MÁSTER EN INGENIERÍA INFORMÁTICA Y DE TELECOMUNICACIÓN

TRABAJO DE FIN DE MÁSTER

OBTENCIÓN AUTOMÁTICA DE PATRONES DE COMPORTAMIENTO PARA AGENTES VIRTUALES DE VIDEOJUEGOS MEDIANTE TÉCNICAS DE PROGRAMACIÓN GENÉTICA

Autor: Álvaro Valera Vázquez

Tutor: David Camacho Fernández

Septiembre de 2012

A mi padre.

Índice

1. Introducción

Tal vez haya quien todavía se sorprenda al hablar de videojuegos en un proyecto académico. Y no es de extrañar tal actitud si pensamos que los videojuegos han sido tradicionalmente asociados al mundo del ocio como meras formas de entretenimiento para niños y jóvenes. Curiosamente, dicha creencia durante años ha chocado con la de las nuevas generaciones de estudiantes que empezaban ingeniería informática esperando aprender allí a programar videojuegos desde el primer día y sin esfuerzo, encontrándose con la (¿dura?) realidad que ni la programación de videojuegos forma parte del temario de ninguna asignatura, ni es posible afrontar esta carrera si no es con trabajo y bagaje en aplicaciones de todo tipo.

Lo cierto es que los videojuegos son un tipo de *software* de aplicación (1) más. Programar un videojuego consiste esencialmente en el diseño y desarrollo de un motor del juego, que guía la lógica de la aplicación, junto con una serie de interfaces a través de cuales el usuario puede interactuar con dicho motor con el objetivo de divertirse (i.e. jugar). Adicionalmente, y cada vez más frecuentemente, un videojuego puede ser jugado *online* y estar sujeto a determinadas restricciones *hardware* impuestas por la arquitectura de la máquina que va a ejecutar dicho videojuego. Así pues, el diseño y desarrollo de un videojuego puede ser estudiado y tratado con las mismas pautas formales que otro *software* de aplicación cualquiera.

Superados esos posibles recelos iniciales, encontramos en los videojuegos otra característica muy interesante: la inteligencia artificial de los personajes no jugadores, tradicionalmente denominados NPC (del inglés *Non-player characters*), o simplemente *bots* controlados por la máquina. Como se verá con mayor detalle en capítulos posteriores, la inteligencia artificial en videojuegos tiene ciertas características propias muy particulares debido al objetivo último de los videojuegos: ¡divertirse! Dichas peculiaridades, lejos de limitar el ámbito de aplicación en este dominio de la inteligencia artificial, diversifican las posibilidades de la misma, haciendo de los videojuegos un excelente banco de pruebas para diversas familias de algoritmos de inteligencia artificial clásica con un sinfín de retos particulares que afrontar en cada caso.

De entre todos los planteamientos posibles en inteligencia artificial, se ha optado en este trabajo por la rama de la computación evolutiva, y en concreto de la programación genética, como se verá más adelante.

En este trabajo también hablamos de zombis. Aceptado su uso por la RAE[1], y conocidos de sobra en el mundo del cine y la literatura, encontraremos en ellos un ecosistema de seres zombis (i.e. según la RAE, “atontado, que se comporta como un autómata”) cuyas prefijadas reglas de comportamiento conformarán, junto a otros elementos de entorno, el marco en el que trataremos de hacer emerger, programación genética mediante, comportamientos inteligentes individuales y de grupo en los agentes humanos supervivientes. Como se analizará, este aparentemente nuevo paradigma superviviente-zombi no es sino una extensión al clásico modelo depredador-presa, ampliamente estudiado en el área de la simulación multiagente.

[1] http://buscon.rae.es/drae/?type=3&val=zombi&val_aux=&origen=REDRAE

2. Estado del arte en programación genética

2.1 Principios bioinspirados

Dejando a un lado la controversia existente acerca del origen de las especies entre las diversas teorías o corrientes de pensamiento existentes, quizás de las cuales son el Evolucionismo y el Creacionismo las más conocidas y enfrentadas, lo cierto es que el concepto de la selección natural, tal y como fue introducido por Charles Darwin en su conocido libro *El origen de las especies* (2), es una poderosa herramienta de adaptación de una población de individuos de una especie a las particularidades de su entorno.

Desde un punto de vista biológico podemos pensar en una población como un conjunto de individuos que se corresponden con una dualidad de realidades distintas pero intrínsecamente relacionadas: Lo que el individuo internamente es, su genotipo, frente a lo que el individuo externamente manifiesta, su fenotipo. Suponiendo que, efectivamente, el fenotipo es una consecuencia del genotipo, y que los individuos de una generación dada son capaces de generar descendencia mediante operadores genéticos como el cruce entre 2 ó más progenitores o la mutación de individuos sueltos, la evolución es el proceso mediante el cual, de manera gradual, cada nueva generación será más apta para su entorno, entendiendo dicha aptitud como una mayor capacidad para generar descendencia que garantice la supervivencia de la especie.

2.2 Computación evolutiva

El modelo y la teoría de adaptación mediante selección natural fueron llevadas a la computación científica por Holland (3) y Goldberg (4) creando un nuevo paradigma de búsqueda u optimización menos orientado en los enfoques matemáticos tradicionales y más apoyado en la propia sabiduría y robustez estocástica de la naturaleza. Este primer avance se materializó en el conocido como algoritmo genético clásico (*Genetic Algorithm* o GA), y dio paso a toda una nueva generación de algoritmos que recibió el nombre de computación evolutiva (*Evolutionary Computation* o EC) (5).

Mediante un GA podemos, por ejemplo, encontrar los puntos singulares de una función matemática de una o múltiples variables dada, $f(x)$, cuya expresión derivada, $f'(x)$, suponemos demasiado compleja como para poder resolver la tradicional ecuación $f'(x) = 0$, si vemos el problema de la siguiente manera:

- Cada individuo representa una posible solución (i.e. un hipotético punto singular) al problema:
  - El genotipo es una secuencia de bits (i.e. los genes) fija
  - El fenotipo es la tupla de valores que se asignará a la variable o variables enteras x de la función y se obtiene separando el genotipo en bloques de bits, uno por variable, multiplicando cada bit por una potencia sucesiva de 2 y sumándolo obteniendo el correspondiente valor por variable

    - La aptitud o *fitness* es el propio valor de $f(x)$

- La población consiste, pues, en un conjunto de soluciones para el problema, y se inicializa tomando secuencias de bits aleatorias para cada individuo
- Para la siguiente generación se van tomando sucesivamente progenitores de manera individual o por parejas con probabilidad proporcional a su *fitness* hasta completar una población entera nueva de descendientes:
    - Una pareja de individuos se cruza tomando un punto aleatorio en sus cadenas de bits recombinando cada parte de la cadena un progenitor con la parte opuesta del otro
    - Un individuo se muta modificando de manera aleatoria uno o varios de sus bits

- El proceso se repite hasta que se alcanza algún criterio de convergencia (e.g. individuos muy parecidos entre sí, número de generaciones máximo alcanzado, etc.)

En total las familias de algoritmos más importantes en EC son cuatro:

- Algoritmos genéticos
- Programación genética
- Programación evolutiva
- Estrategias evolutivas

Cada familia de algoritmos evolutivos tiene sus propias características y variantes aparecidas a lo largo de su historia, y cada una podría ser analizada en profundidad, pero la técnica que copa este trabajo es la conocida como programación genética (*Genetic Programming* o GP) (6) (7), por lo que sólo comentaremos algunos de los detalles principales de cada una de las demás, aparte de los ya introducidos algoritmos genéticos. La programación evolutiva fue introducida originalmente por Lawrence J. Fogel (8) en el contexto de máquinas de estados finitos con el objetivo de optimizar sus parámetros (e.g.: probabilidades de transición, arcos, estados, etc.) para utilizarlas como predictores. Las estrategias evolutivas, contribución de Hans-Paul Schwefel (9), surgieron originalmente en el dominio de la optimización de problemas con vectores de valores reales, diferenciándose esencialmente de los GA en cuanto a que aquí se emplea un único progenitor cuya descendencia se explota en mayor medida para avanzar a la siguiente generación.

2.3 Programación genética

2.3.1 Representación

La principal particularidad de los algoritmos de GP es que el objeto de la evolución son programas informáticos en sí mismos, en vez de parámetros o variables de un problema. De entre las diversas aproximaciones existentes a estos algoritmos la descrita por John R. Koza (6) es la más popular y extendida.

En el planteamiento de Koza los individuos se representan mediante árboles, no existiendo separación entre el genotipo y el fenotipo. Estos árboles se componen de nodos conectados entre sí de manera jerárquica padre-hijo tal y como se puede ver en el ejemplo de la Figura 1.

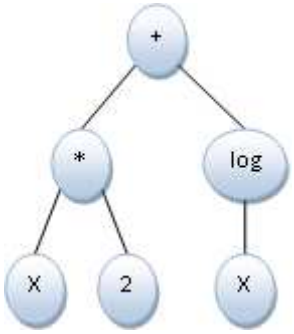


Figura 1. Ejemplo de árbol GP para expresiones matemáticas con operadores, variables y constantes

Cada nodo del árbol contiene una función perteneciente al conjunto de funciones empleado en el problema, de manera que, leyendo adecuadamente el árbol, es posible recomponer fácilmente el programa representado. En el ejemplo de la Figura 1 el programa sería la expresión matemática: $2 * X + \log(X)$, empleando como funciones las operaciones matemáticas de la suma ("+"), el producto ("*") y el logaritmo ("log"), así como variables ("X") y constantes ("2").

El nodo raíz es el más superior del árbol. Bajo él, los nodos que ocupan las posiciones intermedias están ocupados por funciones con hijos, que reciben el nombre de funciones no terminales. Los nodos situados en la parte más inferior del árbol son conocidos como hojas, y albergan funciones que no tienen hijos, también conocidas como terminales.

Se denomina profundidad de un nodo a la cantidad de nodos que hay que atravesar desde el nodo raíz hasta llegar a dicho nodo. La profundidad de un árbol es la mayor de las profundidades de sus nodos hoja.

2.3.2 Inicialización de la población

En GP la inicialización de la población tiene que ver con la generación de árboles, normalmente aleatorios, para los individuos. Existen tres principales métodos de inicialización aleatoria de árboles:

- *Full* → Árboles de profundidad máxima fija:
  - Desde la raíz toma aleatoriamente funciones no terminales hasta llegar a la profundidad máxima
  - Una vez alcanzada la profundidad máxima sólo toma terminales

- *Grow* → Árboles de, como máximo, una profundidad dada:
  - Desde la raíz toma aleatoriamente funciones terminales y no terminales hasta llegar a la profundidad máxima
  - Una vez alcanzada la profundidad máxima sólo toma terminales

- *Ramped half-and-half* → Mezcla de *full* y *grow*:
  - Una mitad de la población se genera con *full*
  - La otra mitad de la población se genera con *grow*
  - En ambos métodos se toma como profundidad máxima un valor aleatorio en un rango dado

2.3.3 Selección de progenitores

En GP se suele emplear selección de progenitores por torneo. Este método consiste en, cada vez que es necesario elegir un progenitor, extraer aleatoriamente un conjunto de individuos de un tamaño dado y tomar el mejor de todos ellos.

Frente a otros métodos de selección como el proporcional al *fitness*, con este método se evita que los individuos con *fitness* relativamente muy superior al del resto de la población se propaguen demasiado para la siguiente generación.

2.3.4 Operadores genéticos

Al igual que en otros algoritmos evolutivos, en GP hay que definir un conjunto de operadores genéticos que serán los encargados de generar descendencia para la siguiente población de individuos a partir de los mejores progenitores seleccionados de la población actual.

Típicamente GP emplea los siguientes operadores genéticos:

- Cruce de subárboles (ver Figura 2):
  - Se seleccionan dos progenitores
  - Se escogen sendos nodos aleatorios en los progenitores
  - Se intercambian los subárboles de los progenitores a la altura de los nodos escogidos tomando como individuos nuevos uno o los dos nuevos árboles obtenidos

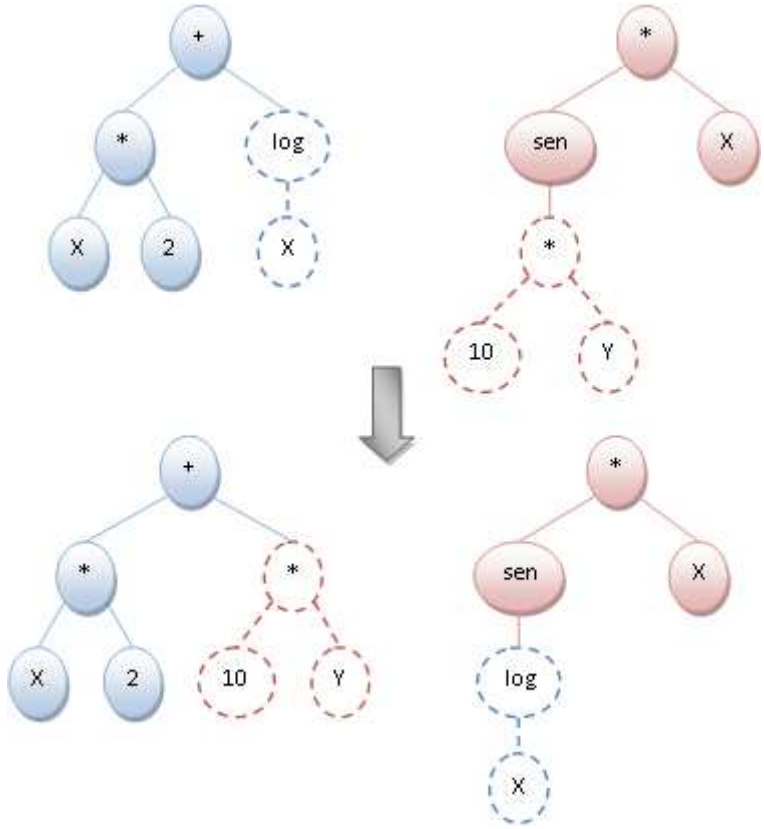


Figura 2. Ejemplo de cruce de árboles. Se intercambian los subárboles de los progenitores a partir de un nodo elegido aleatoriamente en cada uno

- Mutación de subárboles (ver Figura 3):
  - Se selecciona un progenitor
  - Se escoge un nodo aleatorio del progenitor
  - El subárbol a la altura del nodo escogido se sustituye por un subárbol nuevo aleatorio

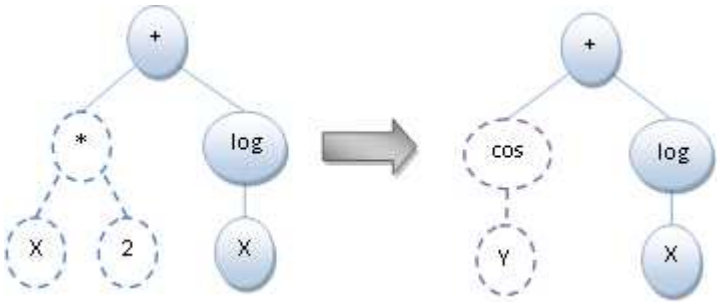


Figura 3. Ejemplo de mutación de árboles. Se genera un nuevo subárbol a partir de un nodo escogido aleatoriamente del progenitor

- Mutación en un punto (ver Figura 4):
  - Se selecciona un progenitor
  - Se selecciona aleatoriamente un nodo
  - La función del nodo se sustituye por otra aleatoria de igual número de hijos

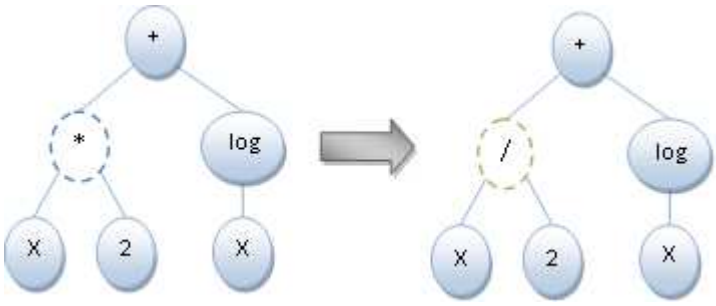


Figura 4. Ejemplo de mutación en un punto. Se selecciona un nodo aleatorio en el progenitor y se sustituye la función por otra con igual número de hijos

Las probabilidades para elegir nodos no tienen por qué ser necesariamente uniformes. De hecho es frecuente tomar con mayor probabilidad nodos no terminales que terminales para equiparar la mayor cantidad de éstos frente a aquéllos, especialmente en árboles grandes con funciones de elevado número de hijos.

Otro tipo de operador empleado es el de reproducción, que simplemente selecciona un progenitor de la población y lo copia para la siguiente generación.

2.3.5 Problemas conocidos

Posiblemente el problema más conocido que puede afectar a la población de individuos a lo largo de una evolución en GP es el denominado en inglés efecto de *bloat*. Típicamente observable en muchos problemas de GP, aunque no es exclusivo de este algoritmo evolutivo, este efecto tiene que ver con el sucesivo incremento en el tamaño de los árboles de los individuos, sin que éste vaya acompañado de una modificación positiva en el *fitness* de tales individuos.

Una de las soluciones más simples a este problema, propuesta por el propio Koza (6), es la de limitar la profundidad o el tamaño de los árboles generados para evitar drásticamente la generación de árboles demasiado extensos.

Existen otras técnicas aplicables para evitar el efecto. Por ejemplo, el uso de operadores genéticos específicos como el ya visto de mutación en un punto, que añade variedad genética al árbol sin incrementar su profundidad o tamaño, o el *size fair mutation* y el *size fair crossover* (10) (11), en los cuales se pone especial cuidado en la elección de los nodos donde realizar el cruce o la mutación para que los árboles no crezcan desmesuradamente.

Otro conjunto de técnicas bien conocido son las representadas por el método *parsimony pressure* de Koza y sus variantes, algunas de ellas analizadas en (12). En este caso, el objetivo se centra en el cálculo del *fitness*, donde se aplican diversas penalizaciones a los individuos de mayor tamaño/profundidad.

2.3.6 Variantes más conocidas

Ciertos problemas pueden exigir restricciones específicas sobre el número o tipo de nodos hijo que un nodo padre puede tener. Ésta es la variante conocida como *strongly typed* GP (13). Un ejemplo de esta clase de problemas se ilustra en la Figura 5.

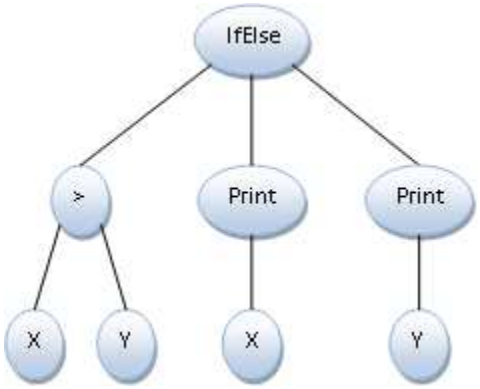


Figura 5. Ejemplo de árbol *strongly typed* GP para un programa en un tradicional lenguaje de alto nivel con expresiones lógicas, variables, comandos, etc.

En este caso, funciones como, por ejemplo, el operador de comparación “>”, requieren dos nodos hijos de tipo numérico, otras, como el controlador de flujo “IfElse”, necesitan un primer argumento de tipo lógico y dos argumentos de tipo comando, etc.

Un paso más allá está la variante conocida como evolución gramatical (14), en el que se emplean restricciones gramaticales sobre la sintaxis de los programas representados.

Una famosa variante a la representación mediante árboles es el GP lineal (15), en la cual se adopta una visión de los programas representados más próxima a la realidad de cómo un compilador de lenguaje máquina trabaja, derivando secuencias de instrucciones que se ejecutan linealmente y empleando registros para intercomunicar los resultados intermedios de las distintas operaciones.

Otra variante es el conocido como GP cartesiano (16) (17). En este caso los programas se representan mediante grafos, pero el genotipo de los individuos se codifica como un cromosoma lineal de números enteros. En dicho conjunto lineal de números enteros cada elemento del grafo toma un subconjunto para representar el tipo de operación y la posición con respecto al grafo.

3. Estado del arte en inteligencia artificial en videojuegos

3.1 Los primeros videojuegos

Los orígenes documentados (18) (19) de los videojuegos nos remontan hasta el año 1960, cuando Stephan Rusell, junto con otros estudiantes del *Massachusetts Institute of Technology*, creó *Spacewar!*, un videojuego para dos jugadores humanos en el que cada uno controlaba una nave espacial que luchaban entre sí. Con anterioridad, ya habían surgido otros rudimentarios videojuegos como el famoso *Tennis for two*, también para dos jugadores humanos, creado por William Higinbotham en el *Brookhaven National Laboratory*. Sin embargo, en este caso el videojuego no se ejecutaba sobre una computadora sino sobre un sistema analógico como es el osciloscopio.

En estos primeros videojuegos no existía ningún tipo de inteligencia artificial (IA). Tal vez fuera porque los juegos tradicionales siempre habían sido jugados entre personas o tal vez porque los recursos de los también primeros computadores de la época eran muy reducidos, todavía tendríamos que esperar un poco para ver los primeros videojuegos donde el jugador humano pudiera desafiar a la máquina.

La popularidad de los videojuegos se disparó unos pocos años después, cuando los ordenadores empezaron a ganar en potencia, y títulos como *Pong* (1972) de Atari, con dos sencillas líneas blancas a modo de raquetas de tenis hacían rebotar un punto blanco de lado a lado de la pantalla, o *Space Invaders* (1978), donde una nave tenía que defenderse de los "ataques" por oleadas de gran cantidad de alienígenas virtuales, dieron comienzo a una nueva forma de ocio digital desconocida hasta el momento.

En títulos como *Pong* ya se empezaron a ver una de las primeras formas o conceptos de IA en videojuegos, la IA "tramposa". Este sencillo modelo de IA se fundamenta y aprovecha en el conocimiento completo del estado del juego por parte de los NPC; esto es, dado que la posición y velocidad del personaje controlado por el humano son conocidas, el personaje controlado por la máquina en *Pong* puede ajustar su posición para devolver la pelota al personaje controlado por el humano. En función de la velocidad y precisión con que el NPC adecúe su posición el desafío y la emoción de la experiencia interactiva para el humano es mayor, lo cual es uno de los objetivos primordiales de la IA en videojuegos.

Con *Space Invaders* o el venidero *Pac Man* se introdujo otro concepto básico en la incipiente IA en videojuegos: hacer creer al usuario que la IA es más potente de lo que realmente es. Con comportamientos prefijados mediante sencillas reglas o patrones (e.g. los invasores de *Space Invaders* moviendo de izquierda a derecha manteniendo la formación, y los fantasmas de *Pac Man* persiguiendo al jugador con diferentes algoritmos para escoger la ruta) el usuario puede llegar a pensar que la máquina está siguiendo intrincadas estrategias o incluso anticipándose a su pensamiento.

Con este auge en popularidad dio comienzo la conocida como época dorada de los videojuegos. De ahí surgieron títulos míticos como *Asteroids, Pac Man o Donkey Kong*, y duró hasta 1983.

Y así los videojuegos siguieron creciendo en número y variedad. Los primeros juegos de carreras como *Pole Position* o *Night Driver*, hacia 1982, eran en realidad mapas lineales, pero se usaban sencillos trucos para dar sensación de ser pistas con curvas.

La IA de estos videojuegos de carreras se construía mediante combinación de patrones: seguimiento del circuito a través de puntos de navegación, detección/evasión de colisiones con el jugador, detección/evasión de salidas de pista, etc. La sensación de realismo y la credibilidad de la IA en estos juegos se debían fundamentalmente a la diferenciación de atributos entre los corredores y a la introducción de aleatoriedad. Incluso en juegos más modernos como *Super Mario Kart* (1992) se seguían manteniendo técnicas similares, haciendo fuerte uso de IA tramposa para, por ejemplo, mantener niveladas y emocionantes las carreras.

Donkey Kong inició el género de los juegos de plataformas, y el clásico *Mario Bros* (1983), uno de los videojuegos más famosos de la historia, lo elevó en popularidad.

En estos juegos surgió la necesidad por tener mayor constancia del estado del personaje controlado por el jugador. Aun siendo controlado por el jugador, el personaje debe comportarse razonablemente según el entorno, pudiendo saltar a una plataforma superior o aplastar a un enemigo dependiendo de dónde esté Mario. En juegos modernos este comportamiento suele modelarse mediante máquinas de estados finitos.

En torno a 1980 aparecieron los primeros juegos de estrategia con títulos como *Missile Command*.

En este caso el desafío para la IA radica en el desarrollo de estrategias de planificación y gestión eficiente de los recursos de los que tanto jugador como máquina disponen. Otro gran problema de estos juegos es la obtención de rutas o *path finding*, donde el algoritmo A* es habitualmente empleado todavía actualmente.

Tras la época dorada, o la posterior "generación *PlayStation*[2]" hasta llegar a nuestros días, el crecimiento en todos los aspectos de los videojuegos ha ido a la par que el exponencial crecimiento en la potencia del *hardware* de las plataformas (i.e. ordenadores, consolas y los modernos *smartphones*) para ejecutarlos, siendo cada vez más impresionante el acabado visual de los videojuegos contemporáneos y requiriendo de la IA mayores competencias de las que los creadores de los primeros videojuegos podrían haber imaginado.

De todo este periodo es difícil destacar sólo unos pocos títulos por la relevancia de los elementos innovadores de IA que introdujeron en el mundillo. Sin embargo, algunos son

[2] http://en.wikipedia.org/wiki/PlayStation_%28console%29

particularmente recurrentes tanto en artículos académicos de IA en videojuegos como en la propia industria de los mismos (20), y por ello los citamos a continuación.

Creatures[3] (1996) representa uno de los primeros juegos/simuladores de vida artificial del mercado. El juego permitía criar y enseñar a criaturas "de verdad" mediante el uso de modelos bioquímicos y técnicas de aprendizaje automático como redes neuronales.

Half-Life[4] (1998) representó un hito importante dentro del género de los *first person shooters* (FPS) entre otros motivos por ser el primero en incluir tácticas de escuadra creíbles en sus *bots*.

Black & White[5] (2001) es uno de los más citados avanzados ejemplos de aplicación de IA en videojuegos. El juego, dentro del género de los juegos de estrategia "modo dios" (i.e. control cenital de un mundo/ciudad/población/etc. enteros), consiste en el adiestramiento de criaturas o monstruos para que actúen posteriormente por su propia cuenta en base a lo que les hemos enseñado. La interacción del propio jugador en tiempo de juego sirve para entrenar redes neuronales o árboles de decisión unidos a arquitecturas de agentes de tipo BDI (*belief-desire-intention*).

F.E.A.R.[6] (2005) representa otro paso más en el género de los FPS, dotando a los *bots* de gran capacidad de reacción, uso a su favor de los elementos de entorno, comportamientos de equipo y otra serie de características muy interesantes no vistas antes.

Spore[7] (2008) es el último juego que comentaremos. En la línea de *Creatures* principalmente, aunque también de *Black & White, Spore* nos permite evolucionar una raza de criaturas desde la fase celular, pasando por la fase de criatura hasta la fase de civilización y carrera espacial. Utiliza modelos evolutivos para guiar la transformación de nuestra raza tanto en sus rasgos genéticos como en comportamiento.

3.2 Diversificación de los videojuegos modernos

Hoy en día existe una variedad tal de videojuegos que podemos hablar de géneros diferenciados. Inspirados por los títulos clásicos, cada género ha ido perfilándose a lo largo de la historia del videojuego, e incluso todavía hoy siguen apareciendo géneros nuevos. Uno de estos nuevos géneros es el caso de los videojuegos sociales, que, aprovechando las nuevas tecnologías de las redes sociales, ha conseguido llegar de manera masiva al público general, fomentando su uso entre los "amigos" de un usuario como parte del propio juego. De esta

[3] http://en.wikipedia.org/wiki/Creatures_%28artificial_life_program%29#Overview

[4] http://store.steampowered.com/app/70

[5] http://lionhead.com/black-white/

[6] http://en.wikipedia.org/wiki/F.E.A.R.

[7] http://www.spore.com/ftl

manera, gracias a la conocida red social *Facebook*[8], el videojuego social conocido como *FarmVille*[9] tiene actualmente más de 38'5 millones de usuarios que lo han probado (i.e. les "gusta") y más de 17'5 millones que lo juegan mensualmente según el propio *Facebook* en la página oficial para el juego[10]. También son relativamente recientes y novedosos los videojuegos de tipo *sandbox*, como el famoso *Minecraft*[11], que se caracterizan por no tener un objetivo predeterminado, siendo los propios jugadores los que determinan qué quieren hacer con las herramientas que se les proporciona.

En general las nuevas tecnologías, y en particular Internet, han provocado un importante giro en el devenir de los videojuegos, pasando de juegos centrados en las acciones de un único usuario a complejos mundos interactivos donde multitud de usuarios participan y ejercen su propia influencia. Géneros como los ya mencionados juegos sociales, los juegos *online* masivos (e.g.: *World of Warcraft*[12], *Eve Online*[13]) o los juegos de navegador (e.g.: *Ogame*[14], *Hattrick*[15]) debieran ser estudiados aparte, pues abren un interesante marco para no sólo desarrolladores de videojuegos, sino también para sociólogos y psicólogos, por las relaciones sociales entre jugadores que se producen en dichos géneros.

Por otra parte, las nuevas generaciones de consolas como *Nintendo Wii*[16] (2006) o XBOX 360 con el periférico *Kinect*[17] (2010), o las consolas portátiles táctiles como *Sony PSP*[18] (2005) o *Nintendo DS*[19] (2004), e incluso la irrupción de los videojuegos en *smartphones* como el primer *iPhone*[20] (2007) o en las *tablets* como el primer *iPad*[21] (2010) con pantallas táctiles, han dado una nueva vuelta de tuerca al diseño de videojuegos, pero no tanto por las técnicas de IA que se pueden aplicar o la temática de los mismos sino en cómo el usuario puede interactuar con ellos, lo cual está fuera de los objetivos del presente documento.

No hay una única clasificación de géneros de videojuegos (21) (22) y para cualquiera de ellas es posible encontrar algún videojuego que incorpore elementos de varios géneros, lo cual indica

[8] http://www.facebook.com/

[9] http://company.zynga.com/games/farmville/

[10] www.facebook.com/FarmVille

[11] http://www.minecraft.net/

[12] http://eu.battle.net/wow/es/

[13] http://www.eveonline.com/

[14] http://www.ogame.com.es/

[15] http://www.hattrick.org/

[16] http://wii.com/

[17] http://www.xbox.com/es-es/kinect

[18] http://es.playstation.com/psp/

[19] http://www.nintendo.com/ds/

[20] http://www.apple.com/es/iphone/

[21] http://www.apple.com/es/ipad/

que no hay claramente una mejor que otra. Así las cosas, tomaremos la clasificación en (21) como referencia en este trabajo.

- Juegos de acción (e.g.: *Left4Dead*[22], *Mount & Blade*[23])
  - El jugador humano controla un personaje, en primera o tercera persona (según la posición de la cámara), en un entorno virtual, normalmente corriendo y matando lo que tiene por delante (los FPS)
  - La IA controla a los enemigos y, en juegos donde el jugador forma parte de un equipo, a parte o la totalidad del equipo

- Juegos de rol (e.g.: *Knights of the old Republic*[24], *The Elder Scrolls V: Skyrim*[25])
  - El jugador humano juega el papel de un personaje de entre varios posibles. A medida que va explorando el mundo virtual va ampliando las habilidades iniciales de los personajes
  - Hay juegos de rol individuales (*role playing games* o RPG) y juegos rol multijugador masivos (*massive multilplayer role playing game* o MMORPG) donde miles de personas juegan simultáneamente y pueden interactuar entre sí
  - La IA controla los enemigos pero también los compañeros y los personajes de apoyo. Estos últimos pueden llevar a cabo acciones propias al margen del jugador humano dando la sensación de un mundo dinámico siempre vivo

- Juegos de aventuras (e.g.: *Monkey Island*[26], *Day of the Tentacle*[27])
  - Respecto a otros géneros se resta importancia a la acción de combate y se le da a la resolución de puzles, la historia principal y las tramas de los personajes
  - El jugador debe resolver los puzles e interactuar con los personajes mientras avanza en la historia, cuyo curso puede verse alterado por sus decisiones
  - La IA se usa para construir de manera realística personajes guiados por objetivos, que son con los que el jugador interactúa.
  - La IA también puede tomar el papel de directora de la historia, ajustándose de manera dinámica a las acciones del jugador. No obstante, es frecuente que los juegos suelan acabar forzando a seguir un determinado curso narrativo lineal

- Juegos de estrategia / Simuladores (e.g.: *Starcraft*[28], *Civilization*[29])
  - El jugador humano controla muchas unidades (bélicas o de otro tipo) y las enfrenta a otros rivales desde un punto de vista de "dios". A menudo el jugador

[22] http://www.l4d.com/blog/

[23] http://www.taleworlds.com/

[24] http://www.bioware.com/games/legacy

[25] http://www.elderscrolls.com/skyrim/

[26] http://www.lucasarts.com/games/monkeyisland/

[27] http://es.wikipedia.org/wiki/Day_of_the_Tentacle

[28] http://us.blizzard.com/es-mx/games/sc/

[29] http://www.civilization.com/

también tiene que gestionar recursos, planificar acciones y organizar planes de ataque y defensa (como es el caso de los famosos juegos *real time strategy* o RTS)
        - La IA controla el comportamiento individual de las unidades del jugador acatando sus órdenes o gestionando su psicología y, como oponente, debe realizar las mismas tareas que el jugador con su unidades propias

- Juegos de deportes de equipo (e.g.: *Pro Evolution Soccer*[30], *Blood Bowl*[31])
    - El jugador desempeña un doble papel como entrenador y como jugador en un determinado deporte
    - La IA es similar a los juegos de estrategia aunque también puede incluir el rol del comentarista, que narra jugada por jugada lo que acontece en el juego

- Juegos de deportes individuales (e.g.: *Need for Speed*[32], *Wipeout*[33])
    - Simulan deportes individuales permitiendo el control en primera o tercera persona
    - La IA es similar a la de los juegos de acción aunque también puede haber comentaristas

- Juegos híbridos (e.g.: *Dungeon Keeper*[34], *Battlezone*[35], *Space Rangers*[36])
    - Son todos aquellos juegos que abarcan varios estilos: Un juego de estrategia que permite "saltar al cuerpo" de una unidad y controlarlo como en un juego de acción (e.g.: *Dungeon Keeper*), juegos de acción que incluyen gestión estratégica de recursos (e.g.: *Battlezone*), juegos de rol que incorporan escenarios de acción, estrategia y aventura (e.g.: *Space Rangers*), etc.

3.3 Inteligencia artificial de los NPC

Los NPC han sido tradicionalmente de manera natural el foco principal de atención para los diseñadores de IA en los videojuegos. (21) propone una relación (ver Figura 6) entre cada género con los típicos roles que debe desempeñar la IA de los NPC en los videojuegos típicos de ese género, así como la relación entre estos roles y los problemas de IA clásica con los que tienen que ver.

[30] http://www.konami-pes2013.com/

[31] http://www.bloodbowl-game.com/

[32] http://www.needforspeed.com/

[33] http://www.wipeouthd.com/

[34] http://en.wikipedia.org/wiki/Dungeon_Keeper

[35] http://en.wikipedia.org/wiki/Battlezone_%281998_video_game%29

[36] http://en.wikipedia.org/wiki/Space_Rangers_2:_Dominators

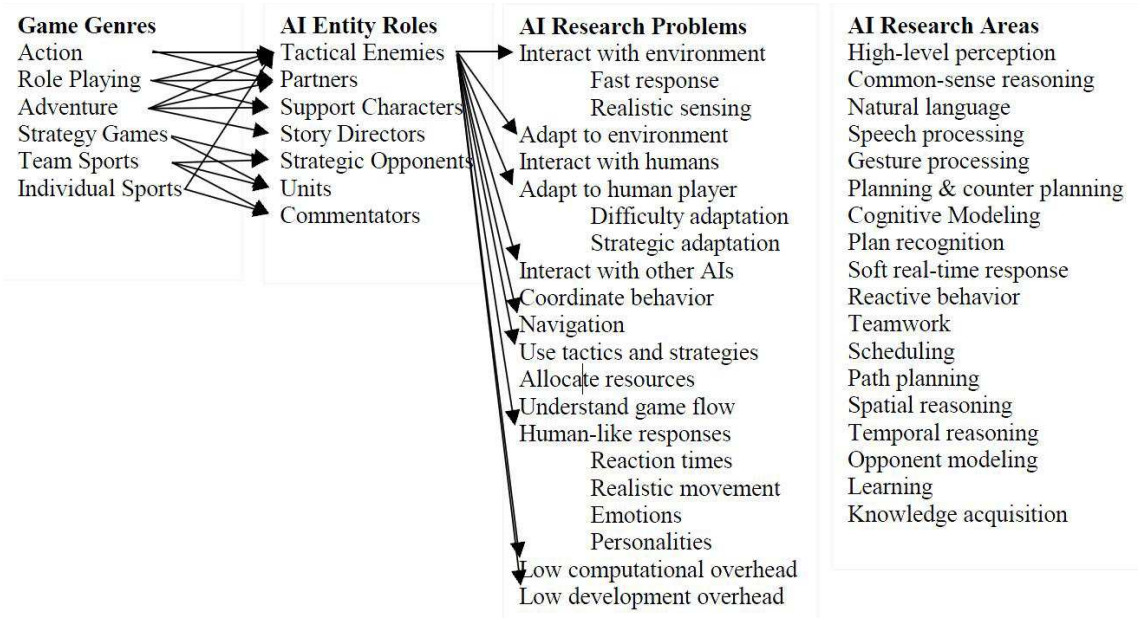


Figura 6. Relaciones existentes entre géneros de videojuegos en cuanto a roles comunes desempeñados por la IA en los mismos, vinculación de los roles con problemas de IA y áreas relevantes involucradas de investigación en IA (21)

A continuación se enumeran y describen los roles identificados:

- Enemigos tácticos
  - En los juegos más antiguos los NPC se limitaban a correr directamente hacia el jugador. Más tarde se introdujo la modelización de los agentes mediante máquinas de estados finitas
  - Juegos más recientes incorporan planificación de rutas y tácticas diversas para parecer más humanos
  - El diseño de estos enemigos supone resolver problemas diversos de IA e integrar las soluciones. Los enemigos deben ser autónomos en la persecución de sus propios objetivos, interactuar y adaptarse a entornos dinámicos complejos mediante comportamientos reactivos y planificaciones a medio/largo plazo, y razonando con "sentido común"
  - Un problema frecuente en estos NPC es la necesidad del uso de IA que hace ciertas trampas para adecuar la dificultad al nivel del jugador (e.g.: mejor armamento, visión a través de muros) ya que esto, una vez sabido por el jugador, resta credibilidad al comportamiento de los agentes virtuales

- Compañeros
  - Estos personajes enfatizan la interacción con el jugador humano. La interacción debe ser robusto y no suponer demasiado esfuerzo por lo que, en gran cantidad de juegos, se reduce a una lista cerrada de comandos que puede emitir el jugador. En un futuro tal vez podrían tener cabida la interpretación y generación de lenguaje natural o el reconocimiento de gestos para una mayor inmersión del jugador humano en el videojuego

- La IA debe coordinar su comportamiento, entender el trabajo en equipo y tener un modelo de los objetivos del humano para poder adaptarse a ellos

- Personajes de apoyo
  - Suelen ser los personajes con la IA menos sofisticada. En gran parte de los casos funcionan con un conjunto de respuestas prefijadas aunque pueden llegar a tener cierta autonomía y algunos objetivos sencillos
  - También pueden ser usados para poblar los juegos y rellenar el tiempo que el jugador humano se ausenta. En este papel deberían ser capaces de interactuar, adaptarse al entorno y a otros personajes humanos o de apoyo para comportarse de manera similar a un humano y, en un futuro, llegar a comunicarse mediante lenguaje natural

- Oponentes estratégicos
  - El uso de ciertos trucos o trampas es frecuente en estos personajes para ofrecer un adecuado nivel de desafío al jugador. Aun con tal ventaja estos oponentes son generalmente predecibles en su comportamiento, por lo que el jugador puede vencerlos fácilmente una vez encuentra sus puntos débiles. Éste es un problema complejo dado que, en algunos juegos, por ejemplo de deportes, es frecuente encontrar jugadores humanos con alto conocimiento del dominio
  - La IA debe constituir una estrategia de alto nivel y encargarse de dos tareas: asignar recursos y enviar órdenes a unidades. La estrategia supone la toma de decisiones de gestión y planificación, razonamiento de "sentido común" y el reconocimiento de los planes del jugador para la elaboración de contra-tácticas adecuadas
  - Dado el desafío que supone para el jugador humano el control de un elevado número de unidades, la IA debe procurar aplicar limitaciones similares a las que tiene el humano para aparentar realismo

- Unidades
  - Las unidades individuales reciben órdenes de alto nivel por el jugador que deben ser llevadas a cabo de manera autónoma. Normalmente se modelan mediante máquinas de estados finitos extendidas con rutinas especiales para la planificación y seguimiento de rutas, aunque también es deseable cierto comportamiento semi-automático emergente fruto del razonamiento de sentido común y la coordinación con otras unidades
  - Dada la gran cantidad de unidades que puede haber las limitaciones en cuanto a carga de memoria y procesador deben ser muy estrictas, aunque las cada vez mejores prestaciones de las actuales plataformas de videojuegos poco a poco van acabando con estas restricciones

- Comentaristas
  - La IA se dedica a observar las acciones del jugador así como de ella misma estableciendo patrones y generando lenguaje natural adecuado para cada situación

3.4 NPC como agentes

Hasta ahora se ha introducido sin más la noción de NPC como elemento inherente a un videojuego. Sin embargo, algunos autores (23) van más allá, sometiendo a análisis diversos NPC de videojuegos para debatir si pueden ser considerados agentes autónomos en el sentido clásico del área de sistemas multiagente, y así poder estudiar su IA bajo las metodologías de dicho área.

