

ALGORITMOS GENÉTICOS Y EVOLUTIVOS

GRADO EN INGENIERÍA INFORMÁTICA

Desafío Final: AGE Racer

Curso 2021/2022

Raúl Giménez de Dios, Grupo 83, Leganés, 100405861@alumnos.uc3m.es
Carlos Gómez Sánchez, Grupo 83, Leganés, 100406016@alumnos.uc3m.es
Samuel Renovell González, Grupo 83, Leganés, 100405878@alumnos.uc3m.es
Jorge Rodríguez Fraile, Grupo 83, Leganés, 100405951@alumnos.uc3m.es
Carlos Rubio Olivares, Grupo 83, Leganés, 100405834@alumnos.uc3m.es

Índice

Planificación Inicial	2
Roles Designados	3
Tareas Realizadas	4
Descripción de las Tareas	6
Creación de reglas y el agente.	6
Chromosome	6
SkeletonMain	8
Algoritmo 1+1.	8
Algoritmo $\mu + \lambda$	10
Ventajas e Inconvenientes	12
Ventajas	12
Desventajas	12
Pruebas	13
Planificación Final	18
Conclusiones	18
Anexo	19

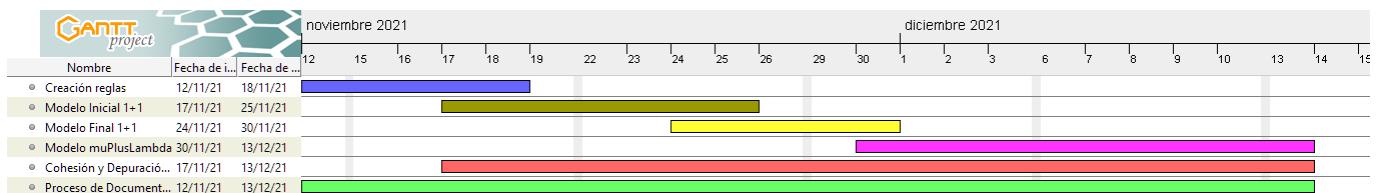
1. Planificación Inicial

Para comenzar con esta práctica, se ha decidido darle una relevancia mayor a la planificación de esta, ya que debido a su extensión la organización es un paso clave para la entrega en el plazo designado.

Para ello hemos dividido todo el trabajo a realizar en 3 grandes grupos:

- **Código:** El nombre es autoexplicativo, en esta parte se recogen todas las líneas de código generadas durante el trabajo. El objetivo principal es utilizar Estrategias Evolutivas aplicando los dos métodos explicados en la asignatura, $1+1$ y $\mu+\lambda$. Por otro lado, antes de comenzar las implementaciones se tiene pensado aplicar una solución ‘manual’ donde se ajustan los valores de las reglas probando diferentes reglas y rangos.
- **Pruebas:** Esta parte se aplica a todas las tareas asociadas a prueba de diferentes cromosomas, circuitos, recogida de resultados, etc. Se tiene como objetivo generar un set de circuitos diferenciados que permitan testear todos los aspectos de los modelos generados y poder afirmar que estos generalizan de manera correcta.
- **Documentación:** Este último apartado se refiere a todos aquellos aspectos relacionados con la cohesión y descripción del proyecto. Recoge desde la adecuación del código, documentación de la memoria, etc. Esta parte se podrá hacer en paralelo con las dos anteriores por lo que deberá ser revisada constantemente.

Una vez explicadas las diferentes fases del trabajo que deseamos realizar, se muestra un diagrama de Gantt con la planificación inicial:



2. Roles Designados

Repartir el trabajo a realizar ha sido una tarea algo más compleja de lo esperado. Esto se debe principalmente a que los integrantes del equipo deseaban participar sobre todo en la parte del código debido a que es la que más peso tiene dentro del proyecto y la que más beneficia en cuanto a conocimientos adquiridos. Debido a esto se ha intentado hacer que los integrantes puedan hacer algo de código además de estar asignados a otras fases del proyecto. El reparto de tareas ha acabado asignado de la siguiente manera:

- **Jorge Rodríguez:** Responsable de la realización del código de $1+1$ y asistencia en $\mu+\lambda$.
- **Carlos Rubio:** Responsable de la realización del código de $1+1$ y redacción de la memoria.
- **Samuel Renovell:** Responsable de la realización del código de $\mu+\lambda$ y creación de circuitos con sus respectivas pruebas.
- **Carlos Gómez:** Responsable de la realización de parte del código de AgentEE, adecuación de cromosomas y de la cohesión y depurado de código.
- **Raúl Giménez:** Responsable de la realización del código $\mu+\lambda$ y creación de circuitos con sus respectivas pruebas.

De esta manera se ha intentado darle a cada miembro del grupo un rol relevante en la creación de código y además poder aportar en otros aspectos del trabajo. Cualquier otra modificación referente a este reparto vendrá reflejado en el apartado de *Planificación Final*.

