UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5

Algoritmos Genéticos en Entornos Virtuales

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Joe Luis Bernal

Omar Ernesto Guevara Fragoso

Tutor: Lic. Yuniesky Coca Bergolla

Ciudad de la Habana

Junio, 2007

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____ _____

Firma del Autor
(Joe Luis Bernal)

Firma del Autor
(Omar Ernesto Guevara Fragoso)

_____

Firma del Tutor
(Lic. Yuniesky Coca Bergolla)

Datos de Contacto

Lic. Yuniesky Coca Bergolla

E-mail: ycoca@uci.cu

Telf: 8372474

Graduado de Ciencias de la Computación en la Universidad Central de Las Villas en el año 2003. Profesor de la Universidad de las Ciencias Informáticas desde ese mismo año. Es Jefe de Dpto. de las asignaturas de la especialidad de la Facultad 5 “Entornos Virtuales”, además de ser líder del Grupo de Desarrollo de Elementos Virtuales Inteligentes. Obtuvo el Sello Forjadores del Futuro en el 2007.

Agradecimientos

… al Licenciado Yuniesky Coca Bergolla por la gran ayuda que nos a brindado en todo momento durante el desarrollo de esta tesis.

… a la todos los profesores que han tenido que ver con nuestra formación en la Universidad de las Ciencias Informáticas.

…a todas las personas que nos ayudaron en la realización de este trabajo, sin ellos no hubiéramos podido lograrlo con la calidad requerida.

Dedicatoria

… a todos nuestros familiares, en especial a nuestros padres que han confiado en nosotros desde el principio y hasta hoy.

…a todos nuestros amigos y compañeros que han compartido estos 5 años con nosotros.

Resumen

En los proyectos que desarrollan aplicaciones de Realidad Virtual en la UCI, no se están aplicando las técnicas más adecuadas de Inteligencia Artificial. Para resolver este problema se plantea la propuesta de aplicar una de las técnicas de Programación Evolutiva llamada “Algoritmo Genético”, en los módulos de Inteligencia Artificial de dichas aplicaciones, mediante tres tareas principales. Una de ellas es la explicación en detalle del funcionamiento de los Algoritmos Genéticos, se definen los mismos y se explican sus principales métodos, Selección, Cruzamiento y Mutación.

La segunda tarea consiste en la realización de una propuesta concreta de cómo aplicar dicha técnica y la implementación de una estructura Algoritmo Genético, para ello se escoge un caso de estudio a partir del cual se define y se aplica la estructura.

Por su parte en la tercera tarea se exponen y argumentan las principales ventajas y limitaciones de haber aplicado un Algoritmo Genético en el caso de estudio. También se realiza una comparación de los Algoritmos Genéticos con otras técnicas de Programación Evolutiva que pudieron usarse, además se exploran otras áreas donde puede ser utilizada la estructura definida.

Concluyendo que los Algoritmos Genéticos son una técnica viable para lograr que la Inteligencia Artificial de las aplicaciones de Realidad Virtual funcione a la altura de los requerimientos actuales.

Palabras claves: Entornos Virtuales, Inteligencia Artificial, Algoritmo Genético.

Índice

INTRODUCCIÓN 1

CAPÍTULO 1 5

1.1 SURGIMIENTO DE LOS AG 5
1.2 EVOLUCIÓN DE LOS AG..... 6
1.3 ESTRUCTURA Y FUNCIONAMIENTO..... 10
1.3.1 Definición de Algoritmo Genético (AG)..... 12
1.3.2 Métodos de Representación..... 13
1.3.3 Selección..... 14
1.3.4 Cruzamiento y Mutación..... 16
1.4- APLICACIONES DE LOS AG. 18
1.5 APLICACIÓN DE LOS AG EN JUEGOS. 22

CAPÍTULO 2 24

2.1 CASO DE ESTUDIO. 24
2.2 DEFINICIÓN DE LA ESTRUCTURA "AG"..... 27
2.3 SELECCIÓN NATURAL..... 30
2.4 CRUZAMIENTO Y MUTACIÓN..... 32

CAPÍTULO 3 38

3.1 COMPARACIÓN CON OTRAS TÉCNICAS. 38
3.2 VENTAJAS DE LOS AG..... 40
3.3 LIMITACIONES DE LOS AG 43
3.4 ÁREAS DONDE SE PUEDE APLICAR LA PROPUESTA 45

CONCLUSIONES.....47

RECOMENDACIONES 51

REFERENCIAS BIBLIOGRÁFICAS 52

BIBLIOGRAFÍA 55

ANEXOS 56

GLOSARIO DE TÉRMINOS.....66

Introducción

En la actualidad, el desarrollo científico está produciendo beneficios prácticos en un campo muy novedoso, “la informática”. Uno de los mismos proviene de una estrategia de programación que consiste en procedimientos de búsquedas basados en teorías evolutivas, una de ellas son los “Algoritmos Genéticos (AG)”. Son de mucha utilidad para el diseño óptimo de armaduras planas y tridimensionales. Teniendo una aplicación considerable en los campos de optimización, búsqueda y aprendizaje de máquinas.

Esta técnica no siempre tuvo el nivel de perfección actual, sino que tuvo sus antecedentes desde la década de 1960, en aquella época el científico John Holland y un grupo de estudiantes universitarios de Michigan investigaban como hacer sistemas inteligentes. A partir de los resultados de sus investigaciones se percató de que el aprendizaje no solo podía efectuarse mediante la adaptación de un solo organismo, sino que la adaptación evolutiva de varias generaciones de una especie también podía ser una manera eficiente de alcanzar dicho propósito, enfocándose entonces en la noción Darwiniana de la evolución para realizar su trabajo, bajo el concepto de que “el más apto sobrevive”.

A medida que ha ido avanzando la ciencia, se han ido introduciendo elementos en el mundo de la optimización. La inserción de la informática en la sociedad ha estado generando una gran demanda del desarrollo de aplicaciones con un nivel de inteligencia y realismo cada vez más avanzado, las aplicaciones de realidad virtual son un ejemplo tangible de ellas y es aquí donde juegan un papel primordial los AG.

Desde los inicios de la UCI la facultad 5 trabaja en el área de los simuladores, y aún más reciente en el área de los videos juegos, pero la inexistencia de técnicas avanzadas de Inteligencia Artificial (IA) en dichos proyectos han limitado el nivel de realismo e inteligencia de los mismos.

Es por ello que el **problema científico** se enfoca en:
¿Cómo lograr mayor nivel de realismo e inteligencia en las aplicaciones de realidad virtual que desarrollan los proyectos de la facultad 5?

El **objeto de estudio** se circunscribe en el proceso de desarrollo de módulos de Inteligencia Artificial de las aplicaciones de realidad virtual.

Con el fin de resolver del problema científico se trazó el siguiente **objetivo**:
Proponer una estructura AG que mejore el nivel de realismo e inteligencia en las aplicaciones de realidad virtual.

En esta investigación se trabajará en la incorporación de la técnica de AG en los módulos de IA, lo cual corresponde al **campo de acción** de la investigación.

Para darle solución a este problema, se desarrollaron las siguientes **tareas investigativas**.

- Una búsqueda bibliográfica sobre todo lo relacionado con el tema de AG.
- Exposición de la historia y antecedentes de los AG.
- Explicación en detalle del funcionamiento de los AG.
- Realización de propuestas concretas de cómo aplicar los AG en los proyectos la facultad 5 de la UCI.
- Argumentación de las ventajas, desventajas y posibles resultados al aplicar nuestras propuestas.

Este trabajo está dirigido además, a defender la idea de que los AG son una técnica viable en aplicaciones de realidad virtual.

La investigación está estructurada en tres capítulos, en el 1ro se exponen los elementos que describen la fundamentación teórica de la misma a través de distintos sub-epígrafes con el objetivo de tocar elementos como:

-Surgimiento y evolución de los AG

-Estructura y funcionamiento de los mismos:

- Definición
- Métodos de representación
- Selección
- Mutación y Cruzamiento
- Ejemplos de aplicaciones de AG de manera general
- Ejemplo de aplicaciones de AG en juegos y simuladores.

El capítulo 2 expone el desarrollo y propuesta de la estructura AG. Para ello cuenta con sub-epígrafes que describen los siguientes elementos:

-Explicación de un caso de estudio.

-Definición de la estructura “AG”.

-Proceso de selección natural en la estructura.

- Proceso de cruzamiento y mutación en la estructura.

-Reposición y Evolución.

Por su parte el capítulo 3 mostrará un mayor acercamiento al objetivo propuesto y cuenta con los siguientes sub-epígrafes.

- Comparación con otras técnicas.

-Ventajas.

-Limitaciones.

-Áreas donde se puede aplicar la propuesta.

Para darle cumplimiento a las tareas se guiará la investigación en los marcos de los métodos científicos de investigación “teóricos y empíricos”. Dentro de los métodos teóricos se utilizará el método “Analítico-sintético”, con el mismo se podrá analizar cada uno de los elementos de manera independiente, posibilitando una mayor capacidad de comprensión y de síntesis sobre los aspectos más importantes. Por su parte la utilización del método “Análisis histórico-lógico” brindará la posibilidad de analizar toda la evolución del problema que se estará estudiando.

Uno de los métodos empíricos que será referenciado es la “Observación” este método permite adquirir información necesaria y puede utilizarse en cualquiera de las fases de la investigación, además ofrece un gran acercamiento a la realidad y permite ver la posible solución del problema desde diferentes ángulos.

Capítulo 1

Título: Fundamentación Teórica

En este capítulo se abordará una reseña histórica del surgimiento y evolución de los AG, algunas de las aplicaciones generales de los mismos en cualquier rama de las ciencias y fundamentalmente en la construcción de aplicaciones de realidad virtual. También será explicado el funcionamiento de los algoritmos genéticos, arribando a la definición formal de los mismos.

1.1 Surgimiento de los AG

Un sueño de todos los hombres desde que surgieron las primeras máquinas ha sido que estas pudieran aprender por si solas. En este empeño hubo un hombre que lo podemos llamar el padre o primer precursor que logró sentar las bases dando criterios muy acertados, su nombre es John Holland, él se preguntaba cómo logra la naturaleza crear seres cada vez más perfectos, lo cual que se lleva a cabo a base de interacciones locales entre individuos y lo que les rodea.

No sabía la respuesta, pero tenía una noción de como hallarla: haciendo modelos de la naturaleza a menor escala, que tuvieran algunas de sus características, y ver cómo funcionaban, para luego extrapolar sus conclusiones a la totalidad, así en la década de los cincuenta tuvo sus primeras interacciones con el mundo de las computadoras e intentó comenzar a desarrollar sus teorías pero no se encontraba en un medio propicio porque le faltaban intelectuales capaces, que lo siguieran y ayudaran , solo hasta los finales de 1960 fue que pudo disponer del medio adecuado para su propósito .

Hace aproximadamente cuatro décadas en la Universidad de Michigan en Ann Arbor dentro del grupo “Logic of Computers”, sus ideas comenzaron a desarrollarse y a tener resultados. En un libro escrito por el biólogo evolucionista, R. A. Fisher, titulado “La teoría genética de la selección natural”, comenzó a descubrir los medios de llevar a cabo sus propósitos de

comprensión de la naturaleza, de ese libro aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fin determinado. A estas primeras técnicas inventadas por Holland se le llamó originalmente "Planes Reproductivos", pero se hizo popular bajo el nombre "Algoritmo Genético". Una definición bastante completa de un algoritmo genético es la propuesta por [Koza, 1992]:

“Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud”.

Cuando Holland se enfrentó a los algoritmos genéticos los objetivos de su investigación fueron dos: imitar los procesos adaptativos de los sistemas naturales y diseñar sistemas artificiales (normalmente programas) que retuvieran los mecanismos importantes de los sistemas naturales.

1.2 Evolución de los AG

Para hacer un recorrido a grosso modo por la evolución y desarrollo a lo largo de estos años de los algoritmos genéticos se puede decir que el marco para las primeras ideas sobre algoritmos evolutivos fue en la década de 1950. Pero ya en 1948 Alan Turing había sugerido la conexión entre ambos aspectos, proponiendo desarrollar programas automodificables capaces de jugar ajedrez y simular otras actividades inteligentes desarrolladas por los seres humanos, utilizando técnicas evolutivas. Los detalles de las ideas y las propuestas de

Turing pueden consultarse en la sección correspondiente en la Enciclopedia de Filosofía de la Universidad de Stamford [Hodges, 2002].

