4

1.3 - Objetivos5

1.4 - Destinatarios5

**2 - Metodología**6

2.1 - Herramientas y lenguajes6

2.2 - ¿Cómo se realizará el programa?6

2.3 - Tareas a emprender9

2.4 - Cronograma10

**3 – Arquitectura del programa**11

3.1 - Diseño de la interfaz gráfica11

3.1.1 - Ventana gráfica11

3.1.2 - Panel de Control13

3.2 – Diagramas de flujo14

3.3 - Estructura de clases15

**4 – Estadísticas**18

**5 – Epílogo**19

5.1 - Futuras mejoras0

5.2 - Conclusión0

**6 – Bibliografía**20

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, rodeadas de una serie de obstáculos que deberán evitar, y con un objetivo o “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, pero 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.

Que observemos cómo las entidades son capaces de llegar por sí mismas al objetivo con el tiempo es por sí sólo algo a tener en cuenta, pero lo que conseguimos con este programa es poder obtener una solución real que 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 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 (la rapidez, la precisión, 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 en boca de todos, como el aprendizaje automático (machine learning), y 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, si no 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, pero 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 ésta 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 encajando toda la lógica de éste 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 a la generación de rutas. En términos de ingeniería, se puede aplicar al diseño de componentes industriales, de vehículos o en robótica. Tiene muchísimo 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.

Y 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 necesariamente 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.

1. **- 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 por la comodidad que supone para un planteamiento orientado a objetos y con un tipado fuerte de datos. Además, existen librerías para Java dedicadas particularmente a los algoritmos genéticos que pueden ser de ayuda en proyectos de mayor envergadura.
* **Eclipse:** El IDE elegido por facilidad de uso, por tener herramientas de debug intuitivas y una estructura de proyectos bien organizada.
* **Processing:** Se trata de un entorno de desarrollo y un 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 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 y concretamente hacer ingeniería inversa con la estructura de clases del proyecto para agilizar su creación.
* **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 del 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.
* **Draw.io:** Aplicación web utilizada para realizar el diseño lógico y estructural del programa, sus diagramas de flujo, de clases y de secuencia que lo definen.

**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 el espacio, y elaborar una clase que haga de colección de las entidades que tendrán una forma identificativa (un triángulo). Estas entidades aparecerán siempre en un punto de la ventana a la vez.

Lo siguiente que habría que hacer una vez tenemos todos los elementos visuales 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 del programa, se desplacen en el espacio de acuerdo a 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).

Dichos parámetros se generarán de manera semi-aleatoria. 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? Es aquí 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 mientras evitan colisionar con los obstáculos (lo que significa que han fracasado), se les deberá aplicar una cantidad determinada de vectores 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 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. Es en esta fase donde crearemos la “Función de aptitud”, que se encargará de calificar el objeto asignando un valor al atributo “Aptitud” inversamente proporcional a lo cerca que se ha quedado de la meta y cuánto ha tardado en llegar (o cuánto tiempo se ha mantenido en ejecución si no ha llegado). Dicho valor se verá recompensado si ha llegado a la meta, y penalizado si se ha chocado con algún obstáculo. 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 de ser escogidos un “pool genético” directamente proporcional a su aptitud, para que si el objeto tiene cualidades que buscamos, tenga más opciones de cruzarse con otro.

Para poder generar una nueva población de entidades, habrá que generar unos objetos “hijos” de los anteriores cuyos genes sean el producto de haber mezclado los de dos objetos “parientes”, 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” (la estructura de datos que los almacenará), teniendo estos más probabilidades 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 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, 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 verían muchas generaciones hasta que demos con los requisitos, puesto que sigue en manos del azar en cierto modo, pero sigue siendo mucho más eficaz y rápido que utilizar cualquier otro método convencional, y 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 código seguiría funcionando igual).

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 y producir el efecto contrario, 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, puede ser práctico incluir un panel de control en la interfaz que permita manejar los parámetros como el ratio de mutación, el número de entidades o la condición de solución, así como poder monitorizar en cierta medida la evolución que se está produciendo y la mejor solución hasta el momento.