Una de las definiciones que utilizan toman como base de partida la toman de (24): "Un *agente autónomo* es un sistema situado dentro de un entorno del cual forma parte, capaz de percibir dicho entorno y actuar sobre él siguiendo su propia agenda y esperando que, con el tiempo, sus acciones tengan reflejo en lo que percibirá en un futuro".

Así enumeran las características que consideran han de estar presentes en un agente autónomo para poder ser considerado como tal:

- Ser reactivo (i.e. poder llevar a cabo acciones en respuesta a los cambios que aparecen en su entorno)
- Ser autónomo. Lo cual lo evalúan mediante los siguientes aspectos:
  - Existencia de supervisión por agentes de más alto nivel
  - Nivel de visibilidad de la comunicación con otros agentes
  - Complejidad en la toma de decisiones
  - Variedad del repertorio de acciones
  - Capacidad de aprendizaje (i.e. obtener información de circunstancias donde, de primeras, no sea posible aplicar una acción adecuada)
  - Consciencia de situación (i.e. estar informado del estado del entorno)
- Tener continuidad temporal (i.e. ser capaz de percibir el entorno y actuar sobre él para afectar a lo que percibirá en el futuro)
- Estar orientado a objetivos (i.e. que las acciones aplicadas persigan alguna meta)

La Figura 7 muestra el resultado de su trabajo para un conjunto de videojuegos seleccionado. La conclusión es que, si bien no todos los aspectos de un agente autónomo se reflejan en un NPC de videojuego, una gran cantidad de ellos sí que lo hacen, variando dependiendo del videojuego, lo que permite analizar la IA presente en esos NPC con los parámetros y metodologías propias de los sistemas multiagente.

		Aspects of autonomy							
	Reactivity	Supervision by higher level agents	Visible inter-agent commu-nication	Complexity of decision making	Variety of action repertoire	Learning	Situatedness	Temporally continuous	Goal-oriented
World of Warcraft	Low	None	Low	Low	Low	No	Yes	Low	Yes
Half Life 2 (Single player mode)	High	None	High	Low	High	No	Yes	Low	Yes
Supreme Commander	Low	High	Low	Low	High	No	Yes	High	Yes
Warcraft 3	Low	High	Low	Low	High	No	Yes	High	Yes
Black & White 2	Low	Low	Low	High	High	Yes	Yes	High	Yes
Tomb Raider: Anniversary	Low	None	High	Low	Low	No	Yes	Low	Yes
Unreal Tournament 2004 (bots)	High	None	High	High	High	No	Yes	High	Yes

Figura 7. Evaluación de los NPC de distintos juegos en base a sus características como agentes (23)

3.5 Técnicas y metodologías

Con la llegada a la madurez de los videojuegos se fueron asentando casi como un estándar una serie de técnicas o algoritmos para resolver los problemas más frecuentes que la IA de un videojuego debe afrontar. De entre ellos destacan principalmente el algoritmo de búsqueda A* y las máquinas de estados finitos (FSM o *Finite State Machines*), ambos ampliamente usados, y circunstancialmente optimizados al caso, por los desarrolladores en gran cantidad de videojuegos comerciales (25).

De forma genérica, el problema de la búsqueda consiste en encontrar una solución en un espacio de búsqueda abstracto. Aplicado a videojuegos su uso más habitual es para encontrar rutas en un mapa. Frente a otros enfoques tradicionales más costosos computacionalmente como la búsqueda en anchura o la búsqueda en profundidad, A* obtiene mejores rendimientos mediante el uso de heurísticas que guían la búsqueda hacia el objetivo.

En un contexto tradicional las FSM son modelos matemáticos que sirven para describir los distintos estados entre los que puede transitar un sistema. En videojuegos se consideran las FSM como IA ya que permiten representar entidades (e.g. mapas, agentes, etc.) del juego como un algo compuesto de varios estados, con transiciones entre ellos y disparadores que fuerzan que ocurran tales transiciones. De esta manera se puede hacer, por ejemplo, que un guardia en estado de patrulla cambie su comportamiento a un estado de búsqueda al oír un sonido realizado por el personaje del jugador, lo cual se entiende como “inteligencia artificial” del guardia.

Sin embargo, más allá de estos algoritmos, el impacto obtenido por la IA clásica en videojuegos comerciales es relativamente reducido (26) (27) debido a varios motivos:

- Limitación, cada vez menor pero aún existente, de los recursos computacionales de las plataformas de videojuegos → La calidad de la representación gráfica suele primar sobre la potencia de la IA
- Indeterminismo en las soluciones encontradas por los métodos clásicos de IA → Implica mayor complejidad en el proceso de *debug* y supone cierto riesgo ante la posibilidad de que se den comportamientos "estúpidos" en la IA (e.g. un *bot* de cara a la pared tratando de moverse hacia ella)
- Falta de tiempo asignado para la IA durante el ciclo de desarrollo *software* o ausencia de la misma hasta avanzadas fases del proyecto
- Incapacidad de los métodos de IA de videojuegos académica para superar los métodos ya asentados o afrontar en modo alguno los problemas de escalabilidad presentes en videojuegos comerciales
- Falta de comprensión del ámbito de la IA como se verá en 3.3

Con el objetivo de acercar la IA clásica al desarrollo de videojuegos comerciales, en (22) se presenta una taxonomía de videojuegos en base al tipo de interacción que los jugadores establecen con el motor del juego y a las características del mundo virtual representado, y se mapean dichas características con conceptos del área de teoría de juegos y sistemas multiagente así como posibles técnicas de IA clásica aplicables en cada situación. De esta forma se da un notable paso hacia la obtención de una metodología que determine la mejor técnica de IA aplicable según las necesidades de un videojuego. La Figura 8 ilustra dicha taxonomía.

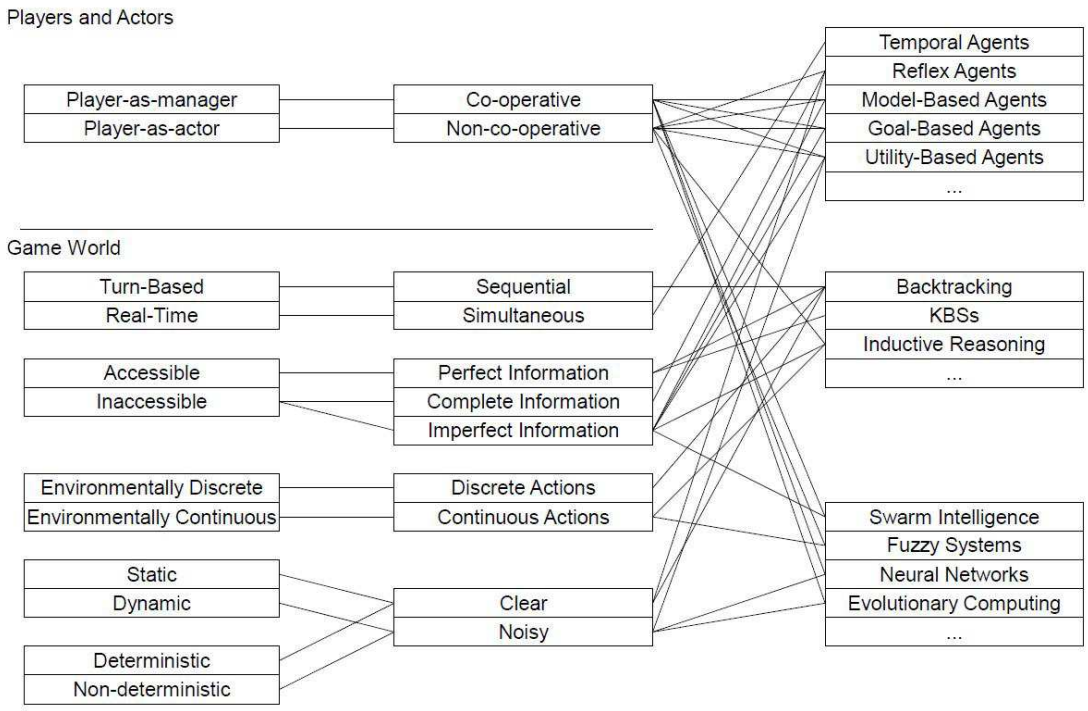


Figura 8. Taxonomía de videojuegos según la interacción del jugador con el entorno y las características del mundo del juego con conceptos de teoría de juegos y técnicas de IA clásica aplicables (22)

A continuación se definen los conceptos principales empleados en la taxonomía para caracterizar un videojuego:

- Según los modos de interacción de los jugadores:
  - Jugador gestor VS Jugador actor
    - Jugador contra el entorno: En el mundo del juego no hay actores y el jugador se enfrenta directamente al entorno desde una perspectiva de "dios". Al no haber rivales generalmente tampoco hay inteligencia
    - Jugador actor: Hay un único actor que el jugador controla y a través del cual recibe información del mundo
    - Jugador gestor: Hay más actores y el jugador se encarga de manejarlos a todos para conseguir los objetivos del juego

- Según el mundo del juego:
  - Basado en turnos VS Tiempo real
    - Basado en turnos: Existe una secuencia de pasos en el juego
    - Semi basado en turnos: Si además se permite pausar el juego para tomar decisiones o planificar acciones
    - Tiempo real: El resto de casos
  - Accesible VS Inaccesible
    - Accesible: La información del mundo está disponible para todos los actores
    - Inaccesible: Existen limitaciones a la información que conocen los actores sobre el mundo (e.g.: la "niebla de guerra" oculta las unidades enemigas cuando no hay ninguna unidad amiga presente en la zona)
  - Entorno discreto VS Entorno continuo
    - Discreto: El rango de acciones que puede tomar un actor en un momento dado es finito
    - Continuo: El rango de acciones es potencialmente infinito (e.g.: el actor se puede mover en cualquier dirección y no se limita a las direcciones cardinales)
  - Estático VS Dinámico
    - Estático: El mundo no se altera mientras los jugadores deciden qué hacer
    - Dinámico: Durante la toma de decisión el mundo puede verse alterado
    - Semi dinámico: El mundo no se altera pero sí pueden llevarse a cabo mediciones de tiempo, rendimiento, etc.
  - Determinista VS No determinista
    - Determinista: El siguiente estado se deduce explícitamente del estado actual y las acciones llevadas a cabo por los actores
    - No determinista: El mundo cambia al margen de las acciones de los actores o existe cierto grado de incertidumbre en el resultado de sus acciones

Y los conceptos de teoría de juegos y sistemas multiagente relacionados con las anteriores características de un videojuego son los siguientes:

- Según los modos de interacción de los jugadores:
  - Cooperativo VS No cooperativo
    - Cooperativo: Los actores y jugadores se agrupan en alianzas
    - No cooperativo: No se permiten alianzas y cada actor o jugador procura obtener el mayor beneficio individual
    - Híbrido: El juego es cooperativo pero los actores y jugadores tienen la opción de no aliarse y llegar a conseguir mayor beneficio yendo en solitario

- Según el mundo del juego:
  - Discreto VS continuo
    - Discreto: El número de participantes, acciones, turnos, etc. es limitado, por lo que es teóricamente posible trazar una matriz de evaluación que abarque todas las posibles situaciones futuras
    - Continuo: Actores y jugadores pueden entrar y salir del juego, modificar sus estrategias, etc. De este manera las circunstancias registradas son sólo un subconjunto de todas las posibles en el mundo del juego
  - Simultáneo vs secuencial
    - Simultáneo: Alguno o todos los jugadores realizan sus acciones a la vez
    - Secuencial: Los jugadores realizan sus acciones en secuencia de una en una
  - Visibilidad de la información
    - Información perfecta: Todos los participantes tienen acceso al estado actual del juego, las estrategias de otros rivales y lo ocurrido en el pasado
    - Información imperfecta: Al menos uno de los actores o jugadores tiene únicamente acceso a parte de la información
    - Información completa: La información es perfecta a excepción de las acciones pasadas, que pueden no ser conocidas por algún actor o jugador
  - Ruidoso VS limpio
    - Ruidoso: Existe una significativa cantidad de información para que los actores y jugadores tomen su decisión pero no toda ella es completamente correcta o adecuada
    - Limpio: La información no contiene errores

Como se ha dicho, el impacto en videojuegos comerciales de las técnicas de IA clásica es reducido, pero a nivel académico son bastantes las muestras que pueden encontrarse del trabajo de distintos investigadores en el área.

Aparte de las ya introducidas FSM y la búsqueda A*, algunas de las técnicas empleadas en videojuegos para la IA de los NPC se presentan a continuación:

- Redes neuronales artificiales (ANN o *artificial neural networks*)
  - Son modelos matemáticos constituidos por neuronas, que son en este contexto pequeñas unidades de procesamiento de información con una o varias entradas

y una función de transformación de dichas entradas para obtener la salida, las cuales se distribuyen por capas y permiten conectarse unas con otras
    - Aunque hay variedad de modelos y finalidades con que puede usarse una ANN normalmente éstas se entrenan proporcionando un conjunto de patrones a la entrada junto con la salida esperada. En caso de no obtener la salida esperada, se ajustan los pesos de las conexiones de la red progresivamente hasta tener un modelo adecuado

- Razonamiento basado en casos (CBR o *case based reasoning*)
    - Es una técnica que consiste en elaborar una base de datos de casos observados de manera que el sistema sea capaz de, dada una situación concreta que se le presente, tomar la decisión más adecuada en base a la información conocida de los casos pasados registrados más similares al presentado

- Sistemas de reglas
    - Estos sistemas se conforman transformando el conocimiento dado sobre un problema en un conjunto de reglas cada una con sus correspondientes condiciones necesarias para la ejecución y las acciones a lanzar. En conjunción con un motor de inferencia, cada vez que el sistema es consultado, las reglas aplicables compiten entre sí y se ejecutan finalmente las acciones que correspondan

- Lógica difusa
    - Es un modelo lógico que extiende el tradicional modelo *booleano* de verdadero/falso. Permite categorizar las variables con distintos grados de pertenencia y emplear esta información junto con un motor de inferencia de manera similar a un sistema de reglas

- Aprendizaje por refuerzo
    - Los algoritmos basados en este paradigma entrenan sistemas recompensando o reforzando aquellas secuencias de acciones que lleven a la consecución de algún objetivo de manera que, una vez entrenados, y dada una situación que se les presente, estos sistemas sean capaces de escoger con mayor probabilidad aquellas acciones que esperen que les vaya a proporcionar la mayor cantidad de recompensa posible del entorno

- Algoritmos evolutivos
    - Una visión global de estos algoritmos se trató en el capítulo 2
    - Neuroevolución → Es una técnica híbrida en la que se emplean redes neuronales que, en lugar de ser entrenadas de manera tradicional en base a una estimación de error, se evolucionan en poblaciones de redes o neuronas utilizando operadores genéticos propios de los algoritmos evolutivos
    - Evolución multiobjetivo → En realidad este tipo de evolución no es exactamente un algoritmo como tal sino una variante de otros algoritmos evolutivos

adecuado cuando el *fitness* para el problema tiene diversas facetas que deben ser optimizadas a la vez sin que necesariamente primen unas sobre otras

3.6 Panorama actual

Aparte de puntuales apariciones en congresos de áreas de IA clásica o sistemas multiagente, varios congresos anuales se celebran ya donde la IA en videojuegos tiene un papel principal, de los cuales caben destacar:

- *IEEE Conference on Computational Intelligence and Games* (IEEE-CIG)[37]
- *European event on Bio-inspired Algorithms in Games* (EvoGAMES)[38]
- *Foundations of Digital Games* (FDG)[39]
- *Conference on Artificial Intelligence and Interactive Digital Entertainment* (AIIDE)[40]
- *Game AI Developers*[41] (si bien ésta se orienta más a la comunidad comercial del videojuego que a la académica)

En CIG tiene lugar la competición conocida como *2K BotPrize Contest* (28), donde los desarrolladores envían *bots* para competir en una plataforma de videojuego FPS con otros *bots* y jugadores humanos, teniendo estos últimos la tarea adicional de juzgar como *bots* o como humanos al resto de jugadores. De esta forma el objetivo del campeonato es determinar si, con los avances actuales, es posible que un *bot* generado mediante técnicas de IA es capaz de hacerse pasar por un humano por su forma de jugar, capacidad que sin duda beneficiaría tremendamente a los videojuegos comerciales: mayor diversión (i.e.: es más divertido enfrentarse a un humano que a una máquina), mayor desafío (i.e.: la IA de la máquina, al contrario que un humano, suele ser predecible y por tanto fácilmente de derrotar una vez se conocen los patrones por los que se rige), etc.

Los trabajos que pueden encontrarse en estos congresos presentan interesantes avances en diversos géneros, como es el caso de los FPS, donde diversos autores han conseguido entrenar *bots* para que jueguen de manera similar a humanos. (29) emplea un modelo de ANN (en concreto SOM o *self organizing map*) que entrena tomando como entrada datos capturados de jugadores humanos desde dentro del propio juego, esperando como salida la misma acción que los jugadores llevaron a cabo. En (30) un modelo de *neural gas* y otro de *clustering* son aplicados conjuntamente con éxito para, dado un conjunto de observaciones de la posición de jugadores humanos, entrenar *bots* para un FPS que, primero, determinen automáticamente puntos de navegación adecuados para un mapa y, segundo, recorran dicho mapa empleando las rutas más habituales para los humanos. (31) implementa la IA de un *bot* mediante un sistema experto de reglas con múltiples objetivos y conocimientos introducidos manualmente

[37] http://www.ieee-cig.org/

[38] http://evostar.dei.uc.pt/2012/call-for-contributions/evoapplications/evogames/

[39] http://www.foundationsofdigitalgames.org/

[40] http://www.aiide.org/

[41] http://aigamedev.com/

sobre diversas tácticas y aspectos del juego. El *bot* mantiene una representación interna del mundo y es capaz de anticiparse a otros *bots* asumiendo que éstos siempre se comportan de manera similar.

Otros autores han buscado mejorar la IA básica de los *bots* proporcionados por el juego restando énfasis a que el comportamiento del *bot* se asemeje al de un humano. (32) consigue evolucionar un conjunto de reglas mediante GA para que el *bot* sea capaz de moverse para esquivar ataques detectando la procedencia de los mismos proyectando rayos detectores de obstáculos en varias direcciones. En (33) se evolucionan con un GA los umbrales de los disparadores de transición de la FSM de la IA básica de un *bot* para conseguir equipos de *bots* más habilidosos que los proporcionados por el juego. De manera similar, (34) emplea GA sobre los umbrales de la FSM para obtener mejores comportamientos y prueba, aunque sin éxito, a utilizar posteriormente GP sobre las reglas de transición resultantes. En (35) también se emplea un GA para mejorar la IA básica proporcionada por la plataforma. En este caso se evolucionan reglas cuyas acciones son comportamientos básicos creados a mano. Pese a lograr patrones de comportamiento interesantes la IA creada no logra superar a la de la plataforma. Los autores de (36) y (37) utilizan GP en un juego 2D de mecánica similar a un FPS para obtener árboles de comportamiento para equipos de *bots* capaces de encontrar y acabar con un enemigo en diferentes condiciones de dificultad. (38) ilustra el uso y necesidad del enfoque multiobjetivo para obtener mediante neuroevolución comportamientos adecuados para un juego 2D similar a un FPS con diversos escenarios separados de objetivos muy distintos diseñados específicamente a mano.

No siempre los autores optan por complejas plataformas similares a los entornos de juegos comerciales para demostrar las capacidades de sus planteamientos, dado que ello complica mucho la labor a los investigadores que sólo quieren centrarse en el desarrollo de la IA. (39) utiliza CBR en un sencillo juego 2D para entrenar *bots* capaces de alcanzar la meta esquivando obstáculos mediante la observación de las posiciones relativas de los elementos en el entorno. Utilizando un entorno de juego depredador-presa en (40) (41) emplean el algoritmo de neuroevolución ESP (o *enforced subpopulations*) para GP y una variedad de operadores de mutación para añadir y borrar dinámicamente modos de comportamiento para lograr coevolucionar a la vez controladores ANN para agentes depredador y agentes presa en un escenario multiobjetivo. Los ya presentados (36) (37) (38) aplican sus ideas sobre juegos similares en mecánica a un FPS pero sin la complejidad de un FPS en 3D real.

En otros géneros también se han obtenido avances similares. (42) aplica GP para evolucionar árboles de comportamiento en un RTS y obtener *bots* mejores que la IA básica del juego en tareas específicas. En (43) se emplea neuroevolución multiobjetivo para entrenar *bots* que imiten el estilo de conducción un jugador humano en base a ciertos criterios cuantificables en un juego de carreras de coches. En (44) se usa la técnica de aprendizaje por refuerzo conocida como *Q-learning* para mejorar la IA de partida en un aspecto muy puntual del juego, los adelantamientos.

Obtener un patrón de comportamiento adecuado para un NPC en juegos donde no hay un único objetivo no es una tarea sencilla. Algunos autores hacen explícitos de manera manual los

distintos escenarios donde se desenvolverá el *bot*. Aparte de los ejemplos ya vistos (31) (33) (35) (38), (45) aplica GP en un controlador de un juego estilo "comecocos" generando un patrón de comportamiento por escenario, y empleando dichos patrones como nodos terminales en una nueva evolución para obtener el patrón de comportamiento definitivo. Otros únicamente entrenan a sus *bots* en una única capacidad de las múltiples que un *bot* completo debe tener: En (30) aprenden a navegar por el mapa mientras que en (32) esquivan disparos enemigos. Finalmente, hay autores que han alcanzado cierto éxito empleando técnicas para descubrir automáticamente todos los modos u objetivos del juego sin tener que explicitarlos previamente a los *bots* como ya se ha visto (40) (41).

En paralelo a la labor más pragmática y directa de los investigadores en los artículos previamente estudiados aplicando técnicas de IA clásica o sistemas multiagente en sus *bots*, otros investigadores han incidido más en el desarrollo de modelos teóricos complejos sobre los cuales sustentar una mayor hipotética mayor complejidad y robustez de IA. (46) presenta una arquitectura cognitiva jerárquica en la que los estímulos se procesan de abajo hacia arriba, agrupándose en estructuras de cada vez mayor nivel, para luego generar acciones de manera inversa, de arriba hacia abajo, traduciendo objetivos y comportamientos de alto nivel en acciones sencillas de bajo nivel. El *bot* creado con dicha arquitectura ganó la edición de 2010 del campeonato *2K BotPrice*[42]. (47) describe una arquitectura de conocimiento basada en la atribución de niveles de confianza a los hechos conocidos y desarrollo de modelos de jugador según los casos observados. Con ello es capaz de generar patrones de comportamiento adecuados según la situación y oponente.

En definitiva, puede resumirse el panorama actual de la IA de NPC en videojuegos de la siguiente forma:

- IA orientada a usuario
  - IA adaptativa
  - IA "humana"
  - IA interactiva

- IA no orientada a usuario
  - IA eficaz

Esto es, pese a que siempre el usuario final de un videojuego es un jugador humano, podemos diseñar la IA teniendo en cuenta de manera explícita a dicho jugador, pero también podemos diseñarla sin considerarlo directamente. Dentro del primer caso tenemos tres posibilidades: IA adaptativa, IA "humana" e IA interactiva.

La IA que no se queda en un mero entrenamiento en la fase de desarrollo del juego (i.e.: *offline*) sino que dinámicamente (i.e.: *online*) se permite que vaya cambiando, es la que se conoce como IA adaptativa. Autores como (47) presentan potentes modelos teóricos cuyos

[42] http://aigamedev.com/open/article/conscious-bot/

principios soportarían esta clase de IA. En (38) pese a que los autores querían encontrar la mejor solución, lo hacían simulando el comportamiento del jugador humano mediante sencillas reglas prefijadas y un determinado nivel de hándicap representando la gradual mejora de las habilidades del jugador y forzar una adaptación de la IA al jugador. En el juego depredador-presa de (40) (41) se llegaron a coevolucionar comportamientos muy complejos entre los dos tipos de agentes de manera dinámica y cada vez más sofisticados, sin ser necesario detener el proceso de aprendizaje o realizarlo de manera *offline*. Estas ideas llevadas con éxito a videojuegos reales favorecerían que los patrones de comportamiento obtenidos no sean predecibles para el usuario, característica que, como ya se ha dicho, es muy deseable para evitar que el usuario acabe aburrido.

Por IA "humana" se entiende todo aquel modelo de IA cuyo principal objetivo es asemejar el comportamiento de un humano a ojos de los jugadores humanos. Varios ejemplos han sido ya retratados (29) (30) (31) (43) (46) donde uno o varios aspectos de los jugadores humanos tratan de ser reflejados en los *bots*.

La IA interactiva, ampliamente extendida en diversas áreas incluida la de los videojuegos (48), es una categorización muy genérica de algoritmos en la que se emplea de manera totalmente directa el conocimiento del usuario humano, ya que éste forma parte esencial propio proceso de entrenamiento. Si bien no es el área más frecuente en videojuegos, innovadoras aportaciones como el trabajo en el videojuego NERO[43] (49) pueden tener mayor repercusión en un futuro. NERO pertenece a un género prácticamente inédito de videojuego en el que es el propio jugador humano el que invierte tiempo en entrenar a sus *bots* soldado para generar los patrones de comportamiento. En este caso no es exactamente el objetivo de la IA parecerse al jugador humano pero sí el de aprender los comportamientos que éste quiera inculcar en los *bots*.

Por otro lado, están los modelos de IA no orientados al usuario, que es lo que podríamos denominar como IA "eficaz". En este caso lo que se busca es conseguir modelos robustos y óptimos para la tarea a desarrollar dentro del videojuego, sin necesariamente estar restringidas las capacidades de los personajes controlados por la IA a las condiciones reales que un personaje controlado por el humano tendría en una situación similar. Es decir, podemos (y quizá debamos) emplear IA tramposa. Estos modelos son los más parecidos a la IA clásica, en la que, generalmente, no nos interesa restringir la información de entrada o las posibles acciones a realizar en pos de hallar la solución más adecuada, la cual es posible que ya sea de por sí bastante complicada de obtener.

Una gran cantidad de ejemplos se han analizado ya en esta categoría (32) (33) (34) (35) (36) (37) (39) (42) (45) en los que los autores buscaban obtener *bots* que se desenvolvieran lo mejor posible o al menos mejor que la IA básica de partida del juego o plataforma. Los problemas de este planteamiento son visibles en trabajos como (44), donde se optimizaba un controlador para un juego de coches obteniendo un patrón de comportamiento con tal nivel

[43] http://www.nerogame.org/

de perfección que sería muy poco creíble como humano en una partida con jugadores humanos.

3.7 Nuevas áreas de aplicación

Todo el panorama analizado hasta el momento en este trabajo recae en algún aspecto de la IA de los NPC, área que si bien puede ser vista como objetivo natural de la IA en los videojuegos, no es la única donde recaen los esfuerzos de los investigadores. Más aún, tal y como se afirma en un reciente trabajo, los objetivos actuales y futuros de la IA en videojuegos debieran reorientarse hacia otras prominentes áreas de interés dentro de los videojuegos donde todavía hay mucho campo abierto para obtener susceptibles mejoras (27).

A continuación se resumen las áreas de interés recalcadas por el autor y los apuntes dados sobre ellas:

- Modelado de la experiencia del jugador (*Player Experience Modeling* o *PEM*)
  - Alentado por la diversificación y escalado en popularidad de los géneros de videojuegos modernos (como se vio en 3.2), es difícil hablar en la actualidad de un perfil único de jugador de videojuegos
  - Mediante PEM se busca predecir algunos aspectos de la experiencia que tendrá un jugador en general, un jugador de un tipo determinado o un jugador en concreto en una situación particular de un juego
  - Existen tres aproximaciones principales según cómo se obtenga la información del jugador:
    - Subjetivo → La expresan los jugadores
    - Objetivo → Se obtiene por vías alternativas de respuesta del jugador
    - Basado en el juego → Se deduce del comportamiento y contexto del jugador en la interacción con el juego
  - Según el caso, los modelos pueden obtenerse mediante algoritmos de clasificación, regresión o aprendizaje de preferencias, técnicas de reconocimiento biométrico (e.g.: gestos, emociones, etc.) o por análisis de las respuestas del jugador al sistema

- Generación de contenido procedural (*Procedural Content Generation* o PCG)
  - Tiene que ver con los algoritmos capaces de crear automáticamente contenido para el juego (e.g.: mapas, historias, misiones, eventos, música, etc.) excluyendo todo lo relacionado con el comportamiento de los NPC
  - Los principales objetivos de esta área son:
    - Aligerar el esfuerzo de la tarea de diseño durante la fase de desarrollo del videojuego
    - Ajustarse automáticamente a las necesidades y preferencias del usuario
    - Poder superar, teóricamente, las capacidades creativas de los diseñadores humanos y ofrecer soluciones inéditas

- Minería de datos en juegos a escala masiva
  - Muchos aspectos de un videojuego, tanto en la fase de desarrollo como una vez puestos en el mercado, requieren análisis masivos de datos para proporcionar *feedback* interesante a los creadores del juego
  - El crecimiento en este área está intrínsecamente relacionado con el desarrollo y estandarización de métricas de juego apropiadas para la obtención masiva de datos de los jugadores (especialmente en juegos *online*) y los algoritmos de inferencia de patrones o perfiles de estilo de juego

4. Entorno de simulación

4.1 La plataforma

Con el objetivo de disponer de un entorno suficientemente complejo como para que pudieran llegar a emerger comportamientos interesantes entre los diversos agentes, se buscaron y valoraron diversas plataformas principalmente del mundo académico de los videojuegos (*BWAPI*[44], *Stargus*[45], *ORTS*[46], *TORCS*[47]) y comercial (*Unity*[48], *Ogre*[49]), pero también en el de los sistemas multiagente, seleccionando finalmente como mejores candidatas a *Pogamut*[50] (50) y *Multiagent Simulation Toolkit* (*MASON*)[51] (51).

4.1.1 *Pogamut*

Pogamut es una potente plataforma para agentes virtuales en entornos 3D bien conocida en la literatura académica reciente en IA de videojuegos del género de los FPS (32) (35) (46), incluyendo el propio *2K BotPrize Contest* (28) que usa dicha plataforma para llevar a cabo el campeonato. Más que una plataforma *software*, *Pogamut* es, en realidad, una capa de abstracción *Java* sobre otra plataforma, conocida como *GameBots*[52] (52), que se sitúa como interfaz entre varias versiones del motor de videojuegos comercial de la saga de juegos FPS *Unreal Tournament*[53] y cualquier *bot* que quiera conectarse a una partida de dicho motor intercomunicándose con él a través de simples mensajes de texto, sin necesidad de conocer la sintaxis propia del lenguaje de *script* propio del juego. Adicionalmente, *Pogamut* tiene un *plugin* para el entorno de programación Java *Netbeans*[54] mediante el cual se puede observar información adicional de la partida (e.g. estado y localización de los agentes, variables de *debug*, etc.) que se esté ejecutando en el servidor de juego. La Figura 9 muestra esquemáticamente todas las piezas que conforman la arquitectura de un sistema que emplea *Pogamut*.

[44] http://code.google.com/p/bwapi/

[45] http://stargus.sourceforge.net/

[46] https://skatgame.net/mburo/orts/

[47] http://torcs.sourceforge.net/

[48] http://unity3d.com/

[49] http://www.ogre3d.org/

[50] http://pogamut.cuni.cz/main/tiki-index.php

[51] http://cs.gmu.edu/~eclab/projects/mason/

[52] http://gamebots.sourceforge.net/

[53] http://planetunreal.gamespy.com/

[54] http://netbeans.org/

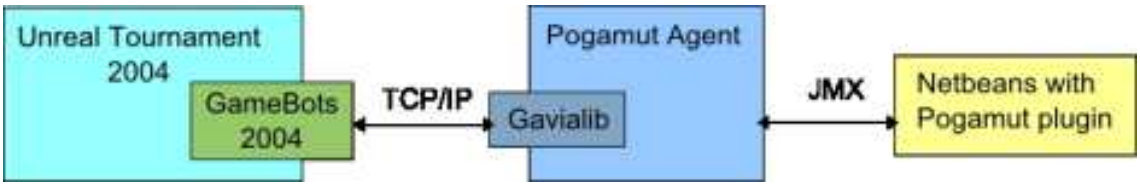


Figura 9. Esquema de arquitectura de la plataforma *Pogamut* con el motor comercial de videojuegos *UT2004*

Gracias a *Pogamut* un desarrollador académico de IA tiene al alcance un auténtico videojuego comercial como entorno para sus experimentos. Empleando código *Java* el desarrollador puede programar la IA de sus *bots* y ejecutarla de dos maneras complementarias: por eventos (i.e. ejecución asíncrona) y programada (i.e. ejecución síncrona). En el primer caso, y de manera muy similar a quien desarrolla una página *Web* HTML, *Pogamut* permite escribir *listeners* (i.e. fragmentos de código que se ejecutan cuando se cumplen determinadas condiciones) en respuesta a toda una tremenda variedad de posibles eventos que puede recibir el agente virtual implementado como percepciones del entorno (e.g.: nuevo agente a la vista, sonido escuchado, etc.). En el modo programado *Pogamut* nos permite ejecutar las líneas que código que queramos con la periodicidad temporal que nos interese. Combinando ambos planteamientos podemos conseguir *bots* con complejas pautas de comportamientos de planificación a medio/largo plazo y combinarlos con rápidas acciones reactivas a situaciones que surgen en tiempo real.

Pogamut no sólo proporciona información al *bot* en cuanto al sistema de visión tridimensional virtual (lo cual incluye datos sobre los agentes e ítems visualizados, muros, puertas, etc.) sino que también provee un complejo elenco de algoritmos esenciales de IA en un FPS como una implementación del A* para la planificación de rutas entre puntos de navegación del mapa o uno para apuntar y disparar automáticamente a otros agentes o posiciones fijas a través de un esquema configurable de preferencias de arma.

Como última característica interesante, dado que hay un servidor con el motor del juego alojando una partida a la que puede conectarse cualquier tipo de jugador, el propio desarrollador puede conectarse a la partida con un personaje y observar en primera persona el comportamiento de los *bots.*

El principal inconveniente que se encontró en *Pogamut* fue la complejidad para adaptarlo a otros modos de juego distintos del conocido como *Deathmatch*[55], que es el modo más frecuente y básico entre los juegos *online* de diversos géneros entre los que se encuentran los RTS o los FPS. Pese a que un agente creado con *Pogamut* es capaz de conectarse a la partida con un rol especial de controlador, distinto del rol común de jugador con el que *bots* o humanos se conectan para jugar, y tener ciertas capacidades sobre el juego como eliminar automáticamente a todos los personajes presentes o cambiar el mapa, se encontraron demasiadas dificultades para, por ejemplo, devolver la partida a un estado inicial dado cada vez que un *bot* pasara por un determinado punto establecido de antemano por el controlador

[55] http://en.wikipedia.org/wiki/Deathmatch

o separar en una partida por equipos a los *bots* en dos facciones de tamaño no balanceado, lo cual hizo inviable seguir empleando la plataforma sin tener que dedicar una gran cantidad de tiempo para dominarla por completo y/o modificar el código necesario para nuestros fines, dado que éste se distribuye mediante licencias de código abierto. Teniendo en cuenta que los objetivos de este TFM no se limitaban simplemente a implementar un modelo de IA para un FPS convencional en modalidad de *Deathmatch,* y pese a que ciertamente se consiguieron ciertos avances iniciales con la plataforma (e.g. en Figura 10 una turba armada de *bots* clónicos zombis nos persigue por el mapa en cuanto visualizaban nuestra posición o escuchaban algún ruido generado por nosotros), se descartó seguir usando *Pogamut* y buscar otra opción no tan vistosa pero sí más flexible.

Figura 10. Agentes Zombis programados para localizar visualmente o por sonido a un jugador (humano) y perseguirle

4.1.2 *MASON*

Implementado también en *Java*, en este caso nos encontramos con una plataforma propia del área de los sistemas multiagente. *MASON* permite modelar agentes como entes independientes capaces de realizar acciones sobre el entorno, normalmente una estructura en 2D, aunque también tiene soporte para entornos 3D, permitiéndonos configurar la secuenciación de éstas en base a distintos parámetros: instante de comienzo, periodicidad, instante de finalización, etc.

MASON separa la capa de definición del modelo y la capa de representación visual del mismo para permitir distintas visualizaciones o incluso ninguna en absoluto. De este modo, es posible definir un modelo con sus agentes, características de entorno y las relaciones entre ambos, y emplear algún tipo de fichero de *log* para obtener trazas de lo que está ocurriendo en la simulación ignorando por completo la representación visual.

En el caso de la modalidad de entorno 2D, que es la única probada en este TFM, la representación del entorno como capas superpuestas de matrices 2D de objetos permite a *MASON* ofrecer una sencilla aunque eficaz visualización del entorno, pudiendo seleccionar las capas que se quieren observar y realizar un seguimiento de toda la información de los agentes u objetos presentes en cualquiera de las capas.

La Figura 11 muestra un esquema de la jerarquía de clases de *MASON* (51). Como se ha dicho, esta jerarquía se divide en dos bloques o capas principales que, a su vez, comprenden una serie de elementos principales que, en conjunto, conforman la totalidad un modelo *MASON*:

- *SimState*: Modelo
  - *Mersene Twister*: Genera los números pseudoaleatorios que necesitemos
  - *Schedule*: Planifica los eventos que queremos que se produzcan en la simulación, principalmente las acciones de los agentes, aunque puede ser cualquier cosa que implemente la interfaz *Steppable*
  - *Field*: Contienen las entidades relevantes de nuestro modelo, asignándoles una localización si es necesario

- *GUIState*: Visualizador del modelo
  - *Console*: Consola gráfica con los controles básicos de la simulación (e.g.: parar, pausar, reiniciar, etc.)
  - *Displays*/*Portrayals*: Asocian representaciones gráficas a los campos del modelo
  - *Inspector*: Permiten mostrar y/o modificar durante la simulación las propiedades de las entidades del modelo

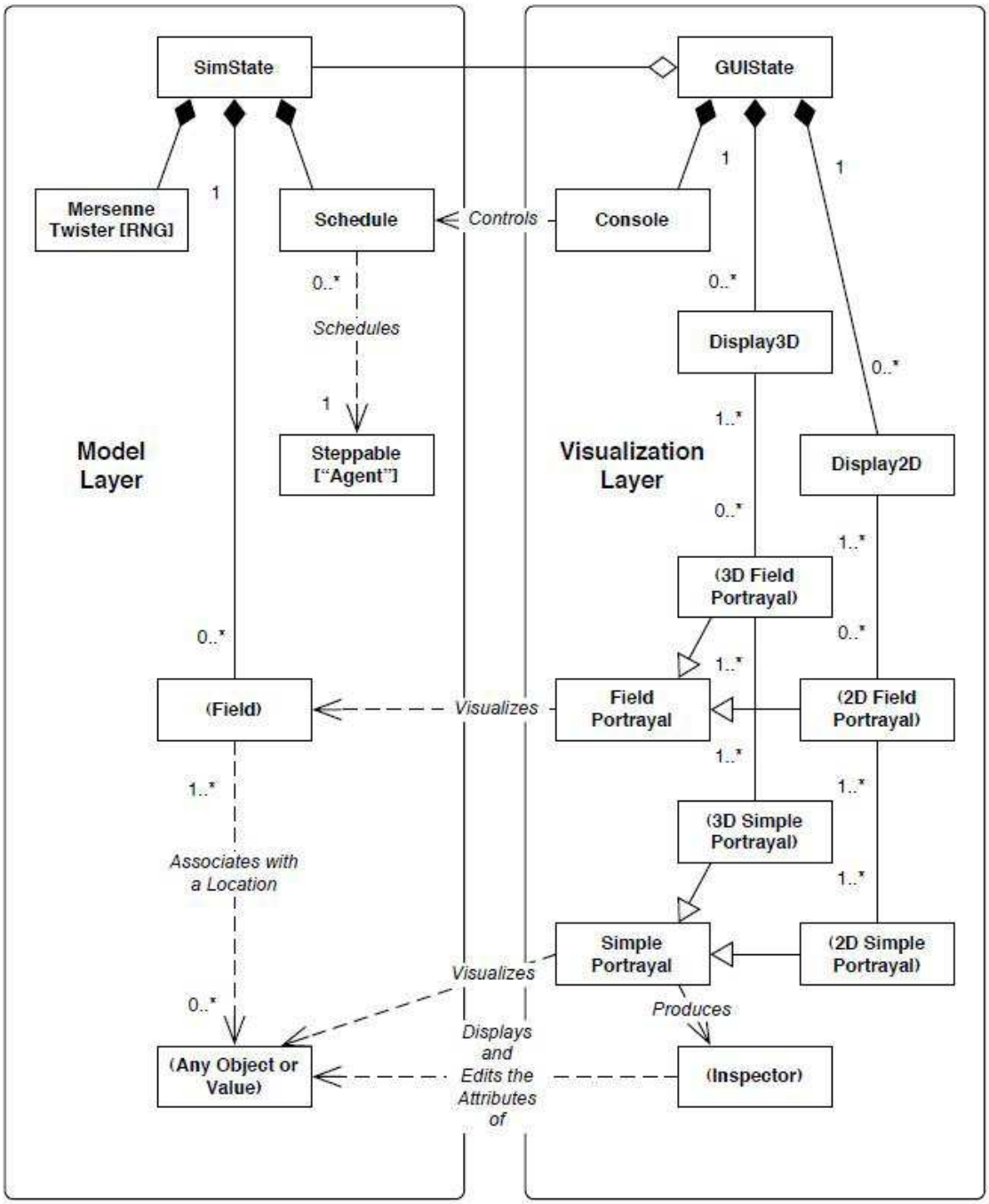


Figura 11. Esquema de la jerarquía de clases de *MASON*

Dado que tenemos total control sobre la información que albergan internamente los agentes y la que pueden llegar a compartir con otros o conocer del entorno en sí mismo, *MASON* es lo suficientemente flexible como para poder ajustarse a las necesidades de la simulación que nos interese en base a los distintos parámetros que nos podemos encontrar en la literatura de los sistemas multiagente (53) como son la complejidad de los agentes, desde agentes sencillos con unas pocas líneas de código a paradigmas más complejos de agente como los que se adecúan al modelo *Belief–desire–intention* (BDI), el nivel de intercomunicación deseado entre agentes, desde agentes que actúan de manera independiente y descoordinada a equipos donde la

comunicación entre unos y otros es crítica para la consecución de sus objetivos, o las funciones a desempeñar por los agentes, desde sistemas homogéneos donde los agentes realizan las mismas tareas y son potencialmente idénticos a sistemas heterogéneos donde cada agente tiene asignadas distintas tareas o roles.

