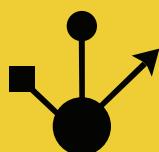




Escuela
Politécnica
Superior

Representación 3D de personajes sintéticos generados mediante algoritmos genéticos



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:

Rubén Rubio Martínez

Tutor/es:

Rafael Molina Carmona

Patricia Compañ Rosique



Universitat d'Alacant
Universidad de Alicante

Representación 3D de personajes sintéticos generados mediante algoritmos genéticos

Herramienta para la visualización y estudio del comportamiento de un
algoritmo genético

Autor

Rubén Rubio Martínez

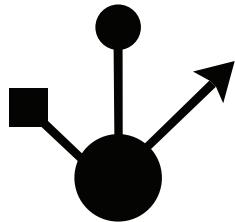
Tutores

Rafael Molina Carmona

Ciencia de la computación e inteligencia artificial

Patricia Compañ Rosique

Ciencia de la computación e inteligencia artificial



Grado en Ingeniería Multimedia



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Diciembre 2020

Resumen

Este proyecto parte con el objetivo principal de facilitar el entendimiento de uno de los múltiples campos con los que cuenta la inteligencia artificial (IA), los algoritmos genéticos (AAGG). La manera de llegar a cumplir este objetivo ha sido mediante la representación visual de su ejecución y datos.

Para conseguir esta representación visual se ha planteado el problema de conseguir enseñar a andar a una estructura bípeda mediante los ya mencionados AAGG. Primero, se ha realizado una investigación exhaustiva de la utilización de dichos algoritmos y se han presentado proyectos similares realizados con ellos.

Ha sido necesario el desarrollo de un motor gráfico y físico para poder recrear y visualizar la evolución y sus resultados. Una de las mejores maneras para entender el funcionamiento de algo es pudiendo variar y estudiar cada parámetro involucrado, por este motivo, la herramienta desarrollada cuenta con multitud de parametrización relativa al funcionamiento del algoritmo genético (AG).

Finalmente, se ha utilizado dicha herramienta para extraer y detallar los resultados finales, es decir, la solución al problema planteado así como el estudio de cada uno de los parámetros que intervienen en la ejecución del algoritmo. De esta manera, el usuario cuenta tanto con el resultado final como con la posibilidad de utilizar la herramienta y extraer sus propias conclusiones, ya que no hay mejor manera de aprender algo como hacerlo tú mismo.

Abstract

One of the goals in this project is to facilitate the understanding about one of the many fields in the artificial intelligence, genetic algorithms. The way to achieve this goal has been through the visual representation of both execution and data.

In order to achieve this visual representation, the problem about teaching how to walk to a bipedal structure with the aforementioned genetic algorithms has been raised. First, an exhaustive investigation of the use of these algorithms has been carried out and similar projects made with them have been presented.

A graphic and physic engine have been necessary in order to be able to visualize the evolution and results. One of the best ways to understand the behaviour of something is being able to apply changes and study every parameter, by that, this tool develops counts with multitude of parametrization relative to the genetic algorithm behaviour.

Finally, this tool has been used to extract and detail the final results, that is, the solution to the given problem and the study of every parameter involved in the execution of the algorithm. This way, the user counts with the final results aswell as the possibility of using the tool to extract his own conclusions since the best way to learn something is by doing by yourself.

Resum

Aquest projecte parteix amb l'objectiu principal de facilitar l'enteniment d'un dels múltiples camps amb els quals compta la intel·ligència artificial, els algoritmes genètics. La manera d'arribar a complir aquest objectiu ha sigut mitjançant la representació visual de la seua execució i dades.

Per a aconseguir aquesta representació visual, s'ha plantejat el problema d'aconseguir ensenyar a caminar a una estructura bípeda mitjançant els ja esmentats algoritmes genètics. Primer, s'ha realitzat una investigació exhaustiva de la utilització d'aquests algoritmes i s'han presentat projectes similars realitzats amb ells.

Ha sigut necessari el desenvolupament d'un motor gràfic i físic per a poder recrear i visualitzar l'evolució i els seus resultats. Una de les millors maneres per a entendre el funcionament d'alguna cosa és podent variar i estudiar cada paràmetre involucrat, per aquest motiu, l'eina desenvolupada compta amb multitud de parametrització relativa al funcionament de l'algoritme genètic.

Finalment, s'ha utilitzat aquesta eina per a extraure i detallar els resultats finals, és a dir, la solució al problema plantejat així com l'estudi de cadascun dels paràmetres que intervenen en l'execució de l'algoritme. D'aquesta manera, l'usuari compta tant amb el resultat final com amb la possibilitat d'utilitzar l'eina i extraure les seues pròpies conclusions, ja que no hi ha millor manera d'aprendre alguna cosa com fer-ho tu mateix.

Motivación, justificación y objetivo general

Durante estos últimos años el mundo esta experimentando un "boom" similar al ya vivido con internet o los dispositivos móviles, en este caso se trata de la IA, un campo de la informática que está llamado a ser el futuro de la evolución tecnológica y del cual quiero formar parte.

Con esta premisa decidí contactar personalmente con profesores especializados para que me pudieran guiar mejor en la elección de mi trabajo final de grado (TFG). Tras varias reuniones moldeamos la idea final en función de lo que yo buscaba aprender y conseguir con este proyecto; profundizar en el aprendizaje de gráficos que ya inicié durante el 4º año de carrera, comenzar un aprendizaje sobre inteligencia artificial de cara al futuro profesional que deseo y sobretodo, un reto que me sirviera para mejorar como programador y poner a prueba los conocimientos aprendidos durante estos 4 años.

El objetivo principal de este proyecto no es solo crear un producto final, es que yo aprenda y me forme durante el aprendizaje, ya que considero que un programador siempre debe estar en constante evolución y no estancarse. Por ello, lo enfocamos reduciendo al mínimo las herramientas previamente implementadas ya que busco entender a la perfección el funcionamiento de aquello que desarrollo, ya que considero que lo que convierte a una persona en un ingeniero y un programador de alto nivel es su capacidad de comprender y solucionar aquello a lo que se enfrenta.

Otros de los objetivos que me atraen de este proyecto es el hecho de conseguir enseñar a otras personas, ya que siempre he pensado que la persona más indicada para enseñar algo siempre será la persona que ha pasado por las mismas dudas y problemas que tú. Por ello, enfocar el proyecto como una ayuda a gente que busque herramientas para entender y formarse en alguna disciplina es algo que me motiva y me hace pensar de otra manera cada funcionalidad que implemento. El hecho de crear herramientas libres y gratuitas para facilitar

el aprendizaje de otras personas es el motivo por el cual evolucionamos y sin esas facilidades yo tampoco podría haber llegado hasta donde estoy ahora.

Agradecimientos

Agradecer a todas las personas que me han ayudado a facilitar el camino hasta llegar aquí.
Ya sean familiares, profesores o artistas que me han ayudado durante estos cuatro años

Índice general

1	Introducción	1
2	Planificación	3
3	Estado del arte	5
3.1	La inteligencia artificial	5
3.1.1	¿Qué es la inteligencia artificial?	5
3.1.2	¿Para qué se utiliza inteligencia artificial en la actualidad?	9
3.2	¿Qué es un algoritmo genético?	10
3.3	Aplicaciones de los algoritmos genéticos.	12
3.3.1	BipedalWalker-v2	13
3.3.2	Flexible Muscle-Based Locomotion for Bipedal Creatures	13
3.3.3	Slime Volleyball	14
3.3.4	CarRacing-v0	15
3.4	Motores gráficos	15
3.4.1	Unity	15
3.4.2	Unreal Engine	17
3.4.3	Spark Engine	18
4	Objetivos	21
5	Metodología	23
5.1	Herramientas de gestión de proyecto	23
5.2	Tecnologías	23
5.2.1	Gestión de versiones	23

5.2.2	Visual studio 2019	24
5.2.3	Motor gráfico	25
5.2.4	Algoritmo genético	26
5.3	Hardware	27
6	Implementación	29
6.1	Representación visual	29
6.1.1	Elementos del motor gráfico	29
6.1.1.1	Árbol de la escena	29
6.1.1.2	Gestor de recursos	34
6.1.2	Dibujado de lineas 3D	36
6.2	Físicas	38
6.2.1	Estructura bípeda	38
6.2.2	Colisiones OBB	40
6.2.3	Cinemática directa vs cinemática inversa	42
6.2.4	Gravedad	44
6.2.5	Dinámica del movimiento	45
6.3	Algoritmo genético	45
6.3.1	Generación de la población	46
6.3.2	Función fitness	48
6.3.3	Selección	48
6.3.4	Crossover o cruce	50
6.3.5	Mutación	54
6.3.6	Visualización de la información	54
7	Experimentación	59
7.1	Margen de error	59
7.2	Tamaño de población	60
7.3	Número de generaciones	61
7.4	Porcentaje de nuevos individuos	63
7.5	Porcentaje de mutación	64

7.6 Función de selección	65
7.7 Operador de cruce	66
7.8 Resultados de la experimentación	67
8 Conclusiones y trabajo futuro	71
8.1 Conclusiones	71
8.2 Trabajo futuro	72
Bibliografía	73
Lista de Acrónimos y Abreviaturas	77

Índice de figuras

2.1	Diagrama de Gantt del proyecto	4
3.1	Comparativa de una red neuronal simple y una red neuronal utilizada en el deep learning. Fuente https://thedataScientist.com/what-deep-learning-is-and-isnt/	7
3.2	Lógica tradicional vs lógica difusa. Fuente https://www.javatpoint.com/fuzzy-logic	7
3.3	Ejemplo de una red bayesiana con ponderación de aristas. Fuente https://es.wikipedia.org/wiki/Red_bayesiana	8
3.4	Ejemplo de segmentación que realiza la inteligencia artificial para poder identificar cada elemento que los seres humanos vemos con nuestra vista. Fuente https://www.slideshare.net/BillLiu31/computer-vision-for-autonomous-driving	9
3.5	Imagen de función de selección por ruleta. Fuente: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm .	11
3.6	Diagrama de flujo de un algoritmo genetico. Fuente: https://www.scielo.br/scielo.php?script=sci_arttext&pid=S1679-78252018000300508&lng=en&nrm=iso&tlang=en	12
3.7	Imagen del proyecto BipedalWalk. Fuente: https://gym.openai.com/envs/BipedalWalker-v2/	13
3.8	Imagen del proyecto Muscle Based Locomotion for Bipedal Creatures. Fuente: https://www.youtube.com/watch?v=pgaEE27nsQw	14
3.9	Imagen del proyecto Slime Volleyball. Fuente: https://github.com/hardmaru/slimevolleygym/	14

3.10 Imagen del proyecto Car Racing. Fuente: https://gym.openai.com/envs/CarRacing-v0/	15
3.11 Diagrama de un programa con paradigma entidad-componente-sistema. Fuente: https://pt.slideshare.net/unity3d/scaling-cpu-experiences-an-introduction-to-the-entity-component-system/	16
3.12 Comparativa de velocidades entre C++ y C#. Fuente: http://www.cyqdata.com/cnblogs/article-detail-40248	17
3.13 Market share en función del motor de videojuegos utilizado para la implementación. Fuente: https://www.reddit.com/r/gamedev/comments/8s20qp/i_researched_the_market_share_of_game_engines_on/	18
3.14 Imagen propia del videojuego <i>Beast Brawl</i> realizado con Spark Engine.	19
5.1 Interfaz de Gitkraken.	24
5.2 Ejemplo de la librería de Dear ImGui. Fuente: https://github.com/ocornut/imgui	26
6.1 Ejemplo de un skybox. Fuente: https://learnopengl.com/	31
6.2 Matrices de transformaciones geométricas definidas para la traslación, escalado y rotación al rededor de los tres ejes. Fuente: http://math.hws.edu/graphicsbook/c3/s5.html	32
6.3 Matriz vista.	33
6.4 Flujo del pipeline gráfico de OpenGL. Fuente: https://learnopengl.com/Getting-started/Hello-Triangle	35
6.5 Todas las formas de renderizado implementadas en OpenGL. Fuente: https://artificialnature.net/gct633/graphics.html	37
6.6 Esqueleto renderizado en la ejecución del proyecto. Fuente propia.	40
6.7 Comparativa de una bounding box axis aligned bounding box (AABB) y una oriented bounding box (OBB). Fuente: https://devdept.zendesk.com/hc/en-us/articles/360011559320-How-Collision-Detection-works	41
6.8 Esqueleto renderizado y dibujado de sus OBBs.	42

6.9 Posibles soluciones que puede producir un movimiento mediante cinemática inversa. Fuente: http://faculty.salina.k-state.edu/tim/robotics_sg/Arm_robots/inverseKin.html	43
6.10 Imagen del menú de configuración del proyecto.	46
6.11 Ejemplo práctico de una selección por torneo con $N = 3$. Fuente: https://medium.com/datadriveninvestor/genetic-algorithms-selection-5634cfc45d78	50
6.12 Cruces entre cromosomas diferentes que generan los mismos resultados en hijos diferentes y hacen converger el resultado muy rápidamente. Fuente: Imagen propia.	53
6.13 Cruce por operador <i>one point</i> . Fuente: https://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php	53
6.14 Imagen de explicación para cada parámetro de la ventana de resumen de las estadísticas en tiempo real del proyecto.	55
6.15 Imagen de los cuatro usos que se le da a la gráfica de líneas para representar los datos del porcentaje de muertes, fitness medio, mejor fitness y peor fitness.	56
6.16 Imagen extraída del programa con la cámara de seguimiento al mejor individuo de la generación pasada.	57
7.1 Secuencia de frames andando del mejor individuo.	70

Índice de tablas

6.1	Tabla de las variables presentes en la clase CLCamera.	30
6.2	Variables encapsuladas dentro de la clase <i>Entity</i>	38
6.3	Variables propias de la clase <i>EMesh</i>	39
6.4	Variables propias de la clase <i>ESkeleton</i>	39
6.5	Variables propias de la clase <i>OBBCollider</i>	42
6.6	Valores posibles en función de la flexibilidad del individuo.	48
6.7	Tabla de valores almacenados en cada generación.	58
7.1	Resultado de pruebas con diferentes tamaños de población y tiempos de vida pero sin selección ni mutación para representar el margen de error de un individuo.	60
7.2	Resultados de las pruebas con diferentes tamaños de población y funciones de selección para determinar el número de individuos óptimo.	61
7.3	Resultados de las pruebas con 25 generaciones máximas y diferentes operadores de cruce para estudiar sus valores y punto de convergencia.	62
7.4	Resultados de las pruebas con 50 generaciones máximas y diferentes operadores de cruce para estudiar sus valores y punto de convergencia.	62
7.5	Resultados de las pruebas con 100 generaciones máximas y diferentes operadores de cruce para estudiar sus valores y punto de convergencia.	63
7.6	Resultados de las pruebas con 200 generaciones máximas y diferentes operadores de cruce para estudiar sus valores y punto de convergencia.	63
7.7	Resultados en función del porcentaje de nuevos individuos y la función de selección.	64
7.8	Resultados obtenidos variando el porcentaje de mutación del algoritmo genético. 65	

7.9 Resultados obtenidos variando la función de selección	66
7.10 Resultados obtenidos variando el operador de cruce	67
7.11 Valores finales del individuo con mejores resultados en los experimentos.	69

Índice de Códigos

6.1	Pseudocódigo del flujo de calculo de la matriz MVP utilizando la librería GLM	33
6.2	Pseudocódigo de ejemplo para realizar la lectura de shaders con el gestor de recursos	36
6.3	Vertex shader para el dibujado de lineas 3D	37
6.4	Sistema de gravedad del proyecto	44

1 Introducción

Desde la invención de los ordenadores y la computación las personas hemos hecho uso de ello para automatizar y agilizar tareas que antes nos llevarían grandes cantidades de tiempo realizar. Aunque esto de por sí fuera una grandísima ayuda, a veces se quedaba corta para muchas de nuestras necesidades ya que dependía previamente de que la persona entendiera a la perfección el problema y las posibles soluciones. En muchísimos casos esto se convertía en trabajo imposible, sobretodo con aquellas tareas que las personas realizamos gracias a nuestra experiencia o memoria muscular: Andar, conducir, dibujar, escribir, dar opiniones subjetivas, etcétera...

Uno de los campos que busca ser la solución a esos problemas de la computación es la inteligencia artificial ya que su principio se basa en no programar la solución a un problema sino en hacer entender a una máquina cuál es el camino para llegar a ella, en definitiva: enseñar. Takeyas (2007) *"la IA es una rama de las ciencias computacionales encargada de estudiar modelos de cómputo capaces de realizar actividades propias de los seres humanos con base en dos de sus características primordiales: el razonamiento y la conducta"*.

Si bien es cierto que la inteligencia artificial no es algo nuevo, ya que la primera vez que se definió el término *inteligencia artificial* fue John McCarthy en 1957, no fue hasta la mejora de potencia en la computación cuando hemos comenzado a sacarle el máximo potencial a esta nueva disciplina de la informática.

Todos somos capaces de andar o subir unas escaleras, pero ¿recuerdas cómo aprendiste a hacerlo? Seguramente tus padres te lo enseñaran a base de prueba y error y tras centenares de intentos, caídas y lloros por fin lograste comenzar a dar unos primeros y tímidos pasos que con los años perfeccionaste hasta llegar a lo que haces ahora. Pues este es un buen símil para entender el funcionamiento de las técnicas de inteligencia artificial como el *machine learning* o los AAGG (Mitchell, 1998).

En el caso concreto de los AG son muy útiles ya que buscan imitar los procesos demostrados en la teoría de *Darwin*, basados en la evolución de una población ante las adversidades del medio. Esto es una herramienta potentísima para los programadores que desconozcan cuál puede ser la solución mas eficiente a un problema planteado y busquen estudiar una evolución y que el propio problema se resuelva a sí mismo mediante la optimización de individuos.

Es precisamente por esta misma premisa por la que en este proyecto haremos uso de los AG para llegar a la solución más eficiente de un problema planteado y poder estudiar cómo afecta al resultado final la modificación de determinados parámetros sobre una población. Esto puede ser una herramienta muy útil para, no solo conseguir la solución a un problema, sino el estudio y entendimiento de los factores tanto internos como externos al problema que se ven involucrados y afectan a la solución final.

En esta memoria también se mencionará otro campo de la informática como son los gráficos por ordenador, ya que han sido necesarios para la representación visual y poder dotar de mejor entendimiento y dinamismo al funcionamiento de la IA. Por otro lado, debido al tema y objetivo del estudio también se hablará sobre la implementación de físicas en los ordenadores ya que ha sido necesaria su implementación para simular unas condiciones similares a las que se dan en la vida real.

2 Planificación

La planificación de este proyecto fue uno de los puntos más importantes en el desarrollo debido a que se escogió la convocatoria de diciembre como fecha de entrega del proyecto y sumado a que desde junio hasta octubre yo estuve realizando mis prácticas de empresa, esto exigió una planificación muy bien pensada y ajustada para poder llegar bien al final del desarrollo.

Las primeras reuniones con mis tutores para hablar y especificar como sería el proyecto fueron a principios de julio. En estas reuniones se dejó claro todo lo que debía hacerse durante el proyecto.

Debido a que paralelamente estaba realizando mis prácticas de empresa, decidimos que hasta terminarlas mis tareas serían de investigación y preparación de materiales y recursos necesarios para el proyecto: Investigación sobre AAGG, proyectos similares realizados, preparación de los entornos como visual studio, repositorio de git y búsqueda de librerías para utilizar.

Durante el verano le dediqué poco tiempo diario al proyecto pero una vez terminadas mis prácticas externas en octubre, dado que durante este cuatrimestre no contaba con clases de la universidad, me dediqué exclusivamente al desarrollo del proyecto. Debido al tiempo con el que contaba y mi situación, me planteé el TFG como si fuese una jornada laboral, para así dedicarle entre 5-8 horas diarias y poder cumplir con los objetivos planteados y aproximadamente las horas que se estipulan para un TFG.

Vi claro dividir el proyecto en 3 etapas. Primero todo lo relacionado con la representación de la información como son la implementación del motor gráfico y el *head-Up Display (HUD)*. En segundo lugar el apartado de las físicas que sería necesario para el movimiento e interacción del personaje. Y por último, la parte del desarrollo del AG.

La implementación de motor gráfico y HUD llevó alrededor de 7 días. Esto es debido a que

el motor gráfico ya fue desarrollado y probado por mí mismo durante otro proyecto realizado el año pasado. Por esto, el trabajo de esta etapa se basó en adaptarlo mínimamente a mis necesidades y realizar una estructura básica en el código para visualizar las mallas que quería.

