UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMÁTICA

TESIS DE GRADO

"EVOLUCIÓN DEL COMPORTAMIENTO DE UN PERSONAJE DE VIDEOJUEGO MEDIANTE EL USO DE REDES NEURONALES ARTIFICIALES"

PARA OPTAR AL TITULO DE LICENCIATURA EN INFORMÁTICA
MENCIÓN: INGENIERÍA DE SISTEMAS INFORMÁTICOS

POSTULANTE	**:**	**Wilfredo Chila Casilla**
TUTOR	**:**	**Lic. Freddy Miguel Toledo Paz**
REVISOR	**:**	**Lic. Jhonny Roberto Felipez Andrade**

LA PAZ – BOLIVIA

2009

Dedicado:

A mis Padres Esperidion y Margarita por brindarme su apoyo incondicional durante todos estos años de estudio.

A mis hermanos Zenaida, Magali, Lizeth, Veronica, Ivan y Paola por ser un gran apoyo moral y espiritual, además de estar conmigo en los días más alegres de mi vida.

AGRADECIMIENTOS

Deseo expresar mi más profundo agradecimiento a todos los que, de un modo u otro, han hecho posible este trabajo, y muy especialmente:

A Lic. Jhonny Roberto Felipez Andrade, por dirigirme en la realización de este documento, por su paciencia en las revisiones y observaciones que sirvieron de mucho para la culminación de la tesis y por el ánimo y la ayuda prestados en todo momento por conseguir transmitirme su entusiasmo en la programación de videojuegos.

A Lic. Freddy Miguel Toledo Paz, por la ayuda prestada a lo largo de todo el proceso de elaboración de este documento.

A mis amigos, por apoyarme y compartir momentos maravillosos durante mi vida universitaria.

Y por último, a mis padres y hermanos, por su colaboración en distintos aspectos del proyecto, y sobre todo, por la infinita paciencia que tienen conmigo.

RESUMEN

La inteligencia artificial es usada en los videojuegos modernos con el fin de simular un comportamiento lo más humano posible en personajes controlados por una computadora. El incremento en capacidad de proceso y almacenamiento de ordenadores y consolas, junto con la progresiva popularidad de géneros como la estrategia en tiempo real o los juegos de acción en primera persona, están impulsando importantes mejoras en este campo, por lo que empiezan a buscarse alternativas como el uso de técnicas más complejas y que sean capaces de representar mejor un comportamiento inteligente.

El presente trabajo de investigación está basado en el paradigma de la neuroevolución de topologías aumentativas (NEAT), consistente en hacer evolucionar la estructura y los pesos de una población de redes neuronales según un algoritmo genético. Así la red neuronal resultante es aquella que representa conductas más complejas, evitando además la homogeneidad y el carácter predecible de técnicas más clásicas como las máquinas de estados. Para observar los resultados se desarrollara un entorno de experimentación, en el que se harán distintas pruebas relativas al cruce de obstáculos y la detección de enemigos.

En el primer capítulo de esta tesis se plantea los objetivos y la hipótesis, la cual es la base de esta investigación, en el segundo capítulo contiene conceptos y definiciones necesarios para poder llegar al objetivo principal, el tercer capítulo presenta la construcción del modelo para dar solución al problema y la construcción del prototipo, el cuarto capítulo expone los resultados obtenidos por medio del prototipo y finalmente el quinto capítulo contiene las conclusiones y recomendaciones de acuerdo a los objetivos trazados en el presente trabajo.

Contenido

CAPÍTULO 1: INTRODUCCIÓN 2

1.1 INTRODUCCIÓN 2
1.2 PLANTEAMIENTO DEL PROBLEMA 3
1.2.1 PROBLEMA PRINCIPAL 4
1.2.2 PROBLEMAS SECUNDARIOS 4
1.3 OBJETIVOS 5
1.3.1 OBJETIVO PRINCIPAL 5
1.3.2 OBJETIVO ESPECÍFICOS 5
1.4 HIPÓTESIS 6
1.5 JUSTIFICACIONES 6
1.5.1 JUSTIFICACIÓN SOCIAL 6
1.5.2 JUSTIFICACIÓN TÉCNICA 7
1.5.3 JUSTIFICACIÓN CIENTÍFICA 7
1.6 METODOLOGÍA 7
1.6.1 MÉTODO CIENTÍFICO 7
1.6.2 PASOS DEL MÉTODO CIENTÍFICO 7
1.7 LIMITES Y ALCANCES 8

CAPÍTULO 2: MARCO TEÓRICO 11

2.1 INTRODUCCIÓN 11
2.2 MAQUINAS DE ESTADOS FINITOS 11
2.2.1 ELEMENTOS DE UNA FSM 12
2.2.2 CAMBIO DE ESTADO 12
2.2.3 REPRESENTACIÓN DE UNA FSM 12
2.2.4 TABLA DE TRANSICIÓN 13
2.3 REDES NEURONALES (NNS) 14
2.3.1 REDES NEURONALES BIOLÓGICAS 14
2.3.2 REDES NEURONALES ARTIFICIALES (ANNs) 16
2.3.3 REGLAS DE ACTIVACIÓN 18
2.3.4 RED FEEDFORWARD 19
2.3.5 TIPOS DE APRENDIZAJE 19
2.4 NEUROEVOLUCIÓN 21
2.4.1 DESCRIPCIÓN DE LA NEUROEVOLUCIÓN 21
2.5 NEUROEVOLUCIÓN DE TOPOLOGÍAS AUMENTATIVAS (NEAT) 22
2.5.1 CODIFICACIÓN GENÉTICA 23
2.5.2 MARCAJE HISTÓRICO 25
2.5.3 PROTEGIENDO LA INNOVACIÓN 27
2.5.4 MINIMIZANDO LA DIMENSIONALIDAD DE LA RED 28
2.6 ALGORITMOS GENÉTICOS 29
2.6.1 EVOLUCIÓN EN LA NATURALEZA 29
2.6.2 TERMINOLOGÍA 30
2.6.3 CONCEPTOS BÁSICOS 31
2.6.4 POBLACIÓN 31
2.6.5 FUNCIÓN APTITUD (FITNNES) 32
2.6.6 SELECCIÓN 32
2.6.7 REPRODUCCIÓN 33
2.6.8 TIPOS DE CRUZAMIENTO 33
2.6.9 MUTACIÓN 35
2.7 GENOTIPO Y FENOTIPO 36

2.8 PRUEBA DE HIPÓTESIS 37
2.8.1 PASOS PARA LLEVAR A CAVO UNA PRUEBA DE HIPÓTESIS 38
2.8.2 PRUEBA DE HIPÓTESIS PARA LA MEDIA POBLACIONAL (μ) 41

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL MODELO 44

3.1 INTRODUCCIÓN 44
3.2 METODOLOGÍA DE TRABAJO 44
3.3 IMPLEMENTACIÓN DE LA MÁQUINA DE ESTADOS 45
3.3.1 ESTADOS 45
3.3.2 FUNCIONES DE TRANSICIÓN 46
3.3.3 MAQUINA DE ESTADOS FINITOS FSMs 46
3.4 IMPLEMENTACIÓN DE REDES NEURONALES CON NEAT 48
3.4.1 EL GENOTIPO EN NEAT 48
3.4.2 INNOVACIONES 51
3.4.3 OPERADOR DE CRUCE 53
3.4.4 OPERADORES DE MUTACIÓN 55
3.4.5 GESTIÓN DE ESPECIES 56
3.4.6 QUÉ OCURRE EN UNA ÉPOCA 58
3.4.7 DEL GENOTIPO AL FENOTIPO 59
3.5 DESARROLLO DEL ENTORNO DE EXPERIMENTACIÓN 61
3.5.1 IA-FOR-GAMES 61
3.6 ARQUITECTURA DEL JUEGO 62
3.6.1 MAPA 63
3.6.2 ENTIDADES DE JUEGO 64
3.6.3 ARMAS Y PROYECTILES 65
3.6.4 TORRETAS 67
3.6.5 NPCs 69
3.6.6 VISIÓN GLOBAL DEL MUNDO 73

CAPÍTULO 4: ANÁLISIS DE RESULTADOS 75

4.1 INTRODUCCIÓN 75
4.2 FORMA DE PUNTUACIÓN 76
4.3 DESCRIPCIÓN DE RESULTADOS 77
4.3.1 REDES NEURONALES CON NEAT 77
4.3.2 MÁQUINA DE ESTADOS FINITOS 79
4.4 OBSTÁCULO I 81
4.4.1 REDES NEURONALES CON NEAT 81
4.4.2 MAQUINAS DE ESTADOS FINITOS 85
4.4.3 ANÁLISIS 86
4.5 OBSTÁCULO II 88
4.5.1 REDES NEURONALES CON NEAT 89
4.5.2 MAQUINA DE ESTADOS FINITOS 91
4.5.3 ANÁLISIS 92
4.6 DETECCIÓN DE ENEMIGOS 94
4.6.1 REDES NEURONALES CON NEAT 95
4.6.2 MAQUINA DE ESTADOS FINITOS 98
4.6.3 ANÁLISIS 98

CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES 101

5.1 CONCLUSIONES 101
5.2 RECOMENDACIONES 103

INDICE DE FIGURAS

Figura 1 – Diagrama de transición de una FSM 13
Figura 2 – Partes de una neurona 15
Figura 3 – Neurona artificial 17
Figura 4 – Distintas funciones de activación 18
Figura 5 – Diagrama de una red feedforward 19
Figura 6 – Dependencia entre los componentes de NEAT [Viscuso, 2004] 23
Figura 7 – Mapeo de genotipo a fenotipo en NEAT [Viscuso, 2004]. 24
Figura 8 – Los dos tipos de mutación estructural en NEAT [Viscuso, 2004]. 25
Figura 9 – Apareamiento de dos topologías con misma aptitud [Viscuso, 2004] 26
Figura 10 - Estructura en doble hélice del DNA 30
Figura 11 - Cromosoma de 8 genes con valores binarios [Larrañaga, 2000] 31
Figura 12 - Cruza en un Punto [Coello, 2002] 34
Figura 13 - Cruza en dos Puntos [DeJong, 2007] 35
Figura 14 - Cruza Uniforme [Mitchell, 1997] 35
Figura 15 - Operador de Mutación [Coello, 2002] 36
Figura 16 – Prueba de una cola 38
Figura 17 – Prueba para dos colas 38
Figura 18 – Estadísticos d prueba según variable usada. 40
Figura 19 - Comportamiento de los NPCs en distintas situaciones. 44
Figura 20 - Diagrama UML simplificado de la clase State. 45
Figura 21 - Codificación de estados (los estados sombreados Están desactivados). 45
Figura 22 - Diagrama UML simplificado de la clase Rule. 46
Figura 23 – Activación de una función (los estados sombreados están desactivados). 46
Figura 24 - Diagrama UML simplificado de la clase FSMs. 47
Figura 25 – Diagrama de transición de una maquina de estados finitos 47
Figura 26 – Diagrama de transición simplificado. 48
Figura 27 - Diagrama UML simplificado de la clase *Genome*, y de las estructuras *NeuronGene* y *LinkGene* 49
Figura 28 - Neurona recurrente 50
Figura 29 - Codificación de una red neuronal en NEAT (los genes sombreados están desactivados) 50
Figura 30 - Diagrama UML simplificado de la clase *InnovationHistory*, y de la estructura *Innovation*. 51
Figura 31 - Innovaciones en una red neuronal. 52
Figura 32 - Cruzando dos redes (los genes sombreados están desactivados). 54
Figura 33 - Añadiendo una neurona. 56
Figura 34 - Añadiendo un enlace 56
Figura 35 Diagrama UML simplificado de la clase *Species* 57
Figura 36 - Diagrama UML simplificado de la clase GeneticAlgorithm 58
Figura 37 - Diagrama UML simplificado de la clase NeuralNetwork, y de las estructuras Neuron y Link 59
Figura 38 - Pantalla de la aplicación 61
Figura 39 - Diagrama UML de alto nivel del entorno de experimentación 62
Figura 40 - Diagrama UML simplificado de las clases usadas en un mapa 63
Figura 41 - Diagrama UML simplificado de las entidades del juego 65
Figura 42 - Diagrama UML simplificado de las clases usadas, para implementar las armas del juego 67
Figura 43 - Representación del campo de visión de la torreta 68
Figura 44 - Diagrama UML simplificado de la clase Turret 68
Figura 45 – Sensores de obstáculos 69
Figura 46 - Sensores de enemigos 70
Figura 47 - Red neuronal de un NPC 71
Figura 48 – Elementos que componen un NPC 71
Figura 49 - Diagrama UML simplificado de la clase Bot 72
Figura 50 - Diagrama UML simplificado de la clase World 73
Figura 51 – Forma de puntuación para un solo NPC. 76

Figura 52 – IA-FOR-GAMES ventanas de diálogo..... 77
Figura 53 – Dialogo Población Info. 78
Figura 54 – Dialogo Grafica de Fitness. 79
Figura 55 – Dialogo Mejores Fenotipos. 79
Figura 56 - IA-FOR-GAMES v.2 ventanas de diálogo. 80
Figura 57 – Dialogo Maquina de Estados..... 80
Figura 58 – Ventana Aptitud. 80
Figura 59 – Mapa con obstáculos I..... 81
Figura 60 – Información sobre la población obstáculo I 83
Figura 61 – Progreso de la aptitud obstáculo I 83
Figura 62 – Mejores Fenotipos obstáculo I 83
Figura 63 – Comportamientos de los NPCs con NEAT obstáculo I..... 84
Figura 64 - Comportamiento de los NPCs con FSMs (obstáculo I) 85
Figura 65 – información de la población de NPCs con FSMs (obstáculo I) 85
Figura 66 – Mapa con obstáculos II 89
Figura 67 - Información de la población (obstáculos II) 89
Figura 68 - Progreso de la aptitud (obstáculos II) 90
Figura 69 – Mejores fenotipos (obstáculos II)..... 90
Figura 70 – Comportamiento de NPCs (obstáculos II)..... 91
Figura 71 – Información de la población de NPCs con FSMs (obstáculo II)..... 91
Figura 72 – Comportamiento de NPCs con FSMs (obstáculo II)..... 92
Figura 73 - Mapa para detección de enemigos. 94
Figura 74 – Información de la población (Detección de enemigos)..... 96
Figura 75 – Progreso de la aptitud (Detección de enemigos)..... 96
Figura 76 – Mejores Fenotipos (Detección de enemigos) 96
Figura 77 – Comportamiento de NPCs (Detección de enemigos)..... 97
Figura 78 – Comportamiento NPCs con FSMs (Detección de enemigos) 98
Figura 79 - Información de la población de NPCs con FSMs (Detección de enemigos) 98

INDICE DE TABLAS

Tabla 1 - Historial de innovaciones antes de añadir la neurona. 52
Tabla 2 - Historial de innovaciones después de añadir la neurona. 52
Tabla 3 – Resultados mapa obstáculo I 86
Tabla 4 – Resultados mapa obstáculos II 93
Tabla 5 – Resultados mapa con torretas fijas 99

1 INTRODUCCIÓN

CAPÍTULO 1: INTRODUCCIÓN

El capitulo presenta los antecedentes y las características del trabajo de investigación, por lo que se presenta el planteamiento del problema que se desea justificar y se establece la hipótesis definida para su demostración, objetivos justificaciones, alcances y aportes.

1.1 INTRODUCCIÓN

Existen muchos tipos de programas que hacen un gran uso de elementos de IA[1] (Inteligencia Artificial), como pueden ser los planificadores económicos, sistemas de reglas de negocio, sistemas de automática y control, etc. Estos elementos incluyen localización, búsqueda en árboles, resolución de problemas, toma de decisiones y aprendizaje. Pero uno de los campos del desarrollo software que más ha ido tomando de la IA ha sido, sin duda, el desarrollo de videojuegos.

La historia de la IA aplicada a los videojuegos se remonta a finales de los años 50, cuándo Arthur L. Samuel, trabajador de IBM, desarrolló un programa para la IBM 701, la primera máquina comercial de IBM, capaz de jugar a las damas mediante aprendizaje.

Existen muchas técnicas que se están aplicando en los videojuegos, como las redes neuronales o algoritmos genéticos entre otros, utilizados para ir adaptando el comportamiento de los jugadores no humanos llamados NPC[2] (Non Player Character). Estos personajes del videojuego son controlados

[1] Termino que fue acuñado en 1956 por John McCarthy, del Instituto de Tecnología de Massachusetts

[2] Frase que se usa para designar a los personajes de un videojuego , los cuales realizan sus acciones enteramente de forma autónoma sin la intervención de un jugador humano,

automáticamente por la inteligencia artificial u otra técnica. Aún así existen muchos retos en los videojuegos en los que la IA tiene mucho que decir, así como mejorar más la sensación de la inteligencia, sobre todo en entornos con muchos jugadores no humanos para que colaboren de una forma más útil entre ellos.

La aplicación de las técnicas de IA en los videojuegos se fue extendiendo, en gran medida debido a que los juegos empezaron a dejar de ser mayoritariamente juegos de jugador contra jugador, lo cual motivo la aparición de técnicas que permitieron a los usuarios sentir que estaban jugando contra otros oponentes humanos o, al menos con un nivel similar para que el juego representase un reto.

Es por medio de la IA que podemos tener experiencias de juego satisfactorias, con enemigos que realmente cuesta vencer, que se mueven e interactúan entre ellos y que den ofensivas impresionantes para lograr videojuegos mucho más atractivos.

1.2 PLANTEAMIENTO DEL PROBLEMA

En la historia de los videojuegos, pocas estructuras de datos han sido usadas tanto como la FSM (Maquina de estados finitos). A pesar de su sencillez, resulta sumamente útil para dividir un problema en partes más pequeñas, y por ende más manejables.

La principal ventaja de las maquinas de estados finitos es su simplicidad. Son fácilmente visualizables y teniendo un buen diagrama de estados, el código prácticamente se escribe solo. Además, dada su flexibilidad, pueden ser usados desde en el movimiento de los personajes hasta en los diálogos, todo es determinista, se pueden replicar directamente los errores encontrados, y la lógica del problema está centralizada. Pero la simplicidad

se convierte también en su mayor inconveniente. Al crecer exponencialmente el número de transiciones respecto al de estados, llegando a un punto en el que se vuelven inmanejables.

Las maquinas de estados finitos al constituirse en una técnica tan usada, los programadores están dejando de usar técnicas mucho más avanzadas. En los videojuegos modernos, estas maquinas de estados finitos se están abandonando en favor de técnicas de IA real, mucho más sofisticadas. A pesar de su escasa utilización.

Por lo que se desea superar las limitaciones de una maquina de estados finitos con el uso de las redes neuronales, también llamadas redes neuronales artificiales (ANNs) para distinguirlas de las biológicas, son algoritmos matemáticos con capacidad para recordar experiencias y hacerlas disponibles para su uso. Están basadas hasta cierto punto en el funcionamiento de un cerebro, a nivel tanto organizativo como funcional, resultan útiles para el reconocimiento de patrones, y para predecir tendencias en conjuntos de datos.

1.2.1 PROBLEMA PRINCIPAL

El aprendizaje es la limitación más grande de una maquina de estados finitos, ya que no está diseñada para este uso, como su nombre lo indica solo es un sistema con un número finitos de estados en los que puede estar, además solo puede estar en un estado en un momento dado. A esto hay que añadir su tendencia a crecer a lo largo del proyecto, según se intenta incluir comportamientos más especializados para tratar de modelar un comportamiento inteligente.

1.2.2 PROBLEMAS SECUNDARIOS

- Un bajo desempeño de los personajes controlados por la inteligencia artificial del videojuego debido a que el comportamiento de estos personajes en repetido y predecible.
- Los videojuegos tienden a tener una muy estrecha posibilidad de interacción ya que una vez que el jugador conoce las posibilidades preprogramadas del juego, este deja de ser atractivo para el jugador
- La simulación de comportamiento no da la sensación de inteligencia artificial.
- Los videojuegos tienen un número determinado de acciones y reacciones que tienen que ser previamente conocidas y programadas por el desarrollador.
- Se carece de estudios sobre los resultados obtenidos después de aplicar las técnicas de inteligencia artificial.

1.3 OBJETIVOS

1.3.1 OBJETIVO PRINCIPAL

El objetivo final del proyecto es el de mostrar las diferencias entre las redes neuronales artificiales y la máquina de estados finitos, en el contexto de la inteligencia artificial como técnica de control de los NPCs dentro un videojuego.

1.3.2 OBJETIVO ESPECÍFICOS

- Fortalecer y profundizar en los conceptos de inteligencia artificial aplicados a la programación de videojuegos.
- Obtener un comportamiento distinto por parte de los NPCs que hacen uso de redes neuronales artificiales en comparación de lo NPCs que usan una maquinas de estados finitos.
- Implementar NPCs cada uno de los cuales tendrá como cerebro una

red neuronal de topología evolutiva.

- Representar el movimiento de los NPCs en un mapa con obstáculos.
- Implementar NPCs controlados por medio de una maquina de estados finitos.
- Dar a conocer las ventajas y desventajas de las técnicas de maquinas de estados finitos y redes neuronales artificiales.
- Evaluar los resultados de acuerdo a la calidad de las soluciones obtenidas.
- Configurar los parámetros de la red neuronal.
- Entrenar a la red neuronal según los estímulos del mundo que la rodea.
- Visualizar los valores obtenidos en aprendizaje y pruebas para su posterior interpretación.

1.4 HIPÓTESIS

"Los NPCs controlados mediante redes neuronales artificiales presentaran diferencias significativas en el proceso de simulación del comportamiento y sensación de inteligencia, en comparación a los controlados por una máquina de estados finitos"

1.5 JUSTIFICACIONES

1.5.1 JUSTIFICACIÓN SOCIAL

La presente investigación beneficia directamente a la comunidad de desarrolladores de videojuegos, ya que de forma clara, concisa y comprensible explica el desarrollo de este tipo de software. También esta investigación va en beneficio de estudiantes, profesionales de áreas graficas, multimedia, simulación y cualquier persona interesada en incursionar dentro de esta área.

1.5.2 JUSTIFICACIÓN TÉCNICA

- Todos los juegos requieren una forma de controlar sus NPC para que sirvan como contrincante u enemigo.
- Se hará uso de las técnicas de máquina de estados finitos y redes neuronales artificiales.
- Se hará uso de una metodología de desarrollo de videojuegos.

1.5.3 JUSTIFICACIÓN CIENTÍFICA

La presente investigación sentara un antecedente acerca de los resultados obtenidos, al comparar la máquina de estados finitos y las redes neuronales como técnicas de control de los NPCs dentro un videojuego.

1.6 METODOLOGÍA

Para alcanzar los objetivos expuestos anteriormente se describe a continuación la metodología a emplear.

1.6.1 MÉTODO CIENTÍFICO

Llamamos método a una serie ordenada de procedimientos de que hace uso la investigación científica para observar un determinado fenómeno.

1.6.2 PASOS DEL MÉTODO CIENTÍFICO

- *Observación*: Consiste en la recopilación de hechos acerca de un problema o fenómeno natural que despierta nuestra curiosidad. Las observaciones deben ser lo más claras y numerosas posible, porque han de servir como base de partida para la solución.
- *Hipótesis*: Es la explicación que nos damos ante el hecho observado.

Su utilidad consiste en que nos proporciona una interpretación de los hechos de que disponemos, interpretación que debe ser puesta a prueba por observaciones y experimentos posteriores. Las hipótesis no deben ser tomadas nunca como verdaderas, debido a que un mismo hecho observado puede explicarse mediante numerosas hipótesis. El objeto de una buena hipótesis consiste solamente en darnos una explicación para estimularnos a hacer más experimentos y observaciones.

- *Experimentación*: Consiste en la verificación o comprobación de la hipótesis. La experimentación determina la validez de las posibles explicaciones que nos hemos dado y decide el que una hipótesis se acepte o se deseche.
- *Teoría*: Es una hipótesis en cual se han relacionado una gran cantidad de hechos acerca del mismo fenómeno que nos intriga. Algunos autores consideran que la teoría no es otra cosa más que una hipótesis en la cual se consideran mayor número de hechos y en la cual la explicación que nos hemos forjado tiene mayor probabilidad de ser comprobada positivamente.
- *Ley*: Consiste en un conjunto de hechos derivados de observaciones y experimentos debidamente reunidos, clasificados e interpretados que se consideran demostrados. En otras palabras la ley no es otra cosa que una hipótesis que ha sido demostrada mediante el experimento. La ley nos permite predecir el desarrollo y evolución de cualquier fenómeno natural.

1.7 LIMITES Y ALCANCES

Durante el desarrollo de la presente investigación se creará un prototipo de videojuego en 2D a modo de entorno de experimentación, que nos ayude a probar las técnicas de máquina de estados finitos y redes neuronales de una forma más cómoda y controlada. Cuando se tenga un producto

medianamente maduro y estable se empezara a realizaran las pruebas correspondientes.

Esta investigación se desenvolverá los siguientes campos:

- Investigación acerca de máquina de estados finitos.
  a. Características
  b. Diseño e implementación en los NPCs
- Investigación acerca de técnicas de inteligencia artificial
  a. Redes neuronales
  b. Algoritmos genéticos
  c. Redes neuronales de topología evolutiva
- Desarrollo de NPCs, haciendo uso de dichas técnicas que permita implementar NPCs que se adapten dinámicamente a su entorno
  a. Modelado de la red neuronal
  b. Codificación de la estructura de la red neuronal en forma de genes
  c. Creación del algoritmo genético que haga evolucionar la topología de la red neuronal
- Creación del prototipo de videojuego para probar el componente
  a. Interfaz gráfica
  b. Movimiento de los NPCs en el mapa.
- Comparación de técnicas.
  a. Maquina de estados finitos, FMS (Finite State Machine)
  b. Redes neuronales artificiales, ANNs (Artificial Neuronal Net)

En esta investigación se excluirán otras técnicas de Inteligencia artificial para videojuegos, por lo cual se pondrá énfasis solo en las presentadas.