**2.3 - Tareas a emprender**

* Estudiar y entender en profundidad los conceptos teóricos y prácticos sobre los algoritmos genéticos y su funcionamiento.
* 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 proporcionarle una inteligencia artificial a las entidades del programa.
* Adaptar los algoritmos a los requisitos, de forma que la evolución del comportamiento de las entidades tenga un proceso y un 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 el proceso evolutivo mediante diferentes opciones que modifiquen los atributos necesarios.
* 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 pulimiento del programa (1 semana)
7. **- Arquitectura del programa**

**3.1 - Diseño de la interfaz gráfica**

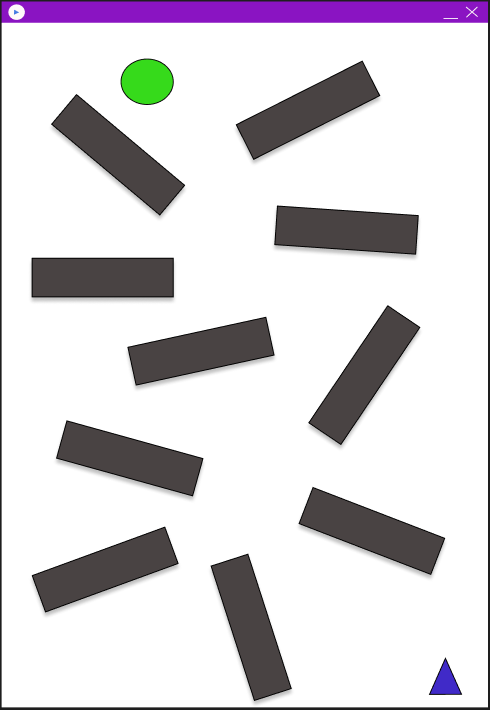
**3.1.1 - Ventana gráfica**

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

Dicha ventana contiene un conjunto de entidades en forma de triángulos 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 que actuará como la meta que deberán alcanzar las entidades al desplazarse, siempre en un punto fijo. Cuando se alcance una solución óptima, aparecerá la ruta que ha tomado la entidad más rápida desde el punto de partida hacia la meta marcada en rojo.



**Meta**

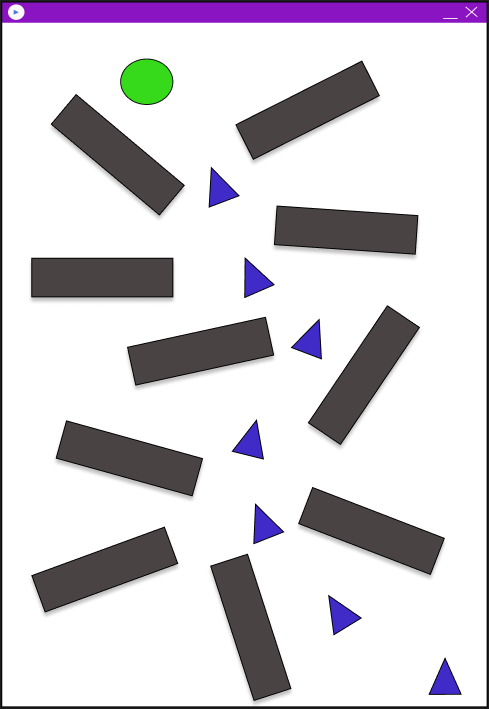
Objetivo que deben alcanzar las entidades

**Obstáculos**

Impiden a la entidad seguir moviéndose. Penalizan en el cálculo de aptitud

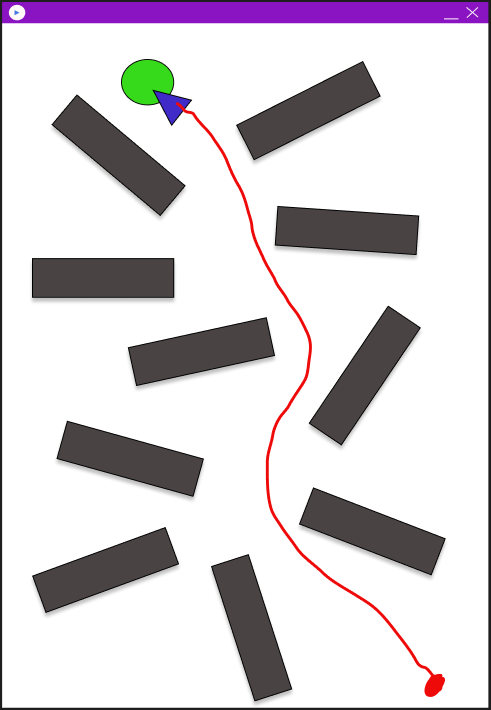
**Punto de partida de entidades**

Aquí aparecerán todas las entidades al comienzo de cada generación

****

**Entidades en movimiento**

A lo largo del programa irán desplazándose independientemente todas las entidades hacia la meta, cada vez de manera más precisa y rápida (dentro de sus límites de velocidad)

****

**Ruta óptima**

Si se han cumplido los objetivos del proceso evolutivo, no seguirán apareciendo más generaciones y se mostrará una ruta marcada en rojo desde el punto de partida hasta la meta

**Entidad que ha llegado a la meta**

Si bien habrá muchas entidades que logren alcanzar el objetivo, solo aparecerá una ruta óptima cuando se hayan cumplido los objetivos de velocidad

**3.1.2 - Panel de Control**

Además de la ventana gráfica, dispondremos de otra interfaz desarrollada en *Java Swing* gracias al 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 “Comenzar” para iniciar el proceso. Tras ello, ese botón cambiará a “Siguiente” y solo podremos pulsarlo cuando acabe el ciclo de vida de todas las entidades 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. 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”.