El apartado de las físicas llevó aproximadamente 21 días debido a que es una parte más compleja y de vital importancia en el proyecto. Fue una etapa algo complicada debido a que el campo de la física es muy complejo y se tuvo que adaptar una solución aproximada a la realidad en un periodo corto de tiempo.

Por último, la parte del AG duró aproximadamente dos semanas (14 días). A pesar de ser la parte más importante del proyecto, es una parte que ya había investigado muchísimo en verano y realizarla no me requirió mucho tiempo ni esfuerzo.

A pesar de que tanto la memoria como el arreglo de errores se ha ido haciendo mínimamente en paralelo a estas etapas, le he querido dedicar a tiempo completo este último mes de desarrollo a iterar sobre la memoria final y los posibles errores que surgieron durante el desarrollo.

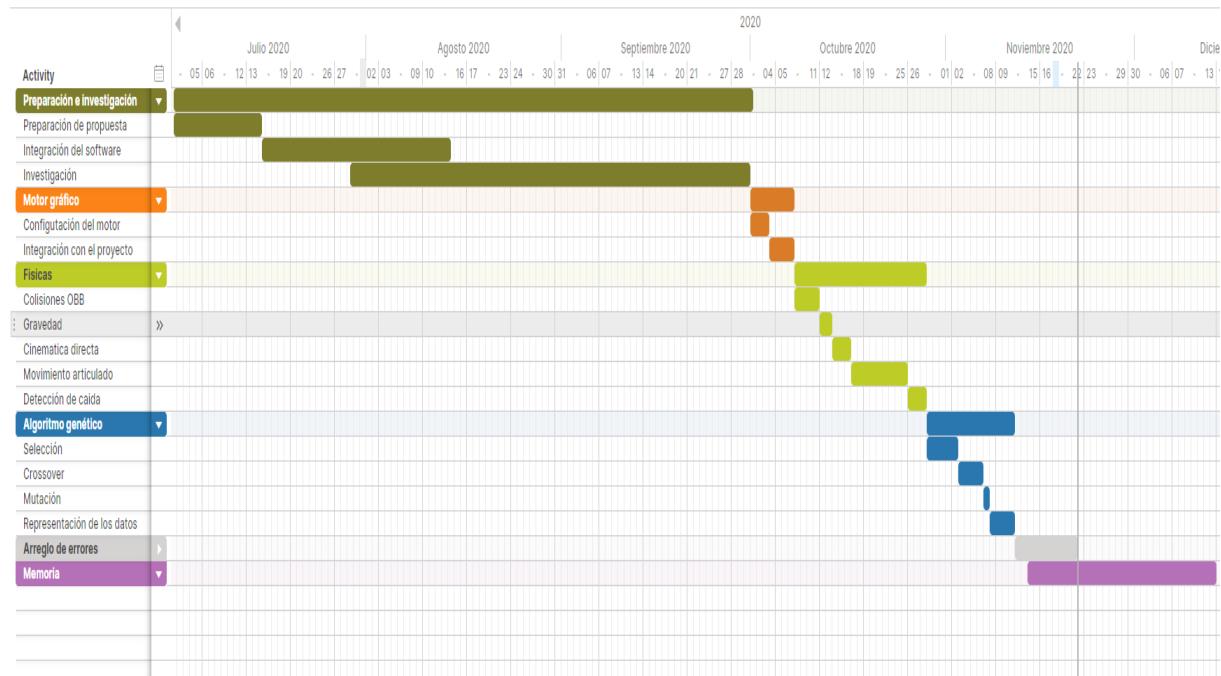


Figura 2.1: Diagrama de Gantt del proyecto

3 Estado del arte

En este apartado se explicará el término *algoritmo genético* y se mencionarán trabajos previos similares al realizado en este proyecto y en qué cosas se ha buscado diferenciar para que el proyecto cuente con una utilidad única. También se hablará del apartado gráfico haciendo referencia al motor gráfico creado y utilizado para este proyecto y algunas de las alternativas existentes en el mercado.

3.1 La inteligencia artificial

A pesar de que la inteligencia artificial ha experimentado una creciente popularidad en estos últimos años, fue en 1956 cuando John McCarthy definió por primera vez este término. Él lo definió como "*La ciencia e ingenio de hacer máquinas inteligentes*".

3.1.1 ¿Qué es la inteligencia artificial?

Utilizando la cita mencionada anteriormente de John McCarthy, podríamos decir que la inteligencia artificial busca conseguir que las maquinas aprendan como si de un ser humano como nosotros se tratara.

Esto es una tarea muy muy compleja ya que el conocimiento que tenemos sobre nuestro cerebro y por tanto de muchos de nuestros sentidos y acciones es muy bajo, por lo tanto, enseñar esas acciones naturales a una máquina se complica mucho. Para solucionar este problema existe gran cantidad de técnicas de IA al igual que muchas vertientes dentro del propio campo que dan mejores resultados en función del objetivo a conseguir. Algunos de los campos más importantes y utilizados son los siguientes:

- **Machine learning (ML):** Por su traducción al castellano (aprendizaje automático) nos deja claro que su objetivo principal es enseñar a la máquina. Se entiende por machine

learning cuando la máquina consigue aprender de manera autónoma, es decir, por sí misma.

El machine learning es un campo muy importante dentro de la IA y debido a ello las técnicas de ML se suelen agrupar en función del tipo de entrenamiento aplicado, por ejemplo:

- **Supervisado:** Aprenden funciones mediante la asociación de valores de entrada a valores de salida. Cuentan con una serie de ejemplos predefinidos para poder deducir la salida resultante de un valor desconocido. Un claro ejemplo son las *redes neuronales artificiales*.

Las redes neuronales son, a su vez, gran parte de la fundamentación de los algoritmos de Deep learning (DL), se las podría considerar los algoritmos padres. Son algoritmos inspirados en la estructura de las neuronas y conexiones del cerebro humano. Su objetivo es introducir datos a cada neurona y mediante una serie de algoritmos y funciones conseguir una salida. Esto se puede escalar para conectar varias neuronas y formar la red neuronal. En la Figura 3.1 podemos ver una comparativa entre las redes neuronales simples y las redes neuronales utilizadas en el DL.

- **No supervisado:** Por otro lado, el entrenamiento no supervisado se trata de lo opuesto. No asociar valores de entrada a valores de salida para no limitar las posibles soluciones. Un campo muy importante dentro de este tipo de entrenamiento es el DL.

El DL se define como un conjunto de algoritmos dentro del machine learning cuyo objetivo principal es trabajar con grandes cantidades de datos e información. Debido a la gran cantidad de datos a procesar se aprovechan muy bien de las velocidades y paralelismo de las unidades de procesamiento gráfico (GPUs). Es muy utilizado en los trabajos con datos de tipo: audio, vídeo o texto donde la cantidad de información es muy grande.

- **Por refuerzo:** Este tipo de entrenamiento esta inspirado en la *psicología conductista*, es decir, conseguir que el agente consiga llegar a una solución mediante unas
-

recompensas que incentiven el buen camino.

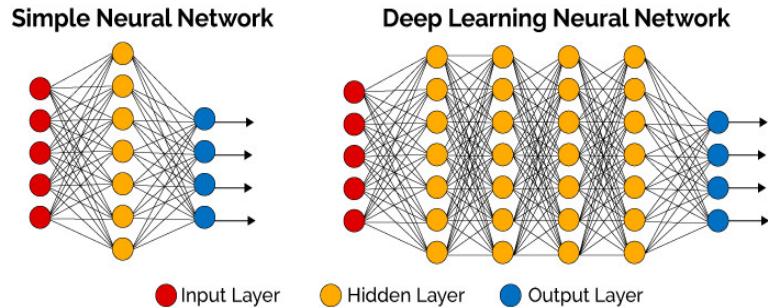


Figura 3.1: Comparativa de una red neuronal simple y una red neuronal utilizada en el deep learning.
Fuente <https://thedataScientist.com/what-deep-learning-is-and-isnt/>

- **Sistemas expertos difusos:** La lógica difusa permite representar el conocimiento de forma más natural a diferencia de la lógica tradicional que presenta dificultad para representar el conocimiento real. Por ejemplo: si sabemos que una temperatura menor a 0° es "frío" y mayor es "calor", la lógica difusa produce el resultado "calor" o "frío" al valor de temperatura que le introduzcas, pero esto también lo puede deducir una lógica booleana tradicional. Gracias a la lógica difusa permite a una máquina entender términos humanos mas complejos como "mucho calor", "poco calor", "calor extremo", "frío extremo", etcétera mediante unos valores relativos definidos previamente. En la Figura 3.2 vemos una comparativa de la lógica tradicional con la lógica difusa.

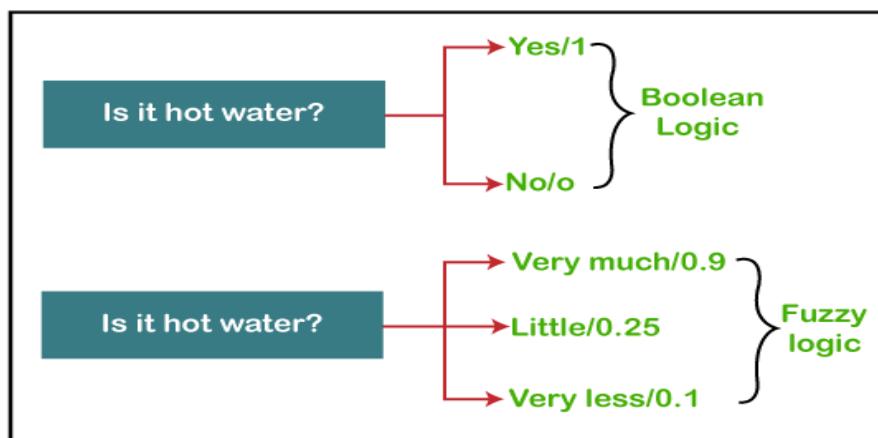


Figura 3.2: Lógica tradicional vs lógica difusa. Fuente <https://www.javatpoint.com/fuzzy-logic>

- **Redes bayesianas (Sucar, 2015):** Las redes bayesianas son grafos acíclicos dirigidos que definen una relación donde los nodos son variables aleatorias y las aristas relaciones de dependencia entre ellas. Este grafo puede contar con una ponderación que equivale a una matriz de probabilidades para conseguir un resultado real. En la Figura 3.3 observamos una red bayesiana con su matriz de probabilidad para determinar por qué la hierba esta húmeda.

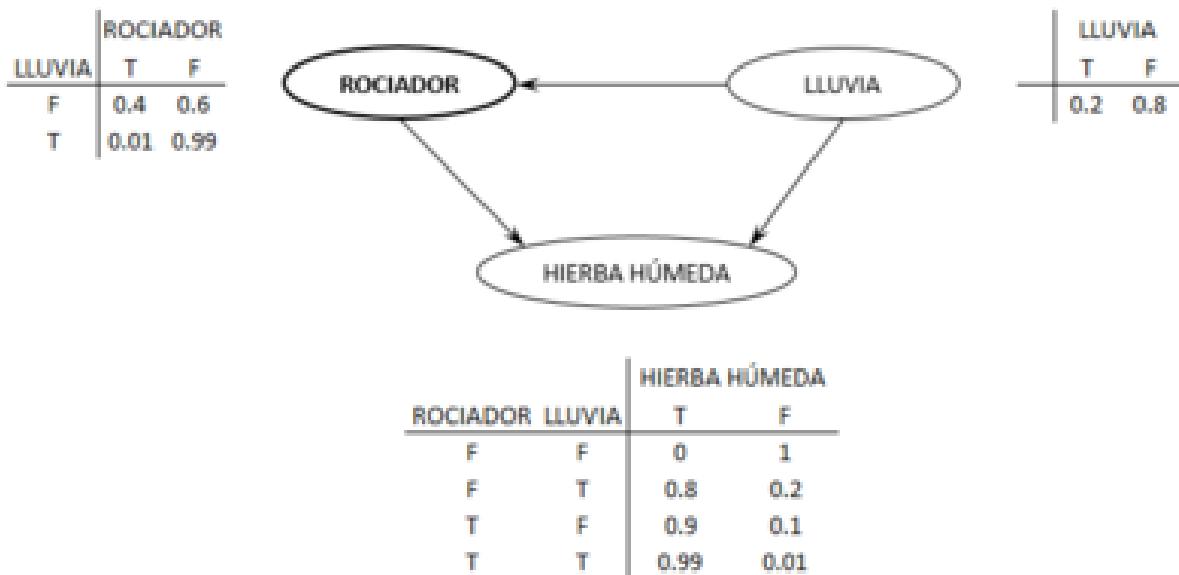


Figura 3.3: Ejemplo de una red bayesiana con ponderación de aristas. Fuente https://es.wikipedia.org/wiki/Red_bayesiana

- **Procesamiento de lenguaje natural (Collobert y cols., 2011):** Este campo de la IA estudia las interacciones persona-máquina con el objetivo de conseguir que una máquina pueda tanto entender como comunicarse con un ser humano. Uno de los últimos avances en este campo se trata de *GPT-3* desarrollado por OpenAI, el cual se trata de una IA capaz de generar textos muy similares o casi idénticos a como lo haría una persona. Uno de los grandes objetivos de este campo es conseguir batir el *test de Turing*¹.

¹Prueba que hasta día de hoy permite diferenciar a una persona humana de un ordenador

3.1.2 ¿Para qué se utiliza inteligencia artificial en la actualidad?

Como ya hemos comentado la inteligencia artificial se originó en los años 50 pero no ha sido hasta esta última década cuando ha comenzado a utilizarse de manera masiva en muchísimos campos. Esto es debido principalmente a la mejora de rendimiento de cómputo de muchos dispositivos, ya que la IA como hemos mencionado requiere mucho tiempo para evaluarse a sí misma y evolucionar para poder perfeccionarse. Este problema se acorta en el tiempo cuando contamos con mayor velocidad para realizar esos cálculos.

Por otro lado, hay muchos campos en la actualidad que sin la inteligencia artificial serían completamente inviables o muy peligrosos para el ser humano. ¿Podríamos imaginar un programa desarrollado por un programador que te tuviera que operar de corazón o que tuviera que conducir tu coche por ti? Posiblemente te negarías en rotundo ya que como seres humanos que somos tenemos nuestros errores. A diferencia de nosotros, una máquina que ha sido entrenada en un entorno seguro y ha conseguido perfeccionar y maximizar el resultado correcto para el objetivo que le planteamos inicialmente no presenta dichos errores. Con esto en mente queda claro que la ventaja de la IA es su capacidad de entrenamiento ya que tendríamos los mismos problemas si dejamos a una IA sin entrenamiento previo realizar acciones como las mencionadas anteriormente.

Con los anteriores ejemplos en mente podemos intuir que uno de los principales casos para los que se utiliza inteligencia artificial es cuando buscamos que una máquina pueda ver lo que nosotros vemos. Este es un campo ampliamente conocido y se denomina *visión por computador*. En la Figura 3.4 podemos observar un claro ejemplo de clasificación y segmentación de una imagen mediante técnicas de ML.

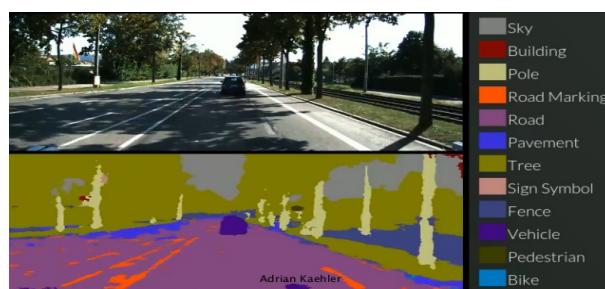


Figura 3.4: Ejemplo de segmentación que realiza la inteligencia artificial para poder identificar cada elemento que los seres humanos vemos con nuestra vista. Fuente <https://www.slideshare.net/BillLiu31/computer-vision-for-autonomous-driving>

Pero por supuesto la IA va mas allá de que un ordenador actúe como nosotros. También es utilizada para el aprendizaje de tareas inverosímiles o muy complejas para un ser humano, como por ejemplo: predicciones en bolsa (Santos, 2020), tareas a gran escala realizadas por robots, traducción de textos y un largo etcétera...

3.2 ¿Qué es un algoritmo genético?

Los algoritmos genéticos son una rama muy importante de la inteligencia artificial que surgieron en los años 70 de la mano de John Henry Holland. Este tipo de algoritmos se clasifican dentro de los algoritmos evolutivos, cuyo objetivo es la optimización de un problema planteado mediante los principios de evolución biológica.

El algoritmo genético comienza con una población inicial formada por individuos conformados por una serie de parámetros (genotipo) que van evolucionando a lo largo de generaciones (periodos de tiempo). El sistema por el cual estos algoritmos evolucionan y consiguen maximizar o minimizar una solución para optimizarla es simple y se basa en las siguientes etapas:

- 1. Generación de la población:** Inicialmente se debe generar una población que servirá para ir trabajando y evolucionando a lo largo de las siguientes etapas. Esto generalmente se hace de manera aleatoria para así no condicionar de ninguna manera la evolución que puede experimentar la población. Cabe destacar que no hay que confundir la generación aleatoria con la acotación de posibles valores que el desarrollador del algoritmo quiera darle, es decir, si el desarrollador tiene conocimientos previos de valores que harán mejorar a la IA durante el proceso puede hacerlo, así como descartar valores críticos que pueden hacer que algunos genes de la población nazcan con características nulas ya que serán descartados tarde o temprano en la evolución genética. El tamaño de la población suele mantenerse desde el inicio hasta el final y la elección del número de individuos puede afectar en el resultado final, por ello es muy importante en esta etapa estudiar distintos valores y comprobar sus resultados a lo largo del tiempo.
- 2. Selección:** Al terminar el ciclo de vida de cada generación se llega a la fase de selección, donde se deben escoger qué individuos pasan a la siguiente fase y qué individuos "mueren" para dejar espacio a nuevos individuos para generarlos. Esta etapa se suele

realizar con las denominadas *funciones de selección* y estos son algunos de los tipos existentes (Bustos y Vázquez, 2000):

- **Elitistas:** Mecanismos que garantizan que los individuos más aptos continúan a la siguiente generación, varias funciones de selección difieren en la probabilidad que se le asignan a cada individuo a la hora de ser seleccionados.
- **Ruleta:** Se les asigna una probabilidad a cada individuo siguiendo una fórmula concreta que se basa en dotar de gran probabilidad a los más aptos y de casi probabilidad nula a los menos aptos. Podemos ver un ejemplo en la Figura 3.5.

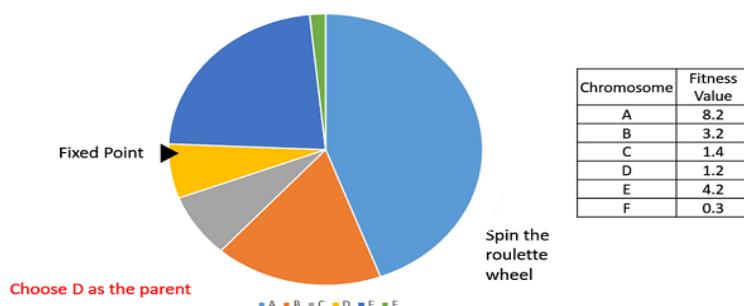


Figura 3.5: Imagen de función de selección por ruleta. Fuente: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm

- **Torneo:** Se seleccionan n individuos de la población y el más apto será el seleccionado. Si contamos con torneos con muchos individuos se perderá gran parte de la diversidad pero se garantizará mayor probabilidad a los mejores individuos.
3. **Crossover o cruce:** Es la etapa más importante del flujo del algoritmo ya que en ella es donde se escogerá a los "padres" que han sobrevivido a la generación para poder generar nuevos individuos para sustituir a los que no pudieron pasarlo. Al igual que en la etapa de selección, aquí también existen muchas estrategias para juntar dos individuos, algunos favoreciendo más el elitismo y otros no. De las elecciones de estas técnicas de cruce dependerá mucho cómo puede converger el resultado final.
 4. **Mutación:** Esta es la etapa final y se centra en alterar alguno de los genes de la población. El porcentaje de mutación suele ser un valor muy bajo (0,1% - 1%) pero por

supuesto es un valor a tener en cuenta y que puede influir mucho en el resultado final.

Sirve para egresar genes perdidos y favorecer a la diversidad genética.