2 MARCO TEÓRICO

CAPÍTULO 2: MARCO TEÓRICO

2.1 INTRODUCCIÓN

El objetivo del presente capítulo es establecer una base teórica adecuada para la construcción del modelo de experimentación y el cumplimiento de los objetivos trazados en el primer capítulo. En primera instancia se describirán las técnicas propuestas que son: Maquina de estados finitos y Redes neuronales artificiales, posteriormente se verá una descripción de la neuroevolución y finalmente la técnica de prueba de hipótesis.

2.2 MAQUINAS DE ESTADOS FINITOS

Las Maquinas de estados Finitos conocidas como FSM (Finite State Machines) por su traducción al inglés o también llamados Autómatas Finitos, nos sirven para realizar procesos bien definidos en un tiempo discreto. Reciben una entrada, hacen un proceso y nos entregan una salida. Notemos que estas maquinas hacen una computación [Gutiérrez Orozco, 2008].

Por lo tanto una computación es capaz de resolver un problema, sí y solo sí tiene una solución algorítmica, es decir, puede ser descrito mediante una secuencia finita de pasos bien definidos. Mediante una computación podemos encontrar soluciones a problemas que teóricamente tienen una representación algorítmica, pero que pueden necesitar tal cantidad de recursos (factores como el tiempo y el espacio de almacenamiento) que desde el punto de vista práctico no se puede llegar a la solución [Gutiérrez Orozco, 2008].

En otras palabras, imaginemos una máquina capaz de seguir una secuencia finita de pasos al introducir un conjunto de datos en ella, solo se

puede leer un dato en cada paso que se realice, por tanto el número de pasos a seguir está dado por el número de datos a introducir. Cada entrada diferente genera una salida diferente, pero siempre el mismo resultado con los mismos datos de entrada.

2.2.1 ELEMENTOS DE UNA FSM

Una FSM recibe secuencialmente una cadena de símbolos (estos símbolos pueden ser valores) y cambia de estado por cada símbolo leído o también puede permanecer en el mismo estado [Gutiérrez Orozco, 2008]. Las partes que componen una FSM son 5 y se pueden definir:

$$A = \{Q, q_0, F, \sum, \partial\}$$

Donde:

Q: Conjunto finito de estados.

q_0: Estado inicial donde $q_0 \in Q$. Debe haber uno y solo un estado inicial.

F: Conjunto de estados finales $F \subseteq Q$. El estado q_0 también puede ser final.

$\sum$: Alfabeto finito de entrada.

∂: Función de Transición $Q \times \sum \rightarrow Q$.

2.2.2 CAMBIO DE ESTADO

Supongamos que la FSM se encuentra en el estado q_l donde $q_l \in Q$, también tenemos el símbolo *a* donde $a \in \sum$. Una entrada a causa que la FSM cambie del estado q_l al estado q_k. La función ∂, llamada función de transición, describe este cambio de la forma $\partial (q_l, a) \rightarrow q_k$ de esta forma obtenemos un nuevo estado. Se entiende por transición como el proceso que hace una FSM al cambiar de estado [Gutiérrez Orozco, 2008].

2.2.3 REPRESENTACIÓN DE UNA FSM

La forma más fácil de imaginarnos una FSM es mediante un diagrama de transición. Un diagrama de transición es un grafo etiquetado con los elementos de una FSM para este caso, pero de hecho se puede representar cualquier FSM por medio de un diagrama de transición, es la forma más común de hacerlo por ejemplo (*ver Figura 1*):

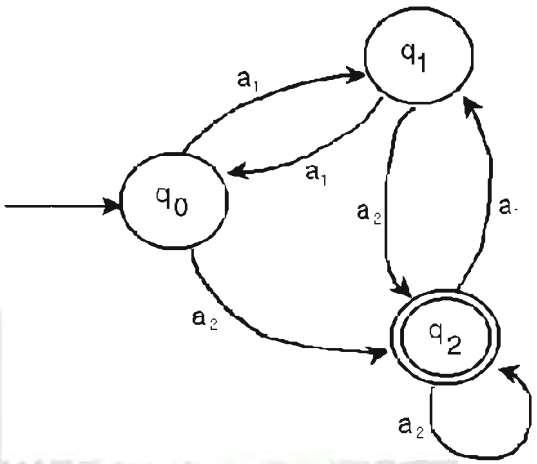


Figura 1 – Diagrama de transición de una FSM

En un diagrama de transición existe un nodo por cada estado q_i de Q. Los estados finales están encerrados en un círculo doble. El estado inicial q_0 es apuntado por una flecha que no proviene de ningún otro estado. Para cada estado q_i y un símbolo a, hay exactamente una y solo una flecha que inicia en q_i y termina en ∂ (q_i, a), es decir en q_k, la flecha es etiquetada como a. Si q_k pertenece a F decimos que la entrada es aceptada.

Debe haber exactamente una flecha saliendo de cada estado por cada símbolo a_0, a_1, a_2...a_n, por tanto todos los estados tienen el mismo número de flechas saliendo de cada uno de ellos. Con esto garantizamos que nuestra FSM pueda ser llamada Determinista. No importa el estado ni el símbolo leído, siempre hay una transición definida.

2.2.4 TABLA DE TRANSICIÓN

Para describir por completo una función de transición ∂ ocupamos una Tabla de Transición. Las columnas se etiquetan con los símbolos de entrada, la filas son etiquetadas con los estados y en las intersecciones se colocan los nuevos estados ∂ (q_i, a), suponiendo que $q_i \in Q$ es la columna y a $\in \sum$ la fila que lo intersecta. La tabla de transición de la *Figura 1* es:

	a_1	a_2
$\rightarrow q_0$	q_1	q_2
q_1	q_0	q_2
$\leftarrow q_2$	q_1	q_2

El estado inicial tiene una flecha que apunta a él, los estados finales tienen una flecha que sale de ellos y los estados que no son finales y no son el inicial no tienen flecha. En caso de que nuestro estado inicial también sea un estado final, se apuntará con una flecha doble $\leftrightarrow$. Una tabla de transición representa una función ∂ la cual recibe un símbolo y un estado. Para luego ver si se hace el cambio de estado o no.

2.3 REDES NEURONALES (NNS)

Las redes neuronales, también llamadas redes neuronales artificiales (Artificial Neural Networks o ANNs) para distinguirlas de las biológicas, son algoritmos matemáticos con capacidad para recordar experiencias y hacerlas disponibles para su uso.

Están basadas hasta cierto punto en el funcionamiento de un cerebro, a nivel tanto organizativo como funcional. Aunque no son un modelo muy realista, sí que resultan útiles para el reconocimiento de patrones, y para predecir tendencias en conjuntos de datos.

2.3.1 REDES NEURONALES BIOLÓGICAS

Los cerebros de los animales consisten esencialmente en un gran número de células nerviosas, llamadas neuronas, conectadas entre sí. Cada neurona tiene una serie de conexiones, tanto de entrada como de salida, con otras neuronas. Las conexiones de entrada a la neurona se llaman dendritas, y las de salida se denominan axones. Las dendritas de una neurona se encuentran muy próximas a los axones de otras, quedando entre ellas un espacio de unas 0.01 micras denominado espacio sináptico o sinapsis. En la *Figura 2* se puede ver un diagrama con las partes de la neurona [M. Buckland, 2002].

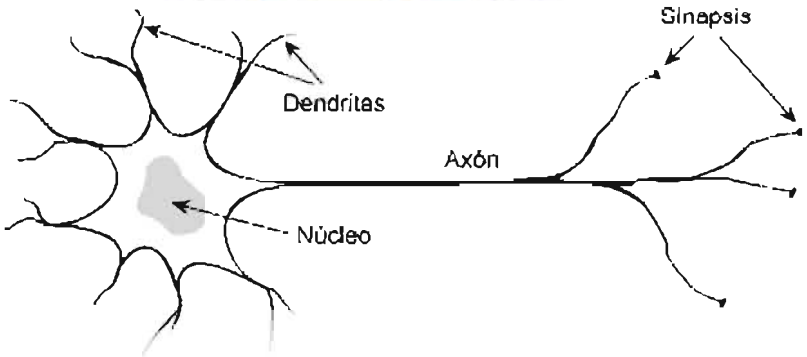


Figura 2 – Partes de una neurona

Las neuronas intercambian impulsos eléctricos que usan en un proceso electroquímico. Se reciben los impulsos provenientes de los axones de otras neuronas por medio de las dendritas. Estas uniones son conocidas como la sinapsis.

Cómo estos impulsos eléctricos se mueven sobre el cerebro es un proceso bastante complicado pero la cosa importante, hasta donde nosotros estamos interesados, es que, sólo las neuronas del cerebro se activan como una computadora moderna que opera manipulando una serie de 1s y 0s. La fuerza del impulso emitido no hace variar-sólo la frecuencia. La neurona suma todos los signos entrantes de la sinapsis de alguna manera misteriosa, y si el impulso eléctrico total excede un valor del umbral, la neurona se activa y un impulso eléctrico se envía al axón. Si el total está

menos del umbral, la neurona no se activa. Bien, la explicación bastará para nuestros propósitos [M. Buckland, 2002].

Es esta cantidad maciza de conectividad que da su poder increíble al cerebro. Definitivamente, los cerebros de los animales funcionan tomando unas entradas, reconociendo patrones en ellas, y tomando decisiones basadas en dichos patrones [M. Buckland, 2002]. El cerebro humano tiene unos 100.000 millones de neuronas, cada una con alrededor de 10.000 conexiones, y opera a unos 100Hz, una fracción de la velocidad de los procesadores actuales. Sin embargo, mientras que un procesador sólo puede manejar unas pocas instrucciones a la vez, el cerebro humano lleva a cabo millones de operaciones en paralelo [B. Schwab, 2004].

2.3.2 REDES NEURONALES ARTIFICIALES (ANNs)

El objetivo de las ANNs es emular esta capacidad de paralelización del cerebro humano. Se construyen ANNs la misma manera como las redes neuronales biológicas con la diferencia que se usan neuronas artificiales.

Una neurona artificial simplemente es una versión simplificada de una neurona real, pero simulada electrónicamente. El número de neuronas artificiales que se usan en una ANN puede variar tremendamente.

Realmente depende del trabajo que van a desarrollar. En la *Figura 3* se muestra una neurona artificial.

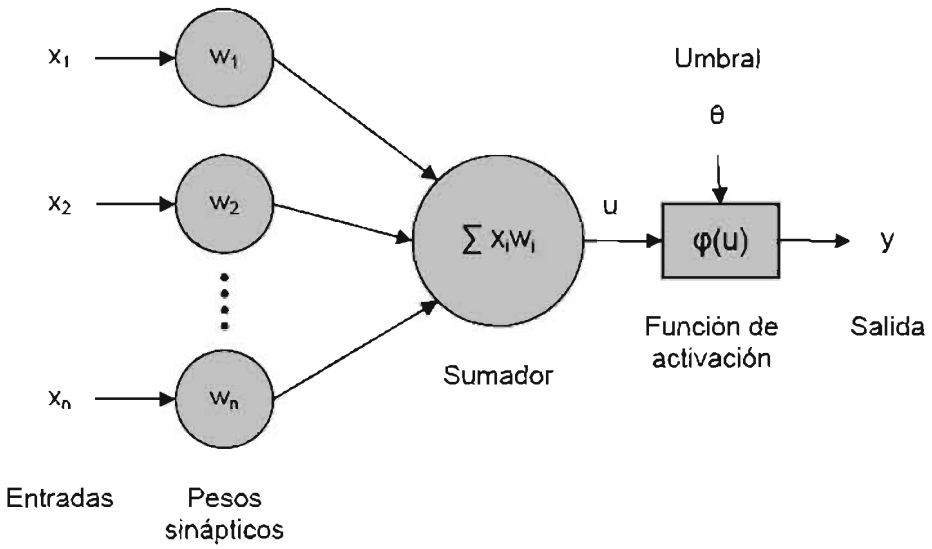


Figura 3 – Neurona artificial

Cada entrada a la neurona artificial lleva un peso asociado, y son estos pesos los que determinan la actividad de la red neuronal. Las entradas a la neurona son multiplicadas por sus respectivos pesos y después sumadas.

Un modelo simple de la *función φ (u)* seria:

$$\varphi(u) = X_1W_1 + X_2W_2 + \ldots + X_nW_n$$

Si el *resultado* de la función *φ (u)* es *mayor* que el valor *umbral*, la neurona se *activa* y emite una *señal* (1) hacia las neuronas de la capa siguiente. Pero, si por el contrario, el *resultado* es *menor* que el valor *umbral*, la neurona permanece *inactiva* (0) y no envía ninguna señal:

$$X_1W_1 + X_2W_2 + \ldots + X_nW_n \leq U \leftrightarrow \textit{Inactivación} \leftrightarrow Y = 0$$

$$X_1W_1 + X_2W_2 + \ldots + X_nW_n > U \leftrightarrow \textit{Activación} \leftrightarrow Y = 1$$

2.3.3 REGLAS DE ACTIVACIÓN

Son aquellas que determinan la activación de una neurona artificial Algunas de las funciones de activación más comunes se pueden ver en la Figura 4.

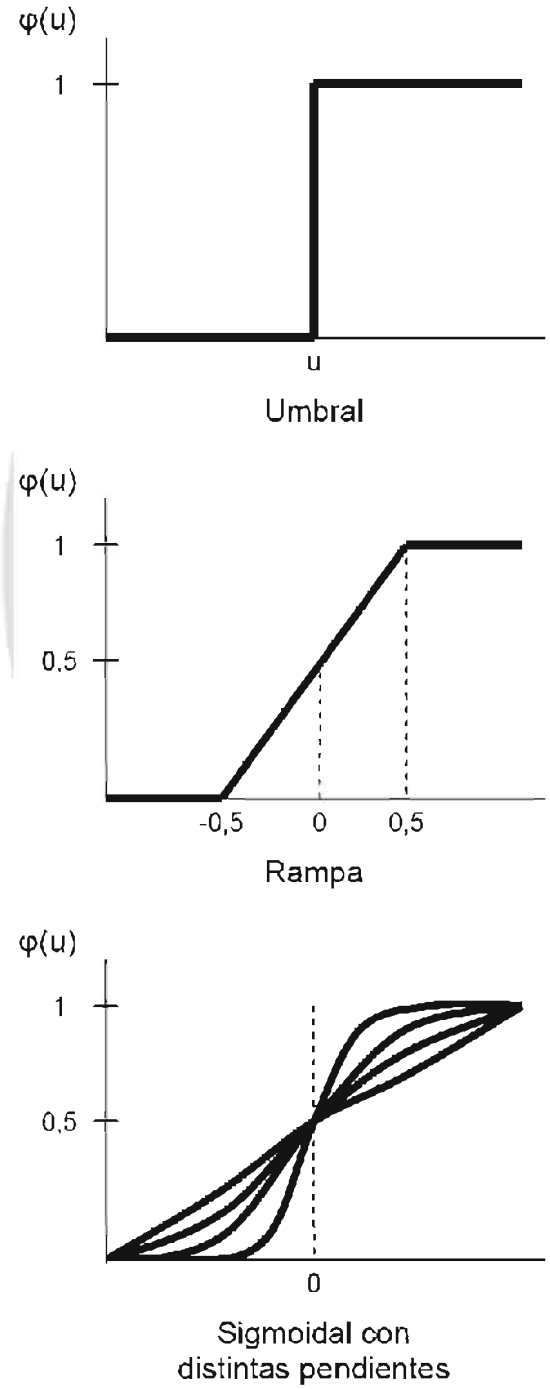


Figura 4 – Distintas funciones de activación

Igual que las neuronas del cerebro están interconectadas, las neuronas

artificiales también se conectan entre sí para formar una red neuronal. Hay muchas formas de conectar estas neuronas, pero probablemente la más sencilla de entender, y la que más nos interesa para este proyecto, es la red feedforward.

2.3.4 RED FEEDFORWARD

Este tipo de red neuronal artificial toma su nombre de la forma en que cada capa de neuronas "alimenta" su salida a la siguiente capa, hasta llegar a obtener la salida. En la Figura 5 se tiene un diagrama de una red feedforward.

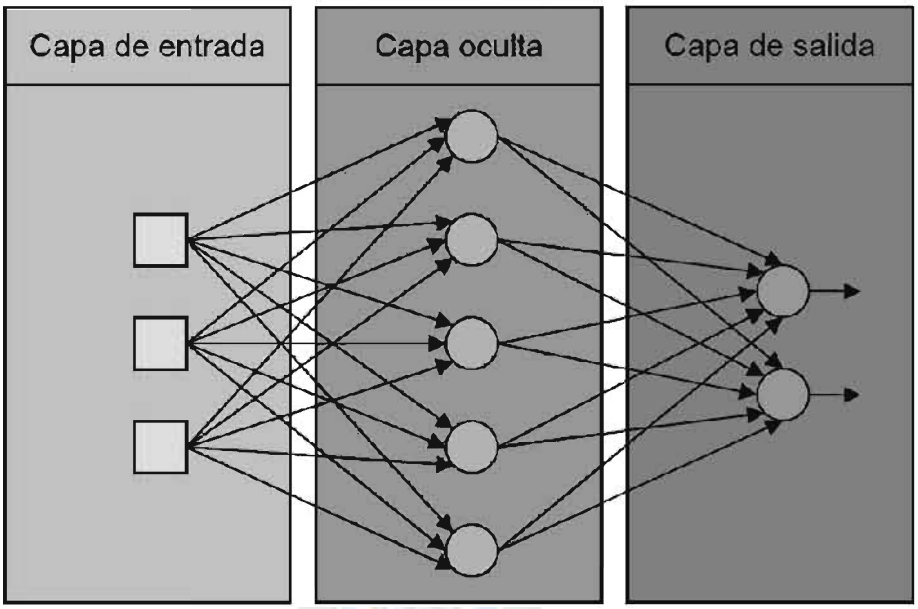


Figura 5 – Diagrama de una red feedforward

2.3.5 TIPOS DE APRENDIZAJE

El primer paso para implementar una red neuronal es elegir una estructura adecuada a la naturaleza del problema a resolver. La estructura se refiere tanto al tipo de red (feedforward, recurrente, lattice, etc.) como a su organización. No existe un método para determinar el número de neuronas

o de capas ocultas en una red feedforward. No obstante, ya que la velocidad de la red disminuye al aumentar su complejidad, convendrá hacerla tan pequeña como sea posible.

El paso siguiente al diseño de la red es su entrenamiento. Para poder aprender, las redes neuronales se sirven de un algoritmo de aprendizaje.

Estos algoritmos están formados por un conjunto de reglas que permiten a la red neuronal aprender (a partir de los datos que se le suministran), mediante la modificación de los pesos sinápticos de las conexiones entre las neuronas (recordar que el umbral de cada neurona se modificará como si fuera un peso sináptico más).

Generalmente los datos que se usan para entrenar la red se le suministran de manera aleatoria y secuencial. Los tipos de aprendizaje pueden dividirse básicamente en tres, atendiendo ha como esta guiado este aprendizaje:

- **Aprendizaje supervisado:** se introducen unos valores de entrada a la red, y los valores de salida generados por esta se comparan con los valores de salida correctos. Si hay diferencias, se ajusta la red en consecuencia.
- **Aprendizaje de refuerzo:** se introducen valores de entrada, y lo único que se le indica a la red si las salidas que ha generado son correctas o incorrectas.
- **Aprendizaje no supervisado:** no existe ningún tipo de guía. De esta manera lo único que puede hacer la red es reconocer patrones en los datos de entrada y crear categorías a partir de estos patrones. Así cuando se le entre algún dato, después del entrenamiento, la red será capaz de clasificarlo e indicará en que categoría lo ha clasificado.

2.4 NEUROEVOLUCIÓN

La Computación Evolutiva (Evolutionary Computation o EC) abarca una clase de algoritmos que pueden aplicarse a problemas de aprendizaje abiertos en inteligencia artificial.

La Neuroevolución (NE) es un dominio particular de la EC que consiste en la evolución de ANNs por medio de algoritmos genéticos. La NE ha tenido resultados exitosos en diversas tareas de aprendizaje por refuerzo [Gomez, F. y Miikkulainen, R., 1999]. La Neuroevolución busca en el espacio de comportamientos una red que tenga buena performance en una tarea determinada. Este acercamiento contribuye una alternativa a las técnicas estadísticas que intentan determinar la utilidad de determinadas acciones en estados particulares del ambiente [Kaelbling, L.P., Littman, M. y Moore, A.W., 1996] y evita problemas comúnmente presentes en el entrenamiento recurrente de redes neuronales como la complejidad computacional y la disminución del gradiente de error [Begio, 1994].

La NE es una técnica prometedora en tareas de aprendizaje por refuerzo por varias razones. Estudios previos han demostrado que la NE es más rápida y eficiente que métodos de aprendizaje por refuerzo como AHC (Adaptative Heuristic Critic) y Q-Learning en al menos un grupo de tareas tales como control de brazos robóticos, control de robots y problemas de balanceo [Moriarty, D.E. y Miikkulainen, R., 1996] [Moriarty, D.E. y Miikkulainen, R., 1997]. Dado que la NE busca un comportamiento en vez de una función de utilidad.

2.4.1 DESCRIPCIÓN DE LA NEUROEVOLUCIÓN

En los acercamientos tradicionales de NE se elige una topología para evolucionar las redes antes de que comience el experimento. Normalmente

se elige una topología con una capa oculta de neuronas donde cada neurona de dicha capa está conectada a todas las entradas y a todas las salidas de la red. La evolución entonces realiza una búsqueda en el espacio de los pesos de las conexiones de esta red totalmente conectada permitiendo modificar dichos pesos por medio de la mutación, posteriormente realizar el entrecruzamiento y así optimizar los pesos que determinan la funcionalidad de las ANNs.

2.5 NEUROEVOLUCIÓN DE TOPOLOGÍAS AUMENTATIVAS (NEAT)

Hay una gran variedad de algoritmos neuroevolutivos. Se suelen distinguir dos grupos principales: aquellos que sólo evolucionan los pesos sinápticos de una red de topología predefinida (NE), y aquellos que, además de los pesos, evolucionan también la topología.

NEAT (NeuroEvolution of Augmenting Topologies) es una implementación particular de un sistema TWEANN (Topology and Weigth Evolving Artificial Neural Networks) desarrollado en la Universidad de Texas por [Stanley Kenneth O. y Miikkulainen Risto, 1999]. Los TWEANN agrupan sistemas de neuroevolución de redes neuronales que tiene en cuenta tanto los pesos como la topología de las redes en el proceso evolutivo.

NEAT cuenta con componentes fuertemente interrelacionados (Figura 6) cuyas características principales son: El proceso comienza desde estructuras mínimas que crecen solo cuando es beneficioso, utiliza marcas históricas en los genes para lograr el entrecruzamiento (*crossover*) preciso y protege la innovación topológica mediante la especiación.

Esta combinación de elementos ha permitido a NEAT resolver problemas difíciles de aprendizaje por refuerzo más eficientemente que otros métodos de neuroevolución [Stanley Kenneth O. y Miikkulainen Risto, 1999].

NEAT ha superado varias pruebas que demuestran que cada componente del sistema contribuye significativamente a su performance global [Stanley Kenneth O. y Miikkulainen Risto, 1999].

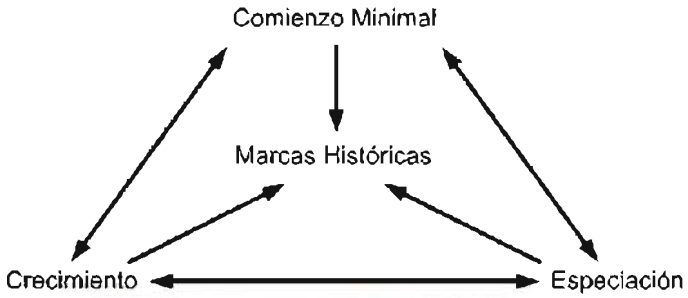


Figura 6 – Dependencia entre los componentes de NEAT [Viscuso, 2004]

2.5.1 CODIFICACIÓN GENÉTICA

El esquema de codificación genética de NEAT está diseñado para permitir que los genes homólogos se puedan alinear fácilmente cuando un par de genomas realizan el cruzamiento (*crossover*) durante el apareamiento. Los genomas son representaciones lineales de la conectividad de la red. Cada genoma incluye una lista de genes conectivos cada uno de los cuales se refiere a dos genes nodales en conexión. Los genes nodales proveen una lista de entradas, nodos ocultos, y salidas que pueden ser interconectadas. Cada gen conectivo especifica un nodo de entrada y nodo de salida, el peso de la conexión, un valor de habilitación y un número de innovación que permite encontrar los genes homólogos [Viscuso, 2004].

En la Figura 7 se muestra un genotipo que produce el fenotipo indicado. Hay 3 nodos de entrada, un nodo oculto, un nodo de salida, siete definiciones de conexiones (con una de ellas recurrente). El segundo gen esta deshabilitado por lo que la conexión que especifica (entre los nodos 2 y 4) no se expresa en el fenotipo.

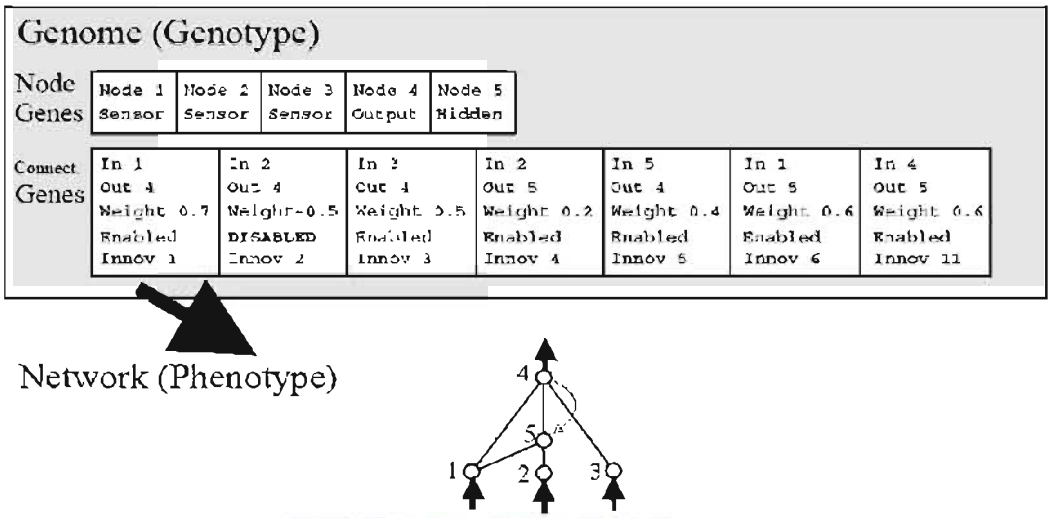


Figura 7 – Mapeo de genotipo a fenotipo en NEAT [Viscuso, 2004].