El origen de lo que se conoce como “Computación Evolutiva” se debe buscar en su razón de ser: los conocimientos sobre evolución se pueden aplicar en la resolución de problemas de optimización. Fue en el período de 1950 a 1960 cuando varios científicos, de modo independiente, comenzaron a estudiar los sistemas evolutivos, guiados por la intuición de que se podrían emplear como herramienta en problemas de optimización en ingeniería. La idea era evolucionar una población de candidatos para solucionar un problema conocido, utilizando operadores inspirados en la selección natural y la variación genética natural.

Fue Rechenberg quien de 1965 a 1973 introdujo las “Estrategias Evolutivas”, método que empleó para optimizar parámetros reales para ciertos dispositivos. La misma idea fue desarrollada posteriormente por [Schwefel, 1975].

El campo de las estrategias evolutivas ha permanecido como un área de investigación activa, cuyo desarrollo se produce de modo independiente al de los algoritmos genéticos, aunque recientemente se ha visto como las dos comunidades han comenzado ha colaborar. Fogel, Owens y Walsh, fueron los creadores de la “Programación Evolutiva”, una técnica en la cual las soluciones candidatas a tareas determinadas, eran representadas por máquinas de estados finitos, cuyos diagramas de transición de estados evolucionaban mediante una mutación aleatoria, seleccionándose el que más se aproximara.

Una formulación más amplia de la programación evolutiva, es un campo de investigación que también continúa en activo. Estas tres áreas, estrategias evolutivas, algoritmos genéticos, y programación evolutiva, son las que forman la columna vertebral de la Computación Evolutiva, y de ellas parten los caminos hacia todos los campos de investigación inspirados en los conocimientos sobre evolución.

Pero es otro científico el considerado creador de los Algoritmos Genéticos: John Holland, que los desarrolló junto a sus alumnos y colegas, durante las décadas de 1960 y 1970. En contraste con las estrategias evolutivas y la programación evolutiva, el propósito original de Holland no era diseñar algoritmos para resolver problemas concretos, sino estudiar el fenómeno de la adaptación tal y como ocurre en la naturaleza, y desarrollar vías de extrapolar esos mecanismos de adaptación natural a los sistemas computacionales.

Holland escribe en 1975 su libro "Adaptación en Sistemas Naturales y Artificiales" que presentaba el algoritmo genético como una abstracción de la evolución biológica. La mayor innovación de Holland fue la de introducir un algoritmo basado en poblaciones con cruces, mutaciones e inversiones. Holland fue el primero en intentar colocar la computación evolutiva sobre una base teórica firme [Holland, 1975]. Hasta hace poco, esta base teórica, fundamentada en la noción de esquemas, fue la estructura sobre la que se edificaron la mayoría de los trabajos teóricos sobre algoritmos genéticos en las décadas siguientes.

En la década de 1970, David Goldberg**,** quien es hoy en día una autoridad en el tema de los algoritmos genéticos, conoció a Holland y se convirtió en su estudiante. Golberg era un ingeniero industrial trabajando en diseño de pipelines, y fue uno de los primeros que trató de aplicar los algoritmos genéticos a problemas industriales. Aunque Holland trató de quitarle esa idea, porque pensaba que el problema era excesivamente complicado como para aplicarle algoritmos genéticos, Goldberg lo hizo, escribiendo un algoritmo genético en una PC personal.

Estas y otras aplicaciones creadas por estudiantes de Holland convirtieron a los algoritmos genéticos en un campo con base suficiente aceptado para celebrar la primera conferencia en 1985, ICGA´85. Tal conferencia se sigue celebrando cada 2 años.

John von Neumann también se interesó en la combinación de técnicas evolutivas y de la computación, en especial en el caso de los autómatas celulares. Sobre el final de su vida se encontraba trabajando en el área, lo cual aparece explícito en su texto inconcluso “Teoría de Autómatas Auto Replicables” [Von Neumann, 1966] que sería editado luego de la muerte del autor por su colega A. Burke.

Von Neumann propuso mecanismos evolutivos basados en la programación para implementar autómatas. Además, conjeturó sobre el comportamiento de poblaciones de autómatas capaces de abordar problemas complejos comunicándose entre sí.

El matemático y biólogo Nils Barricelli utilizó la infraestructura montada por Von Neumann en el Instituto para Estudios Avanzados en la Universidad de Princeton, para llevar a cabo la primera simulación de “Vida Artificial” basada en principios evolutivos, en el período comprendido entre los años 1953 y 1956.

Su trabajo consistió en diseñar un modelo computacional para la evolución darwiniana e investigar el rol de la simbiogénesis en el origen y desarrollo de la vida. La simbiogénesis considera a los organismos vivientes como una sucesión de asociaciones simbióticas entre formas sencillas de vida. Los conceptos de la simbiogénesis y su relación con la computación evolutiva pueden consultarse en los trabajos recientes de [Watson y Pollack 1999, 2000]

De acuerdo a [Golberg, 1989] y De Jong, Bledsoe y Bremermann surgieron las idea de codificación binaria, y el uso de un valor de aptitud en la aplicación de los algoritmos evolutivos para la resolución de problemas de optimización numérica. Bledsoe propuso el esquema de generar individuos, aplicar mutaciones y seleccionar los que produjeran mejores resultados y Bremermann lo extendió para considerar poblaciones de individuos.

Bremermann notó la importancia del operador de mutación para evitar el estancamiento del proceso de búsqueda en mínimos locales del problema. El primer resultado teórico sobre la operativa de los algoritmos evolutivos fue alcanzado por Bremermann al determinar la probabilidad de mutación óptima para resolver problemas linealmente separables.

Aunque la mayoría de sus conceptos se utilizan en la actualidad, y de hecho Bremermann es aceptado como el "creador" de los algoritmos genéticos [Fogel y Anderson, 2000], en su trabajo obtuvo algunos resultados decepcionantes al aplicarlos para optimizar funciones lineales y convexas.

En estos últimos años se ha generado una amplia interacción entre los investigadores de varios métodos de computación evolutiva, rompiéndose las fronteras entre algoritmos genéticos, estrategias evolutivas y programación evolutiva. Como consecuencia, en la actualidad, el término “algoritmo genético” se utiliza para designar un concepto mucho más amplio del que concibió Holland. Por lo tanto se resume que es una técnica que está en constante desarrollo, evolución y aplicación como su propio nombre lo dice.

1.3 Estructura y funcionamiento

A grandes rasgos serán analizados los AG como una técnica de búsqueda que se basa en mecanismos de supervivencia planteados en la teoría de evolución de Darwin. Los algoritmos genéticos establecen una relación entre el conjunto de soluciones de un problema, llamado fenotipo, y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena, generalmente binaria, llamada cromosoma.

Los símbolos que forman la cadena son llamados genes. Los cromosomas evolucionan a través de iteraciones, llamadas generaciones. En cada generación, los cromosomas son evaluados usando alguna medida de aptitud.

Las siguientes generaciones (nuevos cromosomas), son llamadas descendencia. El siguiente segmento de seudo-código simula dicha técnica. [Buckles & Petry, 1992]

Generar población inicial, G (0);
Evaluar G(0);
t:=0;
Repetir
t:=t+1;
Generar G(t) usando G(t-1);
Evaluar G(t);
Hasta encontrar una solución;

Inicialmente la población se genera de manera aleatoria, y estará compuesta por un conjunto de cromosomas, pudieran ser caracteres que representen las posibles soluciones de un problema. En caso de no hacerlo de forma aleatoria, es importante garantizar que dentro de la población inicial, se tenga una representación de la mayor parte de la población posible o al menos evitar la convergencia prematura.

A cada miembro de la población se le aplicará una función de aptitud con el propósito de saber que tan buena es la solución que se está codificando, o sea para ver su comportamiento. Posteriormente se procede a hacer la selección de los cromosomas que van a cruzarse en la próxima generación, como es lógico se escogerán los mejores.
El cruzamiento es el principal operador genético, representa la reproducción sexual, opera sobre dos cromosomas a la vez para generar dos descendientes donde se combinan las características de ambos cromosomas padres.

De este cruzamiento surgirá el nuevo conjunto de cromosomas a los que se le aplicará el proceso anterior. También existe un operador de mutación que cambiará de manera aleatoria un alelo (“un bit de cadena representativa”) de un cromosoma. Esto garantiza la inserción de nuevo material cromosómico y la interconexión total del espacio de búsqueda.

El AG se ejecuta una cantidad determinada de generaciones o hasta que se estabilice la población (“cuando todos o la mayoría de los individuos tengan la misma actitud”).

1.3.1 Definición de Algoritmo Genético (AG).

Según Watson y Pollack [Watson y Pollack, 1999] [Watson y Pollack, 2000] los Algoritmos genéticos son la técnica informática que se basa en el funcionamiento de la evolución natural para resolver determinadas situaciones, buscando, combinando y optimizando soluciones y produciendo innovación. Operan mediante simulación por una computadora, combinando aleatoriamente soluciones posibles a un problema, para producir diversidad, y seleccionando las más aptas, que se vuelven a recombinar y evaluar en sucesivas generaciones. La aptitud media de cada generación va creciendo, y al final, se obtiene una solución si no óptima, convergente con lo óptimo.

Otra de las definiciones que le dan a los AG es la de un método adaptativo que puede usarse para resolver problemas de búsqueda y optimización. Está basado en el proceso genético de los organismos vivos. Un algoritmo genético consiste en una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuales de ellos deben generar descendencia para la nueva generación.

Versiones más complejas de algoritmos genéticos generan un ciclo iterativo que directamente toma a la especie (el total de los ejemplares) y crea una nueva generación que reemplaza a la antigua una cantidad de veces

determinada por su propio diseño. Una de sus características principales es la de ir perfeccionando su propia heurística en el proceso de ejecución, por lo que no requiere largos períodos de entrenamiento especializado por parte del ser humano. [Srinivas y Patnaik, 1998] [Srinivas y Patnaik, 2000]

En este trabajo se tomará como definición de Algoritmos Genéticos:
A la técnica de programación que imita a la evolución biológica como estrategia para resolver problemas computacionales.

Dado un problema específico a resolver, la entrada del AG seria un conjunto de soluciones potenciales a ese problema, codificadas de alguna manera, y una métrica llamada función de aptitud que permite evaluar cuantitativamente a cada una de ellas. Estas pueden ser soluciones que ya se sabe que funcionan, con el objetivo de que el AG las mejore, pero se suelen generar aleatoriamente.

Se definirán entonces como funciones principales de los AG:
Hallar de qué parámetros depende el problema, codificarlos en un cromosoma, y aplicar los métodos de la evolución: selección y reproducción con intercambio de información y alteraciones que generan diversidad.

1.3.2 Métodos de Representación

Para que un algoritmo genético pueda resolver un problema, se necesita algún método que codifique las soluciones potenciales de forma que una computadora pueda procesarlas, algunos de ellos son:

Un enfoque común es codificar las soluciones como cadenas binarias: secuencias de 1s y 0s, donde el dígito de cada posición representa el valor de algún aspecto de la solución.

Otro método similar consiste en codificar las soluciones como enteros o números decimales, donde cada posición representa algún aspecto particular de la solución. Este método permite una mayor precisión y complejidad que el método comparativamente restringido de utilizar sólo números binarios

Un tercer método consiste en representar a los individuos de un AG como cadenas de letras, donde cada letra representa un aspecto específico de la solución.

La virtud de estos tres métodos es que facilitan la definición de operadores que causen los cambios aleatorios en las soluciones seleccionadas: cambiar un 0 por un 1 o viceversa, sumar o restar al valor de un número una cantidad elegida al azar, o cambiar una letra por otra.

1.3.3 Selección

La Selección Natural consiste en determinar cuales individuos llegarán a reproducirse, por tanto los que tengan rasgos que los haga menos válidos para realizar sus tareas tendrán muy poca probabilidad de hacerlo y su patrimonio genético pudiera desaparecer para siempre.

Un algoritmo genético puede utilizar muchas técnicas diferentes para seleccionar a los individuos que deben copiarse hacia la siguiente generación, a continuación aparecen algunas de las más comunes. Algunas de estas técnicas son mutuamente excluyentes, pero otras pueden utilizarse en combinación, algo que se hace a menudo.
Selección elitista: se garantiza la selección de los miembros más aptos de cada generación. (La mayoría de los AGs no utilizan elitismo puro, sino que usan una forma modificada por la que el individuo mejor, o algunos de los mejores, son copiados hacia la siguiente generación en caso de que no surja nada mejor).

Selección proporcional a la aptitud: los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.