La siguiente sección nos permite manipular algunos parámetros del proceso, como la población total de entidades que aparecerá cada generación, la tasa de mutación, el tiempo objetivo en tiempo que deberán alcanzar, y el tiempo de vida del que dispondrán las entidades antes de reproducirse. La población solo podrá modificarse al inicio para evitar romper el proceso.

Por último, tendremos una sección donde consultar información sobre el proceso (el mejor tiempo obtenido, la mejor aptitud encontrada, cuántas entidades han alcanzado la meta, cuántas han chocado con algún obstáculo). Si pinchamos sobre una de las entidades, podremos visualizar información específica en tiempo real, como su velocidad y aceleración, así como su aptitud una vez acabe su ciclo de vida

**

**Aceleración actual**

Magnitud de la aceleración aplicada en ese instante/frame por el vector contenido en un gen (La variación que sufre la velocidad)

**Metas alcanzadas**

Cuántas veces han llegado las entidades a alcanzar la meta a lo largo del proceso

**Velocidad actual**

Magnitud de la velocidad a la que se desplaza la entidad en ese instante/frame

**Entidad seleccionada**

Entidad que hemos seleccionado con el ratón para poder monitorizar sus parámetros en tiempo real

**Mejor aptitud**

La mejor aptitud que se ha llegado a evaluar entre todas las entidades generadas hasta el momento

**Tiempo record**

Mejor tiempo hasta el momento que una entidad ha logrado para llegar a la meta

**Modificador tiempo vida**

Altera el tiempo que duran las entidades activas en el sistema antes de pasar a la selección

**Modificador objetivo**

Altera el tiempo mínimo que tiene que tardar una entidad en llegar a la meta para alcanzar la ruta óptima deseada

**Modificador mutación**

Altera la tasa de mutación para aumentar la variabilidad de los genes a la hora de evolucionar

**Modificador población**

Altera el número de entidades que se generarán cada generación. No se puede editar tras comenzar o antes de reiniciar

**Botón Reiniciar**

Reinicia el programa al estado inicial, donde se creará una primera población con unos genes nuevos

**Botón Comenzar/Siguiente**

Al principio será el botón de comenzar para iniciar el proceso.

Tras ello mostrará siguiente y servirá para pasar a otra generación

**Checkbox Modo Automático**

Al activarlo las generaciones se sucederán solas. Desactiva el botón de “Siguiente”

**Generación actual**

**Aptitud**

Aptitud evaluada para esa entidad una vez termina su ciclo de vida. Solo es visible si el modo automático no está activado ya que no daría tiempo de verlo

**3.2 - Diagramas de flujo**

**3.3 - Estructura de clases**

1. **- Estadísticas**
2. **– Epílogo**
   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.

1. **- Bibliografía**

* [**http://learningprocessing.com/**](http://learningprocessing.com/)
* [**https://processing.org/tutorials**](https://processing.org/tutorials)
* [**https://processing.org/reference**](https://processing.org/reference)
* [**https://happycoding.io/tutorials/java/processing-in-java#calling-processing-functions-from-non-sketch-classes**](https://happycoding.io/tutorials/java/processing-in-java#calling-processing-functions-from-non-sketch-classes)
* [**https://www.tutorialspoint.com/genetic\_algorithms/**](https://www.tutorialspoint.com/genetic_algorithms/)
* [**https://towardsdatascience.com/an-illustrated-guide-to-genetic-algorithm-ec5615c9ebe**](https://towardsdatascience.com/an-illustrated-guide-to-genetic-algorithm-ec5615c9ebe)
* [**https://natureofcode.com/book/**](https://natureofcode.com/book/)
* [**https://www.youtube.com/@FloatyMonkey/**](https://www.youtube.com/@FloatyMonkey/)
* [**https://www.quora.com/What-are-some-real-world-applications-of-genetic-algorithms**](https://www.quora.com/What-are-some-real-world-applications-of-genetic-algorithms)
* [**https://www.quora.com/Are-genetic-algorithms-currently-used-much-in-the-applications-of-AI-If-yes-where**](https://www.quora.com/Are-genetic-algorithms-currently-used-much-in-the-applications-of-AI-If-yes-where)
* [**https://en.wikipedia.org/wiki/Genetic\_algorithm**](https://en.wikipedia.org/wiki/Genetic_algorithm)
* [**https://processing.org/tutorials/pshape**](https://processing.org/tutorials/pshape)
* [**Separating Axis Theorem (colisiones)**](https://youtu.be/Nm1Cgmbg5SQ)