La principal ventaja, y a la postre, la razón última de la preferencia de uso de *MASON* sobre *Pogamut* fue la flexibilidad proporcionada por el primero de cara a la construcción de un entorno totalmente ajustado a las necesidades del presente TFM. Sin embargo, como se explicará detenidamente en detalle en la sección 4.3, las ventajas que proporcionaba *Pogamut* en cuanto a características implementadas a nivel del motor de juego, hubo que recrearlas para el entorno en *MASON*, desde la creación de mapas, pasando por la definición de las interacciones elementales entre agentes y entorno (e.g. un agente no puede ver a través de una pared), la navegación entre puntos del mapa, etc.

4.2 Cambio de paradigma: del depredador-presa al superviviente-zombi

En este TFM no se aborda el clásico problema del depredador-presa (54), ampliamente estudiado en el ámbito de los sistemas multiagente para estudiar algoritmos (entre otros) de cooperación. En su lugar, introducimos en este TFM un nuevo problema o paradigma, el del superviviente-zombi, el cual pensamos que tiene suficiente entidad y complejidad en sí mismo como para ser objeto de futuros estudios, ya sea en la misma línea de este TFM u otra distinta.

4.2.1 Agentes y roles

En la versión más básica, el entorno clásico depredador-presa consta de dos tipos de agentes con objetivos opuestos, depredadores y presas:

- Los depredadores deben cazar a las presas
- Las presas deben huir de los depredadores

Con objetivos tan sencillos y tan claramente contrapuestos, estos entornos dan mucho juego y hasta dentro del mundo académico del videojuego algunos investigadores han depositado allí sus esfuerzos aplicando sus ideas con el objetivo de ver su viabilidad y llevarlas en un futuro a videojuegos comerciales (40) (41).

Nuestra nueva propuesta también utiliza dos tipos principales de agentes, zombis y supervivientes en este caso:

- Los zombis, de manera similar a los depredadores, deben cazar a los supervivientes, sus presas
- Los supervivientes deben encontrar la salida del mapa:
  - Huyendo de los zombis, al igual que hacían las presas con sus depredadores
  - Atacando a los zombis, como si de un depredador se tratara

Es decir, en el zombi se mantiene el rol del depredador intacto como cazador presas, pero a la presa se le proporciona un objetivo de más alto nivel, encontrar la salida, que lo puede alcanzar comportándose como una presa sin más, o volviéndose contra los zombis invirtiendo en cierta manera los roles de depredadores y presas.

4.2.2 Ecosistema

En el entorno clásico depredador-presa es posible instanciar el problema de diversas maneras en función de la cantidad de agentes de cada tipo, condiciones que deben darse para que uno o varios depredadores se consideren que han cazado a una presa, tipo de comunicación posible entre los depredadores, etc. aunque ha sido típicamente estudiada la configuración que parte de cuatro depredadores y una única presa (53).

En otros casos, sobre todo en el ámbito de la simulación en ecología o biología, nos puede interesar diseñar un ecosistema, esto es, incorporar otros elementos de entorno como la comida de las presas, permitir que los individuos generen descendencia en función del alimento que tengan y hacer coexistir diversas especies de depredadores y presas, para poder estudiar la evolución de las distintas poblaciones a lo largo del tiempo.

En el caso de querer obtener algo equivalente a un ecosistema "realista" de supervivientes y zombis hay que tener varias cosas en cuenta, propias de los amplios y variados contextos filmográficos[56], literarios[57] y de los propios videojuegos[58], del mundo de los zombis, que normalmente nos sitúan en mundos de condiciones apocalípticas con el ser humano a punto de la extinción a mano de los zombis:

1. Los zombis son poco "inteligentes":
    - Su comportamiento es, principalmente de carácter reactivo, activándose ante la recepción de estímulos directos del entorno (e.g. un disparo, un olor o un movimiento pueden captar su atención y hacer que el zombi vaya hacia la fuente del estímulo)
    - No llevan a cabo tomas de decisión complejas (e.g. un zombi no escoge la presa más fácil, simplemente corre hacia el mayor estímulo próximo)
    - No son capaces de planificar acciones o tener objetivos a medio/largo plazo pues no tienen memoria (e.g. un zombi puede lanzarse a correr detrás de un humano y, si lo pierde de vista, parar al rato cuando haya olvidado el motivo original de ponerse a correr)
    - Carecen de estrategias de cooperación directas con otros zombis aunque pueden llegar a cooperar indirectamente (e.g.: varios zombis pueden aporrear y echar abajo una puerta que uno solo no podría si perciben algo detrás de ella, pero no porque sepan que cooperando conseguirán su objetivo sino porque sus

[56] http://en.wikipedia.org/wiki/List_of_zombie_films

[57] http://en.wikipedia.org/wiki/List_of_zombie_novels

[58] http://en.wikipedia.org/wiki/List_of_zombie_video_games

medios para alcanzarlo coinciden) y pueden aparecer ciertos patrones de comportamiento emergente (e.g.: un zombi puede detectar y ponerse a perseguir a un superviviente atrayendo con ello la atención de otros zombis próximos que no habían percibido a dicho superviviente para que se unan a la persecución)

2. Aunque el hipotético origen científico de los zombis no está del todo claro, sí que se considera que son antiguos humanos que se mantienen vivos pese a las heridas o daños que hayan podido sufrir en su transformación o vida posterior como zombis, por lo que:
   - Sus atributos físicos son aproximadamente similares a los de un humano superviviente no infectado. Sin embargo, al igual que una persona invidente suele desarrollar más otros sentidos como el tacto o el oído), igualmente un zombi puede tener más aguzados sus sentidos que un humano normal (e.g.: en ambientaciones donde los zombis son ciegos, éstos tienen más desarrollado el sentido del olfato y del oído para poder detectar humanos)
   - No sienten dolor. Lo cual puede hacerles llevar a cabo acciones que un humano, en principio, nunca sería capaz de hacer (e.g.: un zombi puede romper a puñetazos una puerta de madera aunque se destroce, literalmente, las manos y muñecas en ello, y, sin embargo, nunca podría atravesar una pared de hormigón por mucho que la golpeara)

3. Los supervivientes no pueden emplear acciones netamente ofensivas directas contra los zombis y olvidarse de huir o escapar, ya que, normalmente:
   - Los zombis superan a los supervivientes en número
   - La munición, armas y recursos con que cuentan los supervivientes son limitados y han de ingeniárselas para encontrar nuevas formas de defenderse

Si cualquiera de estos puntos no se cumple, la posibilidad de tener algo parecido a un ecosistema natural se rompe, dando lugar a entornos cuya sostenibilidad es poco creíble por no decir imposible, como es el caso de la conocida película *Soy Leyenda*[59], donde la presencia de zombis con capacidades físicas extraordinarias superiores a las de los humanos (lo cual les aproxima más a la cultura popular de los vampiros[60] que a la de los zombis) probablemente acabarían con estos últimos en muy poco tiempo de llegar a ocurrir alguna vez en la vida real.

4.3 El entorno *Left4Sim*

4.3.1 La inspiración: *Left4Dead*

Los videojuegos de la saga *Left4Dead*, el igualmente denominado *Left4Dead* y la continuación *Left4Dead 2*, comienzan su historia como la mayoría de juegos, películas o libros basados en

[59] http://www.imdb.com/title/tt0480249/

[60] http://es.wikipedia.org/wiki/Vampiro#Caracter.C3.ADsticas_y_atributos

historias de zombis: Por alguna razón la infección zombi se ha propagado a escala mundial y sólo unos pocos humanos, los por ello denominados supervivientes, sobreviven a duras penas como pueden en los restos que quedan de un mundo apocalíptico para no ser mordidos por los zombis y convertirse en unos más de ellos.

El juego, que pertenece al género de los FPS, sitúa en cada escenario a un equipo de 4 supervivientes en un refugio inicial. Dicho refugio inicial es seguro, pues sus reforzadas puertas no pueden ser atravesadas por los zombis, y allí disponen de armas de fuego (e.g.: ak47, escopetas, rifles de francotirador, etc.), armas de combate cuerpo a cuerpo (e.g.: hachas, espadas *catanas*, sartenes, etc.), munición y otros ítems del juego como botiquines, desfibriladores, dosis de adrenalina, granadas, etc.

El objetivo en cada escenario de los supervivientes es salir del refugio inicial y buscar el siguiente, que está en alguna ubicación determinada del mapa. Por lo general los mapas no son demasiado laberínticos, aunque sí suelen ofrecer distintas rutas que confluyen más adelante en el camino único final hacia el refugio de salida.

Los zombis están por todas partes en cuanto los supervivientes salen del refugio. Por lo general están en reposo hasta que detectan la presencia muy próxima de los supervivientes o escuchan disparos efectuados por estos, momento en el cual se abalanzan sobre los supervivientes, consumiendo una determinada cantidad (fijada por el nivel de dificultad configurado) de la vida total del superviviente atacado si consiguen aproximarse lo suficiente. Cada cierto tiempo, suena una particular alarma y una horda de zombis nueva se une a los actualmente presentes en el mapa para ir directamente a por los supervivientes.

Adicionalmente a estos zombis básicos, existen zombis infectados especiales que tienen ciertas características que los hacen más peligrosos para los supervivientes:

- *Charger*: Un robusto infectado que carga contra los supervivientes en línea recta arrastrando a todos los que coge por el camino para estamparlos con el suelo en cuanto choca con un obstáculo
- *Jockey, Smoker* y *Hunter* son otros infectados especiales cuyas habilidades, de manera similar al *Charger*, atrapan a un superviviente y no lo sueltan hasta que un compañero acude en su ayuda para liberarlo
- *Spitter*: Una espigada y escuálida infectada que escupe charcos de ácido sobre los supervivientes reduciendo rápidamente su vida
- *Boomer*: Un/a tremendamente obeso/a zombi que se acerca torpemente a los supervivientes para vomitarles encima. Si el vómito les alcanza, éstos quedarán cegados durante unos instantes mientras una nueva horda de zombis básicos aparecerá sintiendo especial predilección por atacar a los supervivientes afectados
- *Tank*: Una mastodóntica criatura que inflige cantidades de daño masivas a los supervivientes y requiere gran cantidad de fuego concentrado por parte de éstos para ser abatido
- *Witch*: Una aparentemente frágil mujer que nunca ataca a los supervivientes salvo que alguno se acerque demasiado o abra fuego contra ello. Cuando es molestada se

dirige velozmente hacia el superviviente que la incitó y le causa grandes cantidades de daño a no ser que se le dispare con gran intensidad

En principio los jugadores humanos controlan a los supervivientes, haciéndose cargo la IA del juego del control del resto de supervivientes no controlados por jugadores humanos (si se da el caso de no haber 4 jugadores humanos presentes) y de los infectados, tanto los normales como los especiales. Existen otros modos de juego *online* en el que los jugadores humanos también pueden controlar a los zombis especiales.

En estos modos de enfrentamiento *online* se asigna puntuación directamente proporcional a lo lejos que cada superviviente del equipo llegó en su recorrido hacia la salida, obteniendo una bonificación especial si llegó finalmente con vida a la salida.

La IA de los supervivientes controlados por el ordenador es muy sencilla: se limitan a acompañar a alguno de los supervivientes controlados por un humano, disparar al mismo objetivo que éstos o proporcionar fuego de cobertura si algún superviviente del equipo está siendo atacado, e ir a apoyar a los compañeros derribados.

En el caso de los zombis especiales también son apreciables ciertas pautas o *scripts* que emplean según el tipo de ataque que tengan (e.g. los *smokers* siempre esperan desde las alturas para atrapar a alguien con su larga lengua, los *boomers* esperan detrás de las puertas, etc.), careciendo, aparentemente, de estrategias de coordinación, más allá de que coincidan vivos en el mismo momento varios infectados especiales atacando al grupo de supervivientes.

Los zombis normales no presentan tampoco comportamientos demasiado complejos. Una vez alertados de la presencia de los supervivientes o alentados por el movimiento de la horda, los zombis corren hacia alguno de los supervivientes para atacarle, escogiendo como objetivo primeramente allá donde haya pota de *Boomer*, ya sea un personaje o incluso una localización, o allá donde se haya lanzado una granada, atraídos por el sonido y los indicadores luminosos de la misma.

Es posible encontrar publicados ciertos detalles de algunos aspectos técnicos del desarrollo del juego, incluidos varios temas relativos a la IA (e.g.: navegación, *spawn* de zombis, etc.) en la propia página oficial de sus creadores, *Valve*[61], aunque la principal y novedosa aportación de este videojuego en términos de IA es el sistema denominado *The Director*, cuyas características principales se pueden encontrar también en la Web[62]. Mediante este sistema se controla y mantiene la tensión a lo largo del juego, provocando nuevas hordas cuando los jugadores estén avanzando muy lento o haya pasado mucho tiempo desde el último episodio de acción y recompensando la cooperación de los jugadores (e.g.: botiquines usados en compañeros, eliminación de zombis a punto de atacar a un compañero, etc.) de distintas formas, distribuyendo por el mapa más o menos ítems de curación, armas y munición o

[61] http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf

[62] http://left4dead.wikia.com/wiki/The_Director

complicando algunas zonas potencialmente laberínticas de los mapas, añadiendo muros o caminos sin salida que en otras partidas no estaban.

Con todo esto, el juego consigue dos fundamentales objetivos: potenciar la (re)jugabilidad (i.e. ninguna partida es igual por lo que es más divertido jugar) y, algo más interesante para nosotros, que es que el juego fomenta explícitamente la cooperación. Un único ataque especial efectuado con éxito por varios de los infectados especiales requiere que otro compañero acuda en tu auxilio, y cuando tu vida está próxima a acabar eres derribado, no pudiéndote mover y sólo pudiendo emplear un arma básica hasta que otro compañero te levante del suelo. Como la partida se acaba cuando todos los supervivientes han sido derribados a la vez (ya que los supervivientes derribados no pueden levantar a otros supervivientes derribados) o han muerto, aunque es posible "resucitarlos" con el uso de los desfibriladores y/o llegando a puntos determinados del mapa donde todos los supervivientes muertos pueden volver al juego, la cooperación es primordial.

Así las cosas, por mucha precisión que se tenga adquirida en otros FPS, y dado que los ataques provenientes de los infectados especiales no siempre es sencillo evitarlos, un gran jugador de FPS "convencionales" que no coopere con sus compañeros es difícil que supere los escenarios. Y es por ello que nos parece este estilo de FPS más interesante para los objetivos de este máster: buscar comportamientos interesantes cooperativos entre agentes.

4.3.2 Visión global

Encontrando, como se ha dicho, inspiración en los videojuegos *Left4Dead*, la plataforma *Left4Sim* nació originalmente como proyecto de ejemplo de simulación de sistema multiagente para la asignatura de *Simulación y análisis de sistemas complejos* de este actual programa de máster.

Left4Sim como tal está programado en *Java* y es esencialmente un modelo *MASON.* En un proyecto como *Left4Sim* la visualización es necesaria para poder observar con total detalle el comportamiento de los agentes, por lo que se optó por implementar una capa de visualización con las herramientas que *MASON* proporciona. Con la capa de visualización activada *MASON* funciona algo más lento, por lo que se diseñó el código para que ésta fuera optativa y poder aligerar la ejecución de grandes cantidades de simulaciones.

El modelo en *Left4Sim* comprende dos tipos básicos de entidades: agentes y elementos de entorno. Existe otro tipo especial de elemento de entorno que son las casillas del mapa de calor o nivel de estímulo que perciben los zombis. Las entidades del modelo se disponen en un mapa plano rectangular comprendido por tres capas solapadas, cada una de ellas con un conjunto distinto de tipos de entidades que puede albergar:

- Capa de agentes:
  - Superviviente
  - Zombi

- Capa de entorno:

- Obstáculo:
  - Barro
  - Agua
  - Pared
- Localización especial:
  - Entrada
  - Salida
- Ítem:
  - Botiquín
  - Caja de munición

- Capa de estímulos:
  - Calor

En cada casilla de cada capa sólo puede haber una única entidad como máximo, aunque en una misma casilla pueden coincidir entidades de diversas capas. Aunque pudiera parecer más complicado que integrar todos los elementos en una única capa, este mecanismo facilita tremendamente la representación visual del entorno, como se puede observar en las imágenes de la Figura 12, Figura 13, Figura 14 y Figura 15 donde se muestra un ejemplo de visualización de una simulación en un instante determinado.

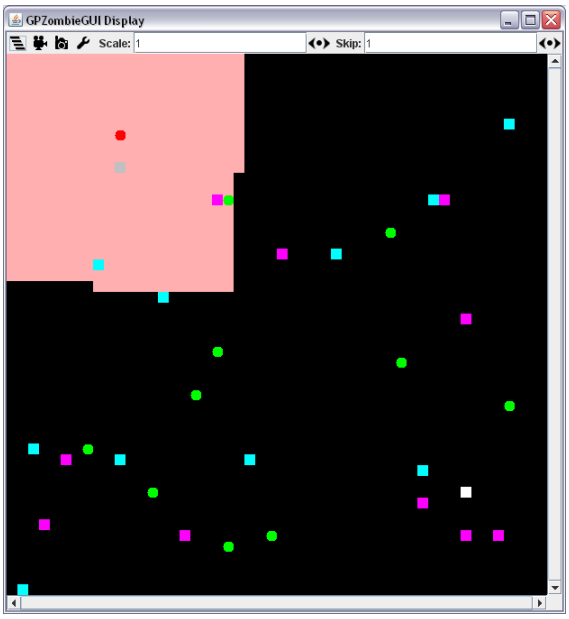


Figura 12. Mapa completo de un instante de una simulación *Left4Sim* (incluida en color salmón una capa virtual con todas las casillas conocidas por los supervivientes)

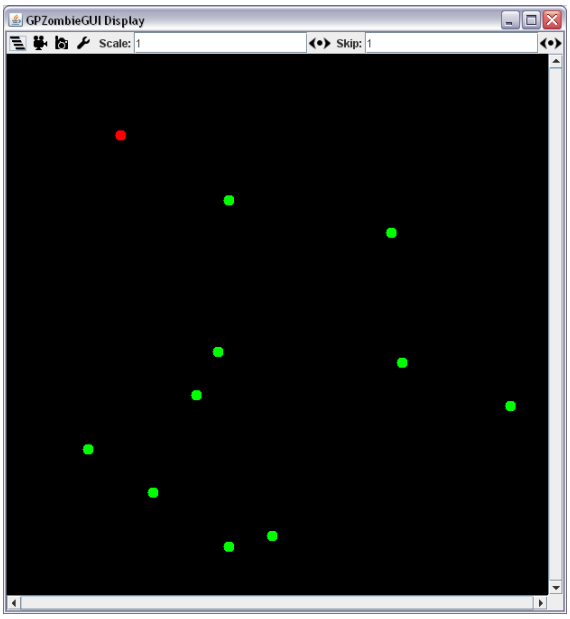


Figura 13. Capa de agentes (supervivientes en rojo y zombis en verde) de un instante de una simulación *Left4Sim*

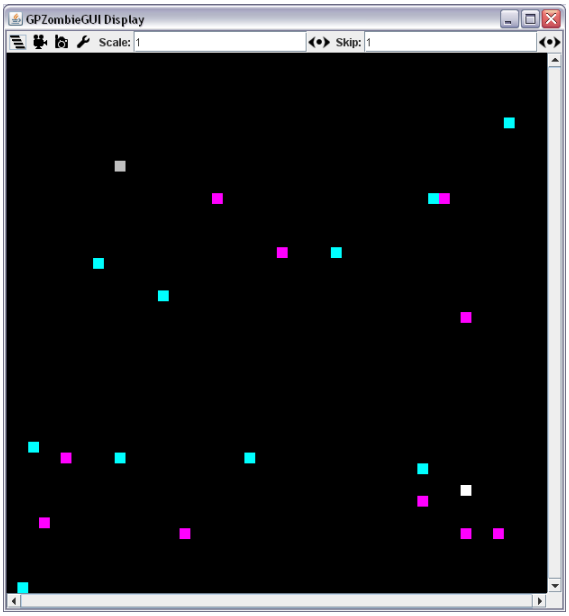


Figura 14. Capa de entorno (entrada en gris, salida en blanco, botiquines en azul y cajas de munición en violeta) de un instante de una simulación *Left4Sim*

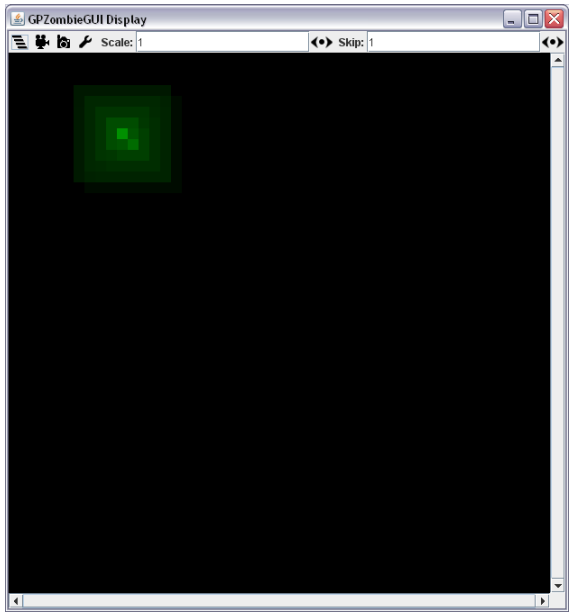


Figura 15. Capa de estímulos (cuanto más claro más intenso es el estímulo) de un instante de una simulación *Left4Sim*

Para atajar algunas dificultades técnicas encontradas en la implementación de la plataforma, principalmente en la implementación del sistema de visión artificial de los agentes que se verá más adelante, se optó por tomar la distancia de *Manhattan* como medida estándar de distancia en todo el proyecto.

Con todas las piezas sobre la mesa, el objetivo de la simulación es introducir un conjunto de agentes de tipo superviviente y agentes de tipo zombi en un determinado mapa para analizar el desempeño de su comportamiento en base a su interacción con el entorno y con el resto de agentes.

Los siguientes apartados profundizan en algunos de los aspectos más importantes del entorno.

4.3.3 Agentes

Todo agente en *Left4Sim* tiene las siguientes características básicas:

- Velocidad: Número máximo de casillas que puede avanzar en un turno
- Rango de visibilidad: Máxima distancia a la que es capaz de avistar objetos en el mapa
- Vida (inicial/actual): Cantidad de daño que puede recibir antes de morir
- Munición (inicial/actual): Cantidad de ataques que puede realizar
- Tipos de obstáculos que no puede atravesar

Cada subclase de agente que se incorpore al entorno debe asignar valor a las anteriores características en el momento de la instanciación de un nuevo objeto. Adicionalmente, estas clases herederas de agente deben implementar el método *step*, invocado por el *Schedule* del

modelo, en el que se define cómo interactúa el agente con su entorno en un único paso de la simulación dado el estado actual de la simulación.

A las características básicas de un agente genérico los zombis añaden las siguientes:

- Rango de ataque: Máxima distancia a la que pueden atacar a un superviviente
- Daño de ataque: Cantidad de puntos de vida que restan a un superviviente cada vez que le atacan

Su secuencia de turno procede de la siguiente manera:

1. Fase de movimiento
    a. Busca casillas a distancia de movimiento con igual o superior estímulo a la de la posición actual del zombi
    b. Elige aleatoriamente una casilla de destino con probabilidad proporcional al estímulo de la casilla en el conjunto de casillas al alcance
    c. Mueve hacia la casilla destino salvo:
        - Presencia de otro agente en el destino
        - Presencia de obstáculo no pasable para zombis en el destino
        - Presencia de obstáculo en la casilla actual que le detenga este turno

2. Fase de ataque
    a. Busca supervivientes en casillas a distancia de ataque del zombi
    b. Elige aleatoriamente un superviviente objetivo
    c. Ataca al superviviente objetivo reduciendo su vida conforme a su daño de ataque y, si ésta llega a 0 o menos, se elimina al superviviente del mapa y se le saca de la cola de eventos del *Schedule*

En el caso de los supervivientes, y dado que la interacción el entorno es más compleja, el listado de características y variables adicionales que los supervivientes requieren es mayor que para los zombis:

- Puntería (%): Probabilidad de acertar correctamente a un objetivo
- Arma:
    - Número de ataques que puede realizar en un turno
    - Precisión (%) con que el arma daña a un objetivo
    - Cantidad de daño infligido por ataque
    - Máxima distancia a la que pueden atacar

- Dirección actual de la mirada
- Ángulo de visibilidad: Ángulo máximo de la visión del superviviente (centrado en la dirección actual de la mirada)
- Estadísticas:
    - Número de zombis asesinados
    - Número de movimientos realizados
    - Mayor camino avanzado hacia la salida más próxima

        - …

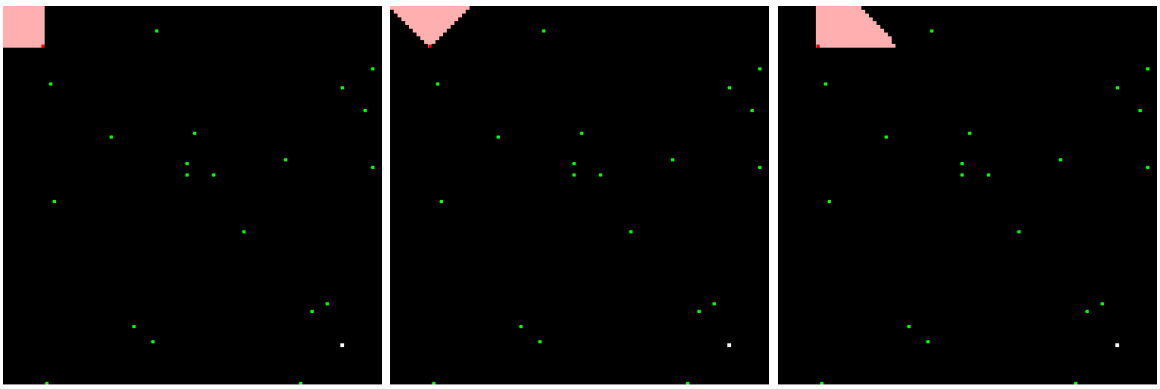
- Planificador de rutas:
    - Ruta/objetivo actual
    - Mapa propio de navegación en base a los puntos conocidos del mapa global

- Percepciones del entorno:
    - Ítems vistos en este momento
    - Obstáculos vistos en este momento
    - Agentes vistos en este momento

- Memoria:
    - Última posición conocida de ítems vistos en algún momento
    - Última posición conocida de obstáculos vistos en algún momento

Los supervivientes constan de un sistema de visión artificial que tiene en cuenta la dirección actual de la mirada, el ángulo y distancia máximos de visibilidad del agente para proporcionarle, cada vez que mira en una dirección, el conjunto de posiciones, ítems, obstáculos y agentes visibles desde su posición actual.

La Figura 16 y Figura 17 ilustran el sistema de visión artificial implementado en los supervivientes con ejemplos de un agente situado en la misma posición observando su entorno desde distintas direcciones. Según la dirección en que mira y el rango y ángulo configurados sólo es capaz de observar las casillas en color salmón. Los agentes y los obstáculos pueden tapar la visión según cómo se configuren (e.g. un zombi oculta la visión pero un botiquín no). Para la implementación del sistema se creó una librería básica de funciones geométricas para la obtención de vectores perpendiculares o paralelos a otros, trazado de segmentos con cierta dirección, origen y longitud, detección de colisiones, etc.



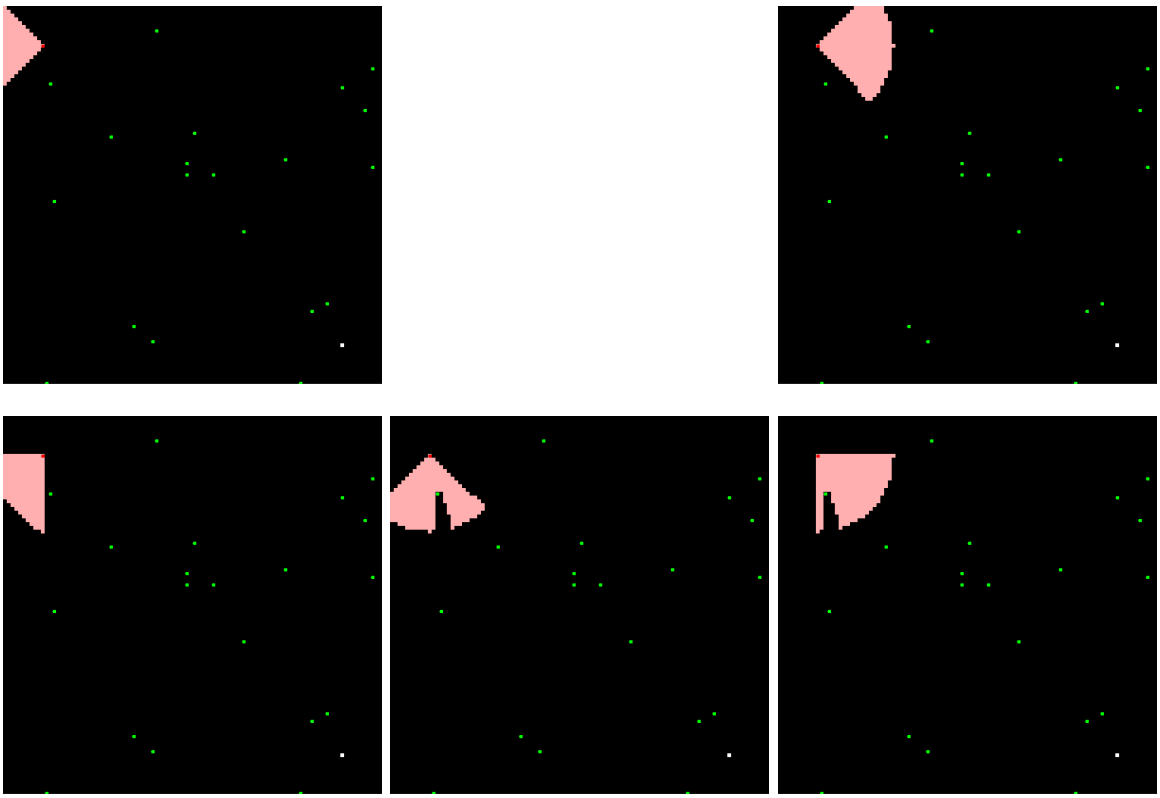
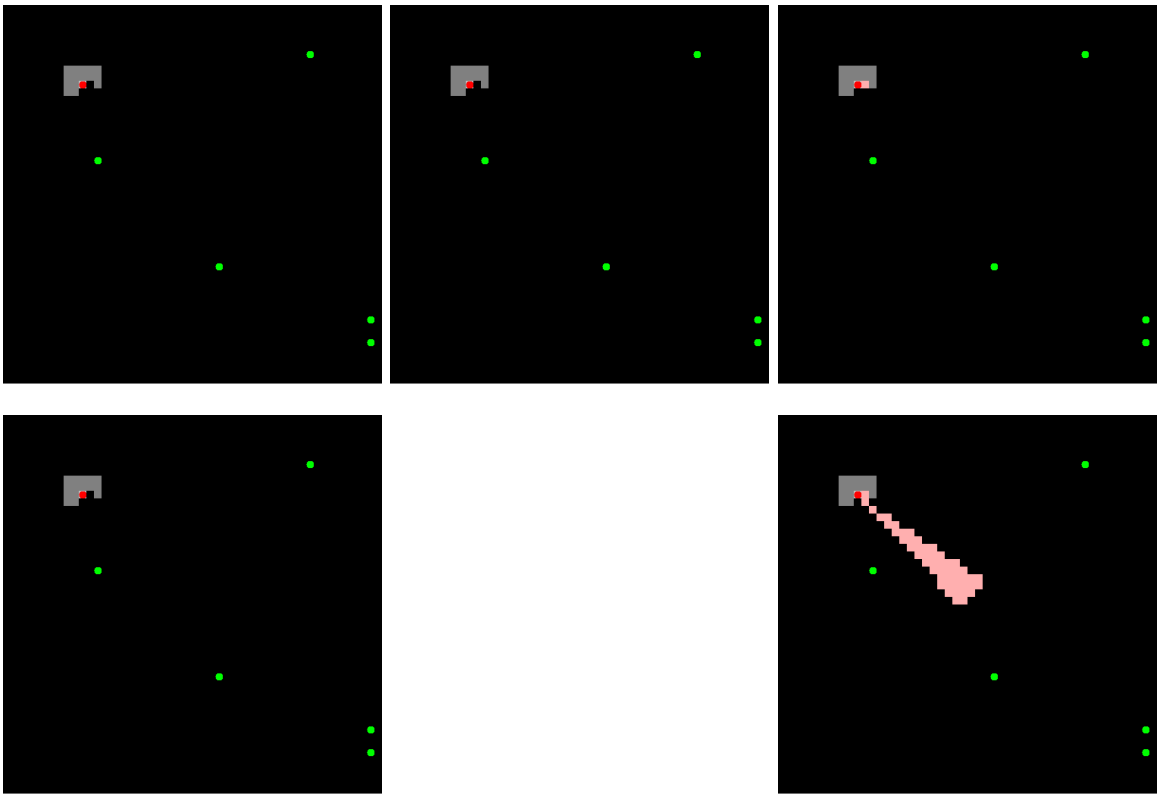


Figura 16. Visión artificial (en color salmón) de un agente (en color rojo) desde 8 distintas direcciones en presencia de otros agentes (en color verde)



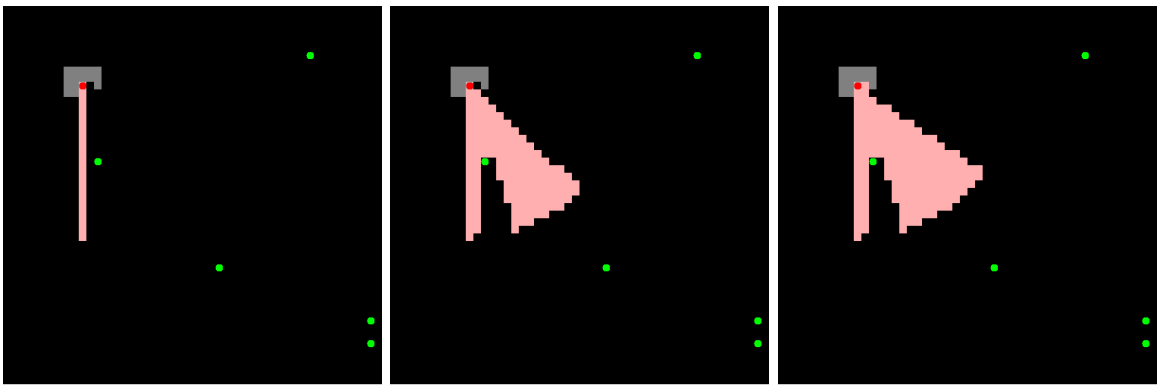


Figura 17. Visión artificial (en color salmón) de un agente (en color rojo) desde 8 distintas direcciones en presencia de otros agentes (en color verde) y obstáculos (en color gris)

Adicionalmente, los supervivientes constan de un sistema de navegación por el mapa mediante el algoritmo convencional de planificación de rutas A*. A medida que va recorriendo el mapa, cada superviviente va memorizando las posiciones vistas y las posibles entidades que había en dichas posiciones. Implementado para el proyecto, este sistema permite a los supervivientes navegar hacia posiciones "relevantes" (e.g.: un agente o ítem visto, una posición observada lejanamente cuyas posiciones detrás de ella son desconocidas, etc.) esquivando los obstáculos del mapa. Este sistema también permite dotar al superviviente de memoria, pues sabe cuál fue la última posición en que vio algo. La memoria proporciona información incompleta ya que puede albergar datos incorrectos o desfasados correspondientes a elementos observables no estáticos (e.g.: agentes que se mueven) o que pueden desaparecer (e.g.: ítems recogidos por otros agentes). Esto fomenta el realismo pues un agente puede decidir ir a buscar un ítem que vio hace un tiempo y que éste ya no se encuentre donde estaba al tener a la vista dicha localización, forzando al agente a actualizar su información eliminando el recuerdo de dicho ítem y a optar por otra posible acción.

Los supervivientes también reciben turnos en los que pueden realizar acciones. Inicialmente se generaron comportamientos deterministas de superviviente para probar el entorno, pero dado que el interés del proyecto está en la generación automática de comportamientos, en concreto mediante búsquedas evolutivas con GP, la explicación de las fases del turno de los supervivientes se pospone al capítulo 5.4.2.

Para moverse a una casilla dada a los supervivientes se les aplican restricciones similares a las de los zombis, esto es, un superviviente no se moverá si existen obstáculos o agentes en la casilla de destino que le bloqueen el paso u obstáculos en la casilla actual del superviviente que frenen su movimiento.

Cuando un superviviente ataca a una posición donde se encuentra un zombi debe reunir las siguientes condiciones para que el ataque tenga éxito:

- Tiene arma y munición
- El objetivo está dentro de su campo de visión y del alcance del arma

- Acierta aleatoriamente el disparo con probabilidad = <puntería> x <precisión del arma>

Si el ataque tiene éxito, se reduce la vida del zombi conforme al daño de ataque del arma y, si ésta llega a 0 o menos, se elimina al zombi del mapa y se le saca de la cola de eventos del *Schedule*.

Hubiera o no zombi en la posición objetivo, tras un disparo se genera estímulo en la posición objetivo y en la ocupada actualmente por el superviviente.

4.3.4 Elementos de entorno

Diseminados por el mapa (en la capa de entorno) se pueden encontrar ubicaciones especiales, obstáculos e ítems.

Todo elemento de entorno debe definir si bloquea el movimiento y/o la visión de los agentes. Ambas características están directamente relacionadas con el funcionamiento de los sistemas de navegación y visión artificiales para supervivientes descritos en el apartado 4.3.3.

Los obstáculos deben definir adicionalmente si quitan la vida de los supervivientes por un lado y/o la de los zombis por otro que pasan por ellos. También deben indicar en qué medida ralentizan el movimiento, de manera que un agente que quiera moverse a una casilla que contiene un obstáculo tiene cierta probabilidad de no poder moverse ese turno.

Toda la lógica de interacción de los elementos de entorno se sitúa en los agentes, por lo que es en cada tipo de agente donde debe especificarse el comportamiento del mismo ante la presencia de una ubicación especial de entorno o un ítem.

En la versión actual de *Left4Sim* todas las ubicaciones especiales e ítems interactúan únicamente con los agentes supervivientes: Las entradas y salidas son, respectivamente, los puntos por donde los supervivientes comienzan en el mapa una simulación y los puntos a los que deben llegar. Si esto último pasa, el superviviente se le quita de la cola de eventos del *Schedule* y se actualiza correspondientemente su estado resultante de la simulación. Por otro lado, las cajas de municiones y los botiquines recuperan, respectivamente, una determinada cantidad, prefijada mediante una constante en la correspondiente clase del ítem, de munición gastada o vida quitada al superviviente que pase por encima de ellos.

Aparte de obstáculos, ubicaciones especiales e ítems, existe un tipo especial de elementos de entorno, los estímulos, que se recogen en una capa exclusiva para ellos. La capa de estímulos es un mapa de calor en que el cada casilla almacena un valor que representa la cantidad de estímulo (e.g.: ruido, olor) proveniente de dicha casilla; cuanto más alto es el valor, mayor calor hay en la casilla y mayor es la intensidad con que “algo” está ocurriendo allí. De esta manera se implementa un sistema común de percepción de estímulos para los zombis sin necesidad de determinar con exactitud qué es lo que cada zombi individual es capaz de percibir.

Cada vez que se produce un evento o acción perceptible por zombis en una casilla de cualquiera de las capas del mapa se introduce en la casilla afectada del mapa de calor un estímulo con la intensidad que le corresponda según esté configurado. Actualmente se reconocen en *Left4Sim* los siguientes tipos de estímulos, ordenados de mayor a menor intensidad: Origen de un disparo, movimiento de superviviente, objetivo de un disparo, presencia de un superviviente.

El sistema unificado de estímulos se completa mediante una constante que define la distancia máxima a la que un estímulo es perceptible, K_D, y otra para la cantidad de turnos que persiste un estímulo (e.g.: el rastro de un olor o el eco de un sonido), K_T, junto con una función, f_D, que define el modo en que el estímulo pierde intensidad según la distancia respecto de la localización original del mismo, y otra función, f_T, que hace lo propio con el tiempo transcurrido desde el turno en que se produjo el estímulo.

De manera más formal, si suponemos que se introduce un estímulo *x* de intensidad I_x en la posición *p*, y es el turno *t* de la simulación, actualizaremos el estímulo actual de cada posición *q*, $E_{q,t}$, de la siguiente manera:

$$E_{q,t} = E_{q,t} + f_D\big(d(p,q)\big) \times I_x$$

$$f_D(d) = \begin{cases} {}^{1}\!/_{(1+d)} & si\ d \le K_D \\ 0\ si\ d > K_D \end{cases}$$

Adicionalmente, a cada posición *p* se le añade el recuerdo del valor de estímulo de turnos anteriores:

$$E_{p,t} = E_{p,t} + \sum_{i=1}^{K_T} f_T(i) \times E_{p,t-i}$$

$$f_T(t) = \begin{cases} {}^{0'1}\!/_{t} & si\ t \le K_T \\ 0\ si\ t > K_T \end{cases}$$

En conjunto se consigue que los zombis sean capaces de sentir a los supervivientes cuando éstos están cerca o seguir su rastro si acaban de pasar por un sitio o han escuchado sonidos suyos generados a cierta distancia.

En la Figura 15 se mostró un ejemplo de visualización de la capa de estímulos tras la realización de unos cuantos movimientos por parte de un superviviente.