3. Tareas Realizadas

Nuestro proceso de trabajo puede ser parecido a una estructura de *pipeline* donde primero realizaremos una implementación inicial de reglas para ver cómo reacciona la vaina a los circuitos y para crear un set de atributos que se ajustarán mediante estrategias evolutivas. Una vez se observe un rendimiento decente del modelo, pasaremos a una implementación inicial de *1+1*.

Esta implementación inicial de *1+1* constará de los atributos generados por la creación de las reglas que acabarán conformando la codificación del problema y un solo valor de varianza. Esta implementación será bastante simple y nos servirá como esqueleto de la implementación final de *1+1* y para advertir si hay mejora con respecto al ajuste de atributos manual.

Una vez efectuada esta implementación inicial de *1+1*, pasamos a una implementación ‘real’ donde se asocia una varianza a cada valor de la codificación del problema. De esta manera tendremos nuestro primer método de estrategia evolutiva funcional y podremos realizar pruebas reales.

Como último punto referente a la creación de código tenemos la realización de $\mu+\lambda$. Esta parte es la más compleja de todo el proyecto, pero el código hecho anteriormente puede ser de ayuda para la generación de este método. Una vez procedido podremos centrarnos totalmente en la creación de pruebas y documentación.

En lo referente a las pruebas, se definen una serie de circuitos y se ejecutan diferentes ejecuciones sobre los mismos, obteniendo un valor de fitness medio y un error sobre el que se evaluarán los modelos generados.

Por último, como es obvio hay un trabajo paralelo de depuración de código y de documentación de todo el trabajo efectuado, de esta manera, cualquier modificación durante el proyecto puede ser modificada al momento evitando revisiones extensas.

Se muestra a continuación el esquema de tareas explicado:

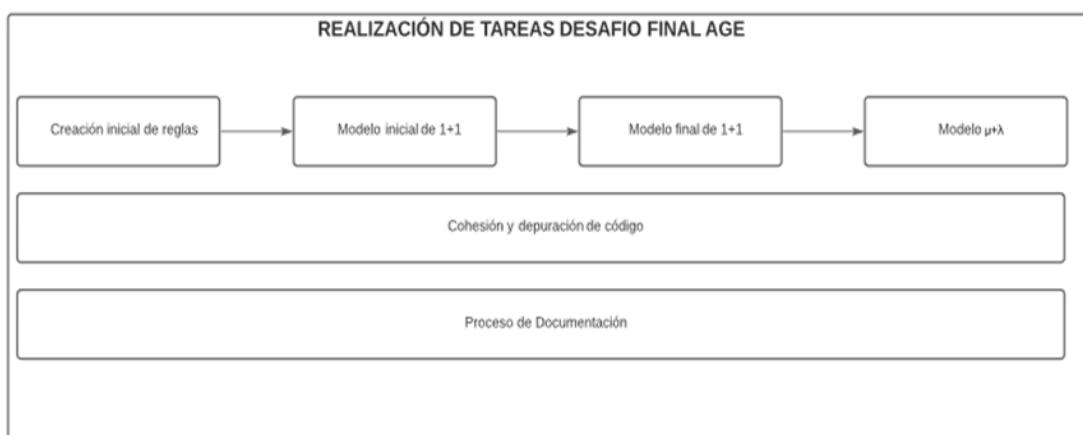


Ilustración 2: Esquema de las tareas realizadas

4. Descripción de las Tareas

4.1. Creación de reglas y el agente.

Para comenzar con el desarrollo del código se comienza aplicando unas reglas de comportamiento para la vaina y poder ver cómo se desenvuelve. Este paso es el inicial para decidir qué tipo de algoritmo usaremos (programación genética, estrategias evolutivas, etc.) En las reglas tenemos en cuenta los siguientes atributos:

- **Distance**: Distancia al siguiente punto que debe pasarse.
- **Distance2**: Distancia al segundo punto que debe pasarse.
- **relAngle**: Ángulo relativo referente al siguiente punto que debe pasarse.
- **relAngle2**: Ángulo relativo referente al segundo punto que debe pasarse.

Una vez definidas los atributos que intervienen en las reglas se procede a explicar las mismas. El algoritmo toma en cuenta rangos de distancias y ángulos, donde 'Distance' y 'relAngle' entrarán dentro de estos rangos (e igual para el segundo punto). Dependiendo de en qué rango se encuentren los atributos, se decide una velocidad.

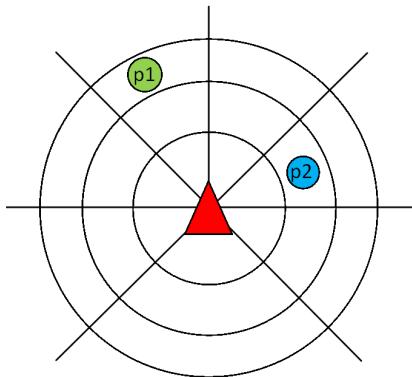
El tamaño y los valores que abarcan los rangos de distancias y vectores para cada punto viene definido por vectores de la clase 'Chromosome', y la velocidad que deberá tomar dependiendo de en qué rango caigan los atributos de entrada también está codificada en dicha clase.