Es importante tener en cuenta que lo que define si un individuo es mejor que otro es lo que se denomina como *función de fitness*, que no es más que una manera numérica de representar la calidad del individuo. El cálculo del fitness del individuo puede realizarse en cualquier momento de la ejecución; al generar los individuos, durante la ejecución (principalmente para estudiar anomalías) o al final de la ejecución antes de comenzar las etapas mencionadas. En la Figura 3.6 presentamos un diagrama de flujo con las etapas que sigue un AG.

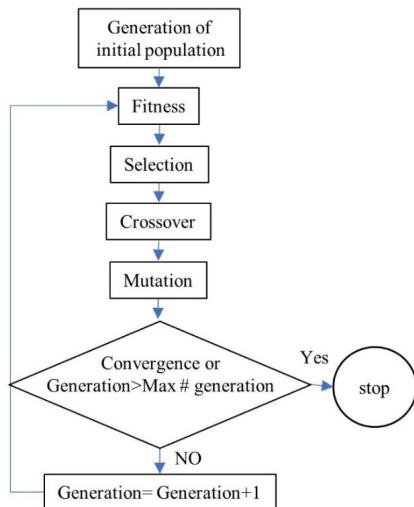


Figura 3.6: Diagrama de flujo de un algoritmo genético. Fuente: https://www.scielo.br/scielo.php?script=sci_arttext&pid=S1679-78252018000300508&lng=en&nrm=iso&tlang=en

3.3 Aplicaciones de los algoritmos genéticos.

Aunque pueda parecer que los algoritmos genéticos son un campo muy científico y de investigación, su uso está muy extendido en el desarrollo software sobretodo en aquellos casos en los que se busque optimizar o llegar a un resultado final al cual el desarrollador no puede llegar mediante un método analítico. A continuación, voy a hablar de algunos proyectos relacionados con el que se ha desarrollado.

3.3.1 BipedalWalker-v2

El proyecto de *BipedalWalker* desarrollado por *OpenAI* consiste en utilizar AG para optimizar los parámetros necesarios para que un ser bípedo ande de manera correcta. En este experimento se toman en consideración algunos de los parámetros que también se han tomado en este proyecto como puede ser las rotaciones y velocidades de las piernas, entre otras. En la Figura 3.7 se aprecia el tipo de individuo seleccionado para realizar esta acción.



Figura 3.7: Imagen del proyecto BipedalWalk. Fuente: <https://gym.openai.com/envs/BipedalWalker-v2/>

3.3.2 Flexible Muscle-Based Locomotion for Bipedal Creatures

Este proyecto (Geijtenbeek y cols., 2013) es una idea algo más avanzada ya que estudia y toma en consideración en mayor profundidad las físicas del movimiento. También consigue soluciones para diferentes tipos de individuos bípedos y con velocidades variables. Este es sin ninguna duda un claro de ejemplo de la escalabilidad de estos algoritmos en función de lo que busquemos, y que se puede adaptar tanto a proyectos más simples como el mencionado anteriormente o a proyectos avanzados como este donde podemos apreciar prácticamente una dinámica de movimientos perfecta y natural. En la Figura 3.8 observamos el mejor individuo de alguna de las generaciones del AG.

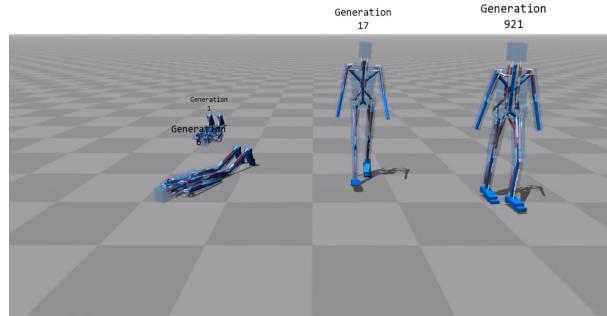


Figura 3.8: Imagen del proyecto Muscle Based Locomotion for Bipedal Creatures. Fuente: <https://www.youtube.com/watch?v=pgaEE27nsQw>

3.3.3 Slime Volleyball

El proyecto de *Slime Volleyball* (Ha, 2020) tiene como objetivo que dos *slimes*² consigan mantener en el aire un balón por el mayor tiempo posible. En este caso cada agente slime compite contra el otro, conseguir un punto restará un punto al slime enemigo hasta perder las 5 vidas disponibles. Es una buena manera de entrenar a una IA mediante competición con otra IA para que gane la mejor de ellas, en lugar de competir por un objetivo final. En la Figura 3.9 observamos una captura de una partida del videojuego.

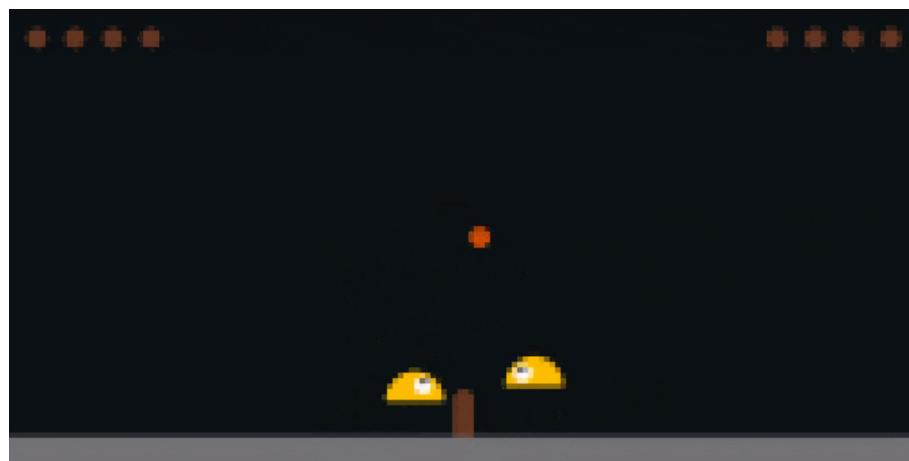


Figura 3.9: Imagen del proyecto Slime Volleyball. Fuente: <https://github.com/hardmaru/slimevolleygym/>

²Un *slime* es un tipo de enemigo conocido en videojuegos con aspecto y movimiento en forma de chicle pegajoso.

3.3.4 CarRacing-v0

El proyecto de *CarRacing-v0* desarrollado por *OpenAI* busca conseguir que un coche de F1 consiga recorrer un circuito sin salirse del trazado marcado. En este caso trabaja a nivel de píxeles para poder saber por qué píxeles del trazado ha pasado y por cuales no y también tiene en cuenta el número de frames que el individuo tarde en terminar el circuito. En la Figura 3.10 vemos una imagen del proyecto.



Figura 3.10: Imagen del proyecto Car Racing. Fuente: <https://gym.openai.com/envs/CarRacing-v0/>

3.4 Motores gráficos

En este proyecto era muy importante el apartado de representación gráfica de todos los datos ya que se busca no solo un resultado final si no un entendimiento completo de todo el desarrollo de la ejecución. Por ello, es importante hablar sobre las opciones de motores gráficos o de videojuegos que estaban disponibles y por qué finalmente se llegó a la conclusión de utilizar un motor propio.

3.4.1 Unity

Unity es un motor de videojuegos desarrollado por *Unity technologies* lanzado en 2005. El motor mantiene una evolución y actualización constante para adaptarse a las necesidades y hacia donde se mueve la industria, por ello recibe unas tres actualizaciones al año.

Unity está principalmente destinado al mundo 3D, pero debido a grandes competencias directas supo adaptarse muy bien al desarrollo 2D sobretodo orientado a dispositivos móviles.

Gracias a esta versatilidad, Unity no solo es utilizado en el desarrollo de videojuegos, sino que también es ampliamente utilizado en el desarrollo de investigaciones o demos visuales.

Como motor gráfico tiene 3 opciones principales; OpenGL para poder usar en Windows, Mac y Linux, Direct3D disponible solo en Windows y OpenGL ES para Android e IOS. Además cuenta con implementaciones de interfaces propietarias como podría ser el caso de la Nintendo Wii.

Gracias al soporte integrado para Nvidia puede hacer uso del motor de físicas *PhysX*. Este motor de físicas proporciona una aceleración por hardware cuando es integrada en las tarjetas gráficas. Gracias a esto y a su característica de *middleware* es capaz de proporcionarnos una abstracción para poder utilizarlo de manera rápida y generar unas simulaciones ultra realistas.

El paradigma de programación empleado en los desarrollos con Unity es el denominado entidad-componente-sistema (ECS). Este paradigma es ampliamente utilizado en el desarrollo de videojuegos. El paradigma se basa en el lema *"composición antes que herencia"* lo que nos brinda una mayor flexibilidad a la hora de poder tratar todos los objetos del juego como entidades. El paradigma es muy simple: Los objetos del entorno se denominan entidades y están compuestas por componentes (velocidad, posición, textura, etc...) y a su vez tenemos sistemas que son los encargados de realizar las operaciones lógicas sobre dichas entidades. En la Figura 3.11 podemos ver un esquema de una implementación con el paradigma ECS.

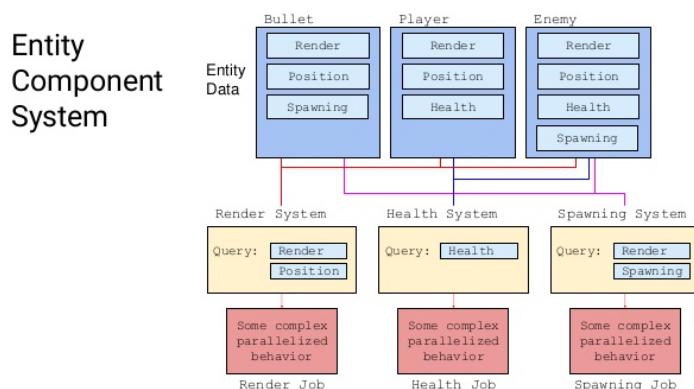


Figura 3.11: Diagrama de un programa con paradigma entidad-componente-sistema. Fuente: <https://pt.slideshare.net/unity3d/scaling-cpu-experiences-an-introduction-to-the-entity-component-system/>

Unity bien podría haber sido una gran elección para ahorrar algo de tiempo en el proyecto,

debido a su usabilidad orientada a usuarios menos expertos. Pero al ser una aplicación centrada en la inteligencia artificial buscábamos mucha velocidad y optimización mas allá de la representación de los gráficos. Es en este punto donde C++ es muy superior a C#, en operaciones muy recurrentes en el proyecto y en general en la IA como puede ser la generación de números aleatorios, ordenación de vectores, comparación de objetos, operaciones aritméticas y un largo etcétera. En la Figura 3.12 podemos observar una comparativa de ambos lenguajes.

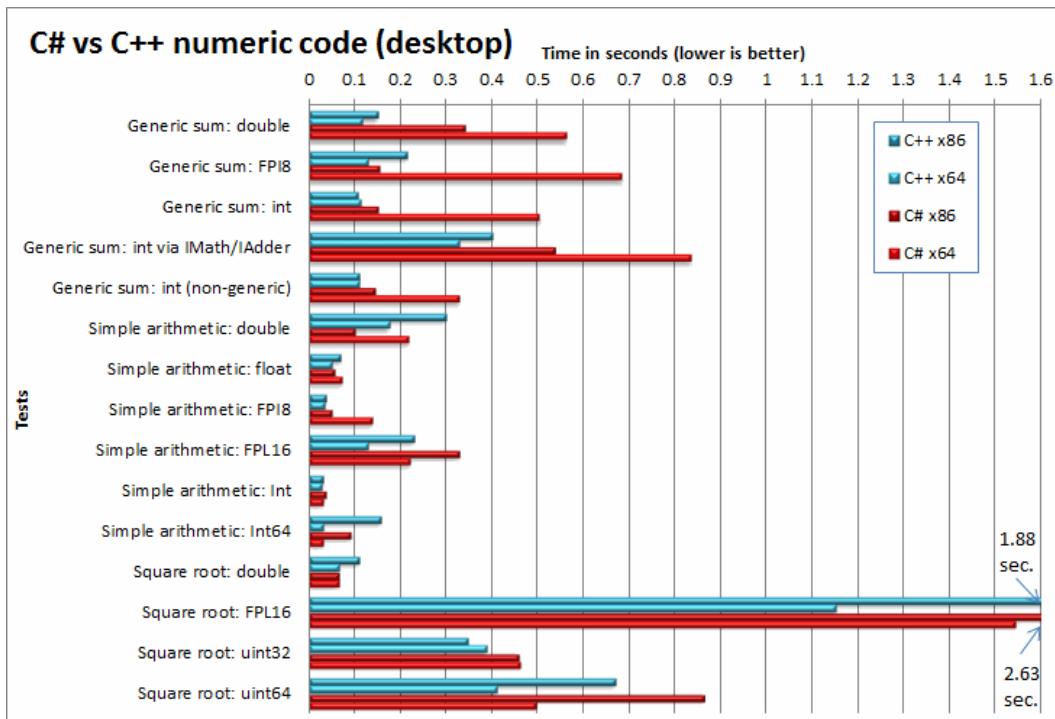


Figura 3.12: Comparativa de velocidades entre C++ y C#. Fuente: <http://www.cyqdata.com/cnblogs/article-detail-40248>

3.4.2 Unreal Engine

Unreal Engine es un motor de videojuegos desarrollado por *Epic Games* y utilizado por primera vez en 1998 en el videojuego *Unreal*. Históricamente Unreal Engine ha sido el enemigo directo de Unity. A pesar de que ambos buscan un mercado diferente, ya que Unreal Engine se centra más en el realismo y Unity más en la estética y jugabilidad cartoon, es inevitable comparar sus resultados.

Unreal Engine sí utiliza C++ como lenguaje de programación, con lo cual como hemos

remarcado en el anterior punto nos brinda una mayor velocidad y optimización para las operaciones que busquemos realizar. Sin embargo, sigue presentando el problema que Unity de cara a este proyecto, hay que aprender su propia tecnología y sintaxis para poder hacer uso del motor, así como cargar por defecto miles de funcionalidades avanzadas de las que no vamos a hacer uso en este proyecto. A continuación en la Figura 3.13 podemos ver un estudio de mercado sobre los motores gráficos utilizados para videojuegos actualmente.

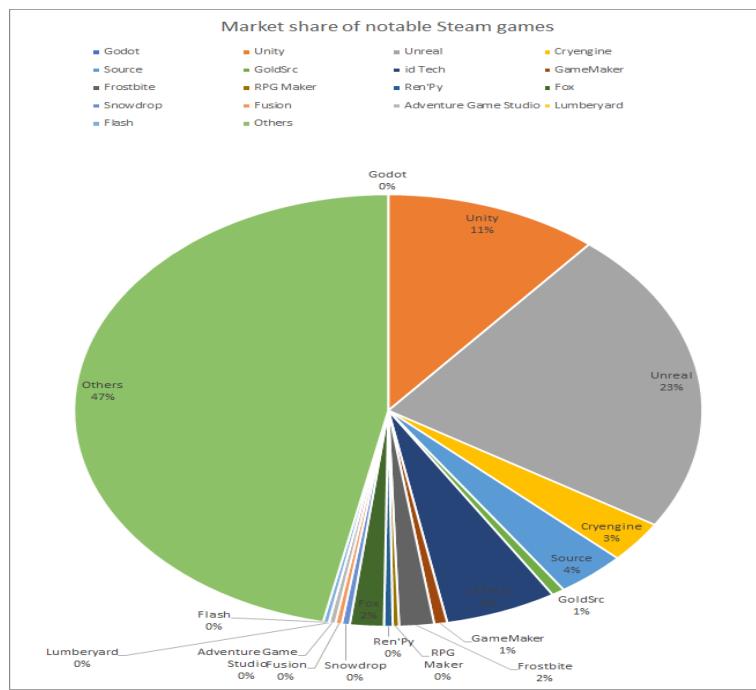


Figura 3.13: Market share en función del motor de videojuegos utilizado para la implementación.
Fuente: https://www.reddit.com/r/gamedev/comments/8s20qp/i_researched_the_market_share_of_game_engines_on/

3.4.3 Spark Engine

Spark Engine es el motor gráfico utilizado para este proyecto. Fue desarrollado por mí junto a un grupo de estudiantes durante el 4º curso de ingeniería multimedia por el equipo llamado Clover Games Studios.

La ventaja de este motor con respecto a los otros mencionados es su sencillez ya que solo incorpora un motor gráfico simple con las funcionalidades necesarias para este proyecto. Con esto conseguimos no sobrecargar con funciones innecesarias que no vamos a utilizar.

Por supuesto, el hecho de haber sido desarrollado por mí me permite modificar cualquier característica que necesite para el desarrollo, a diferencia de Unity o Unreal Engine que son de código privado.

Su carácter abierto me ha sido de gran utilidad para poder incorporar tecnologías externas para su uso en el proyecto y de las cuales se hablará en otros apartados de esta memoria.

A continuación en la Figura 3.14 podemos ver un videojuego desarrollado con el motor de Spark Engine.



Figura 3.14: Imagen propia del videojuego *Beast Brawl* realizado con Spark Engine.

4 Objetivos

El objetivo principal de este proyecto es conseguir que una inteligencia artificial bípeda aprendiera a andar mediante un algoritmo genético evolutivo. A parte del objetivo principal también se plantearon una serie de subobjetivos:

- **Lograr un visualización 3D en tiempo real:** Para que el proyecto fuera entendible de manera más sencilla, se planteó una representación 3D que evolucionara durante el tiempo de ejecución. Este punto es importante ya que muchas veces una simple extracción de datos puede volverse demasiado compleja y poco accesible para algunas personas, por el contrario, una representación en tiempo real de como está evolucionando y aprendiendo le da un carácter de entretenimiento mientras comprendes el proyecto.
- **Maximizar la eficiencia:** Conseguir el resultado más eficiente en relación al problema planteado, es decir, desde el punto de vista fisiológico cual es la mejor manera de andar teniendo en cuenta las condiciones planteadas.
- **Conseguir un carácter educativo:** Una ejecución parametrizada y una interfaz donde mostrar los datos de la evolución en tiempo real era de las tareas más importantes para así conseguir que este proyecto también tuviera un carácter educativo para aquellas personas que buscasen una herramienta para poder entender el funcionamiento y aplicaciones de un algoritmo genético evolutivo.

5 Metodología

Para el desarrollo del proyecto se ha seguido una metodología de tipo Scrum para poder definir unos objetivos claros a corto plazo. Se han realizado reuniones periódicas cada dos semanas con los tutores del proyecto para validar el trabajo realizado y comunicar las tareas a realizar en el próximo sprint del proyecto.

5.1 Herramientas de gestión de proyecto

La herramienta mas importante a la hora de gestionar los tiempos de las iteraciones de este proyecto ha sido Trello, en él dividía las tareas en tres columnas:

- **Por hacer:** Todas las tareas a realizar del proyecto. Se buscó tener una manera visual de poder ver lo qué faltaba por hacer en todo momento, a pesar de ello si había que añadir tareas a mitad del proyecto se podía hacer, pero sí se trató de no eliminar ninguna de las colocadas inicialmente.
- **En proceso:** En esta columna se agrupaban las tareas que se estaba realizando actualmente para esta iteración. Si al terminar la iteración alguna quedaba pendiente se ponía al inicio y sería la primera en realizar en la siguiente iteración.
- **Terminadas:** Columna donde se movían todas las tareas que ya se habían realizado.

5.2 Tecnologías

5.2.1 Gestión de versiones

Para la gestión y guardado de las versiones de nuestro código se ha utilizado Git. A su vez como interfaz gráfica para hacer mas sencillo este trabajo he utilizado GitKraken.

Se ha escogido GitKraken principalmente por su usabilidad y rendimiento a la hora de trabajar con él. Posee una interfaz muy clara y simple de entender y es muy útil si no puedes recordar en todo momento todos los comandos de Git. A su vez a pesar de trabajar solo en este proyecto, su función de unión de ramas es muy clara y fácil de usar y ha ahorrado muchísimo tiempo durante el desarrollo. En la Figura 5.1 podemos observar su interfaz y los *commits* del proyecto.