4.3.5 Parametrización inicial

El entorno se inicializa para la simulación de dos maneras: mediante constantes definidas en las clases donde aplican (e.g. el modelo, los agentes, etc.) y mediante clases configuradoras. Aunque no está completamente acabado de implementar, la idea de las clases configuradoras es abarcar, en la mayor medida posible, todos los parámetros que definen el modelo para

poder centralizar en un único punto la gran cantidad de constantes introducidas en *Left4Sim* para llevar a cabo una simulación de un escenario concreto. Gran parte de las constantes, aquellas relacionadas con los agentes y los elementos de entorno, ya han sido descritas en anteriores apartados, por lo que aquí sólo se presentarán las que se establecen a nivel del modelo a alto nivel junto con algunas de las características de las clases configuradoras:

- Constantes del modelo:
  - Cantidad inicial y frecuencia/cantidad de generación de zombis
  - Separación mínima de los zombis nuevos con respecto a los supervivientes
  - Cantidad inicial y frecuencia/cantidad de generación de botiquines
  - Cantidad inicial y frecuencia/cantidad de generación de cajas de munición
  - Frecuencia de actualización de los avances (i.e. camino restante medio hacia las salidas) de cada superviviente
  - Número máximo de pasos de simulación
  - Número máximo de pasos que un superviviente puede permanecer sin realizar acciones antes de obligarle a suicidarse

- Configuradores:
  - Características del mapa y elementos de entorno:
    - Dimensiones del mapa
    - Tipo y ubicación de obstáculos
  - Características de los supervivientes → Método para instanciar nuevos supervivientes:
    - Comportamiento determinista con atributos al azar
    - Basado en árbol evolucionado de GP leído de fichero
    - etc.
  - Reparto de los supervivientes por las casillas de entrada del mapa:
    - Totalmente al azar
    - Distribución uniforme por todas las entradas
    - etc.

4.3.6 Desarrollo de la simulación

Una simulación consiste básicamente en una sucesión de *steps* o pasos, cada uno de los cuales se compone a su vez de la siguiente secuencia ordenada de fases:

1. Actualización del mapa de estímulos
2. Acciones de los zombis
3. Generación de nuevos ítems y zombis
4. Acciones de los supervivientes
5. Registro de trazas de la simulación
6. Comprobación de final de la simulación:
   - Máximo número de pasos alcanzado
   - No quedan supervivientes activos, esto es, supervivientes que todavía no han escapado, no se han suicidado o no han sido devorados por zombis

5. Dominio de definición del problema

5.1 Planteamiento de objetivos

Muchos son los objetivos que podríamos plantearnos alcanzar una vez la plataforma de simulación está preparada. A lo largo del capítulo 3 se dio un repaso global a lo que la investigación en IA de videojuegos ha dedicado sus esfuerzos en los últimos años, principalmente en cuanto a la inteligencia de los NPC, así como los nuevos focos de interés que están surgiendo.

Para este proyecto, como primera toma de contacto con la plataforma y, en general, con la IA de videojuegos, lo que planteamos como objetivo fue aplicar técnicas de aprendizaje automático en los NPC supervivientes para que desarrollaran comportamientos "interesantes" tanto a nivel individual como de equipo que para hacerles capaces de huir, encontrando alguna de las salidas, o, al menos, sobrevivir durante el tiempo que dura la simulación.

Mediante el aprendizaje de pautas de comportamiento "interesante" buscamos que nuestros NPC supervivientes adopten pautas de interacción con el entorno que sirvan a sus fines de supervivencia, y que además éstas introduzcan cierto grado de novedad o descubrimiento en las soluciones obtenidas, de manera que se pudiera llegar a aportar un valor añadido para el hipotético desarrollador de un videojuego basado en *Left4Sim*. No se trata de llegar a prescindir del conocimiento y capacidad creativa de un experimentado desarrollador de videojuegos, pero sí de proporcionar una valiosa ayuda para encontrar eficazmente otras alternativas de comportamiento inicialmente no vistas y tratar de dar un paso hacia una definitiva ruptura de la brecha existente entre el videojuego académico y el videojuego comercial (26) (27).

Los comportamientos que se encuentren deben dotar al superviviente de capacidad para tomar una decisión en cada momento sobre qué acción realizar de cuantas tiene a su mano:

- Moverse a una casilla dentro de su rango de velocidad de movimiento
- Disparar a una posición
- Mirar hacia una dirección distinta
- Permanecer quieto

5.2 Elección de la representación

Para elegir la representación y técnica a aplicar para encontrar soluciones a nuestro problema, un posible punto de partida es fijarnos en la metodología de la Figura 8 (22). Dado que existen ciertos elementos de aleatoriedad en *Left4Sim* (e.g.: el movimiento de los zombis o el ataque a los mismos), el mundo del juego es no determinista, que equivale a tener un problema ruidoso, lo cual, según el autor, hace apropiado el uso de técnicas de computación evolutiva.

Sin embargo, con esa misma metodología en mente, se podrían haber empleado otras posibles técnicas ateniéndonos a otras características de nuestro entorno, tanto a nivel del rol del jugador como del mundo del juego.

Por una parte, a nivel del rol del jugador humano, en realidad no se ha implementado una interfaz para que un jugador humano pueda interactuar con el juego, pero, de hacerlo, cualquiera de los roles propuestos en la metodología es viable: El jugador podría controlar a un único superviviente en la línea de un FPS, pero también podría controlar al equipo entero mediante órdenes de alto nivel, acercándose entonces más al género de los RTS.

Por la parte del mundo del juego, aparte del ya mencionado no determinismo de *Left4Sim*, el resto de características son:

- Basado en turnos: Cada paso de la simulación es un turno
- Inaccesible: Los supervivientes y los zombis sólo tienen acceso a información sobre su entorno más próximo
- Discreto: Los conjuntos de acciones posibles, las entidades presentes y la propia disposición del mapa como capas 2D solapadas son todos limitados y discretos
- Estático: El *Schedule* de *MASON* asigna turnos a cada agente quedando todo a la espera de la toma de decisión por parte de dicho agente
- No determinista: No es perfectamente predecible el siguiente estado del entorno dadas las acciones decididas por los agentes y el estado actual del mismo dadas las pinceladas de aleatoriedad existentes como:
  - La dirección de movimiento de los zombis en ausencia de estímulos
  - El éxito o fracaso del ataque de un superviviente a un zombi
  - El movimiento a través de obstáculos que ralentizan el paso

Con esta caracterización del entorno la metodología propone gran cantidad de posibles técnicas y algoritmos aparte de los algoritmos evolutivos, como son las redes neuronales artificiales, algoritmos de *Swarm,* razonamiento con *backtracking*, etc.

Centrándonos en los objetivos que perseguimos, pueden identificarse dos capacidades como deseables (si no exigibles), de cara a la elección de la técnica que usemos para representar el comportamiento de nuestros agentes:

1. Flexibilidad y capacidad de subsunción de unos comportamientos en otros
2. Fácil interpretación (por un humano) de los comportamientos obtenidos

Esto es, los comportamientos interesantes que buscamos, una vez encontrados a nivel de un único agente individual del equipo, nos interesaría que se agruparan para formar parte de la inteligencia cooperativa de todo el equipo completo, y que el comportamiento básico que sirve para alcanzar una única meta simple del entorno, agrupado con otros resultara en un comportamiento complejo capaz de alcanzar la meta final de nuestros agentes inteligentes.

Todo lo anterior sin dejar que las estructuras encontradas terminen siendo “cajas negras” ilegibles, es decir, que un humano pueda entender el comportamiento que se ha obtenido para modificarlo, probarlo, adaptarlo, etc.

Así las cosas, optamos por representar la inteligencia de nuestros agentes mediante árboles de comportamiento pues su lectura es inmediata y porque permiten de manera natural llevar a cabo la fusión de unos con otros para obtener estructuras más complejas. Los árboles son las estructuras típicamente usadas para representación de los individuos en GP, por lo que se decidió utilizar este tipo de técnica evolutiva para la obtención automática de los árboles.

5.3 Árboles de comportamiento

Los árboles de comportamiento, tal y como se definen en este proyecto, son estructuras de datos que codifican un conjunto de pautas de comportamiento que permiten a un agente concreto en un momento dado de la simulación y estado particular del entorno tomar la decisión acerca de qué acción llevar a cabo a continuación.

La Figura 18 muestra un ejemplo de árbol de comportamiento de superviviente, el único tipo de agente que, como se describió en la presentación de la plataforma *Left4Sim*, emplea pautas de comportamiento no prefijadas.

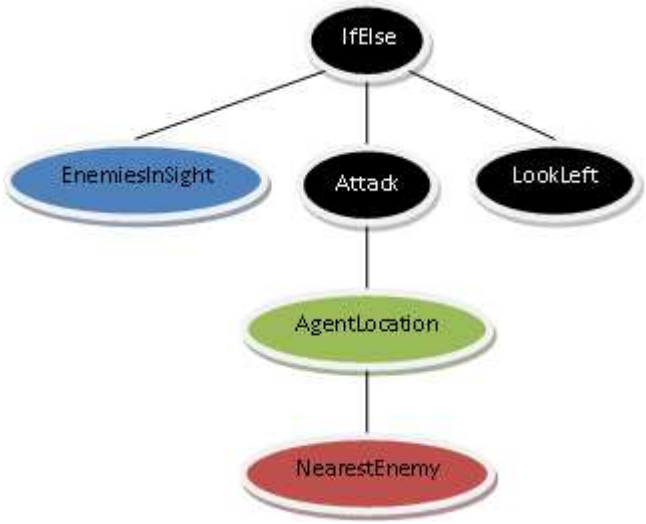


Figura 18. Ejemplo de árbol de comportamiento de un superviviente con varios tipos de nodo: acción (negro), condición (azul), posición (verde) y agente (rojo)

Todo superviviente tiene su propio árbol, que se compone de un único nodo raíz, el cual puede tener nodos hijos, los cuales, a su vez, pueden tener más hijos, y así hasta llegar a los nodos hoja, que son los que ya no tienen hijos.

Los árboles, y en consonancia, el algoritmo de GP empleado, corresponden a la variante *strongly typed*, lo que significa que cada nodo se corresponde con una única función, la cual tiene, por un lado, un único tipo de salida, que es la clase del objeto que el nodo devuelve a su nodo padre cuando es evaluado, y una tupla de tipos de entrada, cada uno de los cuales ha de

ser devuelto por un nodo hijo. Por lo tanto, evaluar un árbol consiste en evaluar su nodo raíz, el cual, recursivamente, irá evaluando todos sus sucesivos hijos hasta llegar a los nodos hoja. Adicionalmente a la información proveniente de los hijos, cada nodo puede consultar al modelo *MASON* para obtener toda la información que el agente superviviente conoce sobre el entorno.

El nodo raíz y, por tanto, el propio árbol de comportamiento de un superviviente en *Left4Sim* siempre devuelven una acción al ser evaluado. El resto de nodos puede devolver cualquiera de los tipos siguientes:

- Agente
- Posición
- Numérico
- Acción
- Condición

Los árboles deben estar bien formados, es decir, todo nodo debe respetar los tipos de entrada y salida de la función que albergan, y situar, como se ha dicho, un nodo de acción como raíz del árbol. Sin embargo, incluso en árboles bien formados, es posible que la evaluación falle, es decir, que un nodo no pueda devolver ningún objeto (e.g.: la función *NearestEnemy*, que se lista más adelante en la Tabla 1, sólo es capaz de devolver un objeto de tipo agente si el superviviente tiene enemigos a la vista). En tales casos se transmite excepcionalmente un objeto especial vacío *NULL*, propagándose desde el nodo origen hacia arriba hasta donde proceda.

En el dominio de *Left4Sim* este efecto se produce porque empleamos expresiones lógicas en el conjunto de funciones. Si una expresión lógica dada como *EnemiesInSight* (ver Tabla 6) aparece en el árbol como hija de un nodo *IfElse* (ver Tabla 5), el cual opta por una acción hija u otra según el valor de la expresión lógica hija, y la misma expresión o una derivada de ella aparece en alguno de los subárboles de acción del nodo *IfElse* como hija a su vez de otro nodo *IfElse* tal y como se aprecia en la Figura 19, dado que el estado de la simulación es constante durante toda la toma de decisión del superviviente, la expresión tendrá un valor determinado cuando se evalúe el *IfElse* superior que se mantendrá hasta la evaluación del *IfElse* inferior, haciendo que la 2º rama de acción del *IfElse* inferior no sea usada nunca en la evaluación. Sin embargo, esta clase de ramas no usadas sí que pueden seguir transmitiéndose por la población mediante los operadores genéticos que se estén aplicando, lo cual puede ser negativo si la propia rama no tiene sentido o no contribuye de forma positiva al comportamiento representado por el árbol. Otra posible circunstancia que da lugar a ramas no usadas es la presencia de tautologías (i.e. expresiones siempre ciertas) o contradicciones (i.e. expresiones siempre falsas) como condición de un nodo *IfElse*, lo cual descarta sistemáticamente una de las ramas de acción. La misma Figura 19, muestra un ejemplo de árbol con ramas que nunca serán usadas en la evaluación, y que, por tanto, en evaluación es idéntico al árbol de la Figura 18.

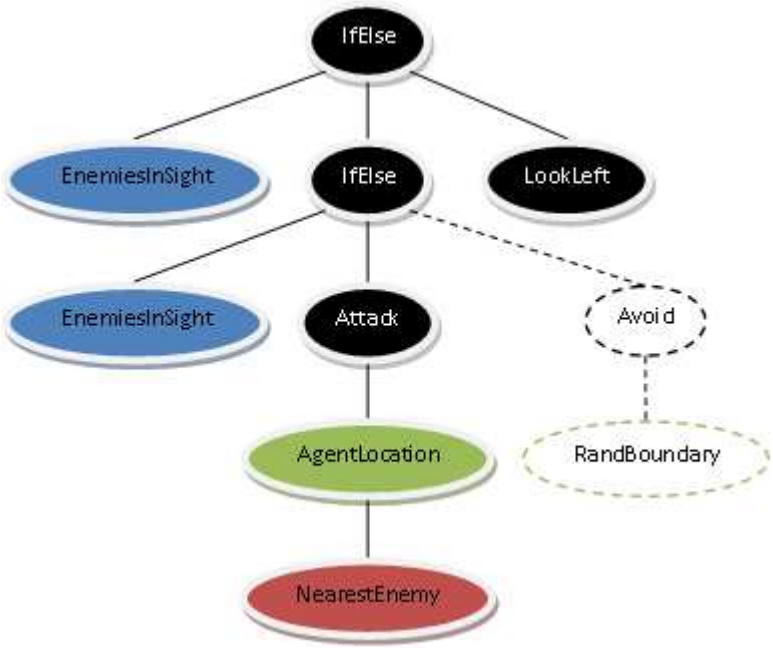


Figura 19. Ejemplo de árbol de comportamiento con ramas no usadas (marcadas con trazos discontinuos)

El conjunto completo de funciones empleado en los escenarios con un único superviviente se detalla en las tablas siguientes: Tabla 1, Tabla 2, Tabla 3, Tabla 4, Tabla 5 y Tabla 6.

Tabla 1. Funciones de agente

Función	Hijos	Retorno
Me	-	Agente propietario del árbol
NearestEnemy	-	Enemigo (visible) más próximo
RandEnemy	-	Enemigo (visible) aleatorio
NearestEnemyInAttackRange	-	Enemigo (visible) más próximo a distancia de ataque
RandEnemyInAttackRange	-	Enemigo (visible) aleatorio a distancia de ataque
NearestAlly	-	Aliado (visible) más próximo
RandAlly	-	Aliado (visible) aleatorio
FarthestAlly	-	Aliado (visible) más lejano

Tabla 2. Funciones de posición

Función	Hijos	Descripción
NearestExit	-	Salida conocida más próxima
NearestBoundary	-	Punto frontera más próximo
FarthestBoundary	-	Punto frontera más alejado
RandBoundary	-	Punto frontera aleatorio
AgentLocation	1 agente	Posición del agente hijo
NearestMediKit	-	Botiquín más próximo
RandomMediKit	-	Botiquín aleatorio
NearestAmmoBox	-	Caja de munición más próxima
RandomAmmoBox	-	Caja de munición aleatoria

Tabla 3. Funciones numéricas

Función	Hijos	Descripción
Distance	2 posiciones	Distancia entre las 2 posiciones hijas
Random	-	Constante numérica aleatoria
AttackRange	-	Rango de ataque del agente propietario del árbol
Health	1 agente	Vida restante del agente hijo
Ammo	1 agente	Munición restante del agente hijo
AlliesInRangeFrom	1 número y 1 posición	Cantidad de aliados visibles a la distancia dada por el número hijo desde la posición hija

Tabla 4. Funciones de acción

Función	Hijos	Descripción
GoTo	1 posición	Planifica una ruta en dirección hacia la posición hija y mueve hacia la siguiente posición de la ruta (o se encara si no la puede ver)
Attack	1 posición	Ataca la posición hija (o mueve hacia ella si no está al alcance) si tiene munición
Face	1 posición	Encara la posición hija
LookRight	-	Gira 90 grados a la derecha
LookLeft	-	Gira 90 grados a la izquierda
Wait	.	No hace nada
Avoid	1 posición	Mueve en dirección opuesta a la posición hija
ShareMap	1 agente	Comparte la información conocida sobre el mapa con un agente aliado (o mueve hacia él si no está al alcance)

Tabla 5. Funciones auxiliares de acción

Función	Hijos	Descripción
Do	2 acciones	Realiza secuencialmente las 2 acciones hijas
IfElse	1 condición y 2 acciones	Si se cumple la condición hija realiza la acción hija 1ª, si no, realiza la acción hija 2ª

Tabla 6. Funciones de condición

Función	Hijos	Descripción
Not	1 condición	Operación lógica *not* sobre la condición hija
And	2 condiciones	Operación lógica *and* sobre las 2 condiciones hijas

Or	2 condiciones	Operación lógica *or* sobre las 2 condiciones hijas
Greater	2 números	Operación lógica > sobre los 2 números hijos
Equal	2 números	Operación lógica = sobre los 2 números hijos
RandomBoolean	-	Constante booleana aleatoria
ExitKnown	-	Cierto si el agente propietario del árbol conoce alguna salida
EnemiesInAttackRange	-	Cierto si hay enemigos (visibles) a distancia de ataque del agente propietario del árbol
EnemiesInSight	-	Cierto si hay enemigos visibles por el propietario del árbol
MediKitKnown	-	Cierto si el agente propietario del árbol conoce algún botiquín
AmmoBoxKnown	-	Cierto si el agente propietario del árbol conoce alguna caja de munición
IsBeingAttacked	1 posición	Cierto si la posición hija está siendo atacada
AmIBeingAttacked	-	Cierto si el agente propietario del árbol está siendo atacado
IsHurt	1 posición	Cierto si la posición hija está herida
AmIHurt	-	Cierto si el agente propietario del árbol está herido
HasAmmo	1 agente	Cierto si el agente hijo tiene munición
HaveIAmmo	-	Cierto si el agente propietario del árbol tiene munición
AlliesInSight	-	Cierto si hay aliados visibles por el propietario del árbol

5.4 Evolución mediante GP

5.4.1 Librería ECJ

Para llevar a cabo la evolución de los árboles se utilizó la implementación de GP de la librería *Java* de algoritmos de computación evolutiva *A Java-based Evolutionary Computation Research System* (ECJ)[63].

Además de GP, ECJ tiene implementaciones de algunos los más populares algoritmos de computación evolutiva como el algoritmo genético clásico, las estrategias evolutivas o la evolución gramatical. ECJ soporta distintos tipos de representación: cadenas de tamaño fijo o variable de valores enteros o decimales, árboles, gramáticas, reglas, etc. Para cada representación ofrece una ingente cantidad de operadores genéticos de mutación y cruce, selectores de progenitores, métodos de inicialización de poblaciones e individuos y un largo etc. de posibilidades para configurar el algoritmo que quieras y adaptarlo a tus necesidades.

La metodología de trabajo con ECJ de manera aislada consiste básicamente en la elaboración de un fichero de configuración donde se especifican los distintos parámetros del algoritmo que queremos usar, y la codificación de unas, en principio, pocas líneas de código donde se implementa la función de *fitness* de nuestros individuos. ECJ asigna valores por defecto a la mayor parte de las variables de configuración, siendo particularmente interesante poder emplear parámetros similares a los que Koza utilizó inicialmente en algunos de sus trabajos clásicos. Adicionalmente es posible crear subclases de prácticamente todos los elementos del proceso evolutivo allá donde nos interese.

ECJ tiene otras características interesantes como la posibilidad de paralelizar algunas de las tareas más costosas en un proceso evolutivo como son la evaluación del *fitness* de la población o la obtención de la población para la siguiente generación mediante los operadores de cruce.

5.4.2 Integración con *Left4Sim*

ECJ lleva el peso del proceso principal en cada una de las pruebas que realizamos en este trabajo. *Left4Sim* entra en juego cuando es necesario evaluar el *fitness* de un individuo cuyo árbol de comportamiento ha sido obtenido por ECJ.

La mayor complejidad de la integración de ECJ con nuestro entorno basado en MASON, *Left4Sim*, fue debida a que se tuvo que reconfigurar éste para poder ser invocado sin interfaz gráfica por ECJ cada vez que fuera necesario evaluar un individuo.

Sin embargo, *Left4Sim* no es meramente una herramienta externa que el proceso evolutivo principal ejecutado por ECJ puede invocar a demanda. Dado que lo que evolucionamos son árboles de comportamiento que los agentes superviviente evalúan en cada paso de la

[63] http://cs.gmu.edu/~eclab/projects/ecj/

simulación para determinar qué acción realizar, fue necesario codificar configuradores de agente específicos que almacenara toda la información necesaria para poder invocar el evaluador de árboles ECJ desde dentro de *Left4Sim*.

En 4.3.3 se describió el comportamiento prefijado de los zombis y se explicaron algunos de los sistemas básicos (esencialmente la visión y la navegación) con que cuentan los supervivientes en *Left4Sim.* A continuación se detalla la secuencia de turno de un superviviente cuyo comportamiento viene determinado por un árbol derivado mediante GP:

1. Visión del entorno en la dirección actual de encaramiento
2. Selección de siguiente acción a ejecutar:
    a. Recuperación de la última iniciada pero no finalizada (e.g.: acciones compuestas con *Do*), u
    b. Obtención de una nueva mediante evaluación del árbol de comportamiento, o
    c. Elección forzosa de acción *Wait* si la evaluación falla
3. Ejecución de la acción escogida
4. Suicidio si se ejecutaron demasiadas acciones de tipo *Wait*

Los supervivientes siempre generan estímulo para los zombis en la casilla donde se encuentran y, además, determinadas acciones pueden generar estímulos adicionales (e.g.: disparar, moverse, etc.).

Un superviviente seguirá recibiendo turnos en la simulación mientras no sea devorado por un zombi, pase por una casilla de salida o, naturalmente, cuando se acabe el tiempo máximo estipulado para la simulación.

5.4.3 Evaluación del *fitness*

Como en todo problema de GP es necesario definir una función de *fitness* que valore adecuadamente el desempeño de un individuo en el entorno y nos lleve hacia el valor óptimo de dicha función.

En problemas como el que se nos presenta en *Left4Sim* no es posible saber de antemano cuál es la solución óptima y, es más, no tiene por qué existir necesariamente tal solución, ni ser ésta única. Sin embargo, con nuestro objetivo en mente de encontrar supervivientes con comportamiento “interesante”, capaces de escapar del mapa o, en su defecto, sobrevivir, tenemos un primer criterio principal para determinar la idoneidad del comportamiento de un superviviente. Únicamente este criterio no es suficiente para guiar el proceso evolutivo, por lo que decidimos buscar una serie de criterios cuantitativos adicionales adecuados para nuestro problema.

La lista completa de criterios empleados actualmente en *Left4Sim* en la valoración del comportamiento de un superviviente es la siguiente:

- Estado final: escapar > sobrevivir > morir > suicidarse
- Distancia recorrida: Mejor cuanto más se haya llegado a aproximar a la salida
- Zombis asesinados: Mejor cuantos más zombis haya asesinado
- Vida restante: Mejor cuanto mayor sea su vida al final de la simulación

De acuerdo a lo anterior, calculamos los siguientes *scores* normalizando en todos los casos para ser siempre 0 el mejor valor posible y 1 el peor:

- *distanceScore* → $score_D = {}^{D^*}/_{D_I}$

Con D^* la menor distancia hacia la salida alcanzada en toda la simulación por el superviviente y D_I la inicial; 0 indica que el superviviente alcanzó la salida mientras que 1 indica que el superviviente no se movió del origen o incluso se alejó de la salida.

- *killScore* → $score_K = 1 - {}^{K}/_{K^*}$

Con K (*rawKillScore*) la cantidad de zombis asesinados por el superviviente y K^* la cantidad de zombis asesinados por el mejor superviviente de la simulación; 0 indica que el superviviente mató tantos zombis como el mejor superviviente de la población mientras que 1 indica que el superviviente no mató ni un solo zombi.

- *lifeScore* → $score_L = 1 - {}^{L_F}/_{L_I}$

Con L_F la vida final del superviviente y L_I la inicial; 0 indica que el superviviente conservó toda la vida inicial, mientras que 1 indica que el superviviente la perdió por completo.

$$R = \begin{cases} 0, si\ el\ superviviente\ escapó \\ 1, si\ el\ superviviente\ sobrevivió \\ 2, si\ el\ superviviente\ murió \\ 3, si\ el\ superviviente\ se\ suicidó \end{cases}$$

$$score = R + K_D \times score_D + K_K \times score_K + K_L \times score_L$$

Con K_D, K_K y K_L los respectivos pesos asignados a cada uno de los *scores* ($K_D + K_K + K_L = 1$); Obteniendo un resultado final entre 0 y 4, que es, esencialmente, el resultado o estado final del superviviente, matizado por los *scores* que obtuvo.

Una vez conocido el desempeño individual o *score* de cada superviviente, el *fitness* del individuo se obtiene promediando los *scores* de los N supervivientes del equipo:

$$fitness = \frac{1}{N} \times \sum_{i=1}^{N} score_i$$

Por último, y dado que en nuestro problema la evaluación del *fitness* es muy ruidosa, esto es, puede variar mucho entre simulaciones debido a la aleatoriedad del entorno (e.g.: un buen superviviente puede morir prematuramente por la ubicación de los zombis e ítems, o un mal superviviente puede alcanzar o quedar cerca de la salida por casualidad moviendo aleatoriamente) el *fitness* final se calcula lanzando varias simulaciones por individuo y promediando los resultados de *fitness* obtenidos. Incidiendo en este punto, los individuos que pasan por elitismo de una generación a otra se vuelven a evaluar en la nueva generación para aproximar mejor su *fitness*.

Al final de la evolución, los mejores individuos se evalúan, si es necesario, hasta llegar a un mínimo establecido de evaluaciones, y poder determinar con menor margen de error el mejor de toda la evolución. El seleccionado como mejor de la evolución se evalúa aún más para estimar adecuadamente sus tasas de escape, supervivencia, muerte y suicidio.

6. Experimentos y resultados

6.1 Configuración de los experimentos

6.1.1 Características del mapa

Todas las pruebas se han realizado en un mapa libre de obstáculos, el más sencillo posible, de 50x50 casillas, con la entrada y salida de supervivientes situadas próximas a extremos opuestos del mapa tal y como se aprecia en la Figura 20.

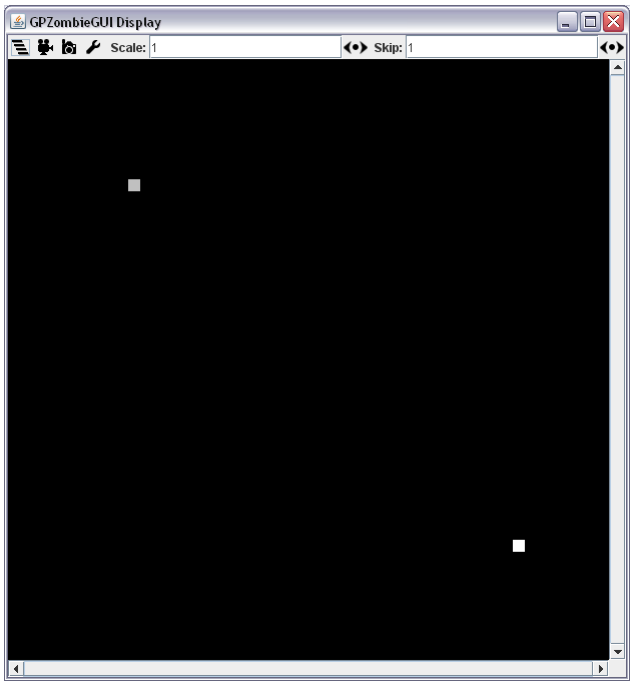


Figura 20. Mapa sin obstáculos para las pruebas. La entrada de supervivientes (gris) se sitúa en un extremo y la salida (blanco) en otro

Inicialmente se distribuyen 10 zombis, 10 botiquines y 10 cajas de munición aleatoriamente por el mapa. Cada 50 turnos se generan aleatoriamente 4 zombis, 1 botiquín y 1 caja de munición nuevos. Cualquier objeto generado debe ubicarse a, como mínimo, una distancia de 8 casillas de todo superviviente activo.

6.1.2 Características de los agentes

- Supervivientes
  - Velocidad = 1 casilla
  - Rango de visibilidad = 10 casillas
  - Ángulo de visibilidad = 120 grados
  - Vida inicial = 100 puntos
  - Munición inicial = 10 unidades
  - Obstáculos no atravesables = {}

        - Puntería = 100%
        - Arma (AK47):
            - Precisión = 65%
            - Número de ataques = 1
            - Rango de ataque = 4 casillas
            - Daño de ataque = 100 puntos

- Zombis
    - Velocidad = 1 casilla
    - Rango de visibilidad = 1 casilla
    - Vida inicial = 100 puntos
    - Munición inicial = infinita
    - Obstáculos no atravesables = {}
    - Rango de ataque = 1 casilla
    - Daño de ataque = 10 puntos

6.1.3 Características del entorno

- Vida recuperada por botiquín = 50 puntos
- Munición recuperada por caja de munición = 10 unidades
- Número máximo de turnos de simulación = 400
- Número máximo de turnos en espera para suicidio forzoso de superviviente = 200
- Frecuencia de actualización de avances de los supervivientes = 5 turnos
- Distancia máxima de percepción de estímulos = 4 casillas
- Memoria máxima de percepción de estímulos = 3 turnos

6.1.4 Número de supervivientes

Se han llevado a cabo pruebas con equipos de:

- 1 único superviviente
- 2 supervivientes
- 4 supervivientes

Todas las funciones que hacen referencia a supervivientes aliados se omiten en los experimentos de un único superviviente para reducir un poco el espacio de búsqueda y mejorar la búsqueda de soluciones evitando, en parte, la generación forzosa de acciones *Wait*.

6.1.5 Escenarios

Para cada una de las configuraciones de equipo se han probado los siguientes escenarios, incluyendo el objetivo del escenario, los valores de las constantes determinar los *scores* de los supervivientes y las modificaciones particulares aplicadas al mapa:

- Básico:
  - Los supervivientes deben escapar, o, en su defecto, sobrevivir, aproximándose lo más posible a la salida (K_D = 0'6), perdiendo la menor cantidad de vida posible (K_L = 0'3) y matando la mayor cantidad posible de zombis (K_K = 0'1)
  - No se modifican el mapa ni las características de los agentes

- Escape:
  - Los supervivientes deben escapar, o, en su defecto, sobrevivir, aproximándose lo más posible a la salida (K_D = 1; K_K = K_L = 0)
  - No se generan zombis, botiquines o cajas de munición

- Matanza:
  - Los supervivientes deben escapar, o, en su defecto, sobrevivir, matando la mayor cantidad posible de zombis (K_K = 1; K_D = K_L = 0)
  - No se generan botiquines

- Supervivencia:
  - Los supervivientes deben escapar, o, en su defecto, sobrevivir, acabando con la mayor cantidad de vida posible (K_L = 1; K_D = K_K = 0)
  - No se generan cajas de munición y los supervivientes empiezan sin munición y a la mitad de su vida máxima

- Combinado:
  - Idéntico al básico
  - Se introducen en el conjunto de funciones como nodos de acción los árboles completos derivados en cada uno de los escenarios especiales: escape (función *escape*), matanza (función *kill*) y supervivencia (función *survive*)

De esta forma, se sigue un planteamiento similar al trabajo de otros autores (45), en los que, dado el carácter multimodal del entorno, se aborda la disparidad de comportamientos necesarios desglosando el entorno por sus objetivos individuales. De esta forma, los diferentes entornos planteados pueden plantearse como pruebas de entrenamiento militares para misiones sencillas, y combinando las soluciones para dar respuesta al escenario completo. No obstante, hay que recalcar que nuestro problema no es exactamente multiobjetivo o multimodal, ya que sí hay soluciones que dominan sobre las demás, pues escapar del mapa prima sobre cualquier otra meta que puedan lograr los supervivientes.

Con este planteamiento, y aprovechando las características de los árboles del algoritmo *strongly typed* GP, la subsunción de árboles en árboles más complejos como nodos de acción se hace efectiva de manera natural en el escenario combinado.

6.1.6 Parámetros evolutivos principales

- Tamaño de la población = 1024 individuos (1 individuo representa 1 ó más supervivientes)
- Inicialización aleatoria de árboles (1 árbol por superviviente):

        - 50% *build* / 50% *grow*
        - 2 <= profundidad <= 6

- *Fitness* estandarizado: [0,+∞), siendo 0 el valor óptimo
- Nº de simulaciones por evaluación de *fitness* = 25
- Generación de descendientes:
    - Elitismo = 100
    - Selección de progenitores → Torneos de 7
    - Cruce (90%) → Recombinación de subárboles de 2 progenitores por sendos nodos seleccionados al azar
    - Mutación (10%) → Sustitución de subárbol en nodo seleccionado al azar por subárbol aleatorio:
        - 100% *grow*
        - Profundidad <= 5
    - Selección de nodos:
        - Nodos hoja (10%)
        - Nodos intermedios (90%)
    - Máxima profundidad de árboles generados = 17

- Duración fija de la evolución = 25/50 generaciones
- Nº de candidatos al mejor individuo de la evolución = 100
- Mínimo nº de simulaciones para la elección del mejor individuo de la evolución = 100
- Nº de simulaciones final del mejor individuo = 1000

6.1.7 Estadísticas analizadas

Las estadísticas estudiadas en este documento incluyen:

1. Resultados del mejor individuo de la evolución:
    a. Árboles evolucionados del equipo de supervivientes
        - Forma original
        - Forma simplificada a mano
    b. *Fitness* promedio y desviación típica
    c. Tasas promedio de estado final de cada superviviente

2. Evolución por generación del *fitness* de los individuos:
    - Valor promedio
    - Mejor valor

3. Evolución por generación de los *scores* (ver 5.4.3) promedio de cada superviviente:
    - distanceScore: [0-1]
    - lifeScore: [0-1]
    - rawKillScore: 0+

4. Evolución por generación de las tasas promedio de estado final de cada superviviente:
    - Escapar

- Sobrevivir
- Morir
- Suicidarse

6.2 Experimento 1: Supervivientes individuales

6.2.1 Básico

En este experimento se buscaron comportamientos para supervivientes individuales que cumplieran el objetivo principal del escenario: escapar del mapa o sobrevivir en su defecto, evitando ser devorado por los zombis o cometer suicidio.

El mejor superviviente encontrado (Figura 21) aprendió a cazar al zombi más próximo (*Attack (AgentLocation NearestEnemyInAttackRange)*), cuando hay alguno a la vista (*EnemiesInSight*). En caso de no tener ninguno a la vista, arbitrariamente (*AmIHurt*) encadena una acción de mirar en otra dirección (*LookLeft/LookRight*) con una de movimiento, a efectos, aleatorio (*Avoid RandBoundary/Avoid NearestBoundary*), o viceversa, de manera que puede ver si hay más zombis aproximándose desde otras direcciones de la que inicialmente está controlando.

Con este comportamiento el superviviente logra sobrevivir la mayor parte de las ocasiones (94% según la Figura 22), matando un número reducido de zombis pero no llega a escapar nunca porque apenas se desplaza hacia la salida desde su ubicación inicial.

