Departamento de Informática

PROYECTO FIN DE CARRERA

COMPUTACIÓN EVOLUTIVA APLICADA AL DESARROLLO DE VIDEOJUEGOS: MARIO AI

Autor: Héctor Valero Capataz

Tutor: Gustavo Recio Isasi

Director: Yago Sáez Achaerandio

Leganés, octubre de 2011

Título: COMPUTACIÓN EVOLUTIVA APLICADA AL DESARROLLO DE VIDEOJUEGOS: MARIO AI
Autor: Héctor Valero Capataz
Tutor: Gustavo Recio Isasi
Director: Yago Sáez Achaerandio

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Agradezco a mi novia, mi familia, mis amigos y a mi gata, los momentos de apoyo que me han dado en todo este proceso de realización del proyecto.

También agradecer a Emilio, Gustavo y Yago la ayuda para iniciar y finalizar este proyecto.

Resumen

A lo largo de la historia el juego y los videojuegos han tenido una evolución paralela, actualmente han evolucionado considerablemente hasta convertirse en productos de alta tecnología. En el caso de los videojuegos se han convertido en una industria muy fuerte con un volumen de negocio comparable al cinematográfico. Hasta hace unos años, el desarrollo de los videojuegos se ha centrado en el apartado gráfico y el apartado sonoro, dejando a un segundo plano el comportamiento de los NPCs (Non Player Character). En la actualidad está habiendo una tendencia a centrarse cada vez más en la inteligencia artificial (IA) de los NPCs, que da lugar a numerosos avances e investigaciones relacionadas con la IA con el objetivo de proporcionar a los usuarios de videojuegos un comportamiento variable e impredecible de los NPC tanto en los enemigos como compañeros que se encuentra el usuario a lo largo del videojuego.

El presente proyecto se centra en aplicar alguna de las técnicas de IA existentes a un videojuego, para ello se ha realizado un estudio con diferentes videojuegos en los cuales se pueden aplicar técnicas de IA, seguidamente se hará una elección justificada de uno de los videojuegos analizados. A continuación se determinarán que técnicas son susceptibles a aplicar al videojuego elegido, y se elegirá una de estas técnicas, en esta investigación se ha elegido la técnica de IA Algoritmos Genéticos (AG) dentro de la Computación Evolutiva, y Mario AI como el videojuego a probar. La novedad de esta investigación reside en que se desarrolla un agente autónomo e inteligente capaz de completar varios niveles del videojuego en cuestión, mediante la utilización de los AGs.

El resultado obtenido tras la realización del proyecto ha sido exitoso. Se ha comprobado que los AGs son apropiados en la creación de agentes que son capaces de superar diferentes niveles de Mario AI con dificultad variable. Para comprobar finalmente la calidad de la solución se decide participar en la competición Mario AI Championship 2011, que se celebrará el próximos mes de noviembre, en GIC2011. Al compararse con los resultados de años anteriores se ha verificado que el agente desarrollado en el presente proyecto obtiene puntuaciones mayores que los participantes de la competición en años anteriores.

Palabras clave: Videojuegos, Super Mario Bros, Algoritmos Genéticos, computación evolutiva, inteligencia artificial, Non Player Character, computación bio-inspirada.

Abstract

Throughout history the games and videogames have had parallel evolution, nowadays it have evolved considerably until it becomes high-tech products. Videogames has become a strong industry with a turnover comparable with movie industry. Until recently, the videogames development has focused on the graphics and sound sections, leaving the background the behavior of NPCs (Non Player Character). But now, there is a tendency to focus increasingly on the Artificial Intelligence (AI) of NPCs, so it makes many advances and researches related to AI in order to provide videogames users variables and unpredictable behaviors of enemies and partners NPCs throughout the videogame.

This project focuses on applying some of the existing IA techniques to a videogame, it has made a study with different videogames which can apply these AI techniques, and afterward it has made a justified election of one of the analyzed videogames. Next, it has determined that IA techniques can be to apply to chosen videogame, and it has chosen one of these techniques, this research has chosen the Genetic Algorithms (GA) in Computation Evolutionary as the AI technique, and Mario AI as the videogame to try. The innovation of this research is to develop an intelligent and autonomous agent that is capable of completing various levels of the videogame in question through the use of GAs.

The obtained results after of making the project have been successful. There is evidence that GAs are appropriate in the creation of agents that able to overcome different levels with varying difficulty of Mario AI. To check finally the quality of the solution it has decided to take part in the "Mario AI Championship 2010" competition, to be held the next November in GIC2011. When it has compared with the previous years results, it has verified that the developed agent, in this project, gets higher scores that the participants of the competition in previous years.

Keywords: Videogames, Super Mario Bros, genetic algorithms, evolutionary computation, artificial intelligence, Non Player Character, bio-inspired computing.

Índice general

Índice de figuras

Índice de tablas

Índice de Acrónimos

2D	*2 – Dimensiones*
3D	*3 – Dimensiones*
ADN	*Ácido DesoxirriboNucleico*
AE	*Algoritmo Evolutivo*
AG	*Algoritmo Genético*
AI	*Artificial Intelligence*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
BBAI	*Behavior-Based Artificial Intelligence*
CBR	*Case-Based Reasoning*
CD	*Compact Disc*
CD-ROM	*Compact Disc - Read Only Memory*
CIG	*Computational Intelligence and Games*
CMS	*Construction and Management Simulation*
CPU	*Central Processing Unit*
DK	*Donkey Kong*
DOF	*Degrees Of Freedom*
EDSAC	*Electronic Delay Storage Automatic Calculator*
ENIAC	*Electronic Numerical Integrator And Computer*
FIFA	*Federación Internacional de Fútbol Asociación*
FPS	*First Person Shooter*
IA	*Inteligencia Artificial*
IBM	*International Business Machines*
IEEE	*Institute of Electrical and Electronics Engineers*
IRPF	*Impuesto sobre la Renta de las Personas Físicas*
IVA	*Impuesto sobre el Valor Añadido*
JGAP	*Java Genetic Algorithms Package*
MIT	*Massachusetts Institute of Technology*

NBA	*National Basketball Association*
NES	*Nintendo Entertainment System*
NPC	*Non Player Character*
PC	*Personal Computer*
PG	*Programación Genética*
REALM	*Rule-based Evolutionary Computation Agent that Learn Mario*
RL	*Reinforcement Learning*
RTS	*Real Time Strategy*
RTT	*Real Time Tactics*
SDK	*Software Development Kit*
SVN	*SubVersioN*
TBS	*Turn-Based Strategy*
TBT	*Turn-Based Tactics*
TORCS	*The Open Racing Car Simulator*
TPS	*Third Person Shooter*
UR	*User Requisites*
URL	*Uniform Resource Locator*
VRML	*Virtual Reality Modeling Language*
XML	*Extensible Markup Language*
YOG	*Ysagoon Online Gaming*

Capítulo 1

Introducción y objetivos

1.1 Introducción

El juego es una actividad que ha acompañado a la humanidad durante toda su existencia, desde los primeros homínidos hasta la actualidad. Incluso en muchas especies animales tienen al juego como una actividad rutinaria, sobre todo en sus primeros meses y años de vida.

Una definición de juego, muy aceptada es la de Huizinga, [1]:

> *El juego es una acción u ocupación libre, que se desarrolla dentro de unos límites temporales y espaciales determinados, según reglas absolutamente obligatorias, aunque libremente aceptadas, acción que tiene fin en sí misma y va acompañada de un sentimiento de tensión y alegría y de la conciencia de -ser de otro modo- que en la vida corriente.*

Como era de esperar el juego también ha evolucionado, como el ser humano, pasando de los primeros sonajeros fabricados con vejiga de cerdo, a los videojuegos en la actualidad.

Otra acepción de juego, en la que se han basado los videojuegos es la siguiente:

> *Juego viene del latín: "iocum y ludus-ludere" haciendo referencia a broma, diversión o chiste, de esta manera se define esta actividad tan simple y primaria en el ser humano, la cual consiste en la interacción del mundo real con el fantástico a*

través de escenas, en las que las personas se recrean a sí mismos como héroes, villanos, animales, otra persona, etc., imitando la realidad a partir de lo que observan.

Los videojuegos son programas informáticos o software desarrollado para el entrenamiento general y la diversión, para ello, este software, se apoya en diferentes dispositivos electrónicos, como las videoconsolas, los ordenadores, las máquinas árcade, o los dispositivos móviles (p. ej., el teléfono móvil), estos dispositivos se conocen también como "plataformas".

Una propiedad fundamental de los videojuegos es que permiten interactuar con una o más personas, las cuales pueden visualizar y modificar el entorno y el estado del videojuego. La mayoría de las plataformas poseen una pantalla visual, que proporciona dicha información del entorno y estado del juego, estas pantallas visuales, pueden ser desde monitores, televisores, hasta pequeñas pantallas de los dispositivos móviles. Por otro lado, estas plataformas nos ofrecen los llamados controladores del videojuego, que varían según la plataforma, éstos permiten, a los usuarios de los videojuegos, modificar e interactuar con el estado y el entorno del videojuego, y pueden ser joysticks, mandos de controlador, pads, teclado y ratón, e incluso, en algunos casos, la propia pantalla, si ésta es táctil.

1.2 Evolución histórica de los videojuegos

De igual forma, que los juegos, los videojuegos han sufrido una gran evolución tecnológica desde sus inicios:

- Durante la **década de 1940**, al final de la 2º Guerra Mundial, se construyeron las primeras computadoras programables como el ENIAC. En 1948, Alan Turing, junto con D. G. Champernowne, escribieron el primer programa de ajedrez.

- En la **década de 1950**, se desarrolló la primera versión del "Tres en Raya" o "**OXO**" por Alexander Sandy Douglas, en la EDSAC de la Universidad de Cambridge, representado en una pantalla de 35x16 píxeles. El programa tomaba las decisiones correctas en cada momento del juego según el movimiento realizado por el jugador, que lo hacía mediante un dial telefónico de rueda que incorporaba la computadora EDSAC.

  William Nighinbottham en 1958 creó un juego llamado Tenis Para Dos (**Tennis for two**) usando un osciloscopio de laboratorio, consistía en interceptar una bola que cruzaba la pantalla moviendo una línea que hacía de paleta. En 1972 fue comercializado por Atari con el nombre de Pong con un gran éxito, por lo que fue el primer videojuego de la historia.

- En la **década de 1960**, Steve Russell escribió **Space War** en 1961 en una computadora PDP-1 en el MIT, Instituto Tecnológico de Massachusetts, la cuna de la cultura hacker justamente en aquella época. El juego era para dos

jugadores, cada uno manejaba una nave espacial e intentaba disparar a la otra, además había en la pantalla una estrella cuya gravedad atraía a las naves hasta destruirlas si las alcanzaba. El código de Spacewar llegó a numerosas computadoras en otras universidades y es el primer videojuego para ordenador de la historia.

- En la **década de 1970**, Nolan Bushnell (fundador de Atari) empezó a crear un aparato especializado en hacer una sola cosa: presentar el juego en pantalla en vez de tener un ordenador mucho más complicado (reduciendo considerablemente el coste en componentes del juego) para su nueva versión del juego Space War, llamado **Computer Space**. Posteriormente, junto con Bill Nutting crearon una carcasa con un toque muy futurista para el juego y un manual de instrucciones bastante complicado. Seguidamente instalaron la máquina en un bar del campus de la Universidad de Stanford para que fuese probada por los estudiantes, que obtuvo un éxito relativo entre los estudiantes.

  En 1972 Nolan Bushnell funda en Estados Unidos, junto con Ted Dabney, la empresa Atari. Ese mismo año presentan una máquina recreativa de monedas con el juego **Pong**, versión de Tennis For Two. Obtuvo un gran éxito y en 1975 Atari ofrece la Atari Pong, videoconsola doméstica que se conectaba a un aparato de TV y permitía jugar al juego Pong en el hogar.

  La mayor parte de los videojuegos de éxito de la época (Space Invaders, Asteroids,…) se comercializaban a través de las máquinas árcade que funcionan con monedas y se ubican, mayormente, en las salas recreativas lo que propició la difusión y popularidad de este nuevo medio de entretenimiento.

- En la **década de los 80**, en la primera edad de oro de los videojuegos, las máquinas recreativas, consolas y ordenadores ofrecen mayor calidad en los entornos gráficos y sonoros y aparecen las primeras consolas portátiles. Además la introducción del ordenador personal en los hogares popularizó enormemente este nuevo entretenimiento. En 1980, Nintendo presenta Nintendo of America Inc. en Nueva York, y empiezan a distribuir las series Game & Watch en todo el mundo.

  La irrupción del color en el mundo de los videojuegos tuvo lugar en 1979 con Galaxian, un título de la japonesa Namco que, siguiendo la estela de Space Invaders, resultó tremendamente popular al tiempo que marcaba la evolución del género. Otros juegos de Atari, combinaban los gráficos vectoriales con el uso del color, al igual que Defender (Williams, 1981), el primer mata-marcianos horizontal y uno de los títulos más rentables de la historia.

  En 1980, se lanza al mercado **Pac-Man**, es un videojuego árcade en el cual se controla un círculo amarillo al que le falta un sector para simular una cabeza con boca. En el juego, aparecen laberintos donde Pac-Man debe comer puntos pequeños, puntos mayores y otros premios con forma de frutas y otros objetos. El objetivo es comer todos los puntos de la pantalla

(llamados pills en inglés), momento en el que se pasa al siguiente nivel o pantalla, aunque para evitarlo hay cuatro fantasmas que recorren el laberinto para intentar comerse a Pac-Man. Se convirtió en un fenómeno mundial en la industria de los videojuegos, llego a tener el Récord Guiness del videojuego de árcade más exitoso de todos los tiempos con un total de 293.822 máquinas vendidas de 1981 a 1987.

En 1981, nace **Donkey Kong**, en manos de Shigeru Miyamoto (diseñador y productor de videojuegos de Nintendo), en donde controlamos a un fontanero llamado Jumpman (más adelante se le conocerá como Mario), que intentará rescatar a su novia de las garras de un gorila llamado Donkey Kong, esquivando todos los objetos que le lanza el gorila y llegar a quitar unos tornillos amarillos para que caigan todas las plataformas y así hacer que DK caiga al vacío y rescatar definitivamente a su novia.

En 1983, aparece **Mario Bros** para las máquinas recreativas, gracias Shigeru Miyamoto y Gunpei Yokoi, que es el personaje que se controla en Donkey Kong, llamado Jumpman, y ahora en el videojuego paso a ser Mario, también aparece su hermano Luigi. El objetivo del juego es derrotar a todos los enemigos en cada nivel. Los dos extremos de cada nivel tienen una característica mecánica que le permite al jugador salir por la izquierda y aparecer a la derecha, y viceversa. Cuantos más niveles cruce Mario, mayor será la dificultad y el número de enemigos que aparecen. Fue el primer videojuego de plataformas de desplazamiento lateral, aportando nuevas formas de juego y de control del personaje hasta entonces no exploradas. Además en la actualidad, sigue siendo superventas y todo un símbolo para una generación de jugadores que han seguido esta saga, atrapadas por el sentido del humor y la ternura de este personaje así como por la acción trepidante de sus aventuras.

En 1985, Nintendo lanza **Super Mario Bros**, que fue el juego que popularizó al personaje Mario, convirtiéndolo en el ícono principal de Nintendo, y uno de los personajes más reconocidos de los videojuegos. Además, presentó por primera vez a la Princesa Peach Toadstool, Toad, Bowser, entre otros personajes. Este juego es considerado el primer videojuego de plataformas de desplazamiento lateral de Nintendo y se ha convertido en un hito debido a la trascendencia de su diseño y papel en la industria de los videojuegos.

También en ese mismo año se crea **Tetris**, fue inventado por Alexey Pazhintov (con la ayuda de Dmitry Pavlovsky y Vadim Gerasimov) durante 1985, inspirado en un juego árabe de pentaminós, cuando estaba trabajando en la Academia de Ciencias de Moscú. Desde la parte superior de la pantalla caen 7 tipos de piezas llamadas Tetriminos y el jugador las debe rotar y colocar de forma que se formen líneas completas. Ha obtenido y tiene una tremenda influencia e impacto popular entre las nuevas generaciones de jugadores y desarrolladores.

- En la **década de los 90**, junto a la consolidación definitiva de la industria y a cierto estancamiento en la creatividad de los creadores de videojuegos,

debido en parte a la estabilización que habían sufrido los distintos géneros, esta década se caracterizó por el declive definitivo de los ordenadores de 8 bits y la llegada de una nueva generación de computadoras más avanzadas, primero de 16 bits, y posteriormente de 32 bits. Aparecen consolas como la Super Nintendo, la cual arrasaría en todo el mundo, gracias en parte al legado de la anterior videoconsola NES y además teniendo una de las máximas rivalidades que ha habido en la historia con la consola Mega Drive.

En 1990 aparece una nueva versión del Super Mario Bros, el **Super Mario World**. Presentaba una gran mejoría en gráficos, sonido y jugabilidad respecto a sus antecesores. Además apareció un nuevo personaje (Yoshi) y nuevas habilidades para Mario, incluyendo la famosa capa. También hay que decir que fue de los primeros juegos que compensaba al jugador al completarlo al 100% (descubrir todos los secretos).

En 1991, Sega con su consola Mega Drive, no conseguía hacer frente a Nintendo, por lo que decidieron desarrollar un videojuego similar al Super Mario, de ahí nació **Sonic**, que tenía que salvar a sus amigos animales secuestrados por el Doctor Robotnik, que quiere utilizarlos como fuente de energía para su ejército de robots con la intención de conquistar el planeta Mobius. Para conseguir su objetivo, Sonic tendrá que recolectar las 7 Esmeraldas del Caos (Chaos emeralds). Sus características principales son una gran velocidad y dinamismo.

También en ese año, aparece el primer videojuego de la saga **The Legend of Zelda**, que tiene como protagonista a joven guerrero llamado Link con misión se rescatar a la Princesa Zelda.

En 1992, sale a la venta el primer título de la saga **Mortal Kombat**, el juego constaba de siete luchadores y la misión del jugador es conseguir ser el campeón del torneo Mortal Kombat, tiene una gran variedad de golpes y el juego es tremendamente violento, pudiendo realizar un "fatality" (para matar al rival), que resultaba violento y gore.

En 1993, llega a las videoconsolas y PC, el videojuego **Doom**, El protagonista era un marine de los Estados Unidos que, después de golpear a un superior por ordenar disparar contra civiles es mandado a una base de Marte donde se desenvuelve toda la acción. El objetivo de cada nivel es ir encontrando el interruptor de salida del sector correspondiente. Técnicamente era muy basto para la época con muy buenos modelados y con un mapa enorme. El motor gráfico del juego fue diseñado por John Carmack.

Otra saga muy importante, en el mundo de los videojuegos, aparece en este año, la saga **FIFA**, poseía una novedosa vista isométrica, que estrenaba, se podía controlar distintas selecciones nacionales (los jugadores no tenían los nombres reales) y se podía disputar todo tipo de torneos y ligas. Además también tenía modos de entrenamiento o partido de exhibición. Tenía gráficos normales pero con una gran cantidad de animaciones. Y poseía un novedoso sistema de repeticiones y una buena inteligencia artificial, por ejemplo, las defensas se mantendrán en una línea más profunda con el fin de

tener una ventaja sobre los atacantes, por otro lado, los porteros han sido mejorados para hacer más difícil marcar goles.

En 1994, aparecen las primeras consolas de nueva generación, como la consola de videojuegos de Sony, PlayStation, o la de Sega, Sega Saturn, Con CPUs de 32 bits, 2Mb de Ram y lector de CD.

En 1996, Nintendo saca su nueva consola, de 64 bits, más avanzada tecnológicamente que las anteriores, pero siguió utilizando cartuchos para almacenar los videojuegos, como sus predecesoras.

La nueva consola de Nintendo, Nintendo 64, trajo consigo una nueva versión del juego Super Mario Bros, **Super Mario 64**, el cual supuso un cambio radical, al pasar de los gráficos en 2D a los gráficos en 3D y con un excelente control del personaje. Llegó a vender más de 12 millones de copias y se ha llegado a considerar uno de los mejores juegos de la historia de los videojuegos, como se indica en [2].

En 1997 la saga **Final Fantasy** se embarca en la consola de Sony, PlayStation, su temática de juego de rol, que combina lo fantástico con la ciencia ficción, la utilización de un sistema de batalla basado en turnos y controlados por los menús, además de unos bonitos gráficos en tres dimensiones la convirtió en una de las sagas de videojuegos con más éxito de ventas y de seguidores.

En 1998 aparece uno de los mejores juegos de las historia, es el primer título en 3D de la saga Zelda, **Legend of Zelda: Ocarina of Time**, que fue diseñado por Shigeru Miyamoto. En ese mismo año, también aparece una nueva saga de videojuegos, que marcaron una época, el **Metal Gear Solid**, en la consola PlayStation, que se convertirá en unos de los mejores juegos de espionaje y acción.

- En la **década actual**, si hay algo que caracteriza la industria del videojuego del siglo XXI es su transformación en una industria multimillonaria de dimensiones inimaginables pocos años antes, además aparecieron nuevos soportes y entornos para jugar con videojuegos como los teléfonos móviles, Internet, o nuevas consolas portátiles.

  En el año 2000 aparece la nueva consola de Sony, la **PlayStation 2**. Al cabo de un año aparece la nueva consola de Nintendo, la **GameCube**, junto también con la nueva consola de Microsoft, la **XBOX**.

  En el 2004 Nintendo lanza al mercado una de las portátiles más vendidas de la historia: **Nintendo DS**.

  Entre los años 2005 y 2006 aparecerán las consolas de última generación de las tres compañías más importantes, en abril de 2005 llega al mercado la nueva consola de Microsoft, la **XBOX 360**, con un procesador Xenon de 3 núcleos a 3,2 GHz cada uno. A finales del año 2006 sale al mercado la tercera consola de Sony, la **PlayStation 3**, que es la consola más potente del

mercado, posee una CPU diseñada por IBM y Thosiba, llamada “Cell” con 7 núcleos a 3,2GHz cada uno, además también incluye un dispositivo BlueRay, pero debido a esto, es la consola más cara de las tres. Y la última consola a destacar, es la nueva consola de Nintendo, **Wii**, con la que apuestan más en la manera de jugar con su mando revolucionario que en la potencia de la misma consola. La principal característica es, pues, el control inalámbrico que lleva incorporado (llamado Wiimote), capaz de detectar el movimiento y rotación en un espacio de tres dimensiones (aparte incluye vibración y un altavoz).

1.3 Objetivos

En el apartado anterior, hemos visto la rápida evolución de los videojuegos, a lo largo del siglo XX y XXI, la evolución más impactante es la tecnológica, pasando de unos gráficos sencillos y pobres con pocos colores, a unos gráficos en tres dimensiones y con sonido envolvente.

Pero también hay otra evolución igual o incluso más importe que la tecnológica, y es la evolución de la inteligencia artificial o comportamiento de los personajes de los videojuegos que no son controlados por los jugadores, los llamados “non-player characters” (NPCs).

Hasta la década de los 90, los videojuegos carecían de inteligencia artificial, o bien porque eran videojuegos que no tenían personajes no controlados (NPCs), ya que se jugaba contra otro u otros oponentes humanos. O bien porque los NPCs siempre tenían el mismo comportamiento, es decir, los movimientos de los enemigos estaban basados en patrones almacenados, o por otro lado, estos movimientos venían determinados por el jugador, es decir, el NPC aplica unas reglas u otras según el comportamiento del jugador.

El problema de estos videojuegos es que acaban siendo abandonados por el jugador, ya que no se siente retado por el juego, debido a que el resulta monótono y predecible.

Ya en la década de los 90, aparecen nuevos avances tecnológicos, nuevas consolas más potentes, el abaratamiento de los dispositivos electrónicos, y una mayor comercialización de los videojuegos. Esto ayuda a que se invierta más en el desarrollo de videojuegos, por lo que se empiezan a tener más en cuenta las técnicas o herramientas de inteligencia artificial a la hora de desarrollar videojuegos, como las máquinas de estados finitos.

La aparición de los juegos de estrategia en tiempo real pone a prueba la IA con muchos objetos, información incompleta, problemas de “búsqueda de caminos”, decisiones en tiempo real, planificación económica, etc. En los últimos videojuegos se están utilizando los métodos de IA “bottom-up” (de abajo arriba), tales como comportamientos emergentes, y evolución de las acciones de los jugadores.

Más adelante, en el capítulo 2, se proporcionará un estudio más detallado de la aplicación de la Inteligencia Artificial en diferentes videojuegos.

Como se ha visto, la mayor parte de la utilización de las técnicas y herramientas de la IA, están enfocadas a los personajes no controlados (NPCs) por el usuario. Pero en este proyecto se quiere dar un nuevo enfoque a estas técnicas de IA en los videojuegos, en la que en vez de estar en contra del usuario o jugador, esté a favor del usuario, e incluso tome el papel del usuario en el videojuego. Es decir, este nuevo enfoque pretende que las técnicas o herramientas de la IA controlen al personaje principal del juego, e intenten completar de forma autónoma y auto-aprendida los diferentes niveles u objetivos en los que se compone un videojuego.

Aunque a priori, este enfoque parece que no beneficie ni afecte a los muchos consumidores de videojuegos, sí lo hace, porque permite evaluar la dificultad de un juego, es decir, se podrán evaluar las diferentes técnicas de IA utilizadas en los NPCs de forma más eficiente y con muchos menos recursos, tanto económicos (al contratar probadores de videojuegos) y de tiempo (se podrá evaluar a los agentes las 24 horas del día, los 365 días del año).

Por lo tanto, el objetivo fundamental del proyecto, es diseñar y desarrollar un agente inteligente, mediante técnicas de IA, que sea capaz de completar uno o varios niveles de un videojuego elegido. Para ello se necesitarán realizar diferentes tareas para desarrollar el presente proyecto, por lo tanto, estas tareas deberán cumplir los siguientes objetivos parciales:

- ✓ Realizar un estudio de los diferentes tipos de videojuegos existentes, que pueden ser modificados para desarrollar un agente que pueda jugar de forma autónoma.

- ✓ Analizar, por cada videojuego, que técnicas de IA funcionan mejor para desarrollar un agente que complete uno o varios niveles.

- ✓ Elegir uno de los videojuegos analizados en el punto anterior, para desarrollar un agente inteligente que sea capaz de jugar de forma óptima, mediante una de las técnicas de inteligencia artificial elegidas, tras un análisis de las mismas.

- ✓ Diseñar un agente basado en una de las técnicas de IA analizadas, que sea capaz de jugar de forma autónoma uno de los videojuegos examinados además de participar en alguna competición ya organizada para el videojuego elegido, e intentar sacar unos buenos resultados, como clasificarse entre los tres primeros y sí es posible ganar dicha competición.

1.4 Estructura de la memoria

A lo largo de este documento, se pueden encontrar las siguientes secciones:

- **Capítulo 1 – Introducción y Objetivos**: se expone la motivación del proyecto y los objetivos que se han perseguido durante su desarrollo.

Además, se ofrece una visión general de los contenidos de la memoria del proyecto fin de carrera.

- **Capítulo 2 – Estado del arte**: en este capítulo se ofrece una descripción general de varios de los aspectos relacionados con el presente proyecto. Se definirán brevemente las diferentes categorías existentes en los videojuegos y se expondrán varios videojuegos que pueden ser objetivo del posterior estudio para implementar un agente que juegue, mediante las técnicas de IA, así mismo, se expondrán una serie de técnicas de IA que podrían funcionar en este agente de forma razonada. Gracias al estudio, se tendrá una idea del nivel actual existente y contaremos con una base firme desde la que comenzar el estudio.

- **Capítulo 3 – Análisis y Diseño del Sistema Propuesto**: en este capítulo, se realizará un análisis del videojuego elegido, entre los comentados en el apartado anterior. Se expondrá una serie de razones que han llevado a elegir ese videojuego, por lo tanto, se detallarán los requisitos que deberá cumplir el sistema a construir. Tras el paso anterior, en el que se expondrá que mejor técnica de inteligencia artificial funciona en el caso concreto, se hará una introducción de las características más importante y su funcionamiento, de la técnica elegida. Una vez analizado el entorno de trabajo y definidos los objetivos que se pretenden alcanzar, en este capítulo también contendrá toda la información relativa al desarrollo del proyecto, detallando el proceso seguido para la realización de cada uno de los prototipos de los agentes implementados.

- **Capítulo 4 – Diseño de Experimentos y Resultados**: En esta sección se incluirán todos los experimentos realizados para comprobar las ventajas e inconvenientes que presenta cada uno de los prototipos desarrollados durante la fase de desarrollo. Para ello, se formularán hipótesis, seguidas de un conjunto de pruebas y un análisis de resultados que se encargará de confirmar o rechazar las conjeturas realizadas inicialmente para cada una de las pruebas.

- **Capítulo 5 – Conclusiones y Líneas Futuras**: En este apartado, se reflexionará acerca de la consecución (o no) de los objetivos globales inicialmente planteados para este proyecto y qué problemas se han encontrado al realizarlo. Finalmente, se citarán diversas líneas de investigación que en un futuro y, teniendo como base el proyecto actual, puedan resultar interesantes de cara a ampliar el estudio realizado en el mismo.

- **Anexos**. Sección que contendrá información adicional del proyecto, referenciada desde apartados anteriores, como la planificación, el presupuesto, porciones de código, capturas de pantalla, etc.

- **Referencias**. Fuentes bibliográficas a las que se ha hecho referencia en el texto.

Capítulo 2

Estado del Arte

2.1 Introducción

A lo largo de este capítulo se presentará una revisión del estado del arte en lo referente a los videojuegos. En el capítulo anterior se introdujo los conceptos de juego y videojuegos, además de su evolución a lo largo de su historia. Por lo tanto, el siguiente paso a dar, es analizar los videojuegos más destacados en la actualidad, seguidamente se expondrá diferentes investigaciones realizadas sobre aplicaciones de técnicas de inteligencia artificial a diferentes videojuegos.

Para facilitar al lector la lectura y comprensión del capítulo, se ha estructurado el capítulo en dos apartados. En el primer apartado se ha realizado una análisis de los videojuegos existentes en la actualidad, para ello se ha clasificado dichos videojuegos en géneros y se indicará las características principales de cada género, y así tener una visión global del mundo de los videojuegos en la actualidad. En el segundo apartado, se expondrá un estudio sobre diferentes videojuegos que son adecuados a aplicar alguna técnica de inteligencia artificial.

2.2 Clasificación por Géneros de Videojuegos

En esta sección se realizará un breve resumen de los principales géneros en los que se dividen los videojuegos. Igual que en las artes escénicas, el género en los videojuegos se utiliza para clasificar a un videojuego según sus características, como su representación gráfica, el tipo de interacción entre el jugador y la máquina, la ambientación, la temática y su sistema de juego o formato. Aunque también es habitual que algunos videojuegos incluyan varios géneros, en algunos casos se fusionan formando nuevos géneros como los de ***survival horror***, que mezclan elementos de *aventura gráfica* con otros de *shoot 'em up*.

2.2.1 Videojuegos de Aventura

Los videojuegos de Aventura se caracterizan por involucrar al jugador como el protagonista del juego dentro de una historia interactiva, en la que se tiene que explorar los escenarios y resolver los diferentes enigmas o puzles que proponga el juego, para avanzar en el juego, además el jugador tendrá que interactuar con otros personajes del juego controlados por la computadora, que proporcionarán información importante para avanzar en la historia. Para mantener el interés en el juego se debe tener un equilibrio entre una buena historia y la capacidad para que sea el jugador quien la viva a través de sus acciones. Este género de videojuegos fomenta la compresión, la observación y la memoria.

Casi todos los juegos de aventura están diseñados para un solo jugador, ya que el énfasis del juego está en la historia y el personaje hace que el diseño multi-jugador sea difícil. Dentro de éste género podemos encontrar varios subgéneros como:

- **Aventuras conversacionales**: Los primeros videojuegos de aventura eran textuales (aventuras textuales, aventura conversacional o ficción interactiva). En estos, el jugador utiliza el teclado para introducir órdenes como "coger la cuerda" o "ir hacia el oeste" y el ordenador describe lo que pasa. Cuando el uso de gráficos se generalizó, los juegos de aventura textuales dejaron paso a los visuales (por ejemplo, con imágenes del lugar presente) que substituyeron de este modo las descripciones por texto, que se habían vuelto casi superfluas. Estos juegos de aventura con gráficos seguían, no obstante, sirviéndose de la introducción de texto. Ejemplos de videojuegos son, *Hi-Res Adventure*, *Mystery House*, *Time Zone* y *King's Quest I*.

- **Aventuras gráficas**: Estos juegos de aventura utilizan los gráficos para transmitir el entorno al jugador. Los juegos estandartes de las aventuras gráficas pueden tener una variedad de tipos de entradas, desde analizadores de texto a interfaces de pantallas táctil. Estas aventuras de "apuntar y hacer clic" ("Point and click", en inglés) son un tipo común de aventuras gráficas, en la que el jugador usa un puntero, normalmente el ratón, para interactuar con el entorno y resolver puzles. Este método de entrada sigue siendo muy popular en el género, y se adapta bien a la interacción con el entorno, en contraposición a los sistemas de control directo. Videojuegos característicos

de este subgénero son, las sagas de *Indiana Jones*, las sagas de *Monkey Island*, *Myst* y *Broken Sword*.

- **Survival horror**: Son aventuras de terror donde el principal objetivo es sobrevivir, este subgénero se caracteriza por combinar características de aventuras gráficas con elementos de suspense y acción envuelto en una atmósfera de terror psicológico donde lo que prima es la supervivencia, y escapara del lugar en donde transcurre el juego. Títulos destacados, como la serie *Wolfenstein*, la serie *Silent Hill*, y la serie *Alone in the Dark*.

- **Hit n' Run**: También llamados juegos de "golpea y corre", son videojuegos en los que se puede realizar cualquier cosa que se quiera, e interactuar con casi todo lo que hay en el entorno. El jugador y el juego avanzan mediante la realización de misiones.

- **Juegos de Rol**: Este subgénero de videojuegos se caracteriza por la interacción con el personaje, dentro de una historia ficticia amplia e intensa, y de una evolución del personaje, dependiente de las decisiones tomadas por el jugador a medida que transcurre la historia y el juego. Para lograr que se avance en el juego generalmente se hace que el jugador se sumerja en una aventura donde irá conociendo nuevos personajes, explorando el mundo para ir juntando objetos necesarios, experiencia, aliados e incluso magia. La inclusión del CD-ROM permitió contar la historia más detallada, utilizando videos de duración media que hacen que el jugador se sienta como dentro de una película. Por lo tanto, estos juegos otorgan al jugador una gran libertad de acción y de interacción, pudiendo elegir en todo momento que acciones realiza para conseguir evolucionar su personaje y llegar al objetivo propuesto. Ejemplos de videojuegos más populares de este subgénero son, la saga *Final Fantasy*, la saga *Zelda of Ocarina*, *Dungeons & Dragons*, etc.

2.2.2 Videojuegos de Deportivos

Los videojuegos deportivos trasladan diversos deportes tradicionales al universo virtual con la posibilidad de jugarlos en distintas modalidades pero siguiendo las mismas reglas del juego original. Casi todos los deportes conocidos han sido recreados con un videojuego, incluyendo béisbol, fútbol, fútbol americano, boxeo, lucha libre, críquet, golf, baloncesto, hockey sobre hielo, bolos, rugby, caza, pesca, etc.

Son videojuegos capaces de reproducir con gran realismo los equipos y estrellas deportivas del momento, estadios, circuitos, campeonatos, además en los últimos años, se está haciendo un gran esfuerzo en mejorar el realismo de los movimientos que pueden realizar los jugadores.

Existen distintas modalidades de juego, en algunas los jugadores deben competir contra la máquina u otros jugadores, y en otras, los jugadores adquieren el rol de manager. Las primeras fomentan las mismas habilidades que los juegos de acción, ya que implica desafíos físicos y tácticos, y pone a prueba la precisión y exactitud del jugador; y

la segunda, habilidades de ámbito cognitivo como el análisis y la gestión de la información.

Los títulos más característicos de este género, pueden ser, *FIFA*, *Pro Evolution Soccer*, *NBA Live*, *NBA 2K*, *Virtual Tennis*, etc.

2.2.3 Videojuegos de Estrategia

Los videojuegos de estrategia forman un género de videojuegos que hace hincapié en el pensamiento habilidoso y planificación para lograr la victoria. En estos juegos se destacan los retos estratégicos, tácticos y a veces logísticos. Muchos juegos ofrecen desafíos también económicos y de exploración. En concreto, el jugador debe planificar una serie de acciones contra los oponentes, y el objetivo suele ser reducir las fuerzas del enemigo. La victoria se logra a través de una planificación superior, y el elemento del azar tiene aquí un papel menor.

Los videojuegos de estrategia generalmente toman una de las cuatro posibles formas arquetípicas, dependiendo de si el juego es por turnos ("TBS", siglas del inglés de *turn based strategy*), en tiempo real ("RTS", siglas del inglés de *real time strategy*), si el juego se enfoca en estrategia militar, o por otro lado en estrategia tácticas.

Estos juegos suelen disponer de interfaces muy completas (y a menudo complejas) que permiten obtener informaciones muy detalladas de todos los elementos del juego, así como, la evolución de diversas variables. Muchos de ellos permiten jugar en el modo en línea, participando en universos compartidos con otros jugadores. Este género, suele fomentar la concentración, la reflexión y el razonamiento estratégico.

Dentro de este género podemos encontrar varios subgéneros:

- **Construcción de imperios**: También conocidos como 4X, son juegos en los que debe explorar, expandir, explotar y exterminar. Pueden ser por turnos o en tiempo real. Quizás el juego más conocido de este subgénero sea la serie *Age of Empires*.

- **Videojuegos de artillería**: Son videojuegos, generalmente por turnos, en los cuales tanques, o similares, se atacan unos a otros. El número de tanques puede ser variable y pueden estar controlados por jugadores o por la computadora. Algunos ejemplos de juegos de artillería son *Worms*, *Gunbound*, *Scorched Earth*, *Tanarus*.

- **Estrategia en tiempo real**: son a menudo juegos que permiten seleccionar múltiples unidades o personajes (los personajes, de los juegos múltiples, pueden ser seleccionados a la vez para realizar diferentes tareas, en lugar de seleccionar sólo a un personaje al mismo tiempo) en una vista aérea (es decir, vemos de arriba abajo). La acción en el juego es continuo, y los jugadores tendrán que tomar sus decisiones y acciones dentro del contexto del juego que cambia constantemente el estado del mismo. Los juegos de estrategia en tiempo real se caracterizan por la obtención de recursos, construcción de

bases, la investigación de tecnologías y la producción de unidades. Ejemplos de juegos de este subgénero son, la saga *Command & Conquer*, las series de *Blizzard's Warcraft* y *StarCraft*.

- **Táctica en tiempo real**: Son también conocidos, por sus siglas en inglés, *RTT* (Real-Time Tactics). Comparten aspectos de los juegos de simulación y juegos de guerra, los juegos de táctica en tiempo real se enfocan en aspectos operacionales y control de guerra. A diferencia de los juegos de estrategia en tiempo real, el manejo económico y de recursos y la construcción de edificios no forman parte de las batallas. Algunos ejemplos son *Warhammer: Dark Omen*, *World in Conflict* y la saga *Close Combat*.

- **Estrategia por turnos**: El termino TBS (siglas en inglés de Turn Based Strategy) está generalmente reservado para ciertos de juegos de ordenador de estrategia para distinguirlos de los juegos de ordenador de estrategia en tiempo real. Al jugador de un juego basado en turnos se le permite un período de análisis antes de realizar una acción del juego. También se pueden dividir en dos tipos, dependiendo de si en un turno, los jugadores juegan simultáneamente o juegan sus turnos en secuencia. Algunos ejemplos de este género son *Civilization*, las sagas *Heroes of Might and Magic* y *Master of Orion*.

- **Táctica por turnos**: Conocidos también por sus siglas en inglés TBT (turn-based tactics). Estos videojuegos se caracterizan por la expectativa de los jugadores por completar sus tareas usando sólo las fuerzas de combate que se les proveen, y usualmente por la disposición de una representación realista (o por lo menos creíble) de operaciones y tácticas militares. Ejemplos del género son *Jagged Alliance* y la saga *X-COM*, así como juegos de rol tácticos como *Final Fantasy Tactics*.

- **Videojuegos de guerra**: Los juegos de guerra son un subgénero de los juegos de estrategia cuyos componentes principales son guerras tácticas o estratégicas en un mapa. Los juegos de guerra pueden ser por turnos o en tiempo real y de estrategia o táctica, títulos característicos de estos videojuegos son, *Panzer General*, *Steel Panthers*, *Combat Mission*, etc.

2.2.4 Videojuegos de Acción

Los videojuegos de acción se caracterizan por ser juegos que ponen a prueba las habilidades psicomotrices del jugador, como la rapidez, la percepción visual o la precisión en el control de los mandos de la consola u ordenador, así como advertir los estímulos y/o mensajes que surgen en la pantalla.

En estos videojuegos, el jugador normalmente controla el avatar de un protagonista. El avatar debe navegar por los niveles del juego, recogiendo objetos, evitando obstáculos y combatiendo enemigos con varios ataques. Al final del nivel o grupo de niveles, el jugador debe, a menudo, derrotar a un gran enemigo jefe, que es más grande y más desafiante que otros enemigos. Los ataques de los enemigos y obstáculos agotan la salud

del avatar y por lo tanto, también su vida, y el juego se acaba cuando el jugador se queda sin vida. Por otra parte, el jugador gana la partida al terminar una secuencia de niveles.

Algunos juegos de acción pueden incorporar otros desafíos, tales como carreras, puzles, o recoger objetos, también pueden surgir problemas tácticos y de exploración, pero no son fundamentales para el género.

Dentro de este género podemos encontrar varios subgéneros como:

- **Beat 'em up**: los llamados "videojuegos de lucha a progresión" son videojuegos similares a los de la lucha, con la diferencia de que en este caso los jugadores deben combatir con un gran número de individuos mientras avanzan a lo largo de varios niveles. Es posible jugar con dos o más personajes a la vez de forma cooperativa para facilitar el progreso. Ejemplos de videojuegos clasificados como Beat 'em up son: la saga *Double Dragon*, *Super Smash Bros. Series* y *God of War*.

- **Lucha**: En los videojuegos de lucha se muestran combates entre dos o más luchadores, controlados por jugadores humanos o por la computadora. El objetivo del juego, es acabar con el contrincante, mediante alguna técnica de combate, como artes marciales, lucha libre y boxeo, en algunas casos, también se añaden, a estas luchas, objetos contundentes como espadas, hachas, martillos, etc. Hay juegos de lucha tanto en 2D como en 3D, pero la mayoría de los juegos de lucha 3D se lleva a cabo en gran parte en el plano 2D y en ocasiones en el plano 3D, sobre todo para esquivar objetos del entorno. Los videojuegos más destacados de este "subgénero" son, la saga *Street Fighter*, la saga *Virtua Fighter* y la saga *Mortal Kombat*.

- **Plataformas**: En los videojuegos de plataformas el jugador controla a un personaje que debe avanzar por el escenario evitando obstáculos físicos, ya sea saltando, escalando o agachándose. Además de las capacidades de desplazamiento como saltar o correr, los personajes de los juegos de plataformas poseen frecuentemente la habilidad de realizar ataques que les permiten vencer a sus enemigos, convirtiéndose así en juegos de acción. Inicialmente los personajes se movían por niveles con un desarrollo horizontal, pero con la llegada de los gráficos 3D este desarrollo se ha ampliado hacia todas las direcciones posibles. Los videojuegos más destacados de este "subgénero" son, *Super Mario Bros*, *Sonic the Hedgehog*, *Rayman* y *Donkey Kong*.

- **Disparos**: Los videojuegos de disparos permiten al jugador pasar a la acción a distancia usando un arma de fuego, teniendo el desafío de apuntar con precisión a los enemigos. A pesar de que disparar es generalmente una forma de violencia, también existen algunos "shooters" no violentos. Este subgénero incluye los videojuegos de primera persona (FPS), y los de tercera persona (TPS). Algunos videojuegos destacados de este género son: *Doom*, *Quake*, *Golden Eye 007*, *Metal Gear*, la saga *Resident Evil*, etc.

- **Árcade**: Los videojuegos árcades se caracterizan por tener a menudo niveles cortos, con un sistema de control sencillo e intuitivo, y una dificultad que

aumenta rápidamente. Esto es debido al entorno arcade, donde los jugadores deber pagar por jugar una partida, que acabará cuando se termine la vida del protagonista o se acabe el juego. Este subgénero tuvo mucho éxito en las primeras salas recreativas. Juegos característicos de este subgénero son, *Space Invaders*, *Asteroids*, *Pac-Man*, *Donkey Kong*.

2.2.5 Videojuegos de Simulación

Este género se caracteriza por marcar un aspecto de la vida real o de un sistema, llevada a un juego, en donde se tiene un control total de lo que ocurre en él. El jugador adquiere un rol de "todopoderoso" sin identificarse con nadie ni nada en concreto.

Gracias a la visualización de la repercusión de las acciones en la pantalla, el jugador extrae sus propias conclusiones como un forma activa de aprendizaje. Algunos de estos videojuegos tienen su origen en recursos de entrenamiento profesional. Fomentan la creatividad, la imaginación y el razonamiento lógico.

En otras subcategorías, se toma un concepto o una situación y se deja al usuario explotar las distintas opciones, entre las más populares están las simulaciones de construcción, que puede abarcar desde construir una casa, hasta proyectos imposibles para una persona, como un parque de atracciones o una ciudad completa.

Dentro de este género podemos encontrar varios subgéneros:

- **Simulación de vehículos**: Son un género de videojuegos que tratan de proporcionar al jugador, con una interpretación realista del funcionamiento, varios tipos de vehículos. Esto incluye automóviles, aeronaves, embarcaciones, naves espaciales, vehículos militares, y una gran variedad de otros tipos de vehículos. El principal reto es dominar la conducción y dirección del vehículo desde la perspectiva del piloto o el controlador, añadiendo, en algunos casos, algún desafío, como carreras o combates con vehículos rivales. A menudo, estos juegos se dividen según su realismo, en algunos juegos, se incluye una física más realista y otros desafíos, tales como la gestión del combustible, etc. Algunos ejemplos de este tipo de videojuegos son, *FlightGear*, *Spy Hunter*, *MechWarrior*, *Tom Clancy SSN*, *GT Legends*, etc.

- **Simulación de construcción y de gestión**: Los CMSs (de las siglas en inglés de Construction and Management Simulation) es un tipo de videojuego en el cual los jugadores construyen, amplían o gestionan comunidades ficticias o proyectos con recursos limitados. El objetivo del jugador es construir algo dentro del contexto de un proceso en curso. Por ejemplo, en los juegos de construcción de ciudades, las acciones del jugador como planificador global o líder, satisface las necesidades y carencias de los personajes del juego poniendo en marcha estructuras para la comida, refugios, salud, atención espiritual, crecimiento económico, etc. El éxito se logra cuando el presupuesto de la ciudad obtiene un beneficio cada vez mayor y los ciudadanos experimentan un estilo de vida actualizado en materia de

vivienda, salud y bienes. Títulos destacados de este género son, *SimCity*, *Theme Park*, *SimTower*, *Transport Tycoon* e *IndustryPlayer*.

- **Simulación de vida**: Los juegos de simulación de vida tratan de mantener y desarrollar una población manejable de organismos, donde a los jugadores se les da el poder de controlar las vidas de las criaturas autónomas o de las personas. En este tipo de videojuegos se puede girar en torno a los individuos y las relaciones, o podría ser una simulación de un ecosistema. Incluso en algunos títulos se puede controlar a un dios, que tiene el control sobre la vida de las personas, y en la gestión de los adoradores o devotos tribales. También en los llamados, simuladores de mascotas, podemos controlar a una o varias mascotas o animales a la que se debe proporcionar los cuidados necesarios para que no muera (virtualmente hablando). Uno de los juegos más característicos de este subgénero son, *Los Sims*, *Nintendogs*, *Tamagotchi*, *Black & White*, y la serie de *Populous*.

2.2.6 Videojuegos Sociales

Aunque la mayor parte de los distintos géneros de videojuegos permiten jugar con o contra otros, en los últimos años ha aparecido un nuevo formato de juego específicamente diseñado para jugar en grupo como actividad familiar o para jugar entre un grupo de amigos. Muchos de estos juegos incorporan a la consola nuevos periféricos como cámaras de video, micrófonos, raquetas, etc.

Dentro de este género podemos encontrar varios subgéneros:

- **Juegos de Concurso**: imitan el modelo de los concursos de preguntas y respuestas más comunes de la televisión o de los juegos de mesas. (Ejemplo, *Buzz*).

- **Juegos Musicales**: permiten bailar, cantar o tocar algunos instrumentos. (Ejemplos, *Sing Star* o *Guitar Hero*).

- **Juegos Deportivos**: permiten reproducir los movimientos de algunos deportes como el tenis, el boxeo, los bolos, etc. (Ejemplo, *Wii Sports*).

- **Juegos de Movimiento Corporal**: trasladan el movimiento físico del cuerpo del jugador a la pantalla. (Ejemplo, *Eye Toy*).

Aunque en el mundo de los videojuegos se suele clasificar a un videojuego en un género dado, en los últimos años, está clasificación se hace mucho más difícil, ya que muchos de estos nuevos títulos incluyen varios géneros, gracias a los avances tecnológicos y al éxito del videojuego como modo de entretenimiento.

2.3 Estudio de Videojuegos Adaptables a la Inteligencia Artificial

En esta tercera sección del capítulo 1, se recopilan una serie de títulos de videojuegos de diferentes géneros, a los cuales, se podría aplicar alguna técnica de Inteligencia Artificial (IA) para desarrollar un agente autónomo que sea capaz de jugar una partida o un nivel del videojuego en cuestión, poniéndose en lugar del jugador humano.

Por lo tanto, cada videojuego analizado tendrá una breve descripción del mismo, mostrando las características más importantes y más atractivas para desarrollar el futuro agente o controlador. Seguidamente, se repasarán diferentes investigaciones, que se han realizado hasta el momento, que aplican Inteligencia Artificial a los videojuegos en análisis u otros videojuegos de las mismas características y género. Finalmente se expondrá unos breves comentarios sobre los aspectos positivos y negativos a la hora de decidir si puede ser un candidato final para la realización del presente proyecto.