Los valores de la clase Chromosome se inicializan con la información del fichero 'file/chromosome.csv', que contiene los vectores que se han mencionado antes. Antes de probar técnicas evolutivas se ha creado un fichero (file/chromosome00.csv) para que no se parta de 0, dedicando los algoritmos evolutivos a mejorar esta configuración.

4.2. Chromosome

En esta clase se describen los rangos sobre los que se basa la regla de decisión de velocidad. Estos son el rango de distancia del primer y segundo punto, ángulo del primer y segundo punto y la velocidad asociada a cada combinación.

Este es un ejemplo donde cada círculo representa el límite de un sector, por lo que sería un vector de 3 con 4 sectores de distancia (contando con el que comienza del último círculo al infinito). Cada radio representa el límite de un ángulo, por lo que aquí se contaría con un vector de 6 (el que está justo delante de la nave está siempre implícito) con 7 sectores de ángulo. Hay entonces 28 sectores donde se puede encontrar un punto, y 784 combinaciones si se cuentan con 2 puntos. El vector de velocidades asociadas tendría que ser entonces una matriz de 7x4x7x4, 784 elementos.



Como se ha dicho anteriormente, durante el desarrollo de este procedimiento han existido dos procedimientos principales, un modelo con solo una varianza asociada y otro con una varianza asociada a cada valor.

En este caso se explicará el segundo caso debido a que es el más complejo y, a fin de cuentas, engloba al primer caso. Esta sería al final la estructura del cromosoma:

- **Distance Ranges:** Una matriz cuyas filas describen los rangos de distancia que se deben tener en cuenta en los dos siguientes puntos a cruzar. Se define entre 0 y 16000.
- **Angle Ranges:** Hace el mismo trabajo que la anterior matriz, pero en este caso se tienen en cuenta los ángulos. Se define entre 0 y 360.
- **Thrust In Range:** Matriz de 4 dimensiones que recoge los valores de velocidad correspondientes en referencia a los rangos de distancias y ángulos. Se define entre 0 y 200.
- **Variance Distance:** Array de varianzas asociada a Distance Ranges.
- **Variance Angle:** Array de varianzas asociada a Angle Ranges.
- **Variance Thrust:** Array de varianzas asociado a Thrust In Range.
- **Fitness:** la función de fitness que se obtiene durante el entrenamiento. Esta será el resultado del tiempo que tarda en recorrer los mapas, y se reduce por cada punto que pasa para que también priorice recorrer puntos en caso de que no llegara a la meta.

Todos estos atributos se encuentran en la clase Chromosome. Esta clase también tiene tres constructores:

- Uno de ellos es el utilizado en $\mu + \lambda$, donde se le pasan dos individuos (en este caso los padres) para hacer una mezcla de estos, tanto en los vectores de codificación como los de varianza.
- El segundo tiene como parámetro otra instancia de la clase cromosoma, que sirve para crear un individuo 'hijo' empleando el algoritmo de mutación sobre el padre.

- El último de ellos recibe un string como parámetro que es el path a un csv para cargar los datos de este sobre el cromosoma. También se tiene una función para sobreescribir un cromosoma en el csv.

Por último, el cromosoma tiene otras dos funciones:

- Una de ellas compara la función fitness con otro cromosoma.
- La otra es capaz de escribir el cromosoma en un fichero CSV para guardarlo y que pueda visualizarse o utilizarse en otro método.

4.3. SkeletonMain

En esta clase se llaman a las demás clases para poder ejecutar la técnica que deseemos, entrenar o mostrar los resultados en el webservice que se nos da.

Por un lado, tenemos dos métodos principales: Uno crea una instancia de $1+1$ y otra de $\mu + \lambda$. Otra función es llamada para entrenar a los modelos, y otra para mostrar la ejecución en el host.

No hay otras funciones destacables en la clase, ya que la función básica de la misma es la de ejecutar creando instancias de clases ya existentes habiendo sido otorgado gran parte del código realizado en esta clase.

4.4. Algoritmo $1+1$.

El proceso principal de $1+1$ es bastante simple debido a que se ha conseguido asimilar parte del proceso en la clase Chromosome. Todo el proceso se resume en los siguientes puntos:

- La clase OnePlusOne recoge todo el algoritmo y tiene como atributos un cromosoma que será el individuo de la población, un contador de ciclos, el n.º de mejoras y los ciclos en los que se evalúan estas mejoras para la regla del 1/5.
- La función ‘execute’ hace la gran parte del trabajo. Se crea un cromosoma hijo a partir del padre a partir del constructor explicado y se calcula su fitness.

$$\text{fitness}(I) = \frac{1}{|E|} \sum_{j \in E} \text{score}(I, j) - 5 \cdot \text{checkpointsCollected}(I, j)$$