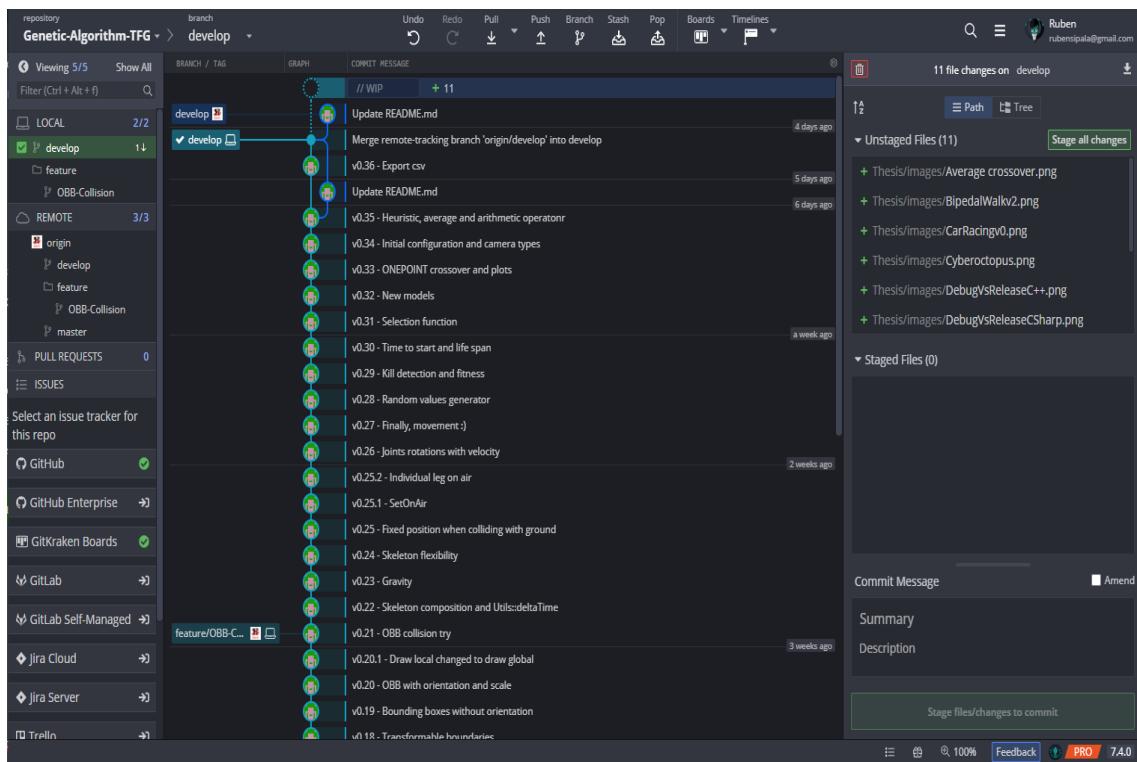


Figura 5.1: Interfaz de Gitkraken.

5.2.2 Visual studio 2019

El entorno de desarrollo integrado (IDE) escogido para desarrollar el proyecto ha sido visual studio 2019, desarrollado por *Microsoft*. Se ha escogido este IDE debido a que facilita mucho trabajar con C++ en Windows y aporta infinidad de utilidades para facilitar las tareas a la hora de programar y preparar todo lo necesario para el proyecto.

Visual Studio facilita y agiliza principalmente el trabajo de enlazar librerías de C++ de las que vas a hacer uso. Sin el uso de un IDE esta tarea se vuelve muy pesada y a veces puedes

perder muchísimo tiempo solo intentando preparar el entorno para programar. Gracias a Visual Studio evitamos esto y es posible centrarse en lo realmente importante del proyecto, programar.

Y por otro lado pero no menos importante, mencionar sus herramientas de debug de código que también nos ayudan a poder depurar el proyecto de una manera sencilla y efectiva.

5.2.3 Motor gráfico

Como ya se ha mencionado, para este proyecto se ha realizado un motor gráfico propio y se va a mencionar qué librerías que se han utilizado para lograr una pequeña capa de abstracción y facilitar un poco el trabajo.

- **OpenGL:** Se trata de una librería que define una API para poder crear gráficos 2D y 3D principalmente mediante el uso de la tarjeta gráfica del ordenador. Se trata de una librería de código abierto desarrollada por *Khronos Group*. Actualmente se encuentra en la versión 4.6 pero para el proyecto se ha utilizado una versión anterior (versión 4.5).
- **GLEW:** Librería multiplataforma responsable de la carga y consulta de extensiones OpenGL. Consigue en tiempo de ejecución determinar qué extensiones de OpenGL son soportadas.
- **GLFW:** Librería multiplataforma para la creación y manejo de ventanas, contextos, inputs, etcétera...
- **SOIL2:** Librería para la lectura de imágenes de disco basada en la librería *STB* y cuyo objetivo es extender alguna de las funcionalidades de la librería original y orientarla para mejorar el rendimiento y agilizar el trabajo al utilizarla con OpenGL.
- **ASSIMP:** Librería para la lectura y gestión de modelos 3D. Nos permite leer hasta 57 formatos distintos y cuenta con muchas funcionalidades internas como la optimización de mallas, la triangulación, el calculo de tangentes, etcétera...
- **GLM:** Librería matemática de gran utilidad para los gráficos 3D que proporciona gran cantidad de funcionalidades. Sigue el estandar especificado de OpenGL Shading Language (GLSL).

- **Dear ImGui:** Librería utilizada para la interfaz del proyecto, muy útil para el debug visual durante la ejecución en tiempo real. En la Figura 5.2 se puede ver un ejemplo de lo que es capaz esta librería y podemos observar una demo de algunas de las funcionalidades que incorpora.
- **ImPlot:** Extensión de *Dear ImGui* especializada para el uso de gráficas en tiempo real.



Figura 5.2: Ejemplo de la librería de Dear ImGui. Fuente: <https://github.com/ocornut/imgui>

5.2.4 Algoritmo genético

Para el resto del proyecto excluyendo el motor gráfico y el HUD también se ha hecho uso de algunas librerías que vale la pena destacar:

- **NUMCPP:** NumCpp es una versión de la famosa librería *numpy* de Python. Contiene muchas facilidades para el trabajo y desarrollo matemático.
- **Random for modern C++ with convenient API:** Librería que facilita mucho el trabajo de la generación de valores aleatorios en C++.

5.3 Hardware

Al tratarse de un proyecto basado en gráficos e IA, es importante recalcar sobre que especificaciones se han realizado todas las pruebas y ejemplos. Aunque más adelante en los apartados de estudios y conclusiones se mencionará y tendrá en cuenta esto, no viene mal dejar claro las capacidades y especificaciones del dispositivo con el que se ha realizado este trabajo:

- **CPU:** Intel Core i5-7600k @ 3.80GHz.
- **GPU:** ASUS Radeon 1060 6 GB.
- **RAM:** 16 GB.
- **Disco duro:** 1 TB HDD.

6 Implementación

En este capítulo se va a hablar de cómo ha sido la implementación del proyecto del que trata este documento. Se va a dividir en las secciones mencionadas anteriormente: motor gráfico, físicas y algoritmo genético. Este es el mismo orden que se siguió a la hora del desarrollo con lo cual se busca dejar reflejado no solo los detalles de la implementación de cada parte sino también por qué se ha seguido esta división a la hora del desarrollo.

6.1 Representación visual

Como ya se ha mencionado anteriormente, para este proyecto se ha hecho uso de un motor gráfico propio. Debido a que no todas las funcionalidades disponibles en el motor gráfico han sido utilizadas en el proyecto, solamente serán mencionadas aquellas de las que sí se haga uso. Gran parte del desarrollo del motor gráfico ha sido extraído de (de Vries, 2020).

6.1.1 Elementos del motor gráfico

El motor gráfico sigue una estructura sencilla y potente en la cual divide todo lo necesario en dos grandes conceptos: en primer lugar el árbol de la escena y en segundo lugar el gestor de recursos.

6.1.1.1 Árbol de la escena

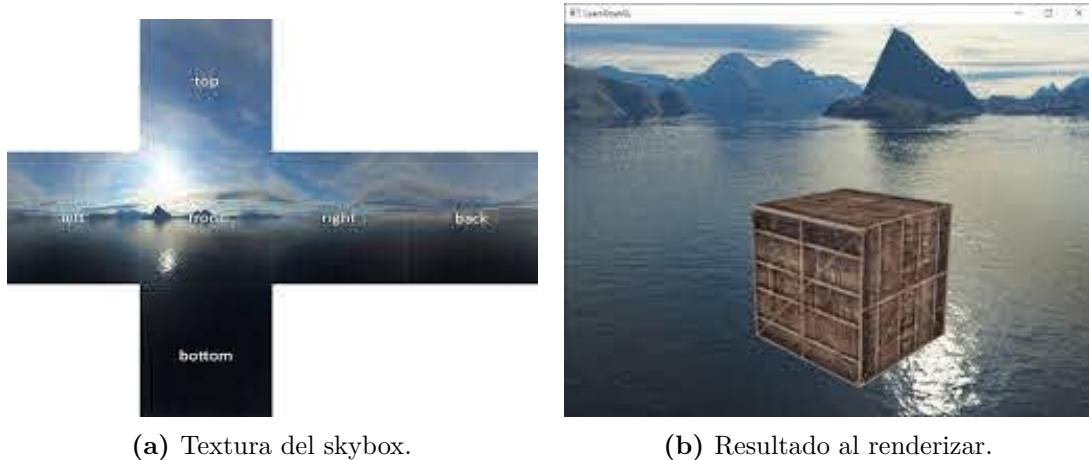
El árbol de la escena es el encargado de almacenar las entidades pertenecientes a nuestra escena en nodos. Mediante el polimorfismo conseguimos una abstracción que nos hace posible almacenar cualquier tipo de entidad que se desee en la escena. Algunas de las entidades disponibles son las siguientes:

- **CLEntity:** Se trata de la clase padre utilizada para que el resto de entidades hereden de ella. A pesar de ello es muy útil para crear nodos del árbol vacías y usarlo como agrupaciones de otras entidades. Determina alguna de las variables y funciones que deben de cumplir el resto de entidades que hereden de él como puede ser un identificador (ID) o una función para el dibujado específico de la entidad.
- **CLMesh:** Es el tipo de entidad que almacena en sí mismo un recurso de malla para ser renderizada en la escena.
- **CLCCamera:** La entidad encargada de la cámara de la escena, es decir, el punto de visión donde se va a renderizar el resto de elementos de nuestra escena. Esta entidad almacena varios datos importantes como el punto hacia donde esta mirando (*target*) o el *field of view (FOV)*. En la Tabla 6.1 se muestran las variables presentes en la clase.

Variable	Tipo de dato	Descripción
m_near	GLfloat	Mínima distancia de renderizado
m_far	GLfloat	Máxima distancia de renderizado
m_right	GLfloat	Píxeles de anchura
m_left	GLfloat	Píxel inicial de anchura
m_top	GLfloat	Píxeles de altura
m_bottom	GLfloat	Píxel inicial de altura
fov	GLfloat	Field of view
active	bool	Estado de la cámara
cameraTarget	glm::vec3	Objetivo de la cámara

Tabla 6.1: Tabla de las variables presentes en la clase CLCCamera.

- **CLSkybox:** El skybox es un recurso muy utilizado en gráficos, principalmente en videojuegos y se utiliza para simular elementos muy distantes por medio de texturas 2D en forma de cubo. OpenGL cuenta con un tipo de dato interno denominado *GL_TEXTURE_CUBE_MAP* el cual interpreta como 6 imágenes 2D que representan las caras de un cubo. En la Figura 6.1a podemos ver la textura en forma de cubo y en la Figura 6.1b el resultado final.



(a) Textura del skybox.

(b) Resultado al renderizar.

Figura 6.1: Ejemplo de un skybox. Fuente: <https://learnopengl.com/>

Todas las entidades mencionadas se almacenan en el árbol de la escena dentro de la clase **CLNode**, que se trata de cada nodo del árbol. Estos nodos cuentan con parámetros que deben contener todo lo introducido en la escena como es: posición, rotación, escalado, visibilidad, padre e hijos. Otro elemento importante con el que cuenta cada nodo es un shader para utilizar en el renderizado de OpenGL, de ello se hablará mas extendido en (Sección 6.1.1.2).

El hecho de escoger la estructura de árbol para el almacenamiento de las entidades dentro de la escena no es una decisión trivial. Gracias a este tipo de estructura concreto conseguimos aplicar trasformaciones de padres a hijos. Un claro ejemplo es el movimiento de un ser bípedo como el que se ha realizado en el proyecto, en este caso el parent central sería el tronco o tórax del cuerpo, sus nodos hijos serían extremidades como caderas u hombros, y al mismo tiempo estas caderas u hombros tendrían hijos como rodillas y codos. Gracias a esto conseguimos que moviendo solamente el tronco del cuerpo se mueva conjuntamente el resto de las partes del cuerpo unidas al mismo.

Para poder aplicar y visualizar todas las trasformaciones mencionadas anteriormente hay que entender muy bien como convierte OpenGL unos datos simples como una posición 3D, por ejemplo, (30,10,-40) a una representación 3D en la escena. Esto se consigue mediante el sistema de coordenadas de OpenGL que se divide en las siguientes etapas:

1. **Local space hacia world space:** Etapa en la que se transforman los valores locales

de las entidades en valores comunes en el mundo. En este punto se dispone solo de los parámetros mencionados anteriormente como son la posición, rotación y escalado que debe ser aplicado a los valores locales como por ejemplo un vértice. Todos estos datos son vectores de tres dimensiones XYZ que hacen referencia a nuestra entidad, independientemente de la escena donde estén. Realizando operaciones geométricas con estos tres parámetros se define una matriz denominada *matriz de transformación* que se trata de una matriz 4x4 en la cual se almacenan la posición, rotación y escalado de la entidad. Estos valores ya son tomado en cuenta el resto de entidades presentes en la escena.

Como se ha dicho, solo se almacenan tres vectores de tres dimensiones, posición, rotación y escalado. ¿Como se convierten esos tres vectores XYZ en una matriz 4x4 XYZW? Esto se consigue gracias a las matrices de traslación, rotación y escalado definidas matemáticamente como se aprecia en la Figura 6.2 donde tx, ty y tz son la posición de la entidad, sx, sy, y sz es el multiplicador de escalado y d es el ángulo en cada eje. Para rotar se tiene que multiplicar tres veces, una para cada eje tridimensional.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Identity Matrix

glTranslatef(tx,ty,tz)

glScalef(sx,sy,sz)

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(d) & -\sin(d) & 0 \\ 0 & \sin(d) & \cos(d) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} \cos(d) & 0 & \sin(d) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(d) & 0 & \cos(d) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} \cos(d) & -\sin(d) & 0 & 0 \\ \sin(d) & \cos(d) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

glRotatef(d,1,0,0)

glRotatef(d,0,1,0)

glRotatef(d,0,0,1)

Figura 6.2: Matrices de transformaciones geométricas definidas para la traslación, escalado y rotación al rededor de los tres ejes. Fuente: <http://math.hws.edu/graphicsbook/c3/s5.html>.

2. **World space hacia view space:** Etapa donde se convierten los valores de mundo a valores relativos a la visión de la cámara. En este punto se cuenta con una matriz en coordenadas de mundo, pero estos valores siguen sin ser suficientes ya que se necesita tener en cuenta una cámara que enfoque hacia dichos objetos. Para ello se emplea una matriz vista (Figura 6.3) que esta ligada directamente con la cámara de la escena. En ella tienen importancia parámetros como los ángulos de cada eje hacia donde esta mirando (el objetivo) y también su propia posición en la escena.

$$\begin{pmatrix} Right_x & Up_x & Look_x & 0 \\ Right_y & Up_y & Look_y & 0 \\ Right_z & Up_z & Look_z & 0 \\ -(Position * Right) & -(Position * Up) & -(Position * Look) & 1 \end{pmatrix}$$

Figura 6.3: Matriz vista.

3. **View space hacia clip space:** Por último es importante tener en cuenta la perspectiva en la que se renderiza la escena. La vista real de un ser humano cuenta con proyección, es decir, objetos grandes que están alejadas pueden parecer mas pequeños que objetos pequeños más cerca, a esto se le denomina *proyección perspectiva*. Por otro lado que un objeto no pierda su dimensionalidad con la distancia es lo que se le denomina *proyección ortogonal*. Es con la última de las matrices, la matriz de proyección con la cual finalmente conseguiremos que nuestra entidad sea vista como si se tratara de la vista de un ser humano.

La multiplicación ordenada de todas estas matrices mencionadas es lo que se denomina *matriz MVP* (*model, view, projection*) y usada para multiplicar cualquier valor local como el vértice de una malla, convierte ese punto 3D local en un punto con dimensionalidad en la escena y su proyección si fuera necesaria. En el ejemplo de Código 6.1 se puede observar un pseudocódigo de cómo se implementaría el flujo mencionado.

Código 6.1: Pseudocódigo del flujo de calculo de la matriz MVP utilizando la librería GLM

```
1 mat4 MVP = mat4(1.0)
2 MVP = glm::scale(MVP, Scale)
3 MVP = glm::rotate(MVP, Rotation.x,(1,0,0)) * glm::rotate(MVP, Rotation.y,(0,1,0)) * glm::rotate(MVP,
4     ↪ Rotation.z,(0,0,1))
5 MVP = glm::translate(MVP, Position)
```

```
5  
6     MVP *= ViewMatrix  
7     MVP *= ProjectionMatrix
```

6.1.1.2 Gestor de recursos

El gestor de recursos es un elemento muy importante del motor gráfico ya que es el encargado de almacenar y gestionar cualquier tipo de recurso que se cargue del disco como por ejemplo: mallas, texturas, shaders, imágenes, etcétera. Alguno de estos tipos de recursos son recursos muy costosos de cargar y conseguir una optimización en su gestión es la diferencia entre que el motor sea usable o no.

Una de las principales funciones del gestor de recursos es asegurarse que no vuelva a cargar algo que ya esta cargado en memoria. Esto se ha conseguido mediante un identificador único para cada recurso que es cargado, en este caso, el nombre del fichero junto a su extensión.

A su vez estos recursos son almacenados por algunas de las entidades comentadas anteriormente para hacer uso de ellos. Más en concreto tenemos 3 tipos de recursos para utilizar:

- **Shaders:** Los shaders son un elemento indispensable en nuestro motor gráfico. En el denominado OpenGL moderno se introdujeron los denominados shaders que se trata de scripts ejecutados en la unidad de procesamiento gráfico (GPU). Estos shaders son ejecutados cuando OpenGL recibe la orden de comenzar a pintar la pantalla y siguen un orden establecido denominado pipeline gráfico, en la Figura 6.4 se muestra un pipeline básico de OpenGL.

OpenGL permite crear multitud de shaders diferentes para ejecutar en momentos distintos del renderizado pero se ha realizado centrado en los tres más importantes: vertex, geometry y fragment.

1. **Vertex shader:** El vertex shader es el encargado de recibir los vértices de nuestras geometrías. En el caso de este motor se le envía la posición de los vértices de las mallas junto a la *matriz MVP* mencionada con anterioridad para poder calcular su posición real en la pantalla. Como se aprecia en la Figura 6.4 es el primero en ser llamado y se ejecuta cuando el método *CLEntity::Draw* es llamado a la hora

de recorrer el árbol de la escena y se invoca a alguna de las funciones de OpenGL disponibles para dibujar como por ejemplo *glDrawElements*.

2. **Geometry shader:** El geometry shader es el único de los 3 mencionados que ya vienen implementados por defecto por OpenGL. Su función es realizar reconstrucciones en los vértices que recibe del vertex shader. Por defecto OpenGL no te obliga a implementar uno personalizado pero el motor gráfico realizado sí contempla la funcionalidad de cargar uno mediante el gestor de recursos.
3. **Fragment shader:** El fragment shader trabaja a nivel de píxel, es decir, se ejecuta una vez por cada píxel contenido en la geometría que vayamos a dibujar. Es obligatorio implementarlo tanto por OpenGL como por nuestro motor gráfico.

En el (Código 6.2) se muestra un ejemplo de como leer shaders gracias al gestor de recursos:

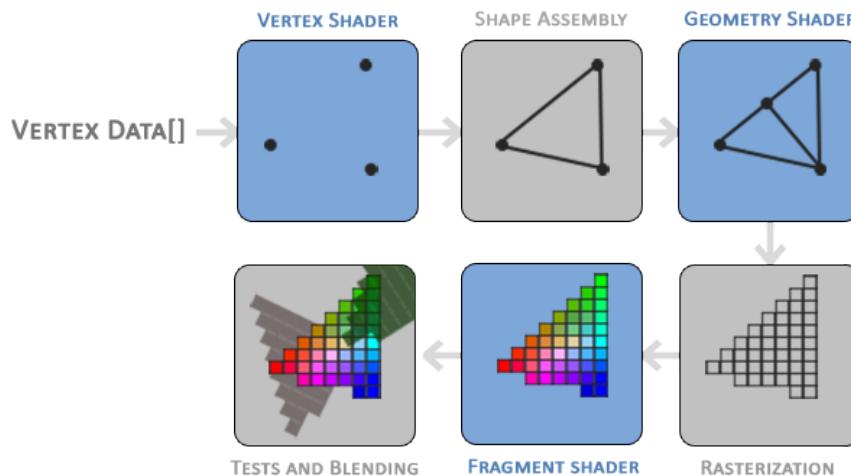


Figura 6.4: Flujo del pipeline gráfico de OpenGL. Fuente: <https://learnopengl.com/Getting-started/Hello-Triangle>.