Durante aproximadamente las 6 primeras generaciones la población apenas sabe atacar, muriendo en casi todos los casos (>90%) (Figura 25). Tras ese número de generaciones, poco después se encuentra la mejor solución (Figura 23) y la población va progresivamente abandonando los intentos por mejorar el *score* inicial de distancia (0'8) para buscar la salida, mejorando por el contrario los *scores* de vida restante (Figura 24) en paralelo con las tasas de supervivencia (Figura 25) hasta estabilizarse todo en torno a la generación 18.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(IfElse (Equal 13.786709799011431 0.6046819510712509) (Avoid RandBoundary) (IfElse EnemiesInSight (IfElse (Greater (Ammo NearestEnemy) (Ammo Me)) (Attack (AgentLocation NearestEnemyInAttackRange)) (Avoid (AgentLocation NearestEnemy))) (IfElse AmIHurt (IfElse EnemiesInSight (Do (Do LookRight (IfElse EnemiesInSight (IfElse (Greater 13.786709799011431 (Ammo Me)) (Attack (AgentLocation NearestEnemy)) (Avoid (AgentLocation RandEnemyInAttackRange))) (IfElse AmIHurt (IfElse EnemiesInSight (Do (Do (Attack (AgentLocation NearestEnemy)) (Avoid NearestBoundary)) (Attack (AgentLocation NearestEnemy))) LookLeft) (Do LookRight (Avoid RandBoundary))))) (Attack (AgentLocation NearestEnemy))) (IfElse (Equal 13.786709799011431 0.6046819510712509) (Avoid RandBoundary) (IfElse EnemiesInSight (Avoid RandBoundary) (IfElse AmIHurt (IfElse EnemiesInSight Wait (Do (Avoid RandBoundary) LookLeft)) (Do LookRight (Avoid RandBoundary)))))) (Do LookRight (Avoid NearestBoundary)))))

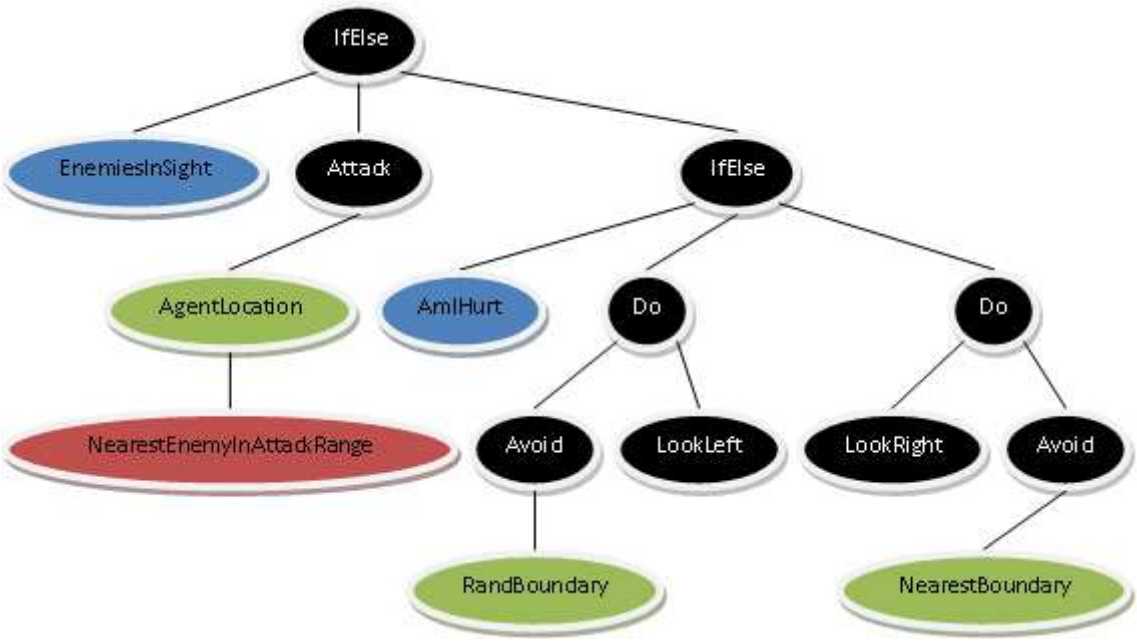


Figura 21. Árbol simplificado del escenario básico con 1 superviviente

b. ***Fitness* promedio y desviación típica** = 1'7614 (0'3098)

c. **Tasas promedio de estado final de cada superviviente**

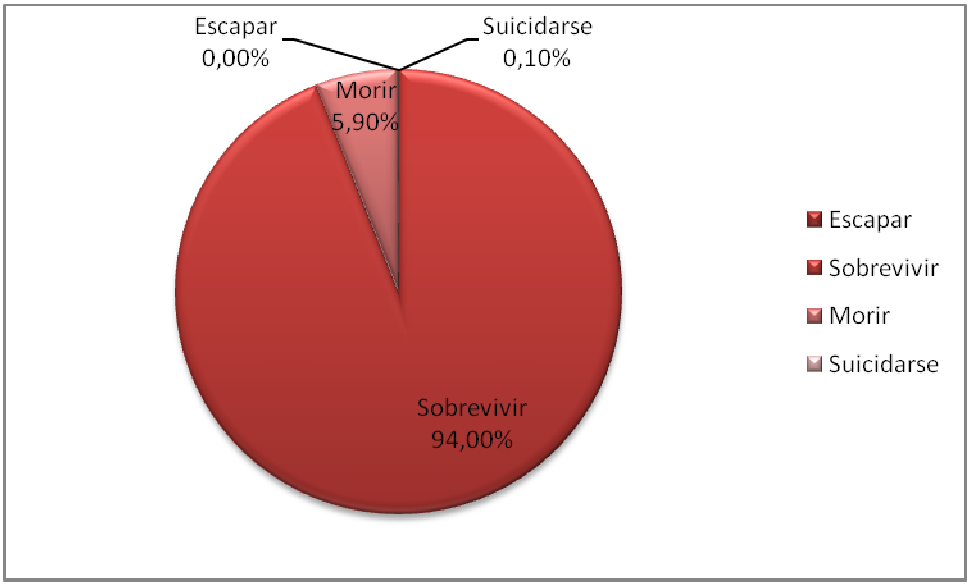


Figura 22. Tasas promedio de estado final del mejor individuo del escenario básico con 1 superviviente

2. **Evolución por generación del *fitness* de los individuos**

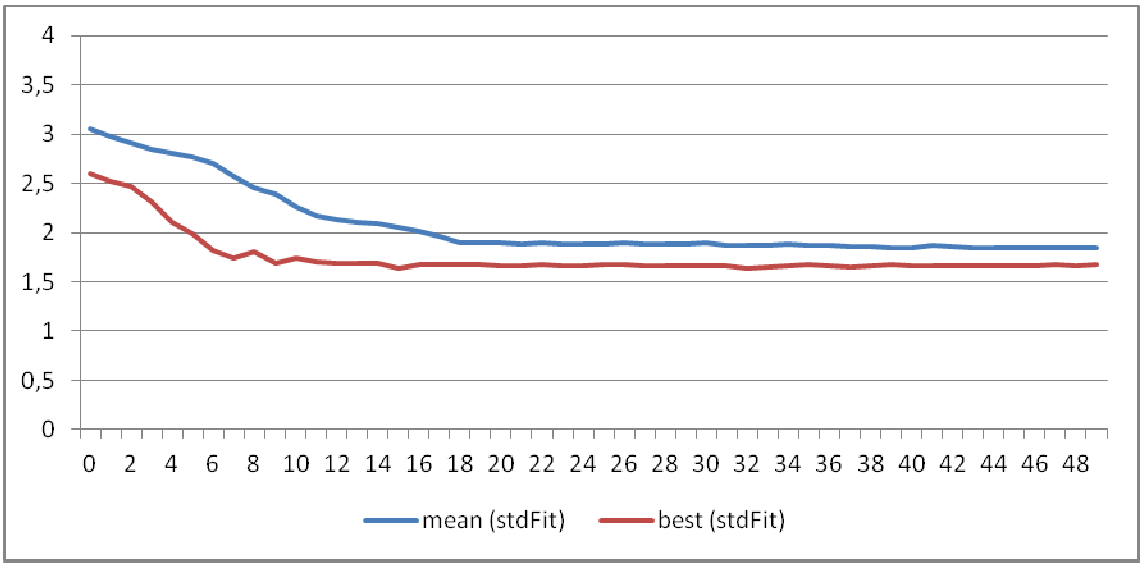


Figura 23. Evolución del *fitness* promedio y mejor del escenario básico con 1 superviviente

3. **Evolución por generación de los *scores* promedio de cada superviviente**

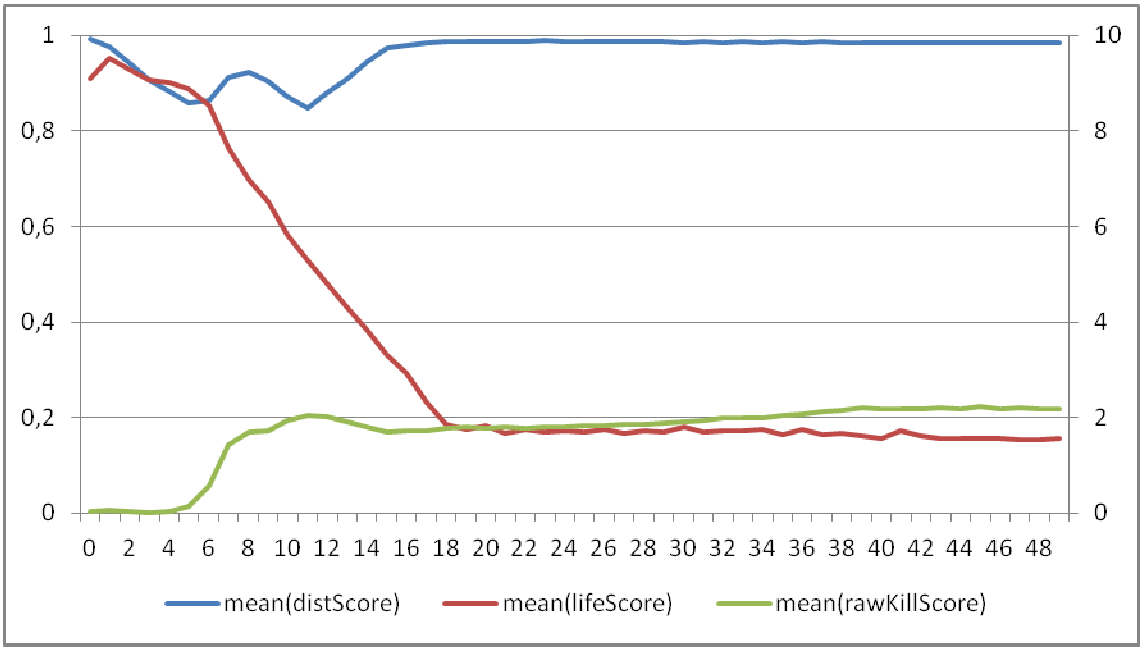


Figura 24. Evolución de los *scores* promedio del escenario básico con 1 superviviente

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

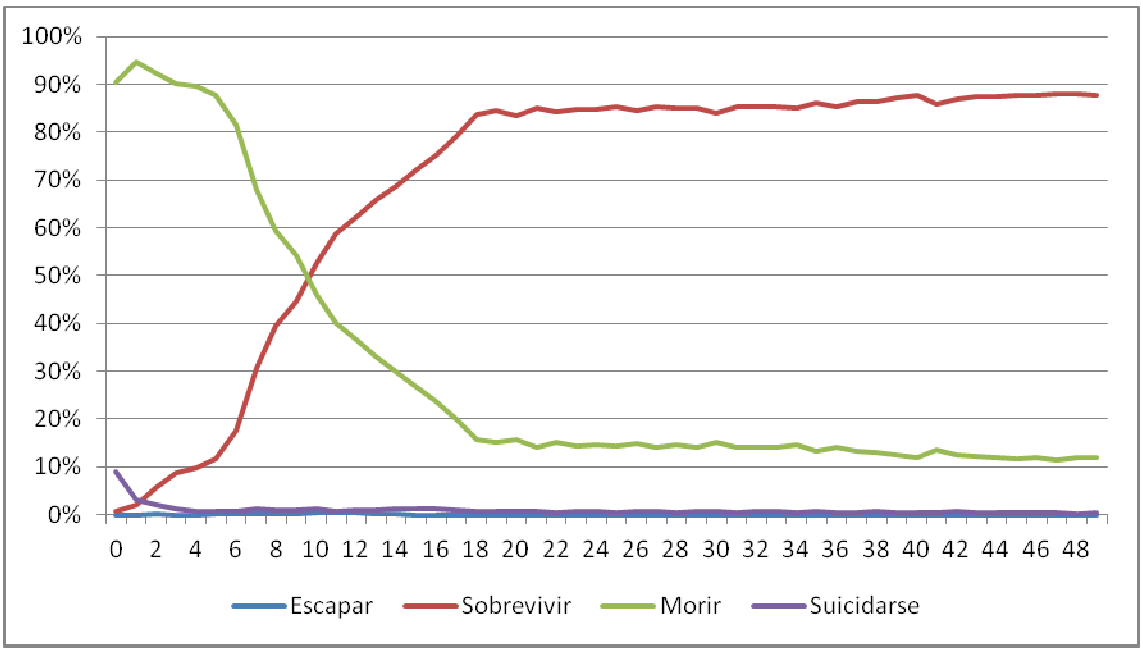


Figura 25. Evolución de las tasas promedio de estado final de superviviente del escenario básico con 1 superviviente

6.2.2 Escape

El objetivo de este experimento fue buscar comportamientos en un único superviviente para hallar la salida del mapa sin que importaran las acciones ofensivas pues no hay zombis ni, por tanto, tiene relevancia alguna la vida restante o el número de zombis asesinados.

El mejor comportamiento evolucionado (Figura 26) sencillamente se dirige hacia la casilla límite más próxima (*GoTo NearestBoundary*).

De esta forma, el superviviente sobrevive el 100% de las veces (Figura 27), quedándose muy cerca de la salida con un *score* de distancia medio poblacional por debajo de 0'2 (Figura 29), pero sin llegar nunca a pasar por ella.

Dada la simplicidad de la mejor solución, ésta se encuentra prácticamente en las primeras generaciones y la población entera apenas tarda unas 5 generaciones en converger (Figura 28). La tasa media de suicidios de la población (Figura 30) en este experimento es más alta que en otros dado que no hay zombis que se coman a los supervivientes rezagados y/o extremadamente cautos; comienza en torno al 75% y, aunque disminuye, sigue existiendo en cierta medida a lo largo de las generaciones posiblemente por el influjo del operador de mutación, que no deja de introducir pequeñas cantidades de material genético nuevo en los individuos.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(GoTo NearestBoundary)

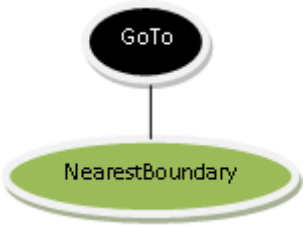


Figura 26. Árbol simplificado del escenario de escape con 1 superviviente

   b. ***Fitness* promedio y desviación típica** = 1’0922 (0’0687)
   c. **Tasas promedio de estado final de cada superviviente**

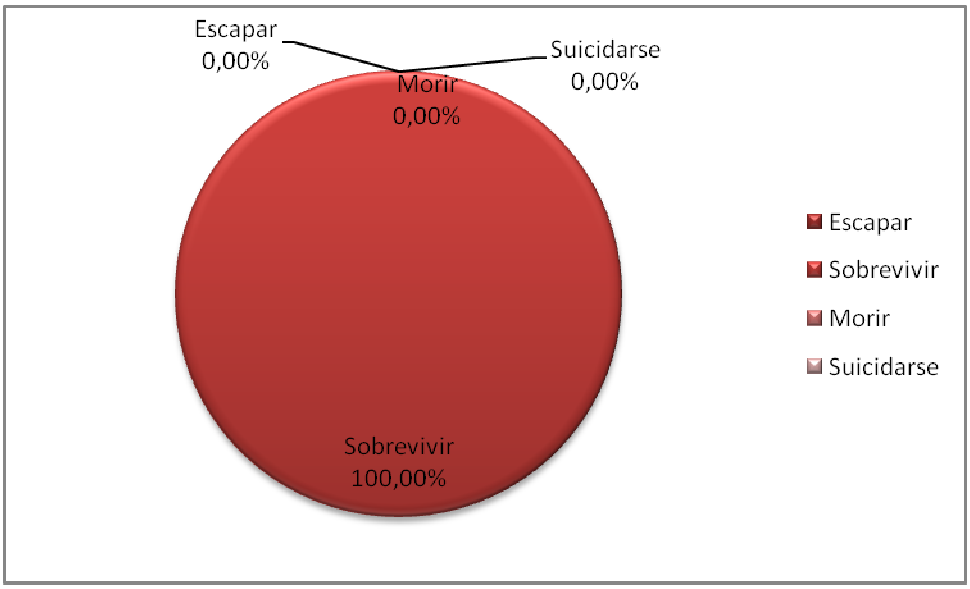


Figura 27. Tasas promedio de estado final del mejor individuo del escenario de escape con 1 superviviente

2. **Evolución por generación del fitness de los individuos**

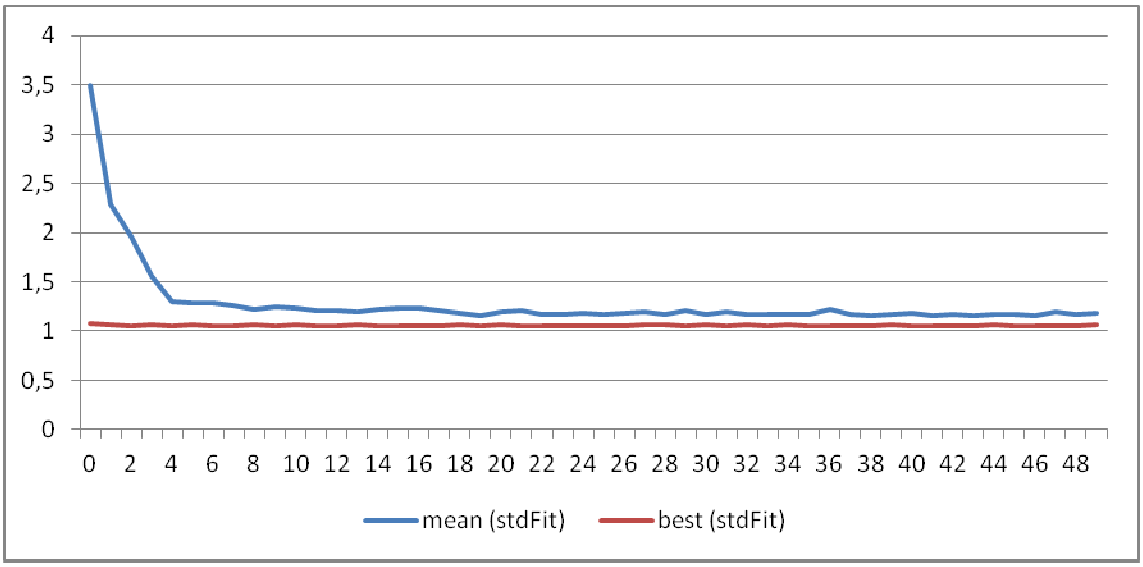


Figura 28. Evolución del *fitness* promedio y mejor del escenario de escape con 1 superviviente

3. **Evolución por generación de los *scores* promedio de cada superviviente**

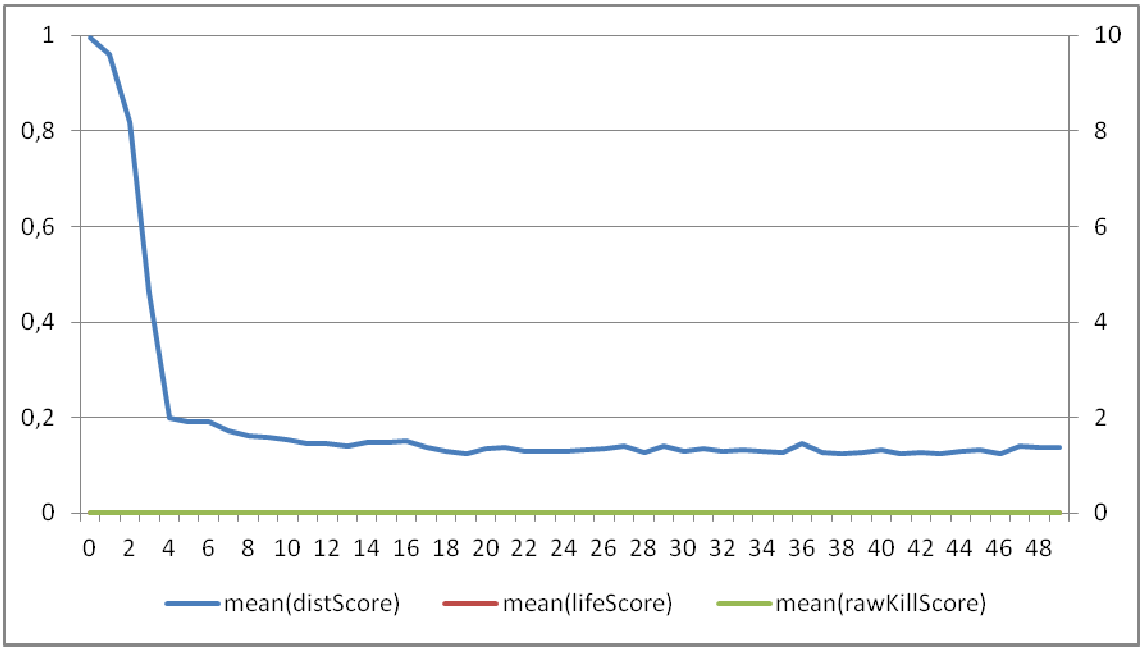


Figura 29. Evolución de los *scores* promedio del escenario de escape con 1 superviviente

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

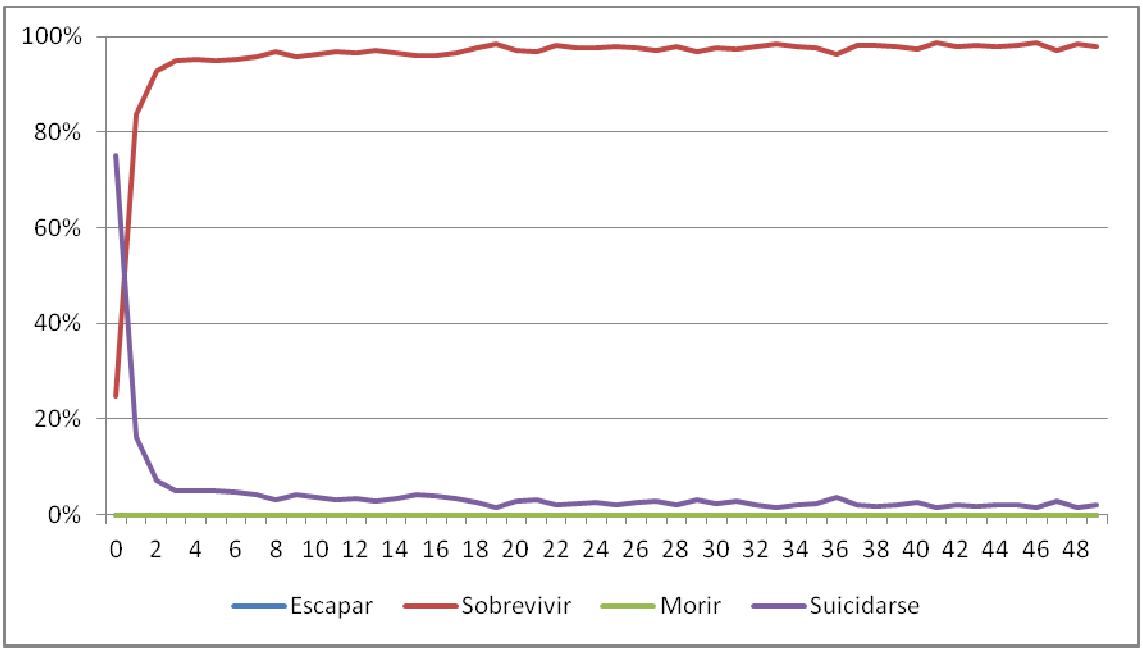


Figura 30. Evolución de las tasas promedio de estado final de superviviente del escenario de escape con 1 superviviente

6.2.3 Matanza

Con este experimento se buscaba encontrar comportamientos netamente ofensivos para que un superviviente individual fuera capaz de acabar con la mayor cantidad posible de zombis.

El aspecto del mejor individuo encontrado (Figura 31) es muy similar al del apartado 6.2.1 alternando levemente el orden las acciones. Si hay zombis a distancia de ataque (*EnemiesInAttackRange/EnemiesInSight*) primero ataca al más cercano de ellos (*Attack (AgentLocation NearestEnemy)/ Attack (AgentLocation NearestEnemyInAttackRange)*) y luego mueve aleatoriamente (*Avoid RandBoundary*). Si no tiene a ningún zombi cercano mira en otra dirección para asegurarse que no se le está aproximando uno por otro lado.

Con este comportamiento el superviviente consigue sobrevivir la mayor parte de las veces (93’5%), muriendo el resto (Figura 32).

Dado que no tienen relevancia los *scores* de distancia, estos prácticamente no bajan del valor inicial, próximo a 1 (Figura 34), que se traduce en no haberse acercado nada hacia la salida. Tras apenas 6 generaciones la mayor parte de la población aprende a atacar (Figura 34) y, en consecuencia, sobrevivir (Figura 35), y tras otras pocas generaciones más, en torno a las 12, se encuentran los mejores individuos y la población comienza lentamente a estabilizarse (Figura 33). En el promedio poblacional los supervivientes de este escenario destacan por tener una mayor cantidad de zombis asesinados próxima a 4 (Figura 34).

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(IfElse EnemiesInAttackRange (IfElse (Not EnemiesInSight) (Do (Attack (AgentLocation NearestEnemy)) (Avoid RandBoundary)) (Do (Attack (AgentLocation NearestEnemyInAttackRange)) (Avoid RandBoundary))) LookLeft)

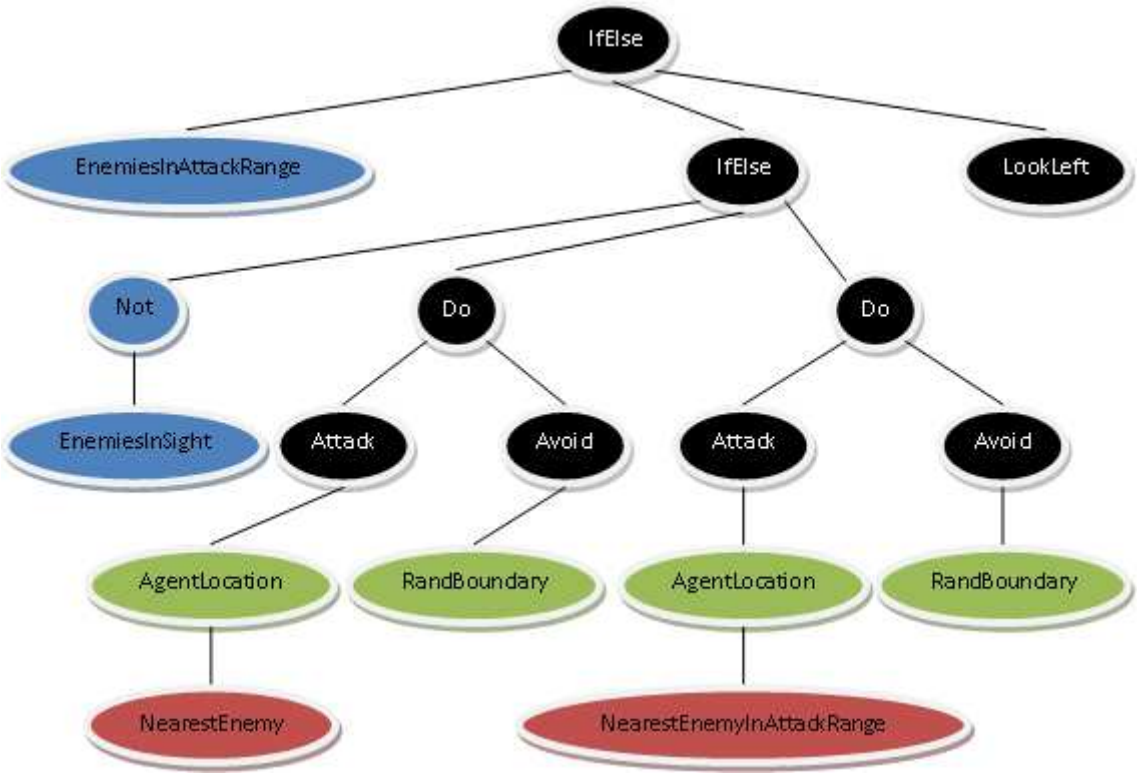


Figura 31. Árbol simplificado del escenario de matanza con 1 superviviente

   b. ***Fitness* promedio y desviación típica** = 1’6966 (0’2851)

c. **Tasas promedio de estado final de cada superviviente**

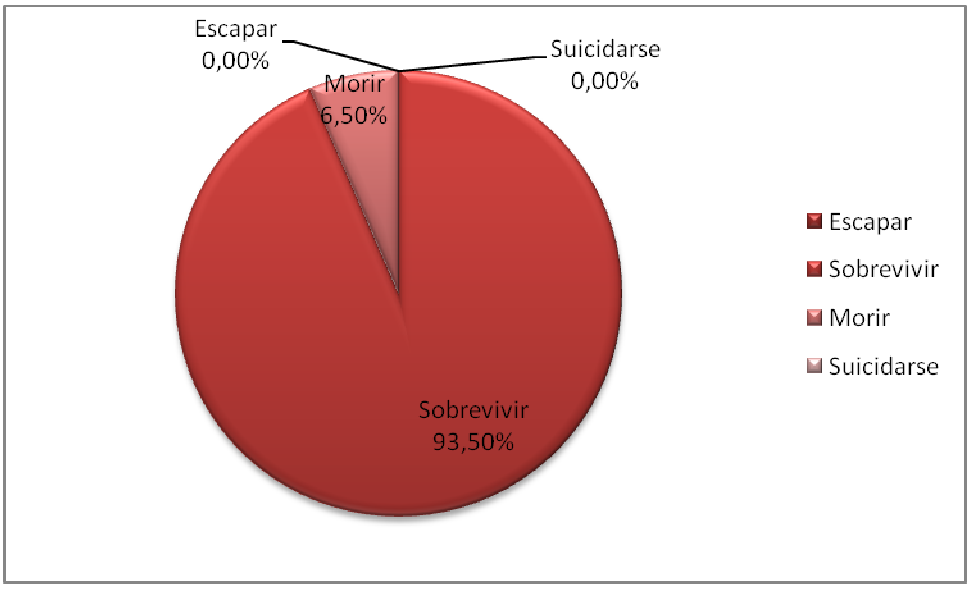


Figura 32. Tasas promedio de estado final del mejor individuo del escenario de matanza con 1 superviviente

2. **Evolución por generación del fitness de los individuos**

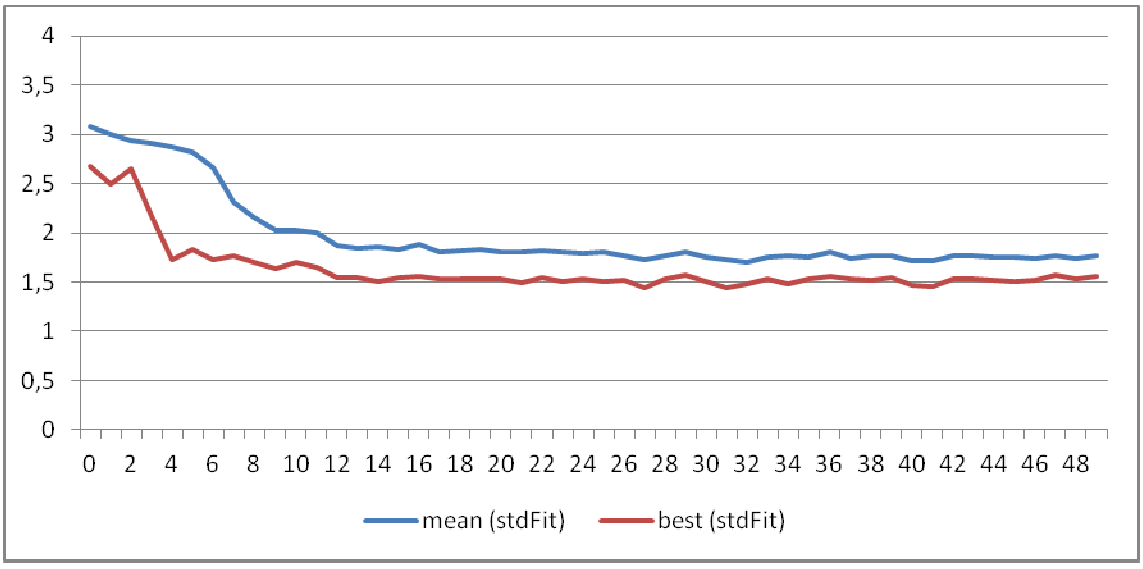


Figura 33. Evolución del *fitness* promedio y mejor del escenario de matanza con 1 superviviente

3. **Evolución por generación de los *scores* promedio de cada superviviente**

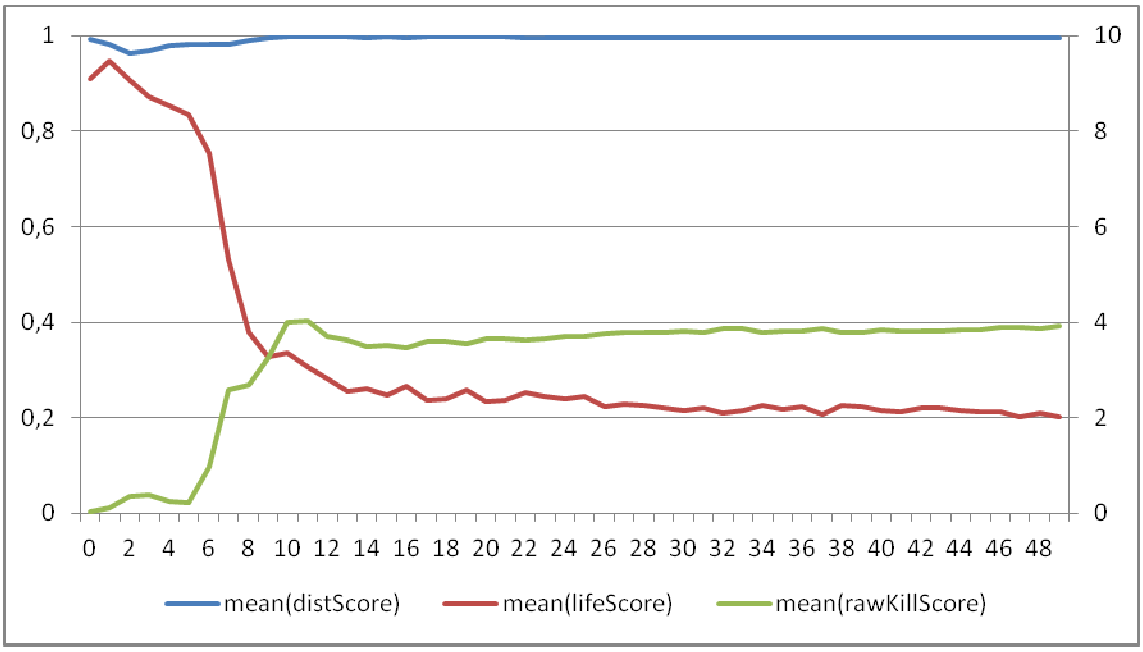


Figura 34. Evolución de los *scores* promedio del escenario de matanza con 1 superviviente

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

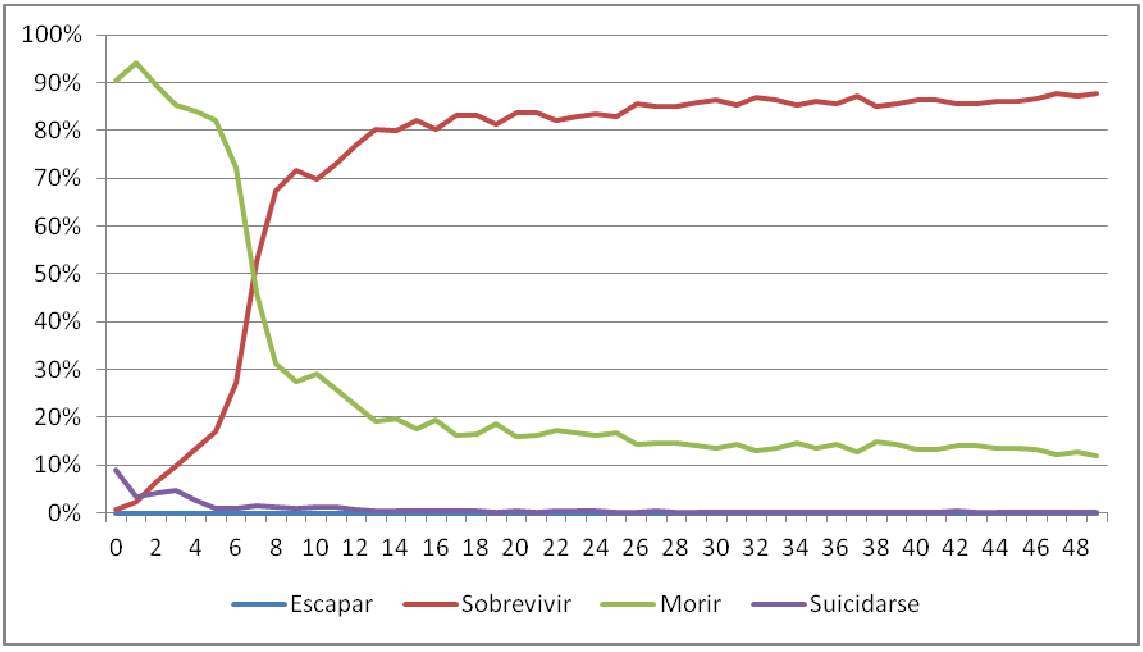


Figura 35. Evolución de las tasas promedio de estado final de superviviente del escenario de matanza con 1 superviviente

6.2.4 Supervivencia

En este caso se limitaron las opciones ofensivas de los supervivientes individuales al no darles ningún tipo de munición y aumentar la importancia de la vida restante para forzar la búsqueda de comportamientos de tipo defensivo.

El mejor individuo encontrado (Figura 36) se asemeja mucho a la parte defensiva de los correspondientes mejores árboles encontrados en los anteriores experimentos, alternando arbitrariamente acciones de movimiento aleatorio (*Avoid RandBoundary*) con variaciones de la dirección de visión (*LookLeft*), pero sin llegar a acciones más complicadas e interesantes como podría ser alejarse explícitamente de la posición del enemigo más próximo o recoger botiquines.

Sin poder atacar a los zombis, el comportamiento evolucionado permite al superviviente sobrevivir en un 18% de las ocasiones, frente a un 82% en el que finalmente es devorado (Figura 37).

El *score* de muertes queda fijado en 0 (Figura 39), lo cual tiene sentido porque los supervivientes no tienen munición. Pasadas las primeras 5 generaciones apenas se producen mejoras en la población o en el mejor individuo (Figura 38). El efecto de *bloating* es especialmente apreciable en este experimento puesto que, una vez los individuos aprenden a moverse mínimamente para evitar cometer suicidio (Figura 40) y descartan mejorar los *scores* de distancia (Figura 44) dado que no tienen repercusión directa (ni indirecta apreciable) en el *fitness*, sucesivas formas equivalentes de árbol a la del mejor individuo encontrado se suceden provocando picos en los valores del mejor *fitness* de cada generación (Figura 38). Dicho de otro modo, los picos que ven en la sucesión del mejor *fitness* por generación se corresponden con evaluaciones afortunadas de formas redundantes equivalentes a solución final hallada (Figura 36), en cuya expresión original previa a la simplificación (ver nota al pie 64) observamos el efecto del *bloating*.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(Do (IfElse EnemiesInAttackRange (Do (Do (Avoid RandBoundary) (Avoid RandBoundary)) (Avoid RandBoundary)) LookLeft) (Do LookLeft (Do (Do LookLeft (Avoid RandBoundary)) (Do LookLeft (Do (Avoid RandBoundary) (Do LookLeft (IfElse EnemiesInAttackRange (Do (Do (Avoid RandBoundary) (Do (Do LookLeft (Avoid RandBoundary)) (Do LookLeft (Do LookLeft (Avoid RandBoundary))))) (Do LookLeft (Avoid RandBoundary))) LookLeft)))))))

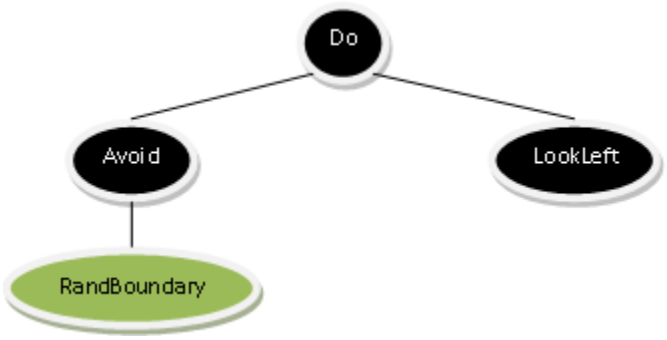


Figura 36. Árbol simplificado[64] del escenario de supervivencia con 1 superviviente

   b. ***Fitness* promedio y desviación típica** = 2'728 (0'5846)
   c. **Tasas promedio de estado final de cada superviviente**

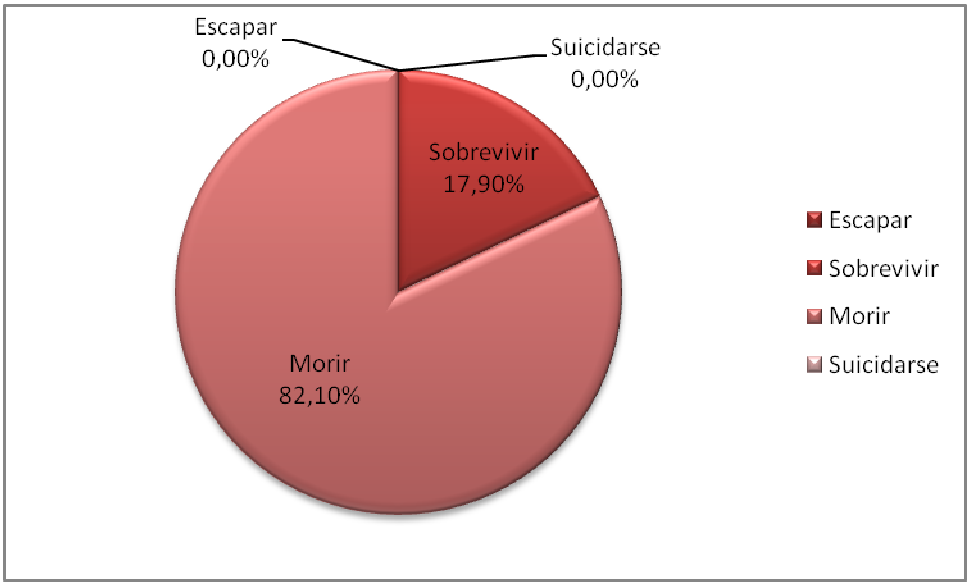


Figura 37. Tasas promedio de estado final del mejor individuo del escenario de supervivencia con 1 superviviente

[64] En este caso el árbol simplificado no se corresponde exactamente con el original, que es una secuencia mucho más intrincada de acciones *LookLeft* y *Avoid(RandBoundary)* anidadas con *Do* que llega a profundidad = 13

2. **Evolución por generación del *fitness* de los individuos**

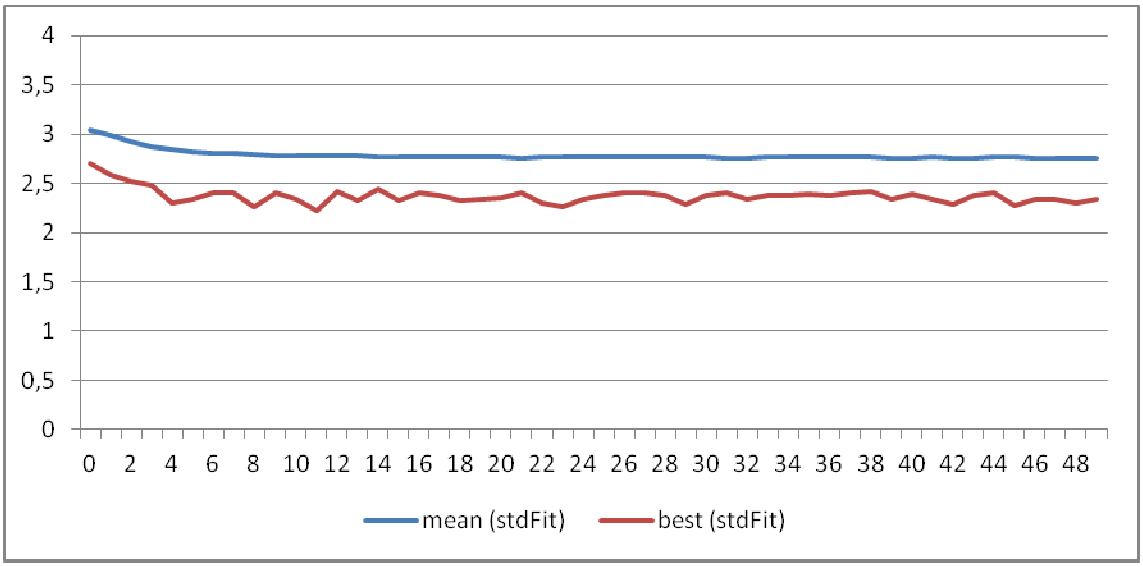


Figura 38. Evolución del *fitness* promedio y mejor del escenario de supervivencia con 1 superviviente

3. **Evolución por generación de los *scores* promedio de cada superviviente**

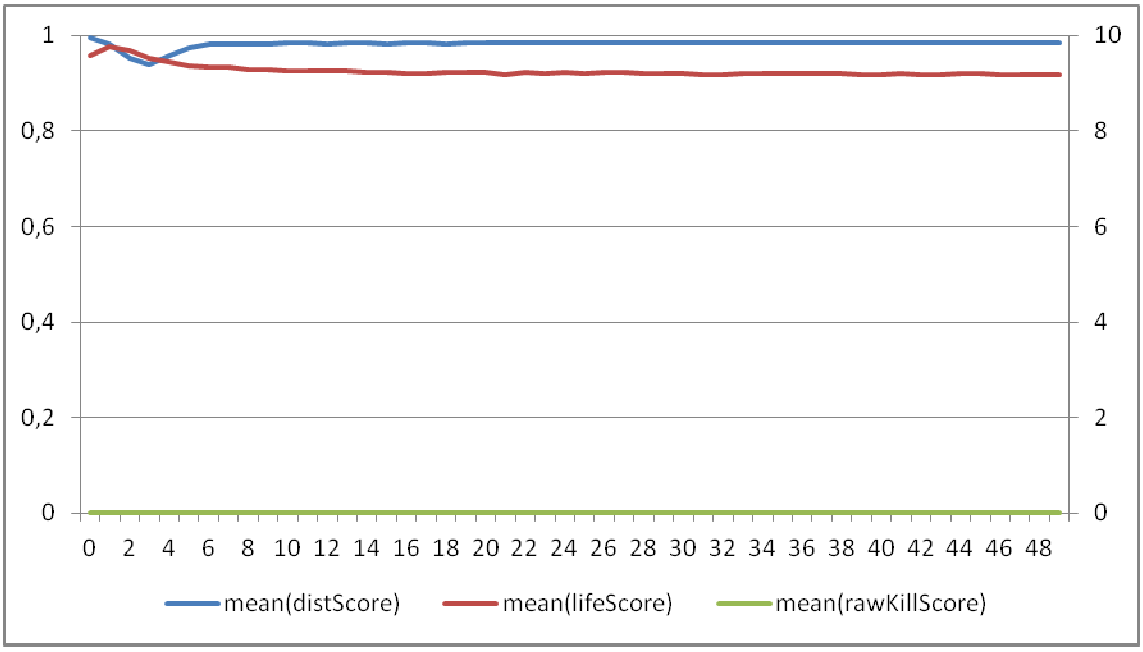


Figura 39. Evolución de los *scores* promedio del escenario de supervivencia con 1 superviviente

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

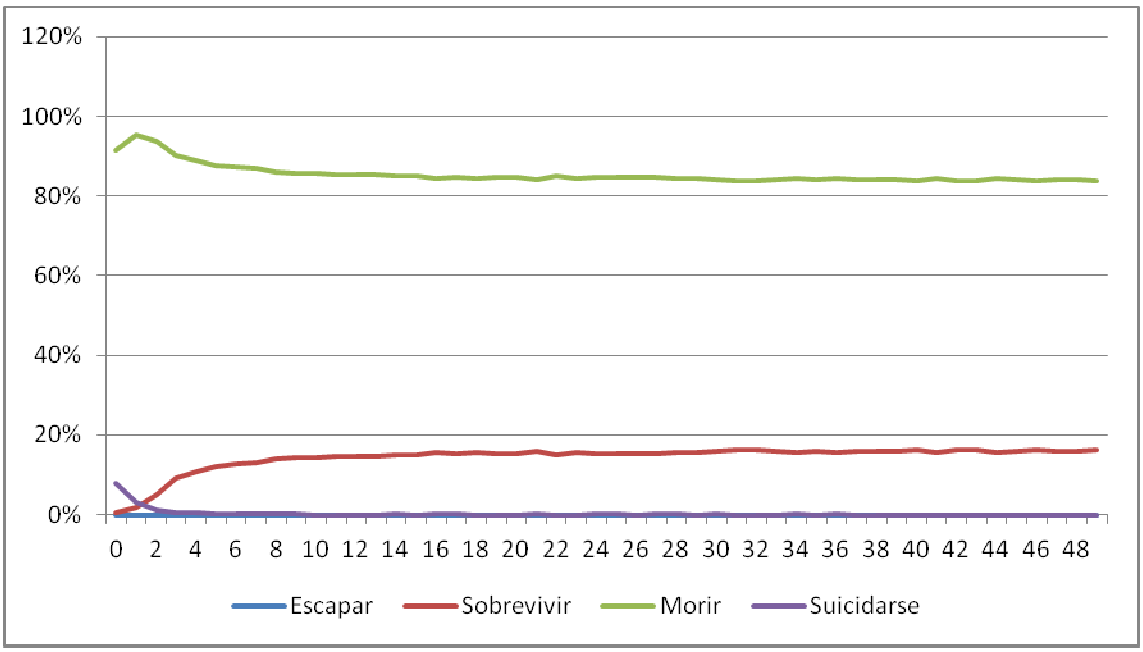


Figura 40. Evolución de las tasas promedio de estado final de superviviente del escenario de supervivencia con 1 superviviente

6.2.5 Combinado

Utilizando como funciones terminales de acción adicionales cada uno de los árboles derivados en los escenarios específicos de Escape, Matanza y Supervivencia para un superviviente, el objetivo de este experimento fue el mismo que para el escenario Básico, es decir, encontrar comportamientos de supervivientes individuales para escapar o, en su defecto, sobrevivir a los zombis.