La mutación en NEAT puede cambiar ambos los pesos de las conexiones y la estructura de la red. Los pesos de las conexiones mutan como en todo sistema de NE, donde cada conexión es afectada o no en cada generación. Las mutaciones estructurales ocurren de dos formas. Cada mutación expande el tamaño del genoma agregando genes. En la mutación de agregado de conexión una única nueva conexión se incorpora a la red con su peso seleccionado al azar y une dos nodos previamente desconectados. En la mutación de agregado de nodo se rompe una conexión preexistente y se agrega un nuevo nodo en lugar de la misma.

Luego se deshabilita la antigua conexión y se agrega dos nuevas conexiones al genoma. La nueva conexión que entra al nodo de aparición reciente recibe un peso de 1, y la otra conexión saliente recién creada recibe el mismo peso que la conexión original rota. Este método de agregado de nodos se elige con la intención de minimizar el efecto inicial de la mutación. La no linealidad que se genera en la conexión cambia la función ligeramente pero la ventaja es que de esta forma pueden agregarse nuevos nodos durante la evolución en oposición al agregado tardío de estructura de esta manera, debido a la especialización, la red tendrá tiempo de optimizarse y hacer uso de la nueva estructura. (*Ver Figura 8*).

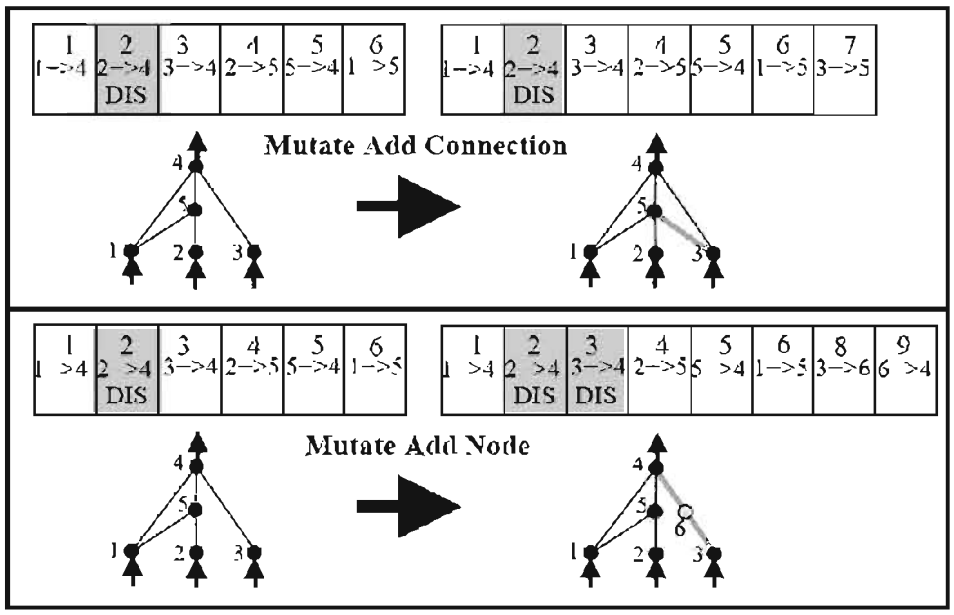


Figura 8 – Los dos tipos de mutación estructural en NEAT [Viscuso, 2004].

A través de los mecanismos de mutación descritos, los genomas en NEAT se agrandan gradualmente y presentan tamaños variables, en ocasiones con distintas conexiones en las mismas posiciones. Entonces como realiza NEAT en entrecruzamiento de manera precisa con topologías diferentes y distintas combinaciones de pesos.

2.5.2 MARCAJE HISTÓRICO

Existe información desaprovechada en el proceso de evolución que nos dice exactamente qué genes se aparean entre si entre cualquiera de los individuos de una población topológicamente variada. Tal información es el origen histórico de cada gen. Dos genes con el mismo origen histórico deben representar la misma estructura (aunque posiblemente con diferentes pesos), ya que ambos han derivado del mismo gen ancestral en algún punto en el pasado. Así, todo el sistema requiere para conocer la concordancia de los genes es llevar un registro del origen histórico de cada gen en el sistema [Viscuso, 2004]. Esta tarea requiere de poco cómputo. Cada vez que aparezca un nuevo gen (por mutación estructural), se

incrementa un número de innovación global y se asigna a ese gen. Los números de innovación representan así una cronología de la aparición de cada gen en el sistema. En la *Figura 9* puede verse en número de innovación en la parte superior de cada gen.

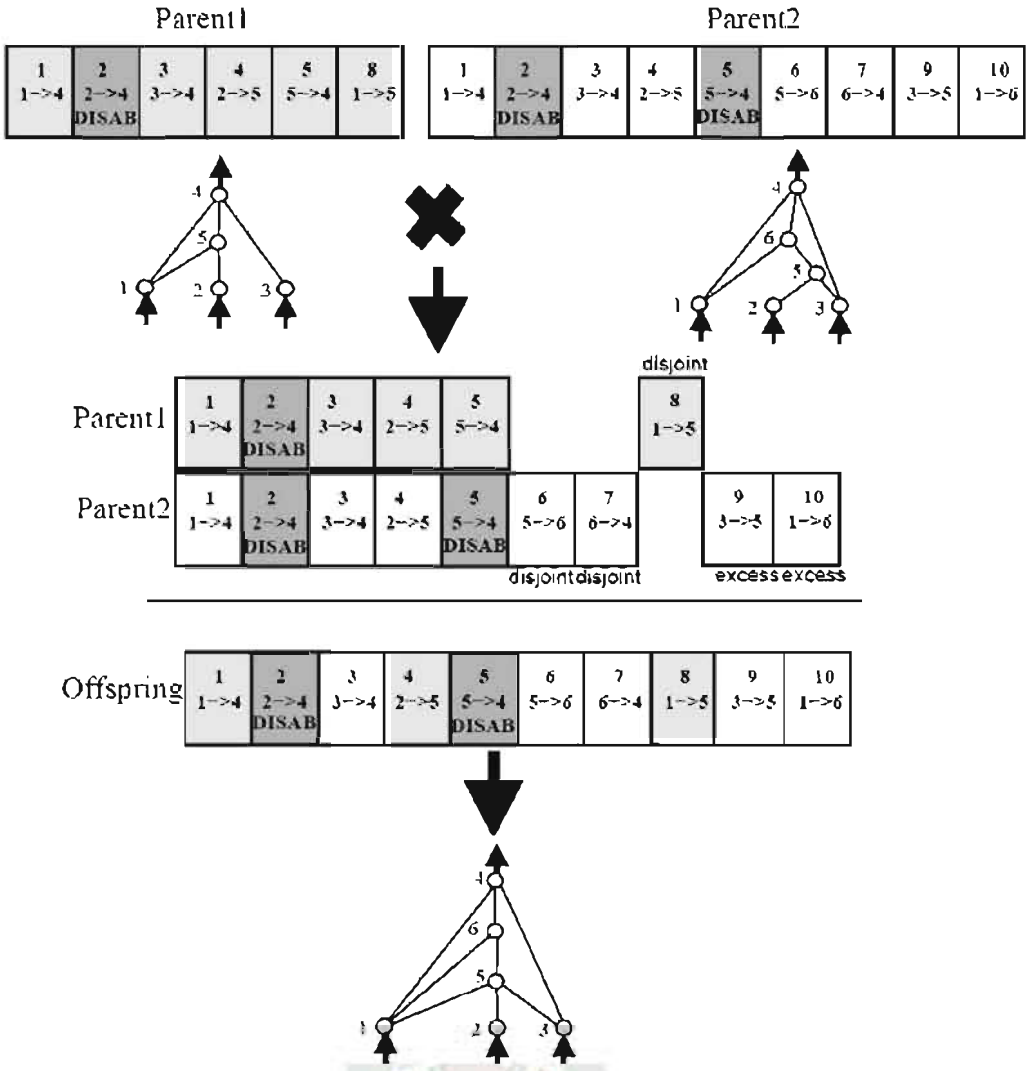


Figura 9 – Apareamiento de dos topologías con misma aptitud [Viscuso, 2004].

El marcaje histórico provee a NEAT de una capacidad poderosa. El sistema sabe exactamente cuales genes se aparean entre sí. Durante el entrecruzamiento se alinean los genes de de ambos genomas cuyos números de innovación coinciden. Estos genes se denominan genes homólogos o concordantes. Los genes con números de innovación distintos

son genes disyuntos o en exceso según se encuentren dentro o fuera del rango de números de innovación de su gen homologo respectivamente. Estos últimos representan estructura que no está presente en el otro genoma. Al generar la progenie se eligen genes al azar de cada progenitor de entre los genes homólogos mientras los genes disyuntos o en exceso siempre se eligen de padre más apto, en el caso que los dos padres tengan la misma aptitud los genes disyuntos o en exceso también se heredan al azar. De esta forma las marcas históricas permiten a NEAT realizar el proceso de entrecruzamiento utilizando genomas lineales sin la necesidad de realizar un costoso análisis topológico [Viscuso, 2004].

El agregado de nuevos genes a la población y el apareo preciso de genomas que representan diferentes estructuras permiten al sistema formar poblaciones con topologías variadas. Sin embargo, tal población no puede mantener la innovación topológica por sí sola. Como las estructuras más pequeñas se optimizan más rápido que las grandes y como el agregado de nuevos nodos y conexiones comúnmente decrementa la aptitud de una red en un principio, las estructuras recientemente aumentadas tiene una baja posibilidad de sobrevivir más de una generación aunque las innovaciones que representan podrían ser cruciales para resolver la tarea a largo plazo. La solución a este problema es proteger la innovación mediante la especiación de la población [Viscuso, 2004].

2.5.3 PROTEGIENDO LA INNOVACIÓN

La especiación de la población permite a los organismos competir con sus propios nichos en vez de con toda la población. De esta forma las innovaciones topológicas son protegidas en nuevos nichos donde cuentan con el tiempo para optimizar sus estructuras compitiendo en su propio espacio. La idea consiste en dividir la población en especies de manera que las topologías similares sean de la misma especie. Esta tarea parece

tratarse de un problema de concordancia de topologías. Sin embargo el marcaje histórico ofrece una solución eficiente a la problemática. La cantidad de genes disyuntos y en exceso entre dos genomas es una medida natural de su distancia de compatibilidad. Mientras más disyuntos son los genomas menos historia evolutiva comparten y por lo tanto son menos compatibles entre sí. Así, puede medirse la distancia de compatibilidad de diferentes estructuras en NEAT como una combinación lineal simple de la cantidad de genes disyuntos y en exceso y de la diferencia promedio de los pesos de los genes homólogos, incluidos los genes deshabilitados [Viscuso, 2004].

NEAT mantiene una lista ordenada de especies. En cada generación, los genomas se ubican en especies de forma secuencial. Cada especie en existencia está representada por un genoma al azar dentro de la misma tomado de la generación previa. Un genoma dado g en la generación actual es ubicado en la primera especie donde g es compatible con el genoma representativo de esa especie. De esa forma, las especies no se solapan. Si g no es compatible con ninguna especie actual se crea una nueva con g como su representante [Viscuso, 2004].

2.5.4 MINIMIZANDO LA DIMENSIONALIDAD DE LA RED

NEAT predispone la búsqueda hacia espacios dimensionales minimales al comenzar con una población uniforme de redes sin nodos ocultos (todas las entradas conectadas directamente a todas las salidas). Se introduce nueva estructura incrementalmente a medida que las mutaciones ocurren y solo sobreviven aquellas estructuras que resultan útiles a lo largo de las evaluaciones de aptitud. En otras palabras la elaboración estructural que introduce el sistema está siempre justificada. Como la población arranca minimal la dimensionalidad del espacio de búsqueda se minimiza [Viscuso, 2004].

2.6 ALGORITMOS GENÉTICOS

Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin.

Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas. [John Holland, 1999]

Un **algoritmo genético** consiste en una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuáles 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, principal defecto de otros métodos para solucionar problemas, como los Sistemas Expertos.

2.6.1 EVOLUCIÓN EN LA NATURALEZA

El proceso de evolución en la naturaleza funciona del siguiente modo. Para sobrevivir, toda criatura debe ser capaz de reproducirse. La reproducción

consiste en ejecutar las reglas necesarias para generar un nuevo organismo. Estas reglas son almacenadas en cadenas de DNA llamadas cromosomas, que se encuentran en toda célula que forma parte del individuo. En la Figura 10 se ve la estructura en doble hélice del DNA.

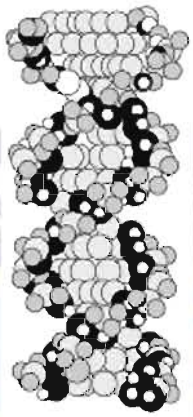


Figura 10 - Estructura en doble hélice del DNA

2.6.2 TERMINOLOGÍA

El concepto básico que se usa en los AG es que se basa en la búsqueda de mejores soluciones de problemas, de la misma forman en que las especies evolucionan a fin de adaptarse mejor a su habitad. Al igual que la biogenética, este proceso ocurre de forma iterativa y evolucionan a lo largo del tiempo. Por lo tanto, la jerga de los AG a menudo habla de la analogía de la supervivencia del mas apto. [Jesús Alfonso López, 2000].

- **Gen:** Partícula de los cromosomas que producen la aparición de caracteres hereditarios.
- **Cromosoma:** Más o menos por la misma época, el biólogo alemán Walther Flemming describió los cromosomas, como ciertos filamentos en los que se agregaba la cromatina del núcleo celular durante la división; poco más adelante se descubrió que las células de cada especie viviente tenía un número fijo y característico de cromosomas.

- **Genética:** Ciencia que estudia los fenómenos relativos a la herencia.
- **Evolución:** Serie de transformaciones sucesivas que han sufrido los seres vivos desde los tiempos geológicos.

2.6.3 CONCEPTOS BÁSICOS

La representación de las soluciones candidatas en los AG es mediante una cadena de valores paramétricos (binario, alfabéticos). Si bien el alfabeto utilizado para representar los individuos no necesariamente tiene que estar constituido por (0.1). En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina fenotipo. Un gen se identifica por la posición que ocupa en el cromosoma la cual representa un atributo del individuo, el cual puede tomar diferentes valores. Como se observa en la *Figura 11*.

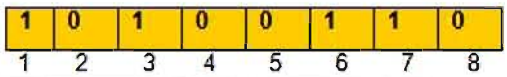


Figura 11 - Cromosoma de 8 genes con valores binarios [Larrañaga, 2000]

Los cromosomas se conocen como individuos específicos de una población dada. Un cromosoma es un arreglo de genes, cuya cantidad es fija e igual para todos los individuos de la población.
Una de las ideas más importantes es definir estructuras *admisibles* en sentido que estén bien definidas y puedan ser evaluadas.

2.6.4 POBLACIÓN

En cuestión del tema computacional, un AG mapea un determinado problema en un conjunto de cadenas (cromosomas), hay que tomar muy en cuenta el tamaño de la población dado que esto influye de gran manera en la representación de la solución potencial o candidata.

La población siempre se escoge generando poblaciones al azar, o se crea una población inicial de soluciones potenciales para un problema en particular.

2.6.5 FUNCIÓN APTITUD (FITNNES)

En el comportamiento de los AGs son dos aspectos que resultan cruciales, una determinación de una adecuada función de adaptación o función aptitud y la codificación utilizada. La función objetivo es la base que determina la probabilidad de la sobre vivencia. Esta función debe ser diseñada de acuerdo a cada problema de manera específica. La regla, general para construir una buena función aptitud es que ésta debe reflejar el valor del individuo de una manera "real", pero en muchos problemas de optimización combinatoria, donde existe gran cantidad de restricciones, buena parte de los puntos del espacio de búsqueda representan individuos no válidos. Se tiene que tener un balance entre una función que haga diferencias muy grandes y diferencias pequeñas.

2.6.6 SELECCIÓN

Una parte fundamental del funcionamiento de un AG es, sin lugar a dudas, el proceso de selección de candidatos a reproducirse. En el algoritmo genético este proceso de selección suele realizarse de forma probabilística (es decir, aun los individuos menos aptos tienen la posibilidad de sobrevivir), a diferencia de las estrategias evolutivas, en las que la selección es extintiva (los menos aptos tienen cero probabilidades de sobrevivir).

Las técnicas de selección usadas en algoritmos genéticos pueden clasificarse en tres grandes grupos:

Selección proporcional: Este nombre a un grupo de esquemas de selección originalmente propuestas por Holland en las que se eligen a los individuos de acuerdo a su contribución de aptitud con respecto al total de población.

Selección Mediante Torneo: En esta selección se escogen grupos de individuos y de cada grupo se selecciona como padre al mejor, es decir el que gana ese torneo. El tamaño de los grupos entre los que es muy pequeño comparado con el tamaño de la población.

Selección de Estado Uniforme: En esta selección los padres se eligen aleatoriamente entre la generación actual. Todos los individuos de la población, independiente de su aptitud, tienen equiprobabilidad de ser elegido. Esta técnica no garantiza que la población tienda a mejores soluciones puesto que no discrimina los individuos con los genes malos.

2.6.7 REPRODUCCIÓN

En esta fase de reproducción, se seleccionan una cantidad de individuos de la población para ser recombinados, los que forman descendientes que luego continuaran la siguiente generación. Los padres serán seleccionados al azar, usando un método que favorece a los individuos mejor adaptados. Después de ser escogidos los padres, sus cromosomas se mezclan y combinan usando operadores genéticos: Cruzamiento y Mutación. [Jesús Alfonso López. 2000].

2.6.8 TIPOS DE CRUZAMIENTO

Cruzamiento: El operador de cruzamiento se basa en generar dos nuevos hijos a partir de dos cadenas de cromosomas padre. Este cruzamiento de dos cromosomas se asemeja a la reproducción sexual de las especies, ya

que se realiza el intercambio de información digital contenida en los genes [Gastón Crevillén y David Díaz, 2001], el cual permite que las próximas generaciones hereden sus características. La probabilidad de cruzamiento aplicada debe ser mayor que la probabilidad de mutación, en caso contrario el cruzamiento será erróneo. Existen tres técnicas básicas de cruzamiento:

Cruzamiento en un punto: Este tipo de cruce se basa en un punto para lo cual se seleccionan dos padres y se corta en la cadena de cromosomas en una posición o punto escogida al azar, para producir dos subcadenas iníciales y dos subcadenas finales. Después se intercambian las subcadenas finales, produciéndose dos nuevos cromosomas completos. Este operador se conoce como operador de cruce basado en un punto. Habitualmente el operador de cruce no se aplica a todos los pares de individuos que han sido seleccionados para emparejarse, sino que se aplica de manera aleatoria, normalmente con una probabilidad comprendida entre 0.5 y 1.0. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando los padres (Figura 12).

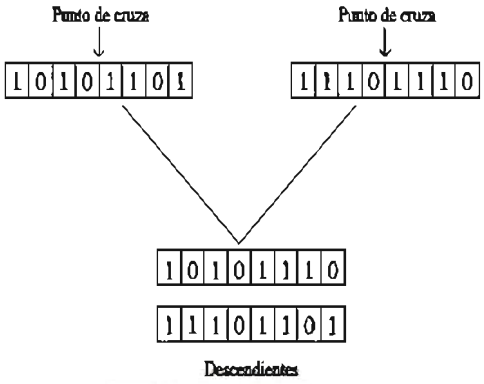


Figura 12 - Cruza en un Punto [Coello, 2002]

Cruzamiento en dos Puntos: DeJong fue el primero en implementar una cruza de n puntos, como generalización de la cruza de un punto. Básicamente es lo mismo que la cruza de un solo punto solo que en esta se corta en dos puntos, las cuales se intercambian, formando así a los descendientes, como se muestra a continuación:

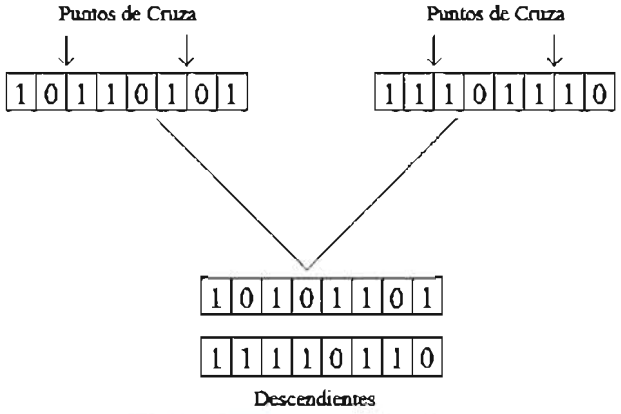


Figura 13 - Cruza en dos Puntos [DeJong, 2007]

Cruzamiento Uniforme: Esta técnica fue propuesta básicamente por Achley, aunque suele atribuirse a Syswerda. En este caso, se trata de una cruza de n puntos, pero en la cual el número de puntos de cruza no se fija previamente. La cruza uniforme tiene un mayor efecto disruptivo que cualquiera de las dos anteriores. Cuando existe un 1 en la máscara de cruza el gen es copiado del primer padre, mientras que exista un 0 el gen es copiado del segundo padre, como se muestra continuación:

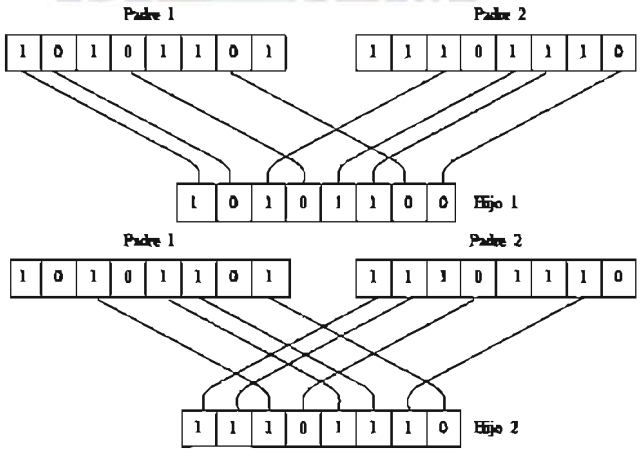


Figura 14 - Cruza Uniforme [Mitchell, 1997]

2.6.9 MUTACIÓN

El operador de mutación se considera como un operador secundario en los AG canónicos. Es decir, que su uso es menos frecuente que el de la cruza,

en la práctica, suelen recomendar porcentajes de mutación de entre 0.001 y 0.01 para la representación binaria.

Algunos investigadores, sin embargo, han sugerido usar porcentajes altos de mutación al inicio de la búsqueda, y luego decrementarlos exponencialmente, esto favorece al desarrollo del AG. Otros autores que $p_m = 1/L$ (donde L es la longitud de cadena cromosómica), es un límite inferior para el porcentaje de mutación.

La mutación es básicamente el intercambio de genes (bits). Como se puede observar a continuación:

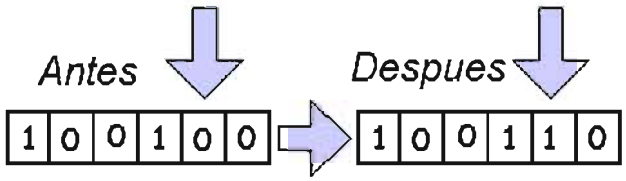


Figura 15 - Operador de Mutación [Coello, 2002]

2.7 GENOTIPO Y FENOTIPO

El genotipo es el contenido genético (el genoma específico) de un individuo, en forma de una cadena cromosómica. Junto con la variación ambiental que influye sobre el individuo, codifica el fenotipo del individuo. De otro modo, el genotipo puede definirse como el conjunto de genes de un organismo y el fenotipo a la expresión del genotipo en un determinado ambiente [Wikipedia1, 2005]. Los rasgos fenotípicos incluyen rasgos tanto físicos como conductuales. Es importante destacar que el fenotipo no puede definirse como la "manifestación visible" del genotipo [Wikipedia2, 2005]. La interacción entre el genotipo y el fenotipo ha sido descrita usando la simple ecuación que se expone a continuación:

Ambiente + Genotipo = Fenotipo

2.8 PRUEBA DE HIPÓTESIS

Para realizar una prueba de hipótesis es necesario conocer los siguientes conceptos.

- ***Hipótesis*** – es una teoría o suposición tentativa para explicar ciertos hechos y llevar a la investigación de otros.
- ***Prueba de hipótesis*** – proceso que nos lleva a tomar una decisión entre dos hipótesis opuestas.
- ***Hipótesis nula (H_0)*** – es la teoría tentativa sobre un parámetro que va a prevalecer hasta que haya suficiente evidencia para refutarla. (El acusado es inocente hasta que se demuestre lo contrario).
- ***Hipótesis alternativa (H_1)*** – es la hipótesis opuesta a la H_0 (El acusado es encontrado culpable.)
- ***Error tipo I*** – es rechazar la hipótesis nula cuando esta es realmente cierta. (Declarar al acusado culpable cuando en realidad es inocente.)
- ***Error tipo II*** – es no rechazar la hipótesis nula cuando esta es realmente falsa. (Declarar al acusado inocente cuando en realidad es culpable.)
- ***Nivel de significancia (α)*** – es la probabilidad de rechazar la hipótesis nula cuando esta es realmente cierta = P (error tipo I).
- ***Potencia de la prueba ($1-\beta$)*** -es la probabilidad de rechazar la hipótesis nula cuando esta es realmente falsa = 1-P (error tipo II).
- ***Estadístico de prueba*** – función de los datos que se usa para tomara la decisión de rechazar o no H_0.
- ***Región critica*** – área o región donde se rechaza la H_0. Es el conjunto de valores del estadístico de prueba para los que rechaza H_0.
- ***Valor critico*** – valor que define donde comienza la región critica.