- **Mallas:** La lectura de mallas es el otro punto importante de las funcionalidades con las que cuenta el gestor de recursos. Como se ha mencionado anteriormente se utiliza una librería llamada *ASSIMP* para realizar estos trabajos de lectura de ficheros de mallas. Gracias a ello podemos leer ficheros de diferentes formatos ¹.

¹Formatos soportados en ASSIMP: http://assimp.sourceforge.net/main_features_formats.html

Los ficheros son leídos también en forma de árbol, esto es debido a que a la hora de exportar desde el programa de modelado es común tener esta estructura de mallas hijas de otras mallas. Cada nodo puede contar con varias mallas pero lo importante viene al leer y guardar malla a malla. El proceso es simple, iteramos recorriendo todos los vértices que contenga la malla y guardamos los valores de sus posiciones, las normales del vértice y sus coordenadas de textura. Finalmente leemos los materiales y texturas asociados a la malla y lo almacenamos para su posterior uso a la hora de renderizar la malla.

- **Texturas:** Las texturas o imágenes son de gran utilidad a la hora de trabajar con gráficos ya que es una manera sencilla de ordenar y dar sentido a valores de colores. En este caso es utilizado principalmente para la lectura de texturas asociadas a las mallas mencionadas anteriormente.

Código 6.2: Pseudocódigo de ejemplo para realizar la lectura de shaders con el gestor de recursos

```

1 ResourceManager* resourceManager = new ResourceManager()
2
3 // Lectura de vertex y fragment.
4 resourceManager->GetResourceShader("shader.vert", "shader.frag")
5 // Lectura de vertex, geometry y fragment.
6 resourceManager->GetResourceShader("shader.vert", "shader.geom", "shader.frag")

```

6.1.2 Dibujado de líneas 3D

El dibujado de primitivas, en este caso una línea, es una de las opciones que nos brinda OpenGL con la función *glDrawArrays(GL_LINES, first, count)*. OpenGL cuenta con más operaciones de dibujado de primitivas como vemos en la Figura 6.5. Estos diferentes tipos de dibujado pueden ser empleados tanto para 2D como para 3D, es una de las grandes ventajas con las que contamos gracias a la implementación personalizada de shaders.

En el motor gráfico se han implementado dos tipos diferentes de dibujado de líneas en 3D, *Draw3DLine* y *Draw3DLineLocal*. Ambos hacen uso del mismo shader (Código 6.3):

- **Draw3DLine:** Es el dibujado de linea 3D estándar. Basta con especificarle un punto inicial, un punto final y un color para que dibuje una linea 3D en tu escena. El meca-

nismo es similar al explicado anteriormente con las matrices view y projection. En este caso al solo contar con puntos en el espacio podemos multiplicar directamente dichos valores relativos al mundo por las matrices mencionadas y conseguiremos posicionar nuestro vértice en pantalla.

- **Draw3DLineLocal:** Es la manera de dibujar una linea 3D en coordenadas locales a un objeto. Es muy útil para dibujar *bounding boxes* de mallas de las que conoces los vértices. En este caso, a diferencia del anterior, el vector de posición de dicho vértice viene en coordenadas locales y gracias a la matriz modelo de dicho objeto al que pertenece conseguimos dibujar la linea 3D.

Código 6.3: Vertex shader para el dibujado de lineas 3D

```

1 layout (location = 0) in vec3 aPos;
2 uniform mat4 view;
3 uniform mat4 model;
4 uniform mat4 projection;
5 uniform bool localMode;
6
7 void main() {
8     if(localMode)
9         gl_Position = projection * view * model * vec4(aPos, 1.0);
10    else
11        gl_Position = projection * view * vec4(aPos, 1.0);
12 }
```

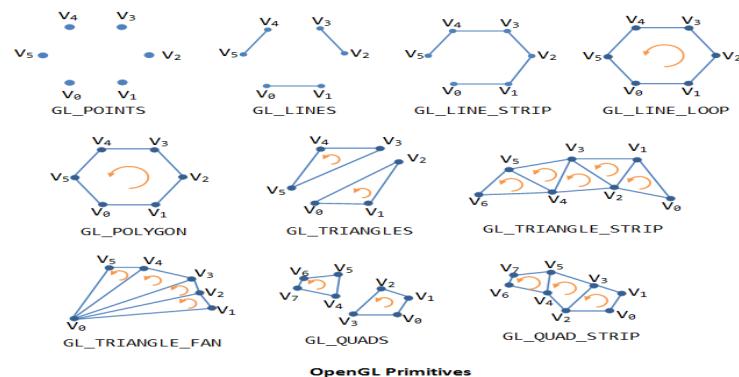


Figura 6.5: Todas las formas de renderizado implementadas en OpenGL. Fuente: <https://artificialnature.net/gct633/graphics.html>.

6.2 Físicas

El apartado de físicas del proyecto ha sido uno de los puntos más importantes debido a su necesidad y complejidad. En el proyecto se ha necesitado recrear una representación con características similares a las presentes en la realidad: la gravedad ejercida por la tierra, la velocidad de movimiento de la estructura bípeda, un movimiento articulado fidedigno o incluso la detección de caídas del personaje al andar.

Es importante recordar que el objetivo de este proyecto es conseguir que un ser bípedo consiga andar de manera óptima. Por ello, las físicas que se van a explicar a continuación van relacionadas con este caso concreto para así conseguir una mayor optimización de rendimiento y ahorro de trabajo innecesario. Es por lo tanto, otra de las ventajas ya mencionadas anteriormente del por qué no utilizar un motor de físicas de terceros.

6.2.1 Estructura bípeda

Al igual que se ha explicado en el apartado del funcionamiento del *árbol de la escena* (Sección 6.1.1.1) qué datos son necesarios para la representación visual de los datos y mallas, en el proyecto se ha recreado una estructura similar para conseguir encapsular dicha funcionalidad. Es por ello que contamos con dos clases principales a la hora de trabajar con entidades: *Entity* y *EMesh*.

La clase *Entity* es la clase padre de *EMesh*, así se consigue generalizar valores para los demás tipos de entidades con las que cuenta el proyecto como puede ser la cámara. En la Tabla 6.2 se muestran las variables encapsuladas más importantes dentro de esta clase.

Variable	Tipo de dato	Descripción
id	uint32_t	Identificador de la entidad
parent	Entity*	Padre de la entidad
position	glm::vec3	Vector de la posición
rotation	glm::vec3	Vector de la rotación
scalation	glm::vec3	Vector de escalado
rotationBoundaries	std::pair<float, float>	Mínimo y máximo valor de rotación

Tabla 6.2: Variables encapsuladas dentro de la clase *Entity*.

Como se ha dicho estos son alguno de los parámetros que contiene la clase *Entity*, ahora es necesario mencionar cuáles son los parámetros con los que cuenta la clase *EMesh* que es la clase utilizada para la representación de las mallas en el proyecto. En la Tabla 6.3 podemos observar sus parámetros más importantes.

Variable	Tipo de dato	Descripción
meshPath	std::string	Ruta de la malla asociada
dimensions	glm::vec3	Ancho, alto y profundo de la malla
collider	OBBCollider (Tabla 6.5)	Datos sobre su bounding box asociada
rotationVelocity	glm::vec3	Vector de velocidad para la rotación en 3 ejes

Tabla 6.3: Variables propias de la clase *EMesh*

Por último, contamos con una clase auxiliar denominada *ESkeleton* que sirve para poder formar una composición de *EMesh*, en este caso cada *EMesh* representará una parte de la estructura bípeda del experimento. Por lo tanto, los objetos principales que contiene esta clase son *EMeshes* para: tronco, codo y brazo derecho, codo y brazo izquierdo, cadera y rodilla derecha y cadera y rodilla izquierda. En la Tabla 6.4 se puede ver más en detalle el resto de variables con las que cuenta, sin ninguna duda, la clase mas importante en las físicas del proyecto.

Variable	Tipo de dato	Descripción
skeletonId	uint16_t	Identificador del esqueleto
flexibility	ESkeleton::Flexibility	Tipo de flexibilidad del esqueleto ²
core	EMesh*	Puntero a la malla del tronco
leg1Joints	std::vector<EMesh*>	Cadera y rodilla derecha
leg2Joints	std::vector<EMesh*>	Cadera y rodilla izquierda
arm1Joints	std::vector<EMesh*>	Hombro y codo derecho
arm2Joints	std::vector<EMesh*>	Hombro y codo izquierdo
onAir	bool	Si se encuentra en el aire
leg1OnAir	bool	Si se encuentra en el aire la pierna derecha
leg2OnAir	bool	Si se encuentra en el aire la pierna izquierda
isDead	bool	Si el esqueleto sigue con vida
fitness	float	Valor de fitness del esqueleto

Tabla 6.4: Variables propias de la clase *ESkeleton*

En la Figura 6.6 se observa la estructura bípeda descrita renderizada en el proyecto. Cada articulación mencionada pertenece a un *EMesh* propio y gracias al gestor de recursos (Sección 6.1.1.2) conseguimos una optimización a la hora de realizar la lectura de las mallas repetidas debido a la naturaleza simétrica del esqueleto humano.

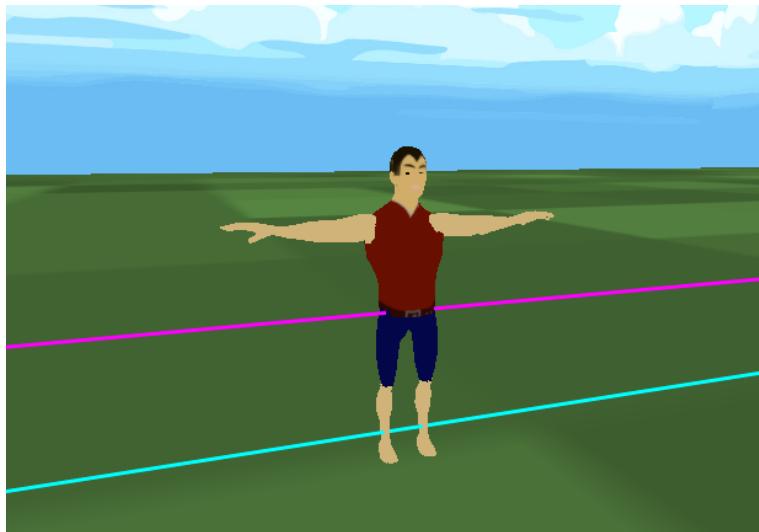


Figura 6.6: Esqueleto renderizado en la ejecución del proyecto. Fuente propia.

6.2.2 Colisiones OBB

Las colisiones son sin duda el punto más importante de las físicas realizadas en el proyecto. Para lograr el objetivo de recrear unas físicas similares a las que se dan en la realidad es necesario conseguir detectar unas colisiones los más acertadas posibles.

Para poder calcular colisiones entre objetos se necesita algún tipo de información relativa al objeto que delimita su zona de colisión, a esto se le denomina *bounding box* y los dos tipos más utilizados son: AABB y OBB. En la Figura 6.7 observamos las diferencias entre ambos sistemas.

- **AABB:** Las *bounding boxes* de tipo AABB son ampliamente utilizadas gracias a su facilidad de implementación y su eficiencia de cómputo. Crea una caja alrededor del objeto basándose exclusivamente en sus valores de anchura, altura y profundidad. Es por ello que en algunos casos puede producir falsos positivos a la hora de detectar colisiones,

²Baja, media o alta. Influye en la el valor mínimo y máximo que puede girar las articulaciones el esqueleto

sobretodo en objetos complejos. Pero consigue colisiones perfectas o casi perfectas en objetos con geometrías similares a cuadrados o rectángulos. Comúnmente esta técnica se suele utilizar para detectar si existe una posible colisión de manera eficiente y si el resultado es positivo aplicar un sistema más potente y eficaz.

- **OBB:** Las *bounding boxes* de tipo OBB (Eberly, 2001) son más complejas de implementar y requieren de una mayor cantidad de computo en su uso. Pero por otro lado, consigue mejores resultados gracias a estar orientada junto al objeto. Mientras que las *bounding boxes* de tipo AABB no toman en cuenta la rotación que sufre nuestro objeto, las de tipo OBB son especialistas en este tipo de casos.

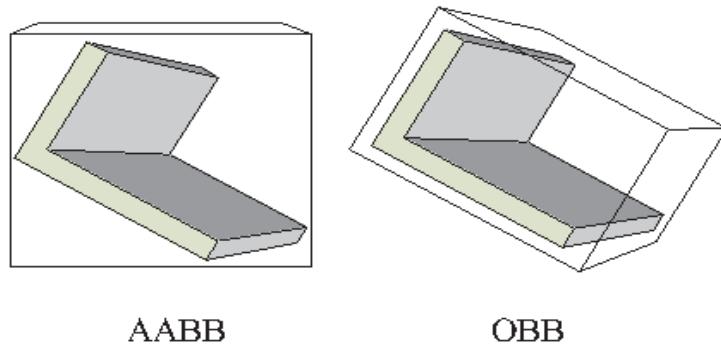


Figura 6.7: Comparativa de una bounding box AABB y una OBB. Fuente: <https://devdept.zendesk.com/hc/en-us/articles/360011559320-How-Collision-Detection-works>.

Dado que en este proyecto se ha fijado como primer objetivo un resultado final lo más fiel a la realidad posible, se ha optado por utilizar OBB en los objetos debido a que principalmente las caderas y rodillas van a estar en constante rotación.

Es importante aclarar la diferencia entre el tipo de *bounding box* utilizada para definir un objeto y la técnica de colisión con la que detectamos colisiones entre dichos objetos. En este caso se ha utilizado *bounding boxes* de tipo OBB pero las colisiones entre objetos se han realizado de una manera optimizada pensando en nuestro caso de uso. Gracias a ello se consigue la precisión de los OBB y se reduce el coste de rendimiento al utilizarlos para calcular colisiones.

Como se ha mostrado anteriormente, las entidades de tipo *EMesh* cuentan con un objeto

llamado *OBBCollider*. Éste es el objeto utilizado para almacenar los datos necesarios para formar un OBB. Los parámetros son los mostrados en la Tabla 6.5.

Variáble	Tipo de dato	Descripción
vertexs	glm::vec3[8]	La posición de los 8 vértices que definen un cubo
center	glm::vec3	Centro de la <i>bounding box</i>
size	glm::vec3	Anchura, altura y profundidad
axes	glm::mat3	Orientación de los ejes

Tabla 6.5: Variables propias de la clase *OBBCollider*

Finalmente en la Figura 6.8 se observan los OBB que presenta cada *EMesh* que componen el esqueleto del proyecto. Especialmente la cadera y rodilla que giran junto a sus OBBs asociadas.

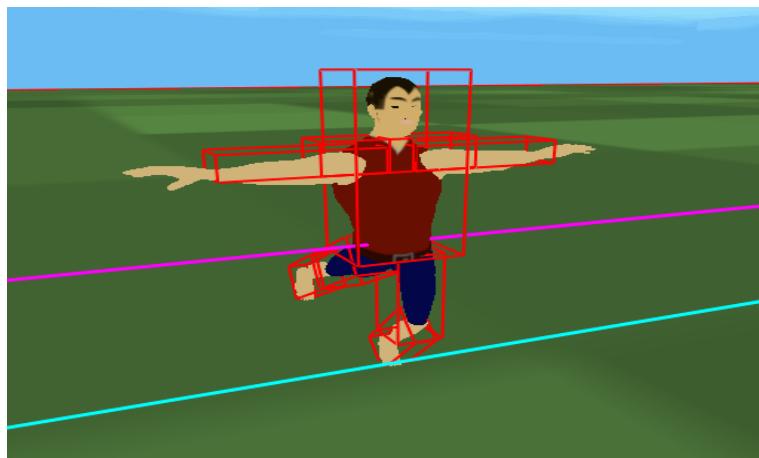


Figura 6.8: Esqueleto renderizado y dibujado de sus OBBs.

6.2.3 Cinemática directa vs cinemática inversa

La cinemática es la ciencia que describe cómo deben de moverse los objetos sólidos sin tener en cuenta las fuerzas que lo originan. En el caso concreto del movimiento de la estructura bípeda del proyecto se debe conseguir mover todas las articulaciones asociadas al tronco del cuerpo cuando únicamente sea éste el que se mueva. Es decir, si el tronco se desplaza, se rota o se escala en XYZ un valor, el resto de sus mallas asociadas deben aumentar o disminuir la

misma cantidad.

Ante esta situación se plantean dos soluciones posibles: cinemática directa y cinemática inversa.

- **Cinemática directa:** La cinemática directa es la manera de calcular las posiciones de partes de una estructura articulada en base a los valores de sus estructuras padres, como por ejemplo: la posición de una mano moviéndose si mueves el codo al que va unido y a su vez el codo moviéndose si mueves el hombro. Es un sistema matemáticamente sencillo de implementar, con un gran rendimiento en su uso y produce resultados muy realistas y fieles a la realidad. Por el contrario, para conseguir esos resultados se requiere ajustar muy bien las posiciones de cada articulación padre.
- **Cinemática inversa:** La cinemática inversa calcula la posición de cada uno de los padres mediante la posición final del denominado *"efector"*. Esto puede generar múltiples soluciones válidas, lo cual puede llegar a ser un problema importante como se aprecia en la Figura 6.9. Por otro lado su implementación es más compleja respecto a la cinemática directa y necesita realizar más cálculos para conseguir una solución debido a que cuanto mayor sea el número de articulaciones padres, más valores se deberán de calcular y tener en cuenta. Pero cuenta con una gran ventaja y es que solamente es necesario especificar un punto final para conseguir el movimiento de todas las articulaciones deseadas.

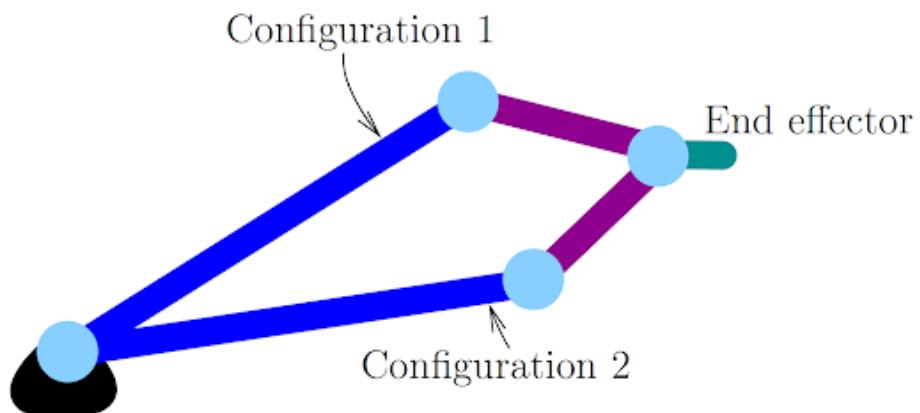


Figura 6.9: Posibles soluciones que puede producir un movimiento mediante cinemática inversa. Fuente: http://faculty.salina.k-state.edu/tim/robotics_sg/Arm_robots/inverseKin.html.

Finalmente en el proyecto se decidió implementar una cinemática directa para el movimiento del esqueleto debido a que su movimiento se basa en el giro de caderas, rodillas y movimiento del tronco. La cinemática directa ha sido la mejor opción ya que en el caso de haberse utilizado la cinemática inversa no se le habría sacado gran partido ya que tendría que calcular la posición para demasiados efectores. A su vez el motor gráfico en forma de árbol explicado en la Sección 6.1.1.1 permite realizar la implementación de este tipo de cinemática más fácilmente.

6.2.4 Gravedad

Lógicamente no se puede conseguir una representación física parecida a la real si no se implemente un sistema de gravedad que afecte a las entidades de la escena. La implementación ha sido muy sencilla ya que se trata únicamente de aplicar una fuerza de aceleración con dirección hacia el suelo.