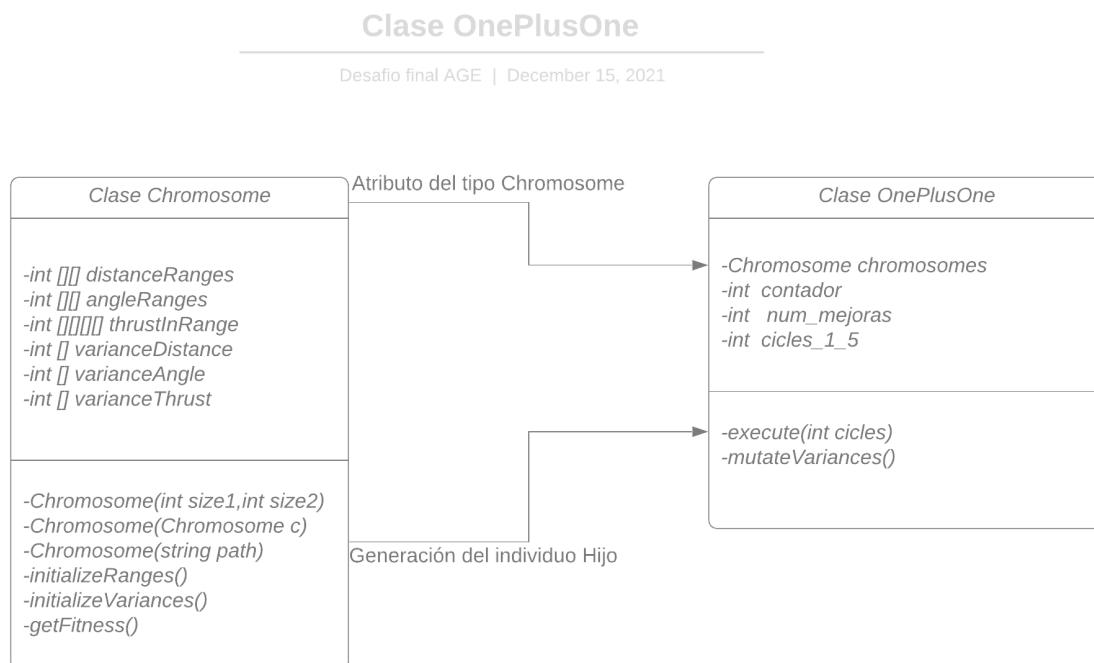
- **E**: Conjunto de mapas de entrenamiento. $|E|$ es la cardinalidad del conjunto.
- **score(I, j)**: Puntuación del individuo I en el mapa j.
- **checkpointsCollected(I, j)**: Número de puntos de control recogidos por el individuo I en la carrera por el circuito j.

Una vez hecho esto, se evalúa el fitness del padre vs. el del hijo para definir qué cromosoma se quedará en la siguiente generación. Por último se realiza la mutación de la varianza usando la regla del 1/5. Este proceso se realiza un número de ciclos determinado por parámetro.

- Para ejecutar este algoritmo se crea en el main un objeto del tipo OnePlusOne y se ejecuta la función execute.

Para concluir con la explicación, resaltar que como se puede observar el modelo final no tiene una varianza asociada a cada valor, sino que más bien que los vectores de ángulo y distancia están duplicados (para ver el siguiente punto), pero las dos filas se fijan en el mismo vector de varianza, lo mismo pasa con la velocidad, estas son cuatro vectores, pero solo necesitan dos de varianza. Esto se debe a que la complejidad era excesiva para generar un cambio mínimo en los resultados.

A continuación se muestra un esquema de todo el proceso explicado anteriormente:



4.5. Algoritmo $\mu + \lambda$

Este algoritmo tiene como objetivo mejorar los mejores individuos generados con la técnica anterior. Las principales diferencias entre los métodos usados se basan en su forma de generación de la población y la mutación.

Para comenzar, debemos definir los elementos clave del algoritmo, en este caso el tamaño de la población, que será de 10 por defecto para evitar un tiempo de computación excesivo. También se fija el valor de λ y se crean 2 razones de aprendizaje para varianzas.

El proceso es el siguiente:

- Como se ha mencionado anteriormente, este algoritmo se utilizará para mejorar individuos de 1+1, por tanto, se leen los individuos deseados en csv hasta llenar la población.
- Una vez creada la población inicial, pasamos a la ejecución del algoritmo en sí. Para ello debemos desordenar la población para que los padres se elijan de manera aleatoria.
- El siguiente paso, en cada iteración, es crear un individuo con los padres adyacentes, este cromosoma hijo se genera en este caso con un constructor de la clase Chromosome donde se le pasan los dos padres. En el caso de que el n.º de hijos exceda el valor de lambda, los padres se escogerán de manera aleatoria.
- Al generar los hijos, debemos evaluarlos y añadirlos a la población intermedia. Hecho esto, se ordenan los cromosomas de la población teniendo en cuenta su fitness, los últimos serán eliminados de la población intermedia de manera elitista, resultando así la nueva población.
- Obtenida la nueva población debemos mutar las varianzas de los individuos seleccionados. La mutación de varianzas en $\mu + \lambda$ es diferente a la de 1+1, donde se usa la regla del 1 / 5. Para esto las varianzas se mutan siguiendo la siguiente fórmula:

$$\sigma' = e^{N(0, \tau_0)} \cdot \sigma \cdot e^{N(0, \tau)}$$

- Donde τ es la tasa de aprendizaje y σ es la varianza que deseamos mutar. Esta mutación se realiza sobre los atributos de *varianceDistance*, *varianceAngle* y *varianceThrust*. Con esto, damos por finalizada la ejecución de un ciclo del algoritmo $\mu + \lambda$.