2.3.1 Age of Conquest

El primer juego a analizar es *Age of Conquest* que es un juego medieval de tipo Risk, basado en estrategia por turnos en el que se tienen que tomar las riendas de un imperio aún en formación y luchar contra los imperios próximos para el control del mundo, en la *Ilustración 1* y la *Ilustración 2* se puede observar algunas de las posibles acciones a realizar, en este caso se tiene una acción de defensa y otra de diplomacia respectivamente. El juego está disponible para los principales sistemas operativos incluyendo Windows, Macintosh, Linux, Android y el IOS (iPhone, iPod Touch, IPAD), las imágenes *Ilustración 3* e *Ilustración 4* muestran diferentes menús en la versión de PCs.

Posee una versión gratuita para descargar, aunque limitada a un escenario y una posición de inicio. Aun así, es suficiente para iniciar un proyecto que diseñe un agente autónomo que sea capaz de jugar con éxito. Además tiene el aliciente de realizarlo en versiones para móviles, debido al aumento de estos dispositivos en la sociedad.

El motivo principal para descartar este videojuego, es que carece de un API (siglas en inglés de Interfaz de Programación de Aplicaciones), ni una guía de cómo está diseñado internamente o cómo es su funcionalidad. Por lo tanto, el primer paso para realizar este agente autónomo sería diseña un API, con la que el agente podrá interactuar con el entorno del juego.

Este es un título que se clasifica dentro del género de *Estrategia por Turnos* (TBS) comentado en el apartado anterior.

Actualmente hay varios trabajos realizados, que tratan de aplicar técnicas de Inteligencia Artificial (IA) a videojuegos de este género. Un ejemplo, es el trabajo realizado por Sánchez-Pelegrín, Gómez-Martín y Díaz-Agudo, [3], que utiliza Razonamiento Basado en Casos (CBR) para desarrollar un módulo de IA. En [4] se utilizan ejemplos generados por humanos para guiar a un algoritmo neuro-evolutivo

Lamarckiano. En [5] se construyó un agente basado en el conocimiento del juego, usando reflexión basado en modelos y auto-adaptación, combinado con aprendizaje por refuerzo. En [6] se combinó un sistema de planificación de Red de Tareas Jerárquicas (HTA) con el aprendizaje analógico y modelismo cualitativo, y que se aplicó a una tarea de gestión de recursos. En [7] se utilizan un algoritmo de capas, en concreto, una red de neuronas combinado con un algoritmo de aprendizaje que genera los pesos para la red de neuronas.

Siguiendo las líneas del comportamiento, de los oponentes de los juegos comerciales en este género, a través de reglas integradas en el juego, se puede utilizar la Programación Genética para crear estar reglas de forma automática, mediante agentes o bien humanos o bien autónomos.

Para obtener más información acerca del *Age of Conques* se puede visitar la web del videojuego [8].

Ilustración 1: Instantánea del juego en una acción de defensa.

Ilustración 2: Instantánea del juego en una acción de diplomacia.

Ilustración 3: Menú de instalación de mapas.

Ilustración 4: Menú de información del estado de la partida.

2.3.2 Alien Arena

Alien Arena es un juego de acción, clasificado entre los shooter multi-jugador en primera persona al más puro estilo Prey, Quake, Doom, y la serie Unreal Tournament. El juego está ambientado en la estética de las películas de ciencia ficción de los años sesenta. *Alien Arena* se centra principalmente en la acción multi-jugador en la red, aunque contiene partidas de un sólo jugador contra robots, en la *Ilustración 5* se observa cómo se puede interactuar con los otro jugador, y en la *Ilustración 6* se tiene un enemigo (robots) a eliminar.

Alien Arena es un juego de alta gama, con 37 mapas, 10 personajes, 8 clases de armas, 5 modalidades de juego, 23 niveles, y un navegador propio incorporado, con el que se gestiona la participación de los usuarios en los distintos torneos que se organizan en la Red. Dispone un excelente motor gráfico capaz de generar efectos visuales muy buenos, como en el diseño de paisajes que se pude observar en la *Ilustración 7*. También se podrá adquirir mejoras para la armadura o las armas y así tomar cierta ventaja con respecto a los adversarios, en la *Ilustración 8* se ve una de las múltiples armas que se utilizan en el juego.

Alien Arena ha sido lanzado para Microsoft Windows, Linux y FreeBSD. El juego ha sido gratis, para jugar, desde su creación, y actualmente no hay planes para cambiarlo. Sin embargo, a partir de la versión 7.20, cuenta con publicidad dentro del juego, en el menú principal y, en algunos mapas. Aunque el contenido del juego es de propiedad exclusiva, el motor CRX es de código abierto, y se puede encontrar el código a través del sistema de control de versiones, SVN [9].

Ilustración 5: Imagen del juego - Compañero

Ilustración 6: Imagen del juego – Enemigos

Ilustración 7: Imagen de Juego - Paisaje

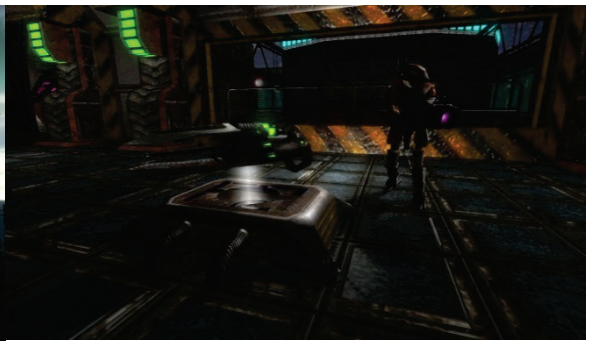


Ilustración 8: Imagen del juego – Armas

Este es un título que se clasifica dentro del género de *Disparos en Primera Persona* (FPS) comentado en la sección anterior.

En la actualidad, en los juegos FPS comerciales, no se utilizan prácticamente técnicas de IA para el comportamiento de los adversarios o "bots" enemigos del juego, en su lugar se utilizan, tradicionalmente métodos no modificables, tales como, máquinas de estado, sistemas basados en reglas, y scripting (secuencia de comandos). Aunque no se utilicen en sistemas comerciales, existen números trabajos que aplican técnicas de IA en este ámbito con buenos resultados, uno de ellos es el trabajo realizado por McPartland y Gallagher, [10], para ello, utilizan aprendizaje por refuerzo (RL) combinado con tres técnicas diferentes, el primer método se combina con un combate previamente entrenado y un controlador de navegación usando RL Jerárquico. El segundo método utiliza reglas simples para determinar cuándo ir a combate o controlar la navegación. El tercer método utiliza RL para "reaprender" las tareas de combate y navegación simultáneamente.

Las conclusiones a las que llegaron M. McPartland y M. Gallagher fueron que los controladores de RL Jerárquico y de RL basado en reglas obtienen mejores resultados que el RL plano en las tareas de combate y de navegación. El controlador RL Jerárquico es el que mejor funciona en disparar al objetivo con precisión, superando al resto de controladores. El controlador RL basado en reglas funciona levemente mejor en los otros objetivos que el controlador RL Jerárquico, sin embargo los resultados que se pueden observar en dicha publicación, se muestran que el controlador RL Jerárquico tiene un rango más amplio de comportamiento en los escenarios de combate. Los controladores de RL Jerárquico y de RL basado en reglas muestran características de jugadores avanzados de FPS, iniciando peleas cuando confían en la victoria. Tanto el controlador de RL Jerárquico como el controlador RL basado en reglas muestran un potencial para la creación de diversos comportamientos contra los "bots" del FPS.

Como se ha comentado anteriormente, la mayoría de los controladores de "bots" (adversarios controlados por la máquina) de FPS son sistemas expertos basados en reglas, en [11] se presentan un método eficiente para utilizar algoritmos genéticos que evolucionan conjuntos de parámetros para los "bots" que llevan a jugar de igual manera que los "bots" cuyos parámetros han sido ajustados por un humano, con conocimiento experto sobre la estrategia del juego, en este caso el videojuego es, el conocido, Counter Strike. Para ello, identifican dos conjuntos de parámetros a optimizar, el primero, los parámetros de selección de armas, y el segundo, un parámetro de *agresividad* (que a la larga afecta a la preferencia del camino a tomar). La función de evaluación utilizada, se basa en una aproximación de la Tabla Estándar PAYOFF/FINE del Counter Strike.

En [12] se propone un controlador inspirado en la arquitectura subsunción, comúnmente utilizada en los comportamientos robóticos (esta arquitectura descompone los controladores en niveles de competencia que trabajan en forma paralela). Un selector de comportamientos decide cuál de los tres sub-controladores toma el control del NPC o "bot" en cada instante de tiempo. Cada controlador es implementado como una red neuronal recurrente, y entrenada con evolución artificial para ejecutar respectivamente el combate, la exploración y el siguiente camino. El selector de comportamiento es entrenado con algoritmos evolutivos multi-objetivos para lograr un equilibrio eficaz de los comportamientos de bajo nivel. El controlador jerárquico evolutivo resuelve mejor la tarea que una aproximación "monolítica" utilizada como comparación.

Para más información sobre el juego se pueden visitar las URLs [13] y [14].

2.3.3 Alien Swarm

Alien Swarm es un juego, y un nuevo lanzamiento, para el Source SDK creado por un grupo de talentosos diseñadores de la comunidad de Mods que fueron contratados por Valve Corporation, que es una empresa estadounidense desarrolladora de videojuegos, sus videojuegos más famosos son, Half-Life y la modificación de éste, Counter-Strike. El Source SDK es un equipo de desarrollo de software de Valve Corporation para que los jugadores que utilicen el motor Source pudieran crear sus propios mapas, modificar caras o crearlas, así como ver modelos con su herramienta de Model Viewer y Model Poser.

Disponible de forma gratuita, el juego lleva a los jugadores a una épica aniquilación de bichos que ofrece una combinación única de juego cooperativo y tácticas de escuadrones. Permite formar con amigos un escuadrón de cuatro clases de Marines que cooperan entre ellos, como se observa en la *Ilustración 9*. Planificar ataques contra una gran variedad de alienígenas haciendo uso de un arsenal de armas a desbloquear con incontables configuraciones, en la *Ilustración 10* se ve algunas de estas armas. Los objetivos principales del juego son arrasar con todo al paso por una colonia invadida fuera del planeta Tierra, erradicando la invasión alienígena en entornos que van desde una superficie planetaria helada a una planta minera subterránea repleta de lava, cooperando entre los jugadores, como se advierte en la *Ilustración 12* e interactuando con las armas obtenidas mediante menús, como el de la *Ilustración 11*.

Junto con el juego se obtiene la base completa del código de *Alien Swarm* que cuenta con actualizaciones del motor "Source", así como del SDK. *Alien Swarm* añade una cámara en tercera persona, la profundidad de campo, mejoras en sombras dinámicas y una gran variedad de complementos de juego para el motor Source.

El código del juego está realizado en el Visual Studio (recomendado la versión 2010), toda la documentación está en el foro de la Web del juego. Además este título se clasifica dentro del género de TPS comentado en la sección anterior.

Ilustración 9: Imagen del juego – Grupo jugadores

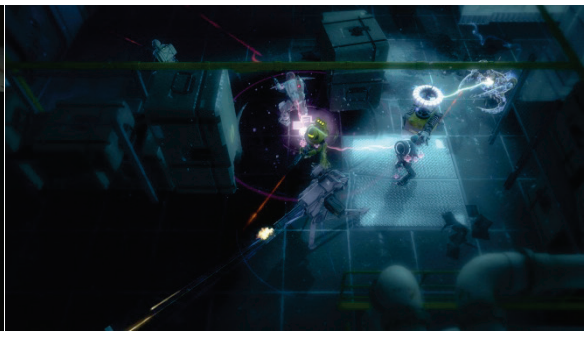


Ilustración 10: Imagen del juego – Armas pesadas

Ilustración 11: Menú durante el juego.

Ilustración 12: Imagen del juego – Ataque en grupo

Aunque el juego tiene un diseño gráfico muy atractivo y un género muy interesante para IA, además de ser atrayente al público en general; tiene el inconveniente de que no tiene un API que permita aplicar IA, ya que los APIs existentes se especifican en su mayoría en el diseño de mapas, personajes, y armas. Por lo que sería necesario crear un API para obtener información del entorno, enemigos, etc.

Actualmente, hay una gran cantidad de investigaciones sobre la aplicación de la IA en los videojuegos del género de FPS, en contrapartida, de las pocas investigaciones que hay para los videojuegos del género de TPS, esto es debido, a que los últimos, hay una mayor libertad de movimientos, además de unos entornos más amplios, por lo que se hace más complejo realizar algoritmos de forma eficiente.

En [15] se muestra un algoritmo de planificación de movimientos en tiempo real, aplicado a este tipo de juegos. Para ello, se apoyan en el concepto de mapas de "espacio CT" (espacio Configuración-Tiempo) para diseñar un nuevo algoritmo de planificación de movimientos que pueda generar movimientos libres de colisiones para la parte superior del cuerpo de un avatar caminando en un entorno virtual desordenado. El sistema es capaz de generar los movimientos de las extremidades de la parte superior del cuerpo para seguir algunos patrones de movimientos deseados, así como evitar los obstáculos del entorno. Aun siendo una buena aproximación para aplicar a estos tipos de juegos, esta investigación está enfocada a entornos generales y no es especifica de un videojuego de TPS en particular.

Para este tipo de videojuegos, sería conveniente utilizar juntas varias técnicas de IA, ya que hay varios aspectos a tener en cuenta, siendo los más importantes, guiar al personaje por el entorno de forma eficiente y acabar con los enemigos que se encuentra a su paso a la vez que se intenta defender de los mismos.

Una técnica que podría funcionar son las colonias de hormigas, aunque se tendría que acotar las direcciones de los caminos a tomar por las hormigas, una opción sería especificar como direcciones a tomar ocho puntos cardinales, Norte, Noreste, Este, Sureste, Sur, Suroeste, Oeste, Noroeste.

Para conocer más datos o información sobre el videojuego se pueden visitar las referencias [16] y [17].

2.3.4 Dolphinity Racer

Racer es proyecto de simulación de coches en multiplataforma (para uso no comercial), usando la física profesional de coches para conseguir una sensación realista y un excelente motor de renderizado con gráficos muy realistas. Los coches, las pistas, los escenarios y similares pueden ser creados con relativa simplicidad (comparados con otros simuladores de conducción). El 3D y otros formatos de archivo están, o deberían estar, documentados. Los editores y los programas de apoyo también están disponibles para obtener un simulador muy personalizable y ampliable. OpenGL se utiliza para la representación (o renderizado).

Las principales características del simulador son:

- Es totalmente gratis (para uso no comercial). Disponible para múltiples plataformas, Windows 2000/XP (En Win95/98/ME pueden trabajar, pero tienen algunos problemas con las fuentes), Linux y Mac OS X.

- Utiliza fórmulas de movimiento de los documentos de ingeniería actual del SAE, por ejemplo. Con una flexibilidad increíble, casi todo es personalizable a través de ficheros ASCII. Motor de renderizado con calidad comercial (con humo, marcas de derrape, chispas, el sol, luces de bengala, pistas iluminadas de color de vértice), en la *Ilustración 13* y en la *Ilustración 16* se observan algunos de estos efectos.

- Soporte para Matlab (archivos de registro que pueden ser convertidos en formato Matlab para su posterior procesamiento y análisis). Soporte para Matrox' Surround Gaming [18].

- Una gran cantidad de coches y pistas adicionales están disponibles en la web (cerca 100 coches y pistas). Fácil integración de coches y pistas personales que se pueden crear en ZModeler, 3D Studio Max (tm), Maya, etc. En la *Ilustración 15* se tiene uno de los números coches disponibles.

- Al menos 15 grados de libertad (DOF) para un coche normal (6 DOF para la carrocería del vehículo, uno para el movimiento vertical de cada rueda, uno para cada rueda, uno para el motor, y varios más por la línea de conducción). Dependiendo en realidad de cuántas ruedas se ponen en el coche. No se limita a 4 ruedas, se soporta, actualmente, cualquier vehículo de dos a ocho ruedas (pero la mayoría están sin probar, y pueden aparecer algunos problemas con los diferenciales en el codificado), los movimientos simulan la funcionalidad del volante del coche, como el de la *Ilustración 14*.

- Reloj interno en tiempo real, sin la dependencia física de imágenes por segundo. Las actualizaciones del controlador también se hacen de forma independiente de la velocidad de fotogramas. Muy pocas limitaciones en los datos de la pista, la información de superficie se toma a partir de datos de los polígonos (pistas de VRML), y los esplines se utilizan para suavizar la superficie de la pista. Las herramientas para modificar las pistas y coches están libremente disponibles en la

web (se recomienda algunas utilidades externas, como el 3D Studio Max, para obtener mejores resultados).

- Algunos algoritmos utilizados se explican en la web, por lo que la web puede ser interesante para aprender a crear un propio software de simulación de coches. Además están disponibles los enlaces y referencias.

Se puede modificar la IA de los coches mediante unos ficheros ".ini", que se pueden modificar fácilmente. Documentación de la IA apartado de AI de la wiki [19]. Esto permite aplicar de forma sencilla, diferentes tipos de algoritmos de IA, el único inconveniente, para elegir como candidato final, es que se tiene que desarrollar una API propia, para que el algoritmo diseñado pueda interactuar de forma eficiente con el entorno del juego.

Ilustración 13: Conducción a través del circuito.

Ilustración 14: Vista interior del coche.

Ilustración 15: Vista exterior del vehículo.

Ilustración 16: Vista área del circuito.

En la actualidad, hay numerosas investigaciones que aplican técnicas de IA a simuladores de carreras de coches, en una de ellas, [20], se examinan dos estrategias eficientes diferentes uno mediante Redes Neuronales y otro mediante Inteligencia Artificial Basada en Comportamiento (BBAI, siglas en inglés). El BBAI, popular en el campo de la robótica, proporciona alguna inspiración para hacer frente a los requisitos computacionales en tiempo real, que necesitan estos tipos juegos, para ello, la inteligencia es percibida como un gran número de componentes modulares relativamente simple y robusto. Cada uno de estos componentes funciona sólo dentro de un conjunto específico

de condiciones que se pueden identificar en el ambiente. BBAI es de carácter reactivo y funciona sin la búsqueda o la deliberación, por lo tanto tiene mucho éxito en aplicaciones de tiempo crítico como la robótica y la realidad virtual interactiva. Mientras, las Redes de Neuronas ofrecen una metodología alternativa para implementar IA en juegos, dentro de la plataforma de juegos de tiempo real. Siendo unos aproximadores universales, las redes de neuronas pueden desarrollar estrategias interesantes y efectivas que se pasan por alto o inadvertidas por los diseñadores basados en comportamientos.

Las conclusiones que se sacan del estudio son que el controlador basado en comportamiento evolutivo fue capaz de superar al controlador de red de neuronas, tanto en términos de las puntuaciones de la competición y la eficiencia computacional. Esto hace que el controlador basado en comportamientos sea un buen candidato para los juegos con gráficos intensos con múltiples agentes que interactúan en el juego donde el uso eficiente de los recursos computacionales es importante para asegurar la presentación interrumpida del juego.

Otro estudio a destacar es el realizado por Cardamone, Loiacono y Lanzi, [21], en el cual utilizan neuro-evolución en línea para evolucionar NPCs del "The Oper Racing Car Simulator" (TORCS), un simulador muy similar al Dolphinity Racer. Para ello, prueban su algoritmos en tres pistas usando dos métodos de neuro-evolución en línea (NEAT y rtNEAT) combinado con cuatro estrategias de evaluación (ε-greedy, ε-greedy-mejorado, softmax y basado en intervalos), y a diferencia de otros estudios similares, en este, se permite una aprendizaje en línea más fino con mejoras en el funcionamiento en cada vuelta. Los resultados que se presentan en el estudio, muestran que, a pesar de varios retos que plantea el aprendizaje en línea, su enfoque 1) puede evolucionar con éxito conducciones a partir de cero, 2) también se puede utilizar para transferir conocimiento evolucionado a otras pistas, y 3) se puede generalizar de forma eficiente produciendo controladores que pueden conducir en pistas a primera vista difíciles. Además indican que el método que mejor funciona es el que se combina con NEAT en línea y las mejores estrategias de evaluación son generalmente "ε-greedy-mejorado" y "softmax".

Por ultimo comentar el trabajo desarrollado por [22], cuyo objetivo es desarrollar un controlador evolutivo para simuladores de carreras de coches utilizando Programación Genética (PG). Para ello, hacen una comparación entre la neuro-evolución y la PG, tanto con y sin estados. Llegan a la conclusión de que dado un problema y una configuración experimental, las varias versiones de PG evolucionan más rápido que las redes de neuronas, pero las redes de neuronas trabajan mejor a largo plazo, especialmente en la versión más complicada de la tarea, que tiene todas las ocho pistas en consideración.

Se puede ver más información en las referencias [23] y [19].

2.3.5 Eufloria

Eufloria es un juego ambientado en la exploración del espacio y conquista, que emplea temas sorprendentes de crecimiento de plantas y evolución bio-mecánica, como se observa en la *Ilustración 18*.

El juego permite al jugador explorar un universo renderizado hermosamente realizado en un estilo que es único e irresistible. La estética de Eufloria es una

reminiscencia de libros infantiles como "El principito" y el juego se apoya en una original banda sonora ambiental de Brian Grainger. El juego gira en torno a la conquista de asteroides en el espacio y usar sus recursos para literalmente crecer, y cultivar nuevas plantas semi-orgánicas y criaturas para la puja del jugador, en la *Ilustración 17* se aprecia diferentes asteroides con sus recursos.

El jugador a continuación se enfrenta contra varios equipos de IA rivales, que todos pueden competir por los mismos recursos y pueden ofrecer una feroz oposición. Según la sección anterior podremos clasificar a este videojuego como árcade, dentro del género de acción, en las ilustraciones, *Ilustración 19* e *Ilustración 20*, se puede ver como se compite por dichos recursos.

La IA se aplica configurando niveles, en ese nivel se configura el comportamiento de las semillas. El tema del videojuego es muy propicio a aplicar técnicas de IA para desarrollar un agente autónomo, en este caso una buena opción sería utilizar una optimización del comportamiento del agente mediante “Colonias de Hormigas”. Además contiene una API con numerosas funciones que permitirá al agente interactuar con nuestro entorno sin problemas [24]. El único inconveniente que nos encontramos, para poder seleccionar a este candidato para nuestro estudio, es que su licencia no es gratuita (solo la demo), el CD cuesta 16.99€ ó 10.99€ en Steam Store y está desarrollado para sistemas Windows.

Enlaces de interés, del videojuego en cuestión, se tienen en [25] y en [26].

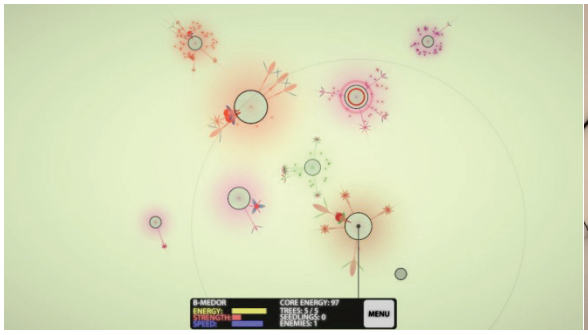


Ilustración 17: Menú e información durante el juego

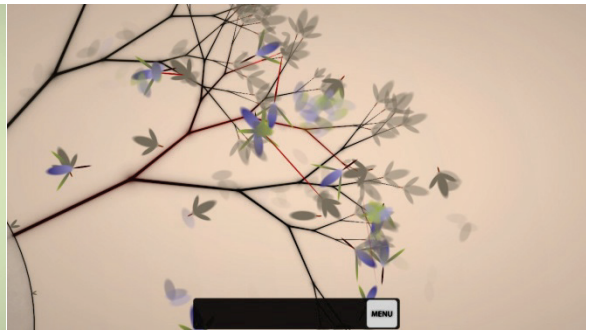


Ilustración 18: Generación de plantas

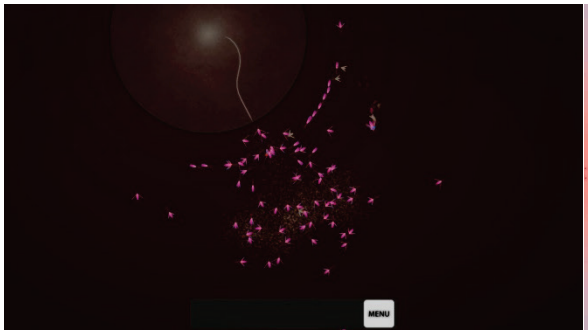


Ilustración 19: Generación de semillas.

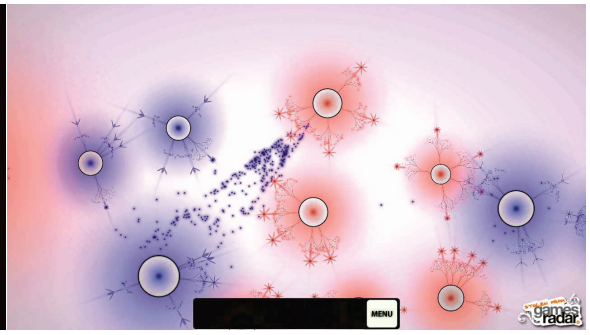


Ilustración 20: Invasión de semillas a otros planetas

2.3.6 Globulation 2

Globulation 2 es un innovador juego de *Estrategia en Tiempo Real* (RTS por sus siglas en inglés) que reduce el mantenimiento asignando tareas automáticamente a las unidades.

Globulation 2 trae consigo una nueva forma de jugar juegos de RTS. El jugador elige la cantidad de unidades que desea asignar a las tareas, y las unidades hacen lo mejor para satisfacer la solicitud. Esto permite que los jugadores controlen más unidades y se concentren en la estrategia más que en la tarea individual de cada unidad. Globulation 2 ofrece características de inteligencia artificial permitiendo elegir entre jugar partidas individuales o haciendo varias combinaciones posibles entre equipos de Humanos-Computadoras. También cuenta con un lenguaje de programación para un modo de juego mucho más versátil; cuenta con tutoriales para aprender a jugar y por si fuera poco un editor de mapas, en *Ilustración 21* e *Ilustración 22* se observan algunas de las diferentes propuestas del tutorial. Globulation2 puede ser jugado, como ya se mencionó, individualmente o por Internet con jugadores de todo el mundo, ya sea mediante la conexión de área local o por Internet con Ysagoon Online Gaming (o YOG para abreviar).

El objetivo es eliminar la colonia de los enemigos. Esto puede hacerse atacando y matando a las unidades del enemigo, convirtiendo a sus unidades al equipo controlado por el jugador, o matando de hambre las unidades del oponente, en las imágenes *Ilustración 23* e *Ilustración 24* se muestran varias de estas acciones. Para ser más fuerte que el oponente se debe crear primero una gran cantidad de unidades, teniendo cuidado de alimentarlos bien para que sobrevivan.

Actualmente hay 5 implementaciones de diferentes IAs desarrolladas o en desarrollo, que proporcionan diferentes comportamientos para enfrentarse con el enemigo, estas implementaciones pueden ayudar a tener un punto de partida a la hora de realizar futuros trabajos, estas implementaciones son:

- *AINumbi*: La IA más simple. Útil para probar y juegos relajantes. Basado en fases, no muy adaptativa, en recursos no amigables.

- *AICastor*: Buena IA básica, útil para juegos estándar. Divertido contra 3 AINumbis. Basado en contexto adaptativo, en recursos amigables.

- *AIReachToInfinity*: Un súper poder económico. Rápidamente construye una gran columna vertebral industrial y empieza a convertir unidades enemigas si no se mantienen al día, pero actualmente no ataca.

- *AINicowar*: excelente IA de GenixPro afinada para la guerra. Significa luchar contra la maquina e intentará destruir al jugador cada vez que sea posible. No siempre seguirá los mismos pasos, así que es difícil predecir su siguiente movimiento.

- *AIWarrush*: IA de Elvish Pillager que está construida para rápida tácticas de batallas. Lanzará una gran cantidad de guerreros hacia a los oponentes hasta que se las arregle para defenderse contra ellos o morir.

Hay 2 IA más experimentales, que son:

- *AINull*: diseñado para no hacer nada, para mapas programados.

- *AIToubib*: AI que jdm trabaja, no hace nada.

Tiene un Centro de Desarrollo, que utiliza el Mercurial para obtener el código de la última versión disponible, en el lenguaje de programación C++. Además tiene diferentes versiones para Linux, Windows y MAC. También posee un API para la IA y un "software" propio para desarrollarla.

Ilustración 21: 2 minutos para perder contra el último edificio de tres Nicowars.

Ilustración 22: Construcción de un campamento base, nota el efecto de las nubes.

Ilustración 23: Ataque al enemigo.

Ilustración 24: Destruyendo al enemigo.

Estos tipos de videojuegos son muy interesantes para la Inteligencia Artificial ya que son ejemplos perfectos de sistemas multi-agentes, por lo que hay un gran número de trabajos que proponen diversos enfoques para la aplicación de IA en estos videojuegos. Uno de ellos es el propuesto por Keaveney y O'Riordan, [27], en la que su estrategia principal es la coevolución, esta estrategia evolucionada utiliza una estructura de sistema multi-agente y una técnica de planificación de refinamiento progresivo que da soluciones para coordinar movimientos y acciones de ataque. Se describen dos tipos de coordinación que son importantes en el juego y se define las medidas para ambos. Se ejecutaron veinte secuencias de coevolución para aprender estrategias para un juego RTS en un entorno geográficamente estático. Definiendo, para ello, dos tipos esenciales de coordinación: coordinación de propagación, y coordinación de ataque. Y cada una de estas medidas de coordinación se utiliza para producir un perfil de coordinación para cada estrategia.

Otro trabajo interesante, para los juegos de RTS, es el realizado por de Freitas Cunha y Chaimowicz, [28], en el cual proponen y desarrollan un sistema de IA que ayuda al jugador durante el juego, dándole consejos de estrategias y tácticas acerca de las mejores acciones que deben tomar de acuerdo con el estado actual del juego. El sistema evalúa el estado del juego siguiendo métricas pre-especificadas, hipótesis elaboradas sobre cómo mejorar el rendimiento del jugador y comunicarle esta información formateado como un conjunto de consejos de estrategias. La información utilizada por el sistema es la misma que la disponible por el jugador, es decir, no utiliza ningún tipo de información interna o privilegiada para analizar el estado del juego. El sistema puede ser considerado como un jugador experimentado que juega junto con el usuario. El objetivo principal es mejorar el rendimiento del jugador durante el juego, ayudando al jugador a hacer frente a la rápida dinámica de los juegos de estrategia en tiempo real. Para aplicar esta estrategia desarrollaron un sistema experto utilizando árboles de decisión que construyen basado en el conocimiento disponible en guías de estrategias desarrolladas para juegos de RTS.

El desarrollo de la IA en juegos de RTS es una tarea ardua y difícil, y la portabilidad de los algoritmos es bastante pobre debido al uso de heurísticas específicas al problema. El aprendizaje en tiempo real puede ser una posible solución, pero involucra un gran tiempo de entrenamiento. En [29] se propone un método de indexado de casos utilizando un enfoque de aprendizaje neuro-evolutivo en juegos de estrategia en tiempo real. La red de neuronas artificial se entrena en las combinaciones colocadas directamente por el resultado del Algoritmo Genético. Con este enfoque híbrido de Redes de Neuronas y Algoritmos Genéticos se pretenden adquirir los conocimientos de forma automática mediante el aprendizaje de los casos anteriores de forma iterativa y memorizar las experiencias exitosas para manejar situaciones futuras.

Este título es un buen candidato para realizar nuestro agente autónomo, ya que tiene una buena documentación, incluye una API del juego, e incluso posee varios ejemplos de agentes de IA. Además se ha visto que este género de videojuego tiene numerosas investigaciones en el campo de la IA, que nos puede ayudar a tomar un punto de partida. Pero tiene el inconveniente que es un género en el que resulta difícil de obtener buenos resultados y que además abarque todos los aspectos del juego.

Para obtener diversa información sobre el juego se tienen las URLs [30], [31] y [32].

2.3.7 Mario AI

Mario AI es un simulador basado en la saga de videojuegos ***Super Mario Bros***, en concreto de la versión de Super Mario World, que es un videojuego dentro del subgénero de plataformas (del género de Acción) desarrollado por Nintendo para la consola Super Nintendo**.** Lo característico de este simulador es que es una versión del *Infinite Mario Bros*, el cual está en código abierto, para poder generar agentes autónomos que controlen al personaje de Mario, mediante técnicas y herramientas de Inteligencia Artificial, y puedan completar el nivel elegido.

Este simulador está desarrollado por Julian Togelius, Sergey Karakovskiy, y Noor Shaker, que han creado una competición internacional llamada, Mario AI Championship que se celebra anualmente, en varias de las conferencias internacionales más importantes que se centran en la inteligencia computacional y en videojuegos.

La competición apareció en el año 2008, a lo largo de los últimos años se han ido definiendo distintas competiciones dentro de la propia competición, cada una de ellas con sus objetivos específicos, por lo tanto, se ha dividido en tres categorías principales: *GamePlay* (Juego), *Learning* (Aprendizaje) y *Level Generation* (Generación de Niveles), además de una nueva categoría, creada en este año 2011, el "Turing Test" (Test de Turing).

La **competición de GamePlay** consiste en desarrollar el mejor controlador (agente) posible, en el lenguaje de programación JAVA, mediante el aprendizaje u otra forma distinta. El trabajo del controlador es ganar tantos niveles (de dificultad creciente) como sea posible. Cada paso de tiempo (24 por segundo en tiempo simulado), el controlador tiene que decidir qué acción tomar (ir a la izquierda, derecha, saltar, etc.) en respuesta al entorno alrededor de Mario.

Uno de los principales objetivos de esta competición es poder comparar diferentes metodologías de desarrollo de controladores unos contra otros, tanto los basados en técnicas de aprendizaje como los que son totalmente codificados a mano. En la actualidad el total de niveles a probar es de 672, la dificultad de cada nivel varía entre los niveles posibles del 0 al 20 algunos de ellos de dificultad elevada, todos con un tamaño longitud de 160, 320 o 480 y con un total de 42 ms para tomar una decisión acerca de que acción ejecutar.

La **competición de Learning** tiene por objeto desarrollar un agente autónomo que sea capaz de aprender un nivel especificado, para ello cuenta con un número máximo de partidas de entrenamiento, en este caso 10.000, en la partida 10.001 el agente será evaluado. Para evaluar al agente se utiliza una función de evaluación, que pondera los aspectos más relevantes del juego como son, tiempo en completar el nivel, monedas recogidas, enemigos eliminados, etc. En esta competición la dificultad de los niveles varía entre los niveles del 1 al 5 y un tamaño de longitud de los mismos de 160, 320, y 480, en este caso el tiempo de respuesta tampoco debe sobrepasar los 42 ms por acción.

En la **competición de Level Generation** se realizan procedimientos que generen niveles, según ciertas especificaciones, para *Infinite Mario Bros*. Se tendrá un número de probadores (testers) que jugará un nivel de prueba in-situ en el evento de la competición. Durante el juego, el comportamiento del juego se registrará. Los niveles generados son

entonces jugados por los probadores y clasificados de acuerdo a lo que más les guste a los jueces.

La **competición de Turing Test** tiene como objetivo desarrollar un controlador autónomo de Mario que juegue del modo más parecido a como lo haría un jugador humano. Los controladores enviados, al igual que en el caso anterior, son evaluados por un jurado.

Aun siendo una competición relativamente nueva, ya ha tenido bastante éxito, y numerosos participantes con resultados y algoritmos muy interesantes, el primero que se va a comentar, es el realizado por S. Bojarski y C. B. Congdon, llamado REALM, en [33] se desarrolla un agente de computación evolutiva basado en reglas, para la categoría de *Learning*, para ello especifican unas metas a conseguir para desarrollar un vocabulario de condiciones y acciones para un sistema basado en reglas, la primera para desarrollar un conjunto de reglas fijas usando el vocabulario, y la segunda para explorar el uso de la computación evolutiva para evolucionar el conjunto de reglas para llevar a cabo la tarea de jugar a Mario. Este agente ganó la competición para las categorías de *GamePlay* y *Learning* en la edición del 2010 de CIG.

Otro trabajo interesante es realizado por E. R. Speed [34], que presenta un método para evolucionar un agente que puede jugar satisfactoriamente un nivel del Mario AI (también para la categoría de *Learning*). Se aplica el reciente algoritmo evolutivo, "búsqueda cuco" que es muy adecuado para las búsquedas de espacios tan grandes cuando se emplea el uso de vuelos "Lévy", que son un tipo de paseo aleatorio en el cual los incrementos son distribuidos de acuerdo a una distribución de probabilidad de cola pesada. Desafortunadamente, estos vuelos "Lévy" no se puede aplicar a los problemas no numéricos tales como Super Mario. Por lo tanto, se presenta, en el trabajo, una modificación del algoritmo que utiliza la distribución "Lévi" para conseguir un cambio apropiado a un conjunto de problemas más amplio, entre ellos Super Mario. Además para optimizar aún más la búsqueda en el espacio del problema de Mario, se presenta una heurística "softmax" para centrase en áreas con posibles soluciones. Los autores concluyen que el uso de la Búsqueda Cuco con vuelos Lévy es una buena elección razonable para un algoritmo evolutivo que juega a Mario. Además, es recomendable para cualquier algoritmo utilizar una heurística "softmax" para centrar la búsqueda en áreas razonables.

En la categoría de *GamePlay* el objetivo principal es completar cuántos más niveles mejor y lo más rápido posible, sin tener en cuenta los puntos u otras bonificaciones, el problema puede ser visto como un problema de optimización de caminos, R. Baumgarten plantea la pregunta ¿Cuál es el camino más rápido a través del entorno el cual no consiga matar a Mario? para resolverla se apoya en la búsqueda A*, como ya es bien conocido, es un algoritmo rápido y simple para encontrar los caminos más cortos, por lo que parece un algoritmo perfecto para la competición de Mario AI. Finalmente se comprueba que la búsqueda A* es los suficientemente rápida y flexible para encontrar rutas a través de los niveles generados en tiempo real. Para implementar este proceso se debe separarse en tres fases: construir una simulación de la física incluyendo los estados del mundo (del juego) y movimientos del objeto; utilizar este simulador en un algoritmo de planificación A*; y optimizar el motor de búsqueda para cumplir los requisitos de tiempo-real con información parcial. Este algoritmo fue el vencedor en la edición del 2009 de la competición.

Otro planteamiento con buenos resultados es el propuesto por S. Polikarpov, apoyándose en la información del entorno que puedes obtener, en concreto una matriz de valores que representan el entorno cercano a Mario, y en el formato de entrada del juego que representa las acciones a realizar por el personaje del juego, desarrolla un agente basado en un nuevo modelo de neurona artificial, la ciber-neurona o neurona "sboxed", en este tipo de neuronas se utiliza tabla de sustitución en lugar de la operación de multiplicación de valores de entrada para los pesos. Esto permite incrementar significativamente la capacidad de información de una neurona, sino que también simplifica enormemente el proceso de aprendizaje. Por lo tanto, para el propósito del juego de Mario AI, desarrolla un agente que está compuesto por una secuencia de acciones, un generador aleatorio (para seleccionar secuencias cuando la neurona está inactiva), neuronas artificiales, asociadas con cada secuencia, una neurona artificial para detectar agujeros y un buffer para almacenar escenas y acciones anteriores. Esta técnica obtuvo buenos resultados en la edición del 2010 de la competición quedando en segundo lugar en la categoría del *GamePlay*.

Este título es un buen candidato para realizar un agente autónomo destinado al presente proyecto, ya que tiene una buena documentación, incluye un API del juego, e incluso posee varios ejemplos de agentes de IA de las diferentes competiciones. Además se ha visto que este videojuego tiene numerosas posibilidades para realizar investigaciones en el campo de la IA, que nos puede ayudar a tomar un punto de partida. Además tiene el aliciente de haber una competición anual, que nos permitirá comparar nuestros resultados obtenidos con el resto de participantes y así valorar que la estrategia elegida puede ser efectiva.

Por todo lo anterior el agente a desarrollar en el transcurso de este proyecto consistirá en un controlador para el juego Mario AI. Asimismo con la intención de participar en la próxima competición, que se celebra entre el 2 y 4 de noviembre en el "IEEE International Games Innovation Conference" de 2011.

Para conocer más datos o información sobre el videojuego y la competición se puede visitar las referencias [35] y [36].

En el siguiente capítulo se justificará la elección de Mario AI para la realización del presente proyecto. La elección se sustentará mediante un análisis sobre el videojuego y además se expondrá una serie de requisitos o funcionalidades que debe cumplir el sistema desarrollado para satisfacer los objetivos expuestos en el apartado 1.3 del capítulo 3.

Capítulo 3

Análisis y Diseño del Sistema Propuesto

3.1 Introducción

En el presente capítulo, se justificará y razonará el porqué de la elección del videojuego elegido y además también se presentará la opción elegida para el diseño del agente autónomo, justificando las razones de su elección. En el siguiente apartado se analizará de forma más exhaustiva el videojuego elegido, para desarrollar un agente autónomo que sea capaz de jugar por sí mismo, además se hará una recopilación de requisitos que tendrá que cumplir el agente a desarrollar, siendo éstos una serie de condiciones o características impuestas o bien por fuentes externas o bien por la naturaleza del problema.

El siguiente paso es por lo tanto detallar cómo se va a construir el algoritmo genético aplicado al videojuego Mario AI para generar dicho agente.

En el apartado sucesivo, se dotará al lector de una visión global de las características principales y del funcionamiento de este tipo de algoritmos; a continuación, en un nuevo apartado, se analizarán aquellos aspectos o características del videojuego que influyen a la hora de diseñar dicho algoritmo; en el posterior apartado, se detallarán los parámetros de configuración del AG, tanto los intrínsecos al algoritmo, como aquellos específicos para el dominio del videojuego; y finalmente en el último apartado, se mostrará en

diagramas de clases el diseño y configuración del controlador diseñado para generar el agente autónomo.

3.2 Justificación de la elección de Mario AI

En esta esta sección del capítulo 3, se presentará las razones que han llevado a que Mario AI sea la elección del videojuego a analizar y desarrollar la implementación de un agente autónomo. En la segunda parte de la sección se justificará que técnica de IA será elegida.

El primer paso de justificación es exponer las ventajas y desventajas que tiene el juego, a la hora de desarrollar un agente autónomo aplicando las técnicas de IA.

Las **ventajas** que nos encontramos son las siguientes:

1. Se tiene una documentación relativamente buena sobre el funcionamiento del juego, se han detallado principalmente aquellas funciones que permiten interactuar y obtener información del juego, y que ayudan al desarrollo del agente.
2. En el presente simulador de *Super Mario Bros* se ha creado una API básica, por parte de los organizadores de la competición Mario AI, en la cual se tiene una serie de funciones y especificaciones que ayuda a que se interactúe el juego con el agente "inteligente" diseñado.
3. En el código fuente del simulador se incluye una serie de ejemplos de agentes en los que se ha aplicado una técnica de IA.
4. La naturaleza del juego permite iniciar varias líneas de investigación en el campo de la IA para desarrollar diferentes tipos de controladores o agentes, como controladores que utilizan redes de neuronas, programación genética, etc.
5. Desde el año 2009 hay una competición internacional del juego, de al menos una edición anual, siendo muy interesante participar en ella para comprobar de forma más precisa, lo eficiente que es o no el agente desarrollado, con respecto a otros agentes que participan en la competición.
6. La competición tiene varias categorías en las que participar, teniendo así más posibilidades de aplicar diferentes técnicas de IA.
7. El videojuego está desarrollado en JAVA, que es un lenguaje muy estudiado durante toda la titulación.
8. El juego y protagonista del juego son un estandarte de los videojuegos, conocido mundialmente.

Las **desventajas** que se pueden tener son las siguientes:

1. No es juego muy llamativo en términos de gráficos del juego.
2. No está documentado el funcionamiento interno del juego (game engine), por lo que hace más difícil entender el comportamiento del juego en algunos instantes que sean de interés.

En definitiva se tiene pocas desventajas, por lo tanto, este juego es ideal para realizar un agente autónomo aplicando algunas de las técnicas de IA, que veremos a continuación, para ello, se propone dos metodologías diferenciadas.

A la hora de diseñar un agente autónomo y atendiendo al momento en el que se evalúa la aptitud de controlador diseñado, se cuenta con dos grandes grupos de posibilidades. En primer lugar podemos observar nuestro entorno y en función de este determinar qué acción tomar. En el lado opuesto a esta propuesta, podemos probar distintos controladores y atendiendo al resultado final de la partida, evaluar el nivel de aptitud de cada uno de ellos. Las ventajas y desventajas de cada una de estas aproximaciones se discutirán a continuación con diferentes propuestas ya realizadas.

La primera de estas dos metodologías consiste en a partir del estado actual del juego, decidir qué acción será la siguiente en ejecutar. A este conjunto de técnicas las denominaremos "en línea". El hecho de tener que decidir qué acción aplicar, en función del estado del juego, implica necesariamente algún tipo de evaluación sobre la aptitud de cada acción del conjunto de acciones posibles. Ahora bien, para poder evaluar la aptitud de una acción, será necesario conocer cuál será el estado siguiente del juego si aplicamos dicha acción, dicho de otro modo, necesitaremos predecir o simular cual será el siguiente estado del juego y evaluar si dicho estado es mejor usando alguna métrica de calidad (puntos, monedas, vida, etc.) respecto al estado anterior.

Una propuesta con este tipo de aproximación, en línea, fue realizada por Robin Baumgarten en 2009, la cual utilizaba el algoritmo de búsqueda A* para llegar hasta la parte derecha de la pantalla. La mayor parte del trabajo se centró en desarrollar el simulador que pudiera predecir cuál sería el siguiente estado al aplicar una acción concreta [37], [38] y [39]. Este tipo de aproximación obtiene excelentes resultados sin importar el tipo de nivel que se juegue, ni si este se mantiene estático o dinámico. Cabe destacar que esta aproximación no tiene en cuenta completamente el estado del juego, sino que utiliza una función heurística, utilizando la distancia entre la posición actual del personaje Mario y la meta (que se encuentra en la parte derecha de la pantalla) asumiendo que no hay enemigos y aceleración máxima.

Otra alternativa, en línea, que sigue el mismo enfoque es utilizar algún tipo de técnica de aprendizaje que en base al estado actual del juego decida cuál es la siguiente acción a ejecutar. Un buen candidato son las técnicas de aprendizaje por refuerzo, en concreto Q-Learning. El principal problema de este tipo de técnica es asociar para cada par [acción, estado] un refuerzo. El espacio de estados puede llegar a ser enorme. Por ejemplo si consideramos la matriz de observaciones de 19x19 y que cada posición está representada por un byte, el espacio de estados sería del orden de $256^{19*19} \approx 2^{8888}$, este espacio de estados es totalmente inabordable con la tecnología actual. Sin embargo y con el fin de reducir la dimensionalidad de dicho espacio de estados, se puede asumir que muchos de estos estados se pueden mapear sobre el mismo estado, en general podemos asumir que se es indiferente el tipo de enemigo que se encuentre en una casilla, incluso podemos mapear los posibles estados de una casilla (8bytes) a solo dos (se puede pasar o no), en cualquier caso y con la misma rejilla (19x19) estaríamos manejando un espacio de estados de 2^{361} el cual sigue siendo demasiado grande de manejar. Así pues la única alternativa plausible sería con este tipo de reducción consistiría en reducir la rejilla a tamaños menores.

Finalmente otra propuesta, en línea, para reducir la dimensionalidad de espacio de estados, propuesta por Mike Hewner's en [39], es considerar el estado como la secuencia de acciones realizadas hasta la fecha, es decir, el estado uno correspondería con haber ejecutado una acción, el estado dos con dos acciones, y así sucesivamente. De este modo y sabiendo que la máxima longitud de las pantallas es de 1024, el número de estados quedaría acotado. Puesto que el nivel se mantiene constante a lo largo de todas las ejecuciones, dicha aproximación funciona moderadamente bien, sin embargo, a la hora de enfrentarse a otros niveles o al mismo nivel con una disposición distinta de enemigos, la política obtenida no tendría por qué obtener buenos resultados.

La segunda metodología propuesta para este juego, consiste en determinar el conjunto de acciones, que obtienen mejores resultados, tras obtener el resultado final de la partida. A este tipo de evaluación la denominaremos “fuera de línea”, es decir, no se tendrá en cuenta el estado del juego en cada instante, sino que solamente se tendrá en cuenta el resultado final de la partida. Este tipo de técnica tiene la ventaja, respecto de las comentadas anteriormente, que no es necesario representar el estado del juego. Por otro lado tiene la desventaja de que una vez obtenido el conjunto de acciones para superar un nivel, este conjunto de acciones sólo valdrán para dicho nivel.

Una propuesta de esta segunda metodología, fuera de línea, es la realizada por Laura Villalobos, [40], que utiliza Programación Genética (metodología basada en algoritmos evolutivos, e inspirada en la evolución biológica), en esta propuesta se define una serie de funciones básicas, que comprueban diferentes estados del juego, por lo tanto, el objetivo final de esta propuesta es generar de forma automática nuevas funciones eficaces, a partir de la combinación de las funciones básicas, que guíen a Mario a completar el nivel.

A pesar de las desventajas citadas, el hecho de no tener que representar el conjunto de estados del juego se ha considerado suficiente para seleccionar esta opción. Para llevarla a cabo se utilizará un **algoritmo genético**, el cual codificará la secuencia de opciones que nos permitirá obtener la máxima puntuación en un nivel específico. Sin embargo la elección de esta técnica plantea algunos problemas, cuya resolución será abordada en el apartado 3.6.

En resumen, se ha elegido al videojuego Mario AI para aplicar alguna técnica de IA que se desarrollará a lo largo del proyecto. La elección de dicho videojuego se sustenta en varios factores, siendo los más importantes, la fácil integración de técnicas de IA dentro del juego, mediante un sencillo API, proporcionado por los organizadores de la competición, además existe varias líneas de investigación, en las cuales se aplican IA, con lo que permite tener una primera referencia para abrir nuevas líneas de investigación. Y por otro lado, se ha especificado que la técnica de IA elegida, para desarrollar el controlador que permita jugar a Mario sin la intervención humana, son los algoritmos genéticos (AG), ya que permite desentenderse de la representación de estados, además de ser una técnica de IA que en este momento no ha tenido ninguna línea de investigación relacionada con Mario AI.

3.3 Análisis del videojuego Mario AI

Mario AI es una versión modificada de *Infinite Mario Bros* de Markus Person, que es un clon de dominio público del juego clásico de plataformas de Nintendo *Super Mario Bros*. El Infinite Mario Bros original se puede jugar en línea a través de la web [41], además también está disponible, en JAVA, el código fuente del videojuego [42].