Selección por rueda de ruleta: una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores. (Conceptualmente, esto puede representarse como un juego de ruleta -cada individuo obtiene una sección de la ruleta, pero los más aptos obtienen

secciones mayores que las de los menos aptos. Luego la ruleta se hace girar, y en cada vez se elige al individuo que “posea" la sección en la que se pare la ruleta).

Selección escalada: al incrementarse la aptitud media de la población, la fuerza de la presión selectiva también aumenta y la función de aptitud se hace más discriminadora. Este método puede ser útil para seleccionar más tarde, cuando todos los individuos tengan una aptitud relativamente alta y sólo les distingan pequeñas diferencias en la aptitud.

Selección por torneo: se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.

Selección por rango: a cada individuo de la población se le asigna un rango numérico basado en su aptitud, y la selección se basa en este ranking, en lugar de las diferencias absolutas en aptitud. La ventaja de este método es que puede evitar que individuos muy aptos ganen dominancia al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable.

Selección generacional: la descendencia de los individuos seleccionados en cada generación se convierte en toda la siguiente generación. No se conservan individuos entre las generaciones.

Selección por estado estacionario: la descendencia de los individuos seleccionados en cada generación vuelven al acervo genético preexistente, reemplazando a algunos de los miembros menos aptos de la siguiente generación. Se conservan algunos individuos entre generaciones.

Selección jerárquica: los individuos atraviesan múltiples rondas de selección en cada generación. Las evaluaciones de los primeros niveles son más rápidas y menos discriminatorias, mientras que los que sobreviven hasta niveles más altos son evaluados más rigurosamente.

La ventaja de este método es que reduce el tiempo total de cálculo al utilizar una evaluación más rápida y menos selectiva para eliminar a la mayoría de los individuos que se muestran poco prometedores, y sometiendo a una evaluación de aptitud más rigurosa y computacionalmente más costosa sólo a los que sobreviven a esta prueba inicial.

1.3.4 Cruzamiento y Mutación

Una vez que han sido seleccionados los mejores individuos, éstos deben ser utilizados para crear la siguiente generación con la esperanza de mejorar su aptitud. Existen dos estrategias básicas para lograr este propósito.

La primera estrategia se llama cruzamiento, el mismo puede realizarse de diferentes formas. Básicamente en cualquiera de ellas lo que se hace es elegir a dos individuos para que intercambien segmentos de su código y así crear una descendencia artificial donde los individuos sean una combinación de sus padres, simulándose el proceso convencional de la recombinación que realizan los cromosomas durante la reproducción sexual.

Una de las formas es el cruzamiento por un punto, es decir se establece un punto de intercambio en un lugar aleatorio del genoma de los dos individuos, y uno de los individuos aporta todo su código anterior a ese punto y el otro individuo también lo hace pero a partir de ese punto para producir una descendencia y un cruzamiento uniforme, en el que el valor de una posición dada en el genoma del individuo descendiente corresponda al valor en esa posición del genoma de uno de los dos padres, teniendo ambos la misma probabilidad de que ocurra.

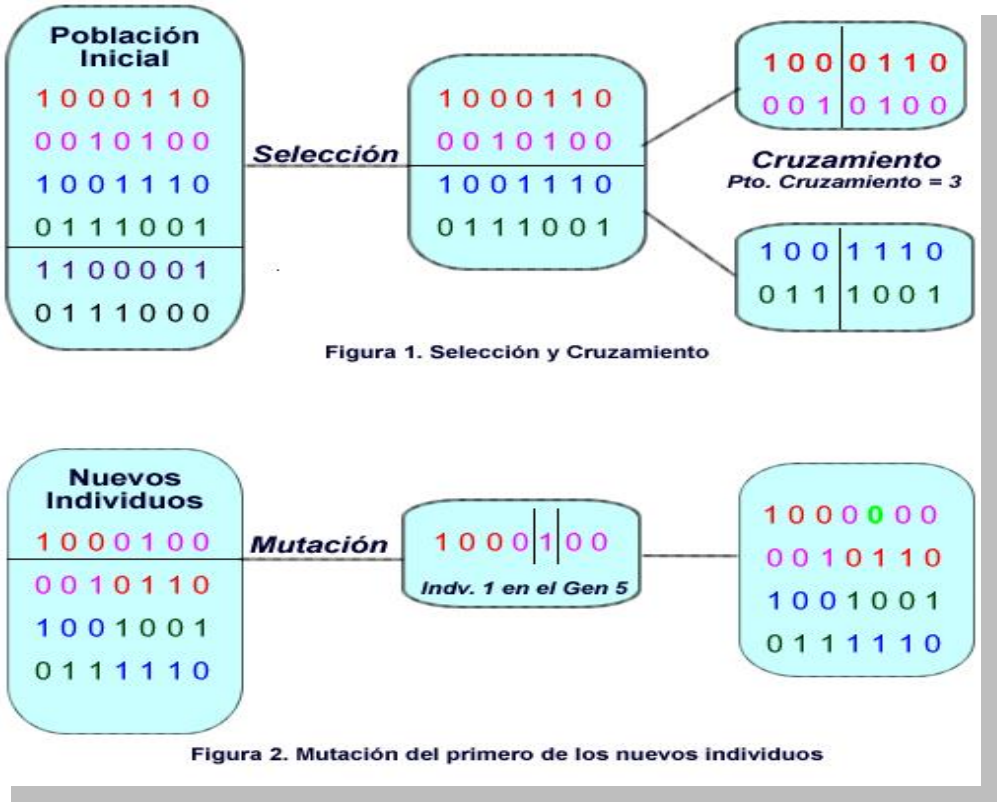
También puede hacerse el cruzamiento teniendo en cuenta más de un punto de cruce. La implementación de esta otra vía de cruzamiento, garantiza que el cruce basado en dos puntos, representa una mejora, mientras que añadir más puntos de cruce no beneficia el comportamiento del AG.

La principal ventaja de hacerlo de esta manera, es decir teniendo más de un punto de cruce, radica en que el espacio de búsqueda puede ser explorado

más fácilmente y la principal desventaja sería que aumenta la probabilidad de ruptura de buenos esquemas.

La segunda y más sencilla estrategia de alterar genéticamente a los individuos se llama mutación, la cual se encarga de cambiar de modo aleatorio, los valores del alelo en algunas localizaciones del cromosoma, es decir de crear una alteración del código genético de un individuo y por último, la inversión, invierte el orden de una sección contigua del cromosoma, recolocando por tanto el orden en el que se almacenan los genes.

A continuación aparece un diagrama que ilustra el resultado de las dos estrategias mencionadas anteriormente en los individuos de una población de cadenas de 7 bits.



En la primera figura se muestra una población de seis individuos donde son seleccionados de manera aleatoria los cuatro primeros para cruzarse, para llevar a cabo el cruzamiento de dicha selección se hace formando dos pares,

uno del primer individuo con el segundo y otro del tercero con el cuarto y se toma un punto de cruzamiento igual a tres.

En la segunda figura aparecen los nuevos individuos resultantes de la primera operación, de ellos se selecciona el primero de manera aleatoria también para alterar su genotipo mediante la mutación. Para ellos se selecciona el gen cinco de manera aleatoria de dicho individuo para alterarlo genéticamente.

1.4- Aplicaciones de los AG.

En este epígrafe serán mencionados a grandes rasgos varios ejemplos donde se aplican algoritmos genéticos en una extensa gama de problemas en casi cualquier campo de la vida cotidiana. Lo que demuestra su importancia y usabilidad en la actualidad y en el futuro.

Acústica

[Sato ,2002] Se utilizaron algoritmos genéticos para diseñar una sala de conciertos con propiedades acústicas óptimas, maximizando la calidad del sonido para la audiencia, para el director y para los músicos del escenario. Esta tarea implica la optimización simultánea de múltiples variables.

[Tang, 1996] analizan los usos de los algoritmos genéticos en el campo de la acústica y el procesamiento de señales. Un área de interés particular incluye el uso de AGs para diseñar sistemas de Control Activo de Ruido (CAR), que eliminan el sonido no deseado produciendo ondas sonoras que interfieren destructivamente con el ruido. Esto es un problema de múltiples objetivos que requiere el control y la colocación precisa de múltiples altavoces.

Ingeniería aeroespacial

[Obayashi, 2000] Fue un algoritmo genético de múltiples objetivos para diseñar la forma del ala de un avión supersónico. Hay tres consideraciones principales que determinan la configuración del ala -minimizar la resistencia aerodinámica a velocidades de vuelo supersónicas, minimizar la resistencia a velocidades

subsónicas y minimizar la carga aerodinámica (la fuerza que tiende a doblar el ala).

Estos objetivos son mutuamente exclusivos, y optimizarlos todos simultáneamente requiere realizar contrapartidas. En un artículo posterior [Sasaki, 2001], los autores repitieron el experimento añadiendo un cuarto objetivo, minimizar el momento de torsión (un conocido problema en los diseños de alas flecha en el vuelo supersónico). También se añadieron puntos de control adicionales para el grosor al conjunto de variables de diseño.

Astronomía y astrofísica

[Charbonneau, 1995] sugiere la utilidad de los AGs para problemas de astrofísica, aplicándolos a tres problemas de ejemplo: obtener la curva de rotación de una galaxia basándose en las velocidades rotacionales observadas de sus componentes, determinar el periodo de pulsación de una estrella variable basándose en series de datos temporales, y sacar los valores de los parámetros críticos de un modelo magneto hidrodinámico del viento solar. Son tres difíciles problemas no lineales y multidimensionales.

Ingeniería eléctrica

[Altshuler y Linden ,1997] Fue utilizado un algoritmo genético para evolucionar antenas de alambre con propiedades especificadas a priori. Los autores señalan que el diseño de tales antenas es un proceso impreciso, comenzando con las propiedades deseadas y luego determinando la forma de la antena mediante conjeturas, intuición, experiencia, ecuaciones aproximadas o estudios empíricos.

Esta técnica requiere mucho tiempo, a menudo no produce resultados óptimos y tiende a funcionar bien sólo con diseños simétricos y relativamente simples. En contraste, con el método del algoritmo genético, el ingeniero especifica las propiedades electromagnéticas de la antena, y el AG sintetiza automáticamente una configuración que sirva.

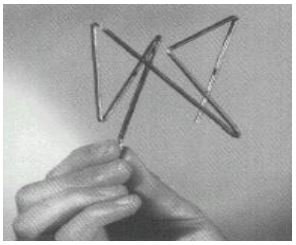


Figure: Una antena genética de alambre doblado (de Altshuler y Linden 1997, figura 1).

Geofísica

[Sambridge y Gallaguer, 1993] Fue creado un algoritmo genético para los hipocentros de los terremotos basándose en datos sismológicos. (El hipocentro es el punto bajo la superficie terrestre en el que se origina un terremoto. El epicentro es el punto de la superficie directamente encima del hipocentro). Los autores concluyen que su algoritmo genético es eficiente para una verdadera optimización global y una herramienta nueva y poderosa para realizar búsquedas robustas de hipocentros.

Matemáticas y algoritmia

[Haupt y Haupt ,1998] Se describe el uso de AGs para resolver ecuaciones de derivadas parciales no lineales de alto orden, normalmente encontrando los valores para los que las ecuaciones se hacen cero, y dan como ejemplo una solución casi perfecta para los coeficientes de la ecuación de quinto orden conocido como Super Korteweg-de Vries.

Ejército y cumplimiento de la ley

[Naik ,1996] informa de un uso interesante de los AGs en el cumplimiento de la ley, describiendo el software “FacePrint'”, un proyecto para ayudar a los testigos a identificar y describir a los sospechosos criminales. FacePrints utiliza un algoritmo genético que evoluciona dibujos de caras basándose en bases de datos de cientos de características individuales que pueden combinarse de infinitas maneras.

El programa muestra a los testigos imágenes de rostros generadas aleatoriamente, y éstos escogen las que más se parecen a la persona que vieron; las caras seleccionadas mutan y se combinan para generar nuevas combinaciones de características, y el proceso se repite hasta que emerge un retrato preciso del rostro del sospechoso.

Reconocimiento de patrones y exploración de datos.

[Hughes y Leyland, 2000] También aplicaron AGs multi-objetivo a la tarea de clasificar objetivos. Los datos de una sección transversal radar de alta resolución necesitan enormes cantidades de espacio de almacenamiento en disco, y producir un modelo realista de la fuente a partir de los datos es muy costoso computacionalmente.

Por otra parte, el método basado en el AG de los autores demostró ser muy exitoso, produciendo un modelo tan bueno como el del método iterativo tradicional, pero reduciendo el gasto computacional y las necesidades de almacenamiento hasta el punto de que era factible generar buenos modelos en una PC.