A pesar de su simpleza se ha buscado optimizar esta tarea al máximo para evitar problemas de rendimiento. Gracias a la cinemática directa explicada en la Sección 6.2.3 solamente es necesario mover el tronco del esqueleto para conseguir mover todo su conjunto de articulaciones. Por otro lado, gracias al uso de variables que incorpora la clase *ESkeleton* como es la variable *bool ESkeleton::onAir* conseguimos optimizar el número de veces que realizamos esta operación. A continuación se muestra la implementación (Código 6.4). Como se observa, a parte de trasladar el esqueleto tenemos que realizar lo mismo con su OBB.

Código 6.4: Sistema de gravedad del proyecto

```
1 if (skeleton->GetOnAir()) {  
2     auto core = skeleton->GetCore();  
3     glm::vec3 movement = gravity * Utils::deltaTime;  
4     core->SetPosition(core->GetPosition() + movement);  
5  
6     skeleton->TraslateOBB(movement);  
7 }
```

6.2.5 Dinámica del movimiento

Una vez se ha explicado todo el funcionamiento de las técnicas utilizadas en las físicas del proyecto, solo queda explicar cómo se mueve el esqueleto.

Como ya se ha explicado y se ha visto reflejado en el efecto de la gravedad, el movimiento de todo el esqueleto se basa únicamente en el movimiento que se le aplica al tronco del cuerpo. En este caso es igual, deberemos calcular el desplazamiento que sufre el tronco en función del impacto y la velocidad que aplican las articulaciones de las piernas con el suelo.

Las articulaciones giran en forma de arcos, es decir, al llegar al máximo de su rotación invierten su sentido. Si el cálculo de la rotación da fuera de dicho margen se situará justo en el valor máximo.

Por otro lado, también es necesario detectar cuando un esqueleto toma valores por los cuales se puede determinar que se ha caído (variable *dead* Tabla 6.4). Estos valores han sido producto de un estudio experimental y definidos de la manera más objetiva posible. Los valores por los que se determina que el esqueleto se ha caído son:

1. La rotación de ambas articulaciones de la cadera es menor a -30°.
2. La rotación de ambas articulaciones de la cadera menos la rotación de las rodillas son mayores a 30°.
3. La rotación de la cadera menos la rotación de la rodilla de la pierna delantera es mayor a 90° y la rotación de la cadera de la pierna trasera es menor a -90°.
4. La rotación de la cadera de la pierna delantera es mayor a 120° y la rotación de la cadera de la pierna trasera es menor a -90°.
5. La rotación de la cadera de la pierna trasera es menor a -30° y la otra pierna no está apoyada en el suelo.

6.3 Algoritmo genético

Finalmente, una vez se ha explicado y entendido el funcionamiento tanto de la representación gráfica del proyecto como de las físicas aplicadas en él, llega el momento de explicar el funcionamiento e implementación del algoritmo genético.

El objetivo del AG, como se ha explicado anteriormente, consiste en encontrar unos determinados valores para diferentes parámetros con los cuales nuestro esqueleto consiga andar de la mejor manera posible.

¿Qué parámetros son estos? Pues bien, como se ha explicado en el apartado de físicas (Sección 6.2.5) el movimiento del esqueleto es producido por las rotaciones de sus caderas y rodillas y las velocidades de las mismas. Es por ello que serán estos parámetros los que sean objeto de estudio por el AG.

Para satisfacer el objetivo de dotar al proyecto con un carácter educativo se han parametrizado en el menú inicial una serie de valores que afectan al resultado del AG. Estos parámetros son: número de individuos, tiempo de vida, número máximo de generaciones, probabilidad de generar nuevos individuos, probabilidad de sufrir una mutación, función de selección y operador de cruce. En la Figura 6.10 se observa visualmente en el proyecto. Más adelante en la Sección 6.3.6 detallaremos en profundidad sus características.



Figura 6.10: Imagen del menú de configuración del proyecto.

6.3.1 Generación de la población

Como se ha explicado anteriormente el primer paso a realizar en el flujo de un AG es la generación de la población inicial. Es de vital importancia que se realice de manera aleatoria

para no predeterminar la evolución de la población.

En el caso del proyecto para generar la población inicial es necesario asignar valores aleatorios a los parámetros mencionados anteriormente que van a ser los que evolucionen con el tiempo en las diferentes etapas del AG.

El primer paso es determinar la flexibilidad del individuo. Se cuenta con el mismo porcentaje para los tres tipos de flexibilidad establecidos (33%) ya que tras una serie de pruebas empíricas se determinó que no es un parámetro que pudiera influir en la evolución debido principalmente a la etapa de *crossover*. Esto es debido a que cuando se comienzan a cruzar diferentes individuos el tipo de flexibilidad se pierde y solo se tiene en cuenta los parámetros definidos.

No obstante, el tipo de flexibilidad es de utilidad para generar los mínimos y máximos valores de rotación de las articulaciones más fácilmente. Como se ha explicado anteriormente la flexibilidad define los valores mínimos y máximos de rotación que puede tomar una articulación (Tabla 6.6).

Para generar los valores mínimos y máximos de rotaciones en las piernas simplemente se ha generado un número aleatorio entre 0 y 1 y se ha multiplicado por el rango de valores posibles. Con esto conseguimos que aunque un individuo tenga buena flexibilidad no pierda la aleatoriedad que buscamos pero si tenga más probabilidad de tener valores más altos.

De igual manera se ha realizado para los valores de velocidad, solo que en este caso los valores mínimos y máximos de velocidad se han prefijado a unos valores que se consideran naturales para el andar de una persona bípeda (30, 80 ud/frame).

A pesar de que la generación de valores se busca que sea aleatoria, sí se han determinado ciertos tipos de valores los cuales evitar para no generar individuos completamente inservibles. Se debe cumplir que la rotación mínima y máxima de la cadera no sean ambas negativas ya que si fuera así el esqueleto nunca avanzaría.

Flexibilidad	Articulación	Mínimo	Máximo
LOW	Cadera	-40º	40º
LOW	Rodilla	-50º	0º
MEDIUM	Cadera	-105º	105º
MEDIUM	Rodilla	-120º	0º
HIGH	Cadera	-160º	160º
HIGH	Rodilla	-160º	0º

Tabla 6.6: Valores posibles en función de la flexibilidad del individuo.

6.3.2 Función fitness

La función fitness de un individuo determina cómo de apto es con respecto al resto de la población para así poder aplicar una selección de los individuos más aptos durante la evolución de la población, es decir, que sean más propensos a avanzar entre generaciones.

Debido a que en el proyecto se ha buscado conseguir la manera más eficaz para andar a lo largo de un periodo de tiempo, se ha visto conveniente determinar la función de fitness como la distancia euclídea recorrida desde el punto de inicio de dicha generación hasta el fin del ciclo de vida. En el caso concreto del proyecto se ha usado la formula para 3 dimensiones XYZ. A continuación se muestra la formula generalizada siendo P el punto final del esqueleto y Q el punto desde donde inicial.

$$P = (p_1, p_2, \dots, p_n)$$

$$Q = (q_1, q_2, \dots, q_n)$$

$$\text{fitnessValue} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

6.3.3 Selección

Una vez se ha definido la función fitness por la cual podemos clasificar individuos como más o menos aptos, llega el momento de la etapa de seleccionar qué individuos de la actual generación formarán parte de la siguiente generación y de los cuales se generarán nuevos indi-

viduos. Como se ha dicho antes, se cuenta con un valor parametrizado con el cual determinar cuántos individuos de la actual generación deben ser seleccionados y cuántos se generarán mediante cruces en la siguiente etapa.

Existen una gran variedad de maneras diferentes por las cuales determinar qué individuos avanzan a la siguiente generación y cuáles no. En este proyecto han sido implementadas 2 funciones de selección diferentes:

- **Selección por ruleta:** La selección por ruleta es un método de selección elitista el cual penaliza de gran manera a aquellos individuos que consiguieron los peores resultados durante el tiempo de vida de la generación. Su funcionamiento se centra en generar segmentos de probabilidad en función de los valores de fitness siguiendo la fórmula (6.1):

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j} \quad (6.1)$$

Donde f_i es el valor fitness del individuo, N el número de individuos de la población y p_i es la probabilidad de ser seleccionado el individuo i .

Generalmente se generan tantas secciones de probabilidades como número de individuos en la población. Un mayor número de secciones en la ruleta dotará al algoritmo de una selección mucho más elitista, asegurando la supervivencia de los mejores individuos y consiguiendo que individuos menos aptos teóricamente, se queden por el camino. Esto podría llegar a ser un problema debido a la pérdida de diversidad en la población.

- **Selección por torneo:** La selección mediante torneo es quizás el método más sencillo de entender e implementar. Su sistema se basa en seleccionar un número N de individuos de la población y hacerlos "pelear" entre ellos para que el más apto (mayor valor fitness) sea el ganador y por lo tanto el que avanzará a la siguiente generación.

Al igual que en la selección por ruleta, contamos con un parámetro que podemos modificar para alterar el funcionamiento del torneo, se trata del número de individuos que participan en cada torneo. El valor comúnmente más utilizado es $N = 2$, es lo que se denomina como "*torneo binario*". Un número de individuos alto para participar

en el torneo aumentará el elitismo de la función de selección mientras que un número más bajo ayudará a garantizar la diversidad entre los individuos de la población. En la Figura 6.11 se muestra un ejemplo práctico.

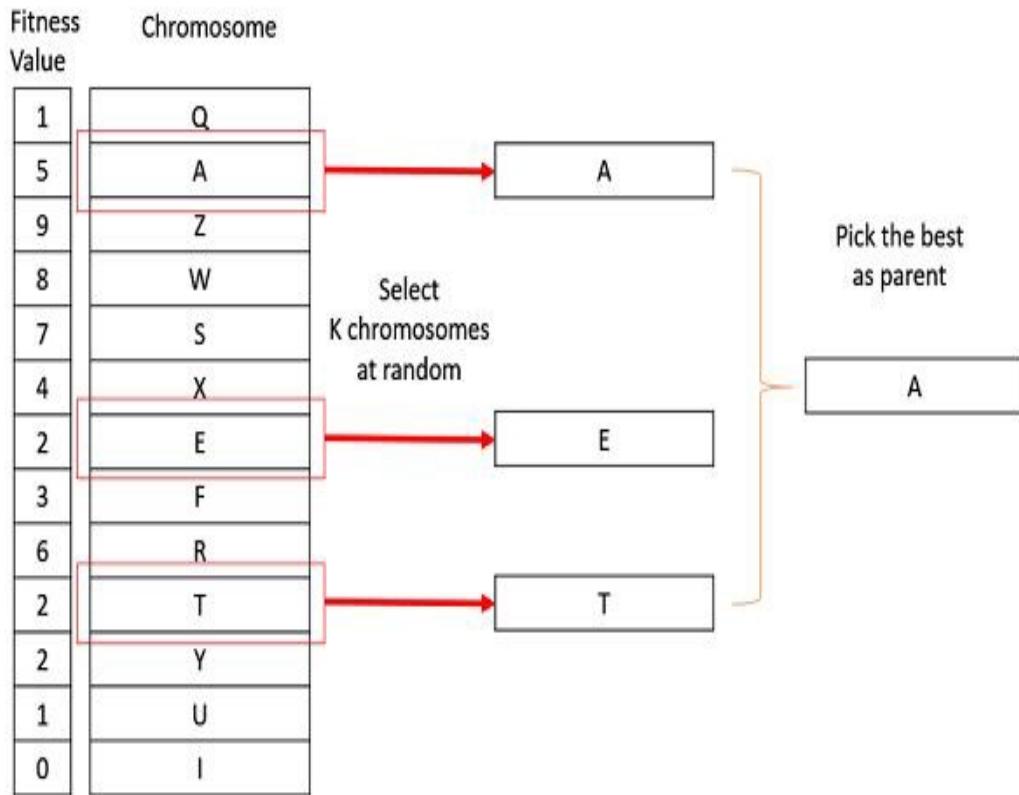


Figura 6.11: Ejemplo práctico de una selección por torneo con $N = 3$. Fuente: <https://medium.com/datadriveninvestor/genetic-algorithms-selection-5634cfcc45d78>.

6.3.4 Crossover o cruce

El operador de cruce o *crossover* es considerada la etapa más importante del AG (Umbarkar y Sheth, 2015), tanto es así que se puede llegar a considerar un AG sin selección pero nunca sin *crossover*. Tal es así que en el proyecto se le ha querido dar especial importancia a la hora de estudiar qué cambios es capaz de generar en los resultados finales y han sido implementados 4 operadores de cruce distintos.

- **Heurístico:** El *crossover* heurístico (Kaya y cols., 2011) sigue una fórmula concreta para calcular el valor resultante entre los dos cromosomas seleccionados para el cruce

(6.2). Es el único de los operadores implementados que puede generar valores que superen a los establecidos como límites. Para ello se utiliza una variable n que indica el número de ejecuciones del algoritmo antes de seleccionar por defecto al peor cromosoma de los dos.

En los siguientes operadores se verá como su función es generar un valor que se sitúe entre el menor y el mayor valor, esto puede ser una ventaja cuando no existen valores nulos entre dichos puntos. Por otro lado en el operador heurístico se pueden conseguir valores superiores o inferiores al que presentan los padres y eso puede ser una gran ventaja a la hora de generar individuos más diversos.

$$\text{offspring} = f_1 + (r * (f_1 - f_2)) \quad (6.2)$$

Donde f_2 es el el valor del individuo con menor fitness, f_1 el valor del individuo con mejor fitness, r un valor aleatorio entre $[0, 1]$ y offspring el valor resultante.

A continuación (6.3) se puede ver un ejemplo práctico para los siguientes valores y el resultado final.

$$\begin{aligned} f_1 &= 80 \\ f_2 &= 30 \\ \alpha &= 0.2 \end{aligned} \quad (6.3)$$

$$\begin{aligned} \text{offspring} &= 80 + (0.2 * (80 - 30)) \\ &= 90 \end{aligned}$$

- **Aritmético:** A diferencia del heurístico, el operador aritmético se basa en hallar un valor contenido sobre la recta que une los dos puntos (mejor y peor valor) utilizando la fórmula clásica de la interpolación lineal (6.4). Se genera un valor aleatorio entre $[0, 1]$ y el valor resultante sera el valor de cruce.

Los valores que genere en el cruce de cromosomas no pueden ser nulos ya que en el caso del proyecto nunca existen valores nulos entre los puntos mínimos y máximos. Por

el contrario puede ser un operador de cruce que no dé espacio a la diversidad en la población, pudiendo llevar a soluciones engañosas.

$$offspring = (A * f_1) + (1 - A) * f_2 \quad (6.4)$$

Donde f_2 es el valor del individuo con menor fitness, f_1 el valor del individuo con mejor fitness, A un valor aleatorio entre $[0, 1]$ y $offspring$ el valor resultante.

A continuación (6.5) se puede ver un ejemplo práctico para los siguientes valores y el resultado final.

$$\begin{aligned} f_1 &= 80 \\ f_2 &= 30 \\ A &= 0.2 \end{aligned} \quad (6.5)$$

$$\begin{aligned} offspring &= (0.2 * 80) + (1 - 0.2) * 30 \\ &= 40 \end{aligned}$$

- **Media:** El operador media o *average* consiste en calcular el valor medio entre el punto mínimo y el punto máximo. Es el operador más sencillo y rápido computacionalmente.

A diferencia de los operadores aritmético y heurístico, este operador no favorece a ninguno de los dos individuos sino que los toma como iguales. También es más propenso que los anteriores métodos a producir valores iguales para diferentes rangos, por ejemplo: $(100 + 20)/2 = 60$ y $(70 + 50)/2 = 60$.

Por otro lado, puede llegar a ser uno de los más rápidos en conseguir converger en un valor exacto ya que dos hijos de los mismos padres siempre tendrán como resultado el mismo valor (Figura 6.12) debido a que carece de aleatoriedad a diferencia de los demás operadores mencionados.

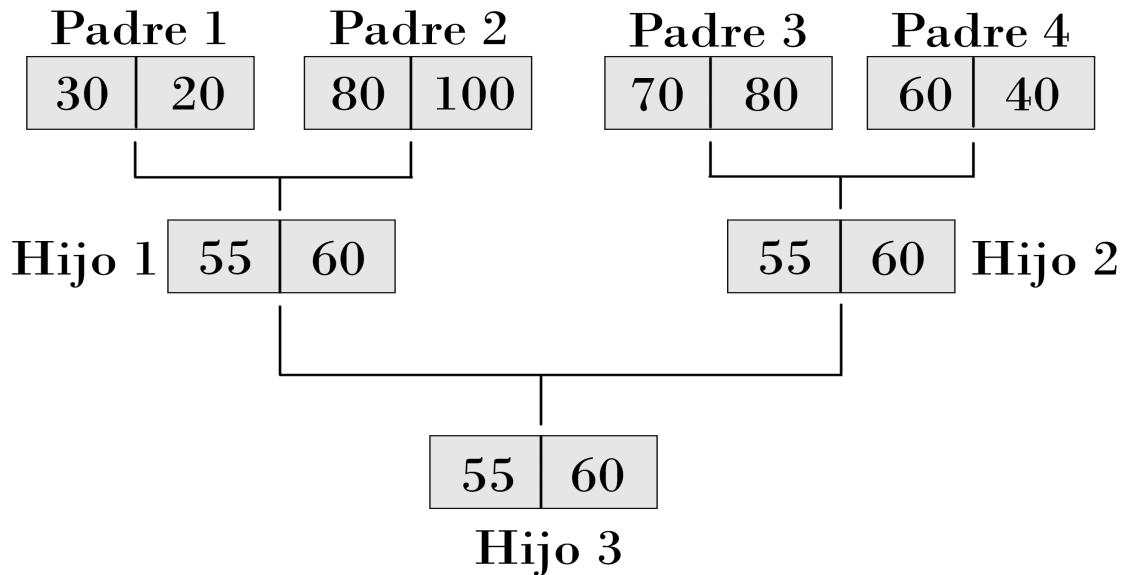


Figura 6.12: Cruces entre cromosomas diferentes que generan los mismos resultados en hijos diferentes y hacen converger el resultado muy rápidamente. Fuente: Imagen propia.

- **One point:** El operador de cruce *one point* es el único implementado que no trata de calcular valores sobre la recta de unión de ambos padres. Este operador se centra en los valores como características independientes y en seleccionar parámetros de ambos padres.

Su mecanismo es simple, genera un punto de pivote aleatorio que situado entre todos los valores con los que realizar el *crossover* selecciona los valores a la izquierda del pivote de un parente y los valores a la derecha del pivote de otro parente (Figura 6.13).

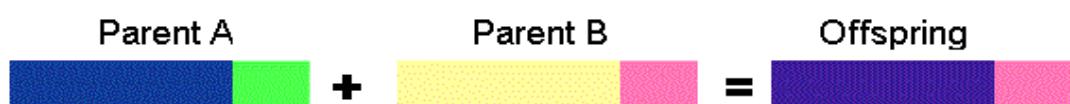


Figura 6.13: Cruce por operador *one point*. Fuente: <https://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php>.

6.3.5 Mutación

La mutación es el último paso antes de volver a iterar por todos los pasos mencionados. En ella se genera una mutación (alteración) en los valores de un individuo escogido para evitar el estancamiento y favorecer la diversidad genética. Es la probabilidad que tiene un individuo de ser mutado antes de avanzar a la siguiente generación.

Para poder estudiar el posible efecto que puede tener este valor en el algoritmo se ha definido un valor parametrizable para definir la probabilidad de que ocurra una mutación. Generalmente el porcentaje suele ser muy bajo ($< 1\%$), en la Sección 7.5 ha sido determinado el valor más eficiente en este caso.

En el proyecto se ha optado por alterar aleatoriamente todos los parámetros de un individuo cuando una mutación deba de ser aplicada. A efectos prácticos es como si se generara un nuevo individuo sin importar como era inicialmente el individuo sin mutar.

6.3.6 Visualización de la información

Finalmente, para poder estudiar la evolución de una manera sencilla ha sido muy importante un correcto sistema de representación de los datos. Para ello, durante la ejecución del proyecto aparecerá una ventana donde se mostrarán los datos en tiempo real. Los datos recopilados en cada generación son los siguientes (Tabla 6.7).

En la Figura 6.14 se han asociado números para explicar a continuación qué representa cada valor.



Figura 6.14: Imagen de explicación para cada parámetro de la ventana de resumen de las estadísticas en tiempo real del proyecto.

En la Figura 6.15 se muestran las gráficas de los puntos 7 (6.15a), 8 (6.15b), 9 (6.15c) y 10 (6.15d) en las que el eje Y representan los valores especificados y el eje X representa el transcurso de las generaciones.