	Situación Real	
Decisión	H_0 es cierta	H_0 es falsa
H_0 no se rechaza	Correcto	Error tipo II
H_0 se rechaza	Error tipo I	Correcto

2.8.1 PASOS PARA LLEVAR A CAVO UNA PRUEBA DE HIPÓTESIS

Hay 3 formas o combinaciones en que se pueden establecer H_0 y H_1:

$H_0: \mu=\mu_0$ $H_0: \mu\leq\mu_0$ $H_0: \mu\leq\mu_0$

$H_1: \mu\neq\mu_0$ $H_1: \mu<\mu_0$ $H_1: \mu>\mu_0$

Figura 16 – Prueba de una cola

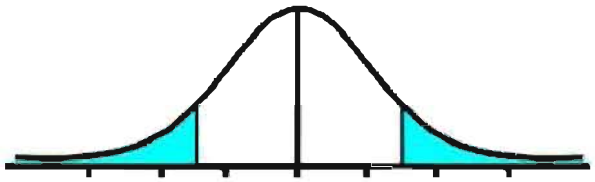


Figura 17 – Prueba para dos colas

El área sombreada en cada grafica corresponde a la región crítica.

- La H_0 y la H_1 siempre se debe escoger de forma que el peor de los 2 posibles errores sea el error tipo I, ya que este es más fácil de controlar que el error de tipo II.

  Determinar la distribución de probabilidad, el nivel de significancia y

la región critica.

La distribución de probabilidad depende del parámetro sobre el que se está llevando a cabo la hipótesis y de las características e información disponible sobre la muestra y/o población.

El nivel de significancia (α) se escoge de acuerdo al riesgo de rechazar equivocadamente la H_0 que se desee. Los niveles más usados son 0.10, 0.05 y 0.01.

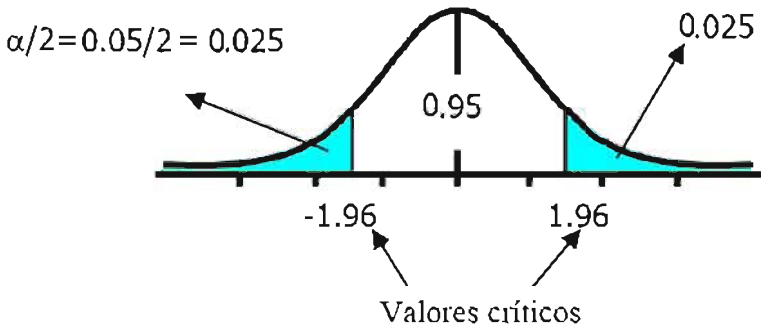
La región critica depende del tipo de hipótesis (una o dos colas), la distribución de probabilidad y el nivel de significancia.

El valor crítico se determina usando el nivel de significancia, una vez que se haya determinado donde se encuentra la región critica.

Ejemplo

Como el nivel de significancia es del 5% = 0.05 se calcula la región critica.
1 – 0.05/2 = 0.975, valor que se busca en la tabla de la distribución Normal, obteniendo el valor de $Z_{0.975} = 1.96$.

Prueba de dos colas α=0.05
(Usando la distribución Normal)



- Obtener la muestra y calcular los estadísticos.
- Calcular el estadístico de prueba.

Parámetro en H_0 (una población)	Información	Distribución	Estadístico de prueba
μ	σ	Normal (Z)	$\frac{\bar{x}-\mu_0}{\sigma/\sqrt{n}}$
	s y población Normal	t-Student (t)	$\frac{\bar{x}-\mu_0}{s/\sqrt{n}}$ *
π	$n\pi \geq 5$ y $n(1-\pi) \geq 5$	Normal (Z)	$\frac{\hat{p}-\pi_o}{\sqrt{\pi_o(1-\pi_o)/n}}$
σ^2	población Normal	Ji-cuadrado($\chi 2$)	$\frac{(n-1)s^2}{\sigma_0^2}$

Figura 18 – Estadísticos d prueba según variable usada.

*Si estamos usando la tabla de la distribución T-Student, cuando n>=30 o n>100 tenemos que usar la tabla de la distribución Normal como aproximación y la distribución en el muestreo de la media seguiría una distribución t con n-1 grados de libertad.

- Tomar una decisión.

  Para tomar una decisión debemos usar una de las dos opciones siguientes:

  - Comparar los valores críticos con el valor del estadístico de prueba.

    Si este último cae dentro de la región crítica entonces rechazamos H_0. De lo contrario decimos que no hay suficiente evidencia para rechazar H_0.

  - Comparar el p-value con el nivel de significancia.

El p-value es la probabilidad de obtener un valor muestral (media o proporción o…) aun más extremo del que obtuvimos. Es el nivel de significancia observado.

p-value < α **se rechaza H_0**

p-value ≥ α **no se rechaza H_0**

2.8.2 PRUEBA DE HIPÓTESIS PARA LA MEDIA POBLACIONAL (μ)

Una propaganda sobre un modelo de carro dice que estos carros hacen 20 millas/galón. Al ver el anuncio un competidor considera que están sobre-estimando las millas/galón al que hace ese modelo.

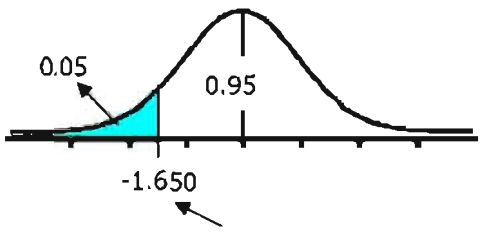
Para tratar de probar a un nivel de significancia de .05 si el anuncio tiene razón se decide escoger una muestra de 45 carros de este modelo.

Formular la hipótesis:

H_0: μ ≥ 20, Este modelo de carro rinde por lo menos de 20 millas/galón.
H_1: μ < 20, Este modelo de carro rinde menos de 20 millas/galón.

Determinar la distribución de probabilidad y la región critica.

Como el nivel de significancia es del 5% = 0.05 se calcula la región critica.
1 – 0.05 = 0.95, valor que se busca en la tabla de la distribución Normal, obteniendo el valor de $Z_{0.95}$ = 1.65.



Valor crítico (de la calculadora o computadora) . Si está usando tablas lo aproxima con -1.645 de la tabla de Z.

Usamos t (asumiendo que tenemos una población Normal)

Obtener la muestra y calcular los estadísticos:

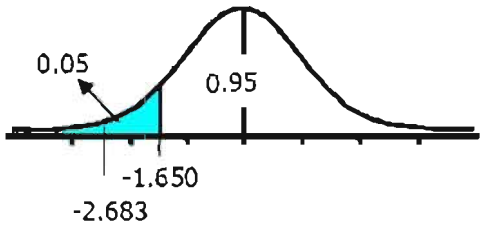
$$\bar{X} = 16 \qquad y \qquad s = 10$$

$$S = \sqrt{\frac{\sum (x - \bar{x})^2}{(n-1)}}$$

$$\overline{X} = \frac{\sum x_i}{n}$$

Calcular el estadístico de prueba:

$$t = \frac{16 - 20}{10/\sqrt{45}} = -2.683$$



Tomar una decisión:

Como cae en la región crítica rechazamos H_0. El competidor parece estar en lo correcto en que están sobre estimando las millas/galon de este modelo de carro.

3

Análisis y diseño del modelo

CAPÍTULO 3: Análisis y diseño del modelo

3.1 INTRODUCCIÓN

Conociendo cómo funciona una Maquina de Estados finitos, una red neuronal, un algoritmo genético, el concepto de neuroevolución, y los fundamentos del paradigma NEAT, estamos en condiciones de desarrollar nuestro componente. Analizaremos el genotipo y el fenotipo de una red neuronal, los operadores aplicados durante el proceso de evolución, y conceptos como las innovaciones o la gestión de especies.

Finalizado el componente, pasaremos crear un entorno de experimentación adecuado para probarlo. Este entorno simulará un videojuego de acción en dos dimensiones, en el que pondremos a prueba a los NPCs en distintas situaciones.

3.2 METODOLOGÍA DE TRABAJO

En el entorno de experimentación se estudiará el comportamiento de los NPCs en distintas situaciones utilizando en primera instancia una maquina de estados finitos y a continuación haciendo uso de una red neuronal artificial evolutiva *(Figura 19)*.

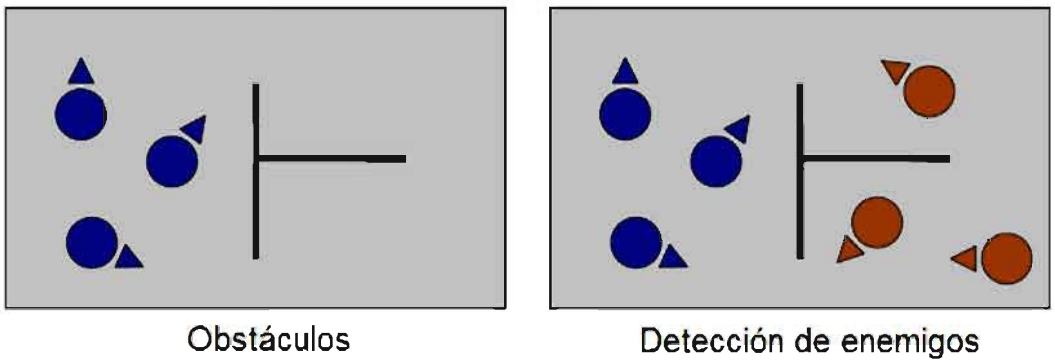


Figura 19 - Comportamiento de los NPCs en distintas situaciones.

3.3 IMPLEMENTACIÓN DE LA MÁQUINA DE ESTADOS

Explicaremos el código que compondrá la máquina de estados finitos la cual servirá para controlar las acciones realizadas por nuestros NPCs, las acciones serán asignadas a estados específicos y activadas mediante funciones de transición.

3.3.1 ESTADOS

Los estados vienen representados por la clase *State*, cada uno con un identificador y un nombre específico y una lista de reglas las cuales pueden variar en número. El método *isActivate* nos indica si el estado está actualmente ejecutándose. En la *Figura 20* se puede ver el diagrama UML de la clase *State*.

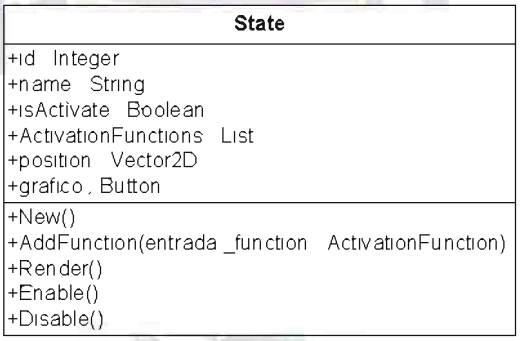


Figura 20 - Diagrama UML simplificado de la clase State.

En la *Figura 21* puede verse la codificación de los estados a utilizar.

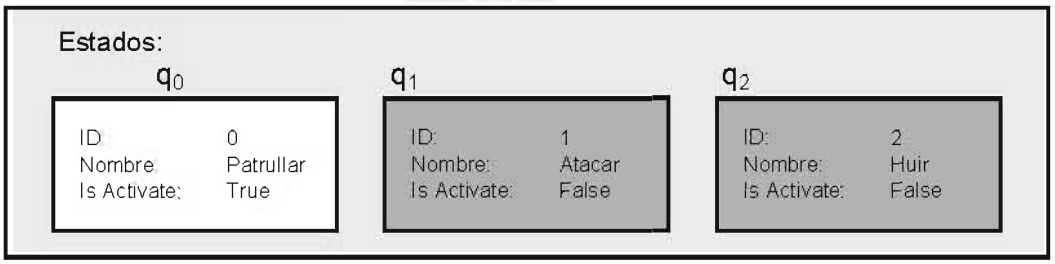


Figura 21 - Codificación de estados (los estados sombreados Están desactivados).

3.3.2 FUNCIONES DE TRANSICIÓN

Estas funciones están representadas por la clase *ActivationFunction*, Un estado puede poseer una o varias de estas funciones, el atributo *goto* nos indica el id del estado que habrá de activarse tras la confirmación del método Ok, el cual nos indica cuales de las funciones se cumplieron *(Figura 23)*, además de contener los métodos de las acciones de nuestros NPCs. En la *Figura 22* se puede ver el diagrama UML de esta clase.

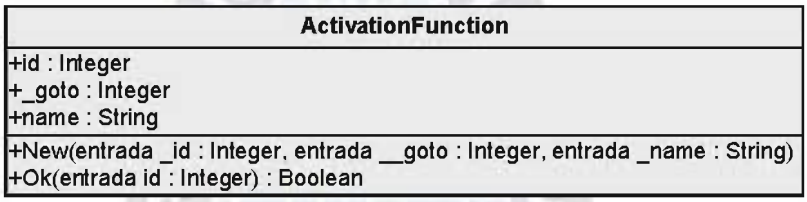


Figura 22 - Diagrama UML simplificado de la clase Rule.

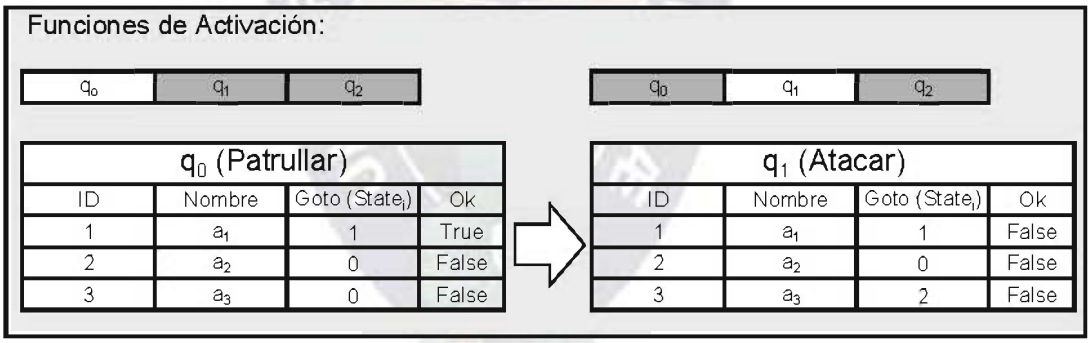


Figura 23 – Activación de una función (los estados sombreados están desactivados).

3.3.3 MAQUINA DE ESTADOS FINITOS FSMs

Nuestra maquina reúne a las clases State y ActivationFunction para así fusionarlos, siendo así una lista de estados, cada uno con sus respectivas reglas. Teniendo la estructura de la máquina de estados finitos, haremos uso de su atributo pointer para acceder al estado que se encuentre actualmente activo y así ejecutar la acción asignada. Mediante el método

Update se analiza en cada instante de la simulación si alguna regla se cumple para hacer así la transición de estado, el método Render muestra el grafico de la máquina de estados. En la *Figura 24* se puede ver el diagrama UML de esta clase.

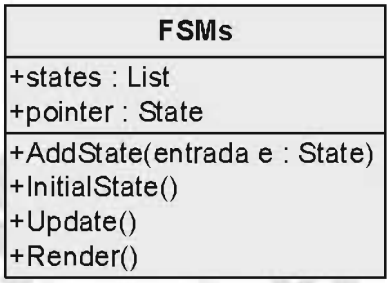


Figura 24 - Diagrama UML simplificado de la clase FSMs.

En la *Figura 25* se puede ver el diagrama de transición de nuestra maquina de estados finitos, con estado inicial q_0.

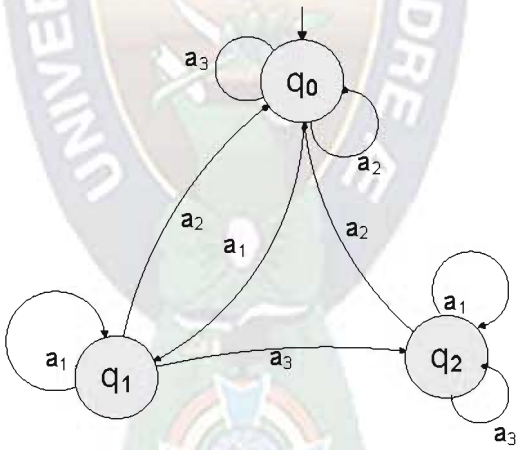


Figura 25 – Diagrama de transición de una maquina de estados finitos

Eliminando las transiciones que son bucle, ya que no colaboran en la transición de estados, y renombrando a las funciones de transición a_1, a_2 y a_3 por las variables booleanas. En la *Figura 26* se ve la representación simplificada de nuestra maquina de estados finitos.

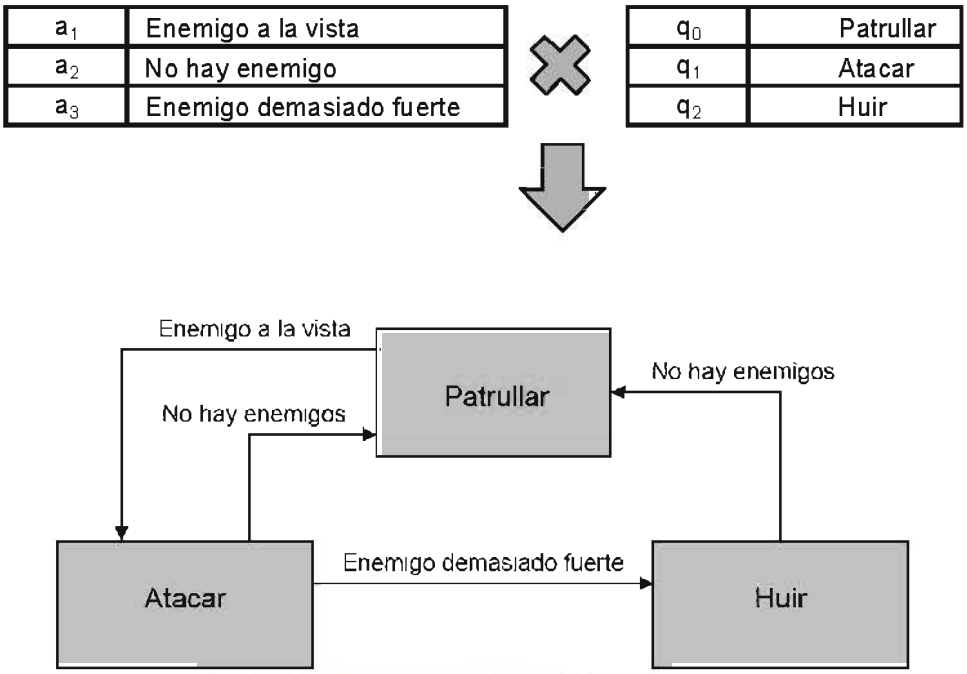


Figura 26 – Diagrama de transición simplificado

3.4 IMPLEMENTACIÓN DE REDES NEURONALES CON NEAT

Para entender cómo funciona la implementación de NEAT que se ha hecho, explicaremos la codificación de la red neuronal en forma de genes, su evolución mediante los operadores de cruce y mutación, y la conservación de la innovación mediante el uso de especies.

3.4.1 EL GENOTIPO EN NEAT

El genoma usado en NEAT consiste básicamente de una lista de genes neurona y otra de genes enlace. En la *Figura 27* se tiene el diagrama UML de las clases y estructuras usadas.

El gen neurona, NeuronGene, posee un identificador único, la función de la neurona dentro de la red (entrada, salida, oculta, o bias), la pendiente de la función de activación sigmoidal, y si es recurrente o no. En NEAT, una neurona se considera recurrente si tiene algún enlace con origen y destino

ella misma *(Figura 28)*.

El gen enlace, LinkGene, contiene los identificadores de las dos neuronas que conecta, el peso asociado al enlace, si está habilitado o no, si es recurrente o no, y un identificador de innovación, cuya función se explicará más adelante. En la *Figura 29* se puede ver la codificación de una red neuronal cualquiera.

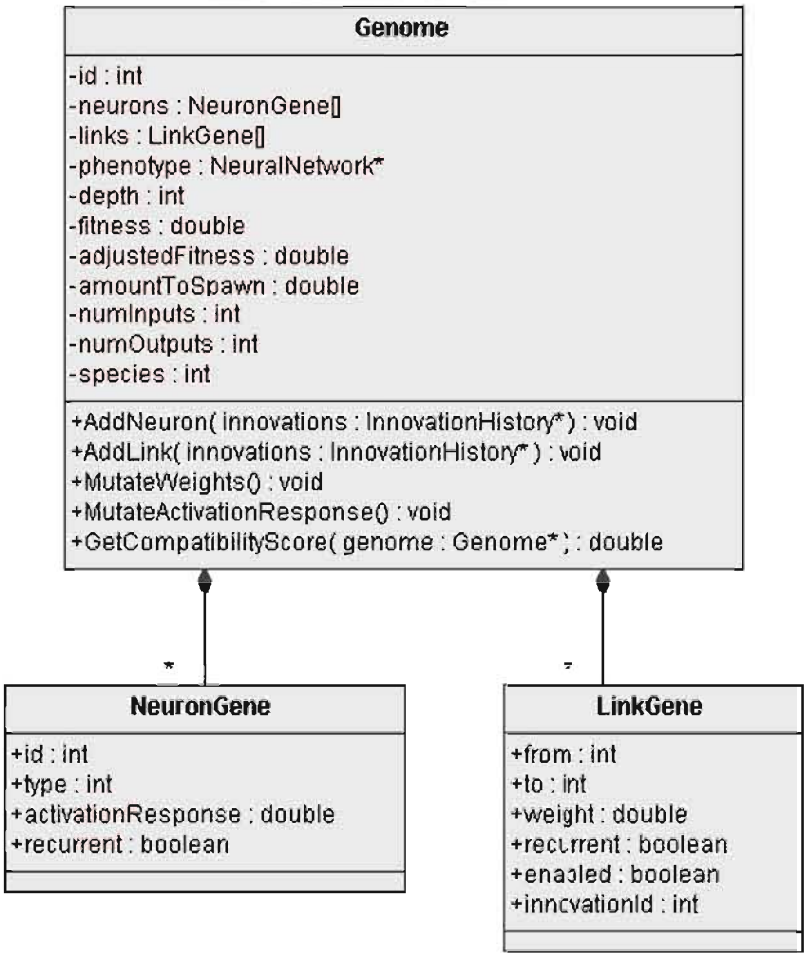


Figura 27 - Diagrama UML simplificado de la clase *Genome*, y de las estructuras *NeuronGene* y *LinkGene*

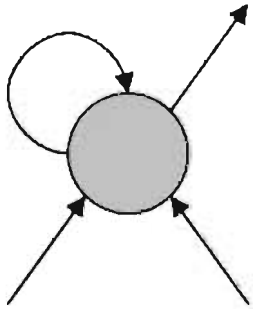
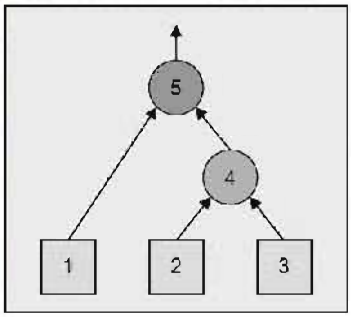
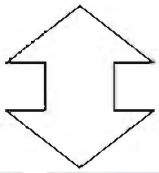


Figura 28 - Neurona recurrente



Red (fenotipo)



Genes neurona:

ID: 1 Tipo: Entrada	ID: 2 Tipo: Entrada	ID: 3 Tipo: Entrada	ID: 4 Tipo: Oculta	ID: 5 Tipo: Salida

Genes enlace:

Enlace: 1→5 Peso: 0.7 Activado: SI Recurrente: No Innovación: 1	Enlace: 2→5 Peso: -0.5 Activado: No Recurrente: No Innovación: 2	Enlace: 2→4 Peso: 0.6 Activado: SI Recurrente: No Innovación: 3	Enlace: 3→4 Peso: 0.2 Activado: SI Recurrente: No Innovación: 4	Enlace: 4→5 Peso: 0.4 Activado: SI Recurrente: No Innovación: 8

Genoma (genotipo)

Figura 29 - Codificación de una red neuronal en NEAT (los genes sombreados están desactivados)

3.4.2 INNOVACIONES

Una innovación sucede cada vez que se incorpora un gen neurona o un gen enlace a la red, y es simplemente un registro de dicho cambio. Se mantiene un historial de todas las innovaciones ocurridas con sus respectivos identificadores, y cada vez que se añade un nuevo gen, se consulta. Si ya se había creado previamente esta innovación, se le asigna al nuevo gen el identificador de dicha innovación. Si no, se crea una nueva, y el gen es etiquetado con su identificador. En la *Figura 30* puede verse un diagrama UML con los elementos usados para llevar la relación de innovaciones.

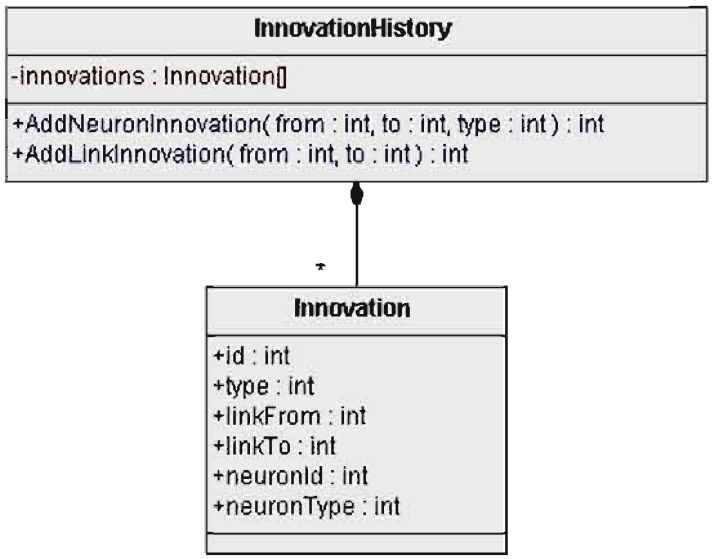


Figura 30 - Diagrama UML simplificado de la clase *InnovationHistory*, y de la estructura *Innovation*.

A modo de ejemplo, imaginemos que estamos evolucionando una red con dos entradas y una salida, a la que añadimos una neurona, como se indica en la *Figura 31*. Al añadir la neurona 4 se producen tres innovaciones: una para la neurona, y otras dos para los enlaces 1 → 4 y 4 → 3. El enlace 1 → 3 todavía existe en el genoma, pero ha quedado desactivado.