El mejor individuo encontrado (Figura 41) hace uso adecuadamente de las funciones específicas desarrolladas para sobrevivir (*Survive*) y matar (*Kill*), optando por la estrategia ofensiva mientras tiene munición (*HaveIAmmo*) y pasando a la defensiva cuando ya no le queda.

El comportamiento evolucionado permite al superviviente acabar con vida en algo más del 93% de las ocasiones, muriendo en el resto (Figura 42).

La mejor solución se encuentra inmediatamente prácticamente en la 1ª generación, convergiendo rápidamente el resto de la población en escasas 3 generaciones (Figura 43). Los *scores* (Figura 44) a los que converge la población toman el buen valor (próximo a 4) de muertes que tenía el escenario de Matanza, y el de vida restante (0'2) de tanto Matanza como Básico, obteniendo un casi 90% de supervivencia de toda la población (Figura 45). En cambio, el *score* de distancia permanece en el peor valor, es decir, 1 (Figura 44), no siendo capaz de ponerse a buscar o, ni tan siquiera, al menos, aproximarse a la salida, con el comportamiento

del escenario de Escape, tal vez porque hay demasiado zombi en el entorno como para poder despreocuparse de ellos.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(IfElse (Not HaveIAmmo) Survive Kill)

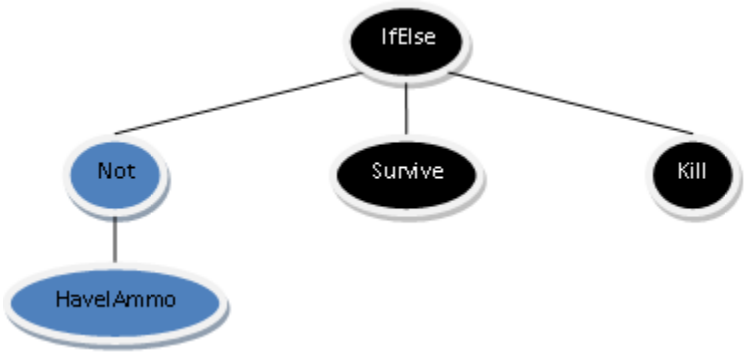


Figura 41. Árbol simplificado del escenario combinado con 1 superviviente

   b. ***Fitness* promedio y desviación típica** = 1'7795 (0'327)
   c. **Tasas promedio de estado final de cada superviviente**

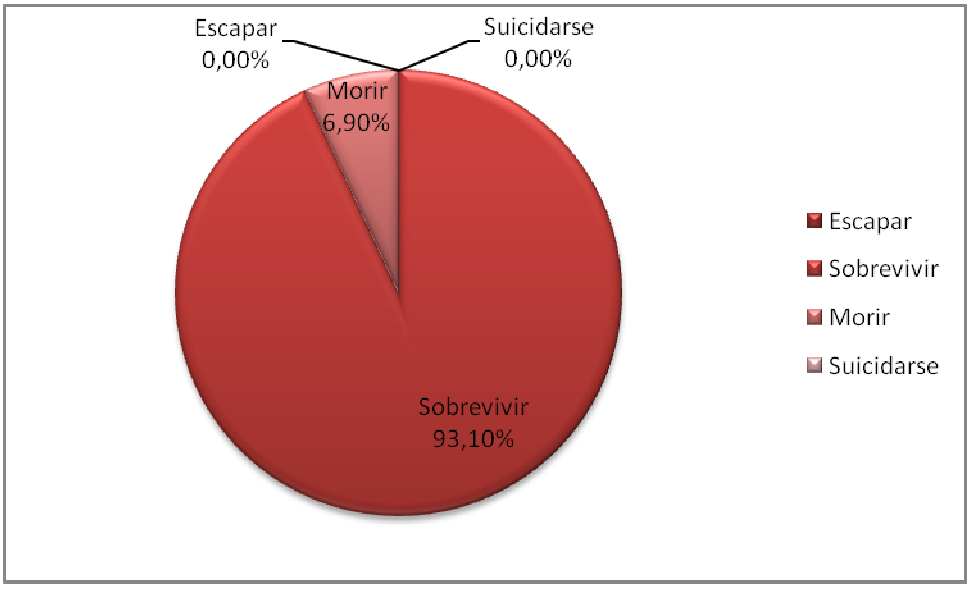


Figura 42. Tasas promedio de estado final del mejor individuo del escenario combinado con 1 superviviente

2. **Evolución por generación del *fitness* de los individuos**

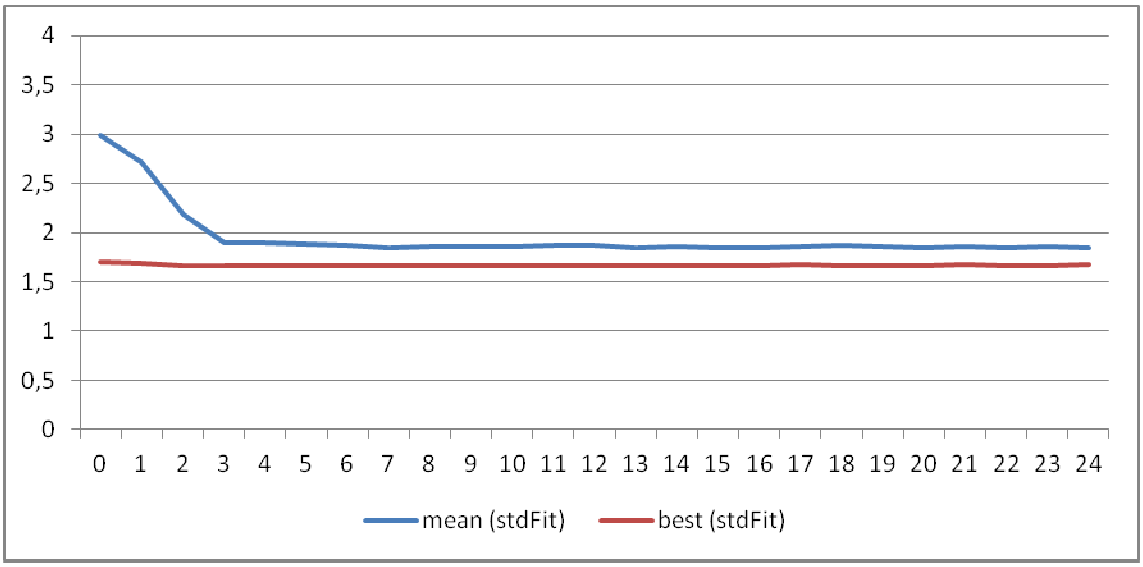


Figura 43. Evolución del *fitness* promedio y mejor del escenario combinado con 1 superviviente

3. **Evolución por generación de los *scores* promedio de cada superviviente**

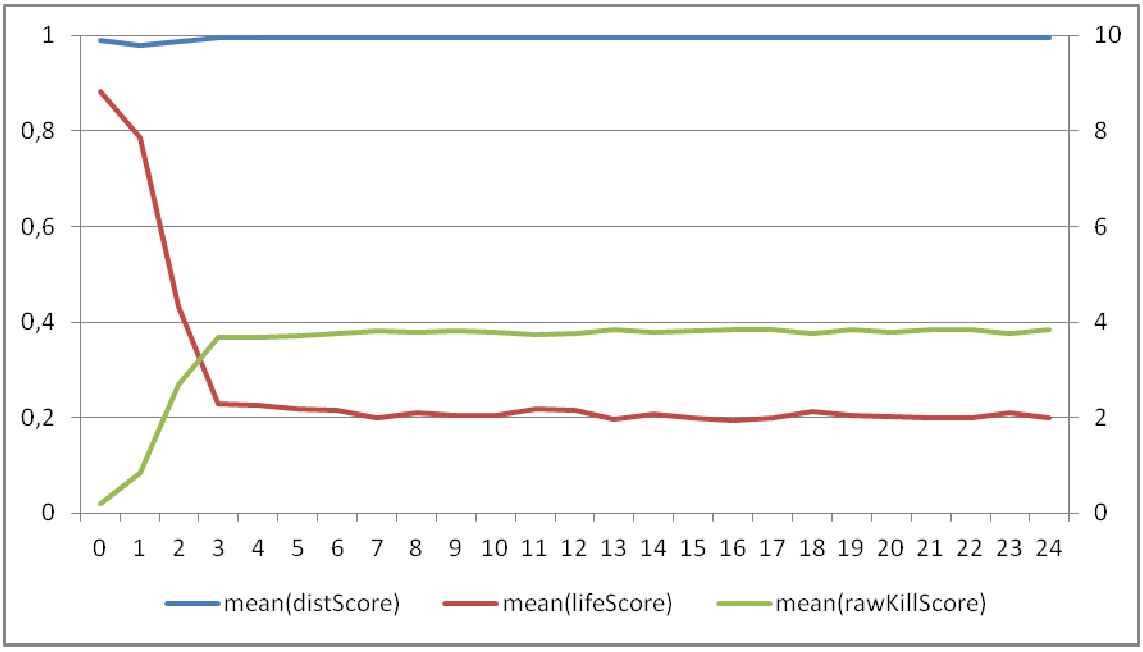


Figura 44. Evolución de los *scores* promedio del escenario combinado con 1 superviviente

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

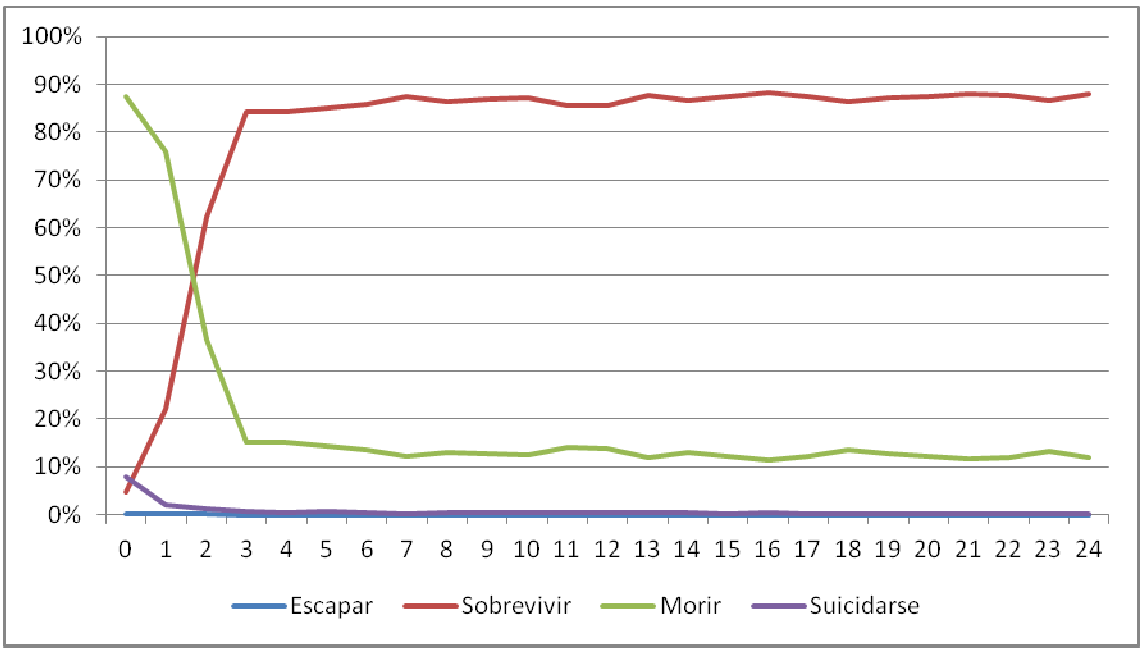


Figura 45. Evolución de las tasas promedio de estado final de superviviente del escenario combinado con 1 superviviente

6.3 Experimento 2: Equipos de 2 supervivientes

6.3.1 Básico

Este escenario repite los objetivos principales del trabajo, la búsqueda de comportamientos para escapar o, si no es posible, sobrevivir, pero, en este caso, evolucionando un equipo de 2 supervivientes.

El mejor equipo encontrado (Figura 46) no aprende correctamente a atacar a los enemigos, desarrollando un comportamiento común curioso (*Attack NearestBoundary*) en el que avanzan un poco por el mapa en dirección hacia sus casillas límite conocidas disparando hacia las mismas, creando con ello una posible pequeña maniobra de distracción. Uno de los miembros del equipo sólo lleva a cabo dicho comportamiento en el que momento que está herido (*AmIHurt*), dedicándose a moverse aleatoriamente (*Avoid FarthestBoundary*) hasta que se cumple dicha condición.

El superviviente de comportamiento más sencillo llega a sobrevivir en casi un 30% de las ocasiones, muriendo en el 70% restante. El otro superviviente sobrevive un poco menos del 25%, pero dado que tiene cierta capacidad de movimiento aleatorio, en unas pocas ocasiones, algo menos del 1%, consiguió escapar casualmente (Figura 47).

Las mejores soluciones no se obtienen hasta pasadas 30 generaciones (Figura 48), si bien los árboles redundantes por el efecto del *bloating* hacen que se sigan produciendo picos casuales

en cuanto al mejor generacional de manera similar a lo que ocurría en el experimento de Supervivencia de 1 superviviente (Figura 38). En el presente experimento, una vez transcurridas las mencionadas 30 primeras generaciones, sigue observándose una mejora en el *fitness* (Figura 48) y en los *scores* de muertes y vida restante (Figura 49) hasta la generación 50. Tanto las tasas de resultado final de superviviente (Figura 47) como los *scores* de la población (Figura 49) manifiestan la aparición del mejor individuo hacia la generación 30 modificando ligeramente sus valores. A partir de ese punto los individuos parecen preferir estrategias más defensivas, incrementando los *scores* de vida restante, en detrimento de la búsqueda de salida, cuyo *score* asociado de distancia acaba pasando del mínimo de 0'8 alcanzado sobre la generación 10 hasta un levemente peor 0'85. Los *scores* de muertes (Figura 49) en ningún momento llegan a 1, pero no dejan de crecer suavemente a lo largo de toda la evolución. En cualquier caso, toda tendencia apreciable en este experimento, con excepción de la rápida desaparición de los suicidios en las 3 primeras generaciones (Figura 50), es muy lenta. Estas dificultades para el aprendizaje tal vez encuentren explicación en la mayor complejidad del espacio de búsqueda con respecto al caso de un único superviviente, ya que estamos duplicando el tamaño del genoma de los individuos y combinando las opciones de un superviviente con las del otro.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(IfElse AmIHurt (Attack NearestBoundary) (IfElse AmIHurt (IfElse AmIHurt (IfElse AmIHurt (Attack NearestBoundary) (Do (Attack NearestBoundary) (Do (Avoid FarthestBoundary) (Avoid FarthestBoundary)))) (Avoid FarthestBoundary)) (Avoid FarthestBoundary)))

(IfElse AmIHurt (Attack NearestBoundary) (Avoid FarthestBoundary))

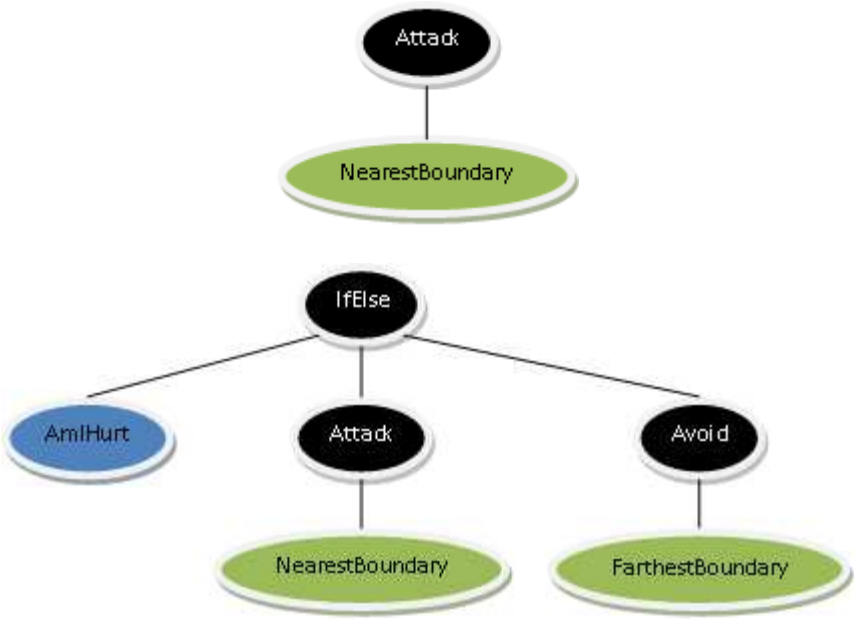


Figura 46. Árboles simplificados del escenario básico con 2 supervivientes

   b. ***Fitness* promedio y desviación típica** = 2'5853 (0'4472)

c. **Tasas promedio de estado final de cada superviviente**

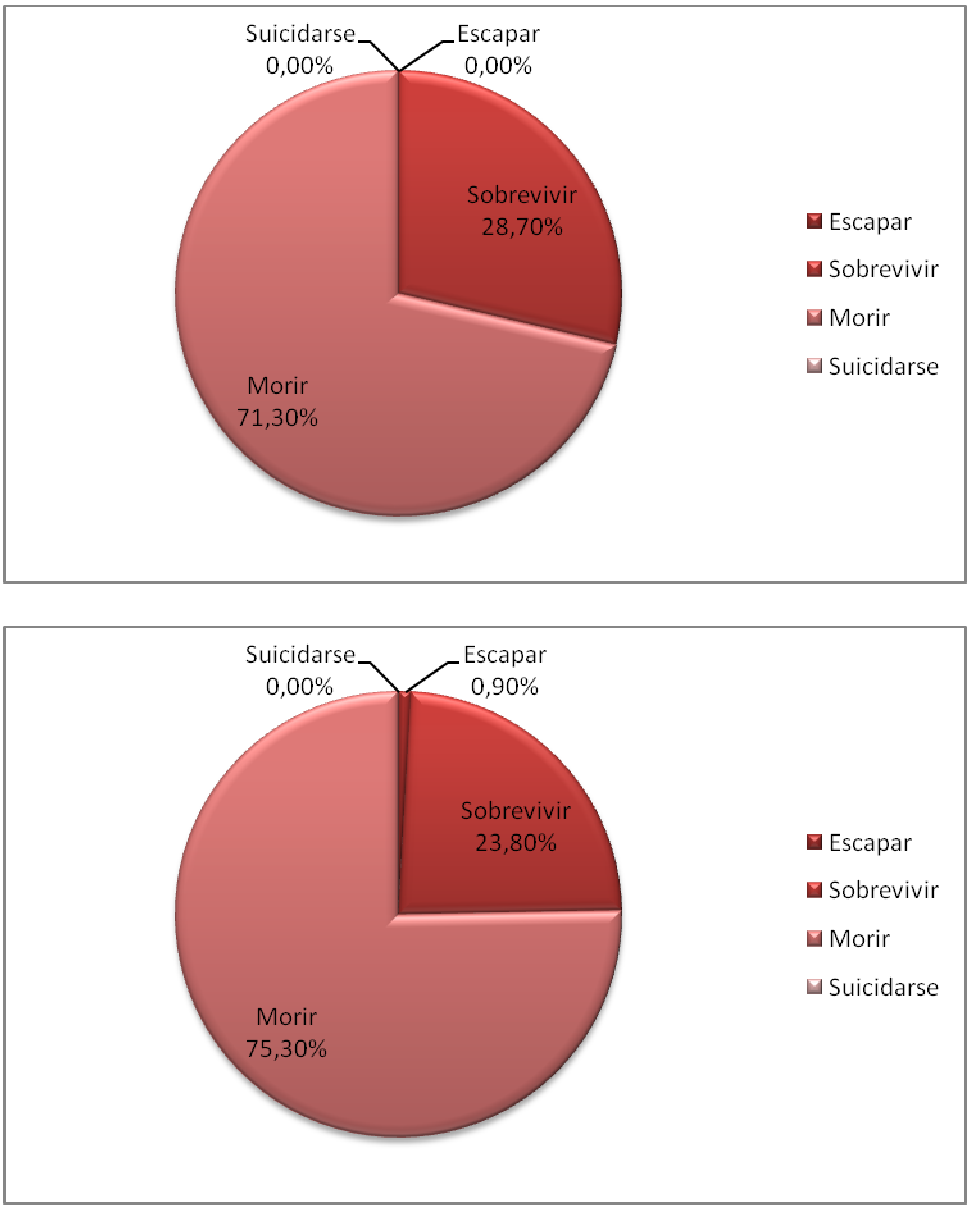


Figura 47. Tasas promedio de estado final del mejor individuo del escenario básico con 2 supervivientes

2. **Evolución por generación del *fitness* de los individuos**

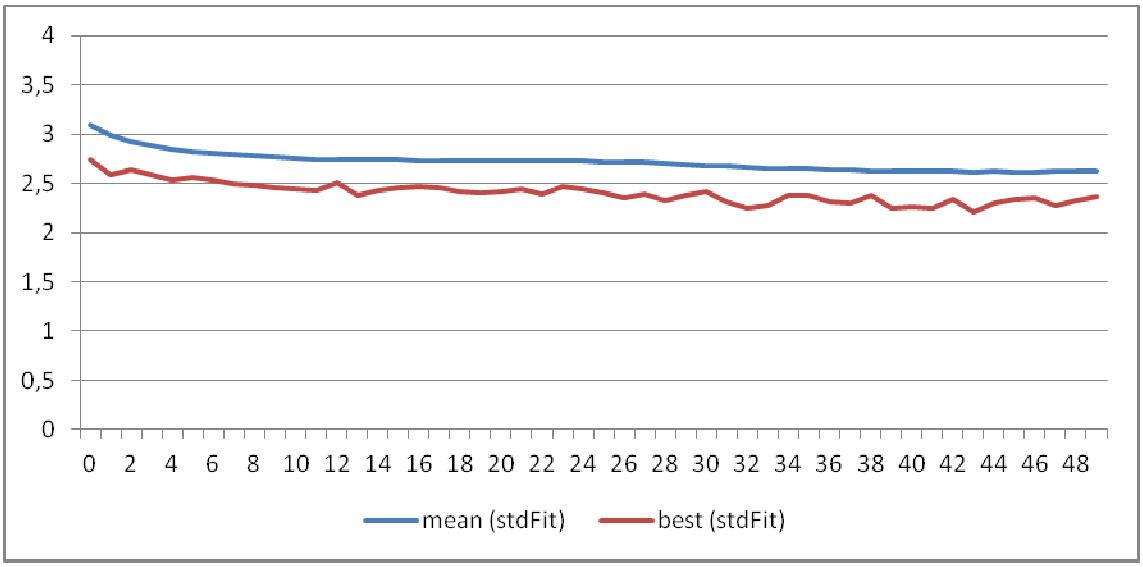


Figura 48. Evolución del *fitness* promedio y mejor del escenario básico con 2 supervivientes

3. **Evolución por generación de los *scores* promedio de cada superviviente**

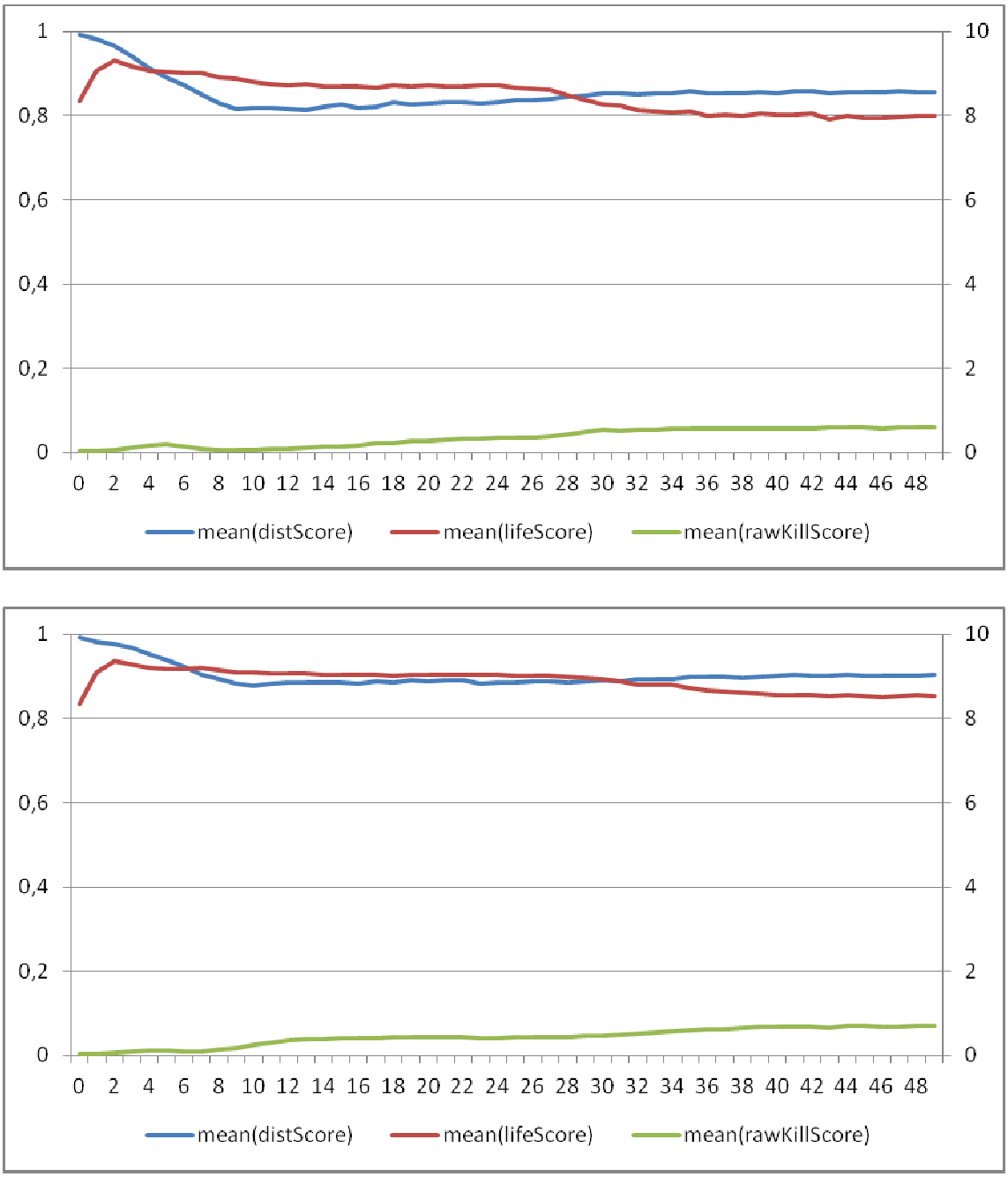


Figura 49. Evolución de los *scores* promedio del escenario básico con 2 supervivientes

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

Figura 50. Evolución de las tasas promedio de estado final de superviviente del escenario básico con 2 supervivientes

6.3.2 Escape

Este experimento perseguía encontrar comportamientos para un equipo de 2 supervivientes mediante los cuales pudieran encontrar la salida sin importar otros factores.

Dado que no ha emergido ninguna forma de cooperación, las soluciones obtenidas (Figura 51) son idénticas a las encontradas (*GoTo NearestBoundary*) en el experimento equivalente para 1 superviviente visto con anterioridad (Figura 26).

Mediante este continuo movimiento hacia las casillas frontera más próximas los supervivientes consiguen sobrevivir en prácticamente el 100% de los casos, dejando una pequeña cantidad de ocasiones en las que los supervivientes cayeron en la casilla de salida por casualidad (Figura 52).

Igual que para 1 único superviviente, la mejor solución se alcanza sobre las 5 generaciones (Figura 53), a partir de la cual la población desecha comportamientos que llevan al suicidio (Figura 55) y estabiliza sus *scores* de distancia (Figura 54), que son los únicos relevantes en este experimento.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(GoTo NearestBoundary)

(GoTo NearestBoundary)

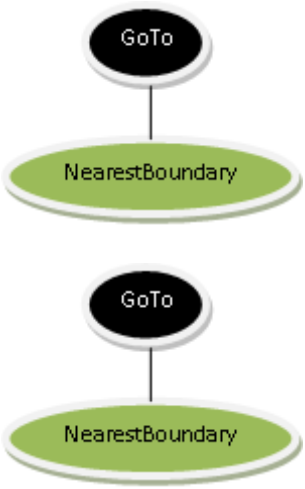


Figura 51. Árboles simplificados del escenario de escape con 2 supervivientes

   b. ***Fitness* promedio y desviación típica** = 1’4077 (0’407)

c. **Tasas promedio de estado final de cada superviviente**

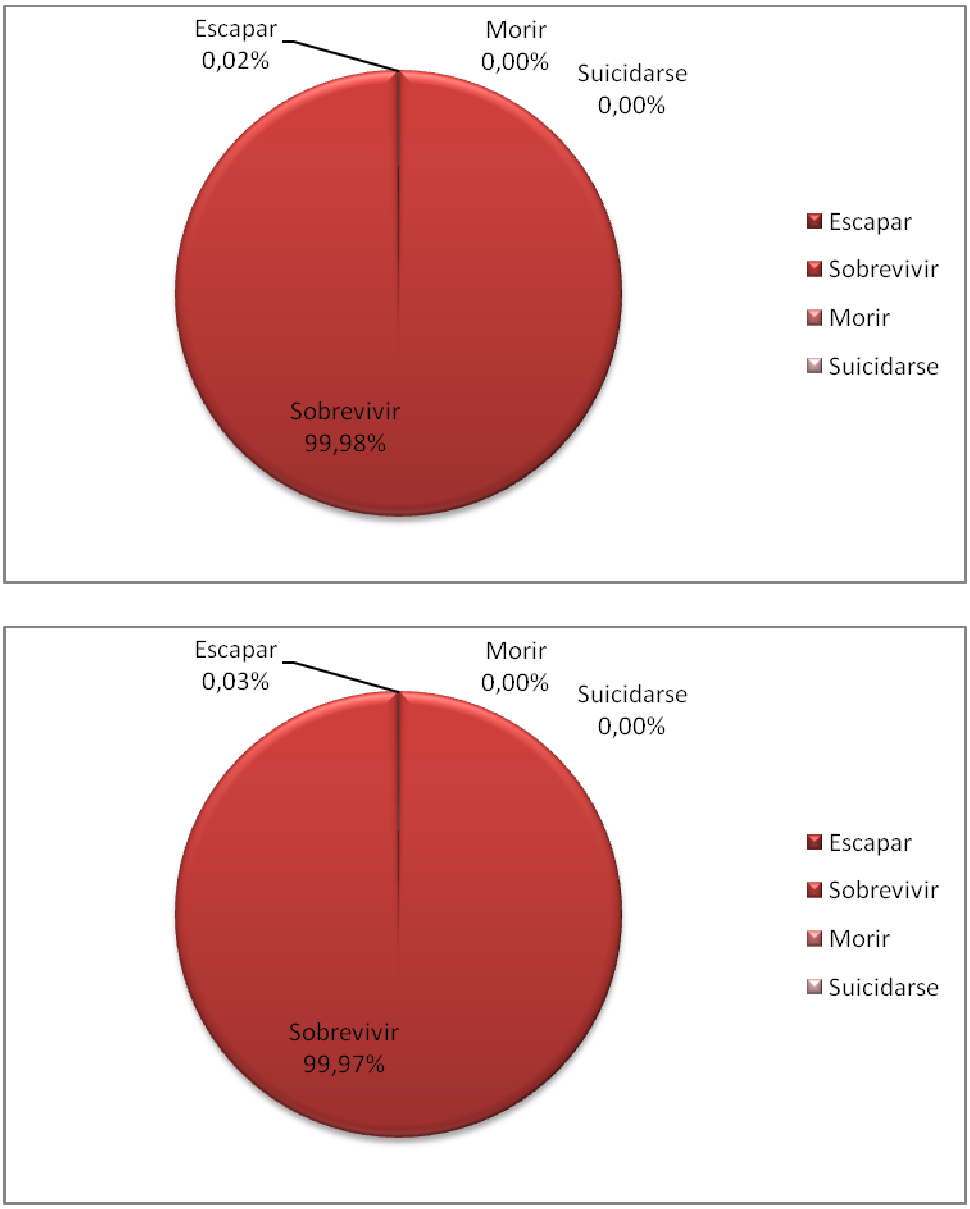


Figura 52. Tasas promedio de estado final del mejor individuo del escenario de escape con 2 supervivientes

2. **Evolución por generación del *fitness* de los individuos**

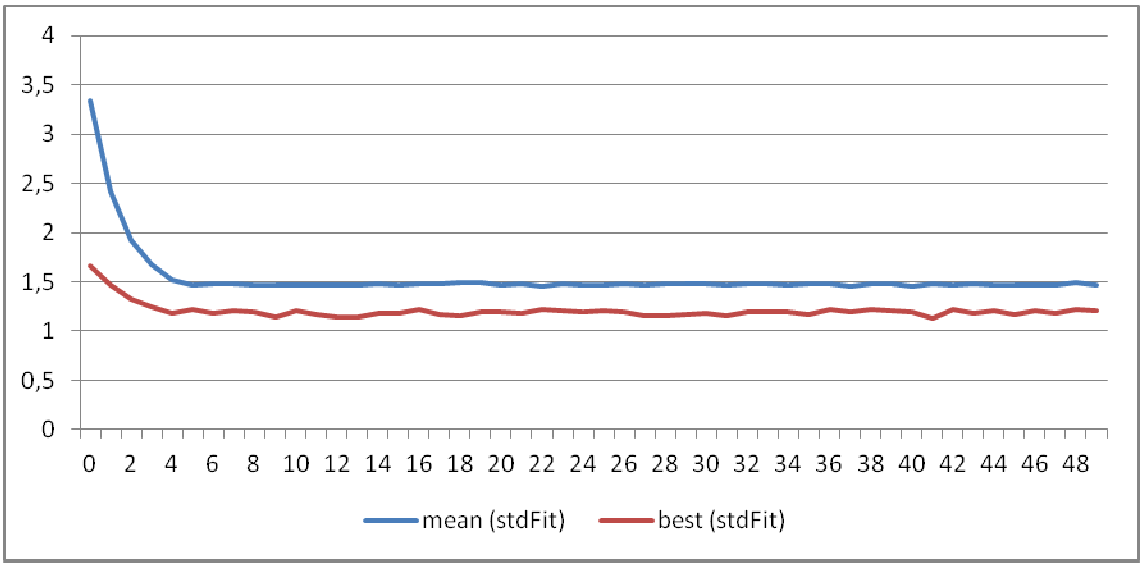


Figura 53. Evolución del *fitness* promedio y mejor del escenario de escape con 2 supervivientes

3. **Evolución por generación de los *scores* promedio de cada superviviente**

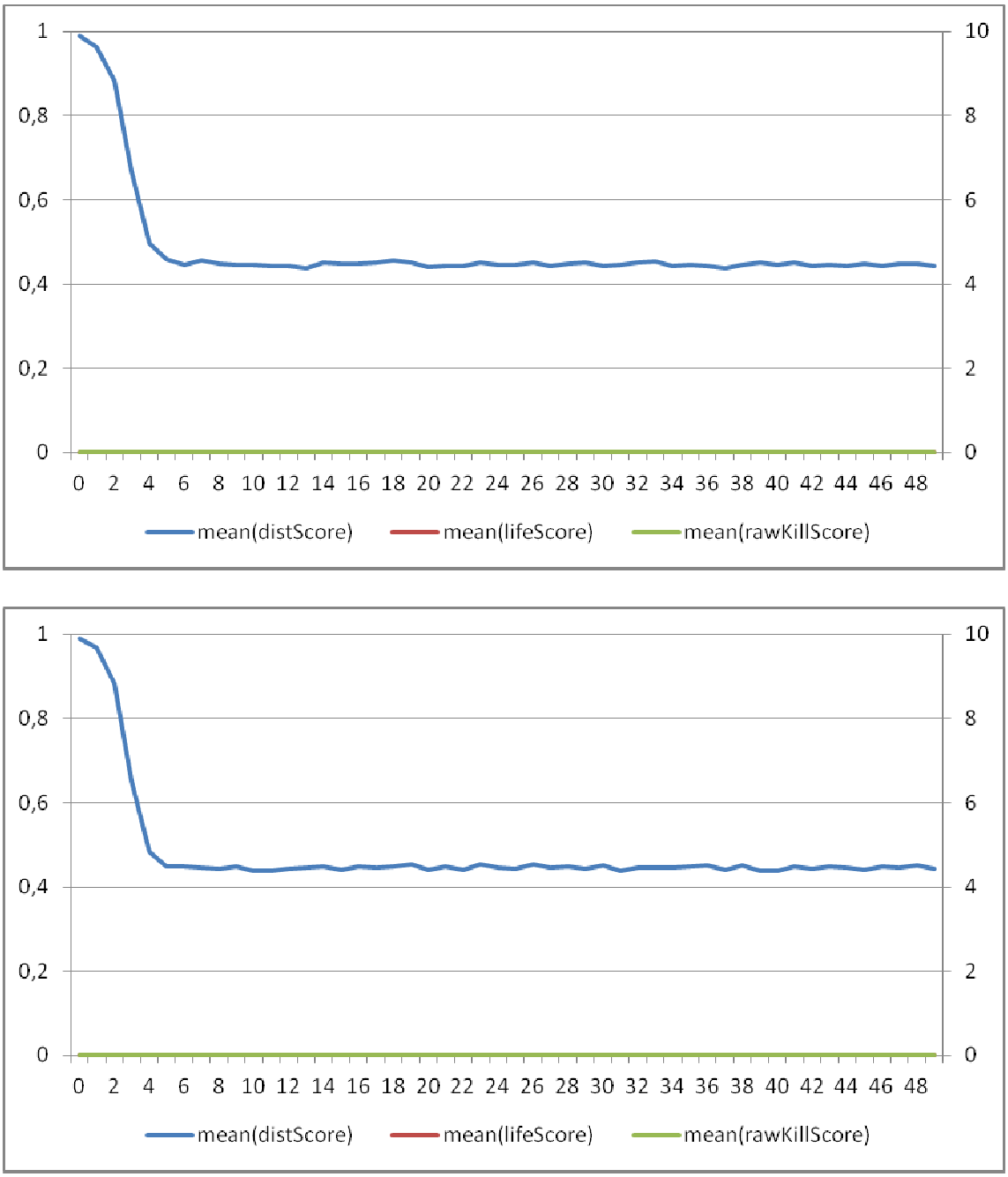


Figura 54. Evolución de los *scores* promedio del escenario de escape con 2 supervivientes

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

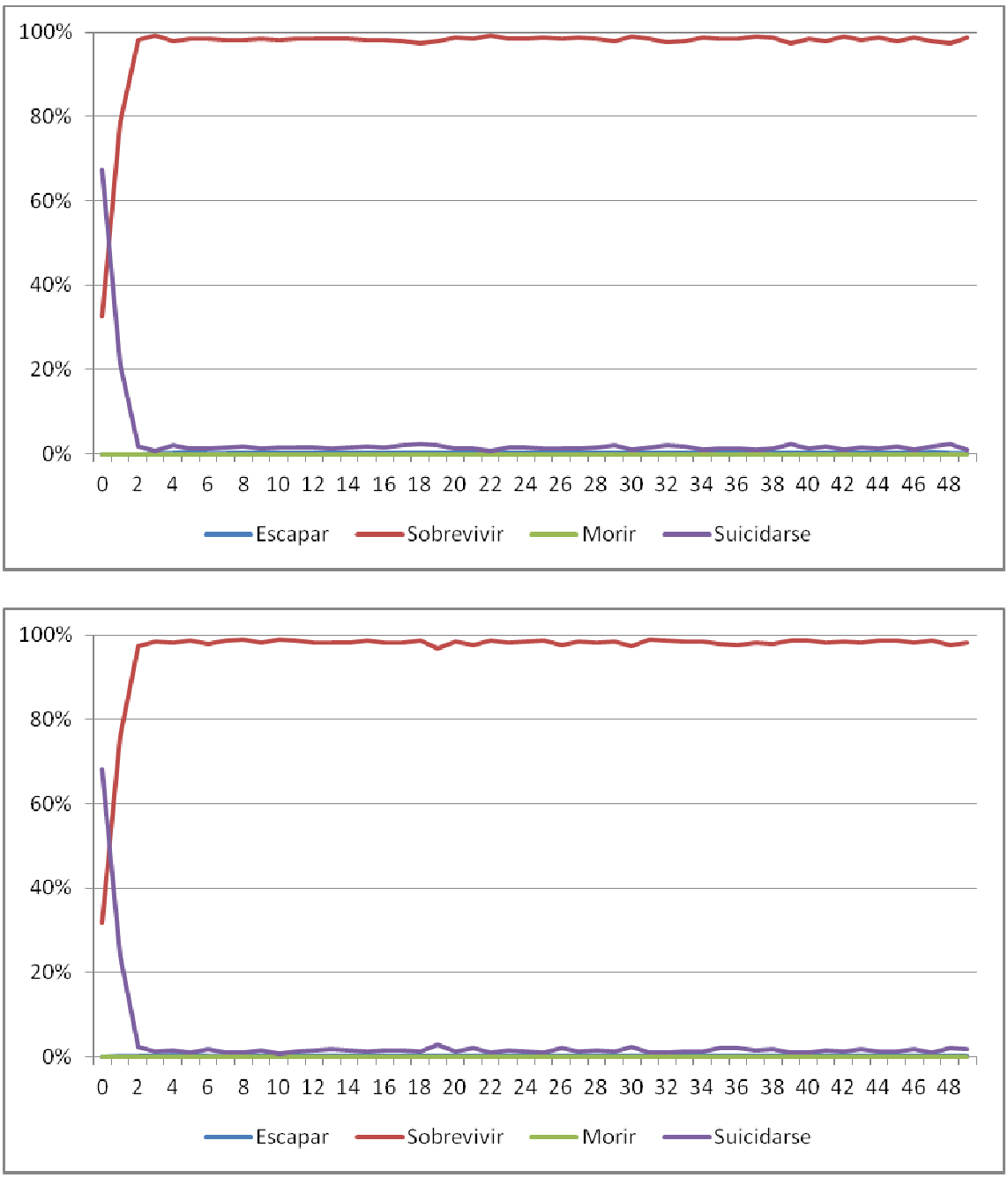


Figura 55. Evolución de las tasas promedio de estado final de superviviente del escenario de escape con 2 supervivientes

6.3.3 Matanza

El objetivo de este experimento fue evolucionar comportamientos en un equipo de 2 supervivientes para matar todos los zombis que fueran capaces.