Clicando en el número 19 (Population) se muestran todos los individuos de la generación que está en desarrollo y los datos ya mencionados que intervienen en el desarrollo de un individuo (rotaciones de articulaciones, velocidades, fitness, etc...).

Finalmente, clicando en el número 20 (Generations) se muestra un histórico de todos los datos recopilados de la Tabla 6.7 en cada una de las generaciones terminadas.

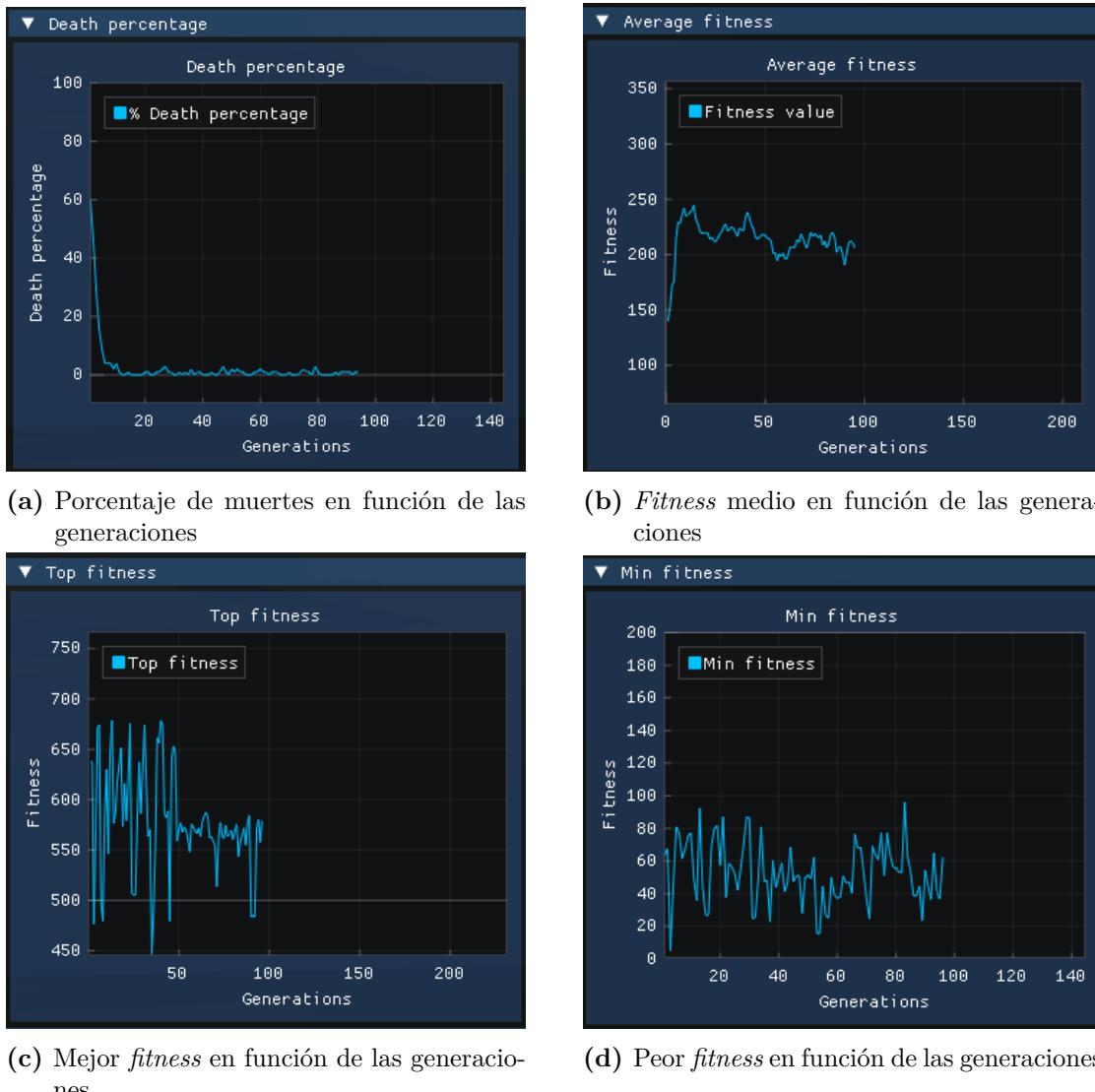


Figura 6.15: Imagen de los cuatro usos que se le da a la gráfica de líneas para representar los datos del porcentaje de muertes, fitness medio, mejor fitness y peor fitness.

Han sido implementadas tres tipos de cámaras distintas para la visualización del proyecto; la primera de ellas es una cámara libre en la que puedes controlar su movimiento, la segunda es una cámara que enfoca y sigue al mejor individuo de la generación y la última cámara que sigue al mejor individuo de la generación pasada (Figura 6.16). Todas estas cámaras presentan la opción de mostrar únicamente al individuo que está siguiendo para así poder fijarnos más fácilmente en qué está haciendo.



Figura 6.16: Imagen extraída del programa con la cámara de seguimiento al mejor individuo de la generación pasada.

Finalmente, se cuenta con una opción para exportar todos los datos recogidos durante la ejecución (Tabla 6.7) a formato CSV para poder analizarlo con herramientas externas.

Variable	Tipo de dato	Descripción
generation	int	Número de generación
bestGeneId	int	Identificador del mejor individuo
deathPercentage	float	Porcentaje de muertes
averageFitness	float	Fitness medio
topFitness	float	Mejor fitness
minFitness	float	Peor fitness
averageHip1Velocity	float	Velocidad media de la cadera derecha
averageKnee1Velocity	float	Velocidad media de la rodilla derecha
averageHip2Velocity	float	Velocidad media de la cadera izquierda
averageKnee2Velocity	float	Velocidad media de la cadera izquierda
topHip1Velocity	float	Velocidad máxima de la cadera derecha
topKnee1Velocity	float	Velocidad máxima de la rodilla derecha
topHip2Velocity	float	Velocidad máxima de la cadera izquierda
topKnee2Velocity	float	Velocidad máxima de la rodilla izquierda
minHip1Velocity	float	Velocidad mínima de la cadera derecha
minKnee1Velocity	float	Velocidad mínima de la rodilla derecha
minHip2Velocity	float	Velocidad mínima de la cadera izquierda
minKnee2Velocity	float	Velocidad mínima de la rodilla izquierda
averageHip1Rotation	<float, float>	Media de valores min y max rotación cadera drch
averageKnee1Rotation	<float, float>	Media de valores min y max rotación rodilla drch
averageHip2Rotation	<float, float>	Media de valores min y max rotación cadera izq
averageKnee2Rotation	<float, float>	Media de valores min y max rotación rodilla izq
topHip1Rotation	float	Valor máximo de rotación de la cadera derecha
topKnee1Rotation	float	Valor máxima de rotación de la rodilla derecha
topHip2Rotation	float	Valor máxima de rotación de la cadera izquierda
topKnee2Rotation	float	Valor máximo de rotación de la rodilla izquierda
minHip1Rotation	float	Valor mínimo de rotación de la cadera derecha
minKnee1Rotation	float	Valor mínimo de rotación de la rodilla derecha
minHip2Rotation	float	Valor mínimo de rotación de la cadera izquierda
minKnee2Rotation	float	Valor mínimo de rotación de la rodilla izquierda
bestHip1Rotation	<float, float>	Rotación de la cadera derecha del mejor gen
bestKnee1Rotation	<float, float>	Rotación de la rodilla derecha del mejor gen
bestHip2Rotation	<float, float>	Rotación de la cadera izquierda del mejor gen
bestKnee2Rotation	<float, float>	Rotación de la rodilla izquierda del mejor gen
bestHip1Velocity	float	Velocidad de la cadera derecha del mejor gen
bestKnee1Velocity	float	Velocidad de la rodilla derecha del mejor gen
bestHip2Velocity	float	Velocidad de la cadera izquierda del mejor gen
bestKnee2Velocity	float	Velocidad de la rodilla izquierda del mejor gen

Tabla 6.7: Tabla de valores almacenados en cada generación.

7 Experimentación

Finalmente, tras comprender el funcionamiento de todas las etapas de este proyecto llega el momento de ponerlo a prueba y experimentar con los parámetros mencionados para determinar la importancia en el resultado final.

Durante este capítulo aparte de estudiar el efecto que causan cada uno de los parámetros disponibles también se determinarán los valores más eficientes para los parámetros definidos para evolucionar en el AG que son: rotación mínima y máxima de ambas caderas y rodillas y velocidad de rotación de ambas caderas y rodillas.

Para las siguientes pruebas se han realizado varias simulaciones y el valor resultante es la media, mediana, etcétera de esos valores. Se entiende por $MEANf(x)$ como el valor medio de todas las ejecuciones, $MEDIANf(x)$ como la mediana de todas las ejecuciones, $MAXf(x)$ como el valor máximo de todas las ejecuciones y $MINf(x)$ como el valor mínimo de todas las ejecuciones. De igual manera para $f_i(x)$ refiriéndose a un individuo concreto de la población.

7.1 Margen de error

Durante el desarrollo de la experimentación se llegó a la conclusión de que un mismo individuo (mismos valores en sus parámetros) puede obtener un valor *fitness* diferente de una generación a otra. Esto es debido principalmente a la disminución de la tasa de frames debido a que realizamos una iteración de cálculos de las físicas por cada fotograma. Debido a que el resultado final es una discretización de los valores intermedios, cuantas más veces se ejecuten las físicas (mayor muestreo), más fiable será el resultado.

Es por ello que esta parte de experimentos la dedicaremos a determinar cuáles son esos errores para más adelante tenerlos en cuenta. También cabe recalcar que a pesar de ello el dato más importante a estudiar será el fitness medio de la generación para ver su evolución

general.

Para determinar el margen de error existente se han estudiado ejecuciones con individuos sin aplicarles ningún tipo de selección o cruce, con esto se consigue que los individuos tengan siempre los mismos valores. Se han realizado pruebas para diferentes tamaños de población y tiempo de vida.

Población	Tiempo de vida	MEAN $f_i(x)$	MAX $f_i(x)$	MIN $f_i(x)$	$E = MAX - MIN$
2	30	316,12	321,61	312,65	8,96
2	30	227,65	236,92	222,38	14,54
100	10	163,64	200,91	140,99	59,92
100	20	329,39	393,70	306,81	86,89
100	30	460,09	500,62	404,35	96,27
100	45	786,13	833,32	734,93	98,39

Tabla 7.1: Resultado de pruebas con diferentes tamaños de población y tiempos de vida pero sin selección ni mutación para representar el margen de error de un individuo.

Los resultados obtenidos (Tabla 7.1) dejan claro que aumentando el tamaño de la población también aumenta el error. De igual manera con el tiempo de vida de la generación. Esto está directamente relacionado con el descenso del frame rate a la hora de realizar los cálculos, recordemos las especificaciones de la máquina con la que se está realizando estas pruebas (Hardware 5.3). Para poder minimizar este error se ha optado por aumentar el número de simulaciones para cada experimento.

7.2 Tamaño de población

El tamaño de la población es un punto muy importante para la evolución. Un número muy pequeño de individuos puede producir una convergencia (el *fitness* se estabiliza) muy temprana y una falta de diversidad mientras que por el otro lado un gran número de individuos puede tardar mucho en converger y aumentar el margen de error debido al alto coste de cómputo.

Dado que el algoritmo de selección puede influir en el tiempo de convergencia del algoritmo, realizaremos varias pruebas con distintos tamaños de población y distintos algoritmos de

selección. El tiempo de vida será siempre 30 segundos, el número máximo de generaciones se fijará en 200, el porcentaje de nuevos individuos 20% y el porcentaje de mutación 1%.

Población	Selección	MEAN $f(x)$	MAX $f_i(x)$	FPS
25	Ruleta	456,13	606,18	212,15
25	Torneo-2	578,82	635,20	220,21
50	Ruleta	434,70	631,47	70,25
50	Torneo-2	598,25	664,15	72,31
100	Ruleta	508,69	829,67	25,51
100	Torneo-2	671,98	835,88	26,05
150	Ruleta	528,76	966,84	10,01
150	Torneo-2	706,50	975,01	11,21

Tabla 7.2: Resultados de las pruebas con diferentes tamaños de población y funciones de selección para determinar el número de individuos óptimo.

El estudio (Tabla 7.2) muestra claramente como al aumentar el número de individuos en la población también aumenta su valor fitness medio y máximo. A causa de ello el *frame rate* desciende drásticamente lo cual aumenta el margen de error como se ha determinado anteriormente. Una población entre 100 y 150 es un buen valor de población para conseguir unos buenos resultados.

7.3 Número de generaciones

El número de generaciones empleadas para estudiar la evolución de un AG es muy importante ya que poder determinar en qué momento nuestra población se estabiliza y no evoluciona más puede ahorrarnos mucho tiempo de estudio.

Un número muy bajo de generaciones puede ocasionar que el algoritmo termine demasiado pronto y aún exista margen de mejora. Por otro lado, un número muy elevado puede llevar muchísimo tiempo de ejecución ($30s * 500generaciones = 4h$) sin mejoras en los resultados. De ahí la importancia de determinar cuál es la generación en la que suele dejar de mejorar el algoritmo.

En este estudio se variará el número máximo de generaciones así como los algoritmos

de cruce. El número de individuos se fijará en 100, el tiempo de vida en 30 segundos, el porcentaje de nuevos individuos en 20%, el porcentaje de mutación en 1% y la función de selección escogida es la selección por torneo binario.

Los valores a tener en cuenta serán el fitness medio de la ejecución, la mediana y el punto donde la población comienza a estabilizarse.

En la Tabla 7.3 se observan los resultados con un valor de 25 generaciones máximas cuyo tiempo de ejecución es ≈ 12 minutos y 30 segundos:

Operador de cruce	MEAN $f(x)$	MEDIAN $f(x)$	Convergencia
One point	424,96	473,98	No converge
Average	178,19	188,30	No converge
Aritmético	222,51	238,58	No converge
Heurístico	250,27	254,24	No converge

Tabla 7.3: Resultados de las pruebas con 25 generaciones máximas y diferentes operadores de cruce para estudiar sus valores y punto de convergencia.

En la Tabla 7.4 se observan los resultados con un valor de 50 generaciones máximas cuyo tiempo de ejecución es ≈ 25 minutos:

Operador de cruce	MEAN $f(x)$	MEDIAN $f(x)$	Convergencia
One point	445,84	495,36	No converge
Average	186,75	187,82	No converge
Aritmético	280,63	296,88	No converge
Heurístico	315,48	332,14	No converge

Tabla 7.4: Resultados de las pruebas con 50 generaciones máximas y diferentes operadores de cruce para estudiar sus valores y punto de convergencia.

En la Tabla 7.5 se observan los resultados con un valor de 100 generaciones máximas cuyo tiempo de ejecución es ≈ 50 minutos:

Operador de cruce	MEAN $f(x)$	MEDIAN $f(x)$	Convergencia
One point	493,38	519,58	~70 generaciones
Average	214,57	217,85	~60 generaciones
Aritmético	271,65	283,53	~60 generaciones
Heurístico	376,66	377,43	No converge

Tabla 7.5: Resultados de las pruebas con 100 generaciones máximas y diferentes operadores de cruce para estudiar sus valores y punto de convergencia.

En la Tabla 7.6 se observan los resultados con un valor de 200 generaciones máximas cuyo tiempo de ejecución es ≈ 100 minutos:

Operador de cruce	MEAN $f(x)$	MEDIAN $f(x)$	Convergencia
One point	671,98	739,65	~70 generaciones
Average	241,93	244,46	~60 generaciones
Aritmético	325,74	332,11	~60 generaciones
Heurístico	458,95	428,66	~170 generaciones

Tabla 7.6: Resultados de las pruebas con 200 generaciones máximas y diferentes operadores de cruce para estudiar sus valores y punto de convergencia.

Los resultados muestran claramente como al aumentar el número de generaciones también aumentan tanto el fitness medio de la ejecución como su mediana. También nos muestra como algunos operadores de cruce tardan más tiempo en estabilizar la población que otros.

Si el tiempo no es un problema, sin duda que al aumentar lo máximo posible las generaciones ayudará a conseguir valores más fiables pero por otro lado se ha demostrado que entre las 100 y 200 generaciones se puede lograr estabilizar el resultado final.

7.4 Porcentaje de nuevos individuos

El porcentaje de nuevos individuos es un parámetro estrictamente ligado a la función de selección. Con este parámetro se determinará cuántos individuos deseamos que pasen a la siguiente generación o cuántos preferimos que se generen nuevos.

La teoría dice que un valor muy bajo puede generar que el algoritmo tarde en exceso en

converger pero por otro lado un valor muy grande generará que el algoritmo no evolucione y los resultados sean prácticamente fruto de la aleatoriedad.

Para estudiar la importancia de este parámetro vamos a realizar varias pruebas con diferentes porcentajes de nuevos individuos, diferentes tipos de selección y dejando fijados el resto de parámetros: 100 individuos, 30 segundos de vida, 200 generaciones máximas, el porcentaje de mutación al 1% y operador de cruce *one point*.

Porcentaje nuevos individuos	Selección	MEAN $f(x)$
1%	Ruleta	239,03
1%	Torneo-2	257,21
10%	Ruleta	514,68
10%	Torneo-2	577,51
20%	Ruleta	508,69
20%	Torneo-2	671,98
50%	Ruleta	376,55
50%	Torneo-2	609,42
75%	Ruleta	440,21
75%	Torneo-2	471,51

Tabla 7.7: Resultados en función del porcentaje de nuevos individuos y la función de selección.

Los resultados muestran (Tabla 7.7) que un porcentaje bajo de nuevos individuos no consigue buenos resultados debido a que necesita un número de generaciones demasiado elevado para estabilizarse. Por otro lado, un número muy elevado nunca consigue estabilizarse y acaba generando valores demasiado aleatorios. Los valores comprendidos entre el 10% y el 50% pueden ser muy útiles para conseguir valores deseados teniendo en cuenta que valores más bajos necesitarán más tiempo para converger.

7.5 Porcentaje de mutación

La mutación es un recurso muy importante en la ejecución del AG ya que es de gran utilidad para proporcionar diversidad a la población y evitar estancamientos en mínimos

locales (Haupt, 2000).

Es uno de los parámetros que más hay que tener en cuenta ya que un valor demasiado elevado de mutación puede ocasionar que el algoritmo nunca se estabilice y un porcentaje muy bajo, como se ha mencionado, puede ocasionar un resultado final falsamente positivo.

Para estudiar la influencia de este parámetro se analizaran varias ejecuciones con valores de mutación distintos y el resto de parámetros fijos: 100 individuos, 30 segundos de vida, 200 generaciones máximas, porcentaje de nuevos individuos al 10%, selección por torneo binario y operador de cruce *one point*.

Porcentaje de mutación	MEAN $f(x)$
0,1%	587,15
0,5%	581,99
1%	671,98
5%	497,41
10%	334,08
25%	184,10

Tabla 7.8: Resultados obtenidos variando el porcentaje de mutación del algoritmo genético.

Como se observa en la Tabla 7.8, los valores bajos de mutación presentan mejores resultados respecto a números más elevados que tienden a empeorarlo. Hay que llevar cuidado con valores muy pequeños ya que aunque puedan proporcionar buenos valores pueden terminar dependiendo en exceso de los valores generados inicialmente.

7.6 Función de selección

Escoger una función de selección que se adecúe al problema planteado es vital a la hora de conseguir buenos resultados. Una función demasiado elitista puede causar la pérdida de diversidad y converger en mínimos locales que conlleven a resultados erróneos. Por otro lado, una función muy poco elitista podría priorizar demasiado peores individuos lo cual puede causar una convergencia muy tardía.

Para determinar qué función de selección se adapta mejor al problema se va a comparar

ambas y estudiar los resultados obtenidos. A excepción de la función de selección, el resto de parámetros estarán fijos para todas las simulaciones: 100 individuos, 30 segundos de vida, 200 generaciones máximas, porcentaje de nuevos individuos al 10%, 1% de mutación y operador *one point*.

Selección	MEAN $f(x)$	MEDIAN $f(x)$	MAX $f(x)$
Ruleta	493,25	501,20	846,93
Torneo-2	584,42	588,77	884,30
Torneo-10	597,49	601,39	967,49
Torneo-50	598,75	594,25	927,71
Torneo-100	627,65	633,60	947,20

Tabla 7.9: Resultados obtenidos variando la función de selección

Los resultados obtenidos (Tabla 7.9) muestran unos mejores resultados conforme el elitismo de la selección aumenta. La selección por ruleta al ser el menos elitista de los 5 deja valores buenos pero algo por debajo al resto (Kumar y Jyotishree, 2012). Por otro lado, la selección por torneo de 100 miembros presenta los mejores resultados de todos a costa de sacrificar la diversidad de la población. Es una opción interesante para buscar más fácilmente un mejor individuo a base de muchas ejecuciones ya que depende en gran medida de los valores iniciales generados.