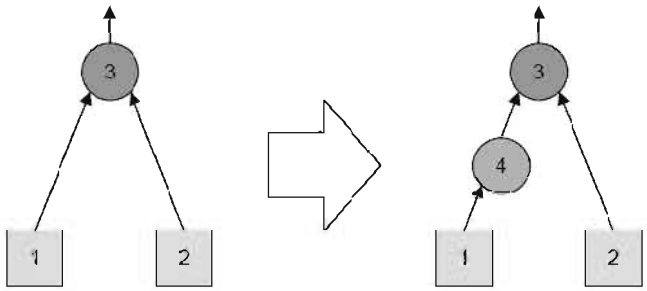


Figura 31 - Innovaciones en una red neuronal.

El historial de innovaciones antes de la mutación sería parecido al reflejado en la *Tabla 1*.

ID	Tipo	Origen del enlace	Destino del enlace	ID de la neurona	Tipo de neurona
1	Neurona	-1	-1	1	Entrada
2	Neurona	-1	-1	2	Entrada
3	Neurona	-1	-1	3	Salida
4	Enlace	1	3	-1	Ninguna
5	Enlace	2	3	-1	Ninguna

Tabla 1 - Historial de innovaciones antes de añadir la neurona.

Después de la innovación, pasaría a ser el de la *Tabla 2*.

ID	Tipo	Origen del enlace	Destino del enlace	ID de la neurona	Tipo de neurona
1	Neurona	-1	-1	1	Entrada
2	Neurona	-1	-1	2	Entrada
3	Neurona	-1	-1	3	Salida
4	Enlace	1	3	-1	Ninguna
5	Enlace	2	3	-1	Ninguna
6	Neurona	1	3	4	Oculta
7	Enlace	1	4	-1	Ninguna
8	Enlace	4	3	-1	Ninguna

Tabla 2 - Historial de innovaciones después de añadir la neurona.

3.4.3 OPERADOR DE CRUCE

El operador de cruce en redes de topología dinámica puede ser muy problemático (tanto que algunos métodos directamente lo descartan y funcionan únicamente con mutaciones). Además de conseguir que el cruce de genomas de longitudes distintas produzca redes válidas, se tiene que evitar la cuestión de las representaciones enfrentadas ya citado.

En NEAT se afrontan ambos problemas usando los identificadores de innovación como marcadores históricos de los genes. Dos genes con el mismo origen histórico necesariamente tienen que representar la misma estructura, ya que ambos derivaron del mismo gen ancestral en algún momento pasado. Esto es todo lo que el sistema necesita para saber cómo recombinar los genomas.

Durante el cruce, los genes de los padres con el mismo identificador de innovación se alinean. Estos son los llamados genes emparejables. Los genes con distintos identificadores son denominados disjuntos o sobrantes, dependiendo de si ocurren dentro o fuera del rango de innovaciones del otro padre, y representan estructuras no comunes.

Para crear el genoma hijo, los genes emparejables son escogidos aleatoriamente de cualquiera de los padres, mientras que de los disjuntos y sobrantes sólo se incluyen los del padre más apto. Si ambos tuviesen el mismo fitness, serían heredados del genoma más corto (nos interesa que las redes sean tan sencillas como sea posible).

En caso de tener la misma longitud, también se escogerían aleatoriamente. En la *Figura 32* se supone que la red de la derecha muestra una mayor aptitud.

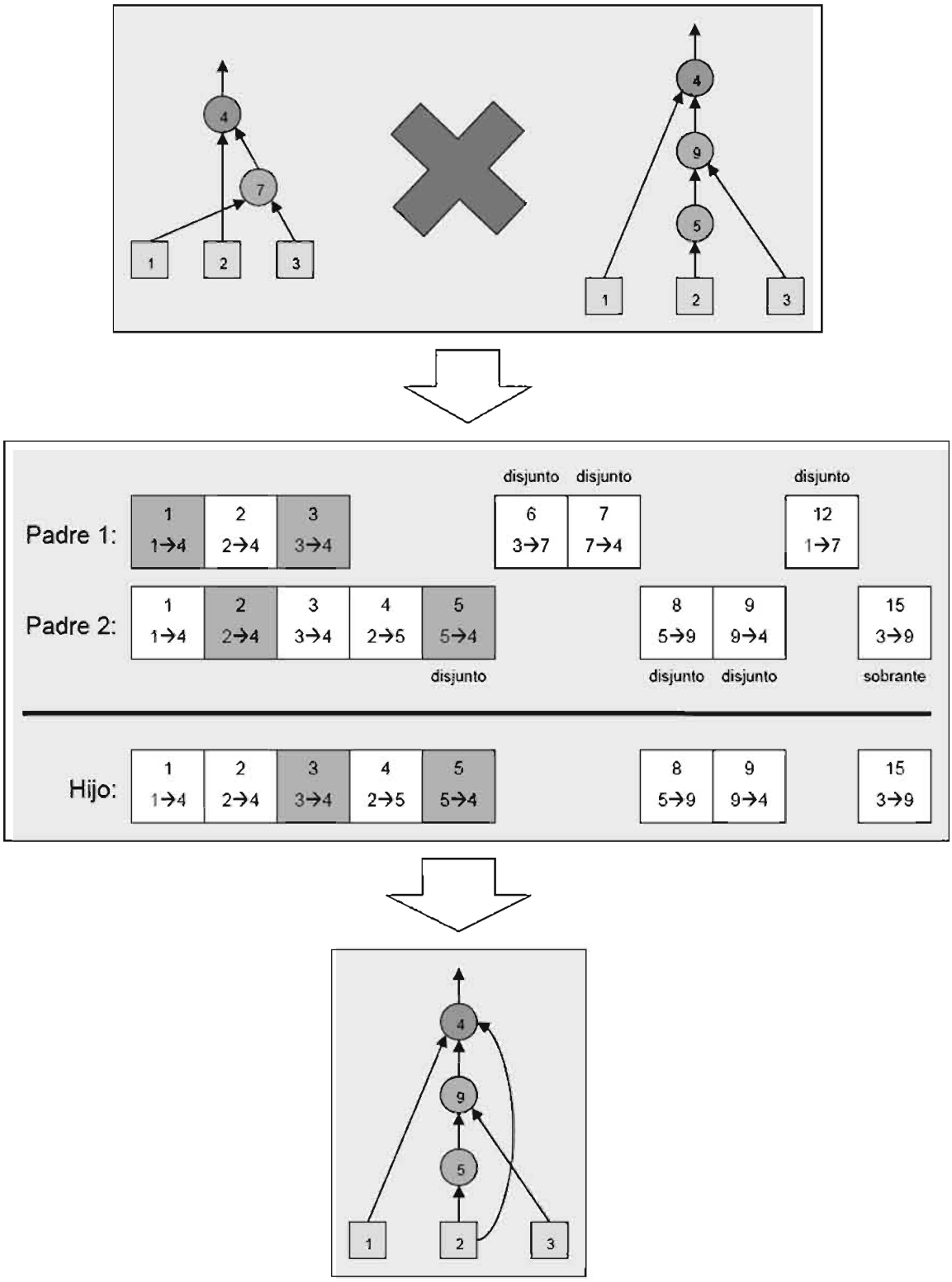


Figura 32 - Cruzando dos redes (los genes sombreados están desactivados).

3.4.4 OPERADORES DE MUTACIÓN

Además del operador de cruce, esta implementación de NEAT cuenta con cuatro operadores de mutación (Ver 2.5.1) que realizan, según probabilidades configurables, las operaciones que se describen seguidamente.

El primer operador que vamos a comentar es el encargado de añadir nuevas neuronas. Para incorporar una nueva neurona a la red, primero se debe seleccionar un enlace y deshabilitarlo. Se crean entonces dos enlaces nuevos para conectar la neurona con sus vecinas. El peso del enlace deshabilitado se usa para uno de los nuevos, y el otro se deja con el valor 1 *(Figura 33)*. Añadir una neurona implica por tanto tres innovaciones, como ya habíamos mencionado anteriormente.

El segundo operador a tratar es el responsable de añadir nuevos enlaces entre neuronas. Puede crear tres tipos de enlaces *(Figura 34)*:

- Un enlace hacia delante.
- Un enlace hacia atrás.
- Un enlace en bucle.

La función del tercer operador es la de modificar los pesos sinápticos de un enlace entre dos neuronas. Puede hacerlo de dos formas:

- Sumando o restando al peso actual del enlace una cantidad aleatoria, dentro de unos límites configurables.
- Sustituyendo el peso por otro totalmente nuevo.

Y el cuarto y último operador lo único que hace es modificar la pendiente de la función de activación sigmoidal de una neurona, sumando o restando una

cantidad aleatoria, también dentro de unos límites adaptables por el usuario.

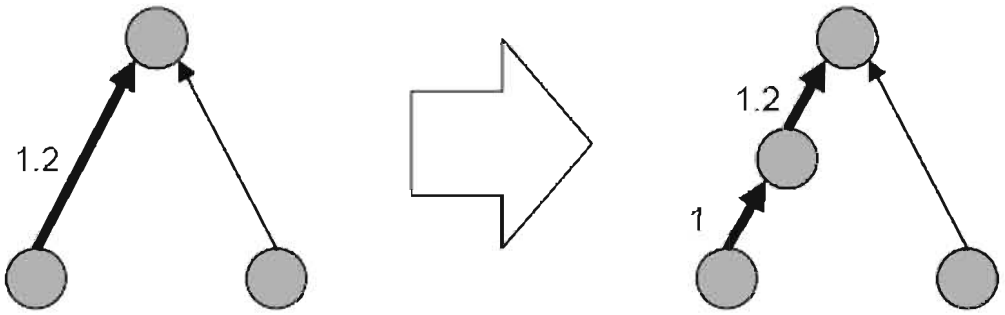


Figura 33 - Añadiendo una neurona.

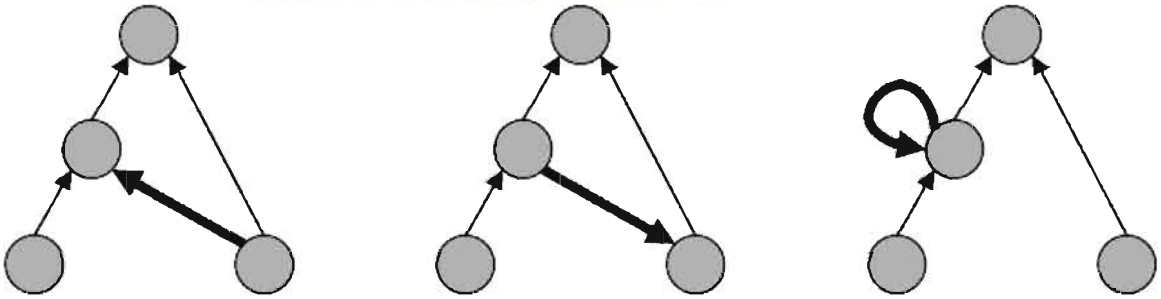


Figura 34 - Añadiendo un enlace

3.4.5 GESTIÓN DE ESPECIES

Cuando se añaden neuronas o enlaces al genoma, es bastante probable que el nuevo individuo exhiba una peor aptitud hasta que tenga la oportunidad de evolucionar y establecerse entre la población.

Desgraciadamente, esto significa que existe una alta probabilidad de que dicho individuo muera antes de demostrar comportamientos potencialmente interesantes. Para preservar estas fuentes de innovación durante los primeros momentos de su evolución, NEAT aplica el concepto de gestión

de especies: dividir la población según sus genes, de modo que individuos similares sólo tengan que competir entre ellos y no con el resto, quedando hasta cierto punto protegidos de una extinción prematura (2.5.2).

La clase *Species* lleva el registro de las especies creadas. En cada generación, se calcula la compatibilidad de todos los individuos de la población con el líder de cada especie. Si dicha compatibilidad supera un cierto umbral, el individuo es asignado a la misma. En la *Figura 35* se tiene el diagrama UML de esta clase.

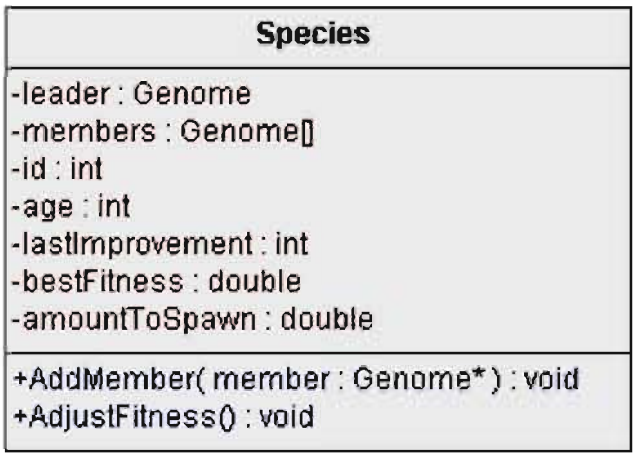


Figura 35 Diagrama UML simplificado de la clase *Species*

Una vez que se asigna un individuo a una especie mediante el método *AddMember*, sólo puede reproducirse con miembros de la misma especie. Sin embargo, haciendo únicamente esto no protegemos la innovación. Tenemos que encontrar un modo de ajustar la función objetivo de modo que ayude a los individuos más jóvenes y diversos a mantenerse vivos durante un periodo de tiempo razonable. En NEAT se premia a las especies más pequeñas y jóvenes, y se castiga a las más grandes y viejas, usando multiplicadores del fitness ajustables. De este modo resulta improbable que una sola especie se haga con toda la población. Éste es el cometido del método *AdjustFitness*.

3.4.6 QUÉ OCURRE EN UNA ÉPOCA

Para tener una perspectiva global de todo lo visto hasta ahora, vamos a repasar lo que ocurre en cada época del algoritmo genético. En la *Figura 36* se puede ver el diagrama UML de la clase *GeneticAlgorithm*, encargada de la manipulación de los genomas, las innovaciones y las especies.

Al llamar al método *Epoch*, lo primero que sucede es que se eliminan todos los fenotipos de la generación anterior. Después se examina cada especie, y se eliminan todos sus miembros excepto el más apto. Además, si la especie no ha mejorado su fitness desde hace un determinado número de generaciones, desaparece.

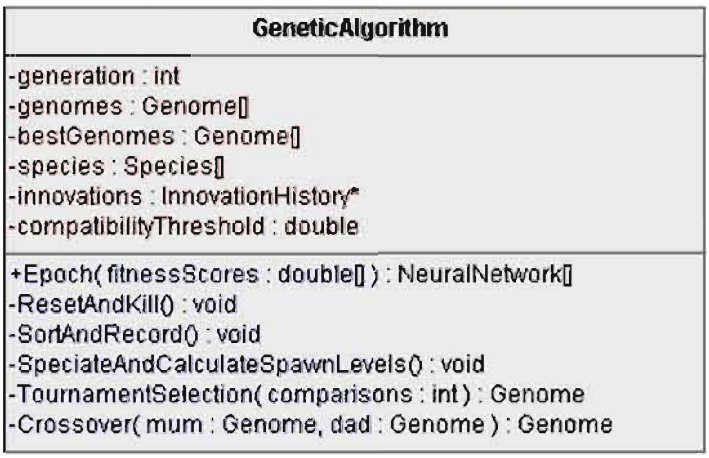


Figura 36 - Diagrama UML simplificado de la clase GeneticAlgorithm

Hecho lo anterior, se evalúa la compatibilidad entre cada genoma y los representantes de las especies que siguen vivas, y si es suficientemente alta se le asigna a esa especie. Si no se encuentra ninguna especie compatible, se crea una nueva compuesta únicamente por ese genoma. Una vez se han asignado todos los genomas a alguna especie, se ajusta su fitness como se ha explicado anteriormente.

Lo siguiente es calcular el número de hijos que debe generar cada especie, basándose en el fitness de sus miembros, y en el fitness medio de la población. Ahora recorremos cada especie generando individuos mediante cruce y mutándolos si corresponde, hasta completar una nueva población.

3.4.7 DEL GENOTIPO AL FENOTIPO

Ya sólo queda saber cómo se pasa del genotipo al fenotipo. En la *Figura 37* se puede ver el diagrama UML de las clases y estructuras usadas.

La clase *NeuralNetwork* es muy sencilla. Consiste simplemente en una colección de neuronas que componen la red. Posee un método *Render* para representar la estructura de la red en pantalla, y otro *Update* que actualiza la red a partir de las entradas proporcionadas y devuelve la salida obtenida. El segundo parámetro pasado a este método, *type*, requiere una explicación.

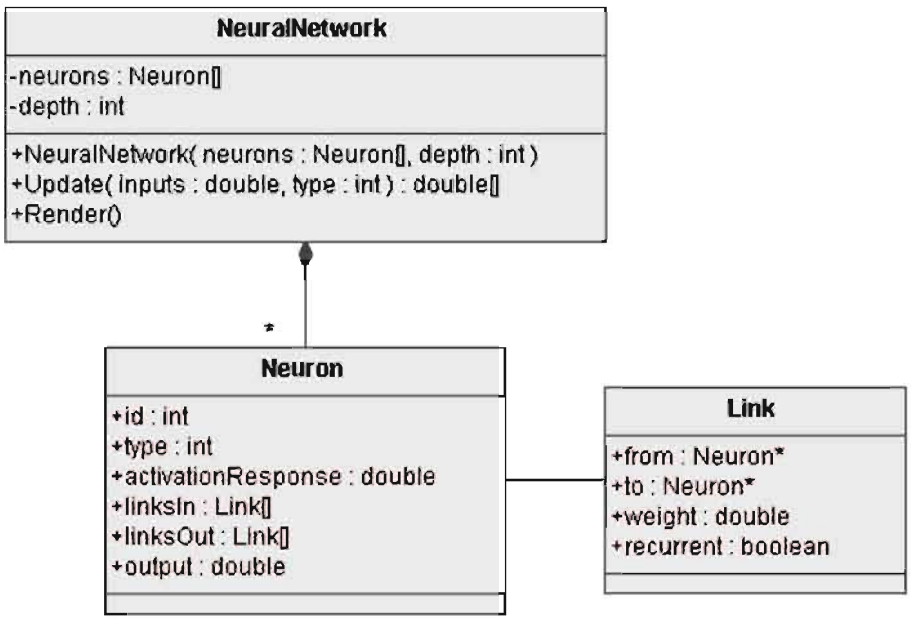


Figura 37 - Diagrama UML simplificado de la clase NeuralNetwork, y de las estructuras Neuron y Link

Dado que en NEAT una red va a poder asumir cualquier topología, con enlaces entre neuronas hacia delante, hacia atrás, o incluso consigo mismas, no se puede usar un algoritmo de actualización normal basado en capas. Por eso, el método *Update* permite dos tipos de funcionamiento:

• *active*: En este modo, los valores de activación viajan únicamente de una neurona a la siguiente, en vez de por toda la red como ocurre usando algoritmos convencionales. Para conseguir el mismo resultado que con este tipo de algoritmos, tendríamos que repetir este proceso varias veces.

• *snapshot*: Usaremos este modo si, por el contrario, queremos que la función se comporte como la de una red neuronal corriente. Para ello tendremos que asegurarnos de actualizar todos los pesos desde la capa de entrada hasta la de salida, por lo que iteraremos por cada neurona tantas veces como profundidad tenga la red. Este tipo de funcionamiento es apropiado si queremos usar un conjunto de entrenamiento.

Las neuronas vienen representadas por la estructura *Neuron*, que almacena un identificador único, el tipo de neurona (entrada, salida, oculta o bias), la pendiente de la función de activación sigmoidal, y la salida de la neurona. También tiene dos vectores de enlaces, uno con los entrantes y otro con los salientes.

Los enlaces entre neuronas están representados por la estructura *Link*, que guarda los punteros a las neuronas que conecta, el peso del enlace, y si se trata de un enlace recurrente.

El método *CreatePhenotype* de la clase *Genome* es el encargado de recorrer el genoma y crear las neuronas y los enlaces necesarios para formar una red completa, la cual servirá de cerebro a nuestros personajes.

3.5 DESARROLLO DEL ENTORNO DE EXPERIMENTACIÓN

Se ha creado un prototipo de videojuego de acción en 2D. Dados unos mapas predefinidos, se añadirán NPCs controlados por redes neuronales evolutivas y Maquina de estados finitos, y se intentará que cumplan una serie de objetivos. A continuación veremos cómo se ha construido este entorno de experimentación, bautizado "IA-FOR-GAMES".

3.5.1 IA-FOR-GAMES

IA-FOR-GAMES es el entorno de experimentación desarrollado para probar la implementación de redes neuronales artificiales con NEAT y maquinas de estados finitos que se han hecho. Imita FPSs (Frames por segundo) y con una vista aérea en 2D. En la *Figura 38* se puede ver una pantalla de la aplicación.

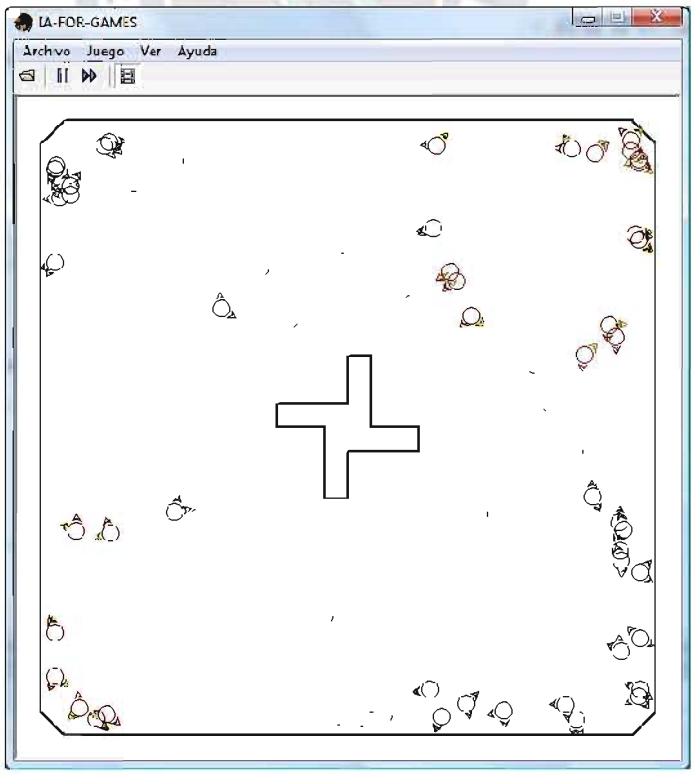


Figura 38 - Pantalla de la aplicación

El sistema de juego es muy básico, pero suficientemente complejo como para probar los algoritmos que nos interesan. Los NPCs son colocados en un mapa y asignados a un equipo, y deben moverse esquivando obstáculos, disparando a entidades enemigas, y evitando ser disparados.

La única forma que tiene el usuario de interactuar con este mundo es mediante la colocación de objetivos, a las que los NPCs tratarán de aproximarse.

3.6 ARQUITECTURA DEL JUEGO

En la *Figura 39* puede verse un diagrama UML de alto nivel de las clases que conforman nuestro prototipo.

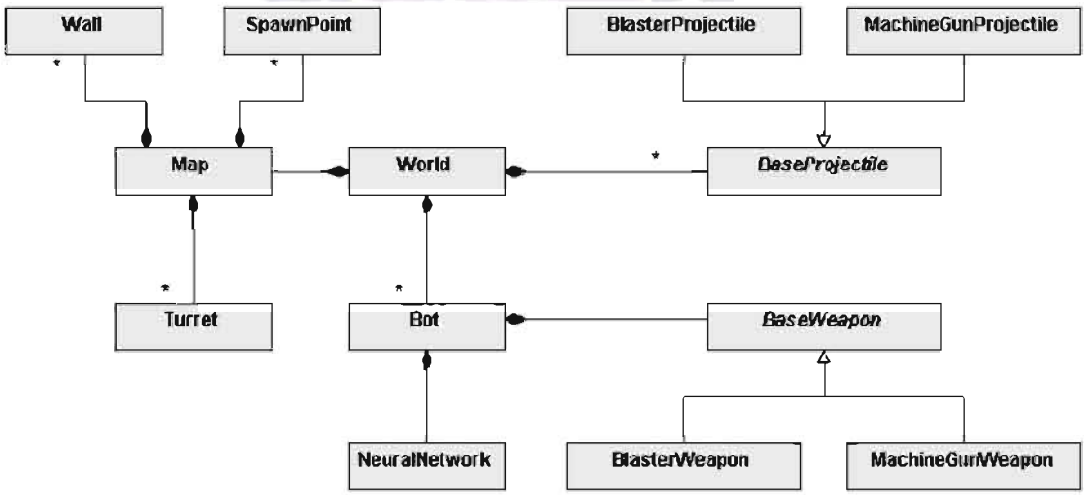


Figura 39 - Diagrama UML de alto nivel del entorno de experimentación

El eje de la arquitectura de IA-FOR-GAMES es la clase *World*, desde la que se controla todo lo que ocurre en el mundo de juego. Analicemos sus partes.

3.6.1 MAPA

Un mapa de IA-FOR-GAMES es un conjunto de paredes que definen las habitaciones y pasillos por las que se moverán los NPCs. El mapa en sí está representado por la clase *Map*, y las paredes por la clase *Wall*.

Los NPCs nacen en posiciones específicas dentro del mapa, llamadas puntos de encuentro. Estos puntos pueden pertenecer a un determinado equipo, de modo que sólo los miembros de ese bando aparezcan ahí, o ser neutrales, en cuyo caso son usados por cualquier NPC. Están representados por la clase *SpawnPoint*.

En un mapa también puede haber torretas, pero hablaremos de ellas más adelante. En la *Figura 40* puede verse el diagrama UML de las clases que se han mencionado.