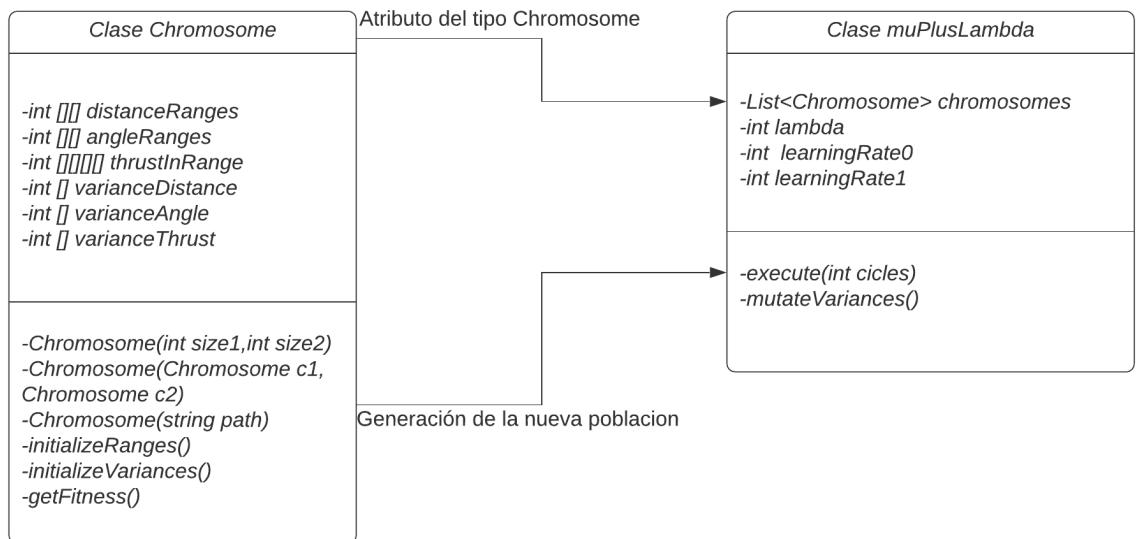
La ejecución general es muy parecida a la de 1+1. En *SkeletonMain* creamos una instancia de la clase *MuPlusLambda* indicando la cantidad de ciclos que deseemos. Esto facilita al equipo enormemente la implementación del entrenamiento.

Cabe recalcar que se generan dos razones de aprendizaje para el objetivo de la experimentación. La clase *Chromosome* es idéntica a la clase empleada en 1+1, ya que el objetivo es generalizar lo máximo posible y evitar duplicado de código innecesario.

A continuación se muestra un esquema de la arquitectura del código referente a esta técnica:

Clase muPlusLambda

Desafio final AGE | December 13, 2021



5. Ventajas e Inconvenientes

El hecho de que existan diversas técnicas de algoritmos genéticos implica ciertas ventajas y desventajas a la hora de implementarlos. Por esto, se explican los pros y contras de nuestro código a continuación:

5.1. Ventajas

- La implementación de $1+1$ y $\mu + \lambda$ son bastante parecidos, lo que nos permite generar el código bastante más rápido de lo esperado.
- El marco teórico de estrategias evolutivas está bastante más fundamentado en todo el equipo debido a la realización de prácticas anteriores.
- La coordinación y división de la carga de trabajo ha facilitado la generación de código, pudiendo cada integrante centrarse en una tarea.
- La documentación del código en paralelo con la generación de este ha permitido la rápida actualización de la memoria y tener un historial de versiones.
- La práctica se ha desarrollado en un repositorio que ha permitido el trabajo en paralelo mediante la creación de ramas, esto ha sido enormemente efectivo en la generación del código de $\mu + \lambda$.

5.2. Desventajas

- El uso de Java como lenguaje de programación ha dado como resultado algunos problemas de adaptación en las primeras fases de desarrollo.
- El uso de estrategias evolutivas no nos asegura el mejor de los resultados.
- Se quedan sin explorar otras alternativas como programación genética o coevolución.