El juego *Super Mario Bros* consiste en mover al personaje controlable, Mario (o Luigi, según versión), a través de niveles bidimensionales, que son vistos de lado. Mario puede andar y correr a la derecha e izquierda, saltar, y disparar bolas de fuego, siempre y cuando el estado de Mario se lo permita. La gravedad afecta a Mario, haciendo necesario saltar sobre agujeros u hoyos para conseguir pasarlos. Mario puede estar en uno de los tres estados posibles: *Pequeño* (Small), *Grande* (Big, puede aplastar algunos objetos saltando en ellos desde abajo), y *Fuego* (Fire, puede lanzar bolas de fuego).

El principal objetivo de cada nivel es llegar al final del nivel, lo que significa atravesarlo de izquierda a derecha. Los objetivos auxiliares incluyen recoger monedas, cuántas más mejor, que están dispersas a lo largo del nivel, finalizar el nivel lo más rápido posible, y conseguir la puntuación más alta, que en parte depende del número de monedas recogidas y de enemigos eliminados.

La presencia de hoyos o agujeros y de enemigos en movimiento hace más complicado el juego. Si Mario cae en un hoyo pierde una vida, si le toca un enemigo se "lastima", es decir, pierde una vida si se encuentra en el estado *Pequeño*, de lo contrario su estado se degrada en un grado, de *Fuego* a *Grande*, y de *Grande* a *Pequeño*. Sin embargo, si él salta y cae sobre un enemigo pueden ocurrir diferentes cosas. La mayoría de los enemigos (por ejemplo, "goombas", una pequeña seta marrón con pies) mueren al saltar sobre ellos, mientras que otros (por ejemplo, las plantas carnívoras) no son vulnerables a esta acción e hieren a Mario, por último, las tortugas se encierran en sus conchas si se salta sobre ellas, y estas conchas pueden ser cogidas por el Mario y arrojarlas a otros enemigos para eliminarlos.

Ciertos elementos se encuentran dispersados por los niveles, ya sea dentro de los bloques de ladrillos visibles u ocultos y sólo aparecen cuando Mario salta, desde abajo, para golpear esos bloques con la cabeza. Los ítems disponibles incluyen monedas, setas que hacen que Mario crezca a *Grande*, y las flores que hacen que Mario pase al estado *Fuego* si él ya está *Grande*.

Varias características hacen de Super Mario particularmente interesante desde el punto de vista de la Inteligencia Artificial (IA). La más importante de ellas es la representación del entorno muy rica y altamente dimensional. Cuando un jugador humano juega al juego, observa una pequeña parte del nivel actual de lado, con la pantalla centrada en Mario. Sin embargo, este punto de vista incluye docenas de objetos tales como bloques de ladrillo, enemigos y objetos que pueden ser recogidos. El entorno estático (hierba, tuberías, bloques de ladrillo, etc.) y las monedas están distribuidos en una rejilla (o grid, del cual la pantalla estándar cubre aproximadamente 22*22 celdas), mientras que los ítems en movimiento (la mayoría de los enemigos, así como los hongos "power-ups") se mueven continuamente en una resolución de un pixel.

El espacio de acción, aunque es discreto, es también bastante grande. En el juego original de Nintendo, el jugador controla a Mario con un mando "D –pad" (arriba, abajo, derecha, izquierda) y dos botones (A, B). El botón A inicia un salto (la altura del salto es determinada en parte por el tiempo que se presiona el botón), el botón B inicia el modo correr y, si Mario se encuentra en el estado de *Fuego*, dispara una bola de fuego. Sin tener en cuenta la dirección de arriba, que no se suele utilizar, significa que la información a suministrar por el controlador en cada paso de tiempo es de cinco bits, produciendo $2^5 = 32$ acciones posibles, aunque algunos de ellos carecen de sentido (por ejemplo, ir a la izquierda junto ir a la derecha).

Otra característica interesante es que hay una suave curva de aprendizaje entre los niveles, tanto en términos de comportamientos que son necesarios y su grado necesario de refinamiento. Por ejemplo, para completar un nivel de Mario muy simple (sin enemigos y sólo unos pocos y pequeños agujeros y obstáculos) podría ser suficiente con caminar hacia la derecha y saltar cada vez que hay algo (un agujero o un obstáculo) justo delante de Mario. Un controlador que haga esto debe ser fácil de aprender. Para completar el mismo nivel mientras se recoge la mayor cantidad posible de monedas en el mismo nivel probablemente exige una cierta planificación, tales como romper un bloque de "power-up" para recoger un seta que haga pasar a Mario al estado *Grande* y así pueda recoger las monedas ocultas detrás de un bloque de ladrillo, y saltar arriba de una plataforma para recoger las monedas que se encuentran allí y luego volver para recoger las monedas ocultas debajo de esta plataforma (formada por bloques de ladrillo). Niveles más avanzados, incluyendo la mayoría de los del juego original del Super Mario Bros, requieren de una colección de comportamientos variados sólo para completarlos. Estos niveles podrían incluir agrupaciones de enemigos de diferentes tipos que sólo pueden ser pasados por el paso coordinado y con precisión de Mario; para pasar los hoyos y plataformas se requiere de una secuencia complicada de saltos; en un camino sin salida se requiere dar marcha atrás, y así sucesivamente.

A continuación, se va a detallar una serie de requisitos de usuario mínimos que debe cumplir el controlador a desarrollador para satisfacer las condiciones expuestas por la competición de Mario AI, además de las restricciones impuestas por la naturaleza del videojuego. Estos requisitos nos ayudarán a clasificar, ordenar e ilustrar los objetivos principales del proyecto.

3.4 Requisitos de Usuario

A partir de los objetivos generales comentados anteriormente, se pueden obtener los demás requisitos involucrados en el proyecto, los cuales se expondrán en forma de requisitos de usuario, éstos expresan qué debe hacer una aplicación. Para ello, cada requisito de usuario se ha definido con una serie de atributos:

- **Identificación**: cada requisito de usuario incluirá una identificación, para facilitar la trazabilidad, que es un valor único identificativo del requisito que lo diferencia del resto. El identificador está formado por las siglas, requisito de usuario en inglés, UR, más un número incremental de tres cifras empezando por el 001.
- **Descripción**: explicación detallada del objetivo principal del requisito.

- **Justificación**: explicación de la razón principal, por la cual se incluye dicho el requisito en el sistema.
- **Prioridad**: cada requisito de usuario incluirá una medida de la prioridad que posee, para que el desarrollador pueda decidir la planificación del desarrollo. Sus valores pueden ser: Alta, Media o Baja.

➢ **UR-001**

- *Descripción*: Diseñar un agente o controlador que juegue de forma autónoma uno o varios niveles del juego Mario AI.
- *Justificación*: El principal objetivo del presente proyecto y de la competición internacional de Mario AI es que el controlador entregado pueda jugar por sí mismo, en la categoría elegida de la competición.
- *Prioridad*: Con prioridad alta.

➢ **UR-002**

- *Descripción*: Participar al menos en la categoría de Aprendizaje (*Learning*) de la competición de Mario AI.
- *Justificación*: Para diseñar y desarrollar el agente se tiene que tener en cuenta las restricciones de cada categoría, en este caso, la categoría que más interesa para aplicarla al presente proyecto es la de aprendizaje, *Learning*.
- *Prioridad*: Con prioridad alta.

➢ **UR-003**

- *Descripción*: Aplicar alguna de las técnicas de Inteligencias Artificial vistas a lo largo de la titulación.
- *Justificación*: El objetivo principal del presente proyecto es aplicar una técnica de Inteligencia Artificial en el videojuego elegido, realizando un análisis previo y posteriormente desarrollar un controlador mediante la técnica elegida.
- *Prioridad*: Con prioridad alta.

➢ **UR-004**

- *Descripción*: Hay una limitación de 10.000 evaluaciones o pruebas por cada nivel para la categoría Aprendizaje (*Learning*) de la competición de **Mario AI**.
- *Justificación*: Una limitación impuesta, por la competición de Aprendizaje (*Learning*), es que el aprendizaje del agente o controlador a desarrollar está limitado a un número de evaluaciones, por lo tanto, el agente cuenta con 10.000 partidas para aprender cada nivel, y en la partida 10.001 se evalúa al agente registrando los resultados obtenidos.

- *Prioridad*: Con prioridad alta.

➢ **UR-005**

- *Descripción*: El tiempo máximo para generar un agente autónomo está limitado. Es recomendable que este tiempo no sea superior a 10 ó 15 minutos.
- *Justificación*: Aunque no se especifica en la competición un tiempo exacto de duración de la prueba, sí se manifiesta que aquellos controladores que no tengan resultado en un tiempo apreciable serán descalificados.
- *Prioridad*: Con prioridad media.

➢ **UR-006**

- *Descripción*: Las pruebas realizadas, en la competición, tienen varios niveles de dificultad, que van desde el nivel 0 al nivel 4.
- *Justificación*: En la categoría de Aprendizaje (*Learning*) de la competición de **Mario AI** se evaluarán a los agentes en cinco niveles de dificultad incremental, de 0 a 4.
- *Prioridad*: Con prioridad alta.

➢ **UR-007**

- *Descripción*: Mario AI proporciona información relevante del estado del entorno del juego en cada instante (o tic) del juego, mediante un matriz de 22*22 bytes.
- *Justificación*: En la mayoría de las técnicas de Inteligencia Artificial es recomendable conocer el estado del entorno en determinados instantes, por ello, el juego brinda a los agentes la posibilidad de conocer dicho estado mediante un matriz de bytes, en donde se representa al entorno mediante la relación valor - tipo de objeto.
- *Prioridad*: Con prioridad alta.

➢ **UR-008**

- *Descripción*: El agente desarrollado tiene que proporcionar al motor del juego la acción a realizar en cada instante (o tic) del juego, mediante una matriz de cinco valores booleanos que indica la acción a realizar. A través de una interfaz específica.
- *Justificación*: Mario AI tiene que recibir una acción determinada en cada instante de tiempo, para ello el juego ofrece una función que se debe implementar para poder realizar dicha acción, en caso contrario, Mario se quedará inmóvil.
- *Prioridad*: Con prioridad alta.

- **UR-009**
  - *Descripción*: Los movimientos que realiza Mario son los siguientes,
    1. Ir a la izquierda.
    2. Ir a la derecha.
    3. Agacharse.
    4. Saltar.
    5. Velocidad o disparo.
    6. Ir hacia arriba.

    codificado como una matriz booleano de longitud 6, representando en cada posición si se activa (true) o no (false) el movimiento en cuestión.
  - *Justificación*: Los movimientos del simulador de *Super Mario Bros* de la competición están representados por dicha matriz de booleanos, ya que el juego permite combinar diferentes movimientos en una acción, en consecuencia el agente tiene que enviar al simulador un movimiento con la misma representación.
  - *Prioridad*: Con prioridad alta.

- **UR-010**
  - *Descripción*: La función de evaluación a utilizar por el agente, es la dada por el juego de Mario AI que consiste en la siguiente función:

$$fitness = k * kills + c * coins + d * distance + w + win + k' * killsByStomp + k'' * killsByFire + k''' * killsByShell$$

    Ecuación 1: Función de evaluación a utilizar por el AG del presente proyecto
  - *Justificación*: Según las normas de la competición la función de aptitud (fitness, en inglés) o evaluación de una partida está determinada por los organizadores de la misma.
  - *Prioridad*: Con prioridad alta.

- **UR-011**
  - *Descripción*: Almacenar los resultados procedentes de los experimentos, en archivos de fácil lectura, como los ficheros de texto o de tipo hoja de cálculo.
  - *Justificación*: Los resultados de los experimentos se deben poder consultar para su posterior análisis y evaluación.
  - *Prioridad*: Con prioridad media.

- **UR-012**
  - *Descripción*: Los experimentos realizados se deberán poder lanzar de modo automático, por parte de otro programa.
  - *Justificación*: La técnica elegida de IA para desarrollar el agente autónomo evalúa a una agente que viene determinado por los parámetros de configuración de dicha técnica.
  - *Prioridad*: Con prioridad media.

- **UR-013**
  - *Descripción*: Permitir definir varios parámetros de configuración de la técnica de IA elegida.
  - *Justificación*: Para poder evaluar las distintas estrategias de la técnica de IA elegida, ha de permitirse modificar los distintos parámetros de funcionamiento del mismo.
  - *Prioridad*: Con prioridad alta.

- **UR-014**
  - *Descripción*: Almacenar el mejor agente autónomo de cada experimento realizado.
  - *Justificación*: Los mejores resultados obtenidos se deben poder recuperar y reproducir dichos resultados.
  - *Prioridad*: Con prioridad alta.

En los apartados siguientes se hará una introducción de las características y funcionamiento de los AG, seguidamente se realizará un análisis del sistema propuesto en el cual se verán cómo se relaciona el AG con el juego Mario AI, y finalmente se detallará como se ha diseñado el sistema que creará un agente que pueda competir en la competición.

3.5 Descripción de la Técnica de IA Elegida

Como ya se mencionó en apartados anteriores, la técnica de IA elegida finalmente son los Algoritmos Genéticos, ya que es una técnica ideal para encontrar la secuencia de acciones que mejor se adapten al nivel evaluado en Mario AI.

Por lo tanto, el objetivo principal de este apartado es de dotar al lector una visión global de las características principales y del funcionamiento de este tipo de algoritmos.

El propósito básico de un algoritmo genético (AG) es imitar el enfoque evolutivo de la madre naturaleza en la ciencia y la ingeniería, incluyendo en la informática. El

algoritmo se basa en el proceso de la selección natural de Charles Darwin "la supervivencia del más apto". Los AGs pueden ser utilizados en la resolución de problemas, la optimización de funciones, aprendizaje de máquinas, y en los sistemas de innovación.

Los AGs están considerados como algoritmos de búsqueda directa basados en los mecanismos de la evolución biológica. El atractivo de los AGs proviene de su simplicidad y la elegancia como algoritmos de búsqueda robustos. Éstas son herramientas poderosas para descubrir rápidamente soluciones adecuadas para problemas difíciles de grandes dimensiones. Los AGs son útiles y eficaces cuando:

- El espacio de búsqueda es grande, complejo, o poco entendible.
- El conocimiento del dominio es escaso o el conocimiento experto es difícil de codificar para reducir el espacio de búsqueda.
- No se dispone de análisis matemático.

Los AG son un tipo de algoritmo evolutivo (AE). Otros tipos de AE son estrategias evolutivas, programación evolutiva y programación genética. Cada uno comparte la filosofía de diseño de AE básico de la generación de hipótesis independientes del dominio seguido por la selección de dominio específico. Sin embargo, difieren en las representaciones principales, la importancia relativa de la mutación contra el cruce, y las estrategias de selección preferidas.

John Holland, en [43], sentó las bases de esta nueva técnica de inteligencia artificial, consistente en la búsqueda heurística de una solución para un problema dado, inspirándose en la selección natural darwiniana [44].

3.5.1 Funcionamiento de los AG.

Cada célula biológica de cada ser vivo contiene cromosomas, que son nada más que las cadenas de ADN. En los AGs los cromosomas pueden ser cadenas de bits, números reales, permutaciones de elementos, listas de reglas y estructuras de datos.

Un genotipo es un conjunto de genes que representen las posibles soluciones en un espacio del dominio. Este espacio, conocido como un espacio de búsqueda, comprende todas las soluciones posibles para el problema en cuestión. Estas soluciones pueden ser codificadas con estrategias, tales como codificación basada en caracteres, codificaciones de valores reales, y representaciones de árboles. Con cada paso en la evolución, conocida como una generación, los individuos de la población actual son decodificados y evaluados de acuerdo a algún criterio de calidad predefinidos, conocida como aptitud (fitness, en inglés) o función de optimización. Los genes fuertes son normalmente representados en la nueva generación. Los genes que no son fuertes se combinan con sus compañeros y se utilizan para generar nuevos individuos en un proceso conocido como recombinación. La Mutación y el Cruce son operadores de recombinación. Si no se realiza dicha operación, la descendencia es una copia exacta de uno de los padres.

Una población inicial se considera que tiene un número fijo de individuos (también conocido como descendientes) que contienen cromosomas que son conjuntos de genes. Estos individuos son evaluados por su capacidad de sobrevivir en situaciones críticas.

Esta prueba es conocida como función de optimización o aptitud. Un individuo apto con genes fuertes se puede reproducir por sí mismo. De lo contrario, en la próxima generación, los genes fuertes de uno o más individuos se pueden combinar, lo que resulta en un individuo más fuerte. Con el tiempo, los individuos de la población se adaptan mejor a su entorno.

Los algoritmos genéticos son algoritmos cuyos mecanismos de búsqueda imitan la evolución de las especies, para ello, los AGs utilizan una población de individuos, cada uno de los cuales representa una posible solución al problema, que se quiere tratar. A cada individuo se le asigna un valor de aptitud que representa la calidad de dicha solución. Una vez que se tiene dicha población de soluciones, que se puede obtener o bien de forma aleatoria o bien a través de alguna heurística, a continuación se realizar el proceso de cruce en el cual algunos individuos de la población son emparejados de forma aleatoria y sobre cada una de las parejas se aplica el mecanismo de cruce con un determinada probabilidad, generando así dos nuevas soluciones. Seguidamente algunos individuos de la población pueden mutar, es decir, pueden ser alterados parcialmente, introduciéndose nuevas características a la población, mediante el operador de mutación. Finalmente, se realiza una nueva selección de la población dependiente del valor de aptitud de los individuos, siendo más probable que se elija individuos con aptitudes más altas.

A continuación se muestras los pasos de un algoritmo genético genérico,

1. *Se inicializa una población inicial al azar, con una población de N cromosomas o individuos (con su correspondiente cromosoma).*
2. *Se calcula la aptitud f(x), donde x es un individuo válido de la población.*
3. *Se crea una nueva población repitiendo los pasos siguientes hasta que la nueva población tenga la cantidad necesaria de individuos aptos.*
   - ***3.1.*** *Se seleccionan dos individuos padres de la población con mejor aptitud.*
   - ***3.2.*** *Se forma un nuevo individuo mediante la aplicación de una operación como la de cruce, mutación o selección.*
4. *Se evalúa la nueva población para obtener la aptitud de sus individuos.*
5. *Se prueba la nueva población para obtener la solución, si satisface la condición o criterio de parada, se termina.*
6. *Se repite los pasos del 2 al 5 hasta que se logre una solución deseable.*

De forma gráfica se puede ver el ciclo del AG en la *Ilustración 25*.

El rendimiento de cualquier AG viene influido por sus operadores. La mutación y el cruce son dos operadores básicos que contribuyen significativamente a aumentar el rendimiento del algoritmo. Sin embargo, antes de que estas operaciones puedan ser aplicadas a los individuos seleccionados, es necesario codificar los individuos. El método más común de codificación es una cadena binaria. Cada individuo representa una cadena binaria. Cada bit de esta cadena puede representar alguna característica de la solución, como se muestra en la *Ilustración 26*.

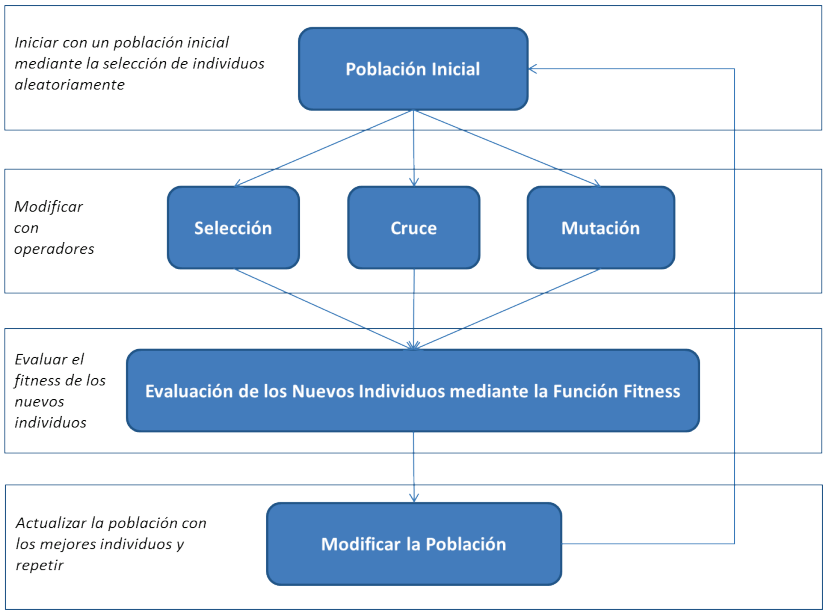


Ilustración 25: Ciclo del Algoritmo Genético

Dependiendo de la naturaleza del problema, se puede adoptar otras formas de codificación de los individuos, como números y letras.

Individuo 1	1	0	0	1	1	1	1	0	0	0	1	1	0	1	1	0
Individuo 2	0	0	1	0	1	1	0	0	0	1	1	1	0	0	1	1

Ilustración 26: Codificación de individuos

3.5.1.1 Operadores Genéticos.

Los operadores genéticos son el corazón del algoritmo genético, permitiendo, en el proceso de búsqueda, crear nuevos individuos y mejores adaptados.

El AG se apoya en tres operadores para encontrar la solución óptima, que son el Operador de Selección, el Operador de Cruce y el Operador de Mutación que se detalla a continuación

- **Operador de Selección.**

Las soluciones individuales son seleccionadas mediante un proceso basado en la aptitud del individuo, en donde las soluciones aptas (según lo medido por una función de optimización) tienen más probabilidades de ser seleccionadas.

La presión selectiva se define como el grado en el cual los mejores individuos se ven favorecidos. Cuanto mayor sea la presión selectiva más se favorece a los mejores individuos. Esta presión selectiva impulsa al AG a mejorar la aptitud de la población en las sucesivas generaciones.

La tasa de convergencia del AG está determinada en gran parte por la magnitud de la presión selectiva, con mayor presión selectiva se tiene tasas de convergencia mayores.

Los AGs deben ser capaces de identificar las soluciones óptimas o casi óptimas en un amplio rango de esquemas de presión selectiva. Sin embargo, si la presión selectiva es demasiado baja, la tasa de convergencia será lenta, y el AG tomará un tiempo innecesariamente largo para encontrar la solución óptima. Por otro lado, si la presión selectiva es demasiado alta, hay un cambio mayor del AG, convergiendo antes de tiempo a una solución incorrecta (sub - óptima). Además de proporcionar una presión selectiva, los esquemas de selección también deben preservar la diversidad de la población, ya que esto ayuda a evitar la convergencia prematura.

Existen varios métodos de selección, que permite tener una variabilidad de opciones a la hora de realizar la selección de los individuos. A continuación se comentarán brevemente los más importantes:

La selección por *Ruleta* consiste en simular una ruleta en la que cada solución se le ha asignado una sección de ésta. El tamaño de la sección asignada a una solución está en proporción al valor de aptitud de ésta. La ruleta se hace girar N veces, donde N es el número de individuos de la población. En cada giro, el marcador de la ruleta indica el individuo que se seleccionar para ser añadido a la nueva población para la siguiente generación. La selección por *Ruleta* es la más sencilla de aplicar, pero es ruidosa, ya que, la tasa de evolución depende de la variación del valor de aptitud en la población.

La selección *Jerárquica* clasifica la población y cada cromosoma recibe su valor de aptitud según la clasificación. El peor tiene un valor de aptitud de 1 y el mejor tiene un valor de N, o viceversa, según cómo se quiera organizar la clasificación. Hay varias formas de seleccionar a los individuos, una de ellas es la siguiente: se selecciona dos individuos de forma aleatoria; el individuo con la posición más alta en la clasificación se convierte en el candidato. Repetir hasta obtener el número de candidatos deseado. Este tipo de selección elimina los efectos de valores de aptitud muy grandes o muy pequeños, además se uniformiza la diferencia de probabilidades entre los individuos.

La selección por *Torneo* proporciona una presión selectiva mediante una competición por torneos entre N_u individuos. El mejor individuo del torneo es el que tiene mayor valor de aptitud de los N_u individuos que participan en el torneo, por lo tanto, este individuo es un candidato para la siguiente generación. El torneo se repite hasta conseguir obtener todos los candidatos que formarán la nueva población para la siguiente generación. Los candidatos seleccionados tienen la media de aptitud más alta de la población. La diferencia de la aptitud proporciona la presión selectiva que lleva al AG a mejor la aptitud de los genes sucesivos. El tamaño del torneo, N_u, permite asignar la presión selectiva; no es necesaria la función de evaluación, ya que solo se compara individuos y permite mantener y regular la variabilidad genética.

Muchos de los procedimientos de selección están en uso actualmente, una de las más simples es la selección de aptitud proporcional de [43], donde los individuos se seleccionan con probabilidad proporcional a su aptitud relativa.

- **Operador de Cruce.**

El Cruce selecciona sub-cadenas de genes de la misma longitud de individuos padres (a menudo llamados descendientes) del mismo punto de la cadena, sustituyéndolos unos por otros, generando un nuevo individuo, llamado hijo. El punto de cruce puede ser seleccionado al azar, en algunos casos puede aparecen varios puntos de cruce. El operador se divide en tres pasos, cuyo resultado se puede visualizar en la *Ilustración 27*:

- Paso 1: Seleccionar al azar un par de individuos para el cruce.
- Paso 2: Seleccionar un punto de cruce al azar dentro de la longitud de la cadena de los individuos.
- Paso 3: Los valores de posición se intercambian entre las dos cadenas después del punto de cruce.

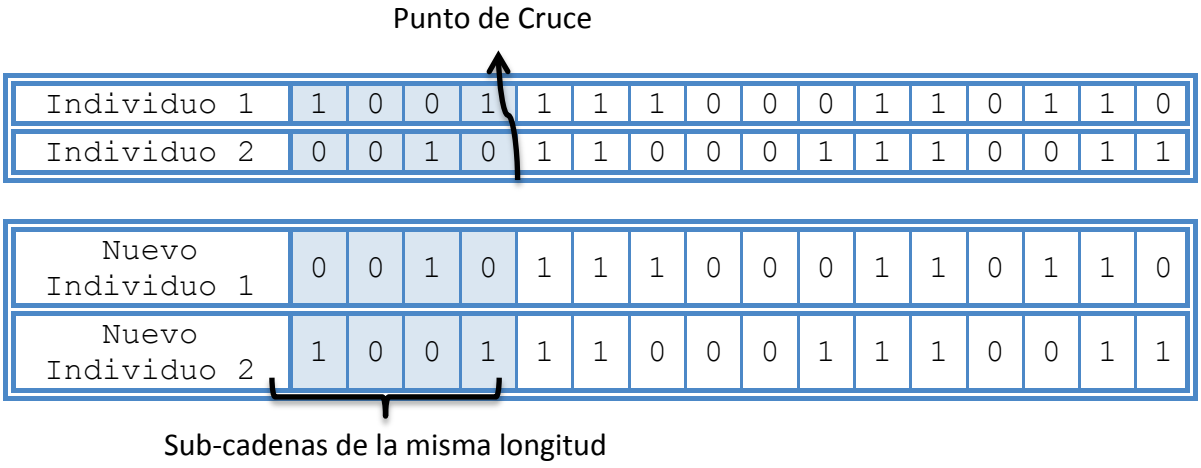


Ilustración 27: Cruce de Individuos

El proceso de cruce no debe consistir únicamente en una combinación de las soluciones para formar nuevas soluciones, sino que esa combinación sea verdaderamente beneficiosa. Al igual que ocurre con el operador de selección se ha desarrollado diversas técnicas de cruce de las cuáles se van a explica brevemente algunas de ellas:

El *cruce simple* o de *1 punto* se utiliza en los AG genéricos o tradicionales, donde los dos cromosomas a cruzar se cortan una vez en los puntos correspondientes y las secciones siguientes al punto de corte se intercambian. El punto de cruce se selecciona al azar a lo largo de la longitud del cromosoma. Si se ha elegido el punto de cruce apropiado, se puede obtener mejores "hijos" mediante la combinación de buenos padres, si no, se empeora severamente la calidad del cromosoma. En la *Ilustración 27*, se puede comprobar el funcionamiento de este tipo de cruce.

El *cruce multipunto* genera t puntos de cruce de forma aleatoria de manera que estos puntos de cruce se encuentre el 1 y la longitud del cromosoma ($1 \leq \langle k_1, k_2, \dots, k_t \rangle \leq l$). Ahora la nueva solución, el primer hijo, heredará los k_1 primeros genes del padre, los genes situados entre k_1+1 y k_2 de la madre, los genes entre k_2+1 y k_3 también del padre, y así sucesivamente hasta llegar a los genes entre k_t+1 y l del padre o madre según corresponda. El segundo hijo, por lo tanto, será los k_1 primeros genes de la madre, los genes situados entre k_1+1 y k_2 del padre, los genes entre k_2+1 y k_3 también de la madre, y

así sucesivamente hasta llegar a los genes entre k_t+1 y l del padre o madre según corresponda.

El *cruce uniforme* genera lo que se conoce como máscara de cruce, que consiste en una cadena de igual longitud, que los cromosomas, compuesta por ceros y unos generados de forma aleatoria; se crea una máscara aleatoria para cada pareja a cruzar; para generar uno de los hijos, se procede de la siguiente forma: si en la posición *i* de la máscara aparece un 1, el hijo hereda dicho gen del padre, y si aparece un 0, el gen es heredado de la madre, para generar el siguiente hijo se hace de forma análoga pero intercambiando a los padres.

- **Operador de Mutación.**

La mutación impide que el algoritmo se quede atrapado en un mínimo local. La mutación desempeña el papel de recuperar los materiales genéticos perdidos, así como desordenar la información genética de forma aleatoria. Se trata de una póliza de seguro contra la pérdida irreversible del material genético. La mutación se ha considerado tradicionalmente como un operador de búsqueda simple. Si el cruce se supone que explota la solución actual para encontrar otras mejores, las mutaciones se supone que ayuda a la exploración del conjunto de espacio de búsqueda. La mutación se ha visto como un operador de fondo para mantener la diversidad genética en la población, que introduce nuevas estructuras genéticas en la población de forma aleatoria modificando algunos de sus bloques de construcción. La mutación ayuda a escapar del problema de los mínimos locales y mantiene la diversidad en la población.

La mutación cambia un gen por otro con ayuda de una pequeña probabilidad de mutación $P_{mutación}$. Si el número aleatorio es menor que $P_{mutación}$, se muta el gen, de lo contrario no se muta. Utilizar la mutación por si sola produce un camino aleatorio por el espacio de búsqueda.

La mutación depende de la codificación. Si se considera que la codificación utiliza combinaciones del alfabeto, la mutación podría ser el intercambio de dos genes de determinados individuos.

Individuo 1	1	0	0	1	1	1	1	0	0	0	1	1	0	1	1	0
Nuevo Individuo 1	1	0	0	1	1	0	1	0	1	0	1	1	0	1	0	0
Individuo 2	0	0	1	0	1	1	0	0	0	1	1	1	0	0	1	1
Nuevo Individuo 2	0	0	0	0	1	1	0	1	0	1	1	1	1	0	1	1

Ilustración 28: Mutación de Individuos

3.5.1.2 Función de Evaluación (o Fitness).

Un AG es una técnica de optimización, es decir, que puede ser utilizado para encontrar el máximo o mínimo de una función arbitraria. La función de evaluación permite mediante métodos estocásticos, es decir, métodos que incluyen procesos

aleatorios, encontrar un mínimo global o lo que es lo mismo, encontrar la mejor solución o la solución óptima al problema dado.

La función de evaluación de sistema propuesto se indica en la *Ecuación 1*, del apartado 3.4.

3.5.1.3 Criterio de Parada.

Los AG ofrece "mejores" resultados en comparación con los tradicionales algoritmos de búsqueda, especialmente en situaciones donde no existe el concepto de una solución óptima. Es por eso que, de acuerdo con el Sistema Frontline (www.solver.com, 2008), los algoritmos evolutivos son los mejores empleados en problemas donde es difícil o imposible probar la optimización. Esto también significa que nunca un algoritmo evolutivo sabe a ciencia cierta cuándo parar, aparte de la cantidad de tiempo o el número de iteraciones o soluciones candidato que se quiere explorar. No existe un criterio de terminación claro y evidente de un algoritmo genético. De acuerdo con Weck, [45], algunas opciones son las siguientes:

- *X* número de generaciones completadas, por lo general 100.
- La desviación media en el rendimiento de los individuos en la población se encuentra por debajo de cierto umbral.
- Estancamiento (mejora mínima o sin mejora de una generación a la siguiente).
- Se encuentra un punto en el espacio de búsqueda, es decir, se encuentra una posible solución.

3.6 Aspectos influyentes de Mario AI en los AGs

La técnica de IA, del sistema propuesto, utiliza los Algoritmos Genéticos para encontrar la secuencia de acciones que obtendrá el mejor resultado posible al final de la partida.

En el apartado 3.2 del capítulo 3, ya se menciona que utilizar los AG planteaba algunas particularidades que hay que tener en cuenta a la hora de configurar el algoritmo para el dominio de Mario AI. A continuación se detallan estas particularidades:

Para completar o intentar completar un nivel cualesquiera en Mario AI se necesita realizar un número de acciones que debe realizar el personaje Mario, este número de acciones es desconocido a priori, ya que además de variar de un nivel a otro, también puede variar en el mismo nivel. Este aspecto influye a la hora de establecer el cromosoma de los individuos (un individuo es un posible agente autónomo que jugará un nivel), ya que en este caso, cada gen del cromosoma representa una acción o movimiento que tiene

que realizar Mario en un instante. Por lo tanto, hay que tener en cuenta el **tamaño del cromosoma** (número de genes del cual se compone), o bien se tiene un cromosoma con longitud dinámica, que vaya aumentado según se necesite acciones, o bien establecer un tamaño máximo suficientemente grande para completar un nivel, obteniendo la máxima puntuación posible. Finalmente se ha decidido por la segunda opción, **establecer un tamaño máximo** para el cromosoma, en concreto un tamaño de **3.000** genes (acciones), debido a que esta opción simplifica la implementación de los operadores genéticos del AG, al tener siempre el mismo tamaño de cromosoma. Este valor se obtiene de contar el número de tics (instantes de tiempo en el cual se actualiza el entorno del juego y se ejecuta una acción de Mario) en un nivel sin enemigos, con un "level time" (tiempo máximo para completar el nivel) de 200 segundos y sin que Mario realice ninguna acción.

El siguiente paso, tras establecer el tamaño del cromosoma, es definir, el **formato del gen**, del cual están compuestos los cromosomas que utilizará el AG. En este dominio, el gen representa la acción a realizar, además el formato del gen tiene que ser el mismo que Mario AI necesita para recibir la acción a realizar, por lo tanto, cada gen se codifica como un número binario de 6 cifras o bits, que nos proporciona un total de 64 movimientos diferentes ($2^6 = 64$). Sin embargo, si se mantiene esta codificación tal cual se tendría un espacio de búsqueda demasiado extenso del orden de $64^{3000} = 2^{18000} \approx 3{,}5 \cdot 10^{5418}$. Llama la atención que de estos 64 movimientos muchos son incoherentes, por ejemplo, ir a la derecha e ir a la izquierda (110000), por lo que es necesario acotar el número de acciones posibles a realizar. Para ello, se tienen dos aproximaciones, en la primera se acota el número de acciones a 11 posibles movimientos, manteniendo fijo el bit correspondiente al movimiento "Velocidad" a **1** y el bit correspondiente a "Ir a hacia arriba" a **0**, en la *Tabla 1* se muestran las configuraciones de los movimientos; y en la segunda aproximación, únicamente se mantiene fijo el bit correspondiente al movimiento "Ir hacia arriba" a **1** cuando el valor del bit "Ir hacia abajo" es **0**, y a **0** cuando dicho bit está a **1**, por lo que se tienen 22 posibles movimientos, en la *Tabla 2* se indican las configuraciones de dichos movimientos.

Acción	IZQUIERDA (LEFT)	DERECHA (RIGHT)	ABAJO (DOWN)	SALTAR (JUMP)	CORRER (SPEED)	ARRIBA (UP)
0	0	0	0	0	1	0
1	0	0	0	1	1	0
2	0	0	1	0	1	0
3	0	1	0	0	1	0
4	0	1	0	1	1	0
5	0	1	1	0	1	0
6	0	1	1	1	1	0
7	1	0	0	0	1	0
8	1	0	0	1	1	0
9	1	0	1	0	1	0
10	1	0	1	1	1	0

Tabla 1: Lista de movimientos posibles en la 1ª aproximación.

Acción	IZQUIERDA (LEFT)	DERECHA (RIGHT)	ABAJO (DOWN)	SALTAR (JUMP)	CORRER (SPEED)	ARRIBA (UP)
0	0	0	0	0	0	1
1	0	0	0	0	1	1
2	0	0	0	1	0	1
3	0	0	0	1	1	1
4	0	0	1	0	0	0
5	0	0	1	0	1	0
6	0	1	0	0	0	1
7	0	1	0	0	1	1
8	0	1	0	1	0	1
9	0	1	0	1	1	1
10	0	1	1	0	0	0
11	0	1	1	0	1	0
12	0	1	1	1	0	0
13	0	1	1	1	1	0
14	1	0	0	0	0	1
15	1	0	0	0	1	1
16	1	0	0	1	0	1
17	1	0	0	1	1	1
18	1	0	1	0	0	0
19	1	0	1	0	1	0
20	1	0	1	1	0	0
21	1	0	1	1	1	0

Tabla 2: Lista de movimientos posibles 2ª aproximación.

A continuación se justifica el uso de ambas aproximaciones. Como se ha visto anteriormente es necesario limitar el número de posibles movimientos, en una primera fase de experimentación se decidió utilizar únicamente once movimientos posibles, ya que eran suficientes para completar un nivel de Mario AI de dificultad media que se probaba en esta fase de experimentación. En la segunda fase de experimentación se probaron niveles de Mario AI con dificultad mayor, en las primeras pruebas de esta fase de experimentación se vio la necesidad de ampliar el número de movimientos, veintidós movimientos, ya que en estos nuevos niveles se incorporan escaleras, las cuales Mario debe subir para seguir avanzando en el nivel, como se observa en la *Tabla 1* el bit correspondiente a arriba (UP) está a cero, por lo que Mario no podía subir las escaleras que se encontrase en el nivel, por este motivo, se decidió utilizar este bit en la segunda fase de experimentación al utilizarse nuevos niveles más difíciles en la experimentación, que se detallarán en el capítulo 4 de Experimentación.

Hasta ahora se han visto, dos características importantes de diseño de un AG, en concreto el tamaño del cromosoma, y el formato del gen de cuales está compuesto el cromosoma, el paso siguiente es, por tanto, la inicialización del cromosoma, que influye en la variabilidad genética de la población. En el AG genérico, el cromosoma se inicializa de forma aleatoria, es decir, se establece un valor cualesquiera a cada gen, dentro de los rangos definidos en el formato del gen. En este dominio, se tienen tres tipos de inicializaciones. La **primera** es una **inicialización aleatoria**, en la que se escoge un número binario aleatorio que se corresponda con uno de los 11 ó 22 movimientos

posibles, según la fase de experimentación. La **segunda inicialización** es **personalizada** o **guiada**, es decir, se inicializan los cromosomas con una serie de movimientos específicos, en este caso son "*DERECHA + VELOCIDAD*" o "*DERECHA + SALTAR + VELOCIDAD*", eligiéndose uno u otro de forma aleatoria en cada gen. Finalmente, la **tercera** y última inicialización consiste en mezclar las dos inicializaciones anteriores, llamada **inicialización mixta**, y así se pueden disminuir razonablemente las desventajas de ambas inicializaciones.

La ventaja principal de la inicialización aleatoria es que aporta una gran variabilidad genética a los cromosomas de la población con lo que permite encontrar una solución óptima en cualquier tipo de configuración del nivel de Mario AI, por el contrario, tiene la desventaja de que la convergencia del algoritmo es más lenta que el resto de inicializaciones. En la inicialización guiada se ha pretendido simular el comportamiento de un jugador humano, además intenta enfocarse en el objetivo principal de este tipos de juegos que es ir hacia la derecha de la pantalla para ir avanzando en el nivel, la ventaja de este tipo de inicialización es que converge a una solución óptima más rápido que en el resto de inicializaciones, en su contra se tiene que hay poca variabilidad genética por lo tanto si la configuración del nivel es compleja, el AG puede no encontrar ninguna solución. Por último, se tiene la inicialización mixta, debido a que las desventajas de las anteriores inicializaciones son significativas, se ha decidido intentar mitigarlas, mediante un nuevo tipo de inicialización que engloba en una las dos inicializaciones anteriores, teniendo en un cromosoma segmentos del mismo con inicialización aleatoria y otros segmentos del cromosoma con inicialización guiada y con la ayuda de los operadores genéticos del AG prácticamente desaparecen las desventajas de las dos primeras inicializaciones. En el capítulo 4 (Experimentación) se detalla con profundidad el comportamiento de estos tres tipos de inicializaciones en el dominio de Mario AI.

Una pregunta que se puede hacer el lector, respecto a las diferentes inicializaciones, es por qué elegir esos dos movimientos específicos en la segunda inicialización. En [46], se menciona que en niveles sencillos de Mario AI, sin enemigos y algunos obstáculos y saltos, podrían ser necesarios únicamente los movimientos "*DERECHA*" y "*SALTAR*" para completar el nivel, en verdad, para un jugador principiante, en este juego, los movimientos mayoritarios son ir hacia la derecha, saltar, y saltar hacia la derecha. Esto hizo pensar que inicializando los cromosomas a estos dos movimientos se ayudaría al agente a completar el nivel más rápidamente y fácilmente, como ya se verá en el capítulo 4.

Otra característica importante, que influye en el aprendizaje de los AG y en el resultado final, es el **número** máximo de **generaciones** que puede tener el AG. En el dominio de Mario AI se limita el número de evaluaciones que puede realizar el AG para aprender un nivel, este límite es de 10.001 evaluaciones, y limitando a su vez el número de generaciones. Una primera aproximación, más imprecisa, es asumir que en cada generación solo se evalúa el número de individuos que forman la población, por lo tanto, se puede calcular el número de generaciones, teniendo como número máximo de evaluaciones 10.000, de la siguiente forma, en la *Ecuación 2*:

$$N_{gen} = \frac{Nmax_{evaluaciones}}{Tam_{Pob}} = \frac{10000}{Tam_{Pob}}$$

Ecuación 2: Cálculo del Número de Generaciones en un AG

Esta aproximación no sería válida para la competición de Mario AI, ya que se debe ajustar teniendo en cuenta los nuevos individuos que se generan en cada generación según la tasa de mutación y la tasa de cruce. Pero es suficiente para el desarrollo del presente trabajo. Finalmente, se ha decidido por establecer dos tamaños diferentes de población, una con 20 individuos y la otra con 50 individuos, por lo tanto el número de generaciones máximo es 500 y 200 respectivamente.

Otro aspecto importante a tener en cuenta es la **granularidad** en la que se realizan las **acciones** de Mario, es decir, cada cuántos "tics" se ejecuta una nueva acción o dicho de otra manera el número de veces que el agente manda la misma acción al juego hasta que pasa a devolver la siguiente acción. Por ejemplo si tenemos una granularidad igual a 2, el agente devolverá dos veces la misma acción, hasta que pase a devolver la siguiente acción que tiene codificada. Esta propiedad permite simular el tiempo de reacción de un humano al pulsar los botones de un mando, por lo general, pulsar una tecla implica tener un retardo a la hora de soltar dicha tecla, durante ese instante los más probable es que transcurran varios "tics" del juego, con lo que la acción se ha repetido durante esos "tics", también influye en la fuerza del salto o la velocidad de Mario cuando corre.

3.7 Configuración del AG y sus parámetros

En la sección anterior se ha visto, que aspectos o peculiaridades del dominio del videojuego Mario AI influyen a la hora de configurar el algoritmo genético. Luego el paso siguiente es detallar cómo se configura el algoritmo genético y sus parámetros de configuración, que se irán viendo a lo largo de esta sección.

- **Codificación del individuo**

Como ya se ha comentado con anterioridad en el presente capítulo, un individuo del AG, que representa a un agente que juega un nivel de Mario AI, se codifica con un cromosoma que contiene 3.000 genes, cuyos genes representa uno de los 11 movimientos posibles, durante la primera fase de experimentación, ó 22 movimientos posibles, en la segunda fase de experimentación, como se muestra en la *Ilustración 29*.

Estos movimientos se pueden representar como una cadena de 6 bits, que codifica las 6 acciones posibles que puede realizar Mario, a saber, "IZQUIERDA", "DERECHA", "ABAJO", "SALTAR", "VELOCIDAD" y "ARRIBA", en ese orden. Aunque para una mayor comodidad a la hora de trabajar con los genes, se ha decidido usar números enteros en vez de números binarios, representando ambos el mismo valor numérico.

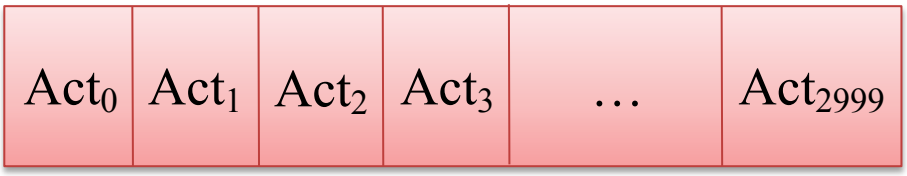


Ilustración 29: Esquema básico del individuo.

- **Operador de Selección**

El operador de selección elegido es el de Torneo, cuyo funcionamiento es: primero se asigna un tamaño de torneo (*t*) (normalmente pequeño), seguidamente se seleccionan *t* individuos de manera uniforme, a continuación se incluye en la siguiente población el de mayor aptitud y finalmente se repite el proceso tantas veces como sea el tamaño de la población.

Se ha decidido que el tamaño del torneo sea variable en función de un porcentaje de la población, ya que el tamaño de la población es configurable (por lo tanto, variable) se quiere dotar al operador la capacidad de adaptarse al tamaño de la población. Aunque se ha establecido que se tenga un tamaño mínimo de torneo de tres individuos.

Como se verá en el siguiente capítulo, Experimentación, durante la primera fase de experimentación se utilizará un algoritmo genético genérico para realizar las primeras pruebas, en esta fase se utiliza esta primera versión del operador, en la segunda fase de experimentación, en la cual se añade mayor dificultad a los niveles probados, se modifica el algoritmo genético para dotarle de información del dominio de Mario, por lo tanto, se decide modificar el operador de selección, únicamente añadiendo elitismo al operador, es decir, al inicio del operador se añade directamente al mejor individuo a la nueva población.

- **Operador de Cruce**

Igual que en el caso del operador de Selección, se tienen dos versiones diferentes del operador según la fase de experimentación, la primera versión, utilizada en la primera fase, es el operador genérico de cruce simple, y en la segunda fase, se tiene una nueva versión del operador que es una modificación específica en la cual se introduce información al dominio del juego de Mario AI.

La primera versión del operador de cruce, consiste en elegir de forma aleatoria un punto de cruce en cada cromosoma *n* (menor que el tamaño del cromosoma), en este caso, se necesita un mismo punto de cruce para el cromosoma "padre" y otro para el cromosoma "madre", como resultado del cruce se crea dos "hijos" nuevos, el primer "hijo" tiene un cromosoma compuesto por los *n* primeros genes del "padre" y los $(Tam_{cromo} - n)$ últimos cromosomas de la "madre"; y el segundo "hijo" tiene un cromosoma compuesto por los *n* primeros genes de la "madre" y los $(Tam_{cromo} - n)$ últimos cromosomas del "padre".

La segunda versión del operador cruce, nace de la necesidad de adaptar este operador al dominio del juego para mejorar el aprendizaje, para ello se asocia a cada gen la posición de Mario dentro del juego, es decir, se vincula la acción realizada con la posición de Mario en el juego, tras realizar dicha acción. Estas posiciones ayudarán a obtener un punto de cruce adecuado a este dominio. La principal característica del nuevo operador de cruce es la asociación entre la posición de Mario en el juego con el punto de cruce y las acciones realizadas por Mario. Para ello se almacena en una tabla hash (estructura de datos que asocia llaves o claves con valores), por una lado la acción y por otro las posiciones en las que se encuentra Mario al realizar la acción.

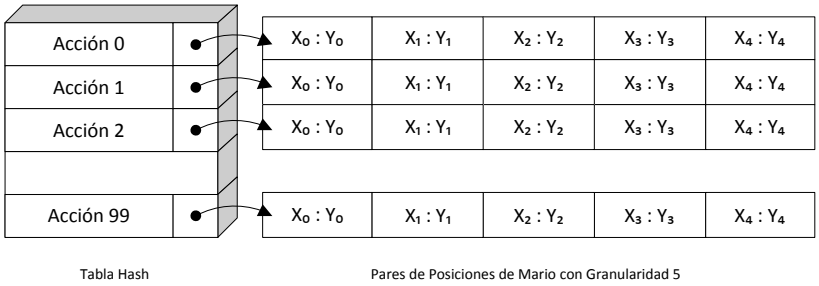


Ilustración 30: Representación de la Tabla Hash, con la asociación de acciones y posición de Mario en el juego.

Cada individuo tiene asociado una tabla hash propia en la que se almacena las acciones realizadas por el individuo y las posiciones de Mario al realizar dichas acciones. En la Ilustración 30 se representa la estructura de la tabla hash utilizada, por un lado se tiene la acción realizada (no se almacena la acción realizada, sino el número de acción realizada), estas acciones tienen asociada una serie de posiciones, dependiendo de la granularidad, en el ejemplo de la ilustración, se tiene una granularidad de 5, por lo tanto, la misma acción se ejecuta 5 veces hasta pasar a la siguiente acción, por lo que con esa acción ejecutada 5 veces, Mario ha pasado por 5 posiciones diferentes durante el juego, que son almacenadas en la tabla hash. El límite de pares acción-posiciones es de 100 pares, para limitar el espacio de memoria del sistema, y se almacena las últimas acciones realizadas, teniendo un comportamiento similar a una estructura de datos de tipo Cola.