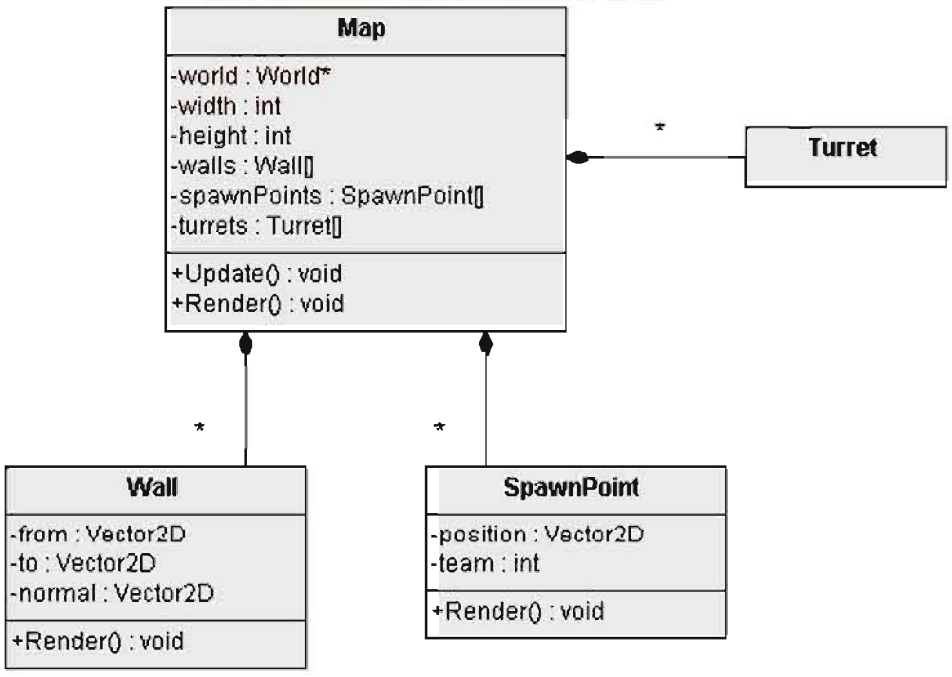


Figura 40 - Diagrama UML simplificado de las clases usadas en un mapa

3.6.2 ENTIDADES DE JUEGO

Para implementar las distintas entidades que aparecen en el mapa, y asegurarnos de que la arquitectura sea fácilmente extensible, se han ideado tres clases abstractas, BaseEntity, MovingEntity y ShootingEntity. Podemos verlas en el diagrama UML de la *Figura 41*.

La clase *BaseEntity* es la raíz de la que derivan todas las entidades. Sus atributos más importantes son *id*, un identificador único, *type*, que indica el tipo de entidad, y *position*, que nos da su posición en el mapa. En cuanto a los métodos, *Update* es llamado en cada instante o tick de juego para que la entidad se actualice según corresponda, y *Render* dibuja la entidad en pantalla.

La clase *MovingEntity* hereda de *BaseEntity*, y añade la información necesaria para simbolizar una entidad en movimiento. Sus atributos más importantes son *heading*, que indica la dirección en la que la entidad se mueve, y *facing*, que indica la dirección en la que mira (no tienen por qué coincidir).

La clase *ShootingEntity*, heredera de *MovingEntity*, se usará para toda aquella entidad que pueda disparar armas. El atributo *weapon* apunta al arma equipada en ese momento, *score* refleja la puntuación actual del NPC, *health* su salud, *status* si está vivo o muerto, y *team* el equipo al que pertenece. Los atributos *fov* y *dov* los veremos aplicados más adelante, cuando hablemos de las torretas y los NPCs.

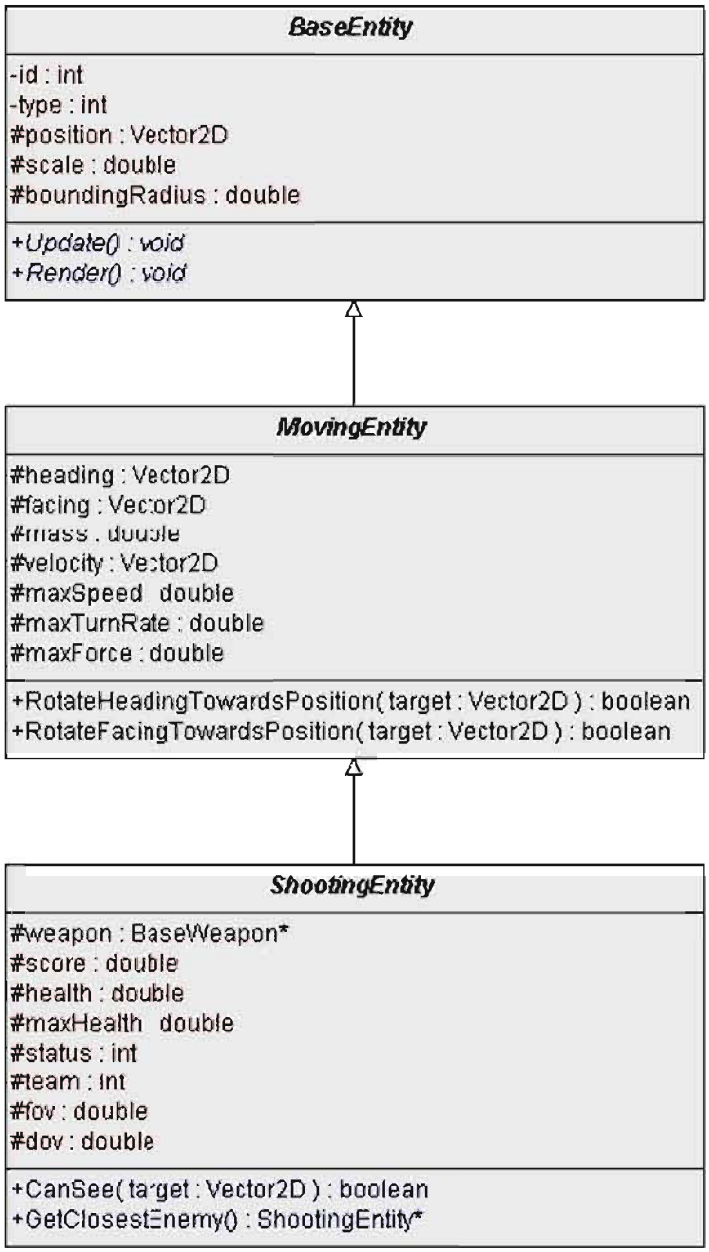


Figura 41 - Diagrama UML simplificado de las entidades del juego

3.6.3 ARMAS Y PROYECTILES

Como acabamos de ver, las entidades de la clase *ShootingEntity* pueden llevar armas, representadas a su vez por la clase abstracta *BaseWeapon*. De ella heredarían todos los tipos de armas del juego. En esta versión sólo

hay dos armas, *BlasterWeapon* y *MachineGunWeapon*, pero la lista se podría ampliar fácilmente. El método *AimAt* hace que el propietario del arma apunte al objetivo indicado por *target*, mientras que *ShootAt* hace que le dispare.

Al disparar un arma, ésta crea un nuevo proyectil del tipo que corresponda. Todos los proyectiles heredan de la clase abstracta *BaseProyectile*, que es a su vez hija de *MovingEntity*. La clase *World* es la encargada de actualizar la posición de los proyectiles llamando a su método *Update*, hasta que colisionan con otra entidad o con una pared y mueren.

Cada NPC empieza equipado con un arma de la clase *BlasterWeapon*, mientras que las torretas usan armas de la clase *MachineGunWeapon*. Parámetros como la munición máxima del arma, su velocidad de disparo, y demás, son configurables por el usuario.

En la *Figura 42* se tiene el diagrama UML de las clases implicadas.

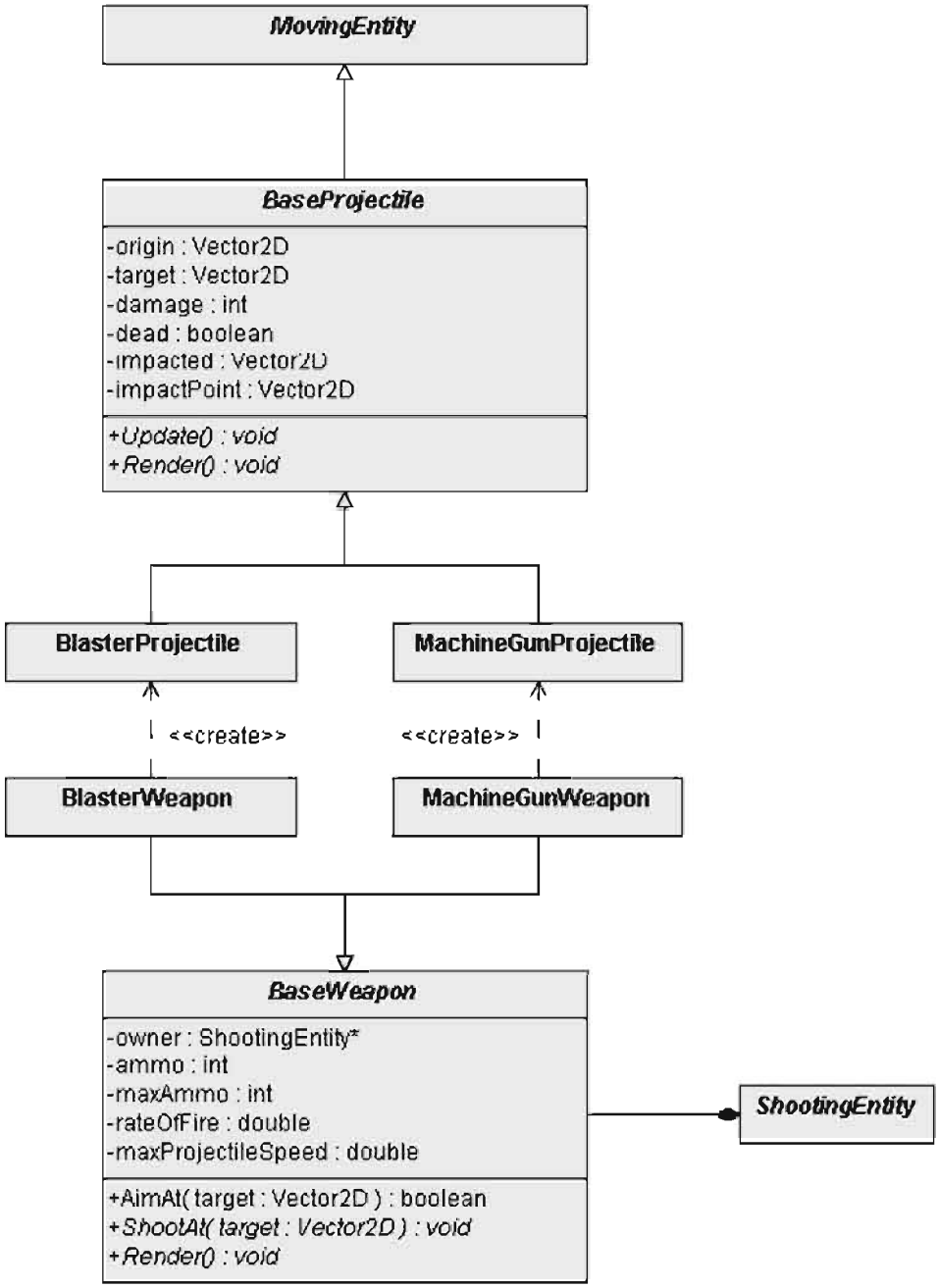


Figura 42 - Diagrama UML simplificado de las clases usadas, para implementar las armas del juego

3.6.4 TORRETAS

Las torretas son entidades de la clase *ShootingEntity*, cuya única función es disparar a todo agente enemigo que entre en su campo de visión. Este campo de visión viene definido por dos atributos a los que se ha hecho

referencia anteriormente: *fov* y *dov*. El primero indica el ángulo de visión que tiene la entidad, y el segundo la distancia hasta la que puede ver. El esquema de la *Figura 43* muestra ambos conceptos.

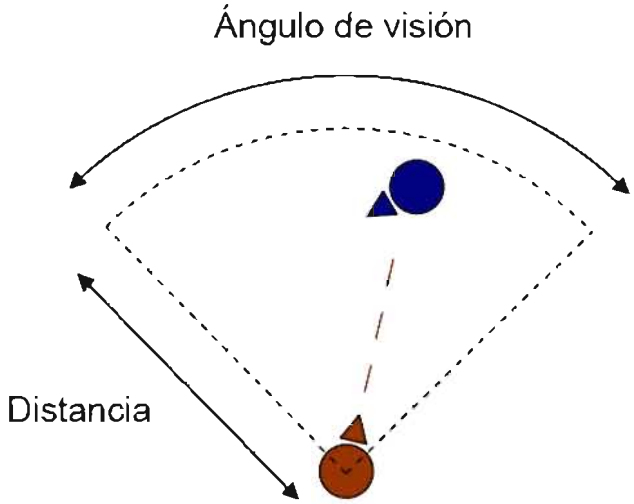


Figura 43 - Representación del campo de visión de la torreta

Cuando un NPC enemigo entra en el campo de visión de la torreta, es guardado en la variable *target*, y hasta que no salga del mismo o muera, la torreta no buscará otro objetivo. También puede usarse una torreta para patrullar una zona, estableciendo los extremos de su trayectoria con los atributos *from* y *to*. En la *Figura 44* se tiene el diagrama UML de la clase *Turret*.

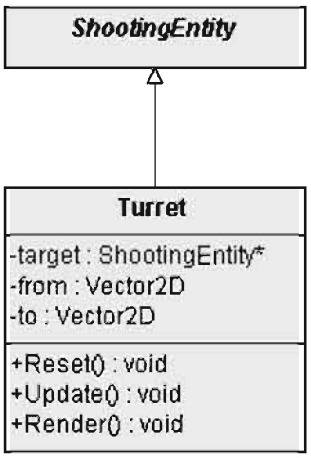


Figura 44 - Diagrama UML simplificado de la clase Turret

3.6.5 NPCs

Una de las tareas que van a tener que realizar los NPCs es navegar por el mapa esquivando los obstáculos a su paso. Para ello, el agente debe ser capaz de:

- Observar la geometría del mapa.
- Llevar a cabo las acciones necesarias para evitar colisiones potenciales.

Para que el agente sea capaz de percibir los obstáculos, hemos decidido dotarle de antenas o sensores que se originan en su centro y cubren un ángulo de 180º. Tanto el número de sensores como su longitud son configurables por el usuario. También usamos el concepto de radio de colisión, que es el del círculo que engloba toda la geometría de la entidad.

En la *Figura 45* se puede verse un NPC con cinco sensores de obstáculos, y su radio de colisión.

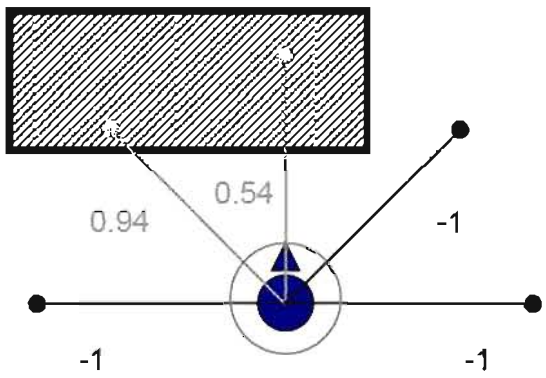


Figura 45 – Sensores de obstáculos

En cada tick de juego se comprueba si existe alguna intersección entre los sensores del NPC y las paredes del mapa. Si es así, se anota la distancia a la que se produjo el impacto, normalizada entre 0 y 1 (0 sería el centro del

NPC, 1 el extremo del sensor). De no existir intersección en un sensor, se anota -1.

Existe otro sensor relacionado con la navegación, que sólo toma los valores 0 y 1, y que se activa cuando un obstáculo entra en el radio de colisión del NPC. Aunque podría parecer que con los sensores de obstáculos descritos arriba el NPC tendría información más que suficiente, mediante experimentación ha quedado demostrado que el uso de este sensor adicional mejora la eficacia del algoritmo.

El otro cometido de los NPCs es atacar a los enemigos y evitar sus disparos, por lo que deben ser capaces de:

- Detectar la presencia de miembros de equipos contrarios.
- Apuntar y dispararles.
- Saber dónde están apuntando e intentar no ponerse a tiro.

Para abordar este problema, se ha seguido el mismo planteamiento que con los obstáculos, sólo que, en vez de cubrir 180º con los sensores, se cubre el campo de visión del NPC. Véase la *Figura 46*

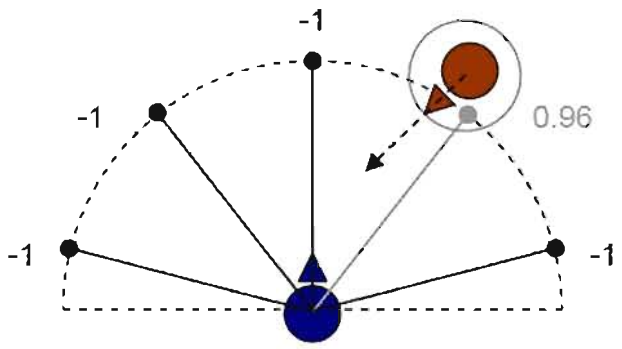


Figura 46 - Sensores de enemigos

Como ocurría antes, anotamos para cada sensor la distancia a la que se ha detectado algún enemigo. Además de estos sensores, usaremos otros tres:

uno para advertir al NPC que tiene un enemigo en el punto de mira, y dos para indicarle la dirección de disparo del enemigo más cercano.

Las lecturas de estos sensores servirán de entradas a la red neuronal que gobernará las acciones del NPC. Sus salidas serán cuatro: moverse hacia delante y hacia atrás, moverse a izquierda y derecha, mirar a izquierda y derecha, y disparar. En la *Figura 47* se tiene una representación de la red resultante, similar a la empleada en el proyecto.

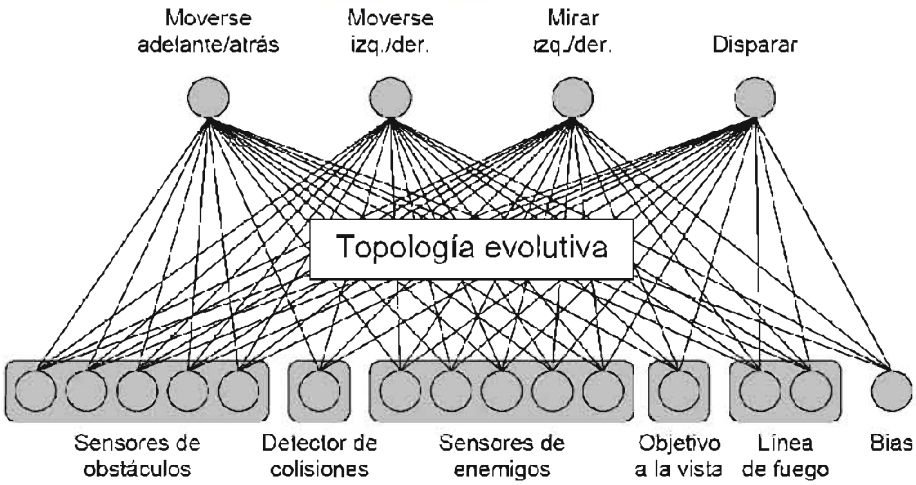


Figura 47 - Red neuronal de un NPC

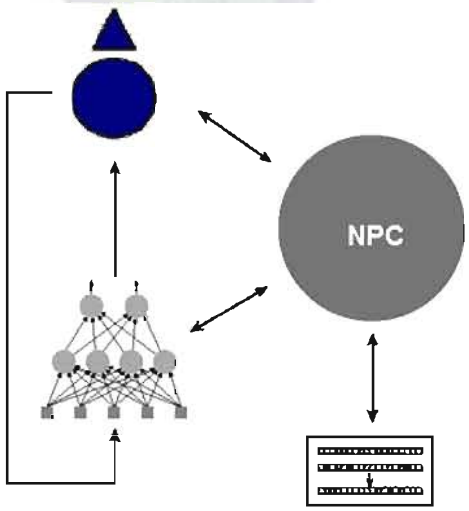


Figura 48 – Elementos que componen un NPC

Uno de los aspectos clave del algoritmo genético que evolucionará los pesos y topología de estas redes es la función objetivo que escojamos. En nuestro caso, hemos tenido en cuenta:

- *Distancia a los objetivos*: El usuario puede colocar marcas en el mapa, y los NPCs que más se acerquen a ellas recibirán mayor fitness dicho de otra forma recibirán mayor puntuación. De este modo conseguimos controlar parcialmente hacia donde se mueven los NPCs.
- *Salud*: Nos interesa que los NPCs aprendan a esquivar los disparos enemigos, por lo que aquellos NPCs que reciban varios disparos por parte de los enemigos morirán.

Para finalizar, en la *Figura 49* se puede ver el diagrama UML de la clase *Bot* la cual representa a un NPC.

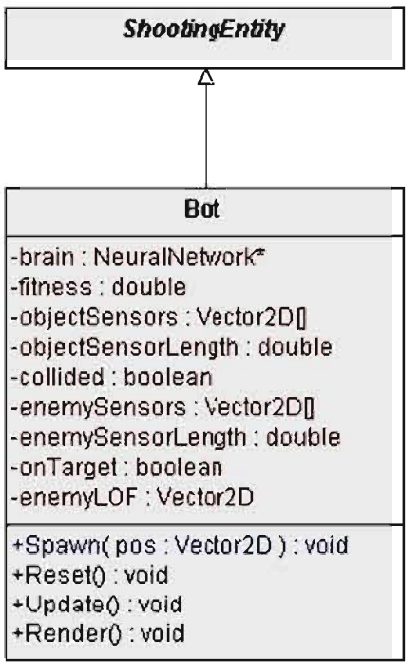


Figura 49 - Diagrama UML simplificado de la clase Bot

3.6.6 VISIÓN GLOBAL DEL MUNDO

Como ya hemos dicho, la clase *World* es la que gobierna todo lo que ocurre en el juego. Cuando el usuario carga un mapa, se crean los genomas de una nueva población de individuos con sus correspondientes fenotipos. Se asignan las redes neuronales resultantes a los NPCs, y se les añade al mapa.

En cada tick del juego, el método *Update* de la clase *World* actualizará las torretas del mapa, y todos los proyectiles que se encuentren activos. Si todavía no se ha cumplido el tiempo que dura una generación, actualizará también los NPC (sensores, posición, etc.). De haberse cumplido, calculará el fitness de cada NPC, y llamará al método *Epoch* del algoritmo genético para que realice las operaciones de selección, cruce y mutación, dando lugar a una nueva población. Sustituirá entonces los cerebros antiguos de los NPC por los nuevos, comenzando de nuevo el proceso.

En la *Figura 50* se tiene un diagrama UML de la clase *World*.

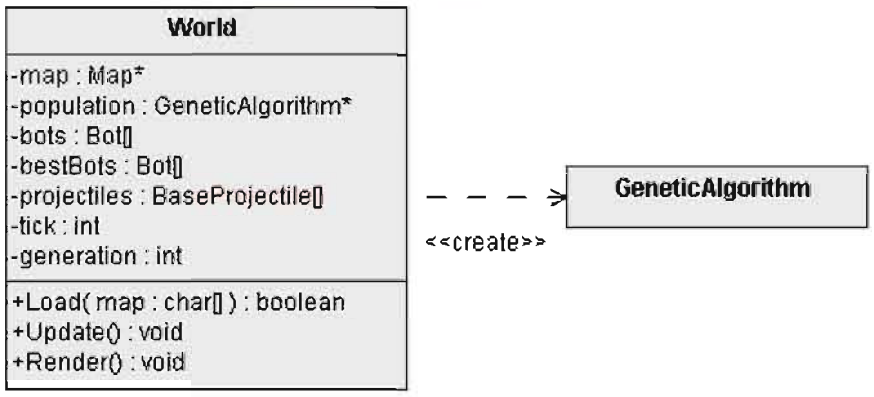


Figura 50 - Diagrama UML simplificado de la clase World

4

Análisis de Resultados

CAPÍTULO 4: ANÁLISIS DE RESULTADOS

4.1 INTRODUCCIÓN

Dentro el marco del método científico y habiendo completado los entornos de experimentación, podemos realizar la parte de la experimentación, para probar la capacidad de las redes neuronales con NEAT y las maquinas de estados finitos, para que permitan a los personajes del videojuego realizar distintas acciones dentro entornos elaborados con el fin de demostrar su comportamiento en distintos aspectos, como la navegación por un área con obstáculos y la detección de enemigos. A dichos entornos se los designo con el nombre de mapas (Ver 3.6.1), cada uno de ellos con un nivel de dificultad distinto.

Para poder realizar un análisis correcto de los resultados obtenidos y como las redes neuronales requieren de una población para entrenarse se ha decidido, realizar 30 pruebas con una población de 50 individuos por cada mapa, tomado como factor de evaluación el puntaje obtenido por toda la población y este será el puntaje medio de la población, la forma de puntuación se la explicara más adelante a medida que se realicen las pruebas, para luego realizar un análisis estadístico de los resultados proporcionados por los prototipos, el análisis se lo realizara por el método de prueba de hipótesis por cada mapa probado.

Se utilizaran dos mapas con obstáculos, en la que los NPCs aparecerán en un punto del mapa SpawnPoint (Ver 3.6.1) y el objetivo será el de llegar a un punto especifico en el mapa, teniendo que aprender a esquivar los obstáculos que encuentren a su paso, el punto objetivo se lo designa haciendo click sobre el mapa.

Posteriormente se probara un tercer mapa en el cual el objetivo será el

mismo que en los dos anteriores mapas, con la diferencia que habrá enemigos llamados Torretas (Ver 3.6.4) posicionados en distintos puntos del mapa, estos enemigos atacaran a los NPCs que se acerquen, lo cual representara una dificultad extra para llegar a su objetivo.

4.2 FORMA DE PUNTUACIÓN

La puntuación se la realizara sobre una base de 125 puntos. A esta puntuación se la denominara "Aptitud", palabra que se usa en los algoritmos genéticos para calificar el comportamiento de un individuo (Ver 2.6.5). A mas cercanía del objetivo más puntaje (Ver Figura 51).