6. Pruebas

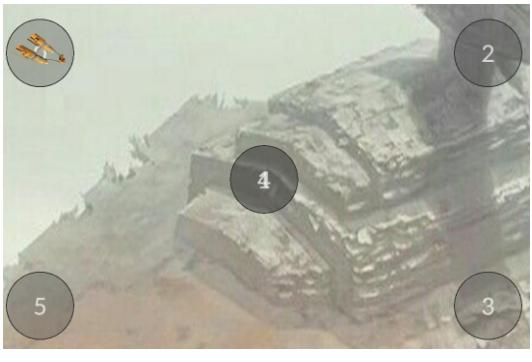
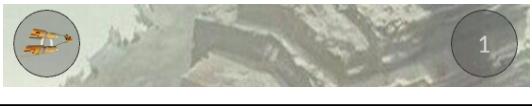
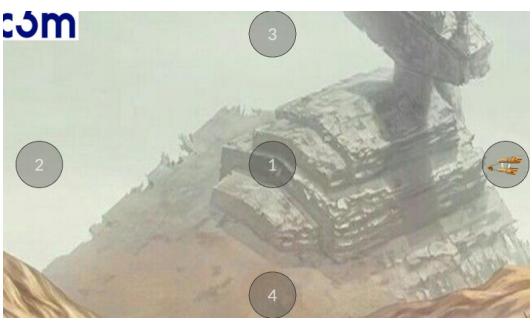
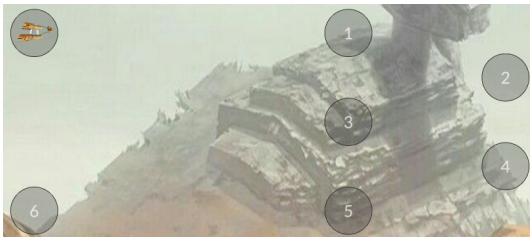
En cuanto a las pruebas realizadas se han tenido en cuenta diversos puntos:

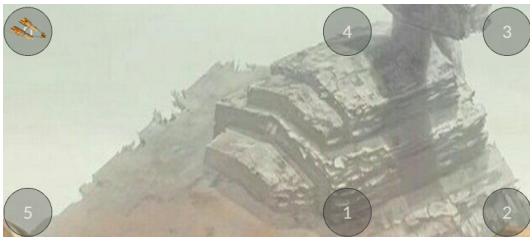
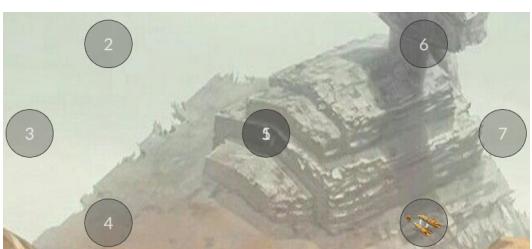
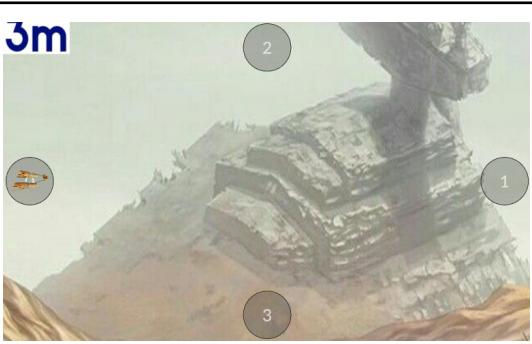
- Se han ejecutado varios modelos con un cromosoma base (no se inicializa aleatoriamente)
- Se generan 10 modelos eligiendo un set de mapas y se entrena utilizando 1+1, cada integrante ha elegido los suyos, además de los hiperparámetros (número de ciclos o inicialización de varianzas), por lo que los individuos resultantes serán diferentes.
- Los mejores cromosomas generados formarán parte de la población de $\mu + \lambda$ entrenando con los 15 mapas.

Una vez obtenidos los resultados podremos procesar qué modelo puede ser el mejor y por qué.

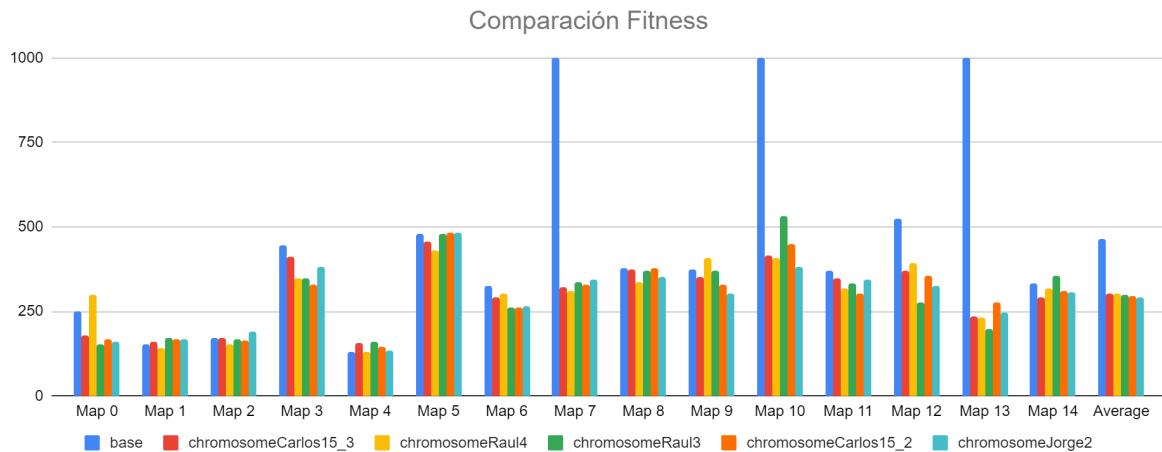
Por otro lado, tenemos que tratar los mapas creados. Se muestran ahora las descripciones de los mapas:

DESCRIPCIÓN	CIRCUITO
Test0: Circuito básico con 4 checkpoints que busca hacer giros de 90° con efectividad.	
Test1: Este circuito busca giros muy cerrados, muy parecidos a la primera mitad de una 'chicane'	
Test2: Con forma triangular, intentamos mejorar el giro de la vaina tras una recta.	