7.7 Operador de cruce

Como ya se ha mencionado con anterioridad, el operador de cruce es el elemento más importante de un AG. Es por ello que determinar cuál se adapta mejor al problema es muy importante.

Realizaremos pruebas con los cuatro operadores de cruce implementados y el resto de valores fijos: 100 individuos, 30 segundos de vida, 200 generaciones máximas, porcentaje de nuevos individuos al 20%, 1% de mutación y selección por torneo binario.

Cruce	MEAN $f(x)$	MEDIAN $f(x)$	MAX $f(x)$
One point	671,19	671,42	987,59
Average	218,14	208,16	677,16
Aritmético	356,28	370,64	705,33
Heurístico	669,36	6Z66,28	1004.40

Tabla 7.10: Resultados obtenidos variando el operador de cruce

Como se puede observar en la Tabla 7.10, los operadores *one point* y heurístico con los dos que presentan unos mejores resultados. Mientras que el primero consigue, por lo general, resultados globales más estables, el segundo es muy bueno maximizando el fitness y encontrando mejores individuos *top*.

Esto es debido a que el operador heurístico, como se explico en la Sección 6.3.4, tiende a aumentar los valores de los parámetros. Esto es muy útil para maximizar la velocidad de los individuos pero puede causar problemas con las rotaciones de las articulaciones.

Por otro lado el operador media o *average* y el operador aritmético proporcionan resultados muy por debajo en comparación al resto. Esto es debido a que los valores calculados por estos operadores siempre se encuentran en la recta de valores entre ambos padres.

7.8 Resultados de la experimentación

Finalmente, una vez se han determinado los mejores valores para los diferentes parámetros toca poner a prueba el algoritmo para hallar los valores óptimos para los parámetros involucrados en el movimiento del esqueleto (rotaciones y velocidades de las rodillas y caderas).

Basadas en los resultados estudiados anteriormente, las conclusiones extraídas para cada parámetro del algoritmo son los siguientes:

- **Tamaño de población:** Se ha determinado que un número elevado de población ayuda a conseguir resultados más precisos a costa de aumentar el margen de error en algunos casos. Por ello, un valor superior o igual a 100 individuos y menor a 150 pueden ser muy útiles.

- **Número de generaciones:** Un gran número de generaciones ha mostrado mejores resultados respecto a números más bajos. A costa de ello se pierde velocidad en la ejecución de las pruebas debido a sus largos tiempos de simulación. Observando la convergencia de los diferentes operadores de cruce se ha determinado que cualquier número de generaciones igual o por encima de 200 es suficiente para asegurar la convergencia de la simulación.
- **Porcentaje de nuevos individuos:** Las pruebas han dado como resultado que un número extremadamente bajo de nuevos individuos en cada generación hace que el algoritmo no presente buenos resultados. Por otro lado, un número muy elevado tampoco consigue estabilizar el algoritmo. Son los valores entre el 10% y el 20% donde encontramos valores útiles para conseguir buenos resultados.
- **Porcentaje de mutación:** Los estudios realizados han mostrado que un porcentaje bajo de mutación ayuda a conseguir mejores resultados respecto a porcentajes más elevados. Siempre teniendo en cuenta que cuanto más se reduzca ese porcentaje de mutación más aumentará la posibilidad de converger en mínimos locales. Un valor de mutación comprendido entre el 0,1% y el 1% es una buena elección para el objetivo del proyecto.
- **Función de selección:** En cuanto a la función de selección los resultados han indicado que la selección por ruleta da buenos resultados aunque algo por detrás de la selección por torneo. Dentro de la selección por torneo al aumentar los miembros de torneo y por lo tanto aumentar el elitismo, también se observa una mejoría en los resultados.
- **Operador de cruce:** Los operadores que han destacado por encima del resto han sido por una parte el operador *one point* y por otro lado el operador heurístico. El operador *one point* consigue mayor estabilidad mientras que el heurístico es algo mejor buscando el individuo más apto.

Una vez hemos determinado parámetro a parámetro cuáles son más eficaces llega el momento de poner a prueba el AG con los parámetros óptimos para hallar los valores (rotaciones y velocidades de caderas y rodillas) que definen al mejor individuo posible.

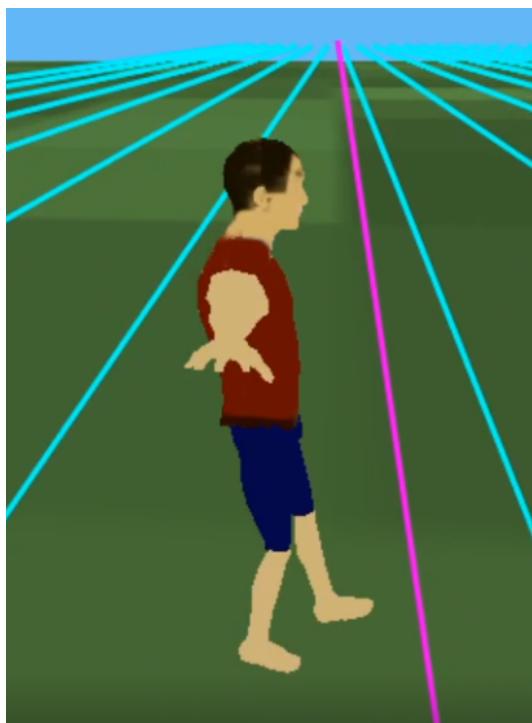
Para ello se ha definido el número de individuos de la población en 100 y el tiempo de vida en 30 segundos. Así es posible conseguir un buen resultado sin aumentar en exceso el margen de error. El porcentaje de nuevos individuos que se ha escogido ha sido 20% y el porcentaje de mutación ha sido finalmente un 1% para que el algoritmo no acabe dependiendo en gran medida de los valores generados inicialmente. La función de selección escogida ha sido el torneo de 100 miembros ya que es la función con más elitismo implementada y la que ha proporcionado mejores resultados. Por último, en cuanto al operador de cruce se van a utilizar el operador heurístico ya que en cuanto a valores máximos consigue mejores resultados mientras que el *one point* consigue una mayor estabilidad que no se busca en este experimento final.

Finalmente, tras muchas pruebas estos son los valores (Tabla 7.11) del mejor individuo generado con un valor fitness de 1004,25:

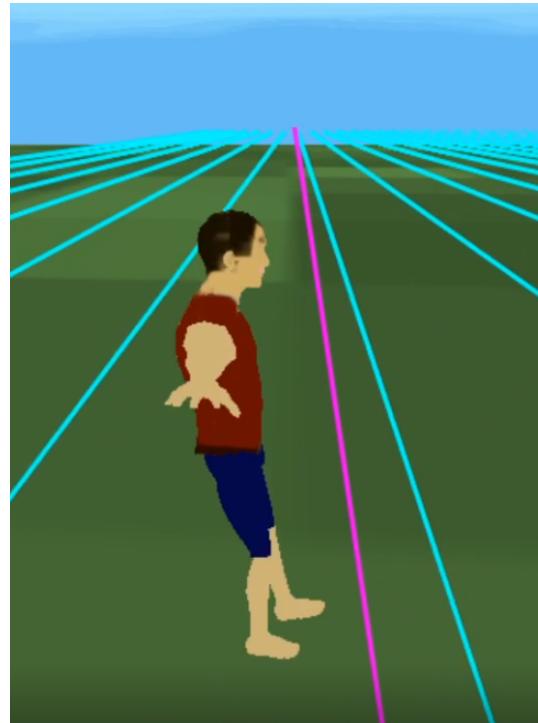
Articulación	Rotación mínima	Rotación máxima	Velocidad
Cadera derecha	-1,5	19,6	79,88
Rodilla derecha	-22	0	79,99
Cadera izquierda	19,8	19,9	79,91
Rodilla izquierda	-0,9	-0,01	79,95

Tabla 7.11: Valores finales del individuo con mejores resultados en los experimentos.

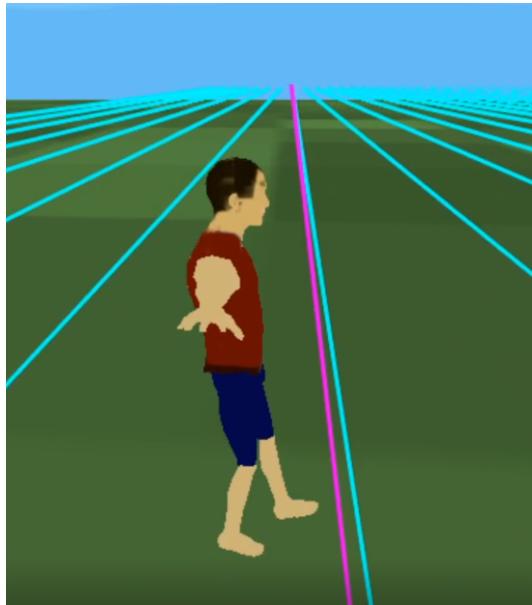
Se observa claramente que las velocidades de las articulaciones tienden a maximizar el valor, como se dijo antes los valores de velocidad están comprendidos entre 30 y 80 ud/frame. Por otro lado las rotaciones mínimas y máximas tienden a reducir su distancia para tardar menos en recorrer la distancia de extremo a extremo. En la Figura 7.1 observamos una secuencia de fotogramas del individuo andando.



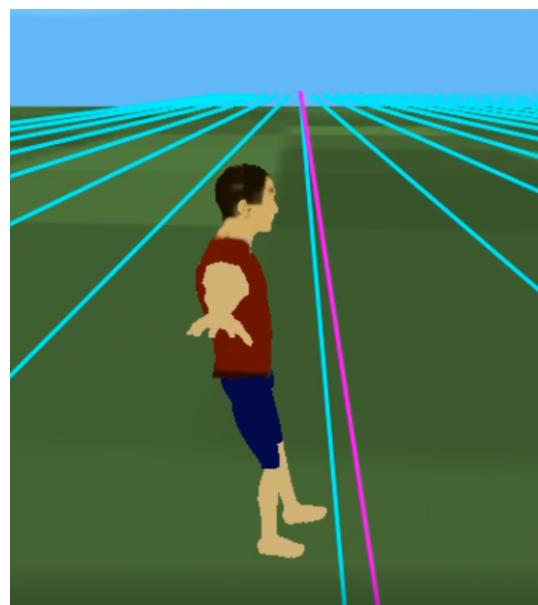
(a) Frame 1



(b) Frame 2



(c) Frame 3



(d) Frame 4

Figura 7.1: Secuencia de frames andando del mejor individuo.

8 Conclusiones y trabajo futuro

8.1 Conclusiones

En el comienzo de este trabajo se comenzó explicando los campos más importantes de la IA, así como sus aplicaciones en proyectos reales. Esto es una manera muy efectiva de desmitificar la IA como un campo únicamente de estudio e investigación y poder comprender su eficacia en problemas cotidianos que necesitan ser resueltos.

Después de la explicación de los diferentes campos que abarca la IA había que escoger uno de ellos a la hora de centrar el proyecto. Los AAGG fueron los escogidos como campo de estudio y desarrollo con el objetivo principal de entender su funcionamiento de una manera visual e intuitiva. Por ello se repasaron algunos proyectos similares al planteado en este trabajo para poder entender su utilidad y el porqué de la elección de los AAGG en este proyecto.

Por ello, se desarrolló una aplicación de escritorio que pueda ser de utilidad para el estudio del impacto de varios de los parámetros que intervienen en un AG. Como siempre, para conseguir comprender su funcionamiento se proporcionó una visualización en tiempo real tanto del experimento como de los datos. Este es un punto importante y diferenciador de gran número de herramientas ya existentes debido a que es una manera de poder visualizar todo aquello que siempre se ha estudiado de manera abstracta.

Finalmente, también se hizo uso de la propia herramienta para realizar una investigación detallada del funcionamiento del AG en este proyecto. Se evaluaron todos los parámetros disponibles y se concluyó cuáles eran los valores óptimos para resolver el problema presentado. De esta manera conseguimos darle solución al problema planteado y al mismo tiempo dejar abierta la posibilidad de hacer pruebas a aquellas personas que quieran probarlo por sí mismas y extraer sus propias conclusiones.

8.2 Trabajo futuro

Dado que el proyecto tenía una fecha límite de entrega hay ciertas funcionalidades que se habría deseado implementar y no se ha podido. También existen otras que aunque no se habían planteado serían una buena continuación para la base implementada hasta ahora.

A pesar de que gran parte del sentido de este proyecto es la representación gráfica, se podría implementar un modo de ejecución sin visualización y que la simulación pudiera hacer uso de la GPU y paralelismos para aumentar el rendimiento y ajustar muchísimo más los resultados finales. Como se ha comentado anteriormente algunas ejecuciones tardan alrededor de dos horas, con este sistema se podrían ejecutar simulaciones mucho más largas y en mucho menor tiempo. Esto abriría la posibilidad de no tener que depender de un ordenador personal para realizar las simulaciones sino optar por servidores remotos de alta computación para ejecutarla.

Otra de las funcionalidades deseadas que no se llegaron a implementar son más funciones de selección y operadores de cruce. Existen multitud de ellos y podrían conseguir resultados diferentes a los estudiados.

Finalmente, remarcar que la buena estructura del proyecto permitiría el estudio de otros tipos de movimientos únicamente implementando su capa de físicas. Con esto se podría estudiar el movimiento de otros tipos de seres articulados como los cuadrúpedos u otros tipos de movimientos como saltos de altura, sprint o movimiento con obstáculos.

Bibliografía

- Ahmed, Z. (2010, marzo). Genetic Algorithm for the Traveling Salesman Problem using Sequential Constructive Crossover Operator. *International Journal of Biometric and Bio-informatics*, 3. doi: 10.14569/IJACSA.2020.0110275
- Bustos, J. C. P., y Vázquez, N. V. (2000). *Algoritmos Genéticos. Aplicación al Juego de las N Reinas*. Descargado de <http://www.it.uc3m.es/jvillena/irc/practicas/06-07/06.pdf>
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., y Kuksa, P. (2011, noviembre). Natural Language Processing (Almost) from Scratch. *The Journal of Machine Learning Research*, 12(null), 2493–2537.
- de Vries, J. (2020). *Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion*. Kendall & Welling. Descargado de <https://www.amazon.es/Learn-OpenGL-programming-step-step/dp/9090332561> (OCLC: 1198376019)
- Eberly, D. (2001, mayo). *Dynamic Collision Detection using Oriented Bounding Boxes*. Descargado de <https://www.geometrictools.com/Documentation/DynamicCollisionDetection.pdf>
- Furqan, M., Hartono, H., Ongko, E., y Ikhsan, M. (2017). Performance of Arithmetic Crossover and Heuristic Crossover in Genetic Algorithm Based on Alpha Parameter. *iosrjournals*. doi: 10.9790/0661-1905013136
- Geijtenbeek, T., Panne, M. v. d., y Stappen, A. F. v. d. (2013, noviembre). Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32(6), 1–11. doi: 10.1145/2508363.2508399

- Gestal, M. (2013, agosto). *Introduccion a los Algoritmos Geneticos.* Descargado de https://www.researchgate.net/publication/237812449_Introduccion_a_los_Algoritmos_Geneticos
- Ha, D. (2017, octubre). *A Visual Guide to Evolution Strategies.* Descargado 2020-11-08, de <https://blog.otoro.net/2017/10/29/visual-evolution-strategies/>
- Ha, D. (2020). *Slime Volleyball Gym Environment.* Descargado de <https://github.com/hardmaru/slimevolleygym>
- Haupt, R. L. (2000, julio). Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors. En *IEEE Antennas and Propagation Society International Symposium. Transmitting Waves of Progress to the Next Millennium. 2000 Digest. Held in conjunction with: USNC/URSI National Radio Science Meeting (C (Vol. 2, pp. 1034–1037 vol.2)).* doi: 10.1109/APS.2000.875398
- Huynh, J. (2009). *Separating Axis Theorem for Oriented Bounding Boxes* (Inf. Téc.). Descargado de <https://www.jkh.me/files/tutorials/Separating%20Axis%20Theorem%20for%20oriented%20Bounding%20Boxes.pdf>
- Jankowski, R. (2020, noviembre). *robertjankowski/ga-openai-gym.*
- Kaya, Y., Uyar, M., y Tekin, R. (2011, enero). A Novel Crossover Operator for Genetic Algorithms: Ring Crossover. *CoRR, abs/1105.0355.*
- Kumar, R., y Jyotishree. (2012). Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms. *International Journal of Machine Learning and Computing, 365–370.* doi: 10.7763/IJMLC.2012.V2.146
- McCarthy, J. (2007, noviembre). *WHAT IS ARTIFICIAL INTELLIGENCE?* Descargado 2020-11-29, de <http://www-formal.stanford.edu/jmc/whatisai/whatisai.html>
- Mitchell, M. (1998). *An Introduction to Genetic Algorithms.* MIT Press.
- Moghaddam, A. H., Moghaddam, M. H., y Esfandyari, M. (2016, diciembre). Stock market index prediction using artificial neural network. *Journal of Economics, Finance and Admin-*

- nistrative Science*, 21(41), 89–93. Descargado 2020-12-10, de <http://www.sciencedirect.com/science/article/pii/S2077188616300245> doi: 10.1016/j.jefas.2016.07.002
- Nazif, H., y Lee, L. S. (2010, enero). Optimized Crossover Genetic Algorithm for Vehicle Routing Problem with Time Windows. *American Journal of Applied Sciences*, 7.
- OpenAI. (2020, noviembre). *openai/gym*. OpenAI.
- Pandey, H. M. (2016, enero). Performance Evaluation of Selection Methods of Genetic Algorithm and Network Security Concerns. *Procedia Computer Science*, 78, 13–18. Descargado 2020-12-10, de <http://www.sciencedirect.com/science/article/pii/S1877050916000065> doi: 10.1016/j.procs.2016.02.004
- Santos, M. (2020, septiembre). *Using Deep Learning AI to Predict the Stock Market*. Descargado 2020-11-13, de <https://towardsdatascience.com/using-deep-learning-ai-to-predict-the-stock-market-9399cf15a312>
- Shukla, A., Pandey, H. M., y Mehrotra, D. (2015, febrero). Comparative review of selection techniques in genetic algorithm. En *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)* (pp. 515–519). doi: 10.1109/ABLAZE.2015.7154916
- Sloss, A. N., y Gustafson, S. (2020). 2019 Evolutionary Algorithms Review. En W. Banzhaf, E. Goodman, L. Sheneman, L. Trujillo, y B. Worzel (Eds.), *Genetic Programming Theory and Practice XVII* (pp. 307–344). Cham: Springer International Publishing. Descargado 2020-12-10, de http://link.springer.com/10.1007/978-3-030-39958-0_16 (Series Title: Genetic and Evolutionary Computation) doi: 10.1007/978-3-030-39958-0_16
- Stroustrup, B. (2013). *A Tour of C++*. Upper Saddle River, NJ: Addison-Wesley.
- Sucar, L. E. (2015). *Probabilistic Graphical Models: Principles and Applications* (1st ed. 2015 ed.). London: Springer London : Imprint: Springer. doi: 10.1007/978-1-4471-6699-3
- Takeyas, B. L. (2011, agosto). *INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL*. Descargado de <http://www.itnuevolaredo.edu.mx/takeyas/Articulos/>

Inteligencia%20Artificial/ARTICULO%20Introduccion%20a%20la%20Inteligencia%
20Artificial.pdf

Umbarkar, D. A., y Sheth, P. (2015, octubre). CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. *ICTACT Journal on Soft Computing (Volume: 6 , Issue: 1)*, 6. doi: 10.21917/ijsc.2015.0150

Lista de Acrónimos y Abreviaturas

AABB axis aligned bounding box.

AAGG algoritmos genéticos.

AG algoritmo genético.

DL Deep learning.

ECS entidad-componente-sistema.

FOV field of view.

GLSL OpenGL Shading Language.

GPU unidad de procesamiento gráfico.

HUD head-Up Display.

IA inteligencia artificial.

ID identificador.

IDE entorno de desarrollo integrado.

IIAA inteligencias artificiales.

ML machine learning.

OBB oriented bounding box.

TFG trabajo final de grado.