La clave de este operador está, por lo tanto, en el concepto de punto de cruce, en lugar de tener un punto de cruce asociado a una posición del genoma, como en el caso del operador de cruce por defecto, se tiene un punto de cruce asociado a la posición de Mario en el juego. Con este punto de cruce se consigue determinar un punto o posición de Mario en común en el nivel entre dos individuos. El procedimiento a seguir para obtener estos puntos de cruce es el siguiente:

1. Se eligen dos individuos diferentes de la población, Ind_1 e Ind_2.
2. Se obtienen las tablas hash de cada individuo, HT_{Ind_1} e HT_{Ind_2}.
3. Para la primera HT_{Ind_1} se obtiene las acciones, empezando por el final, es decir, por la última acción realizada, $HT_{Ind_1} \rightarrow Act_{N-1}, N = 100$
4. Para la segunda HT_{Ind_2} de igual forma que la anterior se obtiene la última acción realizada $HT_{Ind_2} \rightarrow Act_{N-1}, N = 100$
5. A continuación se obtiene las posiciones asociadas a las acciones, $Pos_{HT_{Ind_1} \rightarrow Act_{N-1}}[\,]$ y $Pos_{HT_{Ind_2} \rightarrow Act_{N-1}}[\,]$.
6. Seguidamente se comprueba que haya alguna coincidencia de posiciones dentro de los conjuntos de posiciones, $Pos_{HT_{Ind_1} \rightarrow Act_{N-1}}[\,]$ y $Pos_{HT_{Ind_2} \rightarrow Act_{N-1}}[\,]$. Se establece un margen de coincidencia, es decir, no es necesario que ambas posiciones sean idénticas, sino que sean bastante próximas, por ejemplo, las posiciones {1,2 : 2,5} y {1,3 : 2,4} serían coincidentes.
7. Si hay coincidencias con algunas de los pares de posiciones, se para la búsqueda y se establece el punto de cruce como el par $\{\, \boldsymbol{HT_{Ind_1} \rightarrow Act_{N-1}}, \boldsymbol{HT_{Ind_2} \rightarrow Act_{N-1}} \}$.
8. Si no hay ninguna coincidencia, se escoge un nuevo conjunto de posiciones del segundo individuo, en concreto, las posiciones asociadas a la acción anterior a la anterior $Pos_{HT_{Ind_2} \rightarrow Act_{N-2}}[\,]$. Y se va al paso 6. Si no hay coincidencia se escoge

un nuevo conjunto de posiciones $Pos_{HT_{Ind_2} \rightarrow Act_{N-3}}[\,]$, luego $Pos_{HT_{Ind_2} \rightarrow Act_{N-3}}[\,]$, así hasta $Pos_{HT_{Ind_2} \rightarrow Act_0}[\,]$.

9. Si sigue sin haber coincidencias, se genera un nuevo conjunto de posiciones asociadas las acciones del primer individuo $Pos_{HT_{Ind_1} \rightarrow Act_{N-2}}[\,]$. Y se vuelve al paso 6. Si sigue sin haber coincidencias se escoge un nuevo conjunto de posiciones $Pos_{HT_{Ind_1} \rightarrow Act_{N-3}}[\,]$, luego $Pos_{HT_{Ind_1} \rightarrow Act_{N-3}}[\,]$, así hasta $Pos_{HT_{Ind_1} \rightarrow Act_0}[\,]$.
10. Finalmente si no hay ninguna coincidencia, se elige otro par de individuos, y se repite el proceso, hasta que se generen los puntos de cruce necesarios para el Operador.

Ya con estos puntos de cruce, se realiza el procedimiento habitual de un operador de cruce para cruzar dos individuos. Con esta nueva configuración ha sido posible que Mario supere niveles que con la configuración por defecto no se conseguía.

El número de veces que realizamos el cruce también depende de la población, según el porcentaje indicado, tasa de cruce. Además este porcentaje es parametrizable.

En el siguiente capítulo, se detallarán el funcionamiento y beneficios de esta última versión, sobre el aprendizaje del AG.

- **Operador de Mutación**

En este caso tenemos la misma situación que con el operador de cruce y de selección, por un lado se tiene una versión inicial genérica o por defecto a todos los dominios y otra versión adaptada al dominio del juego de Mario AI, para mejorar los resultados obtenidos con la primera versión.

En la primera versión, el operador de mutación, realiza la mutación para todos los individuos de la población, seguidamente para cada cromosoma del individuo, se muta un gen cada X genes, es decir, se muta cada gen del cromosoma con una probabilidad mutación de $\frac{1}{X}$, siendo X un valor entero mayor que 0. Este tipo de mutación genérica presenta algunos inconvenientes para este dominio, que se verán en el siguiente capítulo.

Para adaptar la mutación al dominio de Mario AI, se decide modificar este operador, el nuevo operador de mutación resultante muta, inicialmente, los dos últimos genes (o acciones) que han sido ejecutados por el individuo en la generación anterior. Para no limitar el número de genes a mutar, únicamente, a dos genes, se define el concepto de *ventana de mutación*, que permite aumentar el número de genes a mutar según la mejoría que presente el individuo tras cada evaluación; el valor inicial de la ventana, como ya se ha comentado, es de dos genes, y sí tras la evaluación del individuo no se avanza en el nivel evaluado la *ventana de mutación* aumenta, sí en el caso contrario el individuo mejora tras la evaluación, el valor de la *ventana de mutación* se establece al valor inicial, de dos. Por lo tanto, se tiene la ecuación siguiente:

$$Ventana_{MUTA}(x) = \begin{cases} Ventana_{MUTA}(x-1) * 2, & Fitness(x) \leq Fitness(x-1) \\ 2, & Fitness(x) > Fitness(x-1) \end{cases}$$

Ecuación 3: Ecuación de la ventana de mutación.

- **Función de Evaluación (o de Fitness)**

Como ya se mencionó en los requisitos de usuario (3.4), la función de evaluación viene impuesta por los organizadores de la competición de Mario AI, siempre que se quiera competir en ella. Para el presente trabajo se ha decido también utilizar esta función de optimización, ya que tiene en cuenta los aspectos más importantes del juego como son:

- Distancia física del juego que ha avanzado Mario, ***distancePassedPhys***.
- "Flores" (Flowers) cogidas por Mario, ***flowersDevoured***.
- Estado del Juego, ganado (1) o perdido (0), ***marioStatus***.
- Modo de Mario, *Fuego*, *Grande* o *Pequeño*, ***marioMode***.
- Setas cogidas por Mario, ***mushroomsDevoured***.
- Setas Verdes cogidas por Mario, ***greenMushroomsDevoured***.
- Monedas recogidas por Mario, ***coinsGained***.
- Bloques ocultos encontrados por Mario, ***hiddenBlocksFound***.
- Total de enemigo matados por Mario, ***killsTotal***.
- Enemigos matados por Mario saltando sobre ellos, ***killsByStomp***.
- Enemigos matados por Mario mediante disparos de fuego, ***killsByFire***.
- Enemigos matados por Mario mediante conchas, ***killsByShell***.
- Tiempo restante que quedaba para completar el nivel, ***timeLeft***.

Por lo tanto, la función utilizada es:

$$\begin{aligned} Fitness = {} & distancePassedPhys + flowersDevoured * 64 + marioStatus * 1024 \\ & + marioMode * 32 + mushroomsDevoured * 58 \\ & + greenMushroomsDevoured * 58 + coinsGained * 16 \\ & + hiddenBlocksFound * 24 + killsTotal * 42 + killsByStomp * 12 \\ & + killsByFire * 4 + killsByShell * 17 + timeLeft * 8 \end{aligned}$$

Ecuación 4: Función de evaluación del AG para participar en la competición de Mario AI

- **Parámetros del AG y de Mario AI**

Como se ha visto en los puntos anteriores, el AG utilizado puede tener varias configuraciones posibles que desconocemos inicialmente cuáles funcionan mejor y cuáles peor. Estas diferentes configuraciones están definidas por una serie de parámetros, que a la hora de la experimentación se especificarán con diferentes valores posibles para obtener la mejor configuración posible.

Los valores parametrizables, utilizados en los experimentos, son:

- ***Operador Genético de Selección***: este parámetro indica el operador de selección elegido, en este caso, para todos los experimentos se elige la selección por **Torneo** (genérico o adaptado) que tiene tres argumentos:
  - *Tamaño del torneo*, medido en un porcentaje de la población.
  - *Probabilidad* de seleccionar el *mejor individuo* del torneo.
  - *Ejecutar antes de los O. Genéticos de Cruce y Mutación*, indica si el O. Selección se ejecuta antes o después de realizar los Operadores Genéticos de Cruce y Mutación.

- ***Operador Genético Cruce***: este parámetro indica qué tipo de operador de cruce se quiere utilizar si el genérico o el adaptado al dominio. En cualquier caso ambos tienen el mismo parámetro que es:
  - *Porcentaje* de *Tasa* de *Cruce*, porcentaje sobre la población.
- ***Operador Genético Mutación***: este parámetro indica qué tipo de operador de mutación se quiere utilizar si el genérico o el adaptado al dominio. En cualquier caso ambos tienen el mismo parámetro que es:
  - *Tasa* de *Mutación* deseada, número que indica la probabilidad mutar un gen, $\frac{1}{X}$ siendo X la tasa de mutación elegida.
- ***Tamaño Población***: este parámetro indica el número de individuos que tiene la población.
- ***Número Máximo Evaluaciones***: este parámetro indica el número máximo de evaluaciones que se le permite realizar al AG. Este valor se especifica como 10.000 en cualquier caso, impuesto por los organizadores de la competición. Con este parámetro y el anterior obtenemos el número de generaciones.
- ***Granularidad***: este parámetro indica cada cuántos tics se envía una nueva acción al juego, o dicho de otra manera, indica cuantas veces se repite la misma acción en el juego.
- ***Tipo Inicialización***: este parámetro indica qué tipo de inicialización se tiene en los cromosomas de los individuos de la población. Pudiendo ser,
  - *Inicialización Aleatoria*, se elige aleatoriamente cualquiera de los movimientos posibles.
  - *Inicialización Guiada*, se elige aleatoriamente uno de los siguientes movimientos, "DERECHA + VELOCIDAD" o "DERECHA + SALTAR + VELOCIDAD".
  - *Inicialización Mixta*, es una inicialización que elige entre la aleatoria o la personalizada de forma aleatoria.
- ***Constante Ventana de Mejora***: este parámetro indica cuantas veces puede un individuo no mejorar, respecto a la evaluación anterior, para que se aumente su ventana de mutación. Es decir, si se tiene un valor de 3, el AG permite a un individuo estar tres generaciones sin mejorar su aptitud manteniendo su ventana de mutación, y si a la siguiente generación sigue sin mejor, se le aumenta la ventana de mutación, como se indica en la Ecuación 3.

➢ Pseudocódigo AG

El funcionamiento del AG diseñado para este trabajo se puede resumir de la siguiente forma:

```
procedimiento aprender()
    población ← inicializar individuos
    para genereación ← 0 hasta NUM_MAX_GENERACIONES hacer
            listaDeCandidatos ← población
            población_aux ← aplicar_OperadorMutación (población, parámetros)
            listaDeCandidatos.agregar(población_aux)
            población_aux ← aplicar_OperadorCruce (población, parámetros)
            listaDeCandidatos.agregar(población_aux)
            evaluar(listaDeCandidatos)
            población ← aplicar_OperadorSelección(listaDeCandidatos, parámetros)
    fin para
fin procedimiento
```

3.8 Diseño del Sistema Propuesto

En las dos secciones anteriores se ha visto los aspectos del juego Mario AI que determinan el diseño del AG, su comportamiento y forma de evaluar, también se ha detallado la configuración que tiene el AG y sus parámetros de configuración. El siguiente paso, es por lo tanto, llevar a cabo toda la teoría vista, al entorno del videojuego.

Como se ha comentado en Capítulo 2 el videojuego está desarrollado en JAVA, por lo tanto, se ha decidido desarrollar el controlador del AG también en el mismo lenguaje de programación. A continuación se detalla el diseño realizado de este controlador, para adaptarlo tanto al entorno de Mario AI como el entorno del AG, y que puedan interactuar sin problemas.

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos.

La principal restricción de la categoría "*Learning*" proporcionado por la competición es que todas las evaluaciones tienen que ejecutarse a través de la clase *LearningEvaluation*. Antes, en una clase personalizada, se podía guardar una tarea (task) personalizada y ejecutar las evaluaciones en cualquier punto del código cuando se necesitase actualizar la aptitud de un agente. Para paliar esta restricción, los desarrolladores de la competición proporcionaron una solución al fijar un punto de evaluación, que se representa al algoritmo de aprendizaje diseñado como una máquina de estados haciendo uso de la interfaz *LearningAgent* para permitir transiciones de estado.

Para facilitar esta integración al entorno de desarrollo del juego, se ha decido establecer tres módulos diferentes. En el primero módulo se encuentra las implementaciones de los agentes necesarios para interactuar con la clase *LearningTrack* que se encarga de realizar la competición, además de interactuar con el AG desarrollado y efectuar el aprendizaje diseñado. En el segundo módulo se encuentra las clases que implementan el AG diseñado, las cuales irán modificando los diferentes individuos para

evolucionar el sistema hasta la solución óptima. El tercer módulo se encarga de recoger la información proporcionada por el AG respecto a su aprendizaje y a su evolución, para ser guardada en ficheros de fácil lectura, como ficheros Excel o de texto plano.

- **Primer módulo – Agentes de Aprendizaje**

En este módulo se tiene dos clases, *GeneticAlgorithmAgent* y *HectorValero_GAAgent*.

La primera, *GeneticAlgorithmAgent*, implementa la interfaz *Evolvable* (Evolucionable) y *Agente*, esta clase representa a un agente que contiene a un individuo de la población del AG, por lo tanto evolucionable, que contiene la serie de acciones a realizar por Mario en el juego, cada vez que se evalúa a dicho individuo.

La segunda, *HectorValero_GAAgent*, implementa la interfaz *LearningAgent*, ésta interfaz es simple:

```
public interface LearningAgent extends Agent
{
    public void learn();
    public void giveReward(float reward);
    public void newEpisode();
}
```

Esas tres funciones representan como el *LearningTrack* interactúa con el Evolvable diseñado. En *LearningTrack* es donde ocurren las llamadas a "evaluar", además de los siguientes pasos:

1. learningAgent.init()
2. learningAgent.learn()
3. learningAgent.getBestAgent()
4. Task.runOneEpisode() (ejecutamos al mejor agente aprendido).

Volviendo a la clase, *HectorValero_GAAgent*, se puede intuir, al ver los métodos que tiene que implementar, que esta clase se encarga de realizar el aprendizaje del AG. Primero se tiene el método *init* el cual crea la población del AG e inicializa los cromosomas de los individuos de la dicha población, además de crear un objeto *GenotypeMarioAI*, que se comenta en el siguiente modulo. A continuación se tiene el método *learn*, en este método llama al método *evolve* del objeto *GenotypeMarioAI*, durante las generaciones especificas del AG, este método *evolve* reemplaza al método *newEpisode* que tiene que implementar *HectorValero_GAAgent*, y a su vez, el método *giveReward(float reward)* no tiene sentido con los AG, ya que este método está enfocado al Aprendizaje por Refuerzo.

En la imagen siguiente, *Ilustración 31*, se pueden observar las relaciones que poseen los dos agentes desarrollados, *HectorValero_GAAgent* y *GenotypeMarioAI*, con el motor del juego Mario AI.

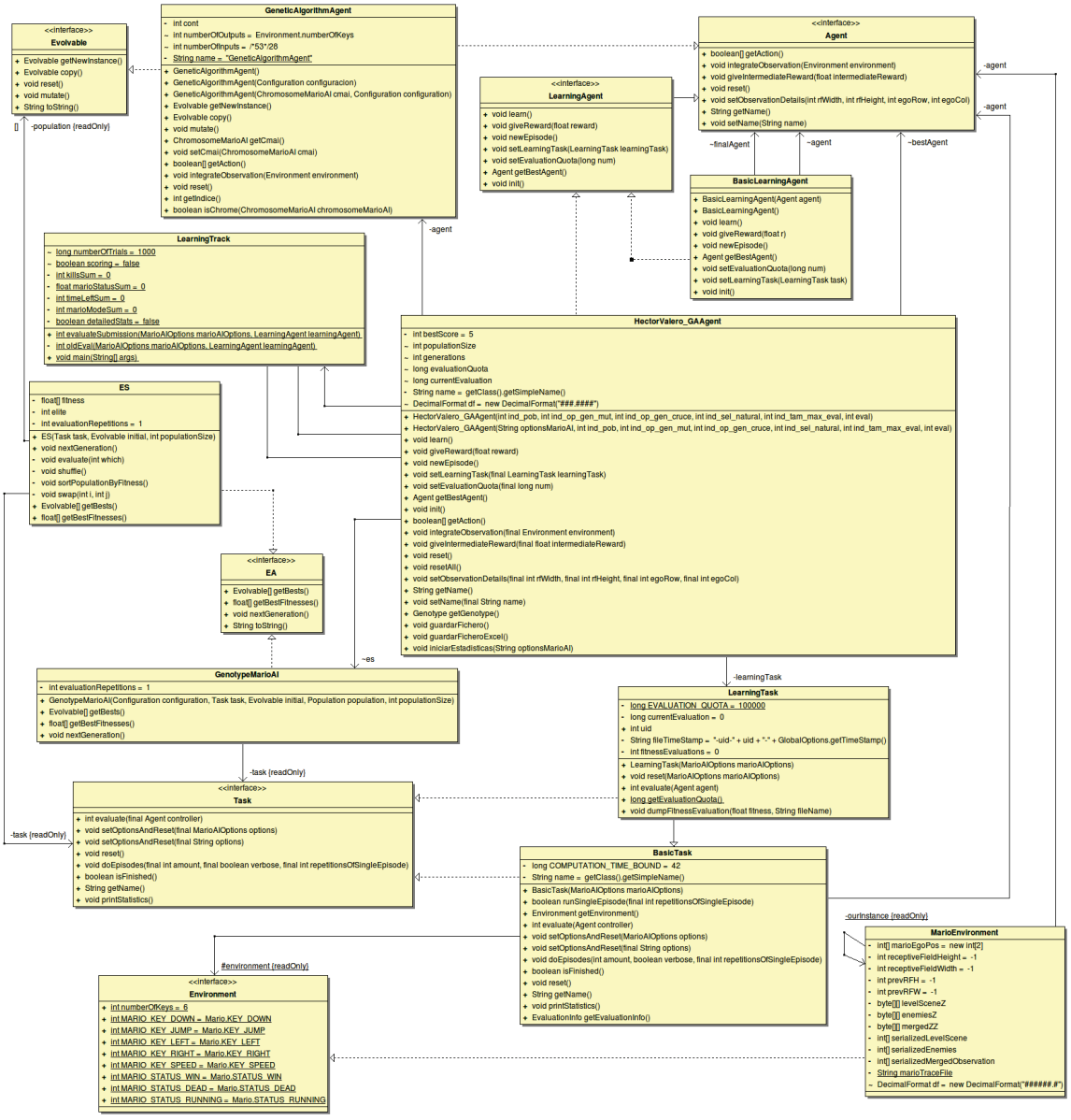


Ilustración 31: Diagrama de Clases del Módulo Agentes de Aprendizaje

El segundo módulo a analizar, contiene la implementación del AG visto en las dos secciones anteriores. Para no partir de cero en el desarrollo del AG, se ha decidido utilizar un framework (plataforma de software definida para un dominio en concreto) orientado a los AG. El framework elegido es el llamado "JGAP" (Java Genetic Algorithms Package) que es un componente de AG y PG suministrado como un framework de Java. Éste proporciona mecanismos genéticos básicos que se pueden utilizar fácilmente para aplicar principios evolutivos a las soluciones de los problemas.

Las primeras clases desarrolladas en este módulo, son *GenotypeMarioAI*, *ChromosomeMarioAI* y *MarioAIFitnessFunction*, tras los primeros experimentos se añadieron nuevas clases a este módulo, a saber, *CrossoverOperatorMarioAI*, *MutationOperatorMarioAI* y *TournamentSelectorMarioAI*. A continuación se hará una descripción de cada clase.

La clase *GenotypeMarioAI* sirve de enlace entre la clase *HectorValero_GAAgent* (ésta a su vez interactúa con el juego Mario AI) y el AG proporcionado por JGAP. *GenotypeMarioAI* extiende de *Genotype* del "JGAP" (es decir, *GenotypeMarioAI* deriva de *Genotype*, heredando sus propiedades, métodos y atributos) e implementa la interfaz *EA* (Algoritmo Evolutivo) de Mario AI. La clase *Genotype*, en el dominio del "JGAP", son poblaciones de cromosomas de longitud fija, cuando se invoca al método *evolve* (evolucionar) de una instancia de *Genotype* la totalidad de sus cromosomas evolucionan una generación, por lo tanto se entrevé, que dicha clase tiene el comportamiento de un AG, de ahí que se implemente la interfaz *EA* de Mario AI, ya que el método *nextGeneration* es, en definitiva, una representación del método *evolve* en el dominio de "JGAP".

En conclusión la función de la clase *GenotypeMarioAI* es realizar las funciones de la clase *Genotype* adaptado al dominio de Mario AI. Estas funciones son, en definitiva, aplicar los diferentes operadores genéticos del AG para ir evolucionando los individuos de la población a lo largo de *n* generaciones.

En la imagen siguiente, *Ilustración 32*, se observa las relaciones entre los diferentes componentes principales que forman el framework "JGAP".

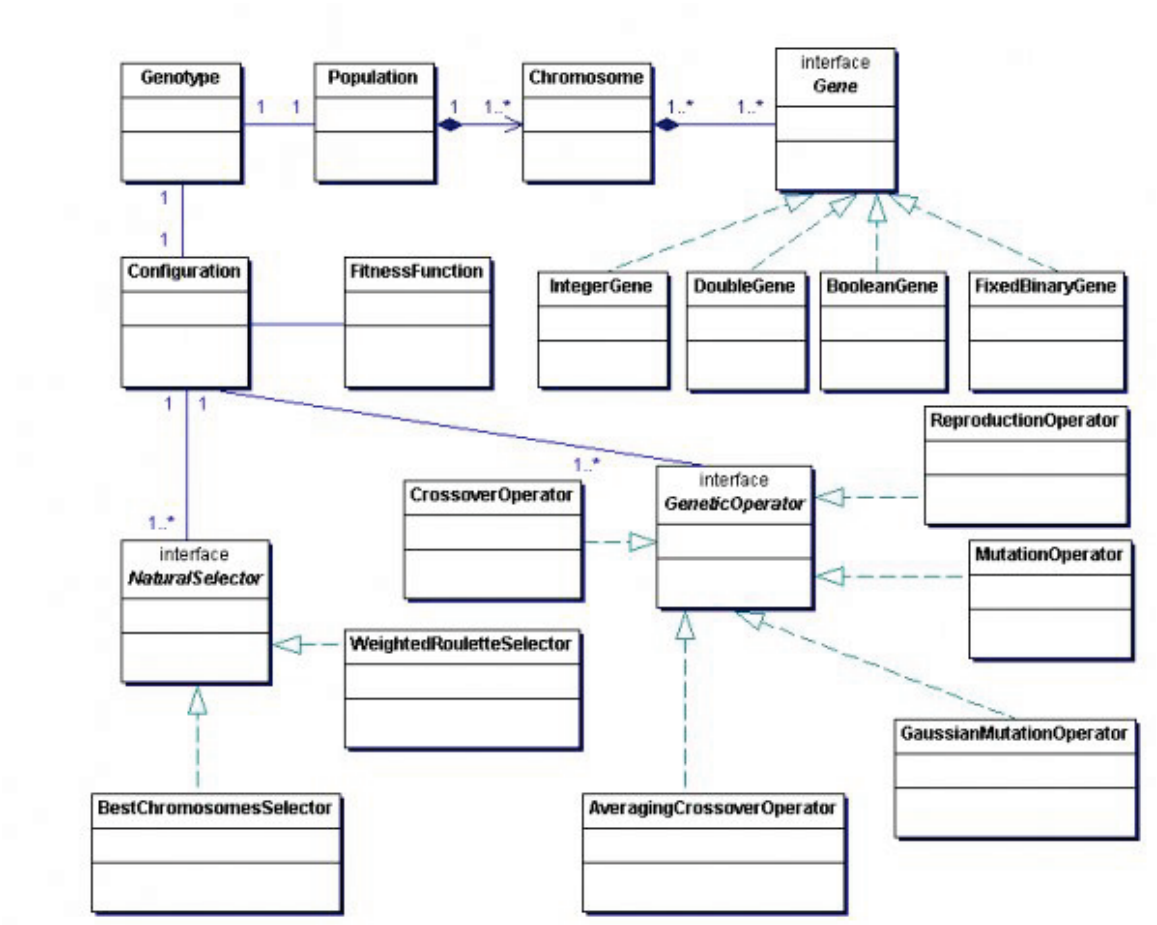


Ilustración 32: Diagrama de Clases que representa la estructura principal de JGAP.

La clase *ChromosomeMarioAI* extiende de la clase *Chromosome* de "JGAP" e implementa la interfaz *Evolvable* de Mario AI. *Evolvable* es una interfaz para un objeto *Evolvable* ("Evolucionable") genérico, estos objetos son necesarios para la ejecución de

un *EA*. La clase *Chromosome* representa a un cromosoma de AG (ya definido anteriormente), el cual contiene una colección de objetos *Gene*, que a su vez puede ser un objeto *IntegerGene*, que es el elegido para representar las acciones de un individuo.

La clase *MarioAIFitnessFunction* se extiende de la clase *FitnessFunction* de "JGAP", *FitnessFunction* es una clase abstracta que se debe extender e implementar el método *evaluate*. La función de optimización se suministra a un *Chromosome* para evaluarlo y debe devolver un decimal positivo que refleja su valor de fitness. Por lo tanto, la clase *MarioAIFitnessFunction* implementa el método *evaluate* modificado al dominio de Mario AI, y esto se realiza mediante la evaluación del cromosoma o individuo en el juego, obteniendo la puntuación otorgada por el juego, siendo ésta la aptitud asociada al cromosoma.

La clase *MutationOperatorMarioAI* es muy similar a la clase *MutationOperator* perteneciente al framework de "JGAP", salvo que se ha modificado para adaptarse al dominio de Mario AI, como se ha comentado en secciones anteriores.

A la clase *CrossoverOperatorMarioAI* le ocurre lo mismo que a la clase *MutationOperatorMarioAI*, tiene la misma estructura que la clase *CrossoverOperator* del framework "JGAP", pero se ha modificado para adaptarlo al problema de Mario AI.

Ambas clases, *MutationOperatorMarioAI* y *CrossoverOperatorMarioAI* se extienden de la clase abstracta *BaseGeneticOperator* que a su vez implementa la interfaz *GeneticOperator*, la cual tiene el método *operate* que se debe implementar, el cual realiza la operación, mutación o cruce, sobre los individuos de la población.

De igual forma que las dos clases anteriores, se tiene la clase *TournamentSelectorMarioAI*, que tiene un comportamiento muy similar a la clase de "JGAP", *TournamentSelector*, pero mejorando el operador, añadiendo siempre al mejor individuo de la población a lista de candidatos de la nueva población. Estas clases se extienden de la clase *NaturalSelectorExt*, que implementa el método *select* el cual se encarga de seleccionar los individuos para la nueva población.

Todas las clases de este módulo tienen relación con la clase *Configuration* del framework "JGAP". Esta clase representa la configuración actual de los complementos y los parámetros necesarios para ejecutar el algoritmo genético, como la función de optimización, el selector natural, los operadores genéticos, y así sucesivamente.

A continuación se muestra el diagrama de clases, *Ilustración 33*, que representa las relaciones entre las clases de este módulo, y las del primero módulo.

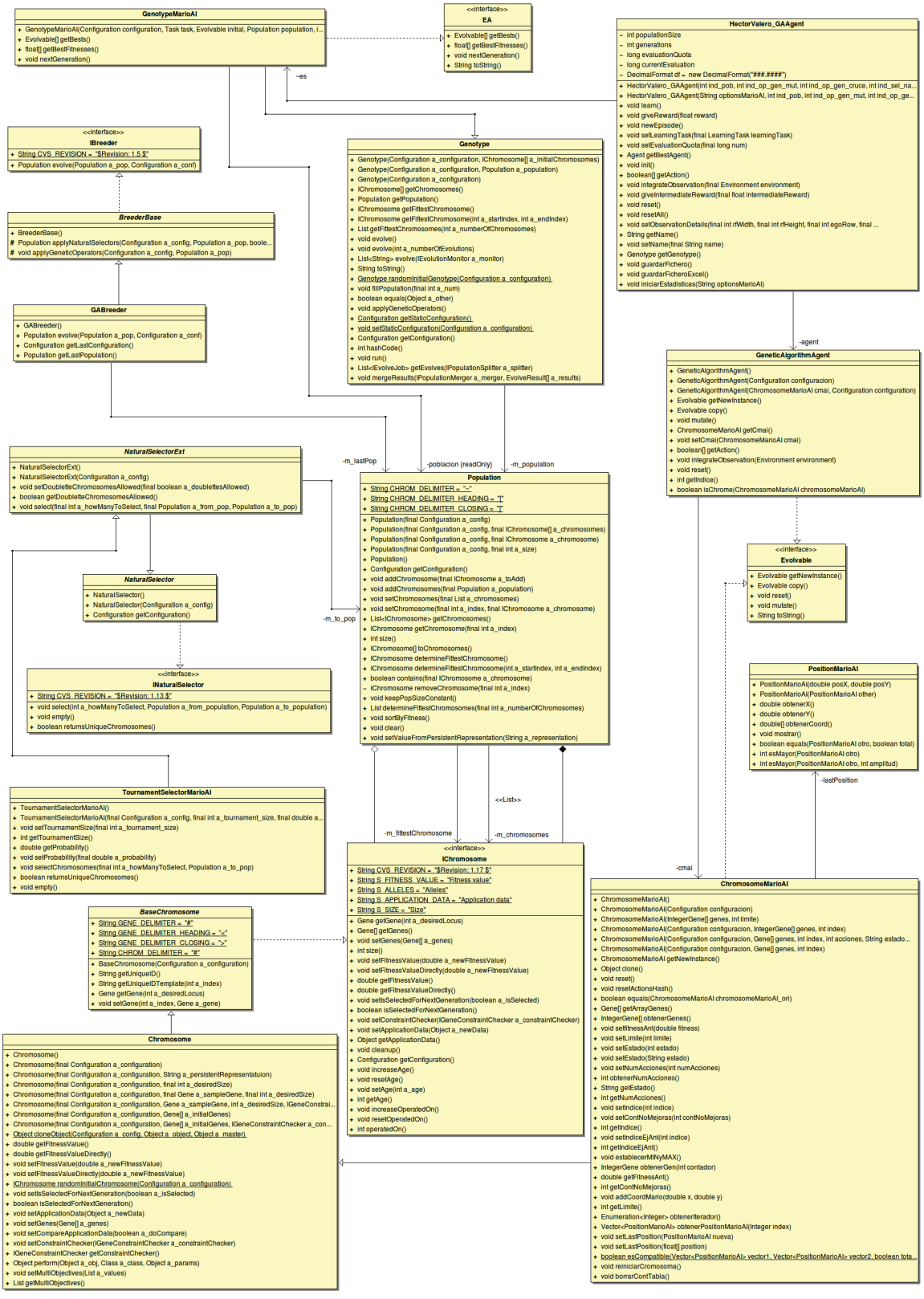
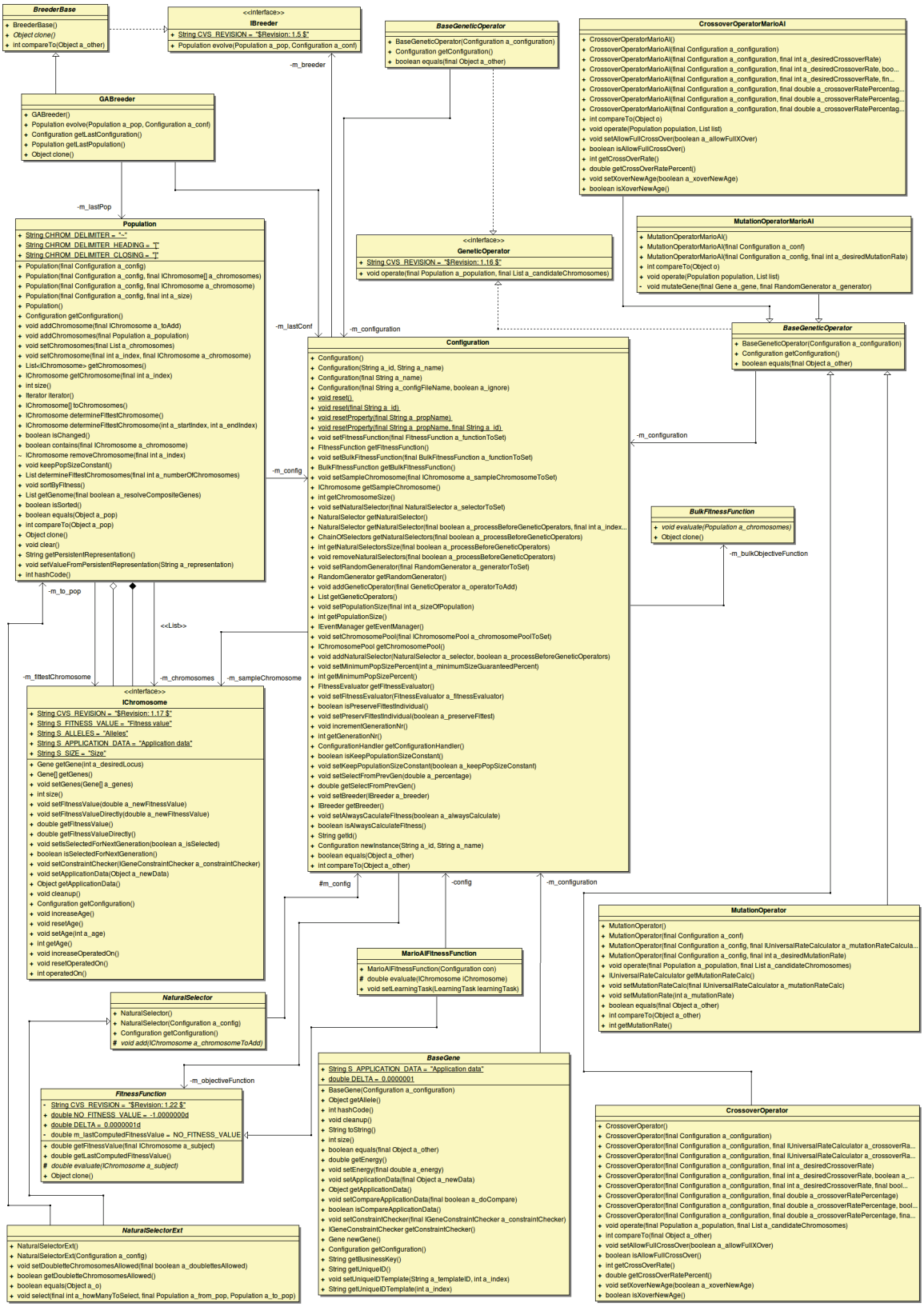


Ilustración 33: Diagrama de Clases del Módulo Genético

Y en la siguiente figura, *Ilustración 34*, se observas las relaciones de la clase *Configuration* de "JGAP" con el resto de clases comentadas.

Ilustración 34: Diagrama de Clases de los Módulos Primero y Segundo



- **Tercer Módulo – Estadísticas**

En este módulo están agrupadas las clases encargadas de generar las estadísticas generadas por el AG, y de almacenar los mejores individuos para cada prueba generada y así, posteriormente poder ejecutarlos de forma independiente y comprobar el comportamiento de Mario en esa prueba concreta.

En la clase *EstadisticasMario* se almacenan los datos generados por el AG en cada generación que son susceptibles a analizar, que son:

- Operadores Genéticos utilizados, representados por la clase *OpeGeneticos*.
- Operadores de Selección utilizados, representados por la clase *OpeSelección*.
- Individuos de la población por cada generación, representados por las clases *Individuos* e *Individuo*.
- Aptitud de cada individuo de la población por cada generación, representado por la clase *EstFitness*.
- Otra información relevante es:
  - Opciones del nivel de Mario AI evaluado.
  - Numero de Generaciones.
  - Tamaño Población
  - Tipo de inicialización.

Toda esta información puede ser luego almacenada en ficheros de texto o ficheros de hojas de cálculo.

La clase *XMLMarioAIDocumentation* proporciona la funcionalidad de transformar un objeto *ChromosomeMarioAI* a un fichero XML, para o bien visualizar sus acciones o para poder volver a ejecutarlo. También esta clase permite ejecutar un agente mediante la cadena de acciones a realizar, representada como números.

A continuación se muestra el diagrama de clases, *Ilustración 35*, que representa las relaciones entre estas clases y entre el primero modulo.

En el siguiente capítulo se verá el diseño de la experimentación a realizar, además de un análisis de los resultados. Para ello se hará una introducción indicando el porqué de la necesidad de dos fases de experimentación, seguidamente se expondrán los parámetros de configuración del AG utilizados en la experimentación y finalmente se mostrará un análisis de los resultados obtenidos en ambas fases de experimentación.

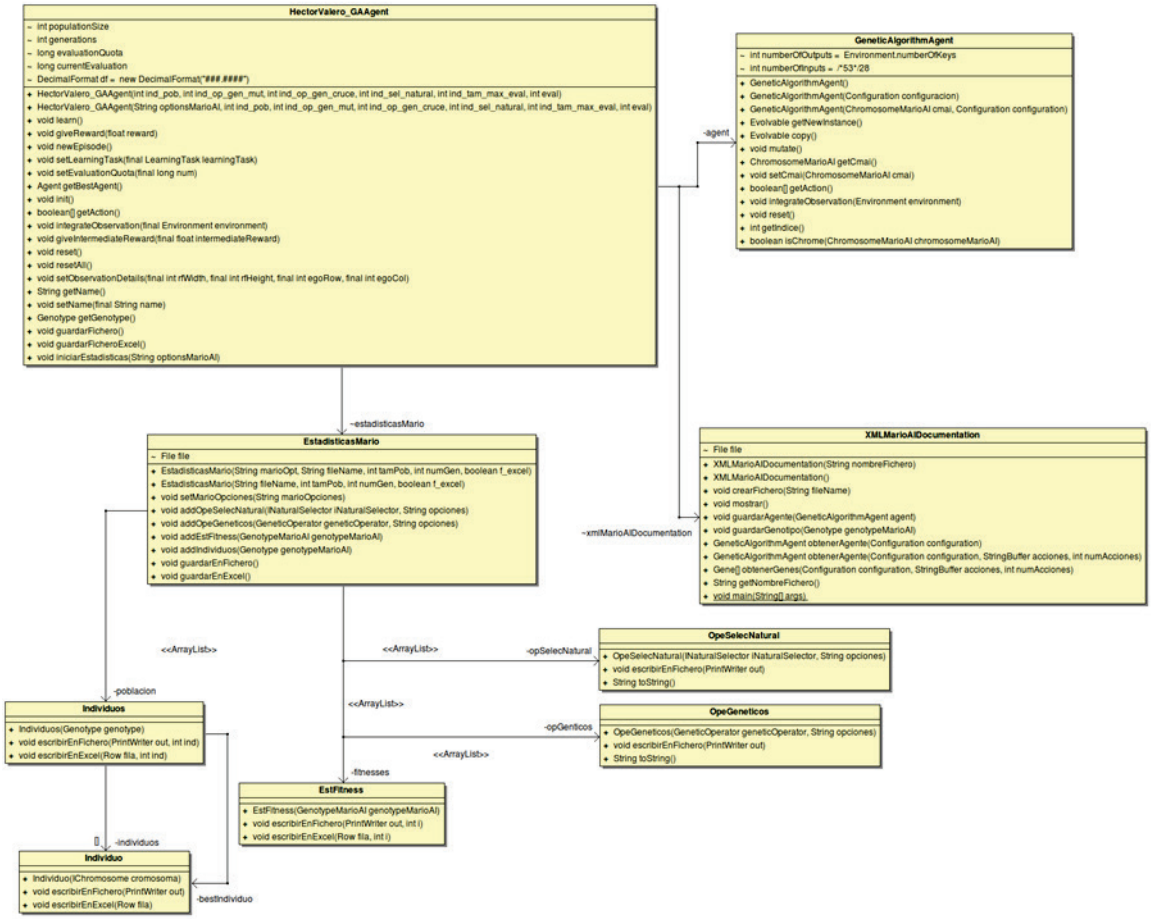


Ilustración 35: Diagrama de Clases del Módulo Tercero

Capítulo 4

Diseño de Experimentos y Resultados

4.1 Introducción

En el capítulo 4, se presentan los experimentos realizados con las diferentes configuraciones del AG diseñado. Estos experimentos se han desarrollado en dos fases principales. En la primera fase se diseñan los primeros experimentos a realizar, con los que se asegurará si los AGs pueden funcionar de forma óptima en el dominio de Mario AI y a su vez, proporcionarán una base de resultados para comparar con los futuros experimentos, es decir, en la primera fase, se establecerá un AG genérico, el cual no se ha incluido ninguna información del dominio del juego dentro de los operadores del AG, para ello se ha utilizado el AG genérico proporcionado por el framework *JGAP* incluidos los propios operadores genéticos de *JGAP*. Por lo tanto, en esta primera fase de experimentación, los resultados obtenidos derivan de un AG genérico no adaptado al dominio, en lo referente a los operadores genéticos. En esta primera fase los experimentos se realizarán únicamente sobre un nivel de Mario AI con dificultad media.

A continuación, en la segunda fase, tras analizar los primeros experimentos, e indicar las limitaciones encontradas con el AG genérico, se determina modificar y optimizar el AG desarrollado, dotándole de información del dominio del juego Mario AI, a través de los operadores genéticos y los parámetros de configuración del AG, por lo tanto, se generarán los últimos experimentos que deberán obtener mejores resultados que en la primera fase, es decir, con la modificación del AG, éste tendrá que ser capaz de

completar niveles de gran dificultad, que el AG genérico no fue capaz de completar, lo que permitirá participar en la competición *Mario AI Championship 2011*, esperando tener un buen papel.

Para realizar los experimentos pertinentes, se debe diseñar un plan de experimentación, en el cual se representan las diferentes configuraciones definidas por los valores de los parámetros del AG. Este plan debe agrupar los experimentos en varios grupos para poder analizar más fácilmente los resultados de la experimentación. Estos grupos son la evolución de la aptitud de la población durante todas las generaciones, el promedio de la aptitud de la población, y el número de individuos que han sido capaces de completar el nivel satisfactoriamente por cada prueba.

Como es de esperar la realización de los experimentos se tiene que hacer de forma automática mediante un archivo de procesamiento por lotes en un entorno de Windows, que a su vez éste ejecuta un procedimiento el cual indica qué valores tienen que tomar los diferentes parámetros que componen una configuración específica del AG. Además este procedimiento se encarga de ejecutar el módulo estadísticas del sistema, el cual proporciona toda la información, necesaria para realizar el análisis de los resultados, agrupado en ficheros de texto o en un editor de hojas de cálculo.

A continuación, en las secciones siguientes se hará un repaso de los parámetros de configuración del AG, ya vistos en el apartado 3.7 del capítulo 3 seguidamente se expondrá la batería de experimentos correspondientes al AG simple junto con un análisis de resultados obtenidos. A continuación se expondrá las mejoras realizadas sobre el AG, y se indicará la batería de pruebas con el AG mejorado con su correspondiente análisis de resultados tras estas pruebas.

4.2 Parámetros de Configuración

A continuación se van a recordar los parámetros de configuración del AG, que se vieron en el apartado 3.7 del capítulo 3, que se van utilizar en la primera fase de experimentación para el primer diseño del AG, el AG Simple o Genérico. A continuación, en el apartado siguiente, se expondrán los valores concretos que tomarán los parámetros de configuración del AG Simple.

a) **Tipo de Inicialización**: con este parámetro se pretende establecer el tipo de inicialización que tomarán los individuos de una población, del AG, en el momento de su creación.

b) **Granularidad**: este parámetro indica cada cuántos tics (instantes de tiempo) del sistema se envía una nueva acción del personaje Mario en el juego, o dicho de otra forma, indica cuántas veces se repite una misma acción hasta que se establece otra nueva acción de Mario.

c) **Tamaño de la Población**: permite establecer el número de individuos que forman una población del AG.

d) **Número Máximo de Evaluaciones**: este parámetro indica en el número máximo de evaluaciones (partidas del juego Mario AI). Debido a las reglas de la competición de *Mario AI Championship*, se impone que el número máximo de evaluaciones sea de 10.000 evaluaciones.

e) **Operador Genético de Selección**: con este parámetro se pretende establecer el tipo de operador de Selección, asignando también los valores correspondientes a los parámetros de dicho operador.

f) **Operador Genético de Cruce**: este parámetro determina qué tipo de operador de cruce se quiere utilizar en el AG.

g) **Operador Genético de Mutación**: permite establecer el tipo de operador de mutación se quiere utilizar en el AG.

h) **Opciones del nivel de Mario AI**: El juego Mario AI permite configuración ciertos parámetros del juego, como dificultad, longitud del nivel, etc. A continuación se exponen las opciones del nivel utilizadas en la experimentación:
   - "**-vis off**": Visualización del juego, permite seguir visualmente las partidas o jugadas que se ejecutan en la evaluación del AG. Los valores que puede tomar son "*on*" para activar la visualización y "*off*" para desactivar la visualización.
   - "**-lhb on**": Bloques ocultos, esta opción se utiliza para activar/desactivar los bloques ocultos, por lo tanto, en la experimentación se encuentra activo.
   - "**-lt 0**": Tipo de Nivel, *Overground* (0, en la superficie), *Underground* (1, subterráneo), *Castle* (2, castillo) y *Random* (3, aleatorio). Está activo el primer tipo de nivel.
   - "**-ld 1**": Dificultad del nivel, un entero entre 0 y 12, en este caso es de 1.
   - "**-ll 300**": Longitud del nivel, un entero entre 50 y $2^{31} - 1$, en este caso es de 300.
   - "**-ls 20002**": Semilla de aleatorización del nivel, un entero entre 0 y $2^{31} - 1$, en este caso es de 20002.

4.3 Batería de Experimentos del AG Simple

Como ya se ha comentado anteriormente, en esta primera fase, se realizan los primeros experimentos, los cuales deben proporcionar la información suficiente para establecer si los AG son una técnica de la IA apropiada. Con estos experimentos se sentará las bases de los resultados y mejores configuraciones del AG para comparar en futuros experimentos.

En esta sección del capítulo, se especificarán los valores de los parámetros de configuración del AG Simple que se desea probar en la experimentación.

I. **Tipo de Inicialización**, dos valores posibles:
   a. *Inicialización Aleatoria*: se elige aleatoriamente cualquiera de los movimientos o acciones posibles.
   b. *Inicialización Guiada:* se elige aleatoriamente uno de los siguientes movimientos, "DERECHA+VELOCIDAD" ó "DERECHA+SALTAR+VELOCIDAD".

II. **Granularidad**, tres valores posibles:
   a. *1 acción.*
   b. *2 acciones.*
   c. *5 acciones.*

III. **Tamaño de la Población**, dos valores posibles:
   a. *20 individuos*
   b. *50 individuos*

IV. **Opciones de nivel de Mario AI**, un valor posible:
   a. *-vis off -lhb on -lt 0 -ld 1 -ll 300 -ls 20002*

V. **Número Máximo Evaluaciones**, un valor posible:
   a. *10.000 evaluaciones.*

VI. **Operador de Selección**, un valor posible, se ha decidido mantener este parámetro fijo a un valor ya conocido por sus buenos resultados, por la necesidad de simplificar el número de pruebas o experimentos realizados, porque si se hubiese establecido varios valores para este parámetro, el número de pruebas realizadas sería intratable:
   a. *Selección por Torneo genérico del framework JGAP (org.jgap.impl.TournamentSelector)*, con los parámetros:
      i. Tamaño del Torneo: 0.15 (15% de la población)
      ii. Probabilidad de Seleccionar el Mejor Individuo: 1.0 (100%)
      iii. Ejecutar después de los Operadores Genéticos de Cruce y Mutación

VII. **Operador Genético de Mutación**, un valor posible:
   a. *Ope. Gen. de Mutación genérica del framework JGAP (org.jgap.impl.MutationOperator)*, con los parámetros:

i. Tasa de Mutación Deseada: 20 (1 gen mutado de cada 20 genes de forma aleatoria, 5% del cromosoma).
ii. Tasa de Mutación Deseada: 30 (1 gen mutado de cada 30 genes de forma aleatoria, 3,3% del cromosoma).
iii. Tasa de Mutación Deseada: 40 (1 gen mutado de cada 40 genes de forma aleatoria, 2,5% del cromosoma).
iv. Tasa de Mutación Deseada: 50 (1 gen mutado de cada 50 genes de forma aleatoria, 2% del cromosoma).
v. Tasa de Mutación Deseada: 75 (1 gen mutado de cada 75 genes de forma aleatoria, 1,3% del cromosoma).
vi. Tasa de Mutación Deseada: 100 (1 gen mutado de cada 100 genes de forma aleatoria, 1% del cromosoma).

VIII. Operador Genético de Cruce, un valor posible**:**

a. Ope. Gen. de Cruce genérica del framework JGAP (org.jgap.impl.CrossoverOperator), con los parámetros:

i. Tasa de Cruce (% población): 0.1 (10 % de la población)
ii. Tasa de Cruce (% población): 0.2 (20 % de la población)
iii. Tasa de Cruce (% población): 0.3 (30 % de la población)
iv. Tasa de Cruce (% población): 0.4 (40 % de la población)
v. Tasa de Cruce (% población): 0.5 (50 % de la población)

Para los parámetros Tipo de Inicialización, Granularidad, Tamaño de la Población, y Operadores Genéticos de Selección, Cruce y Mutación se ha decido probar varias configuraciones posibles, ya que a priori se desconocen cuáles son los valores más eficientes para dichos parámetros, por lo que uno de los objetivos de la experimentación es obtener la configuración de los parámetros del AG que obtengan un mejor comportamiento a la hora de superar el nivel de Mario AI.

A continuación, en la figura se muestra una tabla resumen *Tabla 3* con todas las pruebas realizadas agrupadas en colores, que identifican a una hoja de cálculo, que contiene las estadísticas de la experimentación. Como se observa, en total se ha realizado 360 experimentos, cada uno de ellos con un total de 10.000 partidas o evaluaciones, por lo que en total se ha ejecutado unas 3.600.000 partidas a Mario AI de forma automática.

Por último, destacar que en esta primera experimentación se ha decidido que Mario tenga 12 movimientos posibles, para reducir el espacio de búsqueda. Las acciones o movimientos utilizados son los mostrados en la *Tabla 1* del apartado 3.6 de capítulo 3.