Diseño de rutas y horarios

[Naik ,1996] Los organizadores de los Juegos Paraolímpicos de 1992 utilizaron un AG para diseñar los horarios de los eventos. Como informa [Petzinger, 1995], John Deere & Co. ha utilizado AGs para generar los programas de montaje para una planta de Moline, Illinois, que fabrica plantadoras y otras maquinarias agrícolas pesadas.

Generación de melodías

[Degazio, 1999] propone el empleo de algoritmos genéticos para "la evolución de organismos musicales", es decir melodías que van evolucionando y que el compositor va evaluando hasta encontrar aquellas que sean más compatibles. El software permite la creación de genes o individuos que son básicamente líneas melódicas que irán evolucionando de acuerdo a la evaluación del

usuario, de tal manera que las sucesivas generaciones consistirán en individuos más aptos para los fines del compositor.

1.5 Aplicación de los AG en juegos.

Debido a la gran magnitud económica en nuestros días del mercado de los videojuegos, la optimización de la IA de los mismos es algo medular y serán enunciados ejemplos de esto mediante algoritmos genéticos.

Una de las demostraciones más novedosas y persuasivas de la potencia de los algoritmos genéticos la presentaron Chellapilla y Fogel en el año 2000, [Chellapilla y Fogel, 2000] que utilizaron un AG para evolucionar redes neuronales que pudieran jugar a las damas.

Los autores afirman que una de las mayores dificultades en este tipo de problemas relacionados con estrategias es el problema de la asignación de crédito, es decir ¿cómo escribir una función de aptitud? Se ha creído ampliamente que los criterios simples de ganar, perder o empatar no proporcionan la suficiente información para que un algoritmo genético averigüe qué constituye el buen juego.

Esnuka

Esnuka es un juego de billar en el que el color de las bolas depende de los estados evolutivos en función de las carambolas logradas, de acuerdo con los deducidos de la Teoría General de la Evolución Condicionada de la Vida. En Esnuka no hacían falta tantas variables aleatorias en los procesos de simulación de la evolución porque no producía errores en la expresión ni en la medición y la evolución se establece en un porcentaje constante.

Todas estas gráficas de correlación y regresión múltiple corresponden al Modelo Global de herencia multifuncional incluyendo las limitaciones funcionales derivadas de que existen problemas genéticos. Por supuesto, para lograr un efecto óptico satisfactorio de las variables cuantitativas, se han escogido aquellas gráficas de la simulación de procesos en las que W más se ajusta a una de las H o variables observadas de los hijos.

El Prisionero

Se aplican también en el juego el prisionero, la situación de dos prisioneros mantenidos en celdas separadas. El beneficio o castigo que obtendrá cada prisionero dependerá de la forma en que los dos actúen. Cada prisionero tiene solo dos formas de actuar: colaborar con el otro prisionero capturado o traicionado revelando información. Los algoritmos implementados en el simulador contemplan estrategias de carácter probabilístico. En lugar de conducir a acción única, se buscan valores adecuados para la probabilidades traicionar colaborar ante cada situación.

Ahora se postula la generalización. Si el otro jugador traiciona en la anterior jugada, la siguiente movida debe ser traicionar con una probabilidad del 90%. Los valores de las probabilidades son encontrados por el algoritmo genético.

Se incorpora el concepto de reputación y se permite que cada estrategia incluya su uso optativo. La reputación es de conocimiento global y siempre refleja el comportamiento real que se ha observado para cada estrategia. Cada estrategia genética puede elegir utilizar la reputación en lugar de los valores de los movimientos anteriores. El momento en que se prefiere usar el conocimiento de la reputación del contrario es determinado de manera evolutiva.

Se añade el concepto de localidad, de fauna que se hace más probable la interacción con individuos de las cercanías respecto a los de las lejanías. Cada estrategia puede escoger el área de interacción abarcada controlando el parámetro sigma (dispersión) de una campana de Gauss alrededor de su ubicación. De esta fauna, la estrategia evolutiva termina escogiendo si se concentra en su localidad o se expande a las otras.

El algoritmo genético evoluciona combinando dos opciones principales: Dejar que cada estrategia genética compita con otra de la misma naturaleza y esperando que un comportamiento inteligente emerja de la interacción entre la estrategia evolutiva inicialmente de comportamiento aleatorio.

Capítulo 2

Título: Desarrollo y propuesta de la estructura ‘AG’.

En este capítulo se tratará de demostrar que es posible aplicar la técnica de AG en el desarrollo de aplicaciones de realidad virtual, para ello se tomará como ejemplo los juegos de combate entre varios jugadores, este tipo de juegos es un ejemplo clásico que demanda un alto nivel de inteligencia. Se explicará dicho elemento aplicándole un AG durante el desarrollo del capítulo.

2.1 Caso de estudio.

Cuando se está desarrollando un juego el elemento más imprevisible de su ambiente es el jugador. Es decir, de alguna manera el creador del juego debe poder predecir el comportamiento del jugador para crear a un adversario capaz de desafiarlo eficazmente. Predecir y explicar cada comportamiento posible del jugador puede tornarse una tarea realmente difícil.

La aplicación de un AG es una técnica adecuada para resolver dicha situación. Para ello básicamente se puebla el mundo del juego con muchas soluciones posibles y después el AG determina qué soluciones son las mejores. Por supuesto, las soluciones no serán siempre iguales para cada jugador, esa es una propiedad de los AG que hace que los juegos eleven su nivel de inteligencia.

En síntesis se pretende usar un AG para crear un adversario digno controlado por la PC que pueda combatir la habilidad de un jugador.
Ahora se procede a describir detalladamente el ejemplo, un juego multi-jugador. En dicho juego los jugadores pudieran tener diferentes cualidades. Esto significa que los jugadores controlados por la PC deben desafiar a otros que pudieran tener o no cualidades diferentes.

Se supone que uno de los tipos de jugadores que se pudiera elegir sería un guerrero, cuya cualidad principal es luchar con la fuerza bruta. Por supuesto, con esta clase el jugador se puede elegir diferentes armas, una espada, el hacha, etc. Esta clase de combatiente también podría usar algún tipo de armadura. Pudiera existir otro guerrero que su cualidad principal para combatir sea la magia, este también tendría su propio arsenal de armas y armaduras para resistir los ataques de sus oponentes.

Tomando como ejemplo estos posibles tipos de luchadores, partiendo de que el guerrero es controlado por la PC y que el luchador con armas mágicas lo controla uno de los jugadores, entonces una situación posible puede ser cuando el jugador ataca al luchador controlado por la PC con un arma mágica, este podría crear varias respuestas posibles a esta acción del jugador, atacar al jugador en respuesta, huir u ocultarse. Cada una de estas situaciones pudiera estar asignada como respuesta a un cromosoma.

A continuación se enumeran algunos de los panoramas posibles que serán tratados en este ejemplo hipotético:

Posibles Escenarios

#define Atacado_por_Grupo 0

#define Interectuar_con_Curador 1

#define Atacado_con_Espada 2

#define Atacado_con_Mazo 3

#define Atacado_con_Disparo 4

#define Atacado_con_Magia 5

#define Atacante_con_ArmaduraMetal 6

#define Atacante_con_ArmaduraCuero 7

#define Atacante _con_ArmaduraMagica 8

De esta manera quedarían descritas las situaciones en las cuales los miembros de la población podrán comportarse de manera diferente. Será utilizado un cromosoma para almacenar la respuesta de cada situación.

El cromosoma Atacado_por_Grupo indicará la respuesta del comportamiento al ser atacado por un grupo. El cromosoma Interectuar_con_Curador indica qué comportamiento debe ser utilizado cuando un curador del jugador está presente. Los cuatro cromosomas siguientes, Atacado_con_Espada, Atacado_con_Mazo, Atacado_con_Disparo y Atacado_con_Magia, determinan la respuesta a los diferentes tipos de armas que puede manejar un jugador atacante.

Los tres siguientes cromosomas Luchador_con_ArmaduraMetal, Luchador_con_ArmaduraCuero y Luchador_con_ArmaduraMagica, determinan la respuesta a los varios tipos de armadura protectora que el jugador atacante puede usar. El luchador controlado por la PC pudiera elegir en se arsenal un arma que sea más eficiente para algún tipo de armadura. Por ejemplo, una espada pudo ser más eficaz contra la armadura de cuero.

Un juego verdadero llevaría una lista mucho más cuidadosa de panoramas posibles, dado por su grado de complejidad.

A continuación se analizarán los posibles comportamientos que podría tener el luchador controlado por la PC ante los panoramas descritos anteriormente.

Posibles Comportamientos

#define Retirarse 0

#define Esconderse 1

#define Usar_ArmaduraMetal 2

#define Usar_ArmaduraMagica 3

#define Usar_ArmaduraCuero 4

```
#define Atacar_con_Espada        5
#define Atacar_con_Mazo          6
#define Atacar_con_Magia         7
```

2.2 Definición de la estructura “AG”.

Como propuesta de implementación de la estructura “AG”, será creada una clase base llamada “CCriatura”, utilizando C++ como lenguaje de programación, la misma contendrá los posibles comportamientos en los distintos panoramas. Es decir “CCriatura” contendría un arsenal de cromosomas.

Cada elemento del arsenal representará un comportamiento para el luchador controlado por la PC en un panorama específico. Utilizando un tamaño del arsenal de 12 porque anteriormente fueron mostrados 12 posibles panoramas. Esta clase también contendrá otros atributos significativos como son “Dano_Total_Causado”, “Dano_Total_Recibido” y “Aptitud”, posteriormente será explicada la funcionalidad de los mismos.

Como principales métodos contendrá su constructor, su destructor, los métodos para obtener los valores de sus atributos y actualizar los mismos, un método llamado “Get_Aptitud” y otro llamado “Crear_Individuo”.

Declaración de la clase “CCriatura”

```
#define cromosomas
#include "Math.h"
class CCriatura
{
private:
int list_Cromosomas[cromosomas];
float Dano_Total_Causado,Dano_Total_Recibido,Aptitud ;
public:
CCriatura ();
```

```
~CCriatura ();
float Get_Dano_Total_Causado();
float Get_Dano_Total_Recibido();
float Get_Aptitud();
void Set_Dano_Total_Causado(float dano);
void Set_Dano_Total_Recibido(float ptos);
void Set_Aptitud(float);
int Get_Cromosoma_Poss(int poss);
void Set_Cromosoma_Poss(int poss, int valor);
void Crear_Individuo();
};
```

Teniendo en cuenta que los mejores miembros serán los que representen un mejor desafío, es necesario cuantificarlo, para ello será guardado un total acumulativo de la cantidad de daño que puede realizar un individuo, así como el daño que ha recibido del resto de los jugadores.

Estas dos cuantías se almacenarán en los atributos “Dano_Total _Causado” y “Dano_Total _Recibido” respectivamente, de manera que el método “Get_Aptitud” antes mencionado, tendrá como funcionalidad calcular el valor de aptitud del individuo mediante el cociente “Dano_Total_Causado”/ “Dano_Total_Recibido”, en este ejemplo esta sería la definición de la “función aptitud” y dicho valor se almacenará en el atributo “Aptitud”.

Código del método “Get_Aptitud”

```
float CCriatura::Get_Aptitud()
{
return Aptitud=Dano_Total_Causado/Dano_Total_Recibido;
}
```

Por su parte el peculiar método “Crear_Individuo”, debe ser llamado desde el constructor de la clase para formar parte de la inicialización del objeto. Sin embargo, no se desea inicializar cada individuo con un sistema de constantes predefinidas, porque lo que se quiere es que la población sea tan diversa como sea posible. Una población que no sea diversa no será eficaz a la hora de encontrar la mejor solución.

Para crear a una población diversa deben asignarse los comportamientos al azar y esta es la función principal del método “Crear _ individuo”, teniendo en cuenta que no se deben asignar simplemente los comportamientos de manera aleatoria a los panoramas definidos, porque no todos los comportamientos son aplicables a todos los panoramas.