Sorprendentemente, el mejor comportamiento evolucionado (Figura 56) únicamente presenta como acciones a ejecutar el ataque a la casilla límite más cercana (*Attack NearestBoundary*). De este modo, el equipo de supervivientes evolucionado apenas es capaz de asesinar unos pocos zombis, ya que estos únicamente pueden recibir ataques los turnos en los que se encuentran en dichas casillas límite más próximas a los supervivientes.

Este pobre comportamiento tiene como consecuencia una tasa de supervivencia que no llega al 16% en ninguno de los dos miembros del equipo. En el 84% restante los supervivientes mueren (Figura 57).

Los *scores* de distancia (Figura 59) prácticamente no descienden del peor valor, 1, lo cual es normal teniendo en cuenta que no se están teniendo en cuenta para el *fitness*. Sin embargo, la única evolución útil apreciable son las 6 primeras generaciones durante las cuales el equipo aprende a no suicidarse (Figura 60). A partir de ahí, los *scores* de muertes de la población se estancan en torno a 0’5 (Figura 59), y no se mejora prácticamente nada (Figura 58). Múltiples ejecuciones deberían ser analizadas para buscar una explicación a este mal resultado tan distinto del cosechado con supervivientes individuales.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(IfElse (IsBeingAttacked NearestBoundary) (Attack NearestBoundary) (Attack NearestBoundary)) (Attack NearestBoundary)

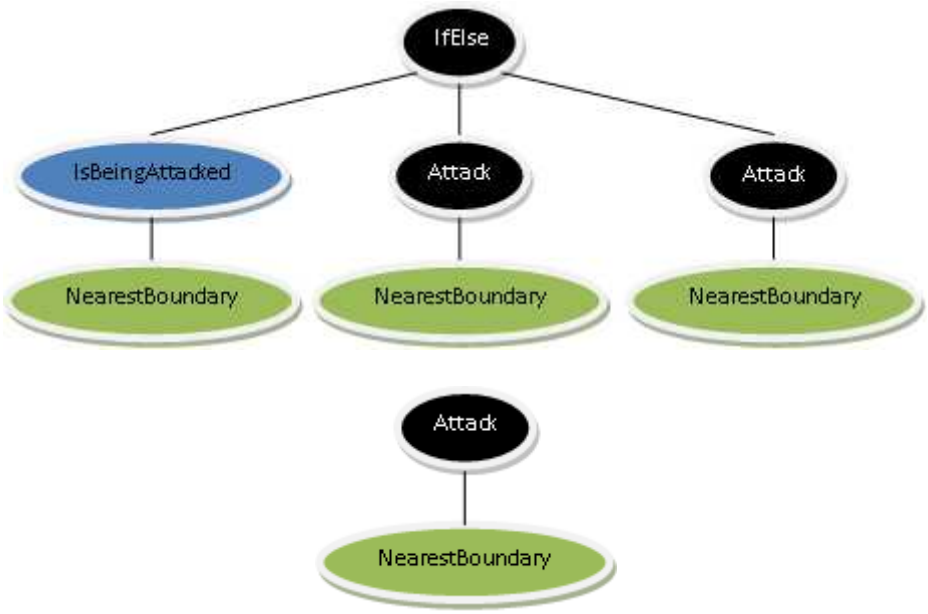


Figura 56. Árboles simplificados del escenario de matanza con 2 supervivientes

   b. ***Fitness* promedio y desviación típica** = 2’7631 (0’3685)

c. **Tasas promedio de estado final de cada superviviente**

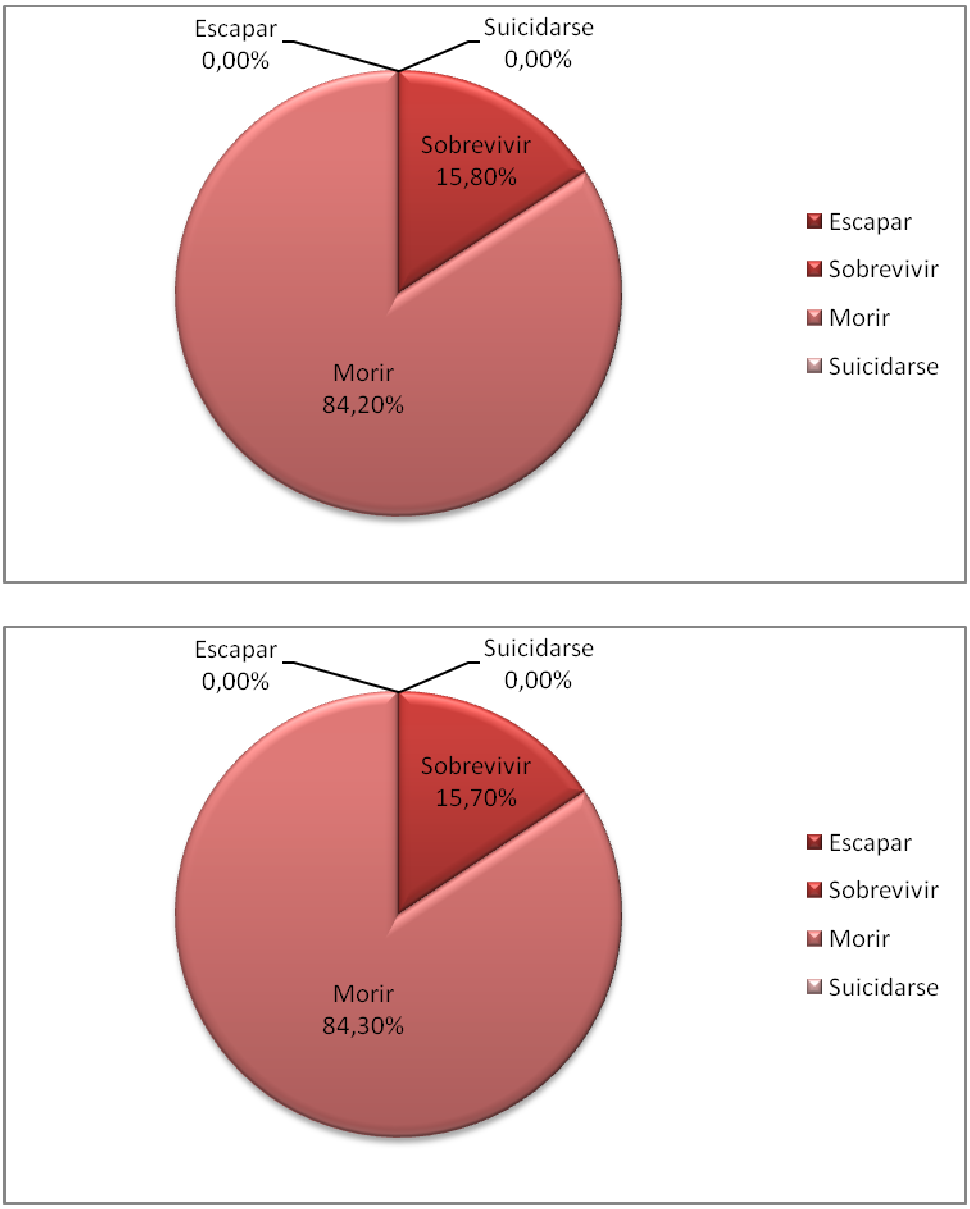


Figura 57. Tasas promedio de estado final del mejor individuo del escenario de matanza con 2 supervivientes

2. **Evolución por generación del *fitness* de los individuos**

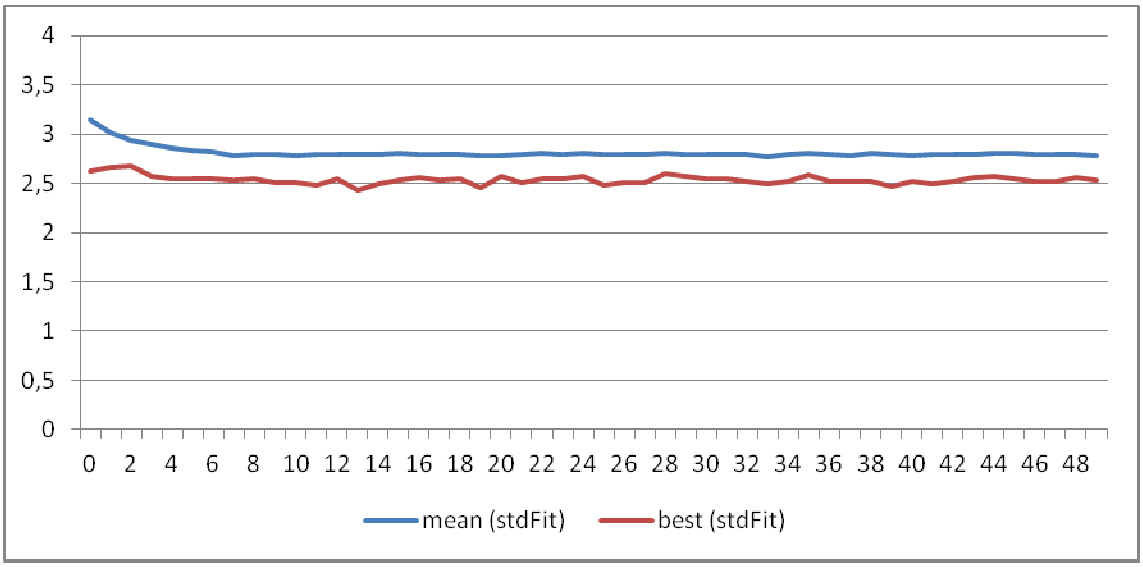
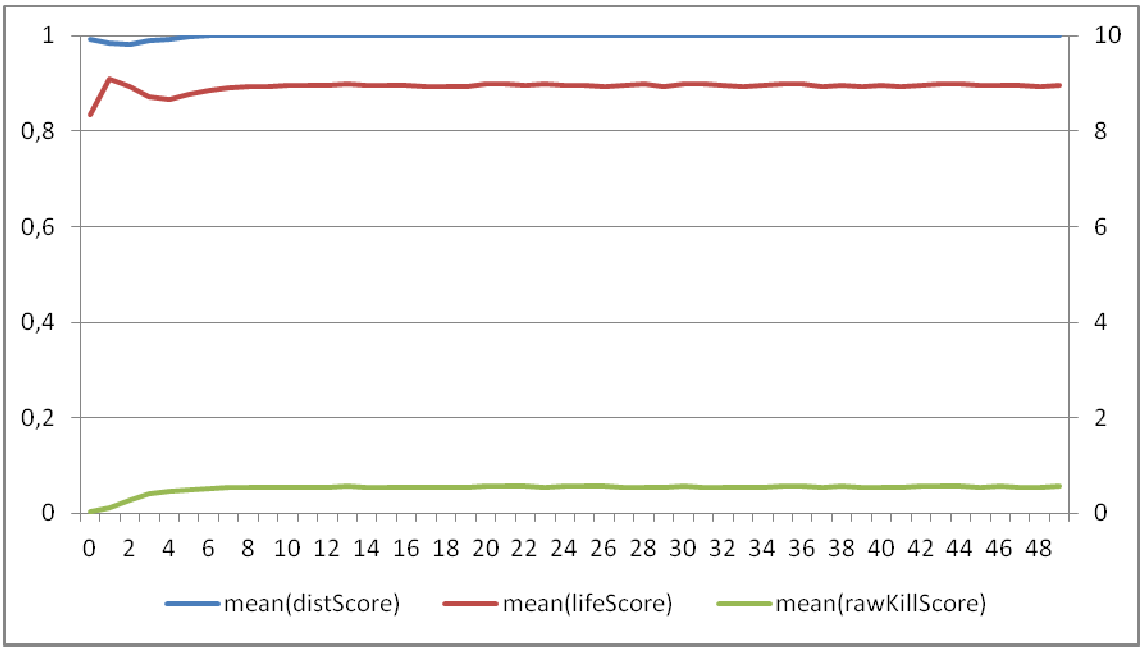


Figura 58. Evolución del *fitness* promedio y mejor del escenario de matanza con 2 supervivientes

3. **Evolución por generación de los *scores* promedio de cada superviviente**



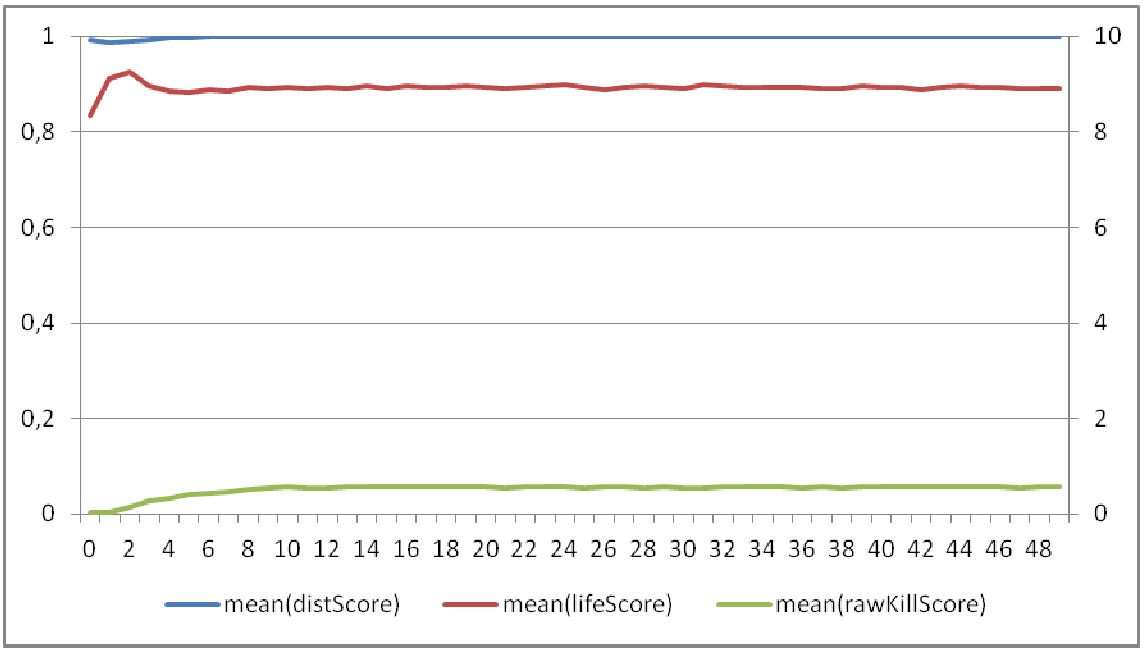


Figura 59. Evolución de los *scores* promedio del escenario de matanza con 2 supervivientes

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

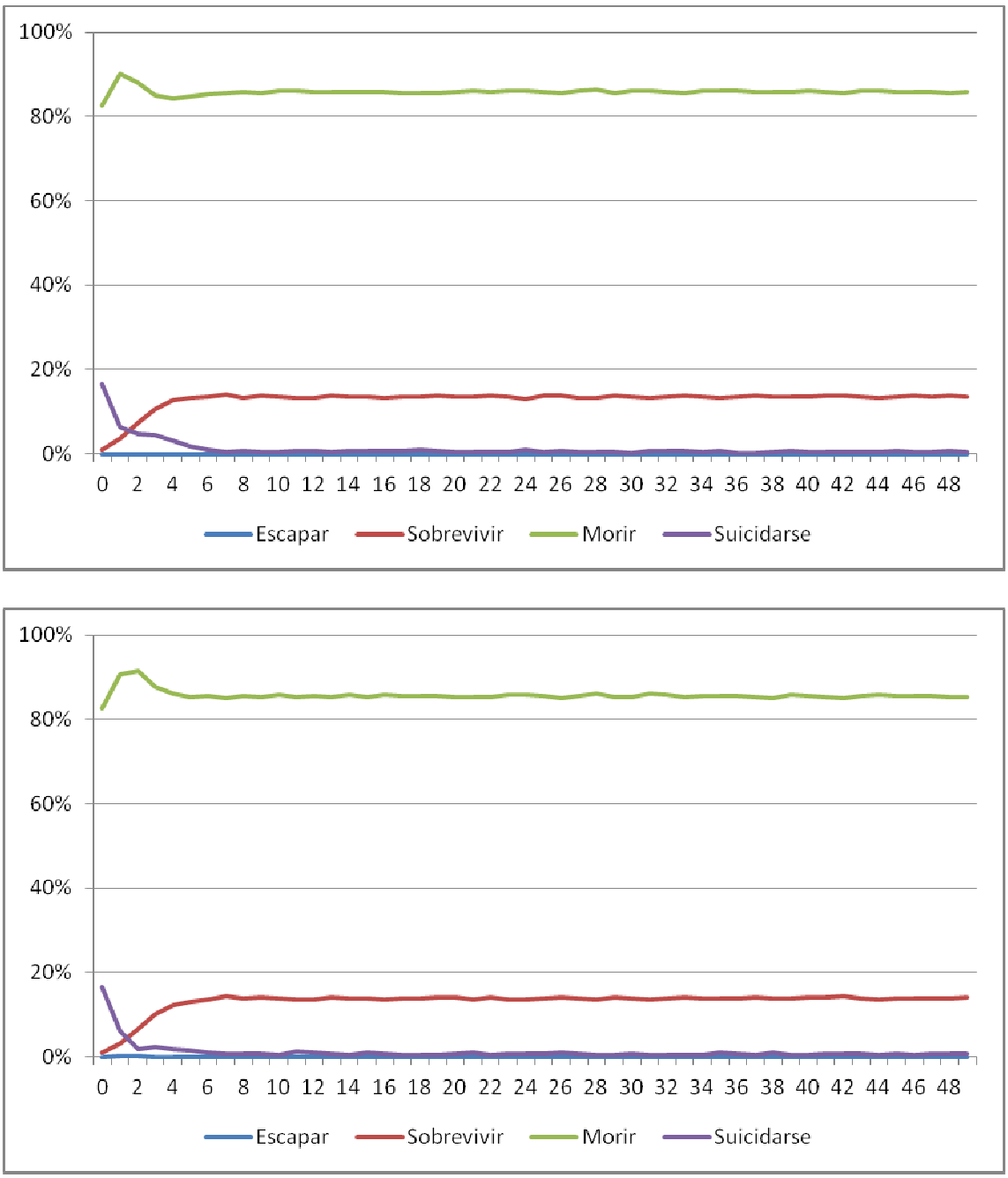


Figura 60. Evolución de las tasas promedio de estado final de superviviente del escenario de matanza con 2 supervivientes

6.3.4 Supervivencia

Con este experimento quisimos encontrar comportamientos puramente defensivos de supervivencia para equipos de 2 supervivientes.

Los mejores comportamientos encontrados (Figura 61) son prácticamente iguales al encontrado en el escenario equivalente para un solo superviviente (Figura 36), que consisten básicamente en alternar acciones de movimiento aleatorio (*Avoid RandBoundary*) con acciones de cambio de dirección de la mirada (*LookLeft/LookRight*). Uno de los dos supervivientes incorpora la particularidad de disparar a la misma casilla donde se encuentra (*Attack (AgentLocation Me)*). Esta acción repercute en el entorno generando una cantidad de estímulo adicional para los zombis en dicha casilla, tanto por origen como por destino del disparo, lo cual, teóricamente es malo, pues está atrayendo con más fuerza a los zombis hacia allí. El hecho de que precisamente este superviviente sobreviva un poco más que su compañero (Figura 62) puede deberse a una simple casualidad, indicando que el efecto de los estímulos añadidos no es lo suficientemente fuerte o que debiera ser mejorado, ya que, en la versión actual del simulador, los estímulos pueden tener distintas intensidades pero siempre tienen el mismo alcance.

Levemente mejores que un superviviente solitario en las mismas condiciones (Figura 37), los supervivientes evolucionados en este experimento logran sobrevivir en torno al 25% y 23% de las simulaciones, muriendo en el resto (Figura 62).

Al igual que en otros experimentos anteriores, la población evoluciona poco y lentamente, encontrando gran cantidad de picos en el mejor valor de *fitness* debidos al *bloating* de los árboles. Tras la generación 24 la población parece estabilizarse finalmente en torno a los valores de la mejor solución encontrada (Figura 63).

Una particularidad observable en los resultados de este experimento es que los *scores* de vida comienzan empeorando en las primeras generaciones (Figura 64) a la par que disminuye la tasa de suicidio a favor de la de muerte y supervivencia (Figura 65). Este, en apariencia, contradictorio efecto, que también es apreciable aunque de manera más leve en la versión del experimento para un superviviente (Figura 40), se debe a que los supervivientes de los escenarios de supervivencia se configuran para empezar con menos vida de su máximo para intentar forzarles a que busquen botiquines, y dado que, en la implementación actual, al suicidarse no se les baja la vida a 0, los supervivientes suicidas provocan un moderado hinchamiento de los *scores* de vida en las primeras fases de la evolución (Figura 64)

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(Do (Do (Do (Attack (AgentLocation Me)) LookLeft) (Attack (AgentLocation Me))) (Do (Avoid RandBoundary) LookLeft)) (Do LookRight (Avoid RandBoundary))

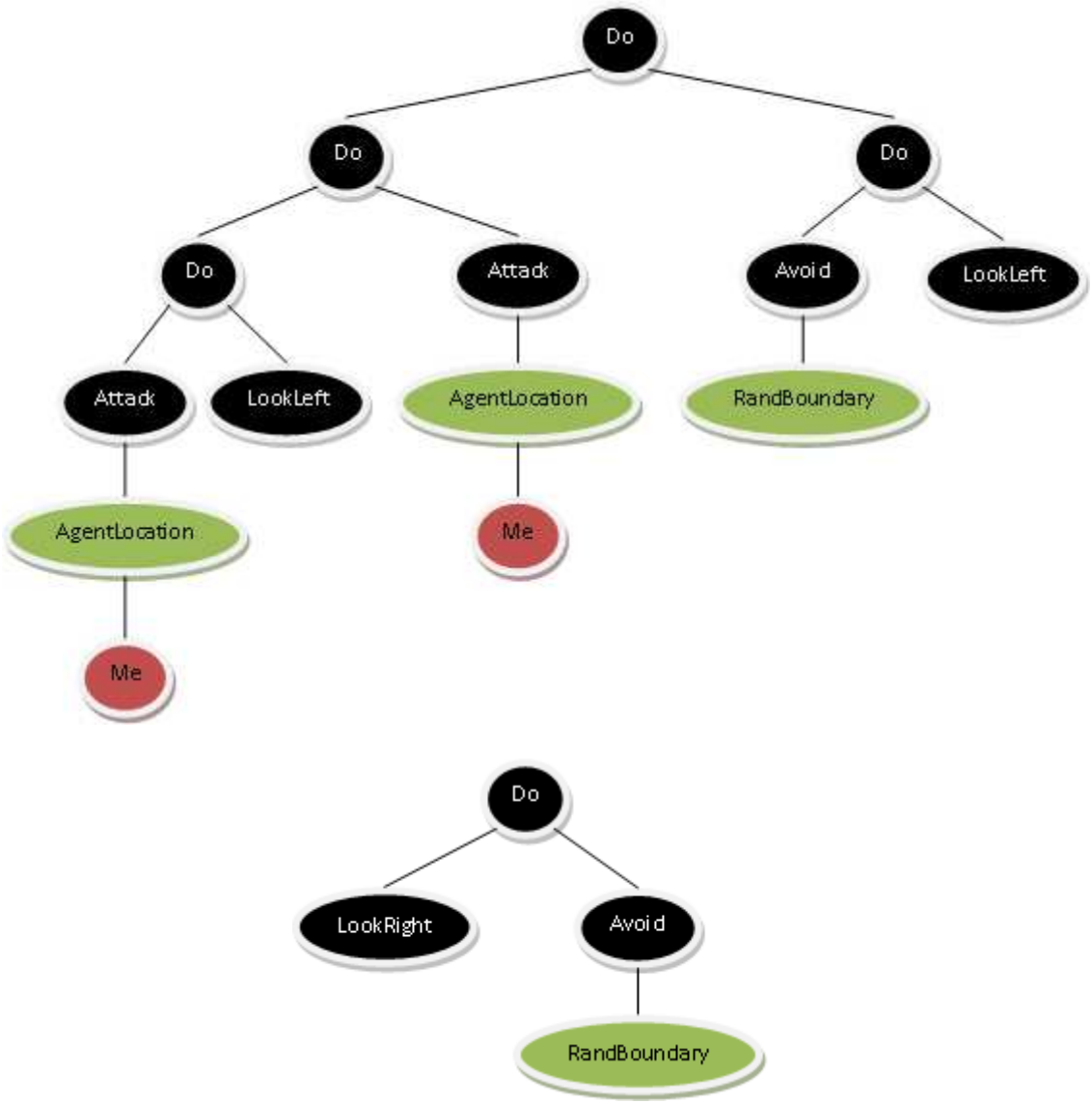


Figura 61. Árboles simplificados del escenario de supervivencia con 2 supervivientes

   b. ***Fitness* promedio y desviación típica** = 2’6385 (0’6081)

c. **Tasas promedio de estado final de cada superviviente**

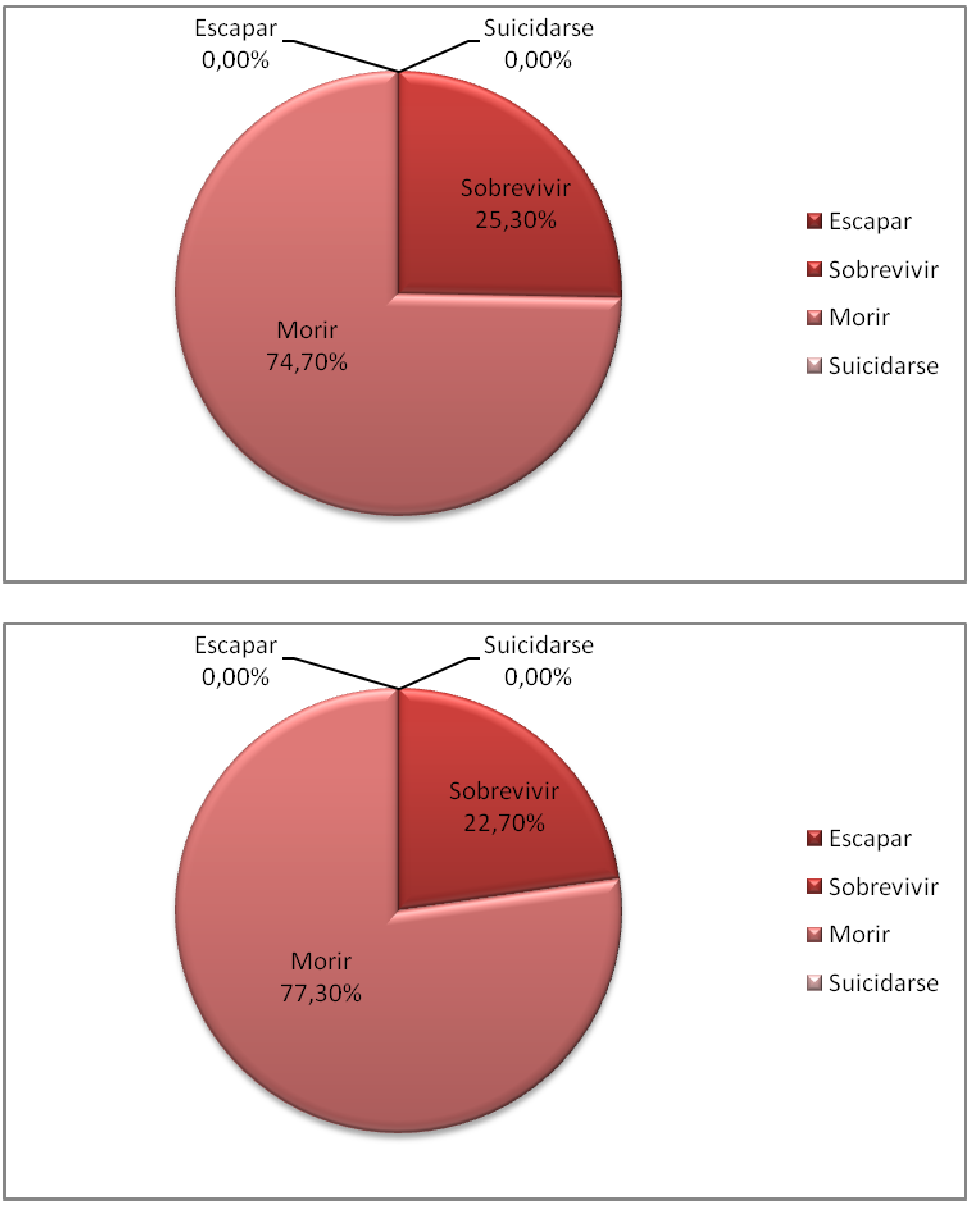


Figura 62. Tasas promedio de estado final del mejor individuo del escenario de supervivencia con 2 supervivientes

2. **Evolución por generación del *fitness* de los individuos**

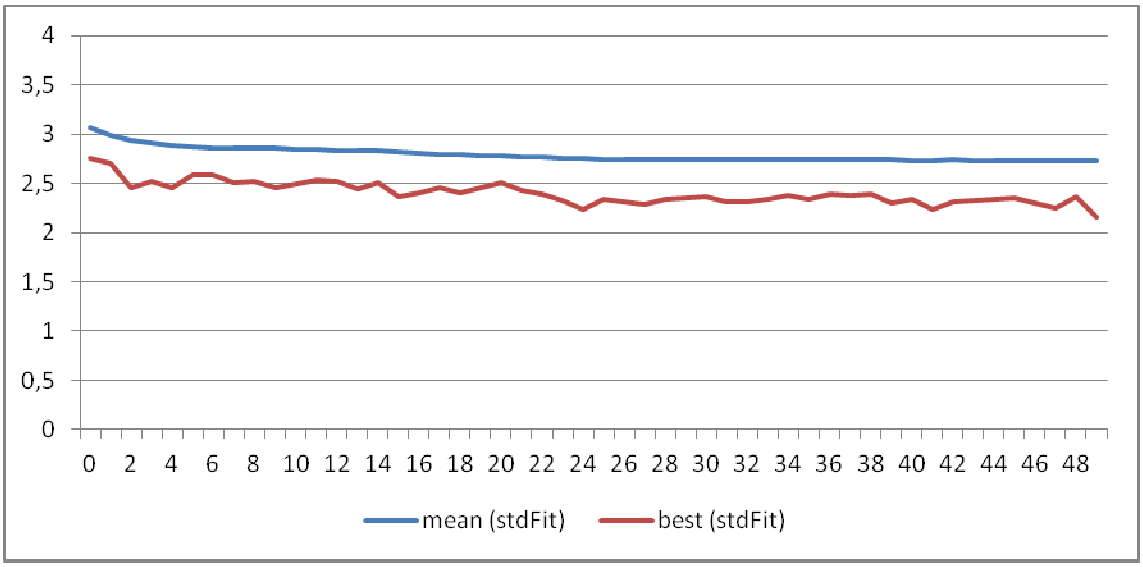


Figura 63. Evolución del *fitness* promedio y mejor del escenario de supervivencia con 2 supervivientes

3. **Evolución por generación de los *scores* promedio de cada superviviente**

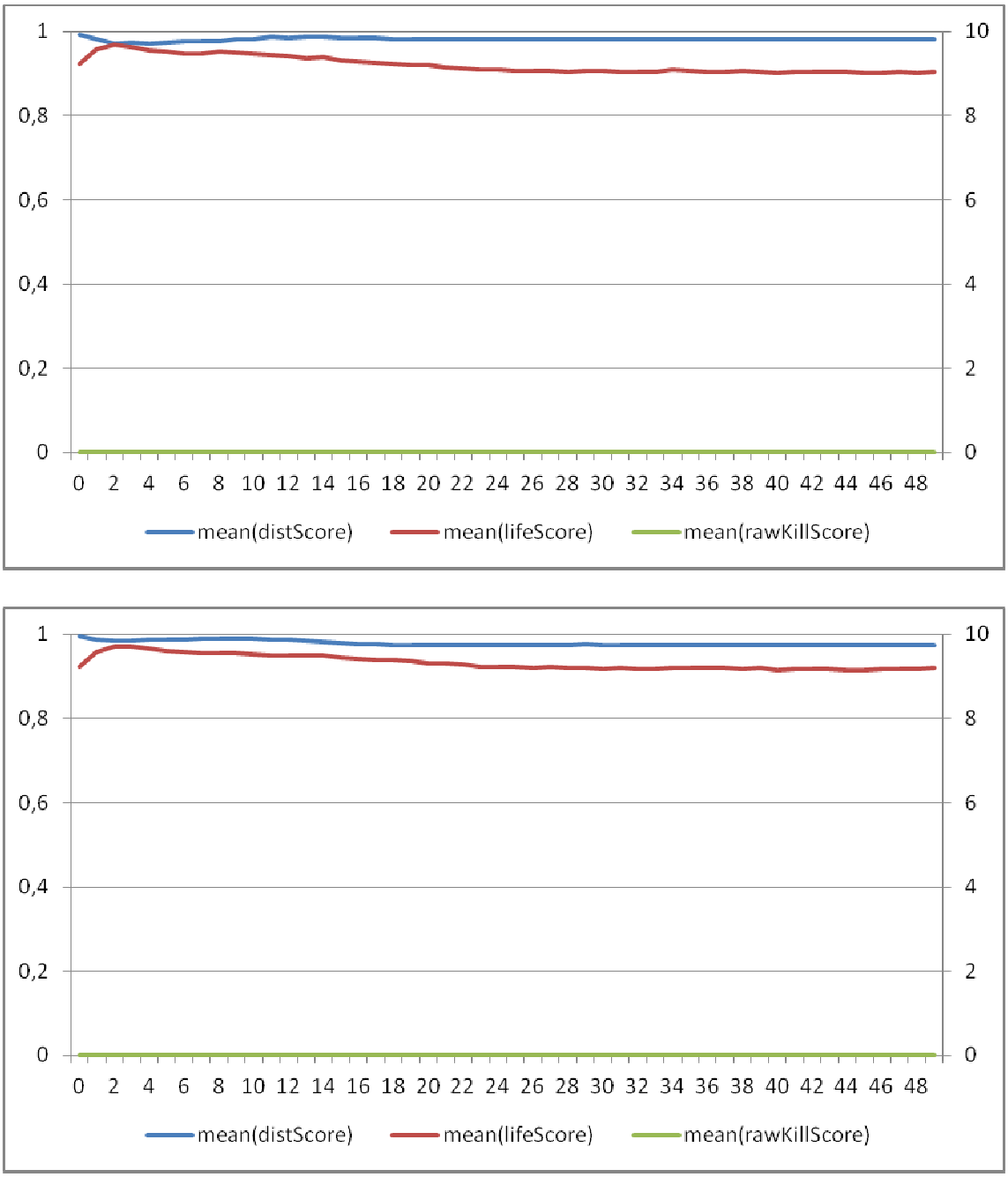


Figura 64. Evolución de los *scores* promedio del escenario de supervivencia con 2 supervivientes

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

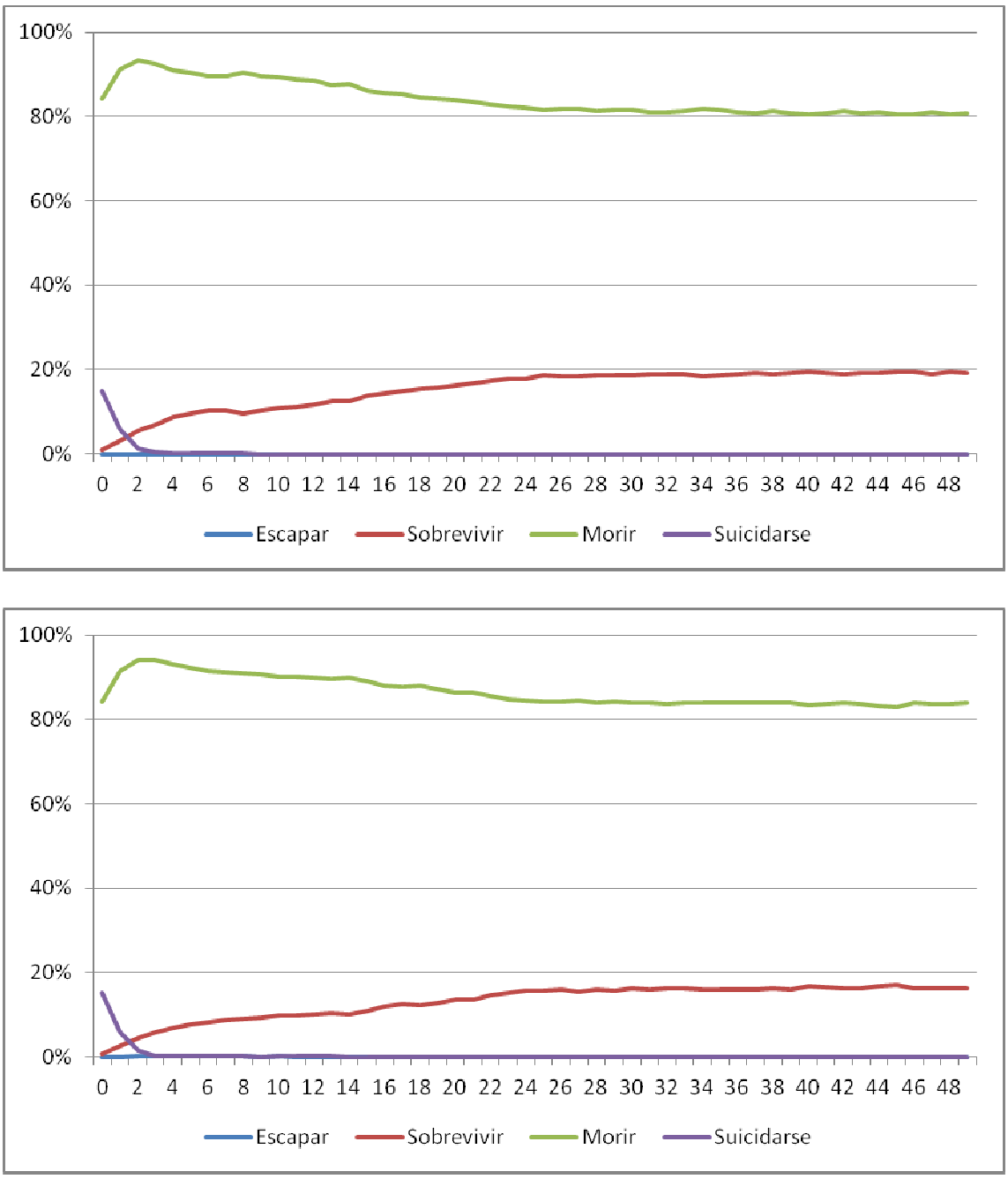


Figura 65. Evolución de las tasas promedio de estado final de superviviente del escenario de supervivencia con 2 supervivientes

6.3.5 Combinado

En este experimento se buscaron comportamientos para un equipo de 2 supervivientes con el mismo objetivo que en el escenario Básico pero que mejoraran los resultados encontrados en éste, utilizando para ello funciones adicionales de acción con los comportamientos individuales simples aprendidos en los escenarios de Escape, Matanza y Supervivencia.