				TAMAÑO DE POBLACION											
				20						50					
				Operador de Mutación						Operador de Mutación					
				20	30	40	50	75	100	20	30	40	50	75	100
INICIALIZACIÓN GUIADA	GRANURALIDAD: 1	Operador de Cruce	0,1	X	X	X	X	X	X	X	X	X	X	X	X
			0,2	X	X	X	X	X	X	X	X	X	X	X	X
			0,3	X	X	X	X	X	X	X	X	X	X	X	X
			0,4	X	X	X	X	X	X	X	X	X	X	X	X
			0,5	X	X	X	X	X	X	X	X	X	X	X	X
	GRANURALIDAD: 2	Operador de Cruce	0,1	X	X	X	X	X	X	X	X	X	X	X	X
			0,2	X	X	X	X	X	X	X	X	X	X	X	X
			0,3	X	X	X	X	X	X	X	X	X	X	X	X
			0,4	X	X	X	X	X	X	X	X	X	X	X	X
			0,5	X	X	X	X	X	X	X	X	X	X	X	X
	GRANURALIDAD: 5	Operador de Cruce	0,1	X	X	X	X	X	X	X	X	X	X	X	X
			0,2	X	X	X	X	X	X	X	X	X	X	X	X
			0,3	X	X	X	X	X	X	X	X	X	X	X	X
			0,4	X	X	X	X	X	X	X	X	X	X	X	X
			0,5	X	X	X	X	X	X	X	X	X	X	X	X
INICIALIZACIÓN ALEATORIA	GRANURALIDAD: 1	Operador de Cruce	0,1	X	X	X	X	X	X	X	X	X	X	X	X
			0,2	X	X	X	X	X	X	X	X	X	X	X	X
			0,3	X	X	X	X	X	X	X	X	X	X	X	X
			0,4	X	X	X	X	X	X	X	X	X	X	X	X
			0,5	X	X	X	X	X	X	X	X	X	X	X	X
	GRANURALIDAD: 2	Operador de Cruce	0,1	X	X	X	X	X	X	X	X	X	X	X	X
			0,2	X	X	X	X	X	X	X	X	X	X	X	X
			0,3	X	X	X	X	X	X	X	X	X	X	X	X
			0,4	X	X	X	X	X	X	X	X	X	X	X	X
			0,5	X	X	X	X	X	X	X	X	X	X	X	X
	GRANURALIDAD: 5	Operador de Cruce	0,1	X	X	X	X	X	X	X	X	X	X	X	X
			0,2	X	X	X	X	X	X	X	X	X	X	X	X
			0,3	X	X	X	X	X	X	X	X	X	X	X	X
			0,4	X	X	X	X	X	X	X	X	X	X	X	X
			0,5	X	X	X	X	X	X	X	X	X	X	X	X

Tabla 3: Tabla resumen con todas las pruebas realizadas en la primera fase

4.4 Análisis de Resultados del AG Simple

El primer análisis que se debe realizar es determinar si los AGs se pueden aplicar a juego Mario AI, para ello, se comprueba si hay una evolución en al aprendizaje del AG Simple utilizado. Las gráficas de evolución, *Ilustración 36*, *Ilustración 37*, *Ilustración 38* e *Ilustración 39*, permiten demostrar que ha habido un aprendizaje desde los individuos de los primeros instantes de las generaciones iniciales a los individuos de las generaciones finales del AG. Se observa en dichas gráficas, que el valor de la aptitud al comienzo del aprendizaje se tiene valores en torno a 2.500 y 3.000, mientras que al final del aprendizaje se tiene valores en torno a 10.000, por lo que se aprecia una clara evolución en el aprendizaje del AG. También se puede observar que la condición de parada del AG llega en las generaciones 400 ó 200, para tamaños de población de 20 y 50 individuos respectivamente, esto es debido a la limitación de 10.000 evaluaciones como máximo impuesta por la competición, como ya se mencionó en el apartado 3.6.

El siguiente paso a realizar es determinar cuál es la mejor configuración, con la cual se obtendrán los mejores resultados posibles, para ello, se necesita analizar los resultados obtenidos en las pruebas realizadas. Para examinar los resultados se comparan dos

medidas, que marcan cuánto de buena es la configuración del algoritmo para completar el nivel especificado. Estas medias son, el promedio de la aptitud de todos los individuos de la población por cada generación y el número de individuos que han sido capaces de completar el nivel satisfactoriamente por cada prueba.

Como es de esperar, es inviable analizar uno por uno todos los parámetros para todas las pruebas, por lo tanto, se ha decidido comparar los mejores resultados por cada parámetro configurable con respecto al parámetro de Tamaño de Población, es decir, para conocer qué valor del parámetro Tipo de Inicialización funciona mejor, se comparará los mejores resultados de tipo de inicialización aleatoria con el tamaño de población 20 y 50, y de igual forma con el tipo de inicialización guiada.

A continuación se irán analizando los diferentes valores de cada parámetro para establecer cuál funciona mejor.

- **Parámetro de configuración "Tipo de Inicialización"**

Más adelante se muestran varias gráficas que reflejan el promedio de la aptitud de todas las generaciones por cada tipo de prueba realizada. En las ilustraciones, *Ilustración 40* e *Ilustración 41*, se tiene una inicialización Guiada con un tamaño de población de 20 y 50 individuos respectivamente; mientras que en las ilustraciones *Ilustración 42* e *Ilustración 43*, se tiene una inicialización Aleatoria con un tamaño de población de 20 y 50.

Para un tamaño de población de 20 individuos (*Ilustración 40* e *Ilustración 42*) se puede observar que el promedio de la aptitud es mayor en la primera gráfica que en la segunda, en 1.400 puntos de diferencia; además es mayor para todas las configuraciones realizadas. En el caso de un tamaño de población de 50 individuos (*Ilustración 41* e *Ilustración 43*) se tiene el mismo comportamiento que en el caso de 20 individuos, así el promedio de la aptitud es mayor en la primera gráfica, con una diferencia de 1.254 puntos. En la *Tabla 4*, se puede comprobar dicho comportamiento.

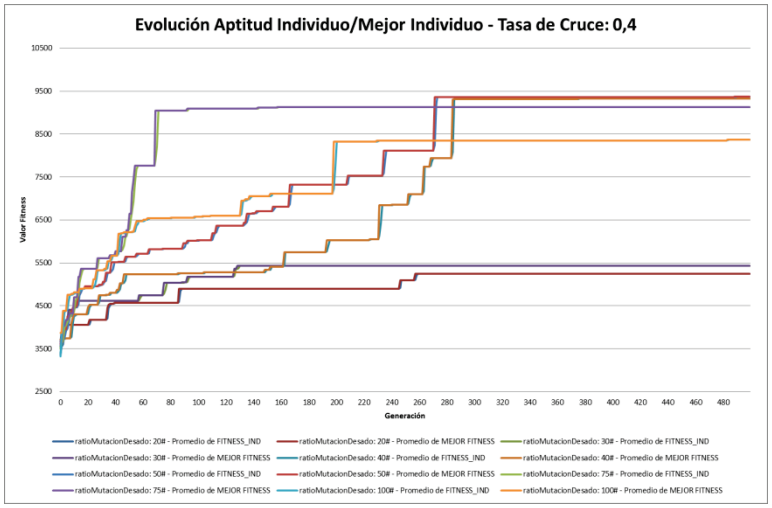


Ilustración 36: Población: 20, Inicialización Guiada

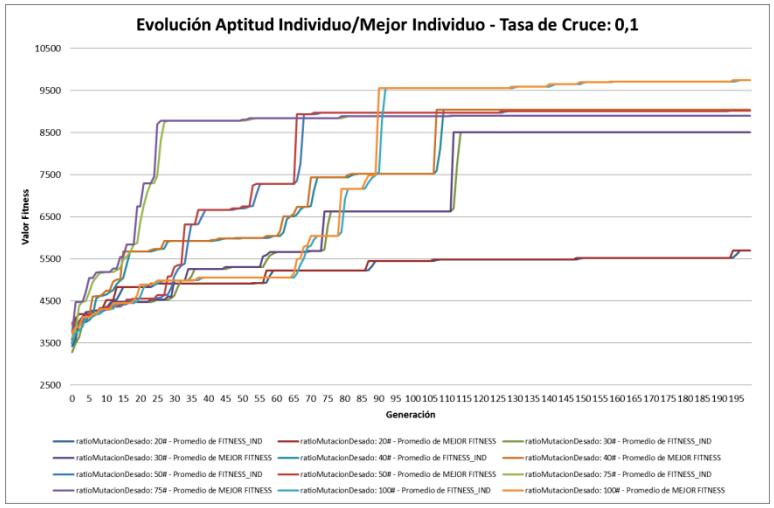


Ilustración 37: Población: 50, Inicialización Guiada

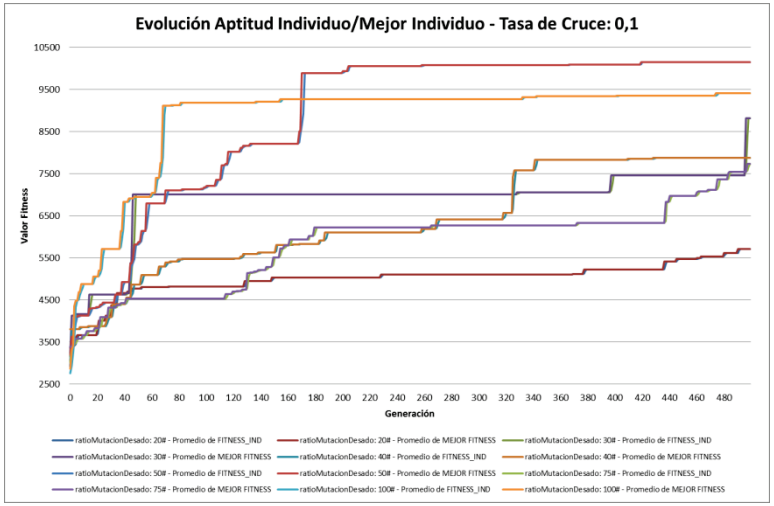


Ilustración 38: Población: 20, Inicialización Aleatoria

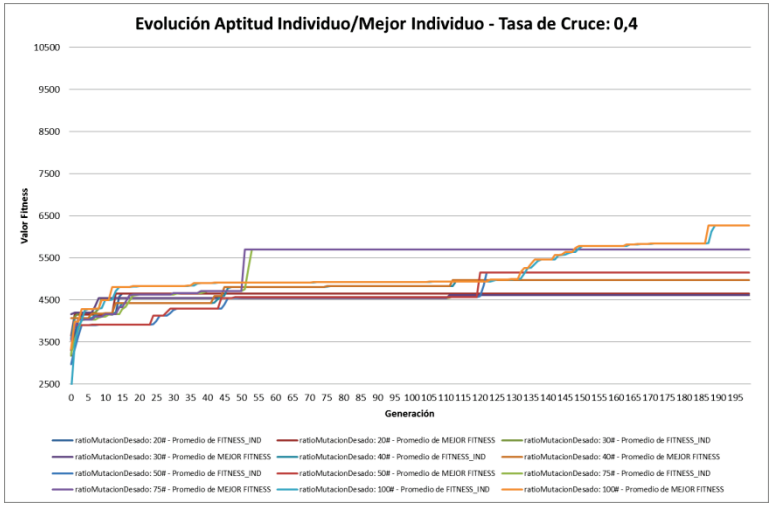


Ilustración 39: Población: 50, Inicialización Aleatoria

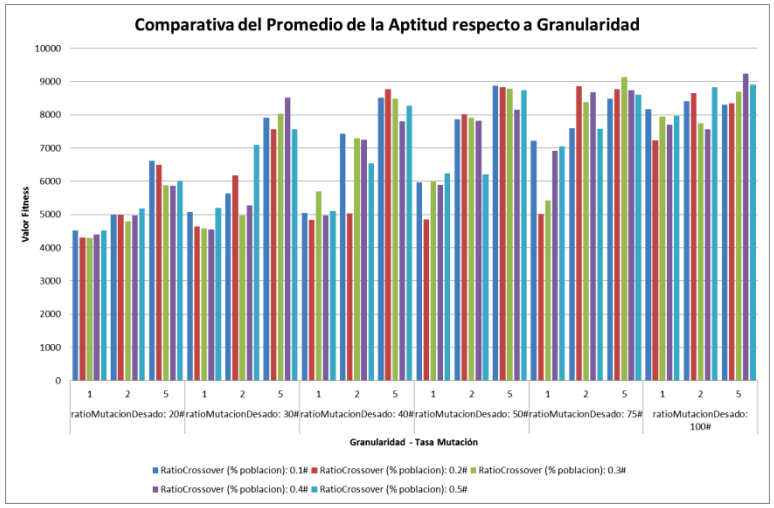


Ilustración 40: Población: 20, Inicialización Guiada

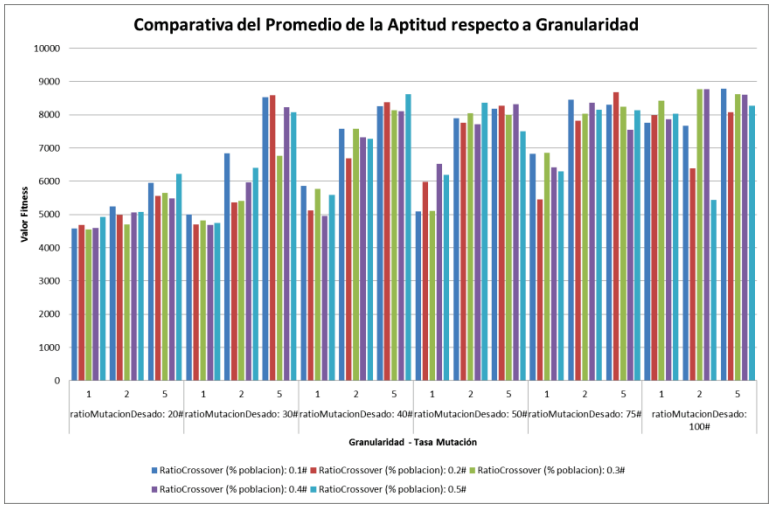


Ilustración 41: Población: 50, Inicialización Guiada

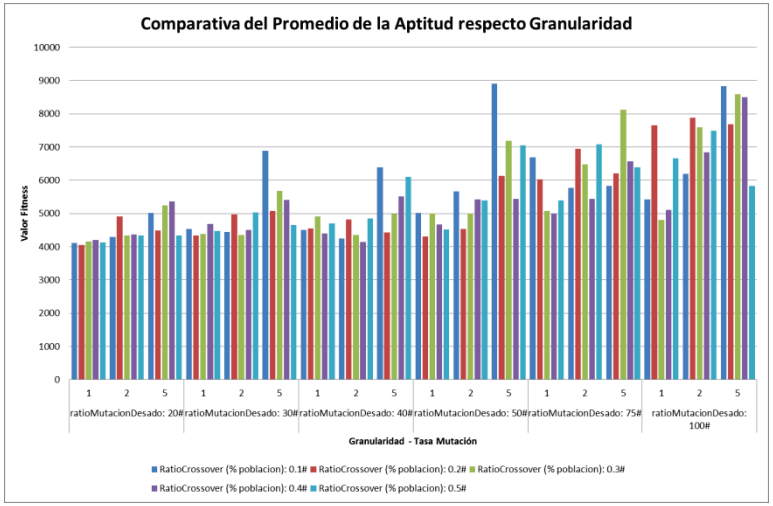


Ilustración 42: Población: 20, Inicialización Aleatoria

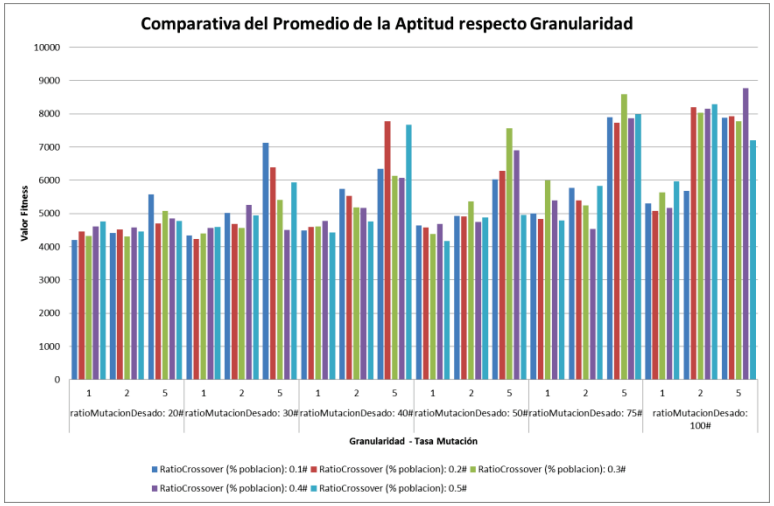


Ilustración 43: Población: 50, Inicialización Aleatoria

		Tamaño de Población	
		20	**50**
Inicialización	**Guiada**	6.913,56	6.876,89
	Aleatoria	5.513,36	5.622,30

Tabla 4: Promedio de la Aptitud de los individuos, según la configuración del AG

Hasta ahora se ha visto, claramente, que funciona mejor la inicialización Guiada que la inicialización Aleatoria, pero falta por conocer cuál es el motivo de esta diferencia entre inicializaciones. Para determinar este motivo se utilizarán como apoyo las primeras gráficas del apartado, *Ilustración 36*, *Ilustración 37*, *Ilustración 38* e *Ilustración 39*, en donde se ve la evolución de la aptitud a lo largo de las generaciones.

En las cuatro gráficas, mencionadas, se observa que tienen un comportamiento similar, pero con una diferencia característica. En las dos primeras gráficas, con inicialización guiada, los incrementos de la aptitud son más bruscos, que en las ultimas gráficas, con inicialización aleatoria, que tiene un incremento de la aptitud algo más suave. Además también se observa que en la inicialización personalizada se consigue antes sobre pasar el límite de 8.000 puntos, que representa que se ha completado el nivel probado. Esto se debe a que la inicialización guiada ayuda a progresar a Mario durante el nivel, ya que la mayor parte de las veces está avanzando en el nivel; mientras que en la inicialización aleatoria se tarda más tiempo en conseguir este comportamiento deseado. Aunque a priori puede parecer que la inicialización aleatoria es mejor, ya que tiene una mayor diversidad genética, que la inicialización personalizada, los resultados nos indican todo lo contrario, el motivo principal de esta situación es la limitación del número de evaluaciones que se pueden hacer durante una prueba, que en número de generaciones se traduce a 500 y 200 generaciones para una población de 20 y de 50 individuos respectivamente, que para una configuración genérica es un número escaso de generaciones, pero que viene impuesta por las reglas de la competición de Mario AI.

Las cuatro gráficas, que se ven a continuación, muestran el número de victorias (o nivel completado por Mario) según la configuración de los parámetros del AG. El número máximo de victorias que se puede tener es, por lo tanto, el número máximo de evaluaciones, 10.000. De igual forma que en el caso anterior, las ilustraciones *Ilustración 44* e *Ilustración 45* se tiene una inicialización Guiada con un tamaño de población de 20 y de 50 individuos, y en las ilustraciones, *Ilustración 46* e *Ilustración 47*, se tiene una inicialización Aleatoria con un tamaño de población de 20 y de 50. Para un tamaño de población de 20 individuos (*Ilustración 44* e *Ilustración 46*) se observa que el número de victorias (o niveles completados) es mucho mayor en la primera gráfica que en la segunda, e incluso en la primera grafica se advierte que en algunos casos se supera las 9.000 victorias (más del 90% de las evaluaciones disponibles). Cuando se tiene el tamaño de población de 50 individuos (*Ilustración 45* e *Ilustración 47*) se percibe que tiene el mismo comportamiento que anteriormente, es decir, que en la inicialización personalizada se tiene bastantes más victorias que en la inicialización aleatoria. En la *Tabla 5*, se puede comprobar dicho comportamiento.

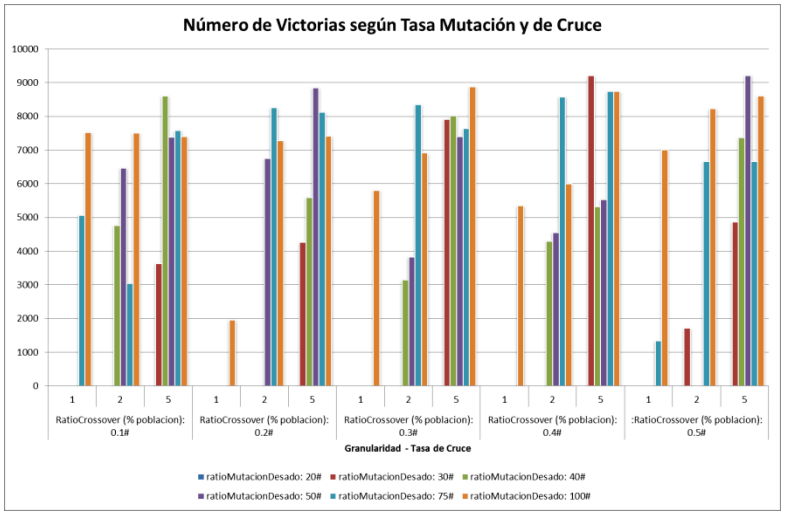


Ilustración 44: Población: 20, Inicialización Guiada

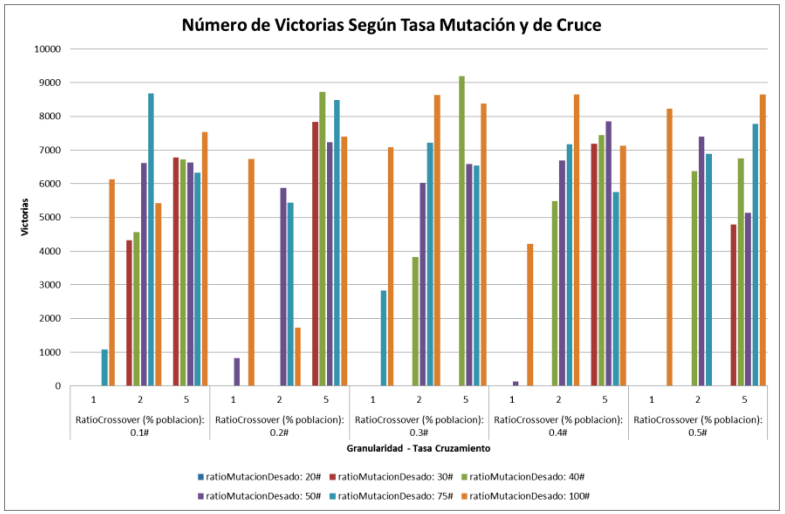


Ilustración 45: Población: 50, Inicialización Guiada

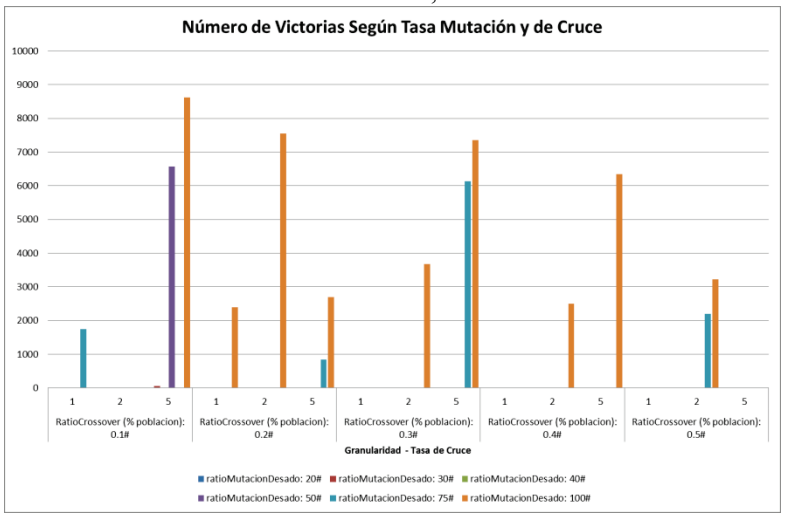


Ilustración 46: Población: 20, Inicialización Aleatoria

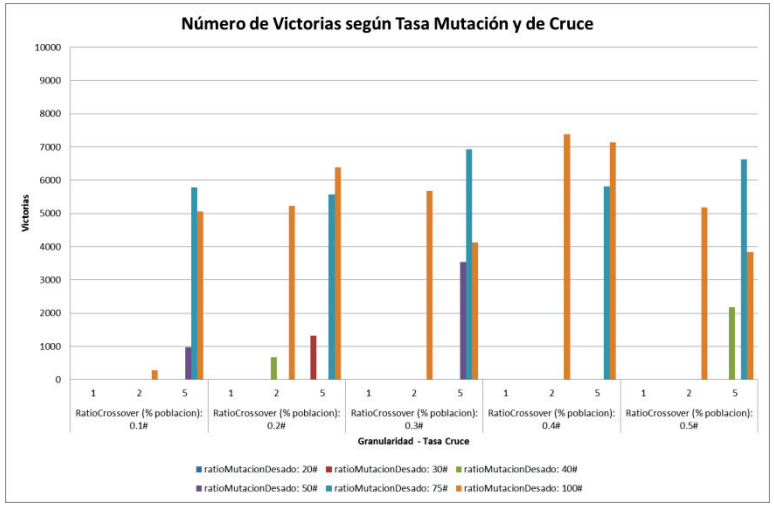


Ilustración 47: Población: 50, Inicialización Aleatoria

		Tamaño de Población	
		20	**50**
Inicialización	**Guiada**	323.677	327.292
	Aleatoria	61.921	89.842

Tabla 5: Número de veces que se ha completado el nivel, según la configuración del AG

Por lo tanto, se puede afirmar que el valor del parámetro Tipo de Inicialización, que mejor rendimiento proporciona en esta primera fase de experimentación, es la Inicialización Guiada.

- **Parámetro de configuración "Granularidad"**

Como se comentó en apartados anteriores, decidir el número de veces que el agente devuelve la misma acción es de relativa importancia. En principio aumentar la granularidad reduce el número de acciones que son ejecutadas, reduciendo así el espacio de búsqueda. Sin embargo, también supone una limitación, ya que presupone que el nivel puede ser completado utilizando X acciones que se repetirán N veces, siendo N la granularidad de las acciones. Por ejemplo con una secuencia de 10 acciones $\{a_0, a_1, a_2, .., a_{10}\}$ y una granularidad igual a dos, las acciones ejecutadas por el controlador serían $\{a_0, a_0, a_1, a_1, a_2, a_2, .., a_{10}, a_{10}\}$.

Las gráficas *Ilustración 48*, *Ilustración 49*, e *Ilustración 50*, representan la evolución de la aptitud de los individuos con una Inicialización Guiada, un tamaño de población de 20 individuos y con un valor de granularidad de 1, 2, y 5 respectivamente. En principio no se observan grandes diferencias entre la *Ilustración 48* y la *Ilustración 49*, que sin embargo se pueden apreciar como independientemente de la tasa de mutación, con una mayor granularidad se obtienen mejores resultados, además con granularidad 1 se tardan más generaciones en completar el nivel (alrededor de los 8000 puntos) con respecto a las otras granularidades.

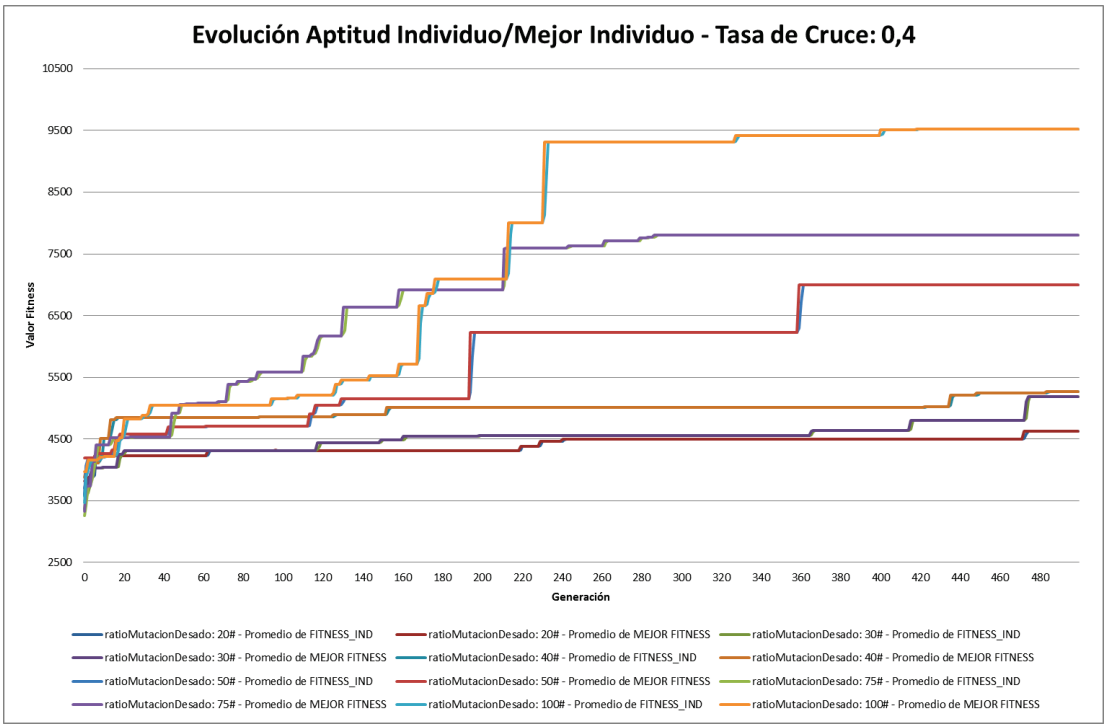


Ilustración 48: Evolución 20 individuos, Granularidad 1, Inicialización Guiada

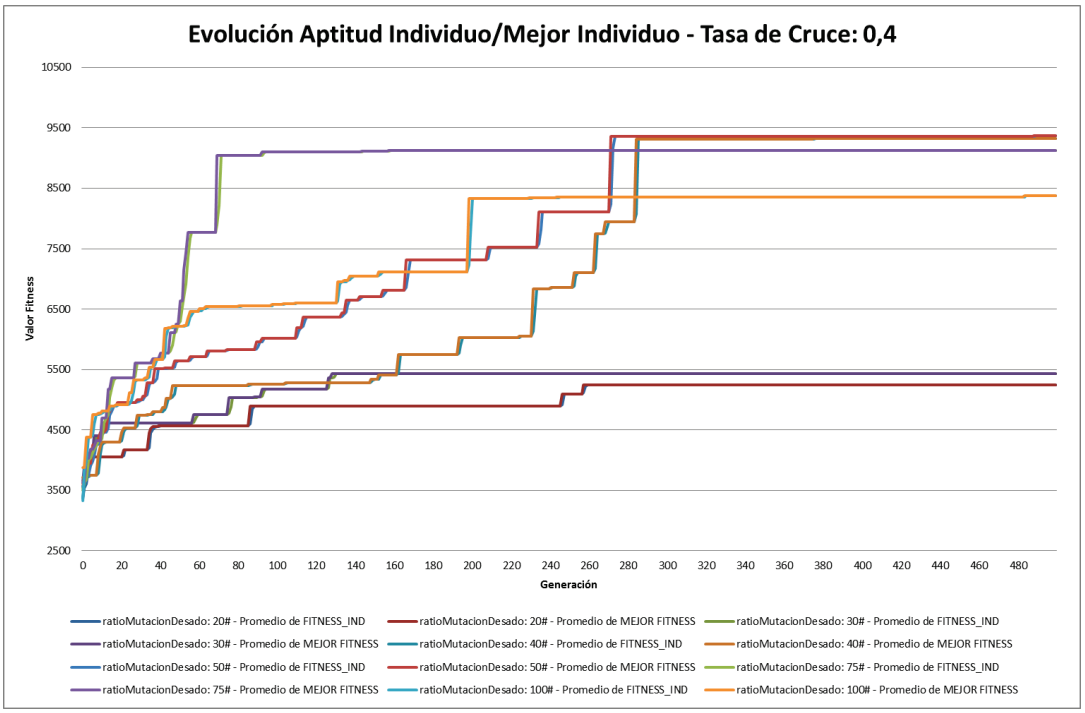


Ilustración 49: Evolución 20 individuos, Granularidad 2, Inicialización Guiada

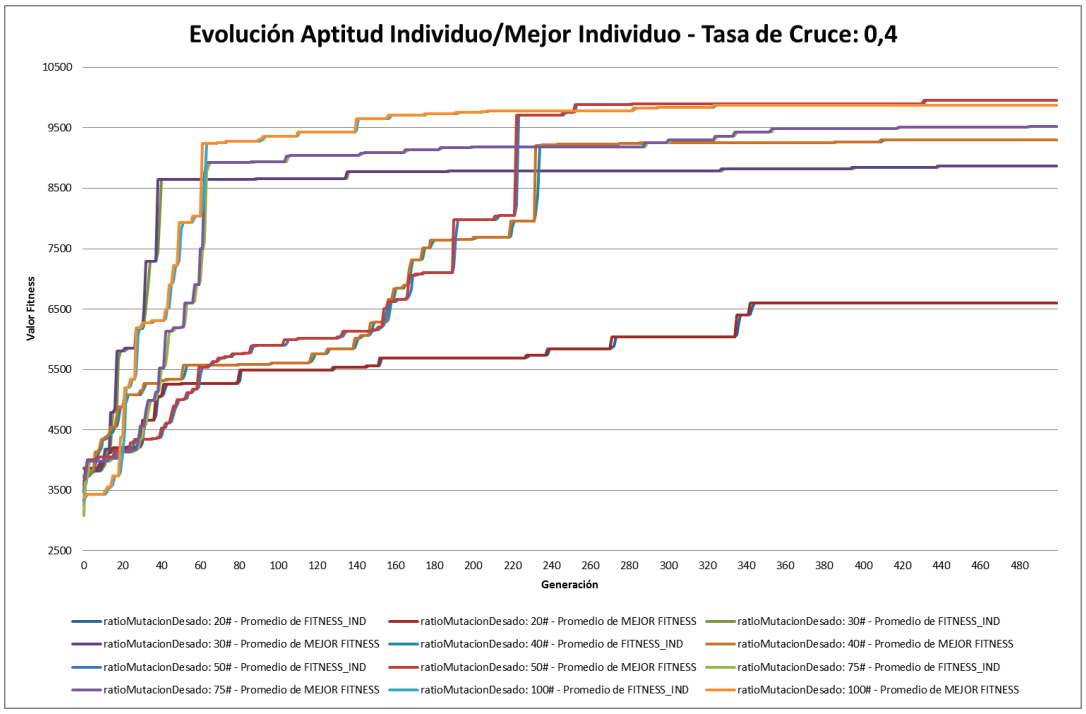


Ilustración 50: Evolución 20 individuos, Granularidad 5, Inicialización Guiada

Este resultado se pude comprobar fácilmente en la siguiente *Ilustración 50*, en la que se muestra el proceso evolutivo con granularidad 5. Como se puede apreciar en dicha ilustración, el proceso evolutivo mejora notablemente al aumentar la granularidad. Comparándolo con los resultados mostrados en la *Ilustración 48*, se aprecia cómo incluso los peores resultados de la *Ilustración 50* obtienen mejores resultados que la mayoría de los de la *Ilustración 48*.

Además de estas tres últimas gráficas, en las gráficas desde la *Ilustración 36* a la *Ilustración 47*, se aprecia que los mejores resultados se obtienen sistemáticamente con granularidad 5.

Por último para ratificar el análisis realizado sobre las gráficas anteriores, se indica en una tabla resumen, *Tabla 6*, el promedio de la aptitud de todos los individuos, clasificado por el tamaño de la población y el tipo de inicialización.

				Tamaño de Población	
				20	50
Inicialización	Guiada	Granularidad	1	5.714,28	5.849,82
			2	6.926,39	6.975,22
			5	8.100,01	7.805,62
	Aleatoria	Granularidad	1	4.917,90	4.770,22
			2	5.391,58	5.438,07
			5	6.230,61	6.658,62

Tabla 6: Promedio de la Aptitud de los individuos, según la granularidad

Por lo tanto, el valor de la granularidad que mejores resultados proporciona es el de granularidad 5.

- **Parámetro de configuración "Tamaño Población"**

El número de individuos de la población puede determinar la variabilidad genética dentro de la población. En principio se podría pensar que a mayor número de individuos mejor, sin embargo a mayor número de individuos, menor es el número de generaciones, ya que el número de evaluaciones permanece constante. Con el tipo de inicialización aleatoria parece claro que es mejor 50 individuos a 20, como podemos observar en las gráficas de la *Ilustración 46* y la *Ilustración 47*, ya que hay una diferencia, a favor de la población de 50 individuos, de 27.921 evaluaciones en las que se ha superado el nivel de Mario AI probado (se puede comprobar el cálculo con la ayuda de la *Tabla 5*).

Con un tipo de inicialización no aleatoria el resultado no es tan claro como el anterior, sin embargo una población con menos individuos parece obtener mejores resultados, como se puede apreciar en las gráficas de la *Ilustración 44* y la *Ilustración 45*. Mientras que en el cómputo global de victorias hay una ligera diferencia, de 3.615 victorias (el 1% del total), a favor del tamaño de la población de 50 individuos.

En la tabla siguiente, *Tabla 7*, se muestra el promedio de las aptitudes máximas de toda la experimentación, y se observa que con un tamaño de población de 20 individuos se han obtenido individuos ligeramente mejores (con mayor valor de aptitud) que con una tamaño de población de 50 individuos.

		Tamaño de Población	
		20	**50**
Inicialización	**Personalizada**	9.545,13	9.380,40
	Aleatoria	8.773,07	8.551,13

Tabla 7: Promedio de Máxima Aptitud, según la configuración del AG

Por lo tanto, se puede deducir que para esta primera fase de experimentación el tamaño de la población no afecta mucho al AG, esto puede ser debido, a que el nivel de Mario AI es de dificultad media, con lo cual, ambos han tenido buenos resultados con este nivel.

- **Parámetros de configuración "Operadores Genéticos Cruce y Mutación"**

En el primer grupo de gráficas expuestas, de la *Ilustración 36* a la *Ilustración 47*, se aprecia que los parámetros de configuración de los operadores genéricos no tienen un comportamiento esperado. Ya que para una misma configuración de operadores genéticos, se puede tener buenos o malos resultados dependiendo de la configuración del resto de parámetros. Aunque en términos generales podemos agrupar algunas configuraciones en las que se obtienen buenos resultados que podemos resumir en los siguientes puntos.

Para el operador de mutación se tiene que las mejores configuraciones son cuando la tasa de mutación es alta, 50, 75, 100, es decir, las probabilidades de mutar un gen de $\frac{1}{50}$, $\frac{1}{75}$, y $\frac{1}{100}$. Esto es debido, a que a probabilidades de mutaciones altas, se pueden mutar genes que no se deberían, empeorando el comportamiento del individuo y de ocasionar que la búsqueda se convierta en aleatoria.

En resumen, el operador de mutación ha demostrado ser más efectivo cuanto menor sea el número de genes a mutar, ya que dada la naturaleza del problema, una mutación en el gen/acción *i* afectará a la validez de todos los genes subsiguientes, es decir:

1. Si se tiene el individuo A, que codifica la secuencia de acciones $\{a_0, a_1, a_2, \dots, a_9\}$.
2. Si se muta la acción a_1, dicha mutación afectará a las condiciones donde se ejecutan las acciones $\{a_2, a_3, \dots, a_9\}$, pudiendo incluso provocar que dichas acciones no lleguen a ejecutarse.

Este operador genérico, por lo tanto, limita la exploración de soluciones óptimas, debido a que no guía correctamente a los individuos a una posible solución óptima, es decir, a superar el nivel de Mario AI.

Para el operador de cruce, es todavía, menos predecible, aunque viendo las gráficas, se puede llegar a la conclusión de que los mejores resultados se obtienen con las tasas de

cruce de 0.3, 0.4 y 0.5. Aunque este operador genérico no es efectivo para este dominio, debido a su naturaleza, es lógico pensar que la mejora de resultados se deba a que el operador proporciona más individuos adicionales a la hora ser elegidos en el torneo.

En resumen, el operador de cruce es ineficiente ya que se utiliza un tipo de cruce posicional, sin embargo en este dominio no es aconsejable utilizarlo ya que,

1. Tenemos el individuo A, que codifica las acciones $\{a_0, a_1, a_2, \dots, a_9\}$.
2. Tenemos al individuo B, que codifica las acciones $\{b_0, b_1, b_2, \dots, b_9\}$.
3. Al seleccionar un punto de cruce cualquiera, $\{b_0, b_1 \uparrow b_2, \dots, b_9\}$ para generar los individuos C y D (donde $\uparrow$ indica el punto de cruce) obtendríamos:
   a. $\{a_0, a_1, b_2, \dots, b_9\} \Rightarrow$ C
   b. $\{b_0, b_1, a_2, \dots, a_9\} \Rightarrow$ D
4. En general las acciones $\{a_1, b_2\}$ y $\{b_1, a_2\}$ procedentes de los individuos A y B no tienen porqué coincidir, en lo que respecta a la posición de Mario en la pantalla y de hecho en la mayor parte de los casos no lo harán, en cambio las acciones $\{a_1, a_2\}$ y $\{b_1, b_2\}$ procedentes de los individuos A y B sí que lo hacían.

Así pues, con este tipo de cruce se puede estar creando nuevos individuos con acciones consecutivas que se aplicaron al principio y al final de la pantalla, lo cual es totalmente ineficiente. Por lo tanto, este operador genérico, limita la explotación de nuevas soluciones a partir de la actual, ya que los nuevos individuos que genera este operador, han sido de forma aleatoria sin dotar ninguna información del dominio del juego.

- **Parámetro de configuración "Operador Genético Selección"**

Para este parámetro sólo hay un valor posible, la Selección por Torneo, esto se debe, a que en la mayoría de los trabajos de investigación sobre AG, una de las conclusiones obtenidas es que uno de los operadores genéticos de selección que mejores resultados se obtienen es el del torneo, como se indica en [47]. Para simplificar la experimentación se ha decidido establecer un tamaño fijo de torneo pequeño, en este caso del 15% de la población, ya que con este valor se tiene una presión selectiva adecuada al problema en cuestión.

El único problema que se ha encontrado en la utilización de este parámetro, es que en ocasiones, el mejor individuo de la población de una generación no se transmitía a la siguiente generación, esto era debido a que el operador de selección elegido no tenía elitismo, es decir, no incluía siempre el mejor individuo de una generación en la siguiente generación.

La primera evidencia que se puede obtener en esta primera experimentación, es que las técnicas basadas en AG son susceptibles de aplicación a la categoría de Aprendizaje (*Learning Track*) de la competición "Mario AI Competition", al menos en los términos

actuales de dicha competición [48]. Sobre los operadores de cruce y mutación, cabe destacar que los implementados hasta la fecha son ineficientes, especialmente el operador de cruce.

Pese a la implementación realizada y a las posibles mejoras de ésta, los resultados son aceptables. El hecho de utilizar una aproximación generalista, basada en AG, y a tenor de los resultados obtenidos, se pone de manifiesto que la utilización de AG en este dominio, posee un potencial de mejora amplio, el cual será explorado en mayor profundidad en la siguiente sección con la segunda fase de experimentación.

A continuación se iniciará la segunda parte del presente capítulo de experimentación, en la cual se incluyen nuevos operadores genéticos modificados, que se les ha dotado de cierta información del domino de Mario AI, para que sean más efectivos en la búsqueda de soluciones óptimas. Además para verificar que estos nuevos operadores son efectivos se ha decidido incluir, en esta segunda fase de experimentación, nuevos niveles de Mario AI, con una mayor dificultad y que el AG Simple no ha conseguido completar.

4.5 Mejora del AG

En este apartado se mencionarán las mejoras introducidas en el AG, para solucionar las limitaciones vistas en el apartado anterior. Los cambios se centrarán en los parámetros del AG, que permitirán adaptarse al problema en cuestión, en este caso al juego Mario AI.

a) **Tipo de Inicialización**: en esta nueva fase de experimentación se añade un nuevo tipo de inicialización, la inicialización mixta, como se detalla en el apartado 3.6, esta inicialización mezcla el funcionamiento de la inicialización aleatoria y de la inicialización guiada.

b) **Granularidad**: este parámetro se mantiene sin modificar en esta segunda fase, únicamente se elimina uno de los valores que podría tomar el parámetro, este valor es 1, que es el valor con peores resultados que tuvo en la primera fase de experimentación.

c) **Tamaño de la Población**: este parámetro se mantiene sin modificar en esta segunda fase.

d) **Número Máximo de Evaluaciones**: este parámetro se mantiene sin modificar en esta segunda fase.

e) **Operador Genético de Selección**: en esta nueva fase de experimentación se tiene un nuevo tipo de operador genético de selección, de igual forma que en la fase anterior, el operador de selección es el de Torneo, este operador está basado en el operador de Torneo proporcionado por el framework *JGAP*, añadiéndosele elitismo al operador, es decir, que el mejor individuo de la población en una generación pase directamente a la nueva población en la siguiente generación, en el apartado 3.7 se detalla más a fondo este operador.

f) **Operador Genético de Cruce**: también se ha modificado este operador, para esta segunda fase de experimentación, ya que el operador genérico cruce proporcionado por el framework *JGAP*, no era eficiente para el dominio de Mario AI, se ha decidido realizar un nuevo operador de Cruce, que tenga en cuenta las posiciones en las que se encuentra el personaje Mario al realizar las acciones; en el apartado 3.7 se detalla más a fondo este operador.

g) **Operador Genético de Mutación**: de igual forma que en el resto de los operadores genéticos, el operador de Mutación se ha modificado en esta fase de experimentación, ya que el proporcionado por el framework *JGAP*, no era del todo eficaz para el dominio de Mario AI. En esta nueva versión del operador, la mutación de los genes el cromosoma del individuo se realiza en las últimas acciones ejecutadas por el individuo, de la misma manera, en el apartado 3.7 se detalla más a fondo este operador.

h) **Constante Ventana de Mejora**: este es nuevo parámetro introducido al AG en la segunda fase de experimentación, este parámetro está relacionado con el Operador Genético de Mutación, ya que indica cuándo se tiene que modificar la ventana de mutación del nuevo operador de Mutación, en el apartado 3.7 se detalla más a fondo este parámetro.

i) **Tamaño Acciones Almacenadas**: este es nuevo parámetro introducido al AG, en la segunda fase de experimentación, que está relacionado con el nuevo Operador Genético de Cruce, ya que dicho operador necesita almacenar las posiciones en las que se encuentra Mario al ejecutar las acciones, este parámetro indica cuántos pares acción-posición de Mario AI almacena en el sistema, almacenando siempre los N últimos pares acción-posición. Este parámetro se ha creado para controlar la eficiencia del AG con respecto al consumo de memoria del sistema.

j) **Opciones del nivel de Mario AI**: se han añadido nuevos parámetros de configuración del nivel del juego Mario AI:
   - "**-lde on/off**": Con esta opción se tiene control sobre los callejones sin salida (dead ends) en un nivel.
   - "**-le on/off**": Esta opción se utiliza para activar/desactivar los enemigos del nivel.
   - "**-lla on/off**": Esta opción se utiliza para activar/desactivar las escaleras en el nivel (level ladder).

   Y los ya utilizados:
   - "**-vis off**": Visualización del juego, permite seguir visualmente las partidas o jugadas que se ejecutan en la evaluación del AG. Los valores que puede tomar son "*on*" para activar la visualización y "*off*" para desactivar la visualización.
   - "**-lhb on**": Bloques ocultos, esta opción se utiliza para activar/desactivar los bloques ocultos, por lo tanto, en la experimentación se encuentra activo.
   - "**-lt 0**": Tipo de Nivel, *Overground* (0, en la superficie), *Underground* (1, subterráneo), *Castle* (2, castillo) y *Random* (3, aleatorio). Está activo el primer tipo de nivel.
   - "**-ld 1**": Dificultad del nivel, un entero entre 0 y 12, en este caso es de 1.

- "**-ll 300**": Longitud del nivel, un entero entre 50 y $2^{31} - 1$, en este caso es de 300.
- "**-ls 20002**": Semilla de aleatorización del nivel, un entero entre 0 y $2^{31} - 1$, en este caso es de 20002.

4.6 Batería de Experimentos del AG Mejorado

De igual forma que en la experimentación anterior, a continuación se especificarán los parámetros de configuración del AG Mejorado que se desea probar. Los parámetros utilizados ya fueron mencionados en el apartado anterior, y por lo tanto, únicamente se especificarán los valores que tomarán estos parámetros en la nueva experimentación.

I. **Tipo de Inicialización**, tres valores posibles, seleccionados de forma aleatoria para cada individuo:
   a. *Inicialización Guiada*
   b. *Inicialización Mixta*
   c. *Inicialización Aleatoria*

II. **Granularidad**, tres valores posibles:
   a. *2 acciones*
   b. *5 acciones*

III. **Tamaño de la Población**, dos valores posibles:
   a. *20 individuos*
   b. *50 individuos*

IV. **Opciones de nivel de Mario AI**, varios valores posibles:
   a. *-vis off -lhb on -lt 0 -ll 300 -lde on -ls 01121987 -ld 4*
   b. *-vis off -lhb on -lt 0 -ll 300 -lde on -ls 201183 -ld 7*
   c. *-vis off -lhb on -lt 0 -ll 300 -lde on -le off -ls 201183 -ld 7*
   d. *-vis off -lhb on -lla on -lde on -ls 333 -ld 7*
   e. *-vis off -lhb on -lla on -lde on -le off -ls 444 -ld 7*
   f. *-vis off -lhb on -lt 0 -ll 300 -lla on -lde on -ls 201183 -ld 4*
   g. *-vis off -lhb on -lla on -lde on -le off -ls 334 -ld 4*
   h. *-vis off -lhb on -lla on -lde on -ls 333 -ld 4*
   i. *-vis off -lhb on -lt 0 -ll 300 -ls 11062011 -ld 4*
   j. *-vis off -lhb on -lla on -lde on -ls 444 -ld 4*

V. **Número Máximo Evaluaciones**, un valor posible:
   a. *10.000 evaluaciones*

VI. **Operador Genético de Selección**, un valor posible:
   a. *Selección por Torneo Mejorado para Mario AI*, con los parámetros:
      i. Tamaño del Torneo: 0.15 (15% de la población)
      ii. Probabilidad de Seleccionar el Mejor Individuo: 1.0 (100%)

iii. Ejecutar después de los Operadores Genéticos de Cruce y Mutación

VII. Operador Genético de Mutación, un valor posible:
- *a. Ope. Gen. de Mutación Mejorado para Mario AI*, con los parámetros:
  - i. Tasa de Mutación Deseada: 100 (1 gen mutado de cada 100 genes de forma aleatoria, 1% del cromosoma).