Fragmento de código del método “Crear _ individuo”

```
void CCriatura:: Crear _ individuo ()
{
Randomize();
switch ((rand()%5)+1){
case 1:
list_Cromosomas [Atacado_por_Grupo]= Retirarse;
break;
case 2:
list_Cromosomas [Atacado_por_Grupo]=Esconderse;
break;
case 3:
list_Cromosomas [Atacado_por_Grupo]=Atacar_con_Espada;
break;
case 4:
list_Cromosomas [Atacado_por_Grupo]=Atacar_con_Mazo;
break;
case 5:
list_Cromosomas [Atacado_por_Grupo]=Atacar_con_Magia;
break;
}
…….}
```

Describiendo lo que haría este método, para el primer caso “Atacado_por_Grupo”, para este cromosoma se eligen cinco comportamientos posibles y se le asigna uno de manera aleatoria: “Retirarse”, “Esconderse”, “Atacar_con_Espada”, “Atacar_con_Mazo” y “Atacar_con_Magia”. La idea es intentar determinarse si cualquiera de estas acciones proporciona una ventaja o una desventaja sensible cuando el luchador controlado por la PC es atacado por un grupo, en este ejemplo, el proceso de la evolución podría demostrar que “Retirarse” es el mejor comportamiento para él.

Para el resto de los cromosomas se realizaría un proceso similar al descrito anteriormente.

Ahora se creará la población, comenzando por la propuesta de la implementación de la estructura “AG” la misma contendrá como atributo principal una lista de tipo “CCriatura”, es decir dicha lista constituirá la población sobre la cual se realizarán todos los procesos del algoritmo genético, también contendrá otros atributos como son “cant_pob”, “cant_Selecc” y “cant_nuevos_ind”. Como métodos principales estarán: “Seleccion_Natural”, “Cruzamiento”, “Mutacion”,” Reposicion” y “Evolucion” además de su constructor y destructor.

Declaración de la Estructura “AG”

```
class AG
{
private:
CCriatura **Poblacion;
float Arr_Prob[1000];
int cant_pob,cant_Selecc,cant_nuevos_ind;
CCriatura** Seleccion_Natural(int n);
CCriatura** Cruzamiento(CCriatura** Lista);
CCriatura** Mutacion(CCriatura** Lista);
void Reposicion(CCriatura **Lista);
public:
double Get_Prob(int poss);
AG(CCriatura** Lista, int cant);
CCriatura* Get_Poblacion(int poss);
int Get_Cant_Ind();
void Evolucion(int mejores);
 };
```

2.3 Selección Natural

Una de las tareas principales del “AG” es seleccionar cuales miembros de la población son los que pasarán a realizar el cruzamiento, este objetivo es responsabilidad del método “Selección_Natural”, el mismo debe ser capaz de castigar a las malas soluciones, y de premiar a las buenas, de forma que estas últimas sean las que se propaguen con mayor rapidez, es decir el método le otorga una mayor probabilidad de ser seleccionados a los individuos que

tengan los mayores valores de aptitud, esto no quiere decir que se excluyen a los menos aptos, solo trata de otorgarle a cada cual la probabilidad que se merecen de ser seleccionados.

Implementación del método "Seleccion_Natural"

```
CCriatura**AG::Seleccion_Natural(int n){
Randomize();
CCriatura**Lista_Selec ;
Lista_Selec=new CCriatura*[n];
float Suma=0;
for (int i=0;i<cant_pob;i++ )
Suma=Suma+Poblacion[i]->Get_Aptitud();
Arr_Prob[0]=Poblacion[0]->Get_Aptitud()/Suma;
for (int i=1;i<cant_pob;i++ )
Arr_Prob[i]=Poblacion[i]->Get_Aptitud()/Suma+Arr_Prob[i-1];
for(int i=0;i<n;i++) {
Randomize();
int j=0;
float x=random(100);
x=x/100;
while(Arr_Prob[j]<x)
j++;
Lista_Selec[i]= Poblacion[j]; }
cant_Selecc=n;
return Lista_Selec;
}
```

En la anterior implementación, se le pasa como parámetro un valor que determina la cantidad de individuos que se desean seleccionar de la población, luego para determinar la probabilidad que tiene cada uno de ellos de ser seleccionado, se realiza un cociente entre el valor de "Aptitud" de cada individuo y el valor de la sumatoria de la "Aptitud" de todos los individuos de la población, con el objetivo de crear intervalos con estas probabilidades, las mismas se van sumando hasta llegar al valor uno.

Posteriormente se generará, una cantidad de veces igual a la cantidad de individuos que se desean seleccionar, un número aleatorio entre cero y uno y se seleccionará en todos los casos al individuo cuyo intervalo de probabilidad contiene al número generado. Cuando se termine el proceso se retornan todos los individuos que fueron seleccionados.

2.4 Cruzamiento y Mutación

Una vez que se ha determinado quienes son los individuos que se cruzarán, el paso siguiente es hacer el cruzamiento para lograr descendencia. El método encargado de esta tarea es el llamado “Cruzamiento”, para lograr este objetivo en el mismo se aplicará la técnica de “cruce por un punto”, donde un padre aporta sus genes que están antes del punto de cruce y el otro padre sus genes que restan después de dicho punto.

Código del Método “Cruzamiento”

```
CCriatura**AG::Cruzamiento(CCriatura** Lista)
  {
   Randomize();
  cant_nuevos_ind=0;
  CCriatura** List_Nuevos_Ind=new CCriatura*[cant_Selecc];
  for (int i=0;i<cant_Selecc;i++)
  {
  CCriatura * nuevoInd= new CCriatura();
  int PtoCruce=rand()%8+1;
  int possP1=rand()%cant_Selecc;
  int possP2=rand()%cant_Selecc;


for (int j=0;j<PtoCruce ;j++)
  nuevoInd->Set_Cromosoma_Poss(j, Lista [possP1]-
>Get_Cromosoma_Poss(j));

  for (int j=PtoCruce;j<cromosomas;j++)
  nuevoInd->Set_Cromosoma_Poss(j, Lista [possP2]-
>Get_Cromosoma_Poss(j));

  List_Nuevos_Ind[cant_nuevos_ind]=nuevoInd;
  cant_nuevos_ind++;
  }
  return List_Nuevos_Ind;
}
```

En la implementación anterior se pasa como parámetro una lista que debe ser la que contenga a los individuos que se seleccionaron para realizar el cruzamiento, luego para cada iteración se genera un punto de cruce con un valor aleatorio entre cero y once porque cada individuo solo tiene doce cromosomas.

Después de la lista de padres se seleccionan de manera aleatoria dos individuos para crear uno nuevo por la técnica de “cruce por un punto” como se había dicho anteriormente. Luego se adiciona el nuevo individuo a una lista que será retornada por el método cuando se terminen el número de iteraciones, este último está determinado por la cantidad de elementos que tenga la lista de padres.

En el proceso de obtención de una generación de individuos la estructura “AG” puede también apoyarse en el método “Mutación”, el mismo será el encargado de aplicarle la operación de mutación a un individuo de la nueva generación, es decir después que se crean nuevos individuos es posible que estos sean sometidos a la mutación para garantizar la inserción de nuevo material cromosómico.

Este proceso se simula semejante a como ocurre en la vida real, con un bajo grado de probabilidad de ocurrencia, Es por ello que el método “Mutación” reasignará los cromosomas aleatoriamente, cada rasgo tiene una posibilidad del 5% de mutar.

Fragmento de código del método “Mutacion”

```
CCriatura** AG::Mutacion(CCriatura** temp)
{
CCriatura**Lista;
Lista=temp;
int i=rand()%cant_nuevos_ind;
   Randomize();
  if (((rand()%20)+1)==1)
     switch ((rand()%5)+1) {
         case 1:
         Lista[i]->Set_Cromosoma_Poss(Atacado_por_Grupo,Retirarse);
         break;
         case 2:
         Lista[i]->Set_Cromosoma_Poss(Atacado_por_Grupo,Esconderse);
         break;
         case 3:
         Lista[i]-
>Set_Cromosoma_Poss(Atacado_por_Grupo,Atacar_con_Espada);
```

```
        break;
        case 4:
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_por_Grupo,Atacar_con_Mazo);
        break;
        case 5:
        Lista[i]->Set_Cromosoma_Poss(Atacado_por_Grupo,
Atacar_con_Magia);
        break;
        }
  …….. }
```

En el fragmento anterior de implementación el método recibe una lista, que debe ser la que contenga a los nuevos individuos y se selecciona de manera aleatoria a cualquiera de ellos para realizarle el proceso, también se determina aleatoriamente el gen que se va a mutar y se retorna la lista, se debe tener en cuenta que durante este proceso es probable que ningún individuo sufra alteración genética.

2.5 Reposición y Evolución

Después que se genera una nueva descendencia, esta debe ser insertada en la población actual, en la estructura “AG” el método que será el encargado de realizar dicho proceso es “Reposición”.

Implementación del método “Reposicion”

```
void AG::Reposicion(CCriatura **Lista) {
int possi=0;
if (cant_nuevos_ind>cant_pob) {
Poblacion=new CCriatura*[cant_nuevos_ind];
for(int i=0;i<cant_nuevos_ind;i++)
Poblacion[i]=Lista[i];
cant_pob=cant_nuevos_ind;
} else
  {
   CCriatura * temp;
   for(int i =cant_pob-1; i >= 0; i--) {
      for (int j = 1; j <= i; j++) {
        if (Poblacion[j-1]->Get_Aptitud()<Poblacion[j]->Get_Aptitud())  {
           temp=Poblacion[j-1];
```

```
            Poblacion[j-1] =Poblacion[j];
            Poblacion[j] = temp;
          }
      }
  }
for(int p=cant_pob; p>cant_pob-cant_nuevos_ind; p--) {
Poblacion[p-1]=Lista[possi];
possi++;
}
}
}
```

En la implementación anterior para insertar la nueva descendencia en la población se tuvo en cuenta que la misma podía ser mayor que la población actual, en dicho caso la nueva generación de individuos pasaría a conformar la población, eliminándose a todos los individuos que la constituían anteriormente.

Este suceso solo pudiera ocurrir si la cantidad de padres que fueron seleccionados es lo suficientemente mayor que la cantidad de individuos de la población en aquel momento.

Ahora en caso que la n cantidad de la nueva descendencia sea menor que la poblacion actual solo serán eliminados los enésimos individuos que contengan los menores valores de aptitud en la población y se insertarán por cada uno de ellos a los individuos de la nueva descendencia.

En este último caso para determinar cuales son los individuos menos aptos se organiza la población de manera descendente según su valor de “Aptitud” utilizando método de ordenamiento “Burbuja”, el mismo tiene como ventaja que es sencillo, de fácil implementación y es eficaz, sin embargo consume bastante tiempo, ya que requiere de muchas lecturas y escrituras en memoria y realiza muchos intercambios, siempre hace la misma cantidad de comparaciones incluso cuando la lista ya está ordenada.

Pudiera considerarse en la implementación de “Reposición” otros métodos de ordenamiento como el “MetodoShell” pudiendo resultar mucho más rápidos en los casos que lo requieran.

Una vez que se han definido e implementado todos los métodos del “AG” que realizan los procesos específicos de la genética, hay que integrar los mismos de alguna manera, esta sería la funcionalidad del método “Evolucion”.

Implementación del método “Evolucion”

```
void AG::Evolucion(int mejores)
{
Reposicion (Mutacion (Cruzamiento( Seleccion_Natural (mejores))));
}
```

Esta sencilla implementación solo se encarga de llamar a los métodos del “AG” de una manera lógica para que se realice de manera correcta cada ciclo de vida del AG.

De esta manera pudiera quedar conformada la estructura “AG” capaz de determinar el mejor comportamiento que pudiera tener el luchador controlado por la PC ante diferentes escenarios con el objetivo de ser un buen adversario para cualquier jugador. Tratando de demostrar que la aplicación de los “AG” en soluciones a problemas de IA es una opción confiable y que pueden alcanzar buenos resultados en este tipo de problemas.

Para ello se tuvo en cuenta que el problema tuviera un espacio de búsqueda delimitado, es decir en el ejemplo en cuestión el espacio de búsqueda consistió en los escenarios que se definieron y en los posibles comportamientos que podía tener el luchador.

Para determinar cuales eran las mejores respuestas, se definió una función de aptitud bastante sencilla, solo estaba determinada por un cociente. También las posibles soluciones fueron codificadas con constantes, esto hizo más fácil la implementación de la estructura y su comprensión.

En conclusión al desarrollo del ejemplo se propone que elementos tangibles como los descritos anteriormente, siempre se tengan en cuenta en la aplicación de un AG en cualquier problema a solucionar en la IA de un entorno virtual, para lograr un funcionamiento correcto del mismo y la consecuente solución satisfactoria del problema planteado.