<p>Test3: Se empiezan aumentando los checkpoints del circuito, haciendo una pequeña modificación sobre el circuito inicial añadiéndole un checkpoint central que obliga a hacer giros más variados.</p>	
<p>Test4: Centrado en el sprint de la vaina.</p>	
<p>Test5: Circuito con giros muy pronunciados y sprints muy largos como en el punto 2-3.</p>	
<p>Test6: Un mapa algo más simple que aumenta las dimensiones del circuito original, para ver cómo se maneja en mapas algo más grandes.</p>	
<p>Test7: Circuito con un recorrido bastante simple, parecido a los giros realizados en el Test5, pero con una extensión mayor.</p>	
<p>Test8: Circuito que prueba la capacidad de hacer zigzags de la vaina.</p>	

<p>Test9: Se le añaden 2 checkpoints bastante alejados al circuito de Test0, modificando un poco su recorrido con unas rectas más grandes.</p>	
<p>Test10: Recorrido Rectangular con numerosos checkpoints, para ver si la vaina se maneja correctamente con un número alto de puntos de control.</p>	
<p>Test11: Se destacan los sprints diagonales y giros cerrados como en el punto de control 2.</p>	
<p>Test12: Mapa que mezcla todo lo testeado anteriormente, giros cerrados, sprint, algo de zigzag y alto número de checkpoints.</p>	
<p>Test13: Versión simplificada de Test12 donde se elimina el punto de control central.</p>	
<p>Test14: Énfasis en giros de 120° (aproximadamente).</p>	

Explicada la distribución de los mapas, pasamos a mostrar los resultados de los 5 mejores modelos de 1+1 comparados con el cromosoma base:

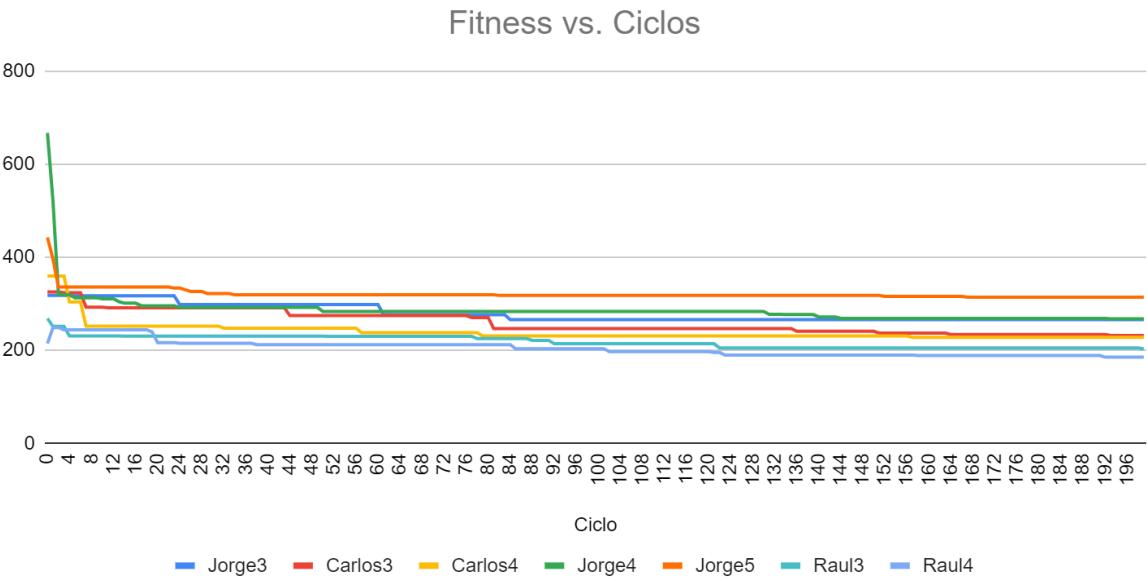


Model	Entrenado con	Average
base	A mano	462,79
chromosomeCarlos15_3	{3, 4, 5, 8, 9, 11, 13}	302,42
chromosomeRaul4	{1, 2, 6, 7, 8, 9, 11, 13}	302,33
chromosomeRaul3	{1, 2, 6, 7, 8, 9, 11, 13}	300,86
chromosomeCarlos15_2	{3, 4, 5, 8, 9, 11, 13}	296,98
chromosomeJorge2	{3, 5, 7, 8, 9, 10, 11, 12, 13, 14}	292,54

Se puede observar una mejora clara en torno al cromosoma inicial, sobre todo en mapas más complicados como el 10. Además, los cromosomas dan valores más o menos parecidos, no teniendo un claro ganador en todos los mapas, por lo que puede ser interesante trasladar estos cromosomas a una población en $\mu + \lambda$. También se han entrenado cromosomas con los 15 mapas para tener una solución con 1+1.