VIII. Operador Genético de Cruce, un valor posible**:**
- *a. Ope. Gen. de Cruce mejorado para Mario AI*, con los parámetros:
  - i. Tasa de Cruce (% población): 0.1 (10 % de la población)
  - ii. Tasa de Cruce (% población): 0.2 (20 % de la población)
  - iii. Tasa de Cruce (% población): 0.3 (30 % de la población)
  - iv. Tasa de Cruce (% población): 0.4 (40 % de la población)
  - v. Tasa de Cruce (% población): 0.5 (50 % de la población)

IX. Tamaño Acciones Almacenadas, valor que indica el número de últimas acciones en las que se almacena las posiciones que ocupa Mario a realizar esas acciones, el valor elegido es:
- a. *100 acciones*

X. Constante Ventana de Mejora, valor que indica el número máximo de veces que un individuo puede no mejorar hasta que se incrementa la ventana de mutación (el número de genes a mutar), los posibles valores que pueden tomar son:
- *a. 2 generaciones*
- *b. 3 generaciones*
- *c. 5 generaciones*

En este caso, se tienen los nuevos operadores genéticos desarrollados específicamente para el dominio de Mario AI, partiendo de los operadores base del framework *JGAP*, y con diferentes configuraciones para el operador de cruce, mientras que el operador de mutación mantiene únicamente una configuración posible, para simplificar el número de pruebas, eligiendo la tasa de mutación que mejores resultados se obtuvo en la primera fase de experimentación.

A continuación, se muestran varias tablas resumen, *Tabla 8*, *Tabla 9*, *Tabla 10*, y *Tabla 11*, con todas las pruebas realizadas agrupadas en colores, que identifican a una hoja de cálculo, que contiene las estadísticas de la experimentación. Como se observa, en total se ha realizado 60*10 experimentos, cada uno de ellos con un total de 10.000 partidas o evaluaciones, por lo que en total se ha ejecutado unas 6.000.000 partidas a Mario AI de forma automática.

Por último, destacar que en esta segunda experimentación se ha decidido que Mario tenga 22 movimientos posibles, ampliando el espacio de búsqueda con respecto a la experimentación anterior, esto es debido, a que en los nuevos niveles probados se requieren más movimientos para completar el nivel, por ejemplo subir escaleras, por lo tanto, estos nuevos experimentos no pueden ser comparados con la versión anterior del AG, ya que no podrían completar dichos niveles que necesiten ejecutar estos nuevos

movimientos o acciones. Las acciones o movimientos utilizados son los mostrados en la *Tabla 2*, del apartado 3.6.

				Opciones Configuracion Mario AI "-vis off -lhb on -lt 0 -ll 300 -lde on -ls 01121987 -ld 4"		Opciones Configuracion Mario AI "-vis off -lhb on -lt 0 -ll 300 -lde on -ls 201183 -ld 7"		Opciones Configuracion Mario AI "-vis off -lhb on -lt 0 -ll 300 -lde on -le off -ls 201183 -ld 7"	
				Tamaño de la Población		Tamaño de la Población		Tamaño de la Población	
				20	50	20	50	20	50
				Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100
CONSTANTE VENTANA: 2	GRANURALIDAD: 2	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
	GRANURALIDAD: 5	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
CONSTANTE VENTANA: 3	GRANURALIDAD: 2	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
	GRANURALIDAD: 5	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
CONSTANTE VENTANA: 5	GRANURALIDAD: 2	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
	GRANURALIDAD: 5	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X

Tabla 8: Tabla resumen con todas las pruebas realizadas en la segunda fase (1 – 4)

				Opciones Configuracion Mario AI "-vis off -lhb on -lla on -lde on -ls 333 -ld 7"		Opciones Configuracion Mario AI "-vis off -lhb on -lla on -lde on -le off -ls 444 -ld 7"		Opciones Configuracion Mario AI "-vis off -lhb on -lt 0 -ll 300 -lla on -lde on -ls 201183 -ld 4"	
				Tamaño de la Población		Tamaño de la Población		Tamaño de la Población	
				20	50	20	50	20	50
				Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100
CONSTANTE VENTANA: 2	GRANURALIDAD: 2	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
	GRANURALIDAD: 5	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
CONSTANTE VENTANA: 3	GRANURALIDAD: 2	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
	GRANURALIDAD: 5	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
CONSTANTE VENTANA: 5	GRANURALIDAD: 2	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X
	GRANURALIDAD: 5	Operador de Cruce	0,1	X	X	X	X	X	X
			0,2	X	X	X	X	X	X
			0,3	X	X	X	X	X	X
			0,4	X	X	X	X	X	X
			0,5	X	X	X	X	X	X

Tabla 9: Tabla resumen con todas las pruebas realizadas en la segunda fase (2 – 4)

			Opciones Configuracion Mario AI "-vis off -lhb on -lla on -lde on -le off -ls 334 -ld 4"		Opciones Configuracion Mario AI "-vis off -lhb on -lla on -lde on -ls 333 -ld 4 "	
			Tamaño de la Población		Tamaño de la Población	
			20	50	20	50
			Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100
GRANURALIDAD: 2	Operador de Cruce	0,1	x	x	x	x
		0,2	x	x	x	x
		0,3	x	x	x	x
		0,4	x	x	x	x
		0,5	x	x	x	x
GRANURALIDAD: 5	Operador de Cruce	0,1	x	x	x	x
		0,2	x	x	x	x
		0,3	x	x	x	x
		0,4	x	x	x	x
		0,5	x	x	x	x
GRANURALIDAD: 2	Operador de Cruce	0,1	x	x	x	x
		0,2	x	x	x	x
		0,3	x	x	x	x
		0,4	x	x	x	x
		0,5	x	x	x	x
GRANURALIDAD: 5	Operador de Cruce	0,1	x	x	x	x
		0,2	x	x	x	x
		0,3	x	x	x	x
		0,4	x	x	x	x
		0,5	x	x	x	x
GRANURALIDAD: 2	Operador de Cruce	0,1	x	x	x	x
		0,2	x	x	x	x
		0,3	x	x	x	x
		0,4	x	x	x	x
		0,5	x	x	x	x
GRANURALIDAD: 5	Operador de Cruce	0,1	x	x	x	x
		0,2	x	x	x	x
		0,3	x	x	x	x
		0,4	x	x	x	x
		0,5	x	x	x	x

Tabla 10: Tabla resumen con todas las pruebas realizadas en la segunda fase (3 – 4)

				Opciones Configuracion Mario AI "-vis off -lhb on -lt 0 -ll 300 -ls 11062011 -ld 4"		Opciones Configuracion Mario AI "-vis off -lhb on -lla on -lde on -ls 444 -ld 4"	
				Tamaño de la Población		Tamaño de la Población	
				20	50	20	50
				Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100	Operador de Mutación: 100
CONSTANTE VENTANA: 2	GRANURALIDAD: 2	Operador de Cruce	0,1	x	x	x	x
			0,2	x	x	x	x
			0,3	x	x	x	x
			0,4	x	x	x	x
			0,5	x	x	x	x
	GRANURALIDAD: 5	Operador de Cruce	0,1	x	x	x	x
			0,2	x	x	x	x
			0,3	x	x	x	x
			0,4	x	x	x	x
			0,5	x	x	x	x
CONSTANTE VENTANA: 3	GRANURALIDAD: 2	Operador de Cruce	0,1	x	x	x	x
			0,2	x	x	x	x
			0,3	x	x	x	x
			0,4	x	x	x	x
			0,5	x	x	x	x
	GRANURALIDAD: 5	Operador de Cruce	0,1	x	x	x	x
			0,2	x	x	x	x
			0,3	x	x	x	x
			0,4	x	x	x	x
			0,5	x	x	x	x
CONSTANTE VENTANA: 5	GRANURALIDAD: 2	Operador de Cruce	0,1	x	x	x	x
			0,2	x	x	x	x
			0,3	x	x	x	x
			0,4	x	x	x	x
			0,5	x	x	x	x
	GRANURALIDAD: 5	Operador de Cruce	0,1	x	x	x	x
			0,2	x	x	x	x
			0,3	x	x	x	x
			0,4	x	x	x	x
			0,5	x	x	x	x

Tabla 11: Tabla resumen con todas las pruebas realizadas en la segunda fase (4 – 4)

4.7 Análisis de Resultados del AG Mejorado

En la segunda fase de experimentación se pretende demostrar que se han solucionado los problemas encontrados durante la primera fase, que en definitiva se traduce en probar nuevos niveles de Mario AI más complejos que el probado en la primera fase. Por lo tanto no se podrá hacer una comparación con los resultados de la primera experimentación porque el Algoritmo diseñado en la primera experimentación de Mario AI no es capaz de completar ninguno de los niveles utilizados en la segunda experimentación.

A continuación se muestra una sucesión de gráficas de evolución de la aptitud de los individuos del AG, según el nivel del juego Mario AI que ha sido probado. Dentro de cada gráfica se muestra la evolución según los valores tomados por los parámetros del AG: granularidad (2 ó 5) y la tasa de cruce (10, 20, 30, 40 ó 50) del Operador de Cruce, por lo tanto dentro de cada gráfica se tiene diez líneas de evolución diferentes. En todas estás gráficas, de la *Ilustración 51* a la *Ilustración 70*, se puede verificar como la evolución de la aptitud es la esperada para un AG, es decir, al inicio de la evolución, en las primeras generaciones, se tienen los valores mínimos de la aptitud, mientras que al final de la evolución, en las últimas generaciones, se tienen los valores máximos de la aptitud, en resumen ha habido un aprendizaje en el AG.

En dichas gráficas, exceptuando las gráficas *Ilustración 59* e *Ilustración 60* que tienen un comportamiento anómalo que serán analizadas más adelante, se comprueba que la evolución es progresiva, hasta un momento de la evolución en el cual se ha superado el nivel, obteniéndose la máxima puntuación (o aptitud, en este caso), o bien se ha llegado al límite de generaciones disponibles, a diferencia de la primera fase, que se tiene una evolución más brusca o a saltos, como se puede apreciar en las gráficas de la *Ilustración 36* a la *Ilustración 39*.

Una última característica que se puede apreciar en dichas gráficas, es que las gráficas que se corresponden con un tamaño de población de 20 individuos, su línea de evolución tiene un comportamiento en forma de sierra o cresta, es decir, aunque la tendencia general de la línea de evolución es en una progresión creciente, en tramos limitados de la generación se tiene progresiones crecientes y decrecientes continuas, de ahí la apariencia de una sierra, mientras que para el caso de una población de 50 individuos, no ocurre esto, teniendo un línea de evolución totalmente progresiva y creciente. Esto es debido, a que hay diferentes individuos en una población con diferentes aptitudes en una generación dada, aunque el mejor individuo siempre se propaga a la siguiente generación, y al hacer el promedio con 20 individuos se producen desviaciones más bruscas que con una población de 50 individuos, que tiene más probabilidad de que se encuentren los mejores individuos, por lo que la probabilidad de generación en generación suele ir aumentando de valor.

Para facilitar la identificación del nivel probado, de Mario AI, se asocia un nivel de Mario AI con un color, que se corresponde con la distribución color-nivel que aparece en la *Tabla 8*, *Tabla 9*, *Tabla 10*, y *Tabla 11*.

Nivel Rojo → Opciones Mario: -vis off –lhb on –lt 0 –ll 300 –lde on –ls 01121987 –ld 4

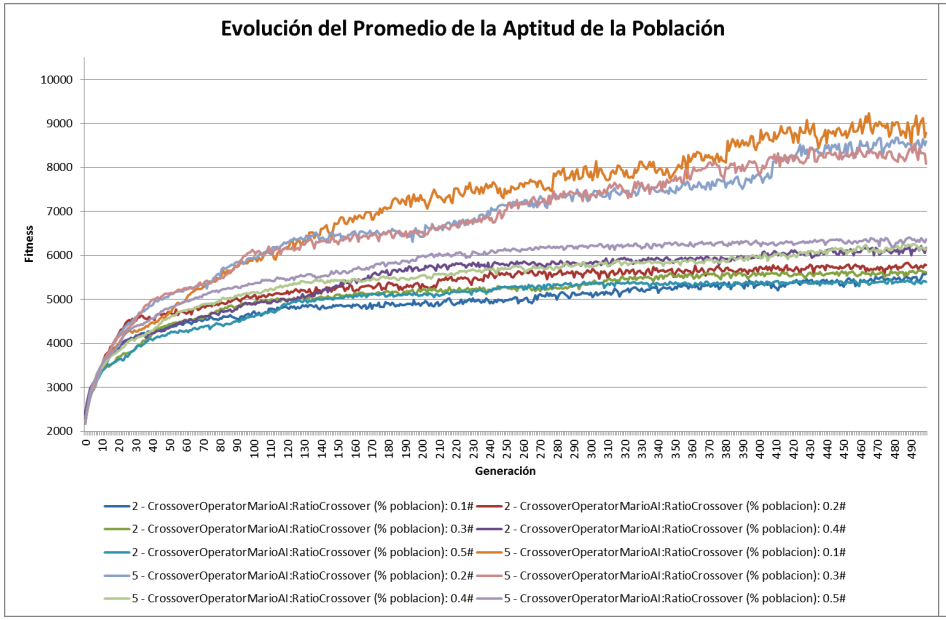


Ilustración 51: Tam. Población 20 individuos

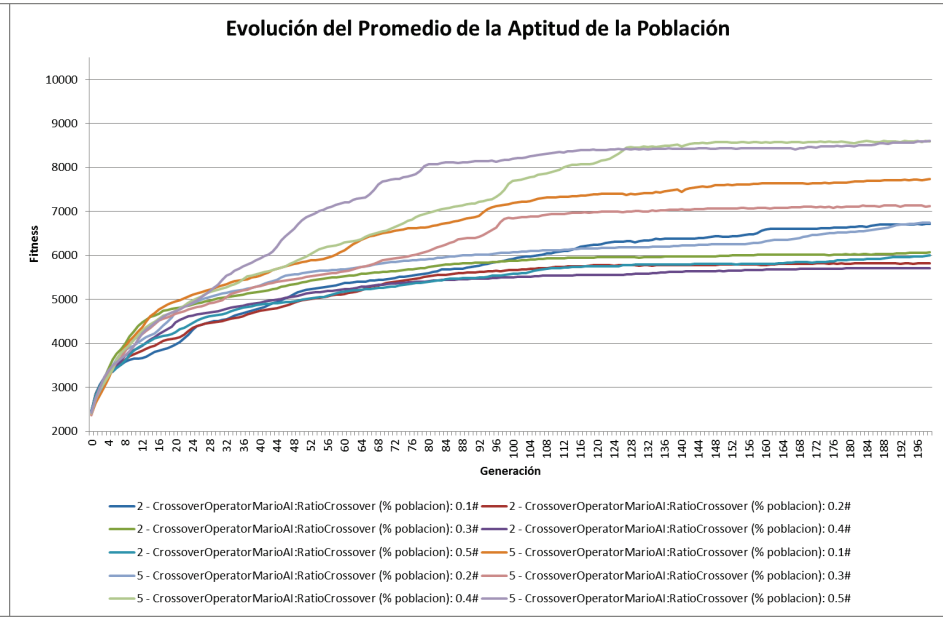


Ilustración 52: Tam. Población 50 individuos

Nivel Amarillo → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -lde on -ls 201183 -ld 7

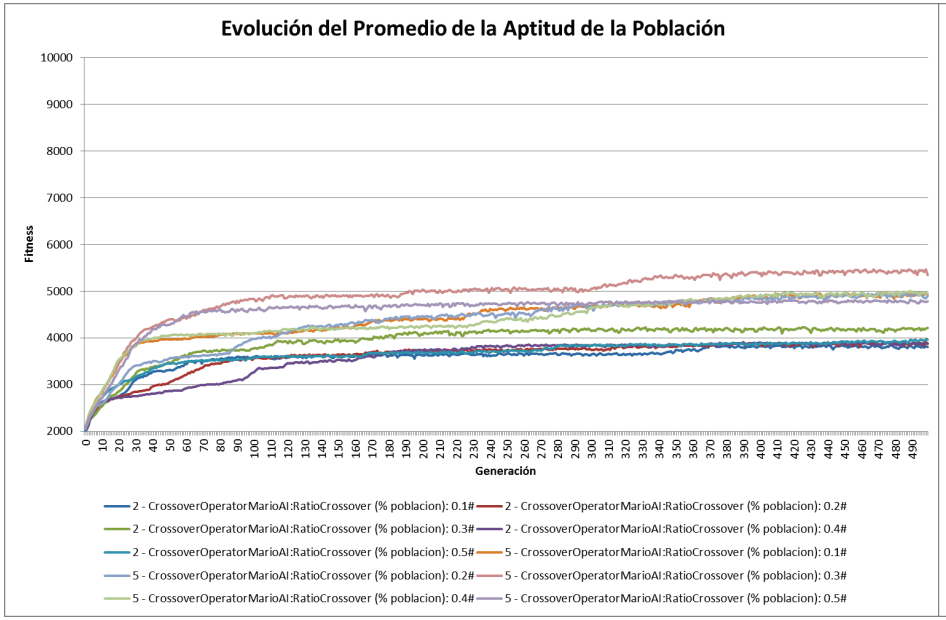


Ilustración 53: Tam. Población 20 individuos

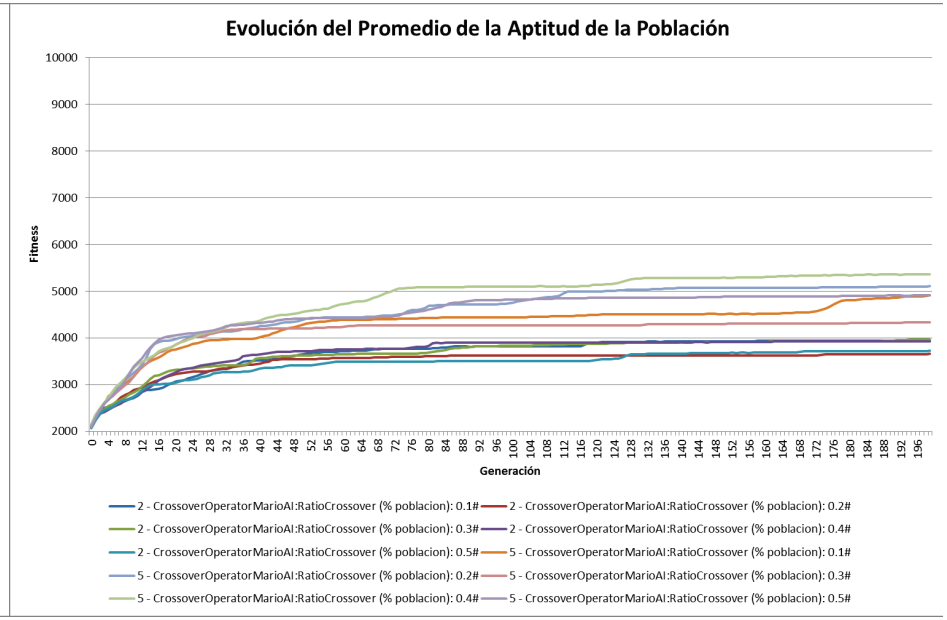


Ilustración 54: Tam. Población 50 individuos

Nivel Verde → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -lde on -le off -ls 201183 -ld 7

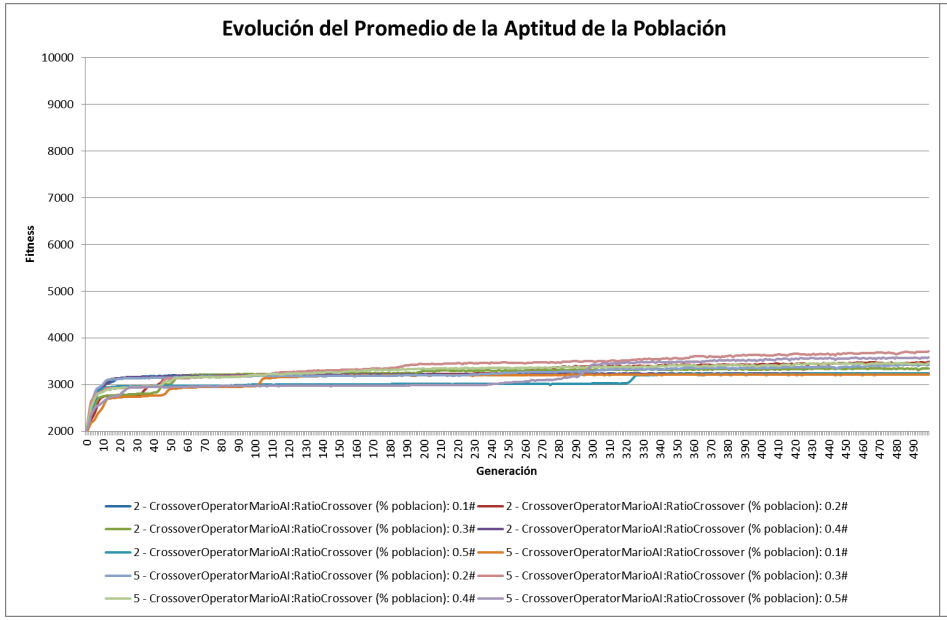


Ilustración 55: Tam. Población 20 individuos

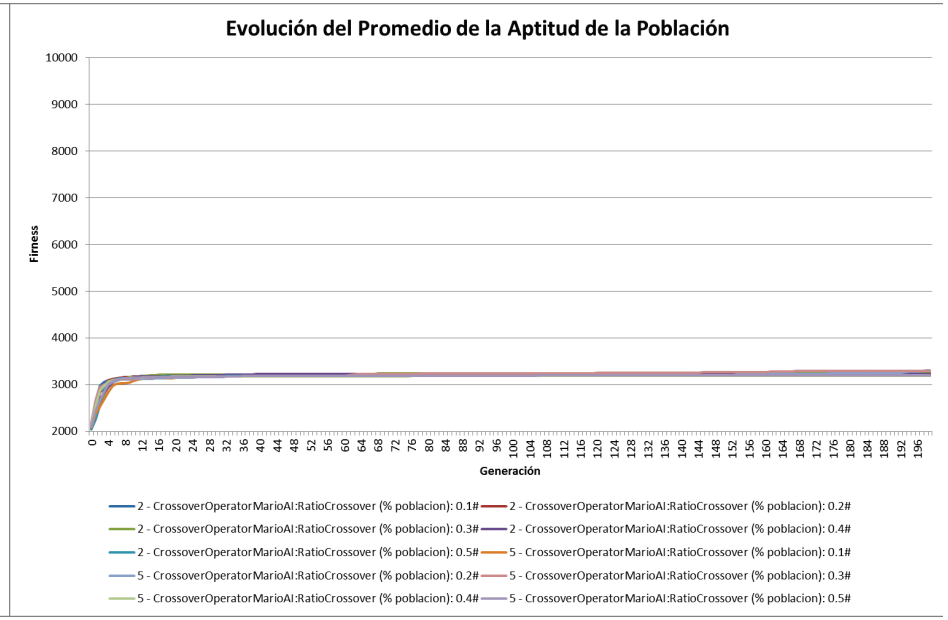


Ilustración 56: Tam. Población 50 individuos

Nivel Naranja → Opciones Mario: -vis off –lhb on –lla on –lde on –ls 333 –ld 7

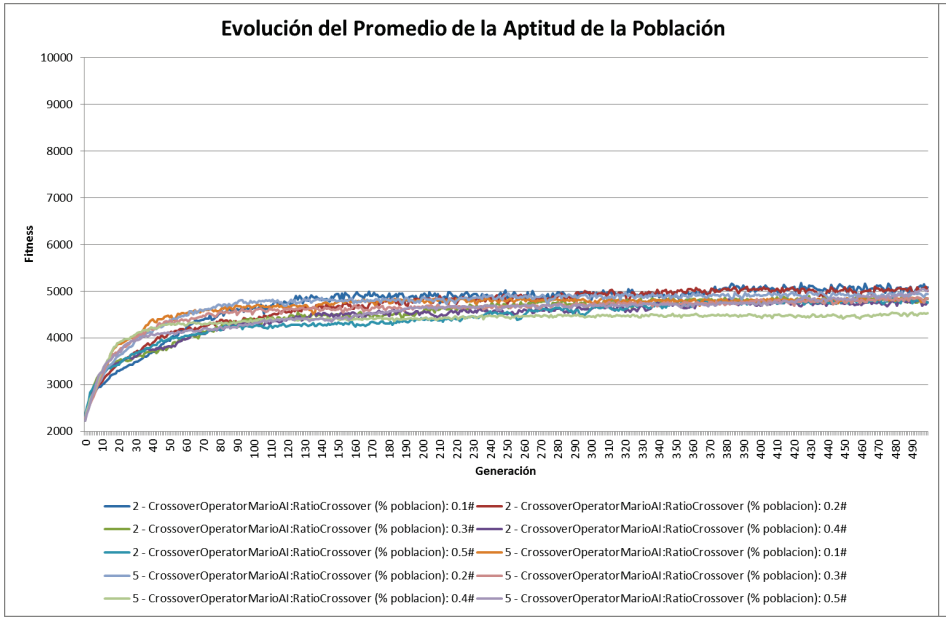


Ilustración 57: Tam. Población 20 individuos

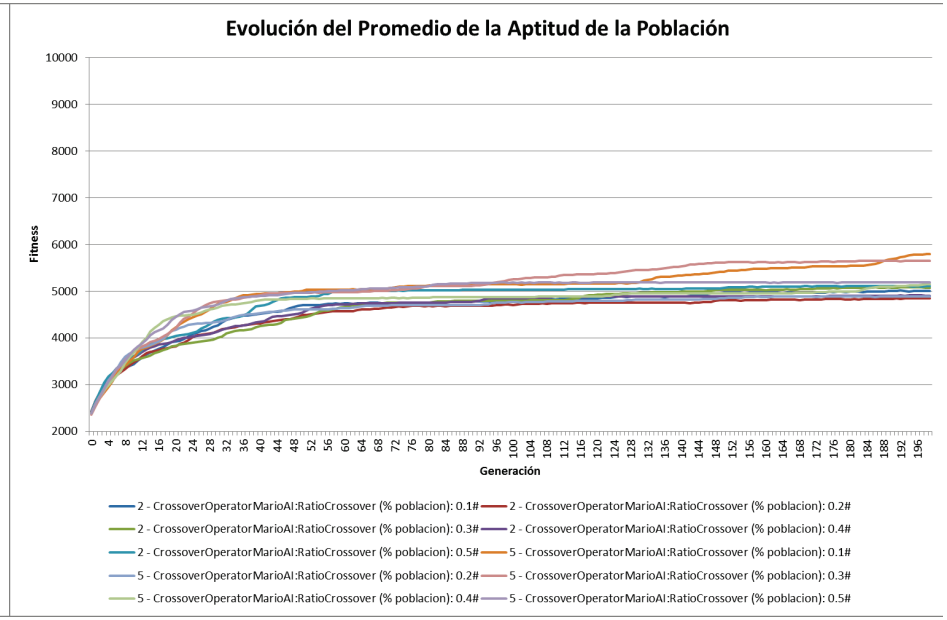


Ilustración 58: Tam. Población 50 individuos

Nivel Purpura → Opciones Mario: -vis off -lhb on -lla on -lde on -le off -ls 444 -ld 7

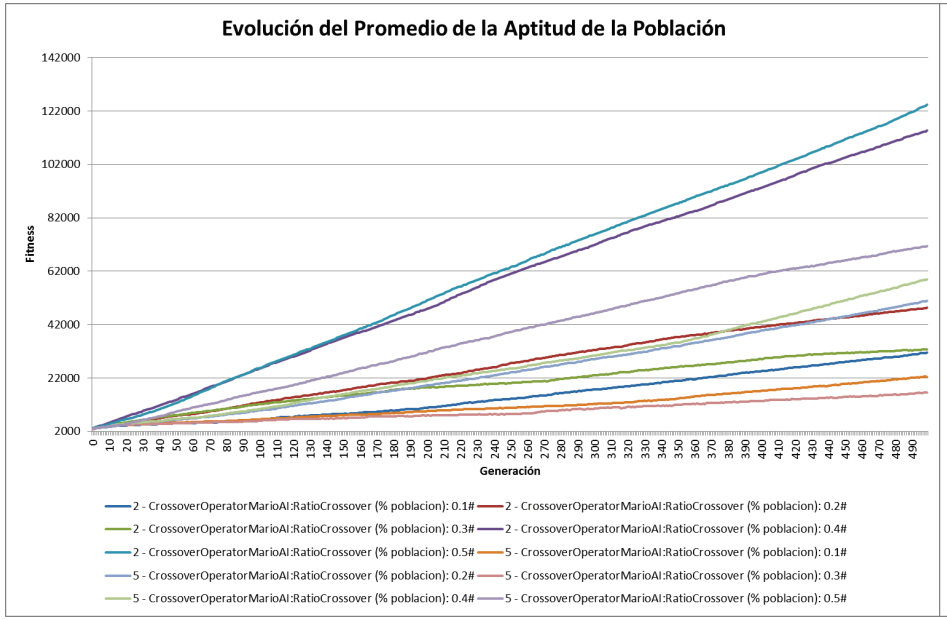


Ilustración 59: Tam. Población 20 individuos

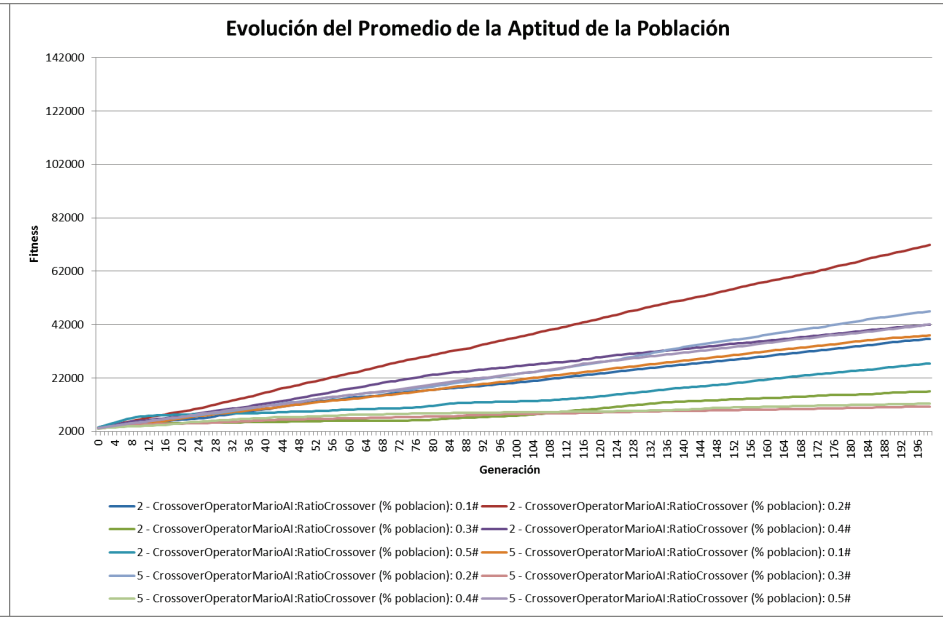


Ilustración 60: Tam. Población 50 individuos

Nivel Azul → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -lde on -ls 201183 -ld 4

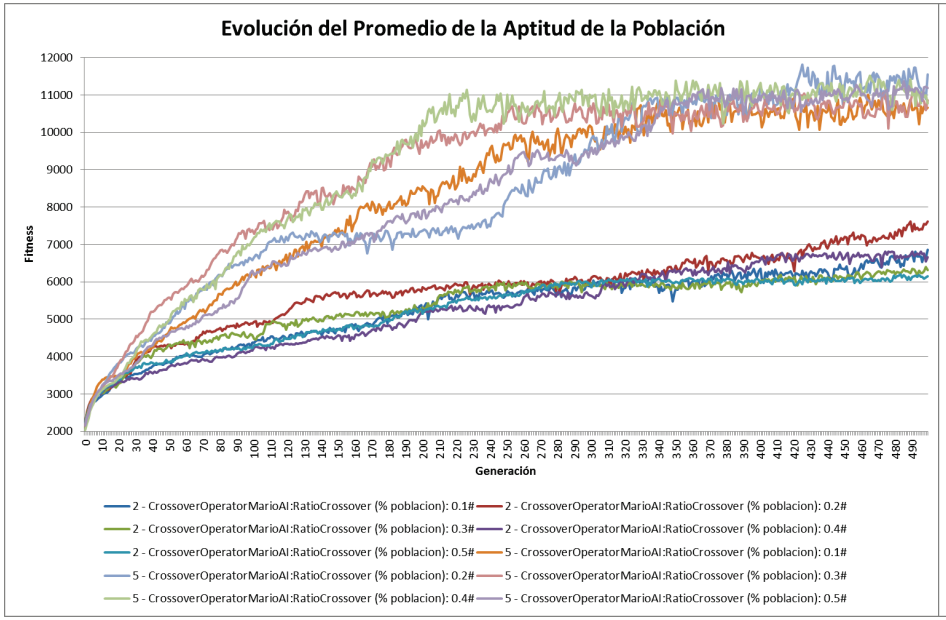


Ilustración 61: Tam. Población 20 individuos

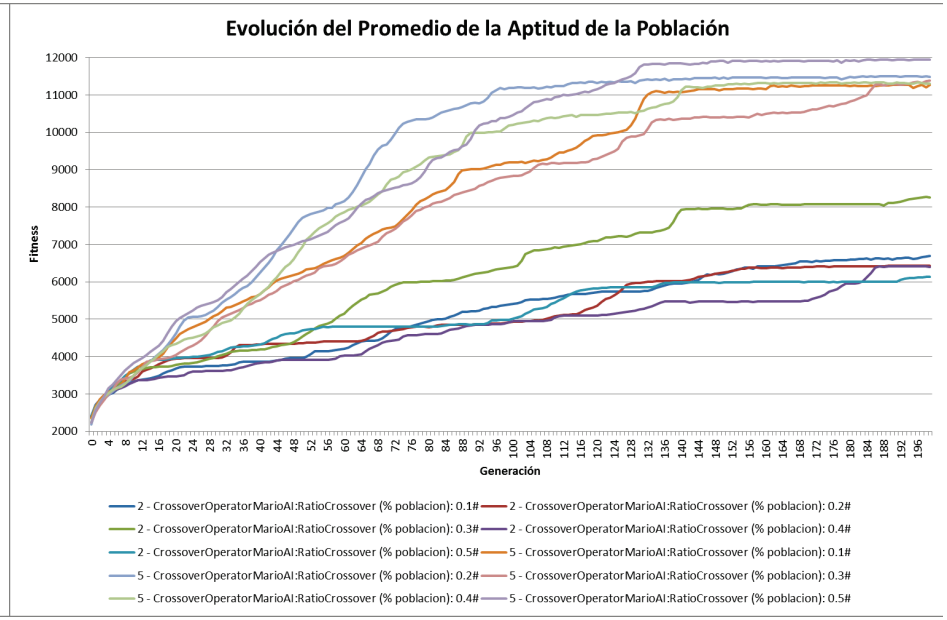


Ilustración 62: Tam. Población 50 individuos

Nivel Marrón → Opciones Mario: -vis off -lhb on -lla on -lde off -ls 334 -ld 4

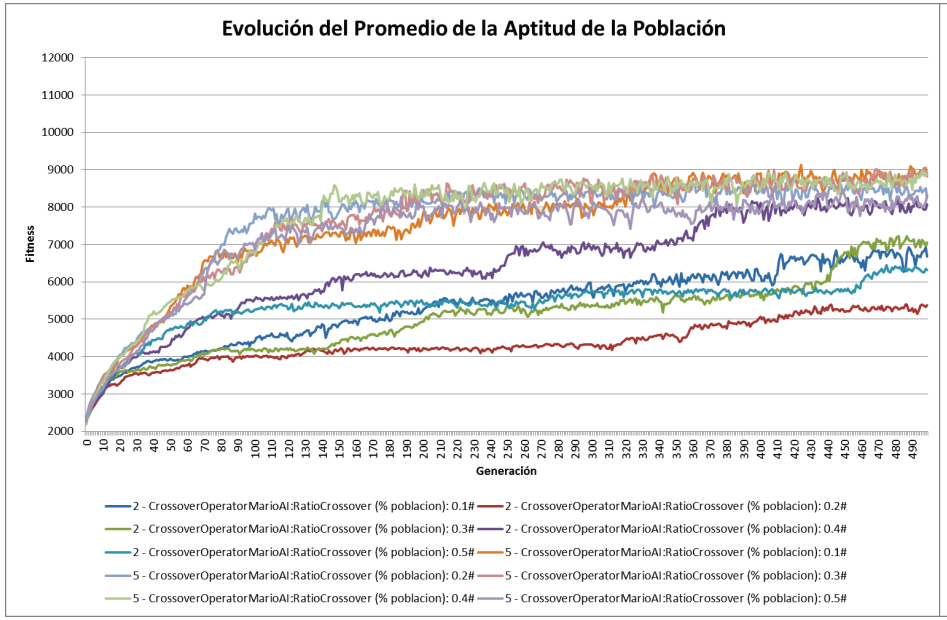


Ilustración 63: Tam. Población 20 individuos

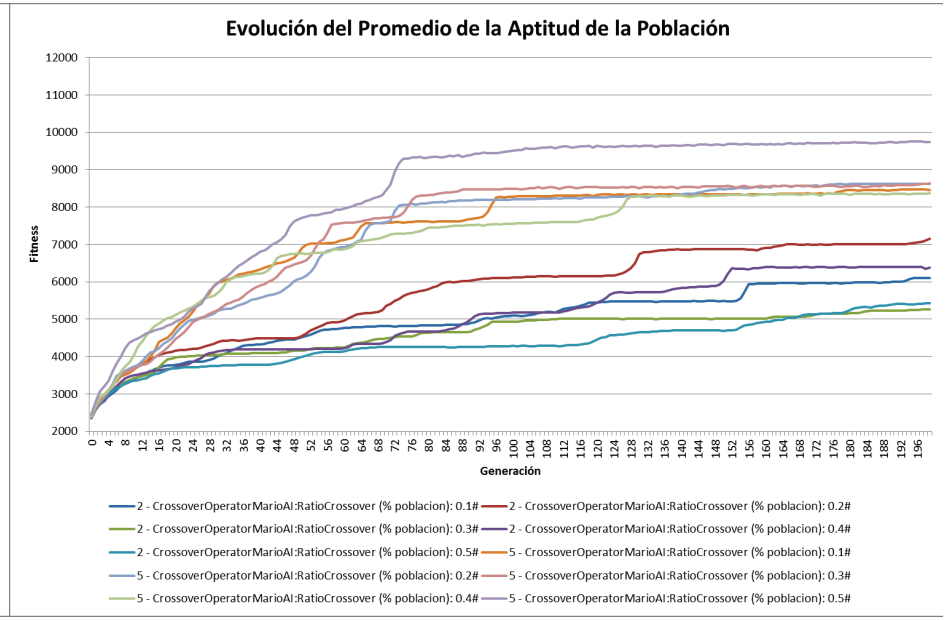


Ilustración 64: Tam. Población 50 individuos

Nivel Salmón → Opciones Mario: -vis off -lhb on -lla on -lde on -ls 333 -ld 4

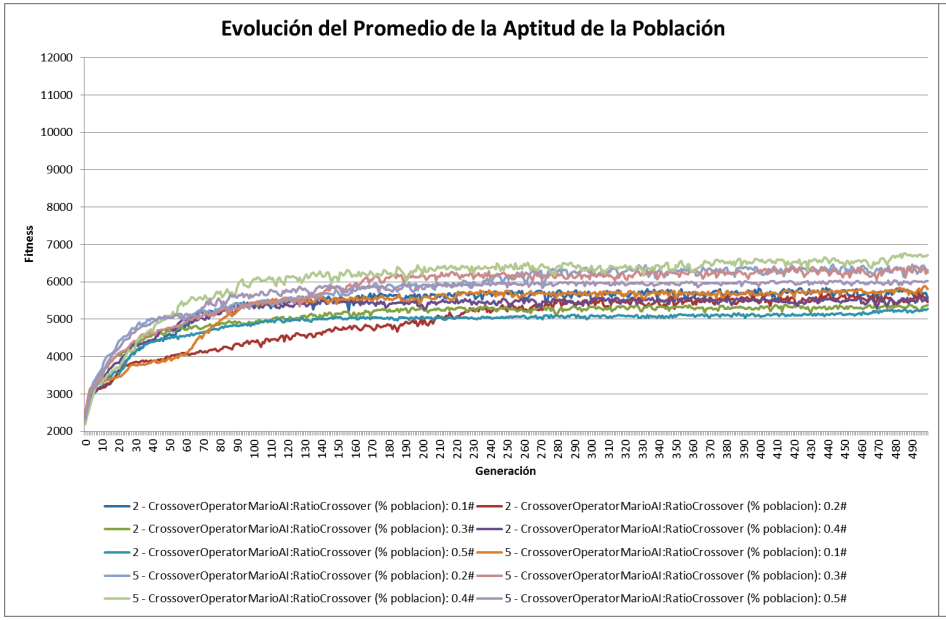


Ilustración 65: Tam. Población 20 individuos

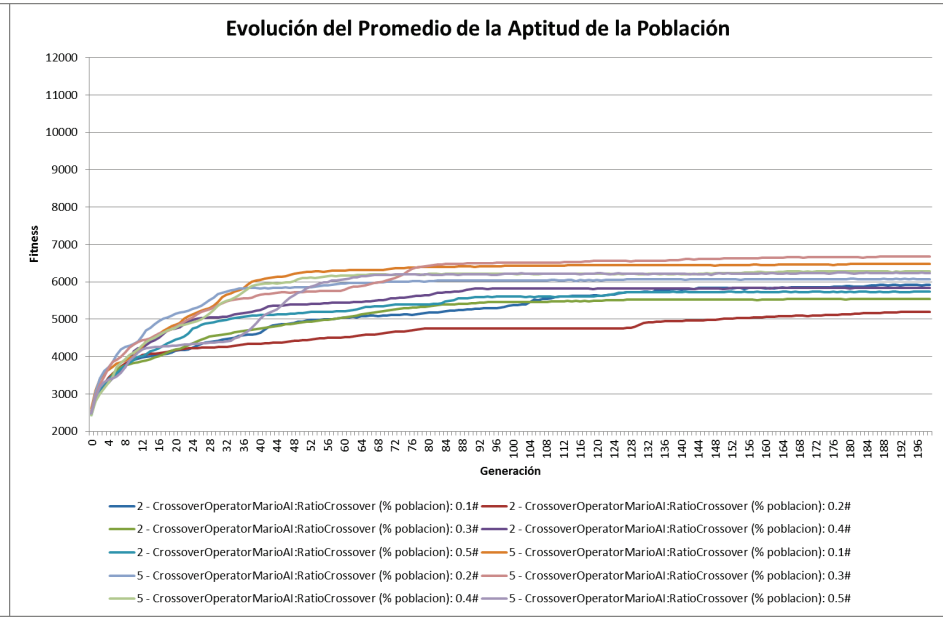


Ilustración 66: Tam. Población 50 individuos

Nivel Rosa → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -ls 11062011 -ld 4

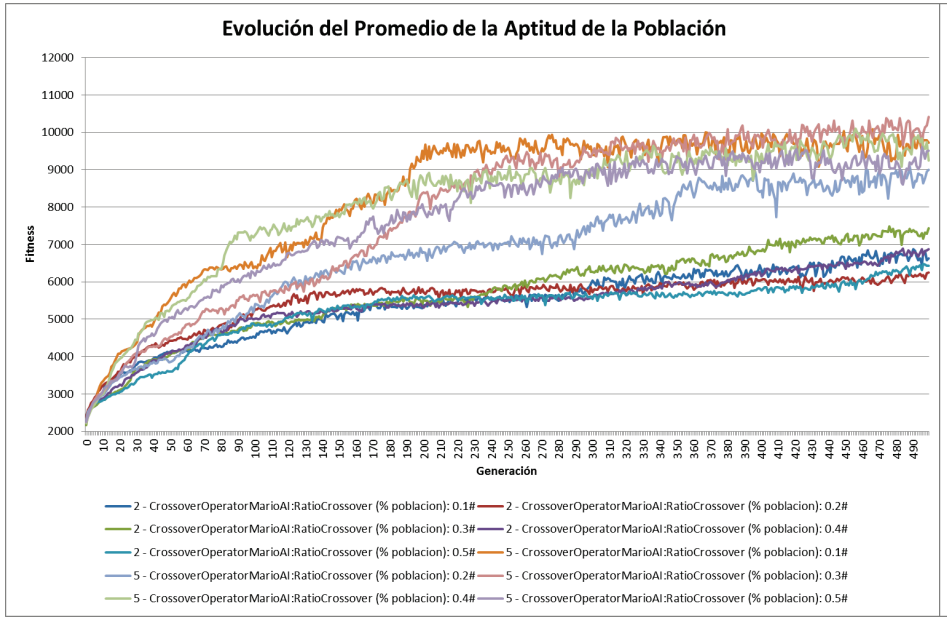


Ilustración 67: Tam. Población 20 individuos

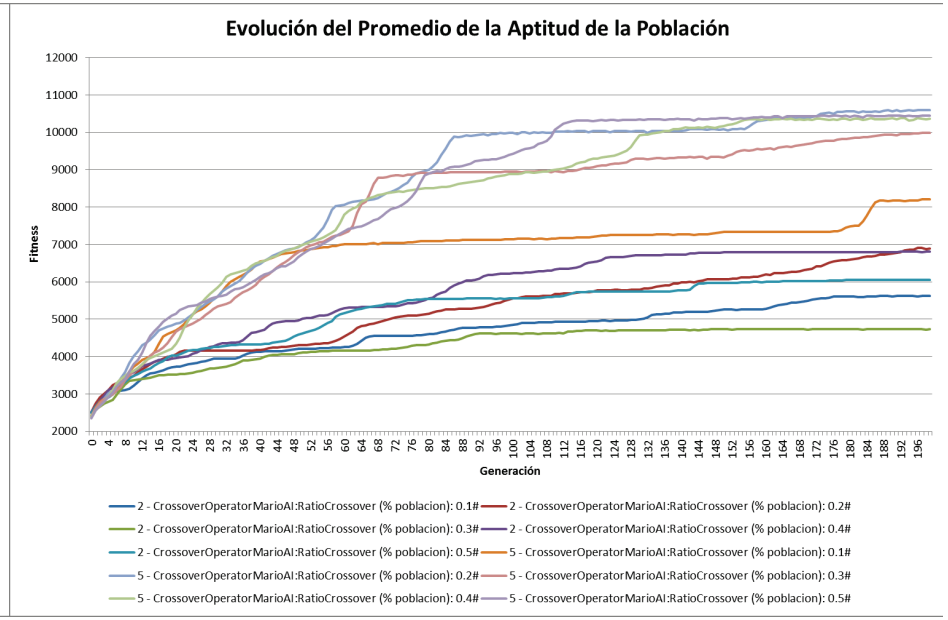


Ilustración 68: Tam. Población 50 individuos

Nivel Granate → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -ls 11062011 -ld 4

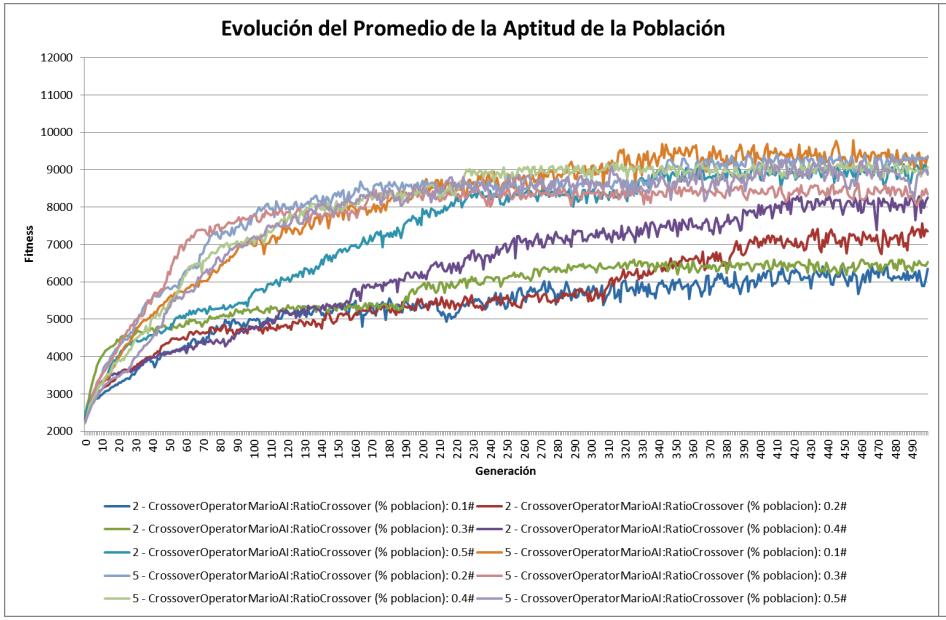


Ilustración 69: Tam. Población 20 individuos

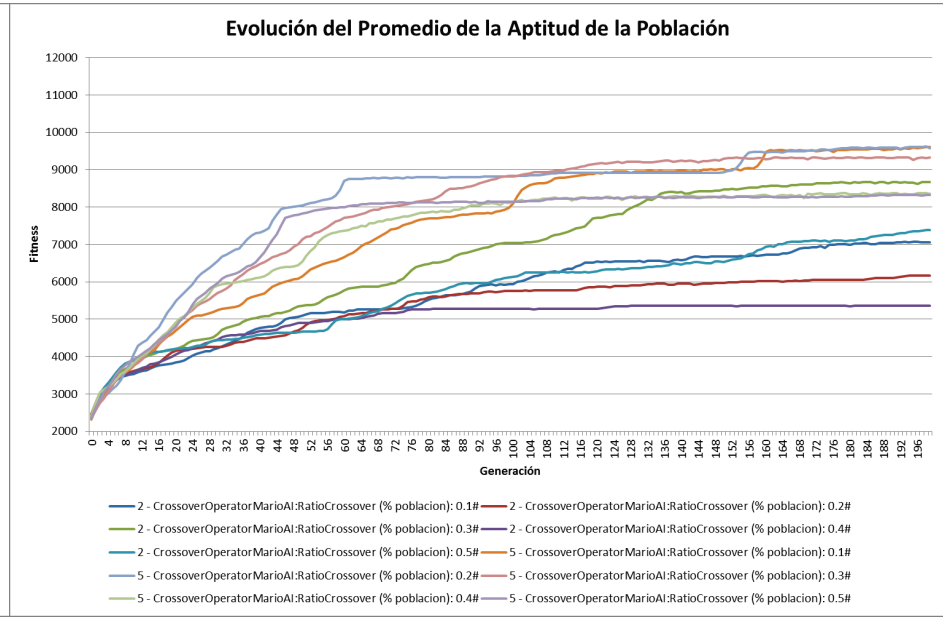


Ilustración 70: Tam. Población 50 individuos

Finalmente, como último punto a analizar, en estas gráficas de evolución, *Ilustración 59* e *Ilustración 60*, en las que se aprecia que la pendiente de la línea de evolución es muy pronunciada y creciente, cuyo comportamiento es anómalo y erróneo, ya que proporciona valores de aptitud desproporcionados, con lo que desorienta al AG, porque no hay una relación entre el valor de la aptitud con que el individuo haya superado el nivel. Este problema aparece con niveles de dificultad superior a 4 (en la competición *Learning Track* de Mario AI únicamente se prueban los niveles de 0 a 4), ya que se quiso probar el AG Mejorado con niveles de dificultad altos. Analizando el problema se llegó a la conclusión que era un fallo interno del juego, que fue reportado a los diseñadores del videojuego para su análisis (en este momento están pendientes de su solución), el comportamiento anómalo viene dado cuando un individuo supera el nivel, a aprender, ciertos valores del entorno del juego, como número de enemigos eliminados, monedas conseguidas, el estado del juego en victoria (es decir, que se ha superado el nivel) se quedan almacenadas en el sistema del juego y no se reinician en las siguientes evaluaciones, y como estos valores forman parte de la función de evaluación, repercute en el valor de la aptitud de todos los individuos proporcionando un comportamiento erróneo en el AG, como tener mejores individuos con valores de aptitud altos, pero que no consiguen superar el nivel, cuando se vuelven a ejecutar las acciones de dicho individuo en el juego. Este problema solo se ha detectado en niveles de dificultad superiores a 4, por lo tanto, no afecta a la competición *Learning Track* de Mario AI.