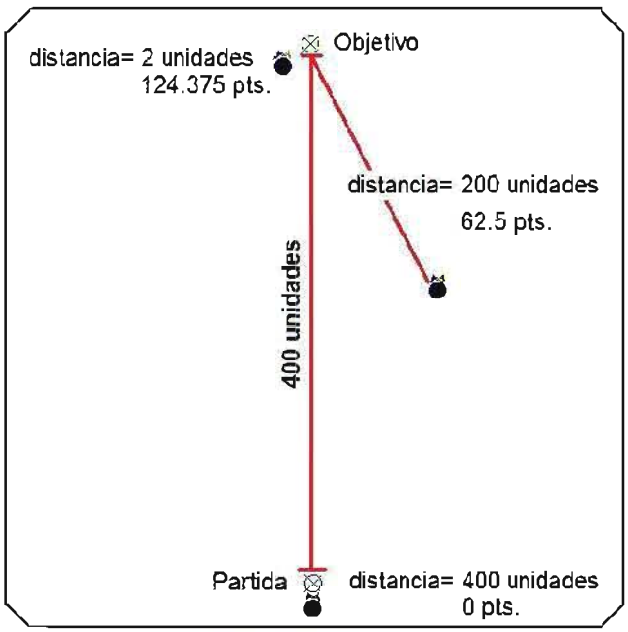


Figura 51 – Forma de puntuación para un solo NPC.

Descripción: Si la distancia máxima a la cual pueden estar separados los NPCs y el objetivo son 400 unidades, entonces el factor de puntuación f seria (125/400)= 0,3125. Si en un siguiente momento nuestro NPC está a una distancia de 200 (su puntaje será 125 – f * 200 = 62,5 pts.) y si se encuentra cerca del objetivo por ejemplo a una distancia de 2 (su puntaje será 125 – f * 2 = 124,375). Como un individuo puede llegar por casualidad,

se tomara la puntuación media de la población a la cual llamaremos "Aptitud media".

$$\text{Aptitud media} = \frac{\text{Suma (Aptitud de cada individuo)}}{\text{Tamaño de la población}}$$

Además cada NPC tiene un nivel de energía de 1000 puntos esta energía reduce en 100 puntos con cada disparo que recibe por parte de un enemigo, si el nivel de energía es menor o igual a cero el NPC muere, para ser claro cambia de color y se queda inmóvil en el mapa.

4.3 DESCRIPCIÓN DE RESULTADOS

Aquí esta una descripción de las ventanas de diálogos creadas para mostrar los resultados proporcionados por nuestros prototipos.

4.3.1 REDES NEURONALES CON NEAT

El prototipo que hace uso de redes neuronales artificiales con NEAT nos muestra tres pantallas (*Ver Figura 52*).

Figura 52 – IA-FOR-GAMES ventanas de diálogo.

La primera de ellas, *Población Info*. La cual nos proporciona información sobre la población de la generación anterior a la actual. Tiene varios campos (*Ver Figura 53*):

Figura 53 – Dialogo Población Info.

- *Generación*: Número de la generación para la que se muestra la información.
- *Aptitud Media*: Valor de aptitud media de la población en esta generación.
- *Mejor Aptitud*: Mejor aptitud de entre todos los individuos de esta generación.
- *Aptitud más alta*: Mejor aptitud encontrada hasta el momento, teniendo en cuenta todas las generaciones.
- *Especies*: Número de especies en las que está dividida la población.
- *Umbral*: Umbral de compatibilidad usado a la hora de determinar si un individuo pertenece a una especie.
- *Distribución*: Muestra un gráfico representando la distribución de individuos entre las distintas especies.

El siguiente elemento, *Grafica de Fitness (Aptitud)*, presenta una gráfica con el progreso de la aptitud hasta la generación anterior a la actual. El eje horizontal representa las generaciones, y el vertical la aptitud. La línea azul corresponde a la mejor aptitud de cada generación, y la roja a la aptitud media de la población. Nos permite conocer, de un vistazo, la evolución que han experimentado los NPCs con el paso del tiempo (*ver Figura 54*).

Figura 54 – Dialogo Grafica de Fitness.

El último elemento es *Mejores Fenotipos (Redes Neuronales)*, que nos muestra los mejores fenotipos de la generación anterior a la actual. Las neuronas siempre son rojas, pero los enlaces, según su tipo, se dibujan de un color u otro, en la *Figura 55 puede verse los mejores cuatro fenotipos*.

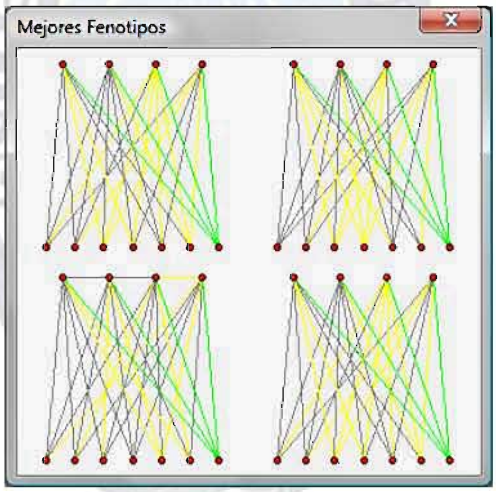


Figura 55 – Dialogo Mejores Fenotipos.

- *Amarillo*: Enlace hacia delante con un peso (menor o igual que cero).
- *Gris*: Enlace hacia delante con peso (mayor que cero).
- *Azul*: Enlace hacia atrás o en bucle, y con peso (menor o igual que cero).
- *Rojo*: Enlace hacia atrás o en bucle, con peso (mayor que cero).
- *Verde*: Enlace que sale de la neurona bias.

4.3.2 MÁQUINA DE ESTADOS FINITOS

El prototipo que utiliza máquina de estados finitos solo nos muestra dos

ventanas de dialogo (Ver Figura 56).

Figura 56 - IA-FOR-GAMES v.2 ventanas de diálogo.

La primera Máquina de Estados nos permite ver el funcionamiento de la máquina de estados finitos, en la *Figura 57* se muestran los estados usados en, verde el estado activo.

Figura 57 – Dialogo Maquina de Estados.

La segunda llamada Aptitud, adema tiene varios campos (Ver *Figura 58*).

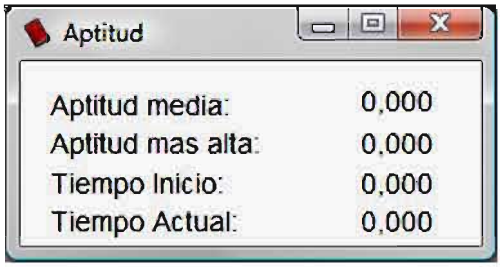


Figura 58 – Ventana Aptitud.

- *Aptitud Media*: Valor de aptitud media de la población.
- *Aptitud más alta*: Mejor aptitud encontrada hasta el momento.
- *Tiempo inicio*: Hora en que se inicio la prueba.
- *Tiempo actual*: Hora actual.

Teniendo ya una buena comprensión de los resultados arrojados por los prototipos, pasamos a la parte de las pruebas.

4.4 OBSTÁCULO I

Los NPCs aparecerán en la parte de abajo del mapa, y tendrán que encontrar la forma de llegar a un punto situado en la parte de arriba. Entre ambos hay un muro, que deberán aprender a sortear. Véase la *Figura 59* .

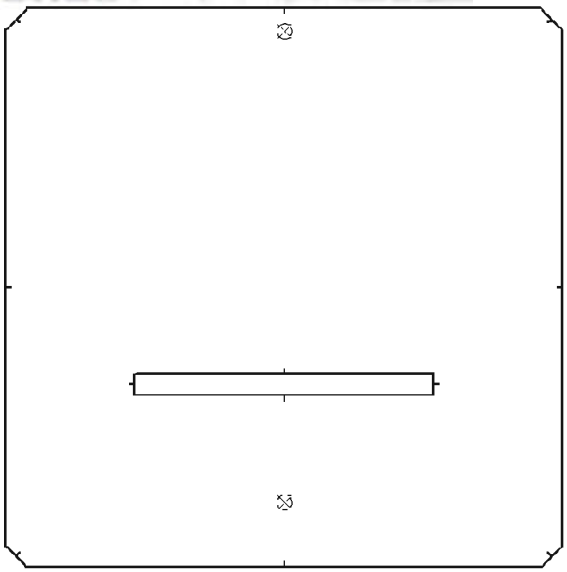


Figura 59 – Mapa con obstáculos I.

4.4.1 REDES NEURONALES CON NEAT

Los parámetros de configuración (Leer 3.4.4) más importantes para este mapa son los siguientes:

- *Tamaño de la población*: 50 individuos
- *Sensores de obstáculos*: 5 - Los cuales se usaran para poder detectar las paredes de nuestros mapas.
- *Tasa de cruce*: 0,7 - Parámetro usado para después de cada generación para poder determinar el número de descendientes de una especie.
- *Tasa de mutación*: 0,15 – Parámetro usado después de cada generación para realizar la mutación de una determinada cantidad de individuos de la descendencia de una especie.
- *Probabilidad de añadir una neurona*: 0,04 – Después de decidir si un individuo mutara si se cumple esta probabilidad, se añadirá una neurona a la topología de la red neuronal, pero únicamente en la capa oculta.
- *Probabilidad de añadir un enlace*: 0,07 - Después de decidir si un individuo mutara si se cumple esta probabilidad, se añadirá un nuevo enlace a la topología de la red neuronal, uniendo dos neuronas cualesquiera.
- *Probabilidad de añadir un enlace recurrente*: 0,05 - Después de decidir si un individuo mutara si se cumple esta probabilidad, se añade un enlace recurrente nuevo en cualquier neurona de la capa de salida de la red neuronal.
- *Probabilidad de reemplazar un peso*: 0,1 - Después de decidir si un individuo mutara si se cumple esta probabilidad, se cambiara el valor de un peso cualquiera dentro de red neuronal.
- *Número máximo de especies*: 5 – Parámetro que limita el número de especies que pueden existir dentro el mapa.
- *Umbral*: 0,26 – Parámetro usado al inicio de cada generación para verificar a que especie pertenece un individuo.

Sólo necesitamos los sensores de obstáculos y el detector de colisiones, ya que no hay enemigos en el mapa. Por eso las redes tendrán únicamente siete neuronas en la capa de entrada: cinco sensores, el detector, y una neurona bias. Los parámetros mencionados anteriormente pueden ser cambiados.

Después de unas 15 generaciones, se empezó a observar unos resultados

satisfactorios. En la *Figura 60* puede verse información sobre la población, en la *Figura 61* una gráfica con el progreso de la aptitud a lo largo de las generaciones y en la *Figura 62* los cuatro mejores fenotipos de la última generación.

Figura 60 – Información sobre la población obstáculo I

Figura 61 – Progreso de la aptitud obstáculo I

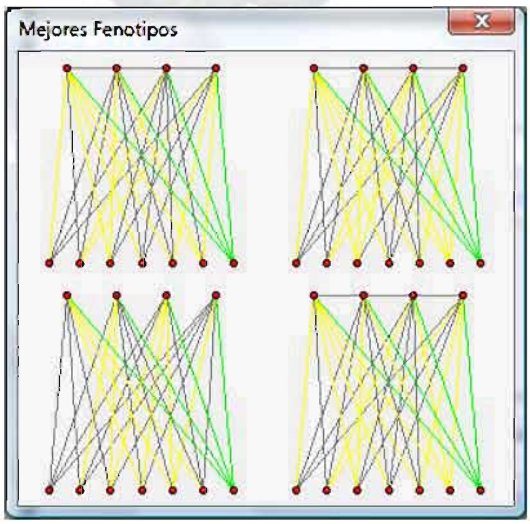


Figura 62 – Mejores Fenotipos obstáculo I

En la *Figura 63* podemos ver una captura de pantalla con indicaciones del comportamiento exhibido por los NPCs. Se observaron tres tendencias claras: un grupo que se movía hacia la izquierda y usaba la pared como guía para llegar hasta el objetivo, frente a otros dos que claramente rodeaban el muro unos por la izquierda y otros por la derecha para luego dirigirse directamente al objetivo.

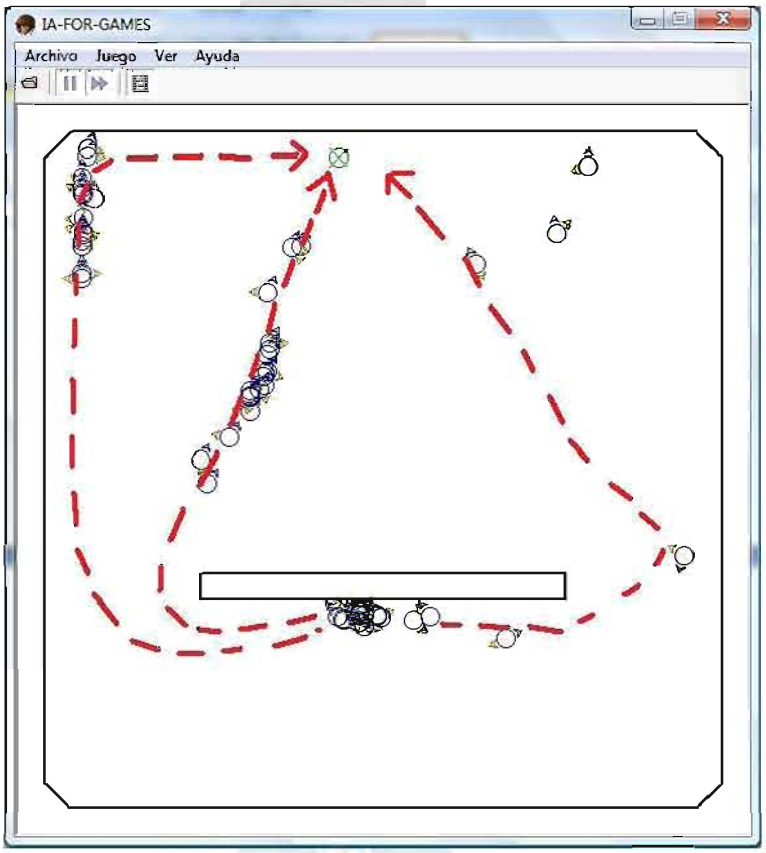


Figura 63 – Comportamientos de los NPCs con NEAT obstáculo I

Viendo el número de generaciones que se ha tardado en encontrar soluciones aceptables, y la estructura de los mejores fenotipos de la última generación (ni siquiera tienen neuronas ocultas), podemos asegurar que este mapa no supone ningún reto para nuestros NPCs.

4.4.2 MAQUINAS DE ESTADOS FINITOS

Nuestro NPCs en varias pruebas realizadas no llegaba al punto objetivo, y en otras ocasiones lo lograba, pero por casualidad. Pero todos los NPCs presentaban comportamientos erráticos (*Figura 64*). En la Figura 65 se ve la información de la población.

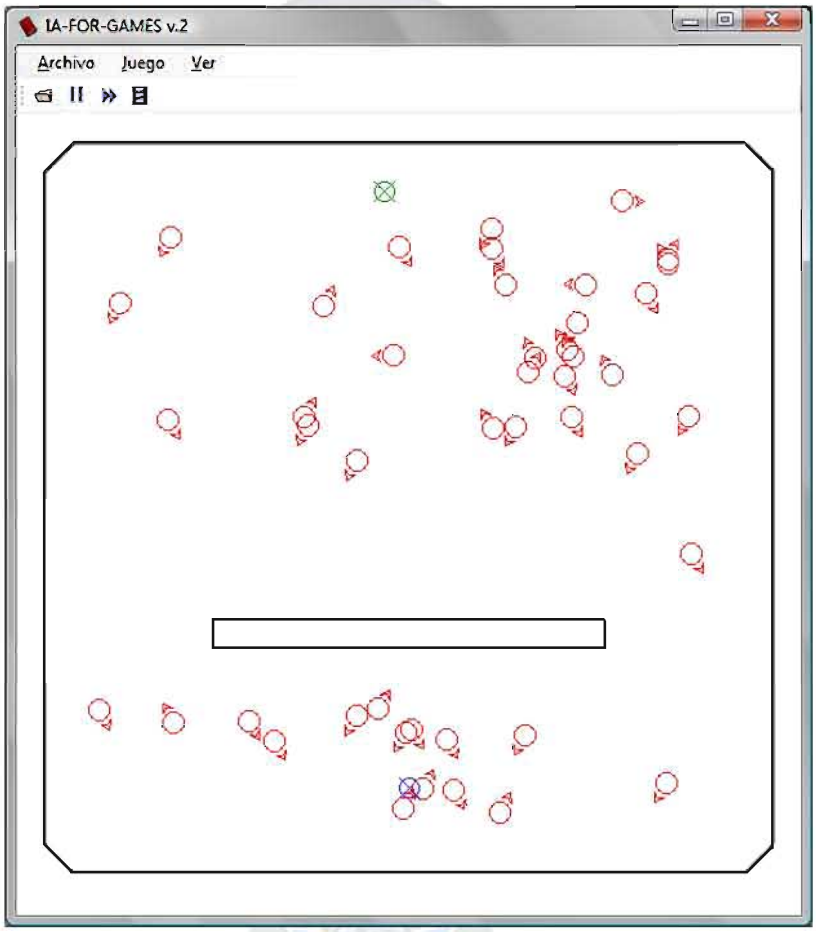


Figura 64 - Comportamiento de los NPCs con FSMs (obstáculo I)

Figura 65 – información de la población de NPCs con FSMs (obstáculo I)

4.4.3 ANÁLISIS

Se va a afirmar que nuestros NPCs que usan redes neuronales con NEAT obtienen un puntaje superior o de al menos 115 puntos, lo que implicaría que los 50 individuos se aproximaron bastante al objetivo. Se utilizara el método de prueba de hipótesis (Ver 2.8)

Para tratar de probar a un nivel de significancia de 0.05 se realizaron n = 30 pruebas de 2 minutos de duración, *(Ver Tabla 3)*.

Nro Prueba	Aptitud media - Redes neuronales con NEAT (Xi)	Aptitud media- Maquina de estados finitos (Xj)	Aptitud esperada	Tiempo de simulacion
1	112,279	57,872	125	2 min
2	116,538	42,758	125	2 min
3	118,953	42,604	125	2 min
4	114,345	40,089	125	2 min
5	116,389	36,001	125	2 min
6	118,331	48,845	125	2 min
7	117,342	41,510	125	2 min
8	114,342	35,604	125	2 min
9	119,203	44,173	125	2 min
10	116,343	37,479	125	2 min
11	114,668	43,933	125	2 min
12	116,345	49,082	125	2 min
13	119,985	50,264	125	2 min
14	114,345	40,692	125	2 min
15	115,436	48,107	125	2 min
16	111,345	43,575	125	2 min
17	116,439	48,986	125	2 min
18	117,342	49,014	125	2 min
19	117,435	36,486	125	2 min
20	115,873	51,717	125	2 min
21	118,006	54,235	125	2 min
22	113,892	48,283	125	2 min
23	115,359	52,034	125	2 min
24	116,274	53,308	125	2 min
25	118,495	52,674	125	2 min
26	116,395	50,739	125	2 min
27	114,239	51,844	125	2 min
28	118,435	49,223	125	2 min
29	110,234	50,107	125	2 min
30	106,719	45,147	125	2 min
$\bar{X}$	115,711	46,546	125	2 min
S	2,840	5,886		

Tabla 3 – Resultados mapa obstáculo I

Teniendo los datos de 30 pruebas, que es lo mínimo necesario para

suponer que la población es normal, la distribución en el muestreo de la media seguiría una distribución t con n-1 grados de libertad. Además usando un nivel de significancia de 0.05 se realizara la prueba de hipótesis

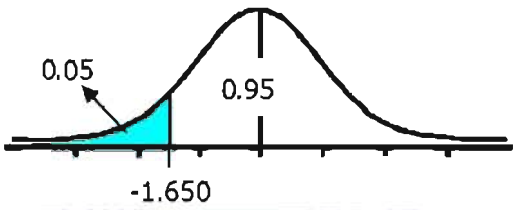
Formulamos las hipótesis.

H_0: $\mu \geq 115$, Los NPCs que usan redes neuronales tienen un puntaje ≥ 115

H_1: $\mu < 115$, Los NPCs que usan redes neuronales tienen un puntaje < 115

Determinar la distribución de probabilidad y la región critica.

Como el nivel de significancia es del 5% = 0.05 se calcula la región critica. 1 – 0.05 = 0.95, valor que se busca en la tabla de la distribución Normal, obteniendo el valor de $Z_{0.95}$ = 1.65.



De la tabla se calculan los estadísticos:

$$S = \sqrt{\frac{\sum (x - \bar{x})^2}{(n-1)}}$$

$$\bar{X} = \frac{\sum x_i}{n}$$

$\bar{X}$ = 115.711 y S = 2.840

Donde:

$\bar{X}$: Media poblacional.

S: Desviación estándar de una muestra.

Calcular el estadístico de prueba:

Se calcula el estadístico $t = (\bar{x} - \mu_0)/(s/\sqrt{n})$ = (115.711 - 115)/(2.840/√30)

Remplazando los datos se tiene t = 1.371

Tomar una decisión:

No está en la zona de rechazo por lo cual se acepta H_0

Ahora para la Maquina de estados finitos.

H_0: *$\mu \geq 115$, Los NPCs que usan Maquina de estados finitos tienen un puntaje ≥ 115 puntos*.
H_1: *$\mu < 115$, Los NPCs que usan Maquina de estados finitos tienen un puntaje < 115 puntos*.

Realizando de la misma manera t = (46.546 - 115) / (5.886 / √30). Se obtuvo t = - 63,695, valor que se encuentra en la zona de rechazo por lo cual se rechaza H_0 y se acepta H_1.

Haciendo esta prueba de hipótesis podemos asegurar que los NPCs que usan redes neuronales con NEAT siempre obtendrán un puntaje mayor o igual a 115 en esta prueba, puntaje que se acerca al resultado óptimo esperado que es 125. Por otro lado los NPCs que usan maquina de estados finitos siempre estarán por debajo de los 115 puntos.

4.5 OBSTÁCULO II

En este mapa se añade otro muro, y un estrechamiento a mitad de camino hacia el objetivo, al igual que en el anterior mapa los NPCs aparecerán en la parte inferior y tendrán que llegar a la parte superior cruzando más obstáculos. Véase la *Figura 66*.

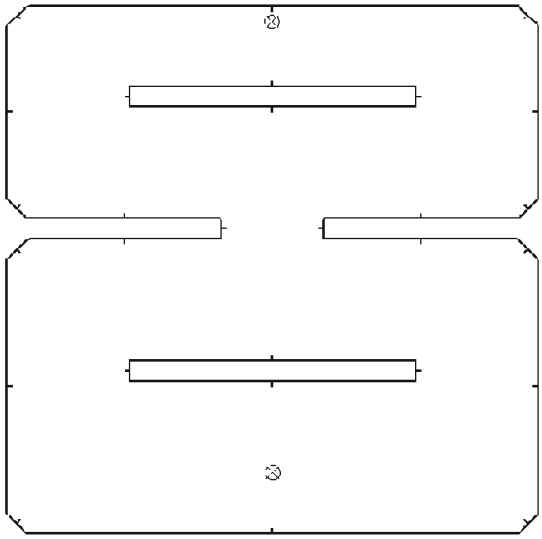


Figura 66 – Mapa con obstáculos II

4.5.1 REDES NEURONALES CON NEAT

Los parámetros usados fueron los mismos que en la prueba anterior. Después de 28 generaciones se detuvo la ejecución. En la *Figura 67* puede verse información sobre la población, en la *Figura 68* la gráfica de la aptitud, y en la *Figura 69* los cuatro mejores fenotipos.

Figura 67 - Información de la población (obstáculos II)

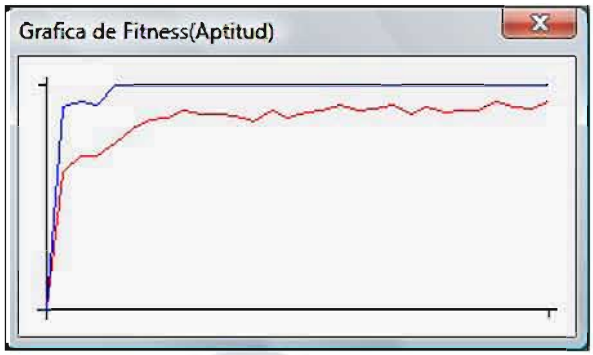


Figura 68 - Progreso de la aptitud (obstáculos II)

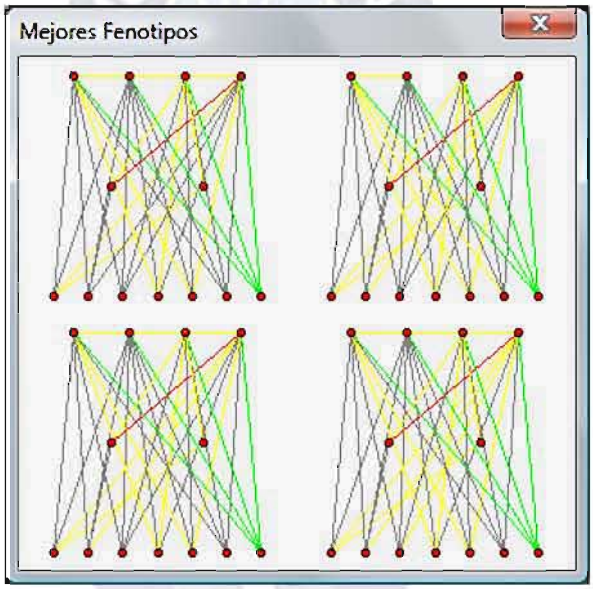


Figura 69 – Mejores fenotipos (obstáculos II)

Durante las primeras generaciones parecían incapaces de alcanzar el objetivo, hasta que en la séptima uno de los individuos esquivó todos los obstáculos correctamente (se puede apreciar en la gráfica del fitness un salto en la línea azul). A partir de ahí no tuvieron demasiados problemas, escogiendo los caminos representados en la *Figura 70*.

Tras estos experimentos queda claro que, usando las redes neuronales con NEAT los NPCs pueden navegar por un mapa esquivando sin ningún desafío.