Los cromosomas iniciales han sido generados de manera manual y no de manera aleatoria, y, como se ha dicho anteriormente, se han entrenado con diferentes sets de mapas, haciendo que unos se especializan en factores donde otros individuos muestran carencias. Esto es clave para generar una población con mucha diversidad en el algoritmo $\mu + \lambda$.

El algoritmo $\mu + \lambda$ ha sido, sin embargo, incapaz de generar una mejor solución que con 1+1. Se han probado diferentes estrategias para que aumente la explotación, como reducir el vector de varianzas a una proporción de 200 ciclos, pero aun así los mejores algoritmos son los entrenados por 1+1. Se concluye pues que para este problema no es necesario ajustar más con $\mu + \lambda$, siendo 1+1 suficiente.



En esta gráfica se puede observar la evolución del fitness de los diferentes agentes comentados previamente a lo largo de los ciclos. La tendencia inicial es reducir más notoriamente el fitness durante los primeros ciclos para aquellos agentes que comienzan con un valor de fitness alto como son Jorge4, Jorge5 o Carlos 4, siendo la posterior evolución más progresiva.

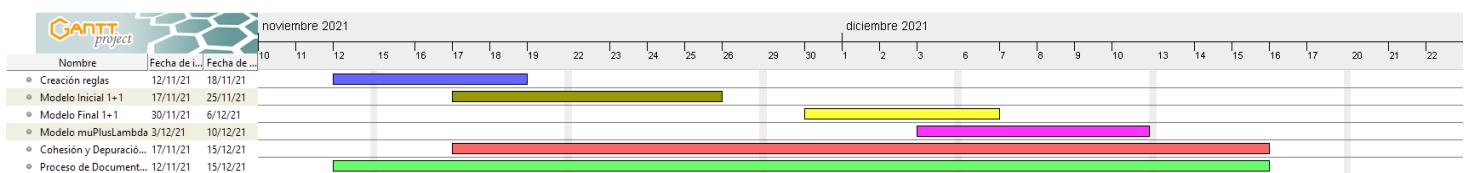
Por otro lado, aquellos agentes que inicialmente comienzan con un valor de fitness más intermedio sufren una evolución progresiva constante a lo largo de los ciclos hasta llegar finalmente a su mejor valor. También podemos observar un caso atípico como es Raul4 que inicialmente sufre una pequeña subida en su valor de fitness al comienzo de los ciclos, sin embargo, pronto vuelve a bajar y evoluciona progresivamente hasta dar con su mejor valor.

7. Planificación Final

Al finalizar la práctica, la división del trabajo ha quedado ligeramente modificada, ya que cada integrante del equipo, aparte de todos los trabajos asignados en la planificación inicial, cada miembro del equipo ha generado dos modelos para las pruebas sobre 1+1. Por lo demás todo se ha mantenido dentro de lo esperado (asumiendo que algunas veces alguien se ha tenido que trasladar a otra parte del código para ayudar con pequeños bugs o errores).

Por otro lado, en cuanto al tiempo asignado para la realización de la práctica y las fases mostradas en el diagrama de Gantt inicial ha habido un desplazamiento bastante grande, debido a que ha habido otros trabajos y proyectos en los que se ha tenido que trabajar de manera simultánea, lo que ha obligado al equipo a posponer algunas de las fases.

A continuación se muestra el diagrama de Gantt con las modificaciones comentadas:



Como se puede observar el modelo final de 1+1 se tuvo que desplazar un tiempo debido a que colisionaba con otros proyectos que los integrantes debían realizar aunque el desarrollo de $\mu + \lambda$ acabó con menos días empleados. Los procesos más generales como cohesión y documentación también se han visto desplazados.

8. Conclusiones

Para concluir, esta práctica ha servido para afianzar nuestros conocimientos sobre las estrategias evolutivas. Además, aplicarlo en un entorno donde se pueda apreciar de manera gráfica los resultados facilita la evaluación de estos y poder aplicar mejoras a los modelos mostrados.

Por otro lado, nos hemos encontrado diversos problemas, sobre todo a la hora de comenzar con el proyecto debido a que no conocíamos en profundidad el código ni la libertad que podríamos tener a la hora de crear el nuestro. Además, el uso de java ralentizó todo el trabajo, ya que es un lenguaje que hace tiempo ningún miembro del equipo utilizaba, aun así, fueron inconvenientes menores que se pudieron resolver durante las clases.

Por último, hemos de mencionar que los resultados obtenidos son bastante mejores de lo esperado sobre todo teniendo en cuenta que se utilizaban Estrategias Evolutivas como elemento central, los tiempos de ejecución no fueron excesivos y creemos que el equipo puede generar muy buenos resultados durante la competición debido al set de mapas tan diverso del que se dispone.

9. Anexo

Hoja de cálculo con todos los experimentos: [Resultados](#)