- **Parámetro de configuración "Tipo de Inicialización"**

En esta misma sección del apartado 4.4 se vio que la inicialización guiada obtenía mejores resultados que con la aleatoria. Pero esta inicialización no era efectiva en ciertos niveles con una gran dificultad.

El principal inconveniente que tiene la inicialización guiada es que se pierde diversidad genética al tener en la mayor parte de su genoma el mismo conjunto de genes o acciones iniciales. Por lo tanto, en el momento que se encuentra atascada la búsqueda de soluciones, es decir, cuando Mario se localiza en un punto del nivel en el cual el conjunto de acciones (genes) a ejecutar no consiguen completar el nivel, por ejemplo, si tiene que volver sobre sus pasos para sortear algún obstáculo; es muy difícil que el AG consiga orientarse, ya que la mayoría de los individuos de la población tienen secuencias de acciones similares, y la potencia del algoritmo quedaría en manos, únicamente, del operador mutación, es decir, el operador cruce no tendría el efecto deseado en el algoritmo.

Aun así, las ventajas de esta inicialización son muy importantes para el entorno en el que se desarrolla el problema, éstas son: se consigue obtener una solución óptima al problema mucho antes que con la inicialización aleatoria, por lo tanto, nos ayuda a cumplir la restricción del número máximo de evaluaciones permitidas; y que el comportamiento del personaje Mario sea más coherente a cómo jugaría una persona.

Por otro lado la inicialización aleatoria tiene un comportamiento opuesto a la inicialización personalizada, es decir, las ventajas de uno son las desventajas del otro y viceversa. Por lo tanto, la inicialización aleatoria tiene más diversidad genética ya que tiene más posibilidades de encontrar una secuencia de acciones óptima, pero en contra, tiene que necesitar muchas más generaciones en obtener esta secuencia de acciones que

en el caso de la inicialización guiada, además el comportamiento de Mario es un poco errático al realizar muchas acciones que no tienen sentido o que no llevan a nada.

Esta situación, lleva pensar en por qué no intentar unir las ventajas de una y otra inicialización, por lo tanto, se desarrolló un nuevo tipo de inicialización, la inicialización mixta, que consiste en lo siguiente: en el momento de inicializar un individuo se elige de forma aleatoria, para cada gen, que tipo de inicialización tomar o bien aleatoria o bien guiada. Con esto, se consigue que la secuencia del genoma del individuo tenga la ventaja de la diversidad genética con la ventaja del guiado de Mario hacia a delante con la inicialización de los movimientos "*DERECHA + VELOCIDAD*" o "*DERECHA + SALTAR + VELOCIDAD*".

Esta nueva inicialización, mixta, obtiene el comportamiento esperado en la cuestión de completar un nivel del juego, pero el comportamiento de Mario no es del todo esperado, ya que en ocasiones, durante el nivel, el comportamiento de Mario es algo errante.

Por lo tanto, como último paso, se decide, que es más conveniente para la población inicial del AG que haya diferentes tipos de inicialización para una misma población, y así, el operador de cruce permitirá que las desventajas de ambas inicializaciones se disminuyan y se mantengan las ventajas a lo largo de las generaciones.

Finalmente la inicialización de la población se realiza de la siguiente forma: para cada individuo se elige de forma aleatoria una de las tres inicializaciones posibles, la inicialización aleatoria, la inicialización guiada o la inicialización mixta. Con este tipo de inicialización se consigue que encuentre más rápidamente una secuencia de acciones que consiga salir de un estancamiento en la búsqueda de secuencias óptimas para completar un nivel satisfactoriamente.

- **Parámetro de configuración "Granularidad"**

Este parámetro se mantiene igual que en el apartado 4.4, aunque se vio en dicho apartado que con una granularidad 5 se obtenían en la mayoría de la veces, mejores resultados que el resto, pero con granularidad 2 también se obtenían buenos resultados, por lo tanto, se ha decidido utilizar ambas granularidades para la última experimentación, y así se puede comprobar finalmente qué configuración de granularidad es la más adecuada para el AG.

A continuación se mostrará una serie de gráficas, de la *Ilustración 71* a la *Ilustración 80* de los diferentes experimentos realizados y se comprobará con qué granularidad se obtienen mejores resultados. Las gráficas presentan el Promedio de la Aptitud de la Población, según la tasa de cruce (0.1, 0.2, 0.3, 0.4 y 0.5), la constante ventana de mejora (2, 3 y 5) y la granularidad (2 y 5). En dichas gráficas, se puede verificar como sistemáticamente en cada una de ellas, los promedios de aptitud de la población con el valor 5 en la granularidad son mayores que con granularidad 2. Esta afirmación se puede comprobar gracias a la siguiente tabla, *Tabla 12*, en la que el promedio de la aptitud en todos los niveles es mayor con granularidad 5 que con granularidad 2, con una diferencia de 2320 puntos aproximadamente. Para facilitar la lectura de la *Tabla 12*, se ha asociado la configuración del nivel de Mario AI con un color que corresponde con el color

mostrado en las tablas, *Tabla 8*, *Tabla 9*, *Tabla 10*, y *Tabla 11*, y que dicho color se indican en el encabezado de las gráficas junto con la opciones del nivel de Mario AI.

				Granularidad	
				2	5
Niveles	Rojo	Tamaño Población	20	5.168,86	6.416,90
			50	5.412,30	6.603,34
	Rosa		20	5.471,68	7.798,00
			50	5.056,71	8.010,11
	Granate		20	6.120,73	7.970,65
			50	5.704,88	7.690,33
	Marrón		20	5.322,51	7.582,70
			50	4.952,99	7.520,60
	Azul		20	5.402,30	8.689,79
			50	5.272,18	8.807,41
	Total			***5388,52***	***7708,98***

Tabla 12: Promedio de la Aptitud de la Población según el Nivel de Mario AI

Nivel Rojo → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -lde on -ls 01121987 -ld 4

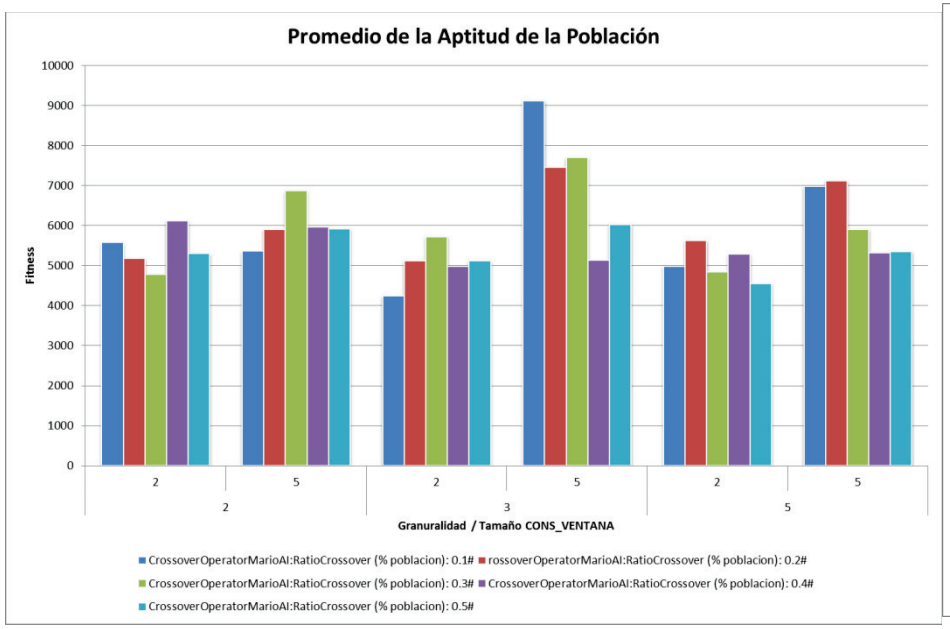


Ilustración 71: Tam. Población 20 individuos

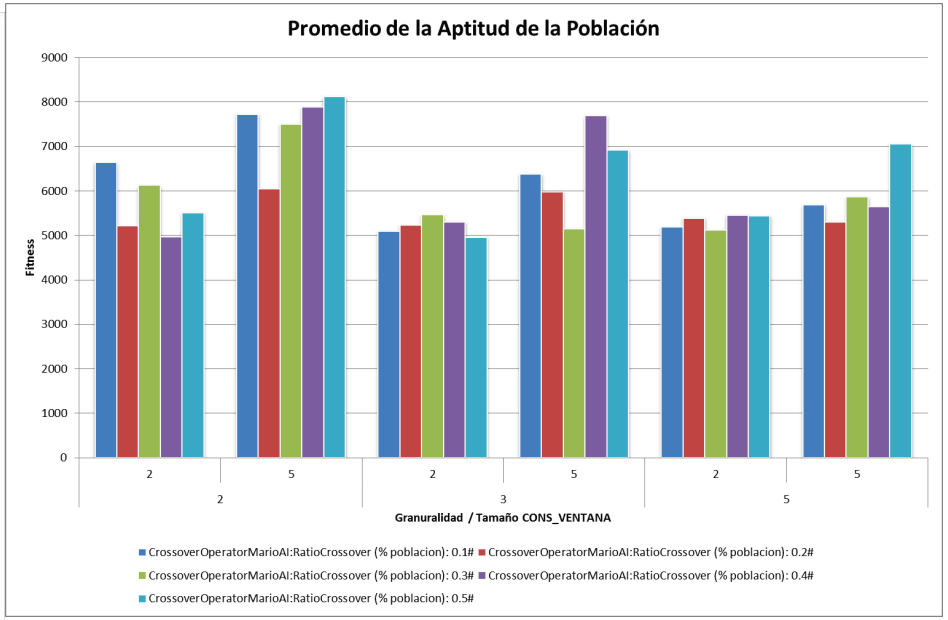


Ilustración 72: Tam. Población 50 individuos

Nivel Rosa → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -ls 11062011 -ld 4

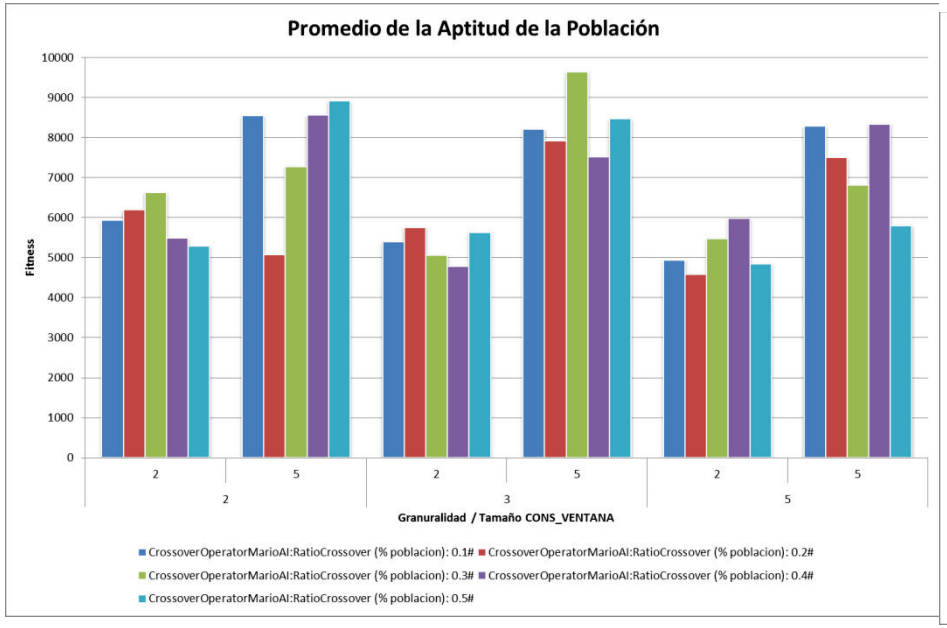


Ilustración 73: Tam. Población 20 individuos

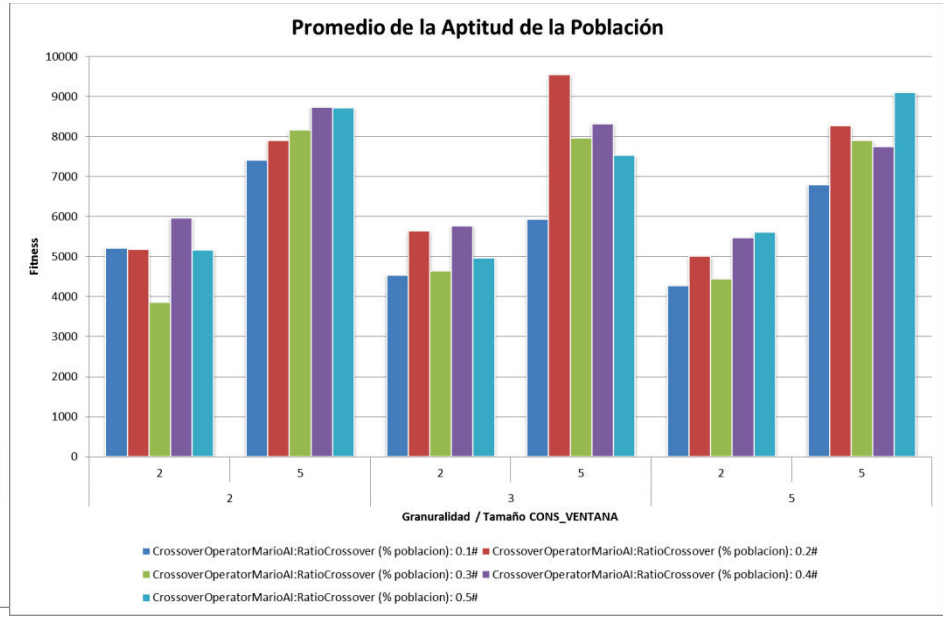


Ilustración 74: Tam. Población 50 individuos

Nivel Granate → Opciones Mario: -vis off -lhb on -lla on -lde on -ls 444 -ld 4

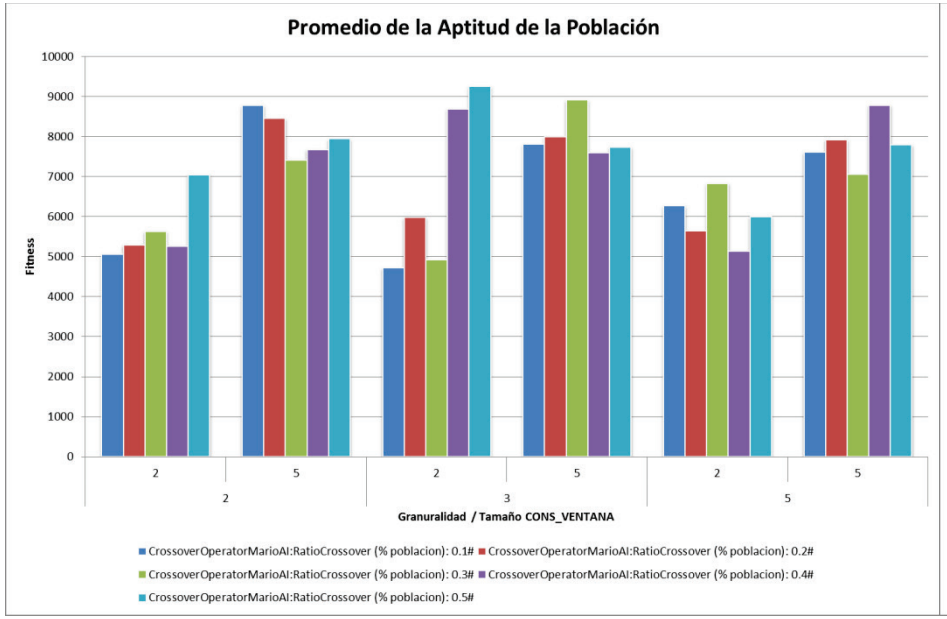


Ilustración 75: Tam. Población 20 individuos

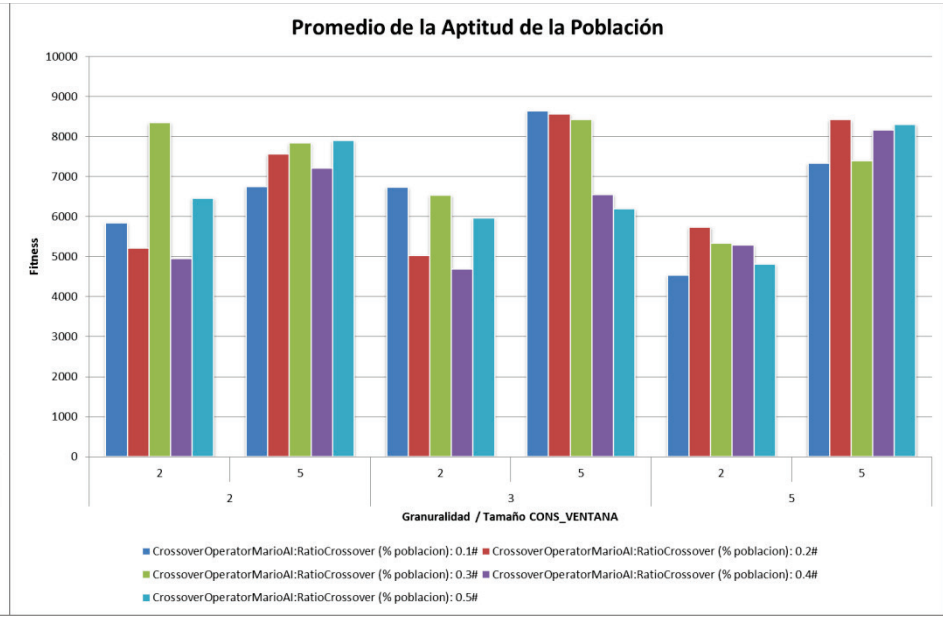


Ilustración 76: Tam. Población 50 individuos

Nivel Marrón → Opciones Mario: -vis off -lhb on -lla on -lde off -ls 334 -ld 4

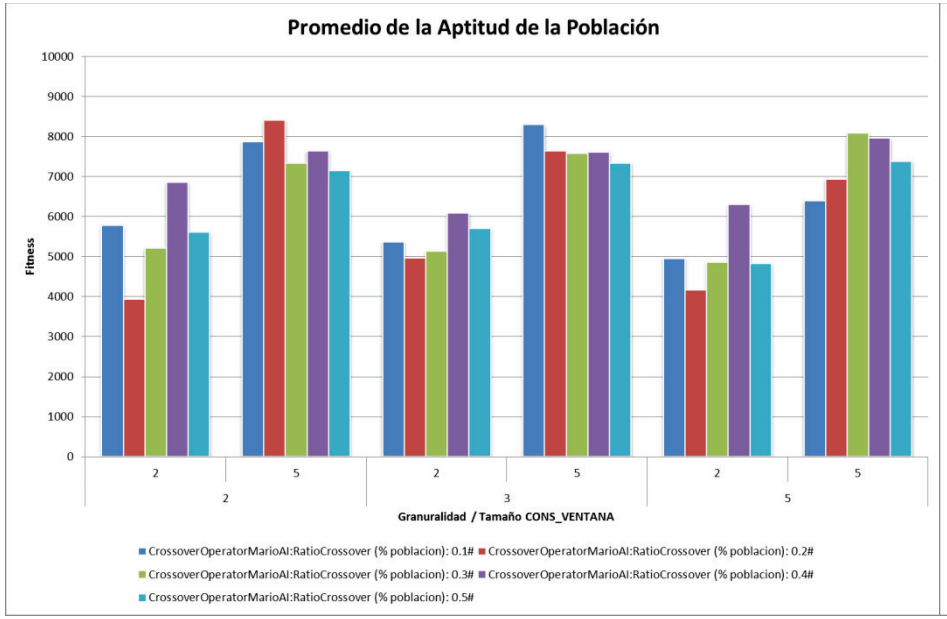


Ilustración 77: Tam. Población 20 individuos

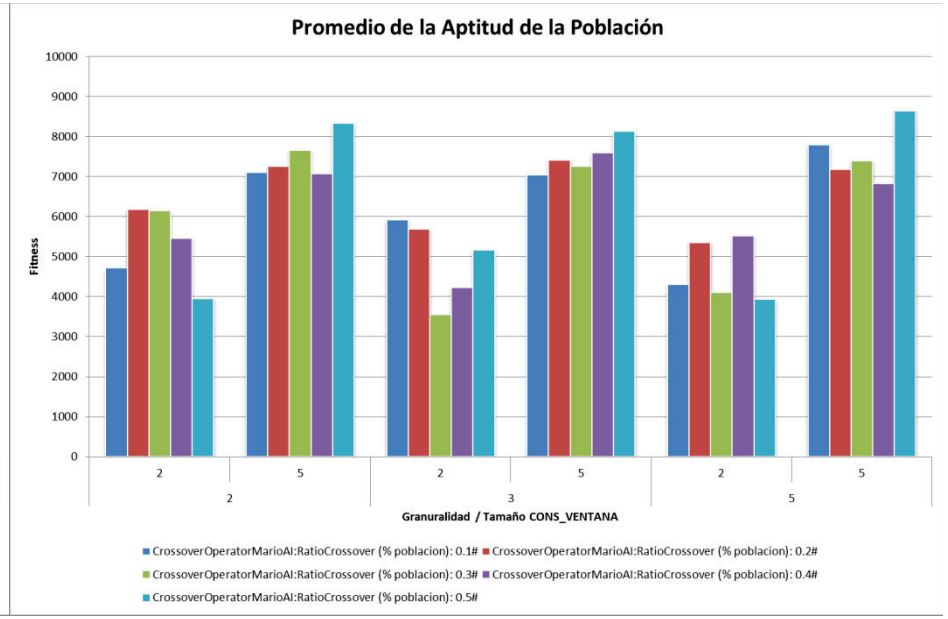


Ilustración 78: Tam. Población 50 individuos

Nivel Azul → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -lde on -ls 201183 -ld 4

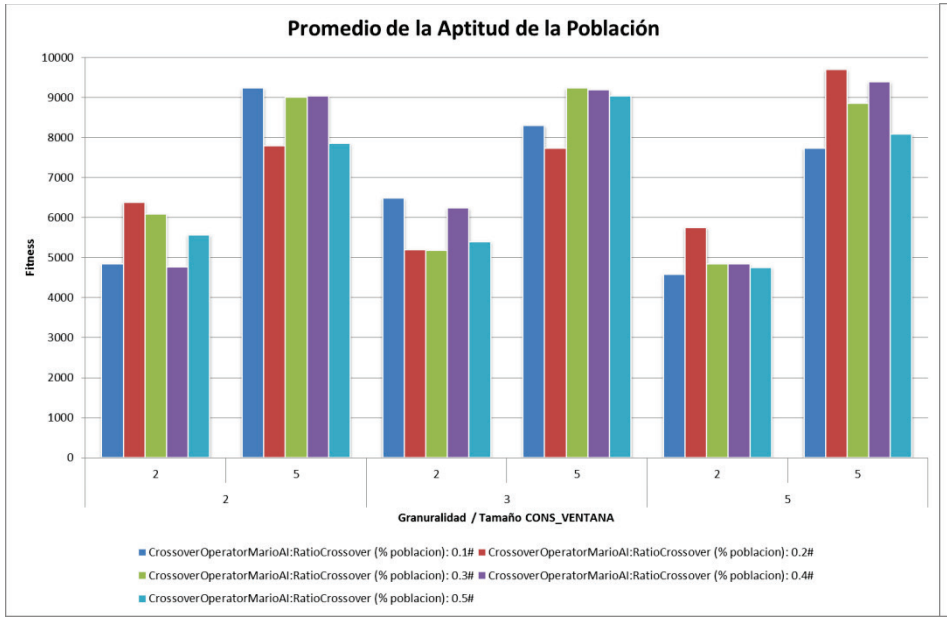


Ilustración 79: Tam. Población 20 individuos

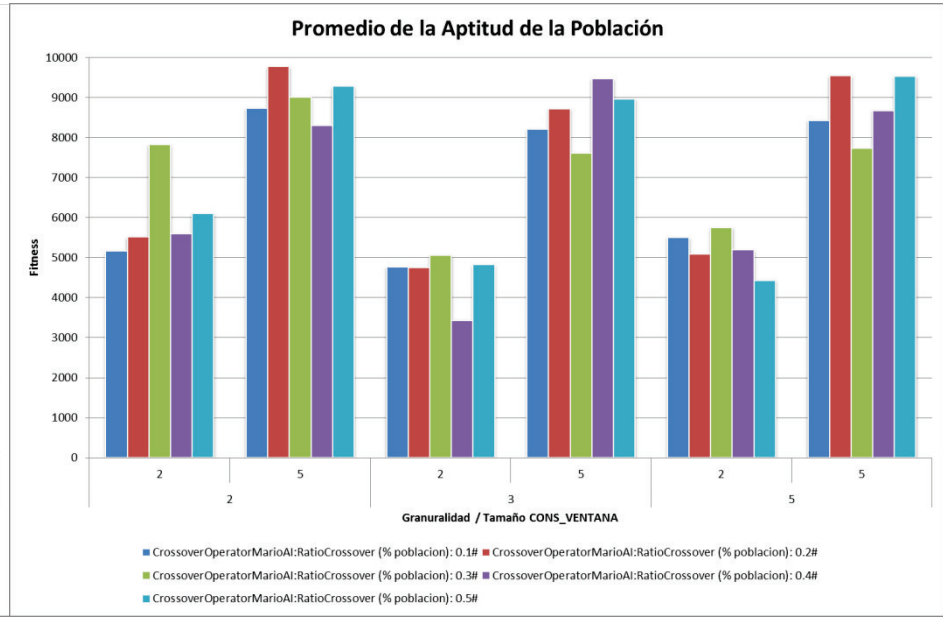


Ilustración 80: Tam. Población 50 individuos

- **Parámetro de configuración "Tamaño Población"**

Como ya se vio en esta sección del apartado 4.4, el tamaño de la población es muy importante para la variabilidad genética en el algoritmo genético, además también se vio que el algoritmo funcionaba ligeramente mejor con un tamaño de población de 50 individuos, pero era muy similar cuando la inicialización era guiada, debido a que la población era muy similar teniendo en un gran porcentaje el mismo tipo de movimientos.

En esta nueva experimentación, se ha cambiado la configuración de algunos parámetros, como los operadores genéticos. Junto a este motivo, y que en la experimentación anterior no había mucha diferencia entre poblaciones de 20 y 50 individuos, se ha decidido mantener los dos valores posibles en este parámetro (20 y 50 individuos).

A continuación se muestra una serie de gráficas, que proporcionará información sobre la influencia del tamaño de la población en los experimentos realizados. Estas gráficas muestran el número de niveles completados por experimentación y tamaño de población. Finalmente se indicará cuál es la configuración elegida, basándose en la que tenga mejores resultados en los experimentos realizados, resumidos en una tabla.

En las gráficas se muestra el número de veces que se ha completado el nivel en los experimentos realizados, para los niveles elegidos. La experimentación engloba otros "sub-experimentos" para cada configuración de los parámetros definidos.

Las gráficas *Ilustración 81*, *Ilustración 83*, e *Ilustración 85* son experimentos en los que se ha establecido el tamaño de la población en 20 individuos. Mientras que las gráficas *Ilustración 82*, *Ilustración 84*, e *Ilustración 86* son experimentos con tamaño de población de 50 individuos.

En las dos primeras y dos últimas gráficas (*Ilustración 81*, *Ilustración 82*, *Ilustración 85*, *Ilustración 86*) se puede observar que el número de niveles superados es bastante mayor en el caso de que el tamaño de la población sea de 50 individuos. Sin embargo, en las gráficas, *Ilustración 83* e *Ilustración 84*, la diferencia se reduce, y los resultados son similares, aun así se tienen más niveles superados cuando el tamaño de la población es de 50 individuos.

Nivel Rojo → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -lde on -ls 01121987 -ld 4

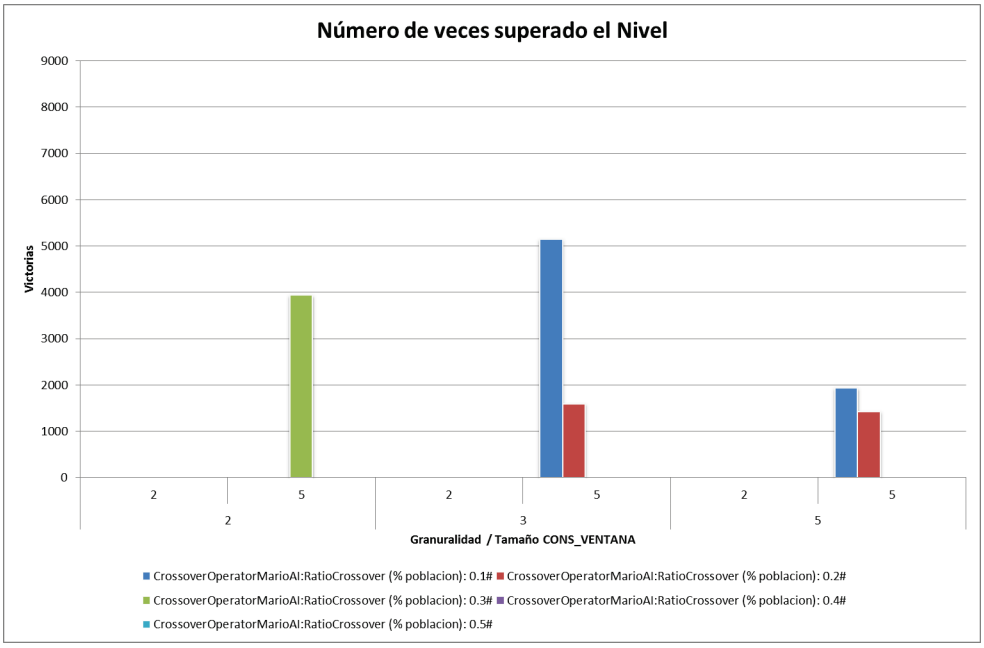


Ilustración 81: Tam. Población 20 individuos

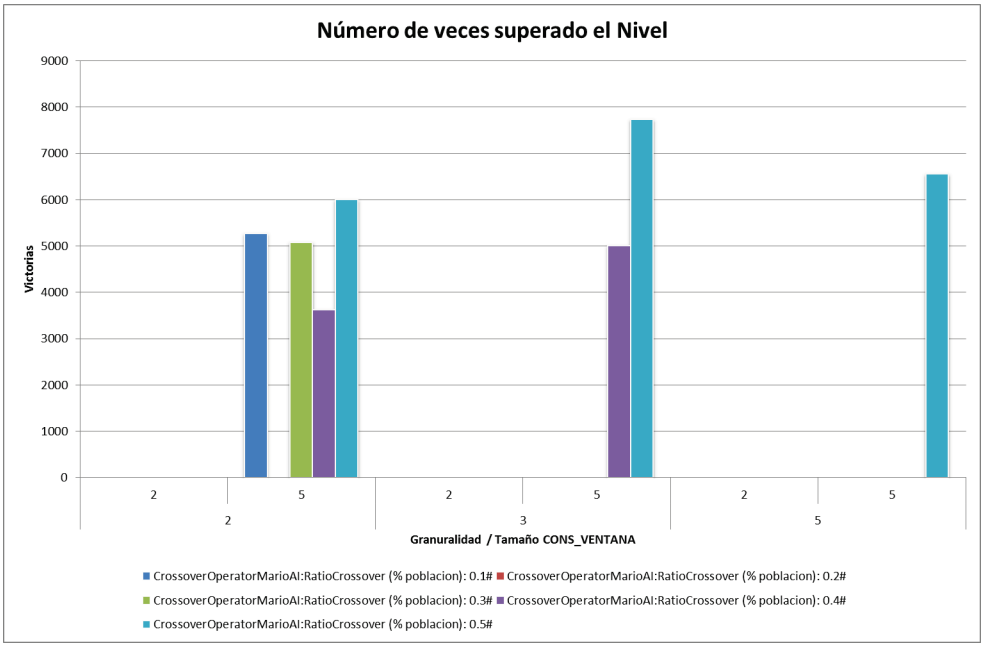


Ilustración 82: Tam. Población 50 individuos

Nivel Rosa → Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -ls 11062011 -ld 4

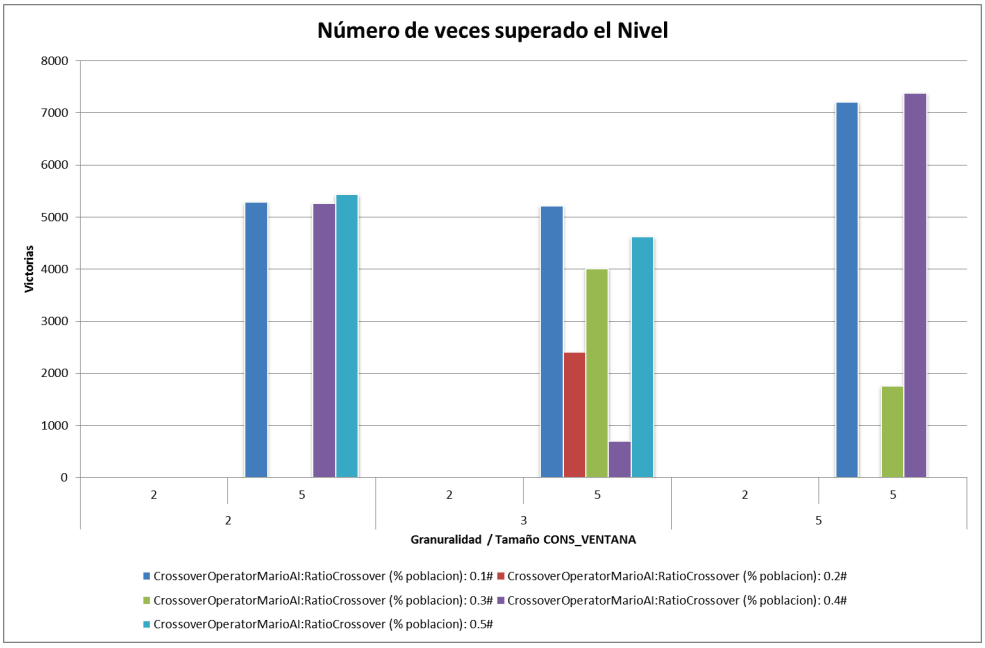


Ilustración 83: Tam. Población 20 individuos

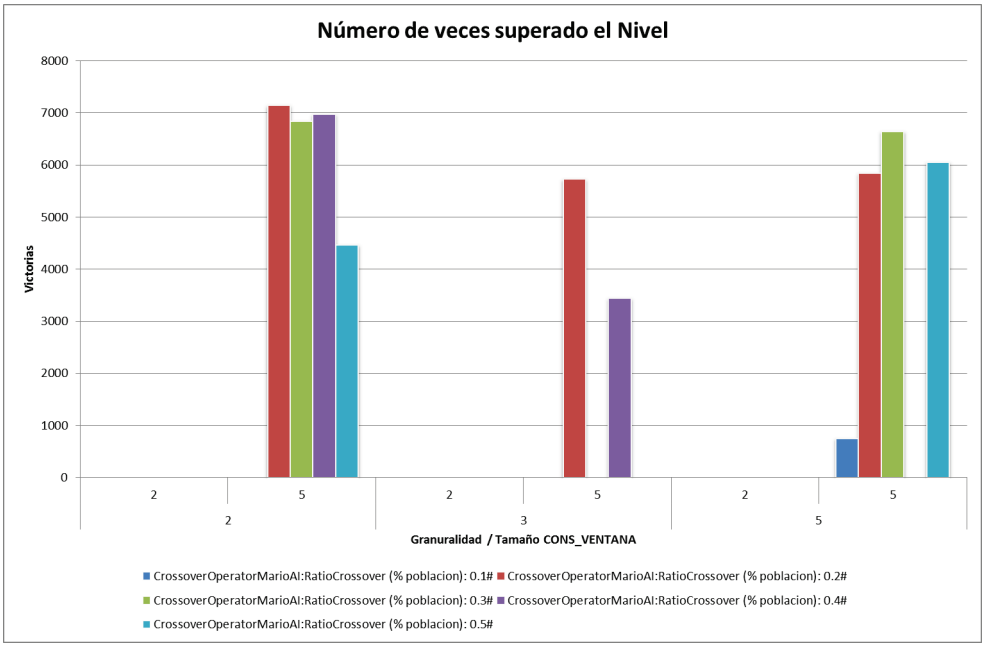


Ilustración 84: Tam. Población 50 individuos

Nivel Granate → Opciones Mario: -vis off -lhb on -lla on -lde on -ls 444 -ld 4

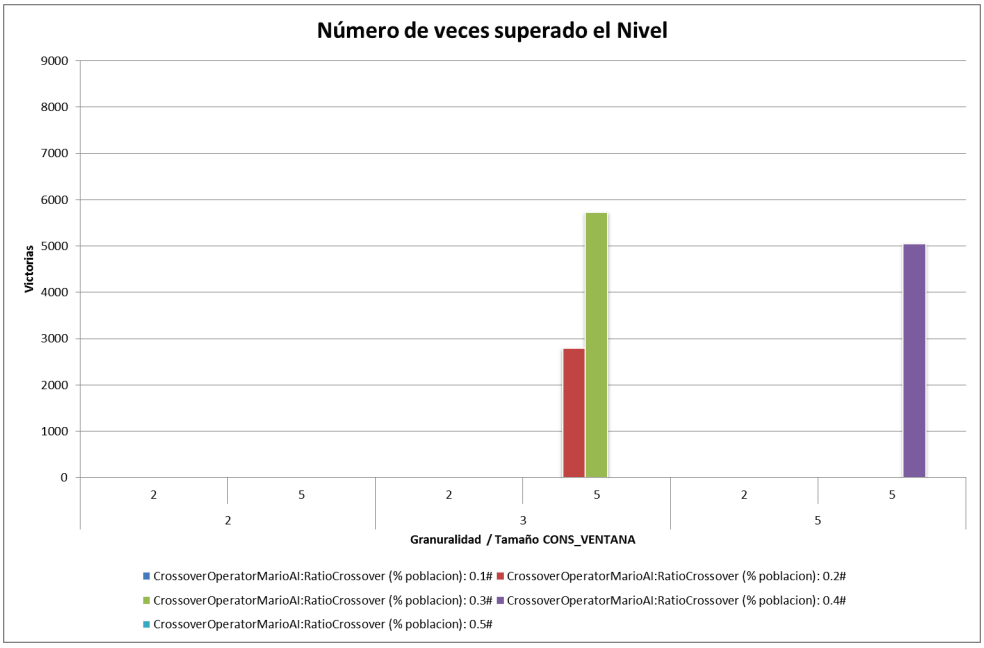


Ilustración 85: Tam. Población 20 individuos

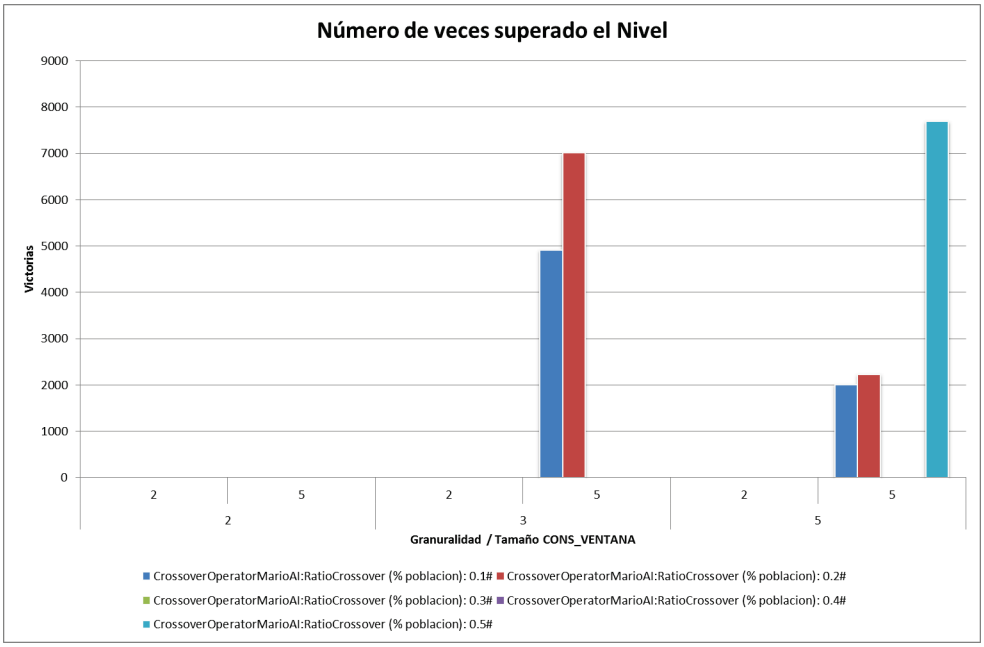


Ilustración 86: Tam. Población 50 individuos

A continuación se muestra una tabla resumen, *Tabla 13*, del número de niveles superados por el AG, con las configuraciones de nivel realizadas en la experimentación, de igual forma que en la tabla anterior, se ha asociado los diferentes niveles de Mario AI con los colores que se corresponden con las tablas, *Tabla 8*, *Tabla 9*, *Tabla 10*, y *Tabla 11*. En esta tabla se puede observar que la mejor configuración del parámetro de Tamaño de Población es la que tiene el valor de 50 individuos en la población que requiere el AG.

	Tamaño de Población	
Nivel	**20**	**50**
Rojo	14.070	39.322
Púrpura	142.199	168.148
Salmón	4.841	3.353
Rosa	49.338	53.935
Granate	13.590	23.875
Marrón	107.604	108.822
Azul	62.158	74.054
Total	***393.800***	***471.509***

Tabla 13: Número de niveles completados según el experimento

- **Parámetros de configuración "Operadores Genéticos Cruce y Mutación"**

En la primera experimentación, se vio la necesidad de tener que modificar los operadores genéticos por defecto que proporciona el framework de *JGAP*. Y aunque se obtienen buenos resultados con estos operadores, en los niveles con mayor complejidad y dificultad ya no se tienen los resultados esperados. Por lo tanto, en esta segunda experimentación se modifican estos operadores para adaptarlos al problema en cuestión, tomando como base los operadores por defecto del *JGAP*.

En el apartado 3.7 ya se detallaron el funcionamiento de los nuevos operadores genéticos, operador de cruce y operador de mutación. A continuación se muestra una serie de gráficas que permitirán determinar qué valores de los parámetros de configuración, que admite el operador, funciona mejor para el dominio de Mario AI. El parámetro que utiliza es la Tasa de Cruzamiento que puede tomar valores de 0.1, 0.2, 0.3, 0.4, ó 0.5. El objetivo de estas gráficas es determinar cuál es la configuración que permite a Mario completar el nivel en el menor número de generaciones, ya que en la competición está limitado el número de generaciones. Las gráficas expuestas están filtradas por el tamaño de población y la granularidad, es decir, son graficas que representan los experimentos con un tamaño de población de 50 individuos, y con

granularidad de 5. Ya que, como se ha visto en los puntos anteriores, es la configuración que mejor funciona, por lo tanto, el objetivo de este punto es determinar que tasa de cruce funciona mejor con esta configuración.

En las gráficas, *Ilustración 87* e *Ilustración 88*, se observa que la tasa de cruce que antes consigue completar el nivel es la de 0.3 alrededor de la generación 42 de 200. En la siguiente gráfica, *Ilustración 89*, obtiene resultados peores que el resto de las tasas, aunque completa el nivel antes de la mitad del número máximo de generaciones. Aunque en la gráfica *Ilustración 90* se ve que la tasa de cruce de 0.3 es la segunda mejor completando el nivel entorno a la generación 42.

El siguiente operador genético utilizado es el operador de Mutación, de igual forma que el operador anterior, en esta experimentación se ha modificado dicho operador para adaptarlo al dominio del problema. Como ya se vio en el apartado 3.7, dónde se definieron los dos tipos de mutación desarrollados, en el primero, en la mutación por defecto, se comprobó que una mutación excesiva del individuo afecta al rendimiento del individuo. Una tasa de mutación de 100 es, por general, la mejor configuración en la primera experimentación, esto quiere decir, que se muta a un gen por cada 100 del cromosoma del individuo.

Por lo tanto, el siguiente paso, a realizar, es definir un nuevo operador de mutación, la cualidad principal de este nuevo operador, es que la mutación se realiza en los genes apropiados, que por lo general es cerca del último gen ejecutado, o lo que es lo mismo, la última acción ejecutada. Para añadirle algo de variabilidad, también se muta de forma aleatoria con una probabilidad de mutación de $\frac{1}{100}$ los genes que forman el cromosoma del individuo, con esto se consigue que pueda aparecer un subcamino alternativo para completar el nivel que obtenga una mejor puntuación.

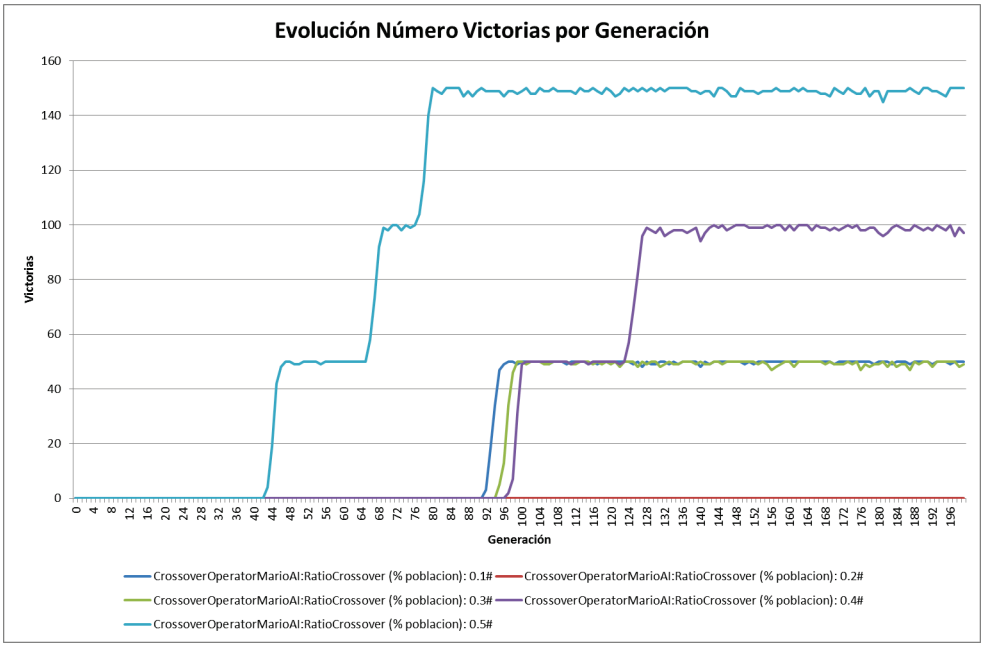


Ilustración 87: Tam. Población 50 individuos. Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -lde on -ls 01121987 -ld 4

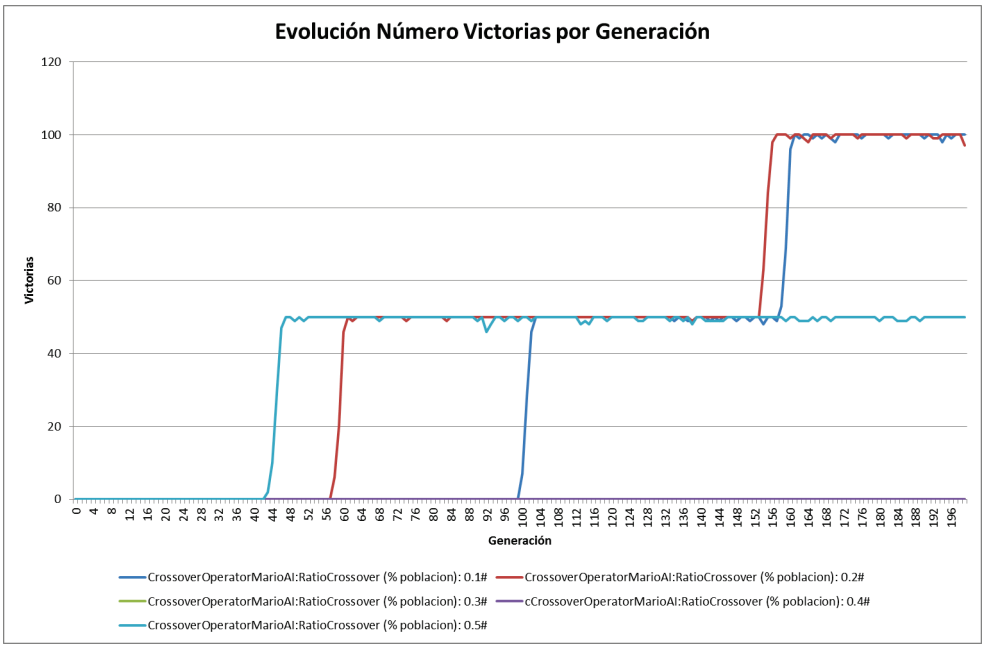


Ilustración 88: Tam. Población 50 individuos. Opciones Mario: -vis off -lhb on -lla on -lde on -ls 444 -ld 4

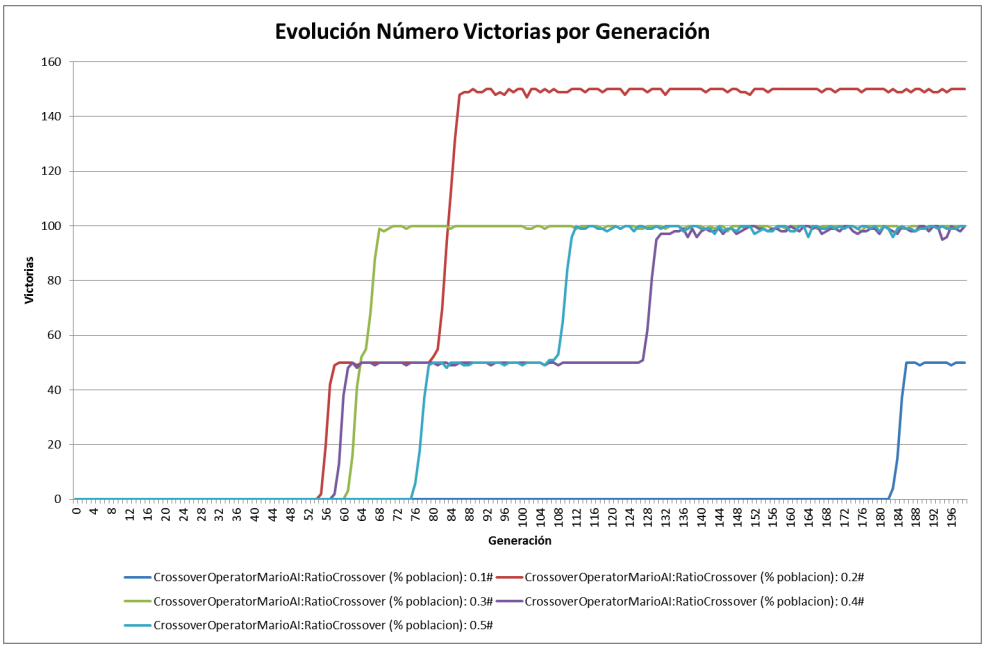


Ilustración 89: Tam. Población 50 individuos. Opciones Mario: -vis off -lhb on -lt 0 -ll 300 -ls 11062011 -ld 4

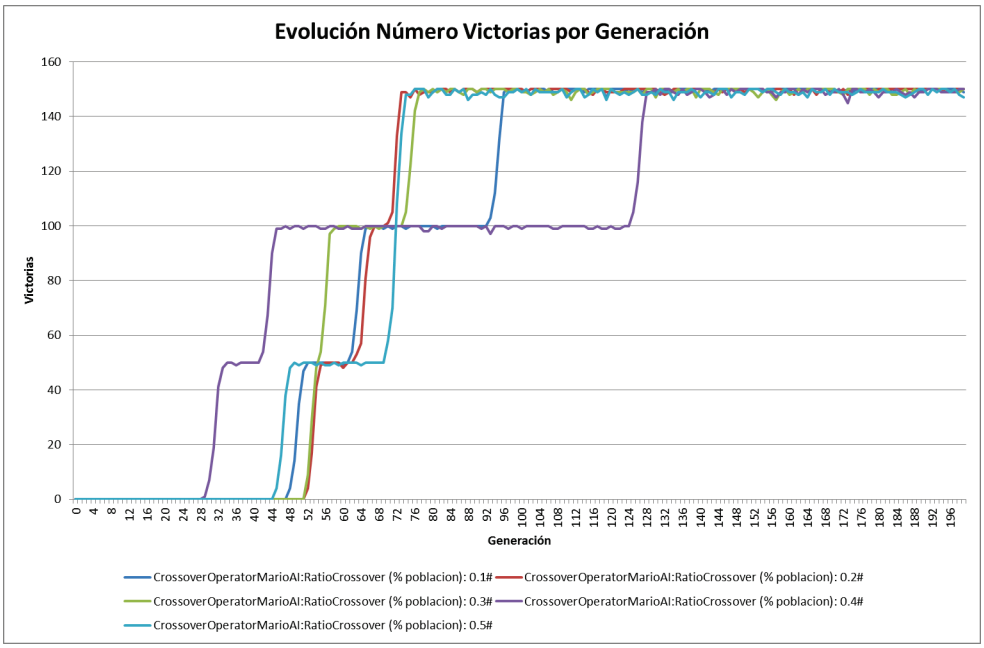


Ilustración 90: Tam. Población 50 individuos. Opciones Mario: -vis off -lhb on -lla on -lde off -ls 334 -ld 4

➢ Parámetro de configuración "Operador de Genético de Selección"

Como ya se vio en la experimentación anterior, para este parámetro de configuración, sólo hay un valor posible, en este caso, con un tamaño de torneo del 15 %. Este torneo, a diferencia, del utilizado en la primera experimentación, es que se le añade el concepto de elitismo, es decir, el primer individuo que se añade a la nueva población, tras una nueva generación, es el mejor individuo que haya habido en la población, el resto de individuos se escoge con el procedimiento habitual de un Operador de Selección por Torneo.