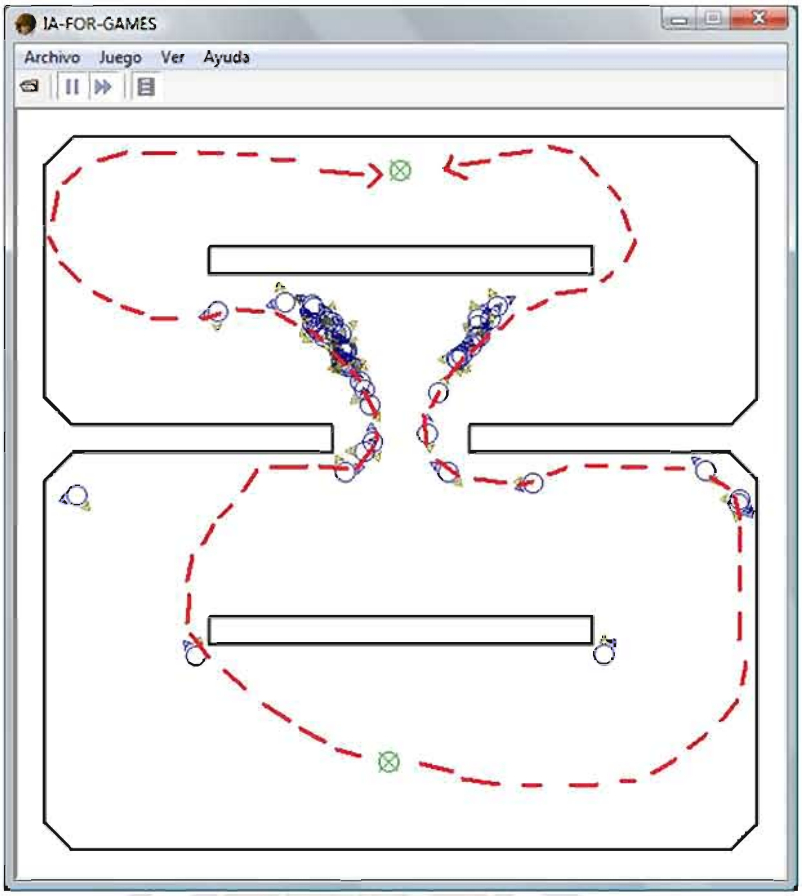


Figura 70 – Comportamiento de NPCs (obstáculos II)

4.5.2 MAQUINA DE ESTADOS FINITOS

Esta vez también la mayoría de los NPC no lograron superar los obstáculos algunos lograron llegar. Al parecer será imposible que lo logren, claro sin proporcionarles la solución al problema. En *Figura 71* la puede verse los resultados obtenidos, y el comportamiento mostrado (*Figura 72*).

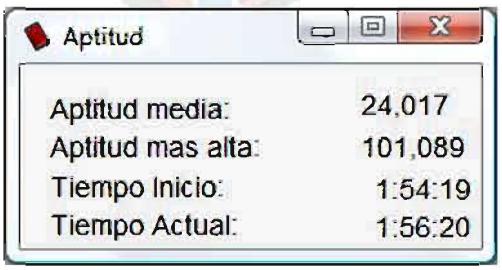


Figura 71 – Información de la población de NPCs con FSMs (obstáculo II)

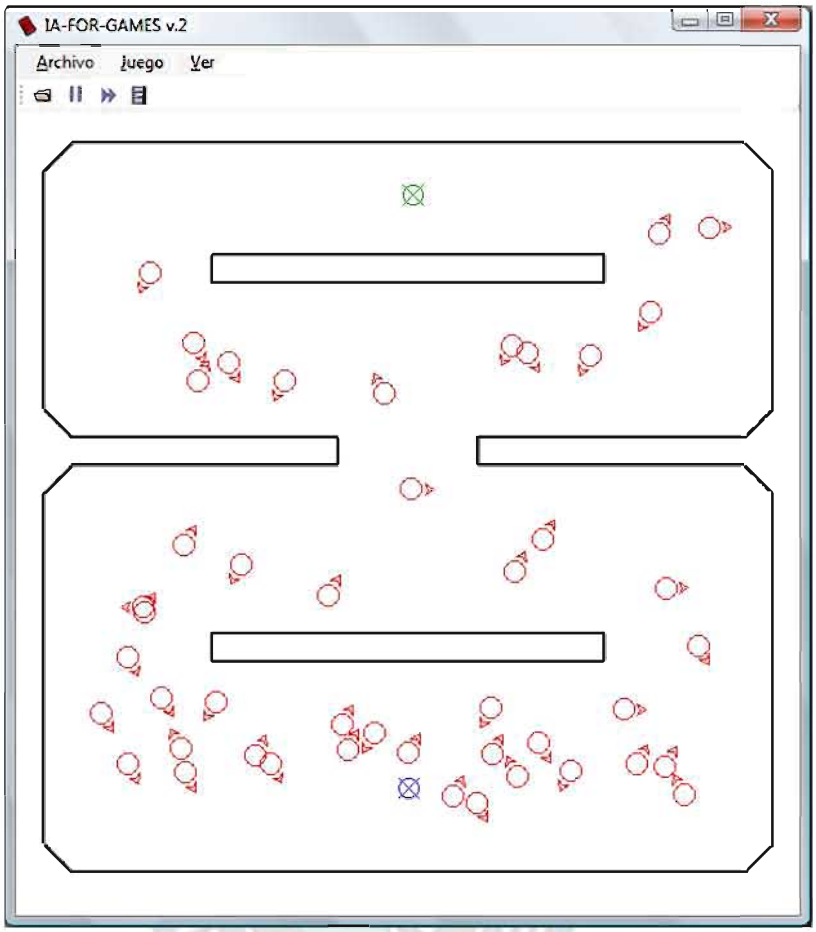


Figura 72 – Comportamiento de NPCs con FSMs (obstáculo II)

4.5.3 ANÁLISIS

Se realizo la misma evaluación de la primera prueba y se obtuvieron los siguientes resultados (ver *Tabla 4*)

Nro Prueba	Aptitud media - Redes neuronales con NEAT (Ai)	Aptitud media- Maquina de estados finitos (Aj)	Aptitud esperada	Tiempo de simulacion
1	115,790	49,685	125	2 min
2	115,181	52,225	125	2 min
3	116,172	48,851	125	2 min
4	115,743	41,066	125	2 min
5	115,830	48,695	125	2 min
6	117,466	36,055	125	2 min
7	114,118	41,815	125	2 min
8	112,107	53,502	125	2 min
9	116,649	35,934	125	2 min
10	115,195	41,070	125	2 min
11	116,359	51,837	125	2 min
12	113,925	48,316	125	2 min
13	114,728	47,291	125	2 min
14	114,211	49,905	125	2 min
15	112,999	49,801	125	2 min
16	116,941	40,683	125	2 min
17	113,854	43,295	125	2 min
18	116,170	41,108	125	2 min
19	116,481	52,104	125	2 min
20	114,372	41,988	125	2 min
21	114,830	42,914	125	2 min
22	118,720	40,956	125	2 min
23	115,278	35,565	125	2 min
24	113,368	39,687	125	2 min
25	113,769	45,891	125	2 min
26	117,221	46,380	125	2 min
27	115,757	35,414	125	2 min
28	114,758	43,291	125	2 min
29	113,230	44,667	125	2 min
30	115,788	24,017	125	2 min
$\bar{x}$	115,234	43,800	125	2 min
s	1,483	6,479		

Tabla 4 – Resultados mapa obstáculos II

Se formulo las siguientes hipótesis:

H_0: $\mu \geq 115$, Los NPCs que usan redes neuronales tienen un puntaje ≥ 115

H_1: $\mu < 115$, Los NPCs que usan redes neuronales tienen un puntaje < 115

Realizando de la misma manera t = (115.234 - 115) / (1.483 / √30). Se obtuvo t = 0.862 por lo que se acepta H_0,

Para la Maquina de estados finitos.

H_0: $\mu \geq 115$, Los NPCs que usan Maquina de estados finitos tienen un puntaje ≥ 115 puntos.

H_1: $\mu < 115$, Los NPCs que usan Maquina de estados finitos tienen un puntaje < 115 puntos.

Calculando t = (115.234 - 115) / (1.483 / √30). Se obtuvo t= - 60,195, valor que se encuentra en la zona de rechazo por lo cual se rechaza H_0 y se acepta H_1

De la misma manera podemos asegurar que los NPCs que usan redes neuronales con NEAT siempre obtendrán un puntaje mayor o igual a 115 en esta prueba. Por otro lado los NPCs que usan maquina de estados finitos siempre estarán por debajo de los 115 puntos.

4.6 DETECCIÓN DE ENEMIGOS

Este mapa, incluye torretas además de obstáculos, como se puede observar en la *Figura 73*. Los NPCs parten de la zona inferior del mapa, y deben llegar a la superior. Para ello, deberán aprender a detectar la amenaza de las torretas y tomar el camino de la izquierda, en vez de seguir recto hacia una muerte segura. Aunque las torretas pueden disparar, nuestros NPCs no dispondrán de munición. No nos interesa que las ataquen, sólo que las eviten y lleguen al objetivo.

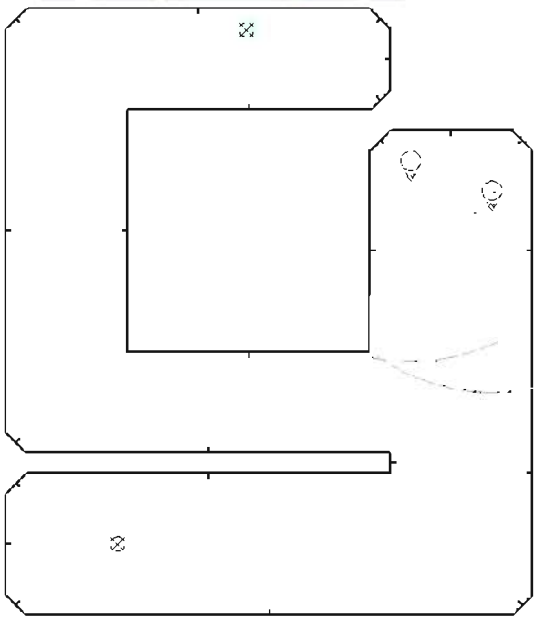


Figura 73 - Mapa para detección de enemigos.

4.6.1 REDES NEURONALES CON NEAT

Los parámetros más importantes de este mapa son los siguientes:

- *Tamaño de la población*: 40 individuos
- *Sensores de obstáculos*: 5, repartidos en 180º
- *Sensores de enemigos*: 5, repartidos en 120º
- *Tasa de cruce*: 0,7
- *Tasa de mutación*: 0,2
- *Probabilidad de añadir una neurona*: 0,04
- *Probabilidad de añadir un enlace*: 0,07
- *Probabilidad de añadir un enlace recurrente*: 0,05
- *Número máximo de especies*: Sin límite
- *Umbral*: 0,26

En este caso se van a usar todos los sensores: cinco sensores de obstáculos, el detector de colisiones, otros cinco sensores de enemigos, el indicador de objetivos, y el de la línea de fuego enemiga. Si añadimos el bias, tenemos un total de quince neuronas en la capa de entrada.

En una primera simulación no se mostraron resultados interesantes, en una segunda esta vez con éxito se simularon 50 generaciones, los resultados mostraron tres tendencias bien marcadas así que se detuvo la simulación la cual lanzo los siguientes resultados. En la *Figura 74* puede verse información sobre la población, en la *Figura 75* la gráfica de la aptitud, y en la *Figura 76* los cuatro mejores fenotipos.

Figura 74 – Información de la población (Detección de enemigos)

Figura 75 – Progreso de la aptitud (Detección de enemigos)

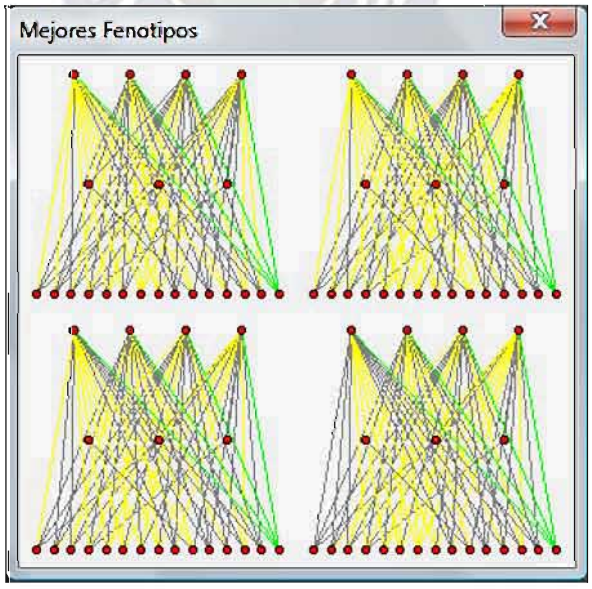


Figura 76 – Mejores Fenotipos (Detección de enemigos)

Durante las 20 primeras generaciones, la mayoría de los nuestros NPCs se dirigía a la derecha y, o bien se paraban al llegar a la pared, o intentaban subir y morían a causa de las torretas. Alrededor de la generación 32 uno de los NPCs, al detectar las torretas, decidió girar a la izquierda, encontrando el camino hacia el objetivo. En generaciones sucesivas se fue incrementando el número de agentes que escogían este camino, recorriéndolo cada vez más eficientemente.

Pero también existían dos tendencias más. Un grupo se dirigía directamente hacia las torretas intentando acercarse lo más posible al punto objetivo, no obstante a pesar de muchas bajas algunas lograron ubicándose tras las torretas, otro grupo de NPCs seguía una pauta muy distinta. Viendo que los que avanzaban hacia la derecha tendían a morir, estos se quedaban parados en la primera pared que encontraban. En la *Figura 77* puede verse las rutas que los NPCs tomaron.

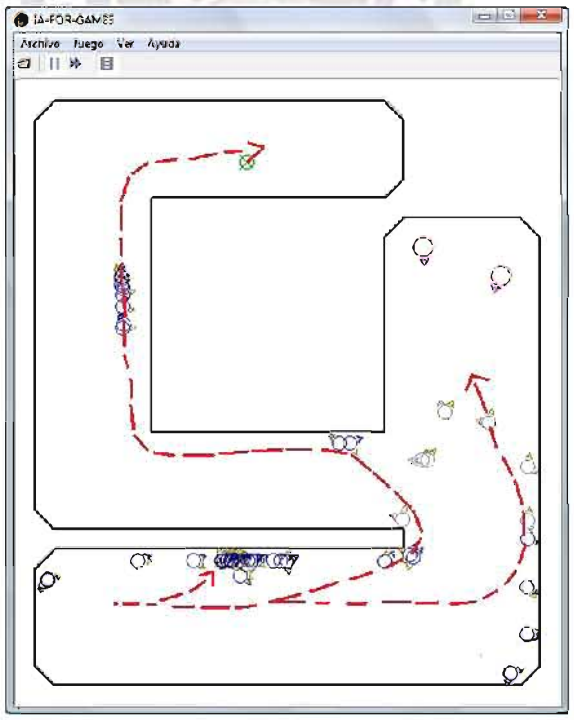


Figura 77 – Comportamiento de NPCs (Detección de enemigos)

4.6.2 MAQUINA DE ESTADOS FINITOS

La mayoría de los NPC no lograron superar los obstáculos y acabaron muertos (Están en recuadro), pero algunos lograron llegar (Ver Figura 78). En la *Figura 79* puede verse los resultados obtenidos.

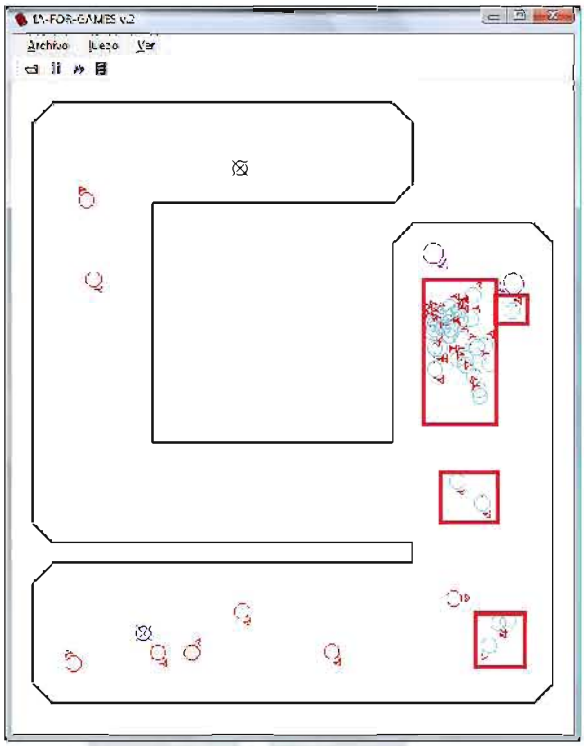


Figura 78 – Comportamiento NPCs con FSMs (Detección de enemigos)

Figura 79 - Información de la población de NPCs con FSMs (Detección de enemigos)

4.6.3 ANÁLISIS

Se realizaron 30 pruebas de 5 min aproximadamente. En *Tabla 5* la puede

verse los resultados obtenidos

Nro Prueba	Aptitud media - Redes neuronales con NEAT (Ai)	Aptitud media- Maquina de estados finitos (Aj)	Aptitud esperada	Tiempo de simulacion
1	91,745	31,976	125	5 min
2	88,575	42,178	125	5 min
3	91,026	36,348	125	5 min
4	90,954	36,191	125	5 min
5	90,173	43,467	125	5 min
6	90,606	49,814	125	5 min
7	91,726	39,616	125	5 min
8	91,271	36,026	125	5 min
9	89,701	32,228	125	5 min
10	90,146	34,872	125	5 min
11	91,378	38,717	125	5 min
12	89,840	41,670	125	5 min
13	88,669	41,897	125	5 min
14	91,366	35,438	125	5 min
15	89,671	39,996	125	5 min
16	91,779	36,618	125	5 min
17	89,303	42,764	125	5 min
18	90,262	43,875	125	5 min
19	90,219	32,955	125	5 min
20	89,883	37,091	125	5 min
21	91,402	41,059	125	5 min
22	90,046	41,029	125	5 min
23	91,237	44,196	125	5 min
24	90,996	44,631	125	5 min
25	91,560	30,208	125	5 min
26	91,457	41,158	125	5 min
27	90,562	42,997	125	5 min
28	90,746	43,039	125	5 min
29	90,557	38,884	125	5 min
30	83,846	42,161	125	5 min
$\bar{x}$	90,357	39,437	125	5 min
s	1,507	4,480		

Tabla 5 – Resultados mapa con torretas fijas

Se hizo las hipótesis:

H_0: Los NPCs que usan redes neuronales tienen un puntaje ≥ 90

H_1: Los NPCs que usan redes neuronales tienen un puntaje < 90

Calculando t = (90.357 - 90) / (1.507 / √30). Se obtuvo t = 1,297 por los que se acepta H_0

Para maquina de estados finitos.

H_0: Los NPCs que usan Maquina de estados finitos tienen un puntaje ≥ 90

H_1: Los NPCs que usan Maquina de estados finitos tienen un puntaje < 90

Calculando t = (39.437 - 90) / (4.480/ √30). Se obtuvo t = -61,818 por lo que se rechaza H_0

5 CONCLUSIONES

CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES

Luego de la recolección de la recolección de datos de prueba y su respectivo análisis, vistos en el capitulo anterior. Se termina el presente documento detallando los objetivos cumplidos.

5.1 CONCLUSIONES

Se profundizo en el tema de IA aplicada a la programación de videojuegos, mediante la incorporación de redes neuronales artificiales como cerebro de un NPC para que pudiera desarrollar determinadas acciones.

Se obtuvo distintos comportamientos, después de hacer las pruebas (Leer Capitulo 4) se pudo observar, que las redes neuronales con NEAT y las Maquinas de estados finitos mostraron una gran diferencia en cuanto a la forma de encarar soluciones a los mapas probados.

Se logro incorporar una red neuronal artificial que hace uso de la neuroevolución de topologías aumentativas (NEAT) en una población de NPCs. Ya que NEAT realiza la evolución de la topología de dicha red neuronal, combinando las ventajas de los algoritmos genéticos y las redes neuronales.

Se logro representar en movimiento de los NPCs mediante el entorno de experimentación IA FOR GAMES (Ver 3.5.1),que permitió observar en tiempo real el proceso de aprendizaje en el caso de las redes neuronales y desenvolvimiento en el caso de las maquinas de estados finitos de cada uno de los individuos dentro de mapas con obstáculos.

Se implementaros NPCs controlados por maquina de estados finitos y NPC controlados por redes neuronales artificiales en un entorno de experimentación denominado IA FOR GAMES.

Se pudo evidenciar las ventajas y desventajas de las redes neuronales con NEAT:

- Representan de una forma especializada un determinado comportamiento.
- Pueden hallar varios tipos de soluciones a un mismo problema.
- La implementación es un poco morosa, pero los resultados lo valen.
- Presentan la desventaja de que una red neuronal es entrenada solo para un determinado entorno. Si el entorno cambiara se tendría que volver a entrenar la red.

Se pudo evidenciar las ventajas y desventajas de las maquinas de estados finitos.

- Su implementación es muy fácil.
- No pueden representar comportamientos complejos a menos de que dicho comportamiento tenga una representación algorítmica.
- Se tiene que tomar en cuenta todas las posibles acciones que los NPCs podrían realizar dentro de un entorno.

Se evaluaron los resultados obtenidos (Ver Capítulo 4). Mediante la captura de la información proporcionada por las ventanas de dialogo que poseen los entornos de experimentación. Y analizando los resultados mediante el método de prueba de hipótesis.

Se configuraron los parámetros más importantes para que la red neuronal pueda hacer uso de NEAT (Ver 4.4.1). A la hora de inicializar esta red neuronal para que sea entrenada.

Se logro evolucionar la topología de la red neuronal siguiendo los estímulos

del entorno de experimentación. Logrando realizar tareas específicas y especializadas para dicho entorno.

Se implemento ventanas de dialogo en los entornos de experimentación, en las que se muestran los resultados de cada prueba para su interpretación (Ver 4.3).

Se analizaron los comportamientos de los NPCs controlados por una maquina de estados finitos y NPCs que usan redes neuronales artificiales, usando distintos mapas, unos con obstáculos, detección de enemigos.

Por todo lo anterior podemos afirmar que las redes neuronales si presentaron un comportamiento más estilizado y especializado en las tareas que se propusieron, en comparación a las maquinas de estados finitos.

5.2 RECOMENDACIONES

NEAT resulto ser un método poderoso para la evolución de redes neuronales artificiales aplicadas a problemas complejos. Las pruebas realizadas demuestran que NEAT permiten resolver problemáticas como evasión de obstáculos, estrategias de alto nivel, comportamiento, detección de enemigos.

Se recomendaría usar las redes neuronales con NEAT, para proyectos de robótica, ya que un personaje de videojuego puede ser considerado una versión digital de un robot.

REFERENCIAS BIBLIOGRAFÍCAS

REFERENCIAS BIBLIOGRAFICAS

[Gutiérrez Orozco, 2008] Jorge Alejandro Gutiérrez Orozco. "*Maquinas de Estados Finitos*". Escuela Superior de Cómputo, 2008.

[Andre, David y Teller, Astro, 1999] Andre, David y Teller, Astro. *"Evolving Team Darwin United"*. 1999.

[B. Schwab, 2004]. B. Schwab. *"AI Game Engine Programing"*. Chales River Media, 2004.

[Begio, 1994] Begio, R.K., Mclnerney, J. Frasconi, P. *"Learning long-term dependencies with gradient descent is difficult"*. 1994.

[Gomez, F. y Miikkulainen, R., 1999] Gomez, F. y Miikkulainen, R. *"Solving non-Markovian control tasks with neuroevolution"*. Austin, Texas: The University of Texas at Austin, 1999.

[Kaelbling, L.P., Littman, M. y Moore, A.W., 1996] Kaelbling, L.P., Littman, M. y Moore, A.W. *"Reinforcement learning: A survey"*. Journas of Artificial Intelligence, 1996.

[Buckland, 2002] M. Buckland. *"AI Techniques for Game Programing"*. Premier Press, 2002.

[Moriarty, D.E. y Miikkulainen, R., 1996] Moriarty, D.E. y Miikkulainen, R. *"Efficient reinforcement learning through symbiotic evolution"*. Machine Learning, 1996.

[Moriarty, D.E. y Miikkulainen, R., 1997] Moriarty, D.E. y Miikkulainen, R. *"Forming neural networks through efficent adaptative co-evolution"*. Evolutionary Computation, 1997.

[Stanley Kenneth O. y Miikkulainen Risto, 1999] Stanley Kenneth O. y Miikkulainen

Risto. *"Evolving Neural Networks througt Augmenting Topologies"*. Evolutionary Computation, 1999.

[Viscuso, 2004] Viscuso, German. *"Neuroevolucion de Topologias Aumentativas"*. 2004.

[Coello, 2002] Coello, C., "*Introducción a los algoritmos Genéticos*". Universidad de Tulane, Nueva Orleáns, EUA. 2002.

[DeJong, 2007] DeJong J.,*"Computación Evolutiva"*. 4ta. Edición, 2000, New York: IEEE Press.

[Mitchell, 1997] Mitchell M.,"*An Introduction to Genetic Algorithms*", Massachusetts, 3ra.Edición. 1997

[Larrañaga, 2000] Larrañaga P., "*Algoritmos Genéticos y Computación Evolutiva*", **[Fecha de visita: 13/11/2008]**.
http://www.Algoritmos genéticos y computación evolutiva.htm

[Jesús Alfonso López, 2000] Tripod, "*Home page sobre Algoritmos Genéticos*", 2000. **[Fecha de visita: 14/04/2009]**
http://members.tripod.com/jesus_alfonso_lopez/AgIntro.html

[Gastón Crevillén y David Díaz, 2001] Nexus 7, "Home page sobre Algoritmos Genéticos", 2001. **[Fecha de visita: 22/04/2009]**
http://www.cinefantastico.com/nexus7/ia/ageneticos.htm

[Wikipedia1, 2005] Genotipo. **[Fecha de visita: 20/05/2008]**.
http://es.wikipedia.org/wiki/Genotipo

[Wikipedia2, 2005] Fenotipo. **Fecha de visita: 20/05/2008]**.
http://es.wikipedia.org/wiki/Fenotipo

DOCUMENTOS