Para lograr una mejor compresión de las ideas generales que han sido abordadas en este capítulo se presenta el siguiente gráfico:

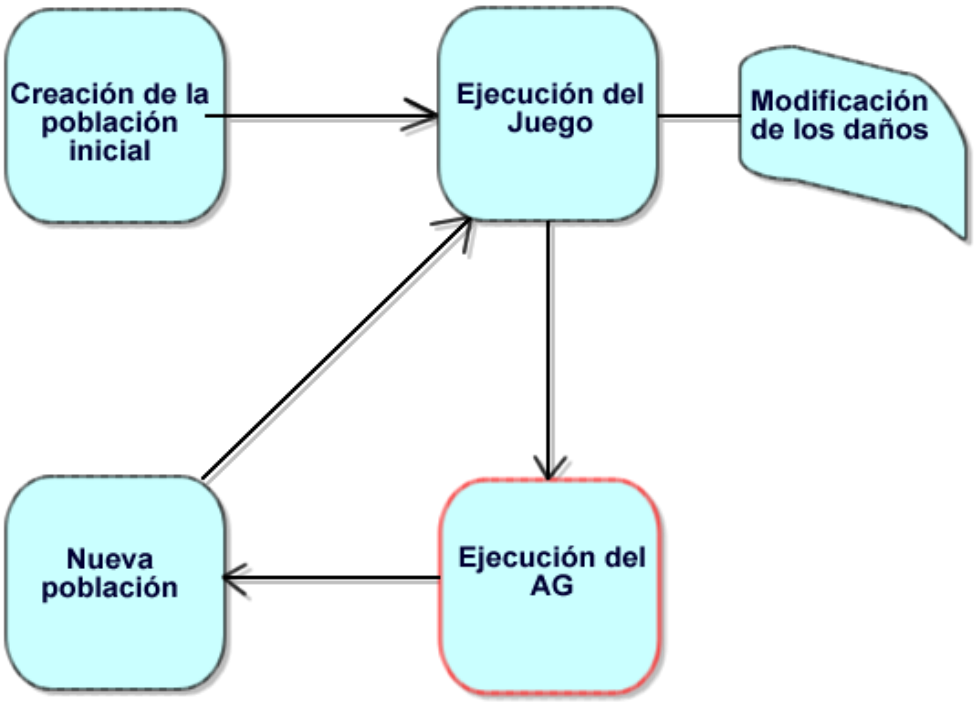


Figura 3. Ciclo de vida de un Juego

En el mismo se ilustran los diferentes estados por los que pudiera estar compuesto un ciclo completo de un juego, para dar una mayor idea del momento en que se pudiera invocar el AG. Un ciclo comienza por la creación de población inicial de los individuos que estarán en los escenarios del juego, acto seguido se inicia la ejecución del juego con todas las interacciones de los individuos que esto acarrea modificándose los valores de los daños causados y recibidos de cada uno, posterior a esto se puede invocar al AG con la población actual para generar nuevos individuos con caracteres diferentes. Realizándose este proceso para cada iteración del ciclo.

Capítulo 3

Título: Resultados.

En este capitulo serán expuestos los resultados alcanzados de la presente investigación mediante la comparación de la estructura realizada con otras técnicas de programación evolutiva, así como las ventajas y limitaciones que tuvo la misma al aplicarse en el caso de estudio y serán descritas algunas áreas donde se pudiera utilizar el aporte científico realizado.

3.1 Comparación con otras técnicas.

Una vez que se ha realizado el caso de estudio se analizará por que los AG son una de las técnicas más adecuadas para este tipo de problemas. Primero se debe saber que otras técnicas de la computación evolutiva pudieron ser aplicadas, ejemplo de ellas son:

“La Programación Genética”, esta técnica al igual que los AG resuelven los problemas generando poblaciones sucesivas a las que se aplican los operadores de cruce y mutación. Cada individuo representa una solución al problema, y se trata de encontrar al individuo que represente a la mejor solución. Solo que la Programación Genética se centra en el estudio de problemas cuya solución es un programa, esto quiere decir que los individuos de la población son programas. Es obvio que para el ejemplo realizado esta técnica no es la adecuada ya que está diseñada para resolver problemas muchos más complejos.

Otra técnica que pudo ser aplicada es “La Programación Evolutiva”, la misma es otro enfoque de los AG, su peculiaridad es que trata de gestionar los operadores genéticos para que imiten lo mejor posible a la naturaleza, yéndosele por encima a la relación de los padres con su descendencia. En este caso no se utiliza el operador de cruce, tomando la máxima importancia el operador de mutación.

Esta técnica trata por todos los medios de buscar una perfección total de los individuos alterando demasiado su proceso de evolución natural, para el caso de estudio anterior esto traería como inconveniente que se crearían súper luchadores y esa no es la idea, es decir se trata de buscar un adversario digno, no invencible lo cual atentaría contra la ejecución lógica y realista del juego.

Otra de las técnicas de aprendizaje más usadas son las “Redes Neuronales”, las mismas tienen múltiples aplicaciones en el mundo de la inteligencia artificial, principalmente en la robótica y aprendizaje de máquinas, Estas son un método de resolución de problemas basado en un modelo informático de la misma manera en que están conectadas las neuronas del cerebro.

En los juegos también tienen sus aplicaciones pero las Redes Neuronales a diferencia de los algoritmos genéticos no son de una fácil comprensión, aplicación e implementación, su nivel de complejidad es considerablemente mayor al de un AG, además de necesitar un tiempo de entrenamiento que a veces se hace difícil de encontrar en el tipo de aplicaciones que se presenta.

Es por ello que pudiera ser trabajoso y difícil a la hora de ser aplicadas en un juego, el tiempo de aprendizaje de un a red neuronal puede ser muy largo, es por ello que para mejorar este aspecto en ocasiones se utiliza un AG para ayudar en el entrenamiento de la red neuronal, esto es un ejemplo de vinculación de varias de estas técnicas para lograr el objetivo propuesto por lo que ellas no tienen que ser excluyentes.

También las Máquinas de Estado Finito (FSM), han sido utilizadas como instrumentos para codificar y manejar la IA de los agentes en los juegos desde el surgimiento de los mismos. Sus principales características son:

La primera de ellas es que todas las diferentes maneras de programar las FSM son de rápida y fácil implementación.

Presentan un bajo uso del procesador; apropiado para dominios donde el tiempo de ejecución está compartido entre varios módulos o subsistemas. Solo el código del estado actual ha de ser ejecutado Es fácil determinar si se puede llegar o no a un estado, en las representaciones abstractas, resulta obvio si se puede o no llegar a un estado desde otro, y qué requerimientos existen para hacerlo.

También son fáciles de ejecutar, pues los comportamientos de los agentes de juegos se dividen en fragmentos más fáciles de manejar y modificar.

Las FSM son considerablemente flexibles, las mismas pueden ser ajustadas de manera sencilla por los programadores para lograr los comportamientos deseados. Solo es cuestión de expandir el área o espectro de los comportamientos de los agentes mediante la adición de nuevos estados y reglas.

La principal desventaja que tendría usar una FSM en el ejemplo que se ha desarrollado es que las mismas tienen un alto carácter predecible ante las posibles situaciones, que atentaría contra la IA que requiere un juego, esto podría solucionarse implementando una FSM no determinista para tratar de elevar el nivel de diversidad del mismo. Sin embargo otras técnicas como los AG y las Redes Neuronales se han vuelto más populares que las FSM en el desarrollo de aplicaciones de IA.

3.2 Ventajas de los AG.

Una de las principales ventajas de los AG y la que más los diferencia de otras técnicas, es que ellos tienen descendencia múltiple, o lo que es lo mismo pueden buscar la solución por muchos caminos a la vez, si una solución encontrada no le sirve pueden eliminarla fácilmente y seguir buscando la óptima en el ejemplo explicado anteriormente se va probando de manera aleatoria hasta al final encontrar el individuo más apto explorando las posibles respuestas.

Cuando se está realizando un juego esto es muy importante porque siempre va a garantizar que se encuentre una solución bastante optimizada otros algoritmos lineales pueden descubrir una solución no óptima y esto los obligaría a comenzar de nuevo lo cual llevaría mucho tiempo y los juegos en tiempo real no pueden darse ese lujo.

Esto se suele llamar también teorema de esquema, constituyendo un punto clave de superioridad de los AG sobre otros métodos de resolución de problemas.

Otra ventaja que justifica la utilización de los AG en el desarrollo de aplicaciones de realidad virtual, como son los juegos, es su potencialidad y calidad en el tratamiento de problemas en los que la función objetivo, es discontinua, ruidosa, cambia con el tiempo, o tiene muchos óptimos locales, donde el mayor problema está en evitar los óptimos locales. Así se podrían enfrentar propuestas de mayor envergadura a este tipo de aplicaciones, siendo mayores las posibles ganancias.

La mayoría de estos problemas prácticos tienen un espacio de soluciones enorme, imposibles de explorar exhaustivamente, muchos otros algoritmos de búsqueda simplemente solo llegarían a óptimos locales concluyendo que han alcanzado la mejor de todas aunque existan picos más altos no explorados.

También el cruzamiento en sí mismo constituye una superioridad de los AG porque con el cruzamiento en los juegos, existe una transferencia de información entre los candidatos prósperos por ejemplo en nuestro caso de estudio los luchadores pueden beneficiarse de lo que otros han aprendido, los esquemas pueden combinarse y de producir una descendencia que tenga las virtudes de sus dos padres dígase las mejores repuestas ante los ataques en cualquiera de lo escenarios que fueron esbozados y ninguna de sus debilidades .

Otra de las cualidades de los AG más palpables es que ellos no saben nada de los problemas que deben resolver, solo se limitan a realizar cambios aleatorios en posibles soluciones y después usan la función objetivo para determinar si esos cambios producen una mejora o no, por el contrario de otras estrategias de resolución de problemas que dependa de un conocimiento previo que deba descartar muchos caminos a priori para ganar tiempo, perdiendo así cualquier solución novedosa que pueda ser hallada.

Además nunca se ven afectados negativamente por la ignorancia de algunos aspectos al iniciar su ejecución, lo cual amplía su espectro de aplicación porque garantiza que puedan ser aplicados en casi todos los campos, en nuestro ejemplo busca de forma aleatoria hasta encontrar la mejor respuesta en cada escenario y ante cualquier tipo de ataque o atacante sin buscar directamente la solución o tener que analizar cual podrá ser antes de comenzar.

Otra cualidad que presentan los AG es su gran habilidad para manipular muchos parámetros simultáneamente. En los juegos todos los problemas no se pueden resolver empleando un único valor de calidad que hay que minimizar o maximizar, sino que deben expresarse en términos de múltiples objetivos, dependiendo del área específica en que se desenvuelva, de manera que uno solamente puede mejorar a expensas de otro.

Los AG son muy buenos resolviendo estos problemas porque su uso del paralelismo les permite producir múltiples soluciones, todas buenas, donde posiblemente una de ella puede optimizar un parámetro y otra puede optimizar otro completamente distinto, y luego el programador puede seleccionar una de esas candidatas para su utilización. Esta ventaja garantiza que la IA de los juegos que se puedan desarrollar sea alta y competente logrando el mayor nivel de realismo de los juegos que se puedan producir así como lograr las respuestas rápidas en tiempo real con menos recursos.

3.3 Limitaciones de los AG

A pesar de la factibilidad de la utilización de los AG como excelente técnica de resolución de problemas, ahora serán señalados sus principales limitaciones o desventajas, la primera de ellas es que se debe tener especial cuidado a la hora de definir la representación del problema, el lenguaje utilizado para especificar soluciones candidatas debe ser robusto porque si esto no se tiene en cuenta , los cambios aleatorios pueden producir constantemente errores fatales o resultados sin sentido.

La solución a este problema o desventaja es la utilización de uno de los métodos de representación referenciados anteriormente, en el ejemplo desarrollado se utilizó la codificación de las soluciones como constantes enteras, donde cada una representaba algún aspecto particular de la solución.
Por lo tanto, el proceso genético implica cambiar estos números y lograr nuevos individuos. En este caso el código es lo que dirige la simulación y hace un seguimiento de los individuos, evaluando sus aptitudes y asegurando que sólo se producen valores realistas y posibles para el problema dado.

Además de elegir bien la función objetivo, también deben elegirse cuidadosamente los otros parámetros de un AG como son el tamaño de la población, el ritmo de mutación y cruzamiento, el tipo y fuerza de la selección. Si el tamaño de la población es demasiado pequeño, puede que el AG no explore suficientemente el espacio de soluciones para encontrar buenas soluciones.

Si el ritmo de cambio genético es demasiado alto o el sistema de selección se escoge inadecuadamente, puede alterarse el desarrollo de esquemas beneficiosos y la población puede entrar en catástrofe de errores, al cambiar demasiado rápido para que la selección llegue a producir convergencia. En los juegos esto se traduce en que no se pueda encontrar el individuo más apto.