➢ Parámetro de configuración "Constante Ventana de Mejora"

Este parámetro de configuración se diseñó para dotar al nuevo operador de mutación una mayor versatilidad, a la hora de cuándo se tiene que aumentar la ventana de mutación de dicho operador. En concreto, este parámetro, indica cuántas veces puede no mejorar un individuo, de una generación a otra para aumentar la ventana de mutación, como se detalla en el apartado 3.7.

A continuación, se presentan dos tablas, en la primera tabla, *Tabla 14*, se tiene para cada valor posible de *Constante Ventana*, el promedio de la aptitud para todos los niveles experimentados según el tamaño de la población. Como se puede observar para un tamaño de población de 20 individuos el valor de *Constante Ventana* que mejores resultados obtiene es de 3, mientras que para un tamaño de población de 50 individuos el valor que mejores resultados obtiene es de 2. Por lo tanto, se puede deducir, que este parámetro va ligado al tamaño de la población que a su vez determina el número de generaciones máximo.

La segunda tabla, *Tabla 15*, tiene para cada valor posible de *Constante Ventana* el número de niveles completados para todos los niveles experimentados según el tamaño

de la población. Esta tabla tiene el mismo comportamiento que la anterior, es decir, para un tamaño de población de 20 individuos el valor de *Constante Ventana* que mejores resultados obtiene es de 3, mientras que para un tamaño de población de 50 individuos es de 2, debido a que dicho parámetro va ligado al tamaño de la población.

Por lo tanto, cuanto mayor número de generaciones se tenga en el aprendizaje de AG menor será la necesidad de tener un valor alto en el parámetro *Constante Ventana de Mejora*, y en el caso contrario, cuanto menor número de generaciones se tenga en el aprendizaje del AG mayor será la necesidad de tener un valor alto en dicho parámetro.

		Tamaño de Población	
		20	**50**
CONST. VENTANA	**2**	7.365,62	7.665,07
	3	7.791,57	7.367,38
	5	7.173,99	7.348,34

Tabla 14: Relación entre el Promedio de Aptitud con la Constante Ventana y el tamaño de la población.

		Tamaño de Población	
		20	**50**
CONST. VENTANA	**2**	49.908	81.377
	3	67.987	68.515
	5	53.816	67.872

Tabla 15: Relación entre el número de niveles completados con la Constante Ventana y el tamaño de la población.

El objetivo principal de esta segunda experimentación ha sido comprobar que los cambios realizados en diferentes aspectos del algoritmo genético inicial han funcionado, obteniendo, gracias a ellos, muy buenos resultados, como completar diferentes niveles con una dificultad bastante alta que en la versión inicial del algoritmo no era capaz de completar.

También, se ha podido determinar qué valores de los parámetros de configuración del AG funcionan mejor, obteniendo los mejores resultados posibles. A continuación se detallarán cuáles son los valores que debe de tener la configuración con mejores resultados.

El parámetro de configuración *Tipo de inicialización*, en esta segunda versión, ha desaparecido, porque se ha diseñado una inicialización que engloba los tres tipos de inicializaciones, comentadas en el apartado anterior y en el apartado 3.6, debido a su buen funcionamiento.

El parámetro de configuración *Granularidad*, en esta segunda versión del AG, se ha podido comprobar que con **granularidad de 5** los resultados son significativamente mejores que con granularidad 2.

El parámetro de configuración *Tamaño de Población*, en esta nueva versión del AG, tiene un mejor resultado, en número de niveles completados, con un valor de **50 individuos** que con una valor de 20 individuos, habiendo una diferencia de *77.709* a favor para un tamaño de población de 50 individuos.

El parámetro de configuración *Operadores Genéticos: Cruce y Mutación*, en esta segunda versión, ha sido más eficiente que el utilizado en la primera experimentación, ya que se ha definido un operador de mutación específico para el problema de Mario AI, el cual no tiene ningún parámetro de configuración. Por otro lado, el operador de cruce, también se ha definido nuevamente para adaptarlo, pero tomando como base el operador de cruce genérico. En este caso si hay diferentes valores de configuración, en concreto, a la hora de determinar la tasa de cruce, que en términos generales se ha comprobado que la **tasa de cruce de 0.3** (es decir, que se cruzan el 30% de la población para crear nuevos individuos) es la que mejor funciona, obteniendo muy buenos resultados en todos los experimentos realizados.

El parámetro de configuración *Operador Genético de Selección*, en ambos, experimentos toma siempre la misma configuración, con un método de selección por Torneo, con un tamaño de torneo del 15%, y en el caso de la nueva versión del AG, con elitismo.

Finalmente, el nuevo parámetro de configuración *Constante Ventana de Mejora*, este parámetro está relacionado con la nueva mutación, permite controlar cuándo aumentamos la ventana de mutación de un individuo. Se ha observado que el valor que mejores resultados se ha obtenido ha sido de **3** con un tamaño de población de **20 individuos**, mientras que para un tamaño de población de **50 individuos** es de **2**. También se ha podido deducir que este parámetro varía indirectamente según el número de generaciones máximo permitido. Ya que para 200 generaciones, como máximo, se tiene que el valor de *Constante Ventana* es de 2, mientras que para 500 generaciones, como máximo, se tiene un valor de 3.

A continuación se muestra una tabla, *Tabla 16*, con los resultados que hubo en el CIG del 2010. El primer clasificado tiene una puntuación de 45.017 puntos en los 5 niveles, por lo tanto, de media se tiene 9.003 puntos por nivel, cuyo valor es menor que los mejores individuos generados por el AG diseñado en los diferentes niveles probados, como se muestra en la tabla siguiente, *Tabla 17*, que indica la mejor puntación obtenida al

completar un determinado nivel por diferentes agentes generados por el AG diseñado en presente proyecto.

Participantes	Asociación	Puntuación	Clasificación
Slawomir Bojarski and Clare Bates Congdon	University Southern Maine	45.017	1
FEETSIES team: Erek Speed, Stephie Wu, Tom Lieber		44.801	2
Laura Villalobos		41.024	3
Robin Baumgarten	Imeprail College, London	19.054	4

Tabla 16: Resumen de resultados de la competición Learning Track de Mario AI

	Tamaño de Población	
Nivel	**20**	**50**
Rojo	13.340	14.169
Rosa	12.416	11.913
Granate	10.975	10.861
Marrón	9.779	10.508
Azul	12.509	12.844
Total	***11.804***	***12.059***

Tabla 17: Puntación (Aptitud) de diferentes agentes generados por AG diseñado.

Capítulo 5

Conclusiones y Líneas Futuras

En este último capítulo del presente proyecto, se expondrá un breve resumen de todo lo que ha conllevado la realización del mismo. En primer lugar, se presentarán los objetivos establecidos al comienzo del proyecto, y la motivación que se ha tenido para realizarlos. A continuación se expondrá, a modo de conclusión, un análisis sobre la consecución de los objetivos definidos anteriormente. Finalmente, se propondrán líneas futuras de investigación que se podrían abordar a partir de esta investigación, con el objetivo de cubrir sus limitaciones.

5.1 Motivación y Objetivos

Como se ha visto en el primer capítulo, la evolución de los videojuegos ha sido muy similar a la evolución del juego en los seres humanos, los primeros videojuegos eran muy simples y primitivos, en los que sólo participaban uno o dos jugadores, mientras que en la actualidad es posible que participen numerosos jugadores en los llamados juegos en línea (on-line) con gráficos en alta definición.

Dentro del mundo de los videojuegos las partes que más han evolucionado han sido la parte gráfica y la parte sonara. Se han invertido grandes sumas de dinero para desarrollar videojuegos con efectos gráficos de alta calidad y efectos sonoros de alta definición. Pero hay otra característica igual de importante o quizás más para los jugadores de videojuegos, y es la jugabilidad. Dentro de la jugabilidad de un videojuego

se encuentra la inteligencia artificial de los enemigos, llamados NPCs. Hasta hace unos años, la industria del videojuego se ha centrado en los aspectos gráficos y sonoros de los videojuegos dejando a un lado la inteligencia o comportamiento de los NPCs, siendo el comportamiento de estos NPCs, un comportamiento prefijado. Por lo tanto, el jugador acabará por aprenderse dicho comportamiento, convirtiéndose el juego muy predecible y aburrido para el jugador. En este caso, se tiene la excepción de algunos juegos clásicos, como el ajedrez, las damas, y similares, cuyos NPCs pueden ser mucho más complicados ya que no son juegos desarrollados en entornos dinámicos, y en los que los NPCs deben responder en tiempo real.

En los últimos años, las compañías de desarrollo están invirtiendo más recursos en la inteligencia de los NPCs, para que el comportamiento de los NPCs no sea tan predecible, y se adapten a las necesidades del jugador, al principio la dificultad del juego es más fácil para el jugador, y mientras el jugador va adquiriendo más habilidades en el juego, la dificultad va aumentado adaptándose a la habilidad del jugador.

La mayoría de las técnicas de IA desarrolladas en los videojuegos se centran en el comportamiento de los NPCs, con el objetivo final, de que el comportamiento de los NPCs sea lo menos predecible posible por el jugador, por lo tanto, que sea variable en diferentes momentos del juego. Pero desarrollar este comportamiento, mediante técnicas de IA, requiere de una fase de pruebas muy exhaustiva ya que se necesita determinar que los comportamientos generados cumplan con las expectativas deseadas. Por lo tanto, surge la necesidad de automatizar estas pruebas mediante agentes autónomos capaces de evaluar la IA de los NPCs desarrollados.

Existen numerosas investigaciones las cuales se centran en desarrollar herramientas de IA para realizar comportamientos complejos y adaptativos para los NPCs, pero estos comportamientos deben ser probados para confirmar que los resultados son lo esperado. Luego, tiene que surgir o está surgiendo una nueva línea de investigación que se centre en desarrollar agentes autónomos, preferiblemente mediante técnicas de IA, y que estos agentes puedan probar la calidad de la IA desarrollada para los NPCs, por lo tanto, los agentes tomarán el papel del jugador en un videojuego. Estas investigaciones también pueden ayudar a crear agentes que, en vez de sustituir al jugador, sean un compañero más en el juego en aquellos juegos en los que es necesario la participación de varios jugadores (videojuegos online, videojuegos multijugadores, videojuegos deportivos, etc.), y otra posible función de estos agentes podría ser la de guía o consejero en el juego, sobre todo al comienzo del juego cuando el dominio del jugador es más reducido.

Finalmente se deduce, que los objetivos del presente proyecto para cubrir las necesidades vistas anteriormente, son los siguientes,

1. Diseñar y desarrollar un agente autónomo inteligente, proporcionado por técnicas de inteligencia artificial, con el objetivo de completar uno o varios niveles de un videojuego.

2. El primer paso para diseñar dicho agente, es realizar un estudio y análisis de los diferentes tipos de videojuegos existentes que permitan su modificación o posean un API para desarrollar un agente que pueda jugar de forma autónoma.

3. Seguidamente, analizar, para cada videojuego, qué técnicas de inteligencia artificial se adaptan mejor para desarrollar un agente que complete uno o varios niveles.

4. Tras el análisis de los diferentes videojuegos, hay que elegir uno de ellos, para desarrollar un agente inteligente que sea capaz de jugar de forma óptima, mediante una de las técnicas de inteligencia artificial elegida, tras un análisis de las mismas.

5. Y finalmente utilizar el agente diseñado para que pueda competir en alguna competición ya organizada, e intentar obtener unos buenos resultados, es decir, o bien ganar la competición o al menos comprobar que se tienen resultados muy similares a los de los mejores participantes.

Tras revisar los objetivos principales del proyecto, se verá continuación cuáles de ellos se han cumplido y cómo se han conseguido cumplir.

5.2 Conclusión

El objetivo de diseñar y desarrollar un agente autónomo inteligente, proporcionado por técnicas de inteligencia artificial, para completar uno o varios niveles de un videojuego, ha sido satisfecho en su totalidad, se ha conseguido diseñar un agente autónomo e inteligente que es capaz de superar niveles de gran complejidad, no superables por un jugador con cierta habilidad. Es decir, el diseño del agente, ha superado en gran medida las expectativas generadas sobre el agente. Para conseguir cumplir el primer objetivo ha sido necesario cumplir antes los tres objetivos siguientes.

En el apartado 2.3 se ha seleccionado una serie de videojuegos, de diferentes géneros, en los que se ha hecho un análisis de sus características y funcionamiento de cada videojuego. Así mismo, se han indicado numerosas investigaciones en las que se han aplicado técnicas de IA, o bien para desarrollar NPCs inteligentes o bien agentes inteligentes capaces de jugar al videojuego de forma autónoma. Finalmente se ha expuesto para cada videojuego, las ventas y desventajas que tiene el mismo, para desarrollar un agente autónomo inteligente. Con este estudio se ha conseguido obtener una visión general de las diferentes investigaciones realizadas con técnicas de IA sobre diferentes videojuegos populares, una de las principales conclusiones que podemos llegar de estas investigaciones es que no es trivial aplicar técnicas de IA en los videjuegos, ya que en la mayoría de las investigaciones se aplica en una parte concreta del videojuego, aunque eso también depende del tipo de videojuego en cuestión, ya que los hay más simples en los que se puede aplicar IA y obtener buenos resultados.

El siguiente paso realizado, es elegir el videojuego a analizar para desarrollar un agente, se ha especificado la elección del videojuego elegido, en concreto, **Mario AI**. En los apartados 3.2, y 3.3, se ha hecho una justificación del porqué de su elección, los principales motivos de su elección han sido tener una documentación relativamente buena sobre el funcionamiento del juego; se tiene una API básica con la que se interactua el agente con el juego; está desarrollado en un lenguaje de programación muy común,

JAVA; existe varios ejemplos simples de agentes ya desarrollados; es un juego muy conocido en el mundo de los videojuegos y finalmente existe una competición anual en la que se puede evaluar la calidad del agente desarrollado. Y a continuación, se ha realizado un análisis y descripción del videojuego, para tener un mayor conocimiento del juego y así poder determinar qué técnicas de IA funcionan mejor.

Finalmete se han visto las diferentes técnicas de IA que se pueden aplicar al juego de AI, por una lado tenemos la llamadas técnicas *en línea*, que son aquellas que deciden qué acción tomar según el estado actual del juego, estas técnicas necesitan de algún tipo de evaluación sobre la aptitud de cada acción del conjunto de acciones posibles, pero para evaluar estas acciones es necesario predecir cuál será el estado siguiente del juego, por lo tanto es necesario simular cual será el siguiente estado del juego al realizar una acción determinada, a favor tiene que esta técnica puede ser utilizada para cualquier nivel o configuración del nivel. Otra alternativa a esta técnica es utilizar algún tipo de técnica de aprendizaje que en base al estado del juego decida cuál es la siguiente acción a ejecutar, pudiéndose utilizar las técnicas de aprendizaje por refuerzo, como Q-Learning. Otras técnicas que se pueden utilizar son las llamadas técnicas *fuera de línea*, las cuales consisten en lo siguiente, al tener un resultado final de la partida se define el conjunto de acciones que obtienen mejores resultados, por lo tanto, no se tiene en cuenta el estado del juego en cada instante, sino que solamente se tiene en cuenta el resultado final de la partida. Este tipo de técnicas tiene la ventaja de que no es necesario representar el estado del juego, por lo que el algoritmo requiere de menos tiempo de ejecución, por el contrario, tiene la desventaja que el conjunto de acciones seleccionado solo funciona para un nivel en concreto. Aun así, y debido a la naturaleza de la competición del *LearningTrack* de la *Mario AI Championship*, se ha elegido esta tipo de técnica en concreto se han utilizado algoritmos genéticos para poder completar un nivel satisfactoriamente.

Por lo tanto se ha diseñado un agente basado en la técnica de IA, Algoritmos Genéticos, con el cuál se ha realizado una gran cantidad de experimentos para poder comprobar que funciona correctamente en cualquier tipo de nivel que se configure. En el capítulo 3 ha quedado recogido un resumen del funcionamiento de los AG y el análisis y diseño del agente desarrollando.

Finalizado el desarrollo de la investigación, se ha presentado el diseño de los experimentos, sus resultados y así como el proceso de optimización de los parámetros del AG, en el capítulo 4. En este capítulo, se ha diseñado dos fases de experimentación, en ambas fases se ha utilizado el framework *JGAP* (proporciona mecanismos de genética básica). En la primera fase de experimentación se utilizado el AG por defecto que proporciona el *JGAP*, es decir, se ha utilizado un algoritmo genético simple que es genérico para cualquier dominio, incluiendo los operadores genéticos que proporciona sin ninguna modificación, con está primera fase de experimentación se pretende obtener unos resultados básicos con los que comenzar, además de verificar que se pueden aplicar los AGs en el dominio de Mario AI, seguidamente en la segunda fase de experimentación se tiene el objetivo principal de mejorar los resultados obtenidos en la fase anterior, para ello se han solucionado los problemas encontrados en la primera fase de experimentación modificando el AG simple, finalmente probar este AG mejorado con niveles de Mario AI de una gran dificultad. En resumen, las conclusiones principales que se tienen en la primera fase de experimentación son las siguientes:

1. Con las técnicas basadas en AG se obtienen buenos resultados para el *Learning Track* de la *Mario AI Championship*, ya que para niveles de dificultad media-alta se ha conseguido completar dichos niveles, con esta primera configuración inicial del AG.
2. Se ha comprobado que el operador de mutación genérico del *JGAP* es más efectivo cuanto menor sea el número de genes a mutar.
3. Por último, también se ha comprobado que el operador de cruce genérico del *JGAP* es ineficiente para el domino de Mario AI.

Después de analizar los resultados de la primera fase de experimentación, se ha decido diseñar nuevos operadores genéticos adaptados al dominio de Mario AI, se rediseñó el operador de mutación, el operador de cruce y el selector de torneo. Por lo tanto, en la siguiente fase de experimentación se han analizado los resultados obtenidos con el nuevo diseño. Para ello, se han escogido varios niveles de dificultad alta, no habiendo sido superados por el diseño inicial del AG. Los resultados obtenidos con este nuevo diseño del AG son realmente buenos, consiguiendo completar niveles que una persona no podría completar. A continuación se exponen las conclusiones principales que se extraen de esta segunda fase de experimentación:

1. Se ha conseguido diseñar satisfactoriamente un agente capaz de completar un nivel con una dificultad elevada.
2. Se ha establecido cuál es la mejor configuración de los parámetros del AG, que son:
    a. **Tipo inicialización**: inicialización que engloba en una, la inicialización guiada, la inicialización aleatoria y la inicialización mixta.
    b. **Granularidad**: el valor de granularidad que mejor funciona es el de 5 acciones.
    c. **Tamaño de Población**: el tamaño de población con los que se obtiene mejores resultado es el de 50 individuos.
    d. **Tasa de Cruce del nuevo operador**: la tasa que mejor comportamiento se tiene es la de 0.3.
    e. **Tamaño de Torneo del nuevo selector natural**: con un tamaño de torno del 15% de población se obtienes los resultados esperados.
    f. **Constante Ventana de Mejora**: el valor de la constante Ventana de Mejora varía según el tamaño de la población se tiene un valor de 3 con una población de 20 individuos, y un valor de 2 con una población de 50 individuos.
3. Se ha presentado un agente con esta configuración al *Learning Track* de *Mario AI Championship 2011* en el *CIG Competition Event* en el mes de agosto de 2011, aunque se han suspendido las categorías de la competición de *GamePlay Track* y de *Learning Track* de dicha competición, por lo que no se han podido comparar los resultados obtenidos del agente desarrollado con otros agentes participantes de la competición, pero se pretende presentar el agente en la siguiente competición que se celebra en el *IEEE International Games Innovation Conference* (GIC) en noviembre de 2011. Aun así, comparado con los participantes de años anteriores se ha comprobado que se puede alcanzar puntuaciones similares o incluso superiores que la de los vencedores de años anteriores.

En el CIG del 2010, el primer clasificado tuvo una puntuación de 45.017 en los 5 niveles, por lo tanto, de media tiene 9.003 puntos por nivel, mientras que los agentes generados por el AG diseñado tienen como puntuación media 12.059 puntos.

Por lo tanto, se puede concluir que el AG diseñado obtiene mejores resultados que los agentes que competieron la competición de Mario AI en el año 2010.

5.3 Limitaciones y Líneas Futuras

A lo largo de la realización de este proyecto, y en concreto, durante la fase de desarrollo del agente automático, se han encontrado varias limitaciones, las cuales han sido acotadas y solucionadas, para poder finalizar el proyecto. Las limitaciones que no han podido ser solucionadas por encontrarse fuera del alcance del proyecto, serán idóneas para realizar trabajos o investigaciones futuras.

Una de las limitaciones de este proyecto, viene definida por la limitación de los AGs de generalizar un problema, dicha limitación en inherente a la propia técnica utilizada. En la primera fase de experimentación en la que se ha aplicado un AG genérico, y aunque se han tenido resultados aceptables, se ha comprobado que los operadores genéticos canónicos pueden no funcionar, en este caso, el operador de cruce no ejercía ninguna mejora a la solución por lo tanto la potencia de búsqueda del AG se encontraba en el operador de mutación. Para solucionar esta limitación ha sido necesario analizar, diseñar e implementar nuevos operadores para adaptarlos al dominio y necesidades del problema en cuestión, la competición Mario AI. El algoritmo diseñado puede ser útil para otros videojuegos de características similares, dentro del género de las plataformas, como el conocido videojuego Sonic [49] [50], por lo tanto, un trabajo futuro posible es realizar un sistema capaz de realizar los pequeños cambios necesarios en el AG para adaptarlo al juego, ya que el objetivo principal de este tipo de juegos es el mismo, avanzar hasta el final del escenario, intentando conseguir el mayor número de puntos posibles sin ser eliminado.

Otra limitación, relacionada con la anterior, es que el resultado proporcionado por el AG diseñado, la secuencia de acciones para completar el nivel, es propia para ese nivel, por lo que esta secuencia de acciones no tendría los mismos resultados en otro nivel diferente. Como se puede apreciar, esto abre una nueva posibilidad de investigación, aplicando alguna técnica de inteligencia artificial que sea capaz de generar una agente que pueda completar diferentes niveles, con una misma configuración del agente. El mayor inconveniente que se ha encontrado para utilizar una de estas técnicas (para que el agente complete diferentes niveles), era simular o predecir el resultado que tendría una futura acción en el juego, para ello, antes de aplicar algunas de ellas es necesario diseñar un simulador que consiga predecir de forma eficiente y precisa el resultado de una acción dado un estado del juego, o bien utilizar el simulador desarrollado por Robin Baumgarten [51], utilizando este simular se pueden aplicar técnicas de computación con inspiración biológica, como la optimización mediante colonias de hormigas. Con esta línea de investigación se podría participar en la categoría *GamePlay Track* de la *Mario AI Championship 2011*.

A la hora de realizar la experimentación se encontró el problema de que existe la posibilidad de que un nivel no tenga solución, es decir, no exista ningún posible camino o conjunto de acciones que pueda completar el nivel. Esto es debido, a que la generación de los niveles es aleatoria según una semilla que se establece en el momento de definir un nivel en el entorno de Mario AI, con lo que muchos niveles de la experimentación no fueron productivos en el aprendizaje. Consecuentemente una posible línea futura sería utilizar el algoritmo desarrollado en el presente proyecto para comprobar si un nivel o no tiene solución. Otra línea futura relacionada con la generación automática de niveles, sería evaluar la dificutlad de un nivel de forma automática. Normalmente las técnicas utilizadas para evaluar la dificultad de un nivel requieren la participación de humanos [52]. El algoritmo propuesto, podría utilizarse para evaluar de un modo automático la dificutlad de un nivel, ya que posiblemente la dificultad de un nivel y el tiempo que emplea el algoritmo propuesto en encontrar una solución para dicho nivel, están correlacionados.

Una última limitación encontrada en el desarrollo del agente, son los caminos o callejones sin salida (*dead ends*), que puede aparecer en un nivel, éstos son zonas del nivel del juego en los que un obstáculo, como una pared u otro objeto impide el paso a Mario hacia la derecha, por lo tanto, para solventar este inconveniente Mario debe volver sobre sus pasos hasta encontrar un camino alternativo, es una idea similar a lo que ocurre con los laberintos, en los que se debe ir probando diferentes caminos para llegar a la salida. El agente es capaz de sortear caminos sin salida de una dificultad media-alta, pero si el callejón sin salida es bastante complejo el agente no suele conseguir sortear dicho camino, debido a la limitación de generaciones que impone la competición *Learning*. Un futuro trabajo de investigación es diseñar una heurística que ayude a las técnicas de IA (tanta a los AGs como a otras técnicas) a sortear los caminos sin salida de forma eficiente.

Anexo A. Planificación y Presupuesto

Durante la realización del presente proyecto se ha podido observar que está compuesto por una sucesión de tareas relacionadas entre si. Cada una de estas tareas tiene asociado una duración temporal que conlleva un coste ligado al proyecto.

En este apartado se va a tratar, por lo tanto, de realizar una planificación de las tareas a desarrollar durante el proyecto. Por otro lado, se realizará un presupuesto del coste total que implica la realización del proyecto.

Planificación

Para documentar la planificación se ha utilizado la herramienta de software Microsoft Proyect 2010. Para facilitar la visualización de la documentación sobre la planificación se ha utilizado un diagrama de Gantt, proporcionado por dicha herramienta. Este diagrama es muy utilizado para representar visualmente el tiempo total de la secuencia de actividades relativas al proyecto. Por lo que es un diagrama muy utilizado en la gestión de proyectos.

Un aspecto muy importante en la elaboración de la planificación es la jornada laboral, en este proyecto, para facilitar la planificación y el cálculo de costes, se ha decidido que sea una jornada completa de ocho horas diarias de lunes a viernes.

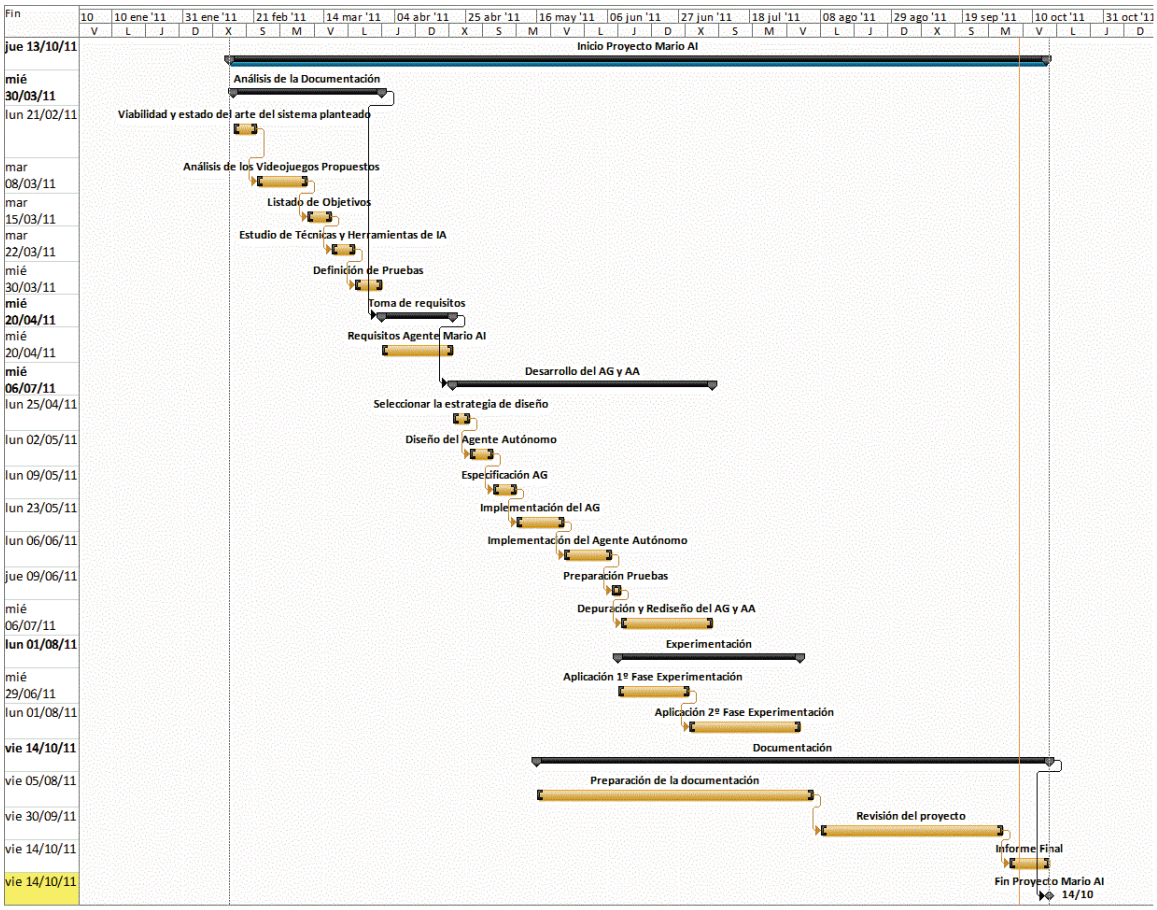


Ilustración 91: Diagrama de Gantt

ID	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Inicio Proyecto Mario AI**	**174 días**	**lun 14/02/11**	**jue 13/10/11**	
2	**Análisis de la Documentación**	**32 días**	**mar 15/02/11**	**mié 30/03/11**	
3	Viabilidad y estado del arte del sistema planteado	5 días	mar 15/02/11	lun 21/02/11	
4	Análisis de los Videojuegos Propuestos	11 días	mar 22/02/11	mar 08/03/11	3
5	Listado de Objetivos	5 días	mié 09/03/11	mar 15/03/11	4
6	Estudio de Técnicas y Herramientas de IA	5 días	mié 16/03/11	mar 22/03/11	5
7	Definición de Pruebas	6 días	mié 23/03/11	mié 30/03/11	6
8	**Toma de requisitos**	**15 días**	**jue 31/03/11**	**mié 20/04/11**	**2**
9	Requisitos Agente Mario AI	15 días	jue 31/03/11	mié 20/04/11	
10	**Desarrollo del AG y AA**	**55 días**	**jue 21/04/11**	**mié 06/07/11**	**8**
11	Seleccionar la estrategia de diseño	3 días	jue 21/04/11	lun 25/04/11	
12	Diseño del Agente Autónomo	5 días	mar 26/04/11	lun 02/05/11	11
13	Especificación AG	5 días	mar 03/05/11	lun 09/05/11	12
14	Implementación del AG	10 días	mar 10/05/11	lun 23/05/11	13
15	Implementación del Agente Autónomo	10 días	mar 24/05/11	lun 06/06/11	14
16	Preparación Pruebas	3 días	mar 07/06/11	jue 09/06/11	15
17	Depuración y Rediseño del AG y AA	19 días	vie 10/06/11	mié 06/07/11	16
18	**Experimentación**	**38 días**	**jue 09/06/11**	**lun 01/08/11**	
19	Aplicación 1º Fase Experimentación	15 días	jue 09/06/11	mié 29/06/11	
20	Aplicación 2º Fase Experimentación	23 días	jue 30/06/11	lun 01/08/11	19
21	**Documentación**	**110 días**	**lun 16/05/11**	**vie 14/10/11**	
22	Preparación de la documentación	60 días	lun 16/05/11	vie 05/08/11	
23	Revisión del proyecto	40 días	lun 08/08/11	vie 30/09/11	22
24	Informe Final	10 días	lun 03/10/11	vie 14/10/11	23
25	Fin Proyecto Mario AI	0 días	vie 14/10/11	vie 14/10/11	21

Tabla 18: Desglose de Tareas

En la figura anterior, *Ilustración 91*, se muestra el diagrama de Gantt completo, que permite apreciar visualmente el alcance temporal de las diferentes tareas de las que se compone el desarrollo del proyecto. A continuación de las imágenes se tiene una tabla, *Tabla 18*, que contiene las diferentes tareas, con sus fechas de inicio y fin, y el número de días utilizados.

Presupuesto

En este apartado se detallan los diferentes costes que se producirán a lo largo del proyecto. Los costes del desarrollo están agrupados en costes de personal, de software, de hardware, de material fungible, de viajes, de dietas y de otros costes. Cada sección contendrá, además de una tabla detallada de costes, una breve explicación de los mismos.

Los costes de personal son los referidos a los salarios de todos los integrantes del grupo de desarrollo, durante el periodo de desarrollo del proyecto. El salario de cada miembro está definido por el rol que desempeñe en el proyecto y por las horas de trabajo. Dependiendo del rol, la hora de trabajo costará más o menos que la de otro miembro del grupo con un rol diferente.

Todos los costes que se indican en la siguiente tabla, *Tabla 19*, son costes brutos, es decir, incluyen todos los impuestos (IRPF, etc.).

Cargo – Rol	Horas de Trabajo	Coste / Hora	Coste Total
Jefe de Proyecto	150	75 €	11.250 €
Analista	256	66 €	16.896 €
Gestor de Calidad	120	63 €	7.560 €
Gestor de Pruebas	528	45 €	23.760 €
Desarrollador	684	38 €	25.992 €
Total acumulado	-	-	85.458 €

Tabla 19: Costes de Personal

Ahora se van a detallar los costes relacionados con los componentes hardwares necesarios para el desarrollo el proyecto.

A continuación se muestra la tabla, *Tabla 20*, que detalla los costes de hardware incluyéndose los precios totales de cada equipo.

Equipo	Unidades	Coste unidad	Coste total
Dell XPS 15z N0815Z08 (Portátil)	2	1.161,23 €	(2.322,46) 348,37 €
Cisco PIX515 (Firewall)	2	900 €	(1.800) 270,00 €
Cisco 2651 XM Dual 10/100 (Router)	1	1.300 €	(1.300) 195,00 €
Total acumulado	-	-	813,37 €

Tabla 20: Costes de hardware

El tiempo de amortización de un ordenador (portátil) se considera de cinco años (60 meses), por lo cual, de los dos ordenadores que son necesarios, sólo se incluye el coste proporcional a los nueve meses en los que los ordenadores son utilizados para el desarrollo del proyecto. En la tabla anterior, *Tabla 20*, se puede ver que hay dos costes totales para los ordenadores, el coste mostrado entre paréntesis es el coste total de los ordenadores, mientras que el coste que aparece a continuación es el coste que se asocia al proyecto, es decir, el coste de los ordenadores para el proyecto es de 348,37 €.

Para conseguir una comunicación completa entre el autor, tutor y director del proyecto, así como poder usar diversas herramientas como Dropbox, se necesita un router que permita el acceso a internet por parte de dichos miembros. Además, puesto que se accede desde los ordenadores en los que se va realizando la documentación y desarrollo del proyecto, se necesita proteger el acceso a dichos documentos, centrándose en evitar accesos inesperados y malintencionados desde el exterior, para ello se adquiere un firewall que controle todos estos aspectos. Al igual que los ordenadores, el tiempo de amortización tanto del router como del firewall se considera de cinco años (60 meses).

A continuación se va a detallar los costes asociados al software necesario para desarrollar el proyecto. En la tabla siguiente, *Tabla 21*, se incluyen los precios totales de cada elemento software.

A diferencia del hardware, el software se compra mediante licencias, es decir, permisos para utilizar un cierto software en un número determinado de ordenadores durante un periodo de tiempo. En algunos casos, dicho periodo de tiempo es ilimitado.

Software	Licencias	Coste Licencia	Coste total
Kaspersky Internet Security 2012 (1 año)	2	35 €	(70) 52,50 €
MS. Windows 7	2	285 €	(570) 85,50 €
MS. Office 2010	2	379 €	(758) 113,70 €
Total acumulado	-	-	251,70 €

Tabla 21: Costes Software

Tanto el antivirus, como el Sistema Operativo y el paquete Office para los dos ordenadores necesitan que se adquiera sus correspondientes licencias para poder hacer uso de ellos. Como la licencia del antivirus tiene una duración de un año, sólo se incluye como costes la cantidad proporcional a nueve meses. Por otro lado, la licencia del Sistema Operativo y del paquete Office no tiene duración limitada, pero se considera cinco años el tiempo de amortización de dicha licencia.

Al observar la tabla de costes, *Tabla 21*, se puede observar que en la última columna (Costes Total), hay dos costes, dónde uno de ellos figura entre paréntesis, es esos casos, el coste que aparece entre paréntesis hace referencia al coste total de la licencia de software, es decir, el coste de la licencia para un determinado periodo de tiempo. El coste que figura a continuación es el coste que se asocia al proyecto, ya que el periodo de desarrollo del mismo es de nueve meses.

Resumen del Presupuesto

Una vez especificados los diferentes costes relacionados con el desarrollo del proyecto, hay que calcular el coste total, para realizar su cálculo se tiene que tener en cuenta el impuesto sobre el valor añadido también conocido como IVA, que en España asciende a un 18% sobre los costes indicados hasta ahora.

También entra en juego dos factores importantes como son el riesgo y el beneficio, ambos clave para la rentabilidad del proyecto así como para la aceptación de la oferta por parte del cliente. Para que el desarrollo del proyecto sea rentable el porcentaje de beneficios es del 20%.

Concepto	Coste Total
Personal	85.458 €
Hardware	813,37 €
Software	251,70 €
TOTAL SIN GASTOS INDIRECTOS	**86.523,07 €**
Gastos Indirectos	8.545,8 €
TOTAL CON GASTOS INDIRECTOS	**95.068,87 €**

Beneficio 20%	1.9013,77 €
Riesgos 10%	9.506,89 €
TOTAL SIN IVA	**123.589,53 €**
IVA 18%	2.2246,11 €
TOTAL CON IVA	**145.835,64 €**

Tabla 22: Costes Totales

El precio final sin IVA asciende a **ciento veintitrés mil quinientos ochenta y nueve euros con cincuenta y tres céntimos** y con IVA a **ciento cuarenta y cinco mil ochocientos treinta y cinco euros con sesenta y cuatro céntimos**.

Referencias

[1] J. Huizinga, Homo Ludens: A Study of the Play Element in Culture, Boston: Beacon Press, 1955.

[2] «Guinness World Records: Gamer's Edition,» [En línea]. Available: http://gamers.guinnessworldrecords.com/index.htm.

[3] R. Sánchez-Pelegrín, M. A. Gómez-Martín y B. Díaz-Agudo, «A CBR module for a strategy videogame,» de *Workshop on Computer Gaming and Simulation International Conference on Case-Based Reasoning*, 2005.

[4] . B. D. Bryant y R. Miikulainen, «Acquiring visibly intelligent behavior with example-guided neuroevolution,» de *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, Menlo Park, CA, 2007.

[5] P. Ulam, A. Goel y J. Jones, «Reflection in action: Model-based self-adaptation in game playing agents,» *Challenges in Game Artificial Intelligence,* vol. AAAI Workshop, pp. 86-90, 2004.

[6] T. R. Hinrichs y K. D. Forbus, «Analogical learning in a turn-based strategy game,» *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence,* pp. 853-858, 2007.

[7] M. Bergsma y P. Spronck, «Adaptive Spatial Reasoning for Turn-based Strategy Games,» de *Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2008.

[8] «Age of Conquest,» Noble Master Games , 17 Mayo 2000. [En línea]. Available: http://www.ageofconquest.com/index.html.

[9] A. A. SVN. [En línea]. Available: svn://svn.icculus.org/alienarena/trunk.

[10] M. McPartland y M. Gallagher, «Creating a Multi-Purpose First Person Shooter Bot with,» de *IEEE Symposium on Computational Intelligence and Games (CIG08)*, 2008.

[11] N. Cole, S. J. L. Miles y C. Miles, «Using a Genetic Algorithm to Tune First-Person Shooter Bots,» *IEEE,* pp. 139-145, 2004.

[12] N. van Hoorn, J. Togelius y J. Schmidhuber, «Hierarchical Controller Learning in a First-Person Shooter,» de *IEEE Symposium on Computational Intelligence and Games*, 2009.

[13] «Alien Arena Guide,» [En línea]. Available: http://alienarena.co.uk/.

[14] «Alien Arena Forum,» [En línea]. Available: http://corent.proboards.com/index.cgi?.

[15] S.-W. Hsu y T.-Y. Li, «Third-Person Interactive Control of Humanoid with Real-Time Motion Planning Algorithm,» de *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, 2006.

[16] «Alien Swarm,» Valve Corporation, [En línea]. Available: http://www.alienswarm.com/.

[17] «Alien Swarm Forum,» Steam Users Forum, [En línea]. Available: http://forums.steampowered.com/forums/forumdisplay.php?f=930.

[18] «Matrox Graphics for Professionals,» Matrox, [En línea]. Available: http://www.matrox.com/graphics/surroundgaming/en/contact/.

[19] «RacerWiki,» The Racer Pit Stop, [En línea]. Available: http://pitstop.totalnfs.net/racerwiki/index.php/AI.

[20] H. Tang, C. H. Tan, K. C. Tan y A. Tay, «Neural Network versus Behavior Based Approach in Simulated Car Racing Game,» *IEEE,* 2009.

[21] L. Cardamone, D. Loiacono y P. L. Lanzi, «Learning to Drive in the Open Racing Car Simulator Using Online Neuroevolution,» *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES,* vol. 2, nº 3, pp. 176-190, 2010.

[22] A. Agapitos, J. Togelius y S. M. Lucas, «Evolving Controllers for Simulated Car Racing using Object Oriented Genetic Programming,» de *Genetic and Evolutionary Computation Conference (GECCO2007)*, 2007.

[23] R. v. Gaal, «Racer the real deal,» Dolphinity B.V., [En línea]. Available: http://www.racer.nl/.

[24] L. s. i. Eufloria, «Euflorium,» [En línea]. Available: http://www.dyson-game.com/smf/index.php?topic=212.0.

[25] «Eufloria,» Eufloria-Game, [En línea]. Available: http://www.eufloria-game.com/news.php.

[26] «Euflorium - Forum Eufloria,» [En línea]. Available: http://www.dyson-game.com/smf/index.php.

[27] D. Keaveney y C. O'Riordan, «Evolving Coordination for Real-Time Strategy Games,» *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES,* vol. 3, nº 2, pp. 155-167, 2011.

[28] R. L. de Freitas Cunha y L. Chaimowicz, «An Artificial Intelligence system to help the player of Real-Time Strategy games,» de *Brazilian Symposium on Computer Games and Digital Entertainment*, Brazil, 2010.

[29] H. Wang, P. H. F. Ng, B. Niu y C. K. Shiu, «Case Learning and Indexing in Real Time Strategy Games,» de *Fifth International Conference on Natural Computation*, 2009.

[30] G. 2. team, «Globulation 2,» [En línea]. Available: http://globulation2.org/wiki/Es:Main_Page.

[31] «Wiki AI Globulation2,» [En línea]. Available: http://globulation2.org/wiki/Making_An_AI_With_Echo_%28part_1%29.

[32] «Globulation 2 Forum,» [En línea]. Available: http://globulation2.org/forums/index.php.

[33] S. Bojarski y C. B. Congdon, «REALM: A Rule-Based Evolutionary Computation Agent that Learns to Play Mario,» de *IEEE Conference on Computational Intelligence and Games (CIG10)*, 2010.

[34] E. R. Speed, «Evolving a Mario Agent Using Cuckoo Search and Softmax Heuristics,» de *2nd International IEEE Consumer Electronics Society's Games Innovations Conference*, 210.

[35] J. Togelius, S. Karakovskiy y N. Shaker, «Mario AI Championship,» [En línea]. Available: http://www.marioai.org/.

[36] J. Togelius, S. Karakovskiy, T. Schaul y J. Koutnik, «Mario AI Benchmark. AI and Machine Learning Experiments based on Super Mario Bros.,» Google Code, [En línea]. Available: http://code.google.com/p/marioai/.

[37] M. Hewner, «Mike Hewner's Homepage,» Marzo 2011. [En línea]. Available: http://hewner.com/wp-content/uploads/2011/03/mario-ai.pdf.

[38] M. Hewner, «Mike Hewner's Homepage,» 09 Marzo 2011. [En línea]. Available: http://hewner.com/2011/03/09/reforcement-learning-mario-ai/.

[39] J. Togelius, «Mario AI Competition,» [En línea]. Available: http://julian.togelius.com/mariocompetition2009/.

[40] S. Karakovskiy y J. Togelius, «The Mario AI Championship - 2010,» 20 Agosto 210. [En línea]. Available: http://www.idsia.ch/~sergey/files/marioai/MarioAI-CIG-2010.pdf.

[41] Mojang, «Infinite Mario Bros,» [En línea]. Available: http://www.mojang.com/notch/mario/.

[42] M. Persson, «Source Infinite Mario Bros,» [En línea]. Available: http://www.mojang.com/notch/mario/release.zip.

[43] J. H. Holland, Adaptation in Natural and Artificial Systems, Ann Arbor, MI: The University of Michigan Press, 1975.

[44] C. Darwin, "El Origen de las Especies", Circulo de Lectores S.A., 1997.

[45] O. D. Weck, «Heuristic techniques: A basic introduction to genetic algorithms,» 2004. [En línea]. Available: http://ocw.mit.edu/NR/rdonlyres/Aeronautics-and-Astronautics/16-888Spring-2004/D66C4396-90C8-49BE-BF4A-4EBE39CEAE6F/0/MSDO_L11_GA.pdf.

[46] J. Togelius, S. Karakovskiy, J. Koutník y J. Schmidhub, «Julian Togelius,» 2009. [En línea]. Available: http://julian.togelius.com/Togelius2009Super.pdf.

[47] S. N. Sivanandam y S. N. Deepa, «Terminologies and Operators of GA,» de *Introduction to Genetic Algorithms*, Springer, 2008, pp. 39-82.

[48] S. Karakovskiy y J. Togelius, «Learning Track - Mario AI Championship,» [En línea]. Available: http://www.marioai.org/LearningTrack.

[49] «Sonic Wiki,» [En línea]. Available: http://es.sonic.wikia.com/wiki/Portada.

[50] «Sonic Channel,» Sega, [En línea]. Available: http://sonic.sega.jp/SonicChannel/index.html.

[51] R. Baumgarten, «Infinite Super Mario AI,» [En línea]. Available: http://www.doc.ic.ac.uk/~rb1006/projects:marioai.

[52] N. Shaker, J. Togelius, G. Yannakakis, B. Weber, T. Shimizu, T. Hashiyana, N. Sorenson, P. Pasquier, P. Mawhorter, G. Takahashi, G. Smith y R. Baumgarten, «The 2010 Mario AI Championship: Level Generation Track,» de *Computational Intelligence and AI in Games, IEEE Transactions on*, 2011.