Las mejores soluciones encontradas (Figura 66) son idénticas para ambos supervivientes y consisten sencillamente en matar (*Kill*) utilizando el comportamiento evolucionado en el escenario de matanza de un superviviente (Figura 31).

Con este comportamiento obtuvimos el equipo de supervivientes con mayor tasa de supervivencia de todos los experimentos (a excepción de los escenarios de Escape donde no es posible morir): 99% y 97%. En el resto de las ocasiones los supervivientes mueren, pero no escapan ni se suicidan (Figura 67).

Dado que este mejor comportamiento de equipo encontrado consiste sencillamente en tomar comportamientos individuales ya conocidos, la población encuentra la mejor solución prácticamente inmediatamente para converger posteriormente hacia ella en unas 4 generaciones (Figura 68). En las gráficas (Figura 69) podemos ver cómo los *scores* de distancia prácticamente son ignorados desde el principio, mientras que los de muertes finalizan algo por encima de 2 (i.e. un equipo promedio acaba con 4 zombis por simulación) y los de vida en menos de 0'1 (i.e. el superviviente promedio apenas recibe un único impacto por simulación). Análogamente las tasas de suicidio y muerte prácticamente desaparecen en las primeras generaciones a favor de la de supervivencia (Figura 70). Apenas se observan picos de ningún tipo en los resultados, dado que el comportamiento óptimo al que se ha convergido es muy estable con un bajo valor de desviación del *fitness*, 0'129, en relación al valor promedio, 1'7126.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

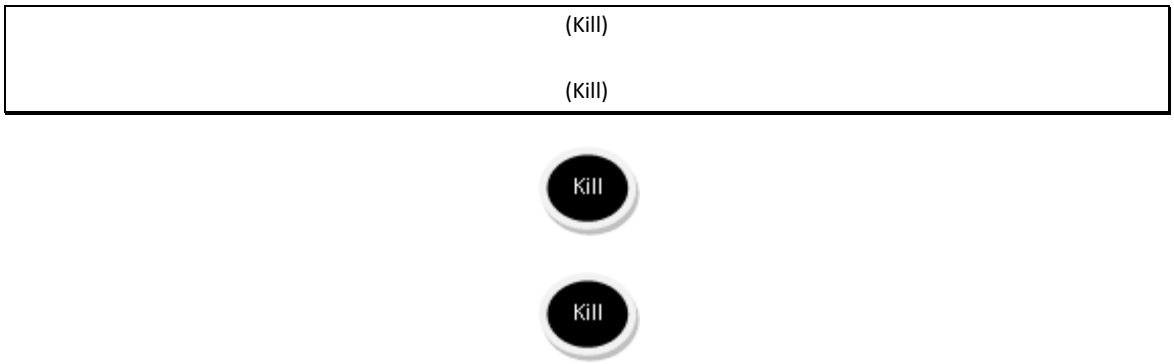


Figura 66. Árboles simplificados del escenario combinado con 2 supervivientes

   b. ***Fitness* promedio y desviación típica** = 1'7126 (0'129)

c. **Tasas promedio de estado final de cada superviviente**

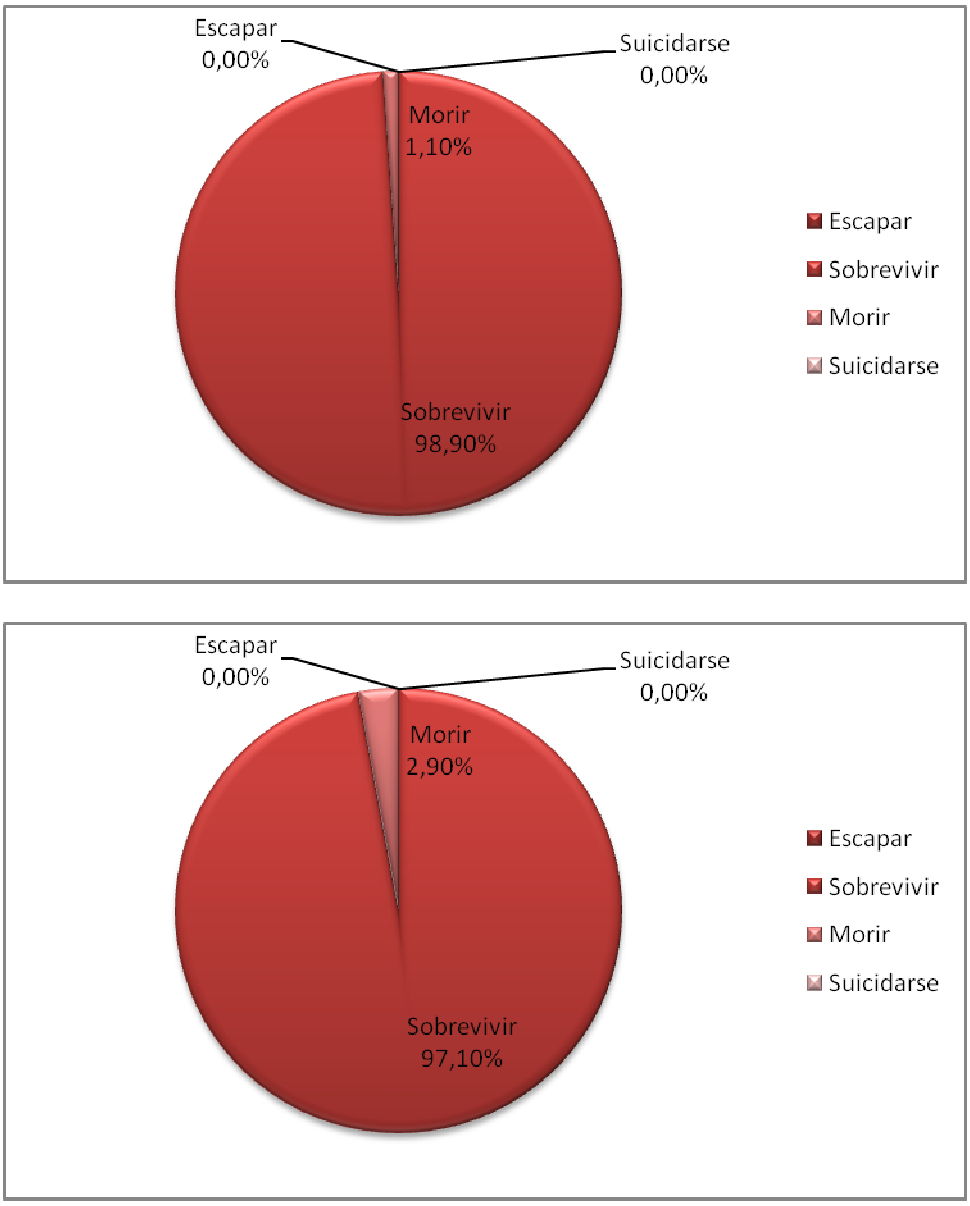


Figura 67. Tasas promedio de estado final del mejor individuo del escenario combinado con 2 supervivientes

2. **Evolución por generación del *fitness* de los individuos**

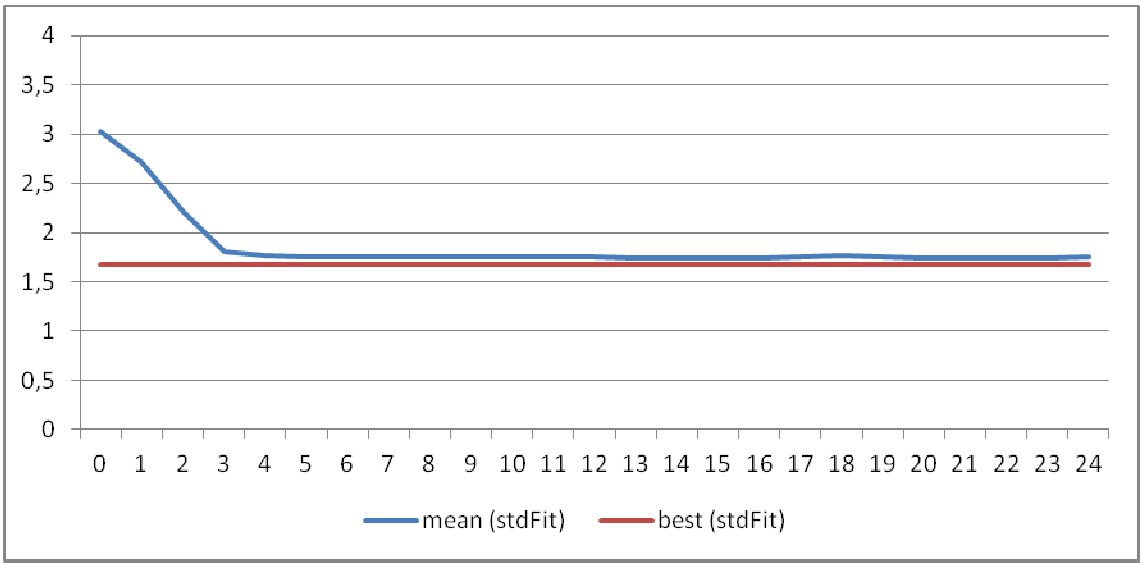


Figura 68. Evolución del *fitness* promedio y mejor del escenario combinado con 2 supervivientes

3. **Evolución por generación de los *scores* promedio de cada superviviente**

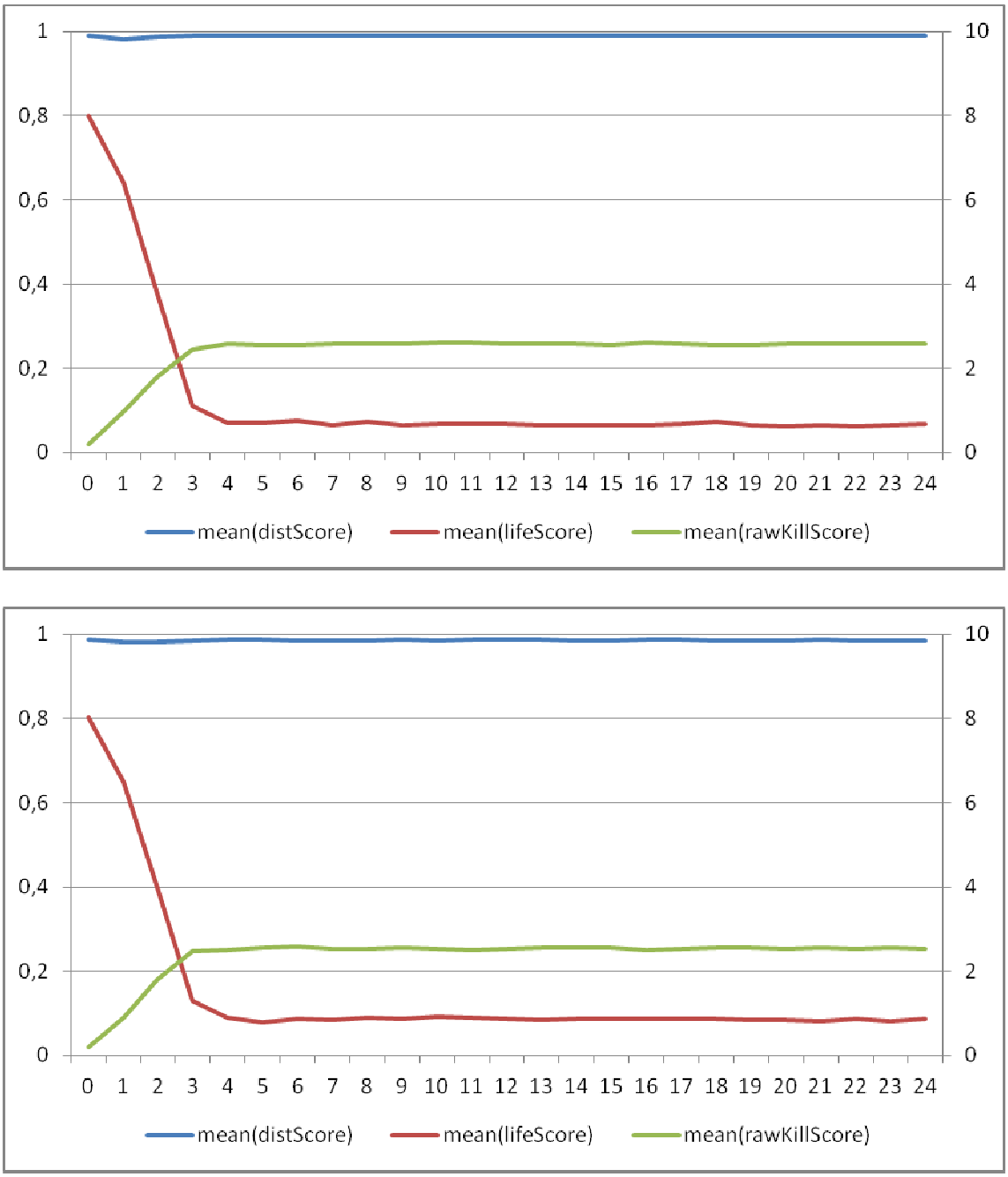


Figura 69. Evolución de los *scores* promedio del escenario combinado con 2 supervivientes

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**

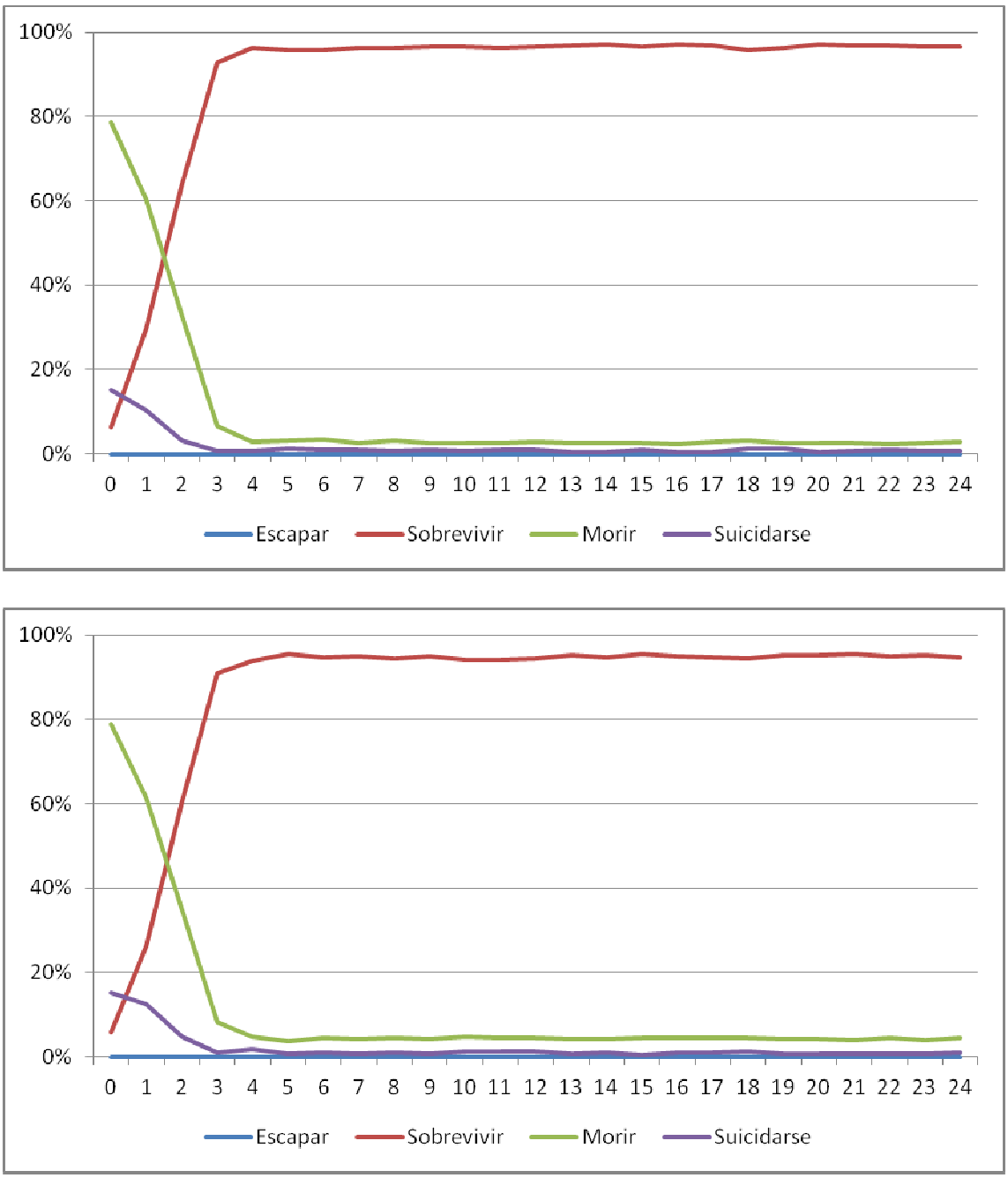


Figura 70. Evolución de las tasas promedio de estado final de superviviente del escenario combinado con 2 supervivientes

6.4 Experimento 3: Equipos de 4 supervivientes

6.4.1 Básico

A modo de caso de estudio, se probó este experimento utilizando los mismos objetivos de los escenarios Básicos de 1 y 2 supervivientes para un equipo de 4 supervivientes.

Las mejores soluciones encontradas (Figura 71) parten al equipo en 2 subgrupos de comportamientos distintos de 2 supervivientes cada uno. Por un lado, uno de los subgrupos aprende a cazar al zombi más próximo (*Attack (AgentLocation NearestEnemy)*) o mirar hacia otro lado (*LookRight*), de manera similar a los comportamientos ofensivos vistos en otros escenarios. Sin embargo, el conjunto de condiciones para decidir atacar no es el óptimo. Las condiciones evolucionadas incluyen factores razonables como estar herido (*AmIHurt*), para atacar sólo cuando está siendo atacado, no tener ningún botiquín a la vista (*Not MediKitKnown*), ya que, aunque finalmente en la mejor solución no se ha aprendido, podría en ese caso haber aprendido a optar por intentar recoger un botiquín antes que buscar el enfrentamiento, y tener aliados a la vista, de manera que nunca se pierda la coherencia de unidad con el resto del equipo. Pero dicho conjunto de condiciones no incluye factores muy importantes como es el tener enemigos a la vista (*EnemiesInSight*) a los que poder seleccionar y atacar, lo cual puede obligar al superviviente a realizar acciones de espera si opta por atacar y no tiene zombis a la vista. El otro subgrupo aprende sencillamente a atacar/moverse siempre hacia la casilla límite más cercana (*Attack NearestBoundary*).

Tanto los miembros de un subgrupo como los del otro tienden a sobrevivir un poco por encima del 50% y no escapan nunca (Figura 72). La diferencia fundamental la manifiestan los del subgrupo cazador, que se suicidan en torno al 5% de las veces por el efecto comentado anteriormente provocado por una elección subóptima de condiciones para iniciar un ataque.

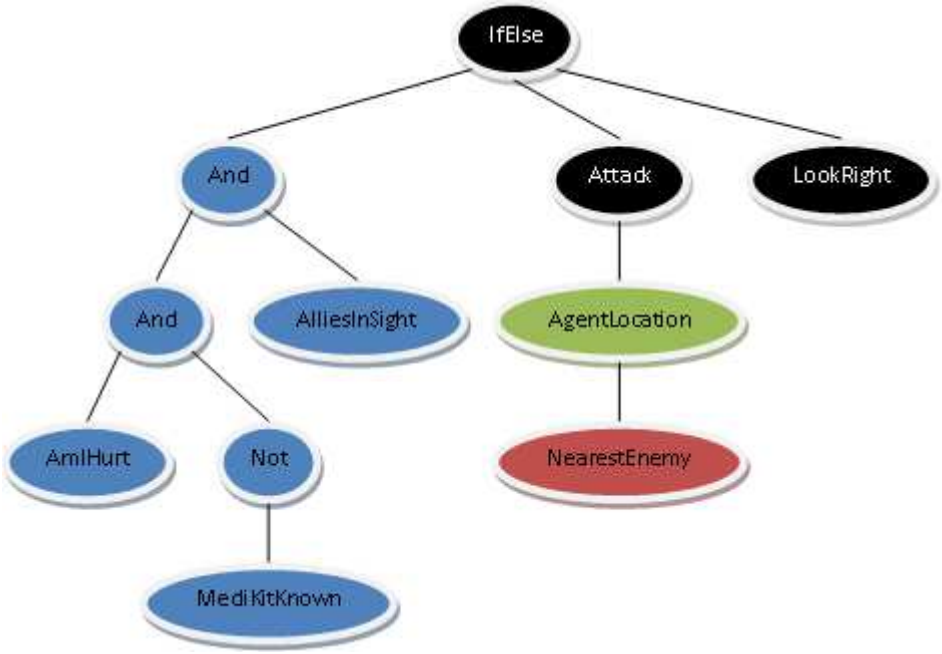
Tras las 25 generaciones configuradas como límite para este experimento todavía se aprecian ciertas tendencias al final de todas las curvas, en particular en la evolución del *fitness* (Figura 73). Debería prolongarse la evolución en sucesivas pruebas de este escenario para ver si es posible encontrar mejores soluciones.

Como ya ha ocurrido en más experimentos, la evolución de los *scores* de vida restante presenta cierto sesgo inicial debido al efecto de los suicidios (Figura 75). Pasadas estas entre 3 y 7 primeras generaciones, los suicidios bajan hasta estabilizarse en valores próximos al 0% mientras la tasa de supervivencia crece hasta estabilizarse en torno al 40%. Parte de los suicidios son absorbidos en la tasa de muertes durante las primeras 3 generaciones, pero luego baja para estabilizarse más o menos a la vez que las otras tasas, quedándose en torno al 60%. En todo este proceso, los *scores* (Figura 74) de distancia apenas bajan del 1 inicial, mientras los de vida restante se desplazan coherentemente de acuerdo a las tasas de supervivencia y muerte. Los *scores* de muertes se mueven muy lentamente, no llegando en ninguno de los 4 supervivientes el promedio de la población ni a 0’5. Sin embargo, como se ha desatacado en el anterior párrafo, aproximadamente en torno a las 23 generaciones, los *scores* de vida empiezan a mejorar un poco e incluso el *score* de muertes de uno de los supervivientes

cazadores también empieza a crecer levemente conjuntamente con la tasa de supervivencia y también un poco la de suicidios, lo cual da pie a pensar que los mejores individuos encontrados hacia el final de la evolución realizada es posible que pudieran haber sido mejores de haber prolongado el proceso evolutivo.

1. **Resultados del mejor individuo de la evolución**
   a. **Árboles evolucionados del equipo de supervivientes**

(IfElse (And (And AmIHurt (Not MediKitKnown)) AlliesInSight) (Attack (AgentLocation NearestEnemy)) LookRight)
(IfElse (And (And AmIHurt (Not MediKitKnown)) AlliesInSight) (Attack (AgentLocation NearestEnemy)) LookRight)
(Attack NearestBoundary)
(Attack NearestBoundary)



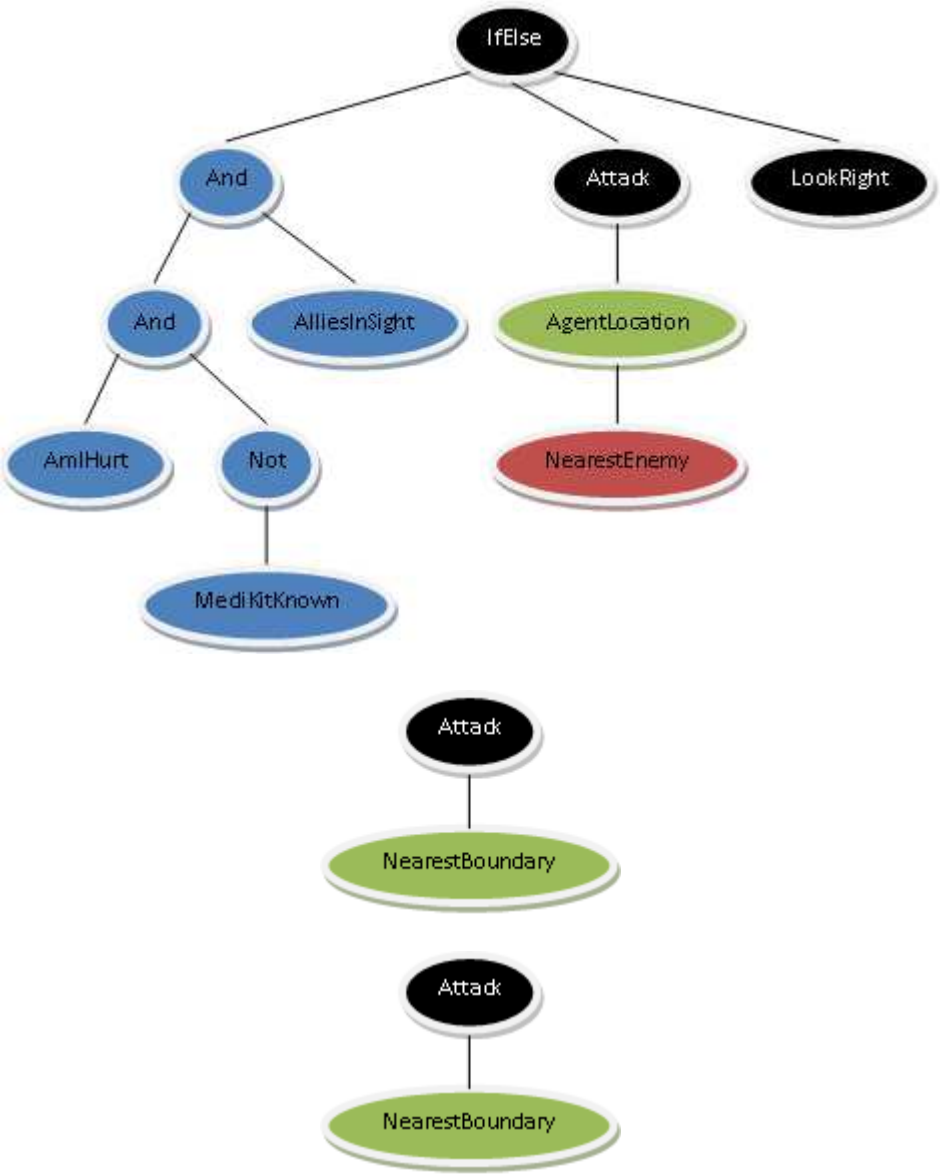
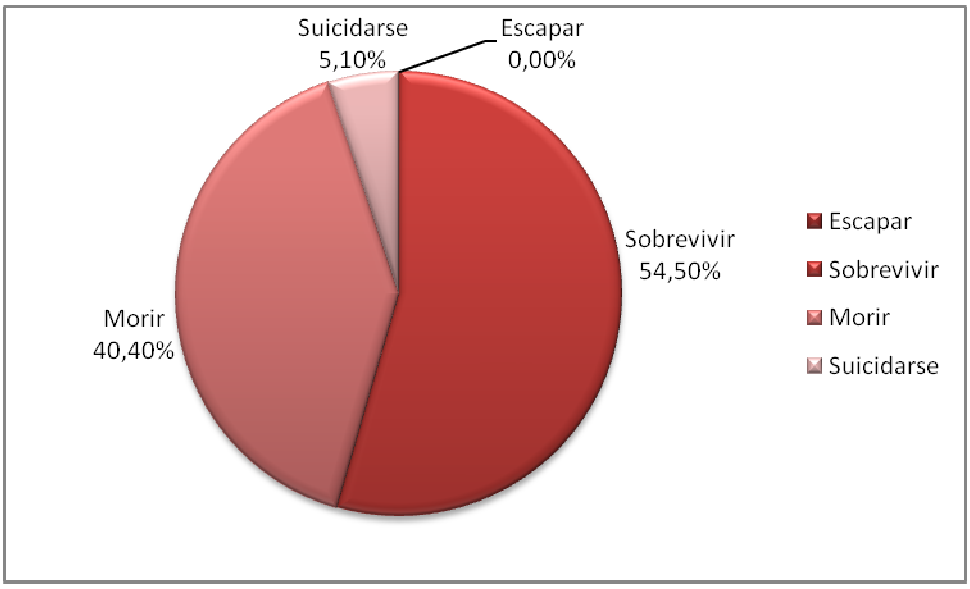
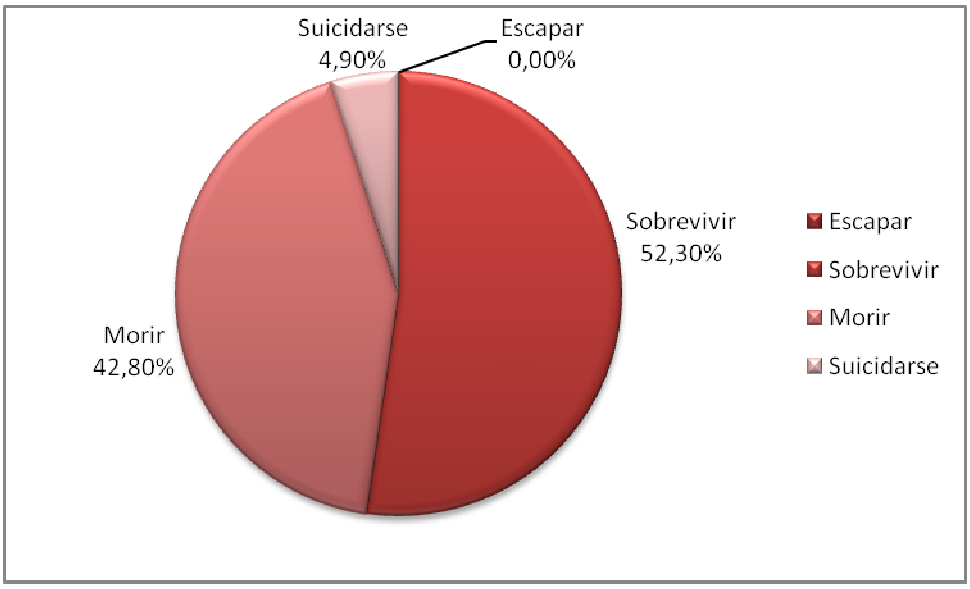


Figura 71. Árboles simplificados del escenario básico con 4 supervivientes

b. ***Fitness* promedio y desviación típica** = 2’3457 (0’5712)

c. **Tasas promedio de estado final de cada superviviente**





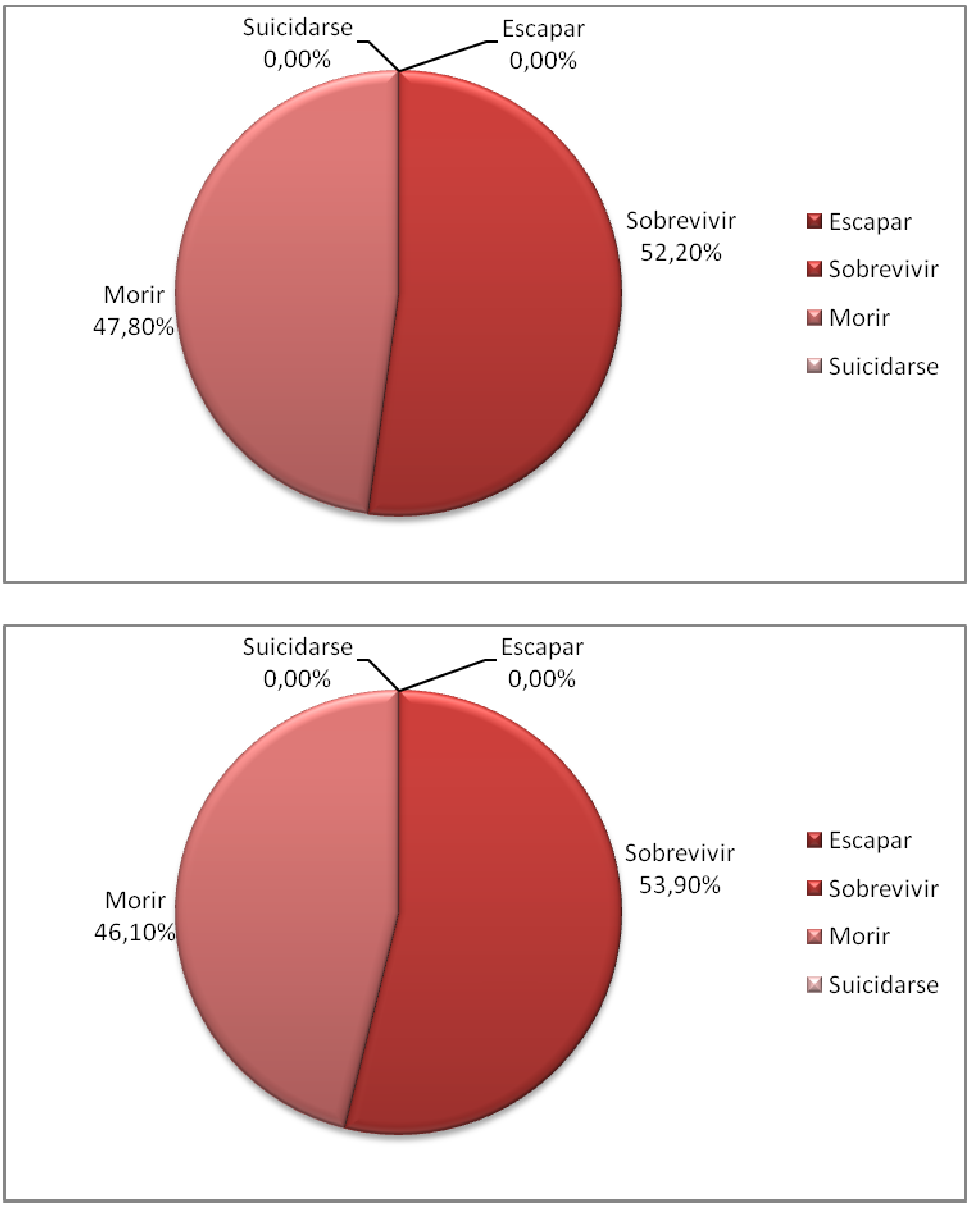


Figura 72. Tasas promedio de estado final del mejor individuo del escenario básico con 4 supervivientes

2. **Evolución por generación del *fitness* de los individuos**

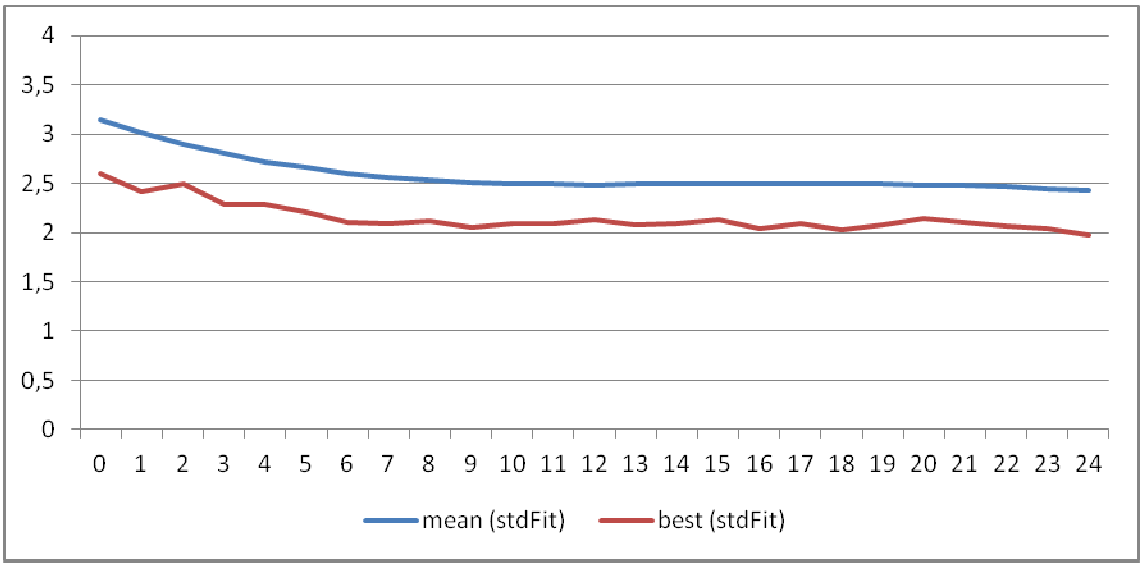
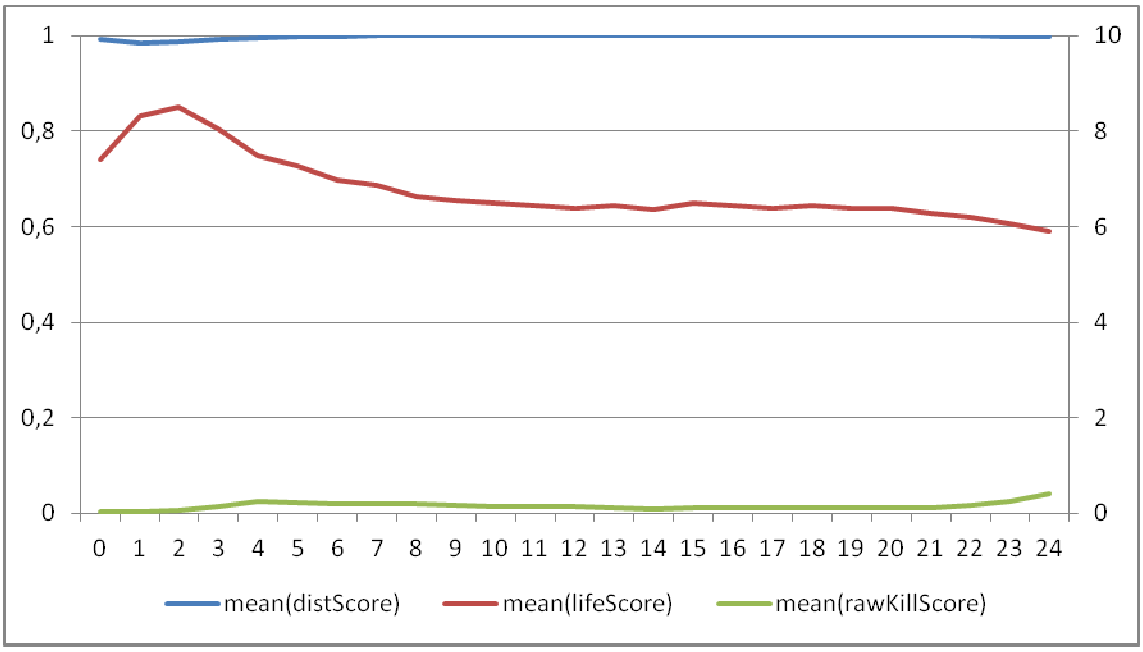
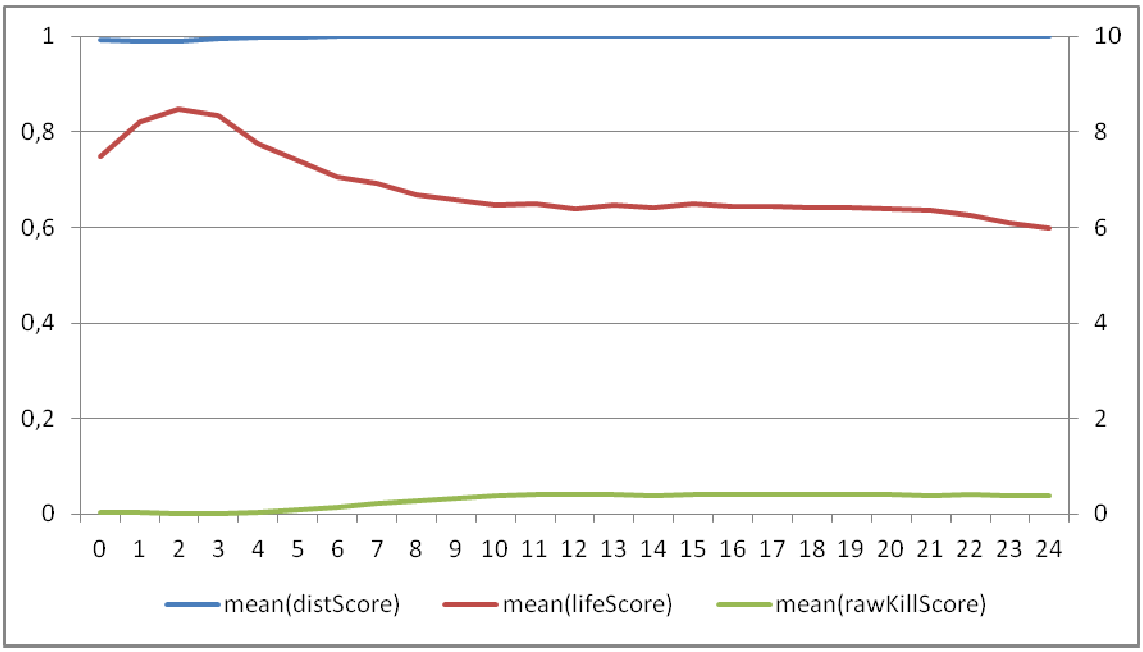


Figura 73. Evolución del *fitness* promedio y mejor del escenario básico con 4 supervivientes

3. **Evolución por generación de los *scores* promedio de cada superviviente**