Otra dificultad se puede presentar a la hora de concebir la función objetivo que realmente de solución a la tarea de encontrar el individuo más apto en el problema en cuestión, si esta tarea no se realiza con calidad puede traer fatales consecuencias porque es posible que el AG no encuentre la solución adecuada o sencillamente resuelva otro problema no deseado, en fin que no cumpla su función

Por ejemplo en el caso de estudio analizado anteriormente si se hubiera escrito una función aptitud que no fuera fiel al contexto puede que nunca se llegara a determinar quienes son los luchadores más aptos o se hubieran obtenido resultados infieles, por eso cada programador debe prestar especial atención a este tema a la hora de programar el juego.

También existe otra problemática conocida de los AG y es a la llamada “convergencia prematura”. Que en el ejemplo analizado se podría percibir si uno de los luchadores que es mucho más apto que la mayoría sale a relucir rápidamente, se puede reproducir tantas veces que merme la diversidad de la población muy rápido produciendo que el algoritmo converja hacia el máximo local que constituye ese luchador, en vez de rastrear todo el espectro de los luchadores en busca del óptimo global.

Este es un problema común en las poblaciones pequeñas que se debe tener muy en cuenta, recomendándose por los expertos como mejor solución para el mismo controlar la fuerza selectiva para no proporcionar tanta ventaja a los individuos extremadamente aptos, existen varios métodos para lograr este objetivo en específico pero uno de los más adecuados es el escalado sigma, éste se centra en una comparación estadística de la aptitud media de la población.

De manera general todas estas dificultades son de cierta manera lógicas porque la evolución como estrategia de resolución de problemas no necesariamente debe encontrar la mejor solución, sólo podemos decir que es

bastante buena, porque todos estos problemas por lo general también ocurren en la naturaleza, que es por donde se guiaron los creadores de los AG.

3.4 Áreas donde se puede aplicar la propuesta

Juegos de carreras de autos

Una de las áreas referidas es el proyecto “Juegos de Consola”, en el mismo se está desarrollando una aplicación de realidad virtual que consiste en un juego de carreras de autos multi-jugador. Uno de los requerimientos del juego es que permita al usuario realizar carreras contra varios autos controlados por la PC, los mismos requieren un cierto nivel de inteligencia para darle al jugador la mayor sensación de realidad posible.

En este punto es donde pudiera aplicarse la propuesta de AG desarrollada en este trabajo, pues en este caso se presentan varias similitudes con el ejemplo expuesto anteriormente, de manera que se puede lograr que los automóviles controlados por la PC se comporten de una manera natural en el entorno simulado.

Por ejemplo para una situación donde, el auto de carreras controlado por la PC se encuentre en la misma dirección que otro auto del entorno pero en sentidos opuestos, de manera que se provocaría una colisión, una posible respuesta que pudiera dar el auto controlado por la PC sería frenar, para evitar colisionar, o cambiar de senda.

Para el resto de los escenarios posibles se codificarían las respuestas que pudieran ser aplicadas a los mismos, de manera que el auto controlado por la PC siempre actúe de forma lógica.

Así pudiera definirse una función objetivo determinada por elementos como: el tiempo total de la carrera, la posición final que obtuvo el auto, la cantidad de autos que saco de competencia, veces que tuvo que reincorporarse, etc.

Juego de guerra

También en el proyecto “Tiro” se está desarrollando un juego de guerra, donde uno de sus requerimientos consiste en crearle a un jugador un escenario donde varios soldados controlados por la PC traten de llegar a él para eliminarlo. Se puede ir aumentando la capacidad de reacción de los elementos que van apareciendo al usuario, que vayan surgiendo nuevos soldados con características más adecuadas para evitar el disparo del jugador.

Por ejemplo, uno o varios soldados controlados por la PC para tratar de eliminar al jugador pudieran hacerlo moviéndose entre obstáculos que habrían entre ellos, una situación pudiera ser, que el jugador tenga un blanco en la mira, comience a disparar y el soldado controlado por la PC se esconda detrás del obstáculo más próximo a él, para evitar los disparos. En este caso la función de aptitud pudiera determinarse por factores como: distancia que logró aproximarse al jugador, cantidad de disparos acertados, etc.

Simulador de Auto

En el proyecto “Simpro “se está desarrollando un simulador de autos para el aprender a conducir, uno de los requerimientos del mismo es que elementos del entorno, como son los automóviles, se comporten de una manera inteligente, cada automóvil tiene varias características que pueden ir variando de uno a otro para encontrar los valores que más se acerquen a la realidad.

Estas características pueden ser: máxima velocidad permitida, máxima velocidad de rotación, máximo valor de fuerza aplicado, tiempo de respuesta, nivel de osadía, etc. todas ellas con el fin de proporcionar al usuario situaciones como por ejemplo: Estar en un semáforo y que los automóviles del entorno lo respeten al igual que él, que un automóvil al percibir una proximidad inadecuada por parte del usuario frene repentinamente y se provoque un accidente, etc. Con esto se lograría una codificación, donde la función objetivo pudiera ser, por ejemplo, una relación entre los hechos deseados y los no deseados.

Conclusiones

Al Finalizar el desarrollo del presente trabajo de investigación se han resuelto satisfactoriamente los objetivos propuestos.

De manera que a partir de la problemática que dio origen a la investigación, o sea, la carencia de una técnica avanzada de IA que permitiera alcanzar los requerimientos actuales de este elemento, se propusieron varias tareas a desarrollar las cuales se cumplieron a cabalidad.

Primero a través de una exhaustiva explicación, se mostraron los elementos que componen un Algoritmo Genético, lográndose una amplia visión y comprensión del funcionamiento de los mismos.

Después se obtuvo como resultado una estructura “AG”, proveniente del análisis previo, el cual fue aplicado en un caso de estudio implementado en c++, que constituye una propuesta concreta de cómo utilizar los algoritmos genéticos en las aplicaciones de realidad virtual.

Además se fundamentó la factibilidad de los Algoritmos Genéticos en los entornos virtuales mediante la comparación de los mismos con otros métodos de programación evolutiva, argumentándose las ventajas y limitaciones de haber utilizado dicha técnica y exponiéndose posibles áreas donde pudiera ser aplicada la misma.

Recomendaciones

1- Que se tengan en cuentan las propuestas que fueron formuladas en esta investigación a la hora de programar los módulos de inteligencia Artificial de los proyectos de realidad virtual de la facultad 5 y otras áreas relacionadas con el tema.

2- Que este trabajo sirva como punto de partida y base de estudio para futuras tesis o investigaciones relacionadas con los Algoritmos Genéticos y otras técnicas evolutivas en la UCI.

3- Que este trabajo sirva de base para las investigaciones y aplicaciones de técnicas de la Inteligencia Artificial a entornos virtuales.

Referencias Bibliográficas

[Altshuler y Linden ,1997] Altshuler, Edward y Derek Linden. ``Design of a wire antenna using a genetic algorithm.'' Journal of Electronic Defense, vol.20, no.7, p.50-52 (Julio de 1997).

[Buckles & Petry, 1992] Buckles, Bill P. and Petry, Frederick E. Genetic Algorithms. IEEE Computer Society Press
Technology Series. 109 p., 1992.

[Charbonneau, 1995] Charbonneau, Paul. ``Genetic algorithms in astronomy and astrophysics.'' The Astrophysical Journal Supplement Series, vol.101, p.309-334 (diciembre de 1995).

[Chellapilla y Fogel, 2000] Chellapilla, K.; Fogel, D.B. Evolutionary Computation, 2000. Proceedings of the 2000 Congress on Volume 2, Issue, 2000 Page(s):857 - 863 vol.2.

[DEGAZIO, 1999] Bruno Degazio. “La evolución de los organismos musicales” (The evolution of musical organisms). In Miranda, Eduardo Reck (ed.) Música y nuevas tecnologías: Perspectivas para el siglo XXI. Barcelona: L'Angelot. 1999.

[Fogel. y Anderson, 2000]. Revisiting Bremermann's Genetic Algorithm: I. Simultaneous Mutation of All Parameters. Proceedings of the Congress of Evolutionary Computation, pp. 1204-1209, IEEE Press.

[Goldberg, 1989]. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.

[Haupt y Haupt ,1998] Haupt, Randy y Sue Ellen Haupt. Practical Genetic Algorithms. John Wiley & Sons, 1998.

[Hodges, 2002]. *Alan Turing at The Stanford Encyclopedia of Philosophy*, Edward N. Zalta
(Ed.). Disponible en http://plato.stanford.edu/archives/sum2002/entries/turing.

[Holland, 1975] Holland, John H. Adaptation in Natural and Artificial Systems. Ann Harbor: University of Michigan Press.

[Hughes y Leyland 2000] Hughes, Evan y Maurice Leyland. ``Using multiple genetic algorithms to generate radar point-scatterer models.'' IEEE Transactions on Evolutionary Computation, vol.4, no.2, p.147-163 (Julio de 2000).

[Koza, 1992] Koza, John R. Genetic Programming. On the Programming of Computers by Means of Natural Seleccion, 1992

[Naik, 1996]. Naik, Gautam. ``Back to Darwin: In sunlight and cells, science seeks answers to high-tech puzzles.'' The Wall Street Journal, 16 de enero de 1996, p. A1.

[Obayashi, 2000] .Obayashi, Shigeru, Daisuke Sasaki, Yukihiro Takeguchi, y Naoki Hirose. ``Multiobjective evolutionary computation for supersonic wing-shape optimization.'' IEEE Transactions on Evolutionary Computation, vol.4, no.2, p.182-187 (Julio de 2000).

[Sambridge y Gallaguer 1993] Sambridge, Malcolm y Kerry Gallagher. ``Earthquake hypocenter location using genetic algorithms.'' Bulletin of the Seismological Society of America, vol.83, no.5, p.1.467-1.491 (octubre de 1993).

[Sasaki, 2001] .Sasaki, Daisuke, Masashi Morikawa, Shigeru Obayashi y Kazuhiro Nakahashi. ``Aerodynamic shape optimization of supersonic wings by adaptive range multiobjective genetic algorithms.'' In Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001, Zurich, Switzerland, March 2001: Proceedings, K. Deb, L. Theile, C. Coello, D. Corne y E. Zitler (eds). Notas de la confenrencia en Computer Science, vol.1993, p.639-652. Springer-Verlag, 2001.

[Sato, 2002]. Sato, S., K. Otori, A. Takizawa, H. Sakai, Y. Ando y H. Kawamura. ``Applying genetic algorithms to the optimum design of a concert hall.'' Journal of Sound and Vibration, vol.258, no.3, p. 517-526 (2002).

[Schwefel, 1975]. *Evolutionsstrategie und Numerische Optimierung*. Ph. D. Tesis,
Technical University, Berlin, Alemania.

[Srinivas, M., y Patnaik, Junio 1998, 2000] L. M., "Genetic Algorithms: A Survey", IEEE Computer, Junio 1998, 2000, pp. 17-26.

[Tang, 1996].Tang, K.S., K.F. Man, S. Kwong y Q. He. ``Genetic algorithms and their applications.'' IEEE Signal Processing Magazine, vol.13, no.6, p.22-37 (noviembre de 1996).

[Von Neumann, 1966]. Theory of Self-Reproducing Automata. University of Illinois Press

[Watson y Pollack, 1999]. How symbiosis can guide evolution. Proceedings of the Fifth European Conference on Artificial Life, pp. 29-38, Springer-Verlag.

[Watson. y Pollack, 2000]. Symbiotic Combination as an Alternative to Sexual Recombination in Genetic Algorithms. Proceedings of Parallel Problem Solving from
Nature, pp. 425-434, Springer-Verlag.

Bibliografía

- Bourg, David M. y Seeman, Glenn, Julio 2004, “AI for Game Developers”, ISBN: 0-596-00555-5.

- Buckland, Mat, 2002, “AI techniques for game programming”.

- Dr. Guerra, Alejandro, abril 2004, “Aprendizaje Automático: Algoritmos Genéticos”.

- Dr. Coello, Carlos A. 2006,”Introducción a la Computación Evolutiva”.

- Dr. Coello Coello, Carlos A. 2000,”Algoritmos Genéticos y sus Aplicaciones”.

- Gil Londoño, Natyhelem 2006,”Algoritmos Genéticos”.

- Hidalgo Pérez, José Ignacio 2002, “Una revisión de los algoritmos evolutivos y sus aplicaciones”.

- Tolmos Rodríguez-Piñero, Piedad 2000, “Introducción a los Algoritmos Genéticos y sus Aplicaciones”.

- Tozour, Paul, 2001, “The Evolution of Game AI”.

Anexos

Implementación del método "Crear_Individuo"

```
void CCriatura:: Crear_Individuo ()
{
Randomize();
switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Atacado_por_Grupo]= Retirarse;
break;
case 2:
list_Cromosomas [Atacado_por_Grupo]=Esconderse;
break;
case 3:
list_Cromosomas [Atacado_por_Grupo]=Atacar_con_Espada;
break;
case 4:
list_Cromosomas [Atacado_por_Grupo]=Atacar_con_Mazo;
break;
case 5:
list_Cromosomas [Atacado_por_Grupo]=Atacar_con_Magia;
break;
}
Randomize();
switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Interactuar_con_Curador ]=Retirarse;
break;
case 2:
list_Cromosomas [Interactuar_con_Curador ]=Esconderse;
break;
case 3:
list_Cromosomas [Interactuar_con_Curador ]= Atacar_con_Espada;
break;
case 4:
list_Cromosomas [Interactuar_con_Curador ]= Atacar_con_Mazo;
break;
case 5:
list_Cromosomas [Interactuar_con_Curador ]= Atacar_con_Magia;
break;
}
//Respuesta a los ataques
Randomize();
   switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Atacado_con_Espada]=Retirarse;
```

```
break;
case 2:
list_Cromosomas [Atacado_con_Espada]=Esconderse;
break;
case 3:
list_Cromosomas [Atacado_con_Espada]=Usar_ArmaduraMetal;
break;
case 4:
list_Cromosomas [Atacado_con_Espada]= Usar_ArmaduraMagica;
break;
case 5:
list_Cromosomas [Atacado_con_Espada]= Usar_ArmaduraCuero;
break ;
}
Randomize();
switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Atacado_con_Mazo]=Retirarse;
break;
case 2:
list_Cromosomas [Atacado_con_Mazo]=Esconderse;
break;
case 3:
list_Cromosomas [Atacado_con_Mazo]= Usar_ArmaduraMetal;
break;
case 4:
list_Cromosomas [Atacado_con_Mazo]= Usar_ArmaduraMagica;
break;
case 5:
list_Cromosomas [Atacado_con_Mazo]= Usar_ArmaduraCuero;
break;
}
Randomize();
switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Atacado_con_Disparo]= Retirarse;
break;
case 2:
list_Cromosomas [Atacado_con_Disparo]=Esconderse;
break;
case 3:
list_Cromosomas [Atacado_con_Disparo]= Usar_ArmaduraMetal;
break;
case 4:
list_Cromosomas [Atacado_con_Disparo]= Usar_ArmaduraMagica;
break;
case 5:
list_Cromosomas [Atacado_con_Disparo]= Usar_ArmaduraCuero;
break;
}
```

```
Randomize();
switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Atacado_con_Magia]=Retirarse;
break;
case 2:
list_Cromosomas [Atacado_con_Magia]=Esconderse;
break;
case 3:
list_Cromosomas [Atacado_con_Magia]= Usar_ArmaduraMetal;
break;
case 4:
list_Cromosomas [Atacado_con_Magia]= Usar_ArmaduraMagica;
break;
case 5:
list_Cromosomas [Atacado_con_Magia]= Usar_ArmaduraCuero;
break;
}
Randomize();
switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Atacante_con_ArmaduraMetal]= Retirarse;
break;
case 2:
list_Cromosomas [Atacante_con_ArmaduraMetal]= Esconderse;
break;
case 3:
list_Cromosomas [Atacante_con_ArmaduraMetal]= Atacar_con_Espada;
break;
case 4:
list_Cromosomas [Atacante_con_ArmaduraMetal]= Atacar_con_Mazo;
break;
case 5:
list_Cromosomas [Atacante_con_ArmaduraMetal]= Atacar_con_Magia;
break;
}
Randomize();
switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Atacante_con_ArmaduraCuero]= Retirarse;
break;
case 2:
list_Cromosomas [Atacante_con_ArmaduraCuero]= Esconderse;
break;
case 3:
list_Cromosomas [Atacante_con_ArmaduraCuero]= Atacar_con_Espada;
break;
case 4:
list_Cromosomas [Atacante_con_ArmaduraCuero]= Atacar_con_Mazo;
break;
```

```
case 5:
list_Cromosomas [Atacante_con_ArmaduraCuero]= Atacar_con_Magia;
break;
}
Randomize();
switch ((rand()%5)+1) {
case 1:
list_Cromosomas [Atacante_con_ArmaduraMagica]= Retirarse;
break;
case 2:
list_Cromosomas [Atacante_con_ArmaduraMagica]= Esconderse;
break;
case 3:
list_Cromosomas [Atacante_con_ArmaduraMagica]= Atacar_con_Espada;
break;
case 4:
list_Cromosomas [Atacante_con_ArmaduraMagica]= Atacar_con_Mazo;
break;
case 5:
list_Cromosomas [Atacante_con_ArmaduraMagica]= Atacar_con_Magia;
break;
   }
}
```

Implementación del método “Mutacion”

```
CCriatura** AG::Mutacion(CCriatura** temp)
{
CCriatura**Lista;
Lista=temp;
int i=rand()%cant_nuevos_ind;
   Randomize();
  if (((rand()%20)+1)==1)
     switch ((rand()%5)+1) {
          case 1:
          Lista[i]->Set_Cromosoma_Poss(Atacado_por_Grupo,Retirarse);
          break;
          case 2:
          Lista[i]->Set_Cromosoma_Poss(Atacado_por_Grupo,Esconderse);
          break;
          case 3:
```

```
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_por_Grupo,Atacar_con_Espada);
        break;
        case 4:
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_por_Grupo,Atacar_con_Mazo);
        break;
        case 5:
        Lista[i]->Set_Cromosoma_Poss(Atacado_por_Grupo,
Atacar_con_Magia);
        break;
        }

Randomize();
  if (((rand()%20)+1)==1)
    switch ((rand()%5)+1) {
        case 1:
        Lista[i]->Set_Cromosoma_Poss(Interactuar_con_Curador,Retirarse);
        break;
        case 2:
        Lista[i]->Set_Cromosoma_Poss(Interactuar_con_Curador,Esconderse);
        break;
        case 3:
        Lista[i]-
>Set_Cromosoma_Poss(Interactuar_con_Curador,Atacar_con_Espada);
        break;
        case 4:
        Lista[i]-
>Set_Cromosoma_Poss(Interactuar_con_Curador,Atacar_con_Mazo);
        break;
        case 5:
        Lista[i]-
>Set_Cromosoma_Poss(Interactuar_con_Curador,Atacar_con_Magia);
```

```
        break;
       }
          Randomize();
  if (((rand()%20)+1)==1)
    switch ((rand()%5)+1) {
        case 1:
        Lista[i]->Set_Cromosoma_Poss(Atacado_con_Espada,Retirarse);
        break;
        case 2:
        Lista[i]->Set_Cromosoma_Poss(Atacado_con_Espada,Esconderse);
        break;
case 3:
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Espada,Usar_ArmaduraMetal);
        break;

        case 4:
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Espada,Usar_ArmaduraMagica);
        break;
        case 5:
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Espada,Usar_ArmaduraCuero);
        break;


  }
          Randomize();
  if (((rand()%20)+1)==1)
    switch ((rand()%5)+1){
        case 1:
        Lista[i]->Set_Cromosoma_Poss(Atacado_con_Mazo,Retirarse);
        break;
        case 2:
        Lista[i]->Set_Cromosoma_Poss(Atacado_con_Mazo,Esconderse);
        break;
        case 3:
```

```
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Mazo,Usar_ArmaduraMetal);
        break;
      case 4:
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Mazo,Usar_ArmaduraMagica);
        break;
        case 5:
        Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Mazo,Usar_ArmaduraCuero);
        break;
    }
if (((rand()%20)+1)==1)
    switch ((rand()%5)+1) {
        case 1:
          Lista[i]->Set_Cromosoma_Poss(Atacado_con_Disparo,Retirarse);
          break;
        case 2:
          Lista[i]->Set_Cromosoma_Poss(Atacado_con_Disparo,Esconderse);
          break;
        case 3:
          Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Disparo,Usar_ArmaduraMetal);
          break;
        case 4:
          Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Disparo,Usar_ArmaduraMagica);
          break;
        case 5:
          Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Disparo,Usar_ArmaduraCuero);
        break;
    }
```

```
            Randomize();
 if (((rand()%20)+1)==1)
     switch ((rand()%5)+1) {
          case 1:
             Lista[i]->Set_Cromosoma_Poss(Atacado_con_Magia,Retirarse);
            break;
          case 2:
            Lista[i]->Set_Cromosoma_Poss(Atacado_con_Magia,Esconderse);
            break;
 case 3:
            Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Magia,Usar_ArmaduraMetal);
            break;
          case 4:
            Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Magia,Usar_ArmaduraMagica);
          break;
          case 5:
            Lista[i]-
>Set_Cromosoma_Poss(Atacado_con_Magia,Usar_ArmaduraCuero);
          break;
     }
             Randomize();
  if (((rand()%20)+1)==1)
     switch ((rand()%5)+1) {
        case 1:
       Poblacion[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraMetal,Retirarse);
          break;
       case 2:
       Poblacion[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraMetal,Esconderse);
```

```
          break;
        case 3:
        Poblacion[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraMetal,Atacar_con_Espada);
       break;
       case 4:
       Poblacion[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraMetal,Atacar_con_Mazo);
         break;



case 5:
        Poblacion[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraMetal,Atacar_con_Magia);
        break;
     }
              Randomize();
  if (((rand()%20)+1)==1)
     switch ((rand()%5)+1) {
          case 1:
     Lista[i]->Set_Cromosoma_Poss( Atacante_con_ArmaduraCuero,Retirarse );
        break;
          case 2:
        Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraCuero,Esconderse);
          break;
          case 3:
       Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraCuero,Atacar_con_Espada);
          break;
         case 4:
         Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraCuero,Atacar_con_Mazo);
```

```
            break;
            case 5:
           Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraCuero,Atacar_con_Magia);
            break;
      }
                  Randomize();
    if (((rand()%20)+1)==1)
      switch ((rand()%5)+1) {
            case 1:
            Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraMagica,Retirarse);
            break;
            case 2:
             Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraCuero,Esconderse);
             break;
            case 3:
            Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraCuero,Atacar_con_Espada);
            break;
            case 4:
            Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraCuero,Atacar_con_Mazo);
             break;
            case 5:
            Lista[i]-
>Set_Cromosoma_Poss(Atacante_con_ArmaduraCuero,Atacar_con_Magia);
                  break;
      }
    return Lista;
}
```

Glosario de Términos

Algoritmo: Un conjunto de reglas bien definidas para la solución de un problema en un número finito de pasos.

Aleatorio: Al azar, que no sigue un patrón, secuencia u orden determinado.

Aplicación: En informática las aplicaciones son los programas con los cuales el usuario final interactúa, es decir, son aquellos programas que permiten la interacción entre el usuario y la computadora. Esta comunicación se lleva a cabo cuando el usuario elige entre las diferentes opciones o realiza actividades que le ofrece el programa.

Heurística: Regla que permite orientar un algoritmo hacia la solución de un problema. Técnica de programación que permite a un sistema la creación gradual de un valor óptimo para una variable específica por medio del registro de los valores obtenidos en operaciones anteriores. Técnica empleada en los sistemas de inteligencia artificial.

Informática: Es la ciencia que estudia el tratamiento automático y racional de la información.

Inteligencia artificial: Es una rama de la Informática que pretende desarrollar programas en los que las computadoras desarrollen conductas típicas de los seres inteligentes.

Iteraciones: número determinado de veces que se realiza un proceso.

Máquinas de Estado Finito: Un autómata finito o máquina de estado finito es un modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce.

Programación evolutiva: La programación evolutiva (PE) es una rama de la computación evolutiva (o algoritmos evolutivos). La programación evolutiva es prácticamente una variación de los algoritmos genéticos, donde lo que cambia es la representación de los individuos. En el caso de la PE los individuos son tripletas cuyos valores representan estados de un autómata finito. Cada tripleta esta formada por: *El valor del estado actual*un símbolo del alfabeto utilizado*El valor del nuevo estado

Realidad Virtual: Simulación de un medio ambiente real o imaginario que se puede experimentar visualmente en tres dimensiones. La realidad virtual puede además proporcionar una experiencia interactiva de percepción táctil, sonora y de movimiento.

Redes Neuronales: Referidas habitualmente de forma más sencilla como redes de neuronas o redes neuronales, las redes de neuronas artificiales (RNA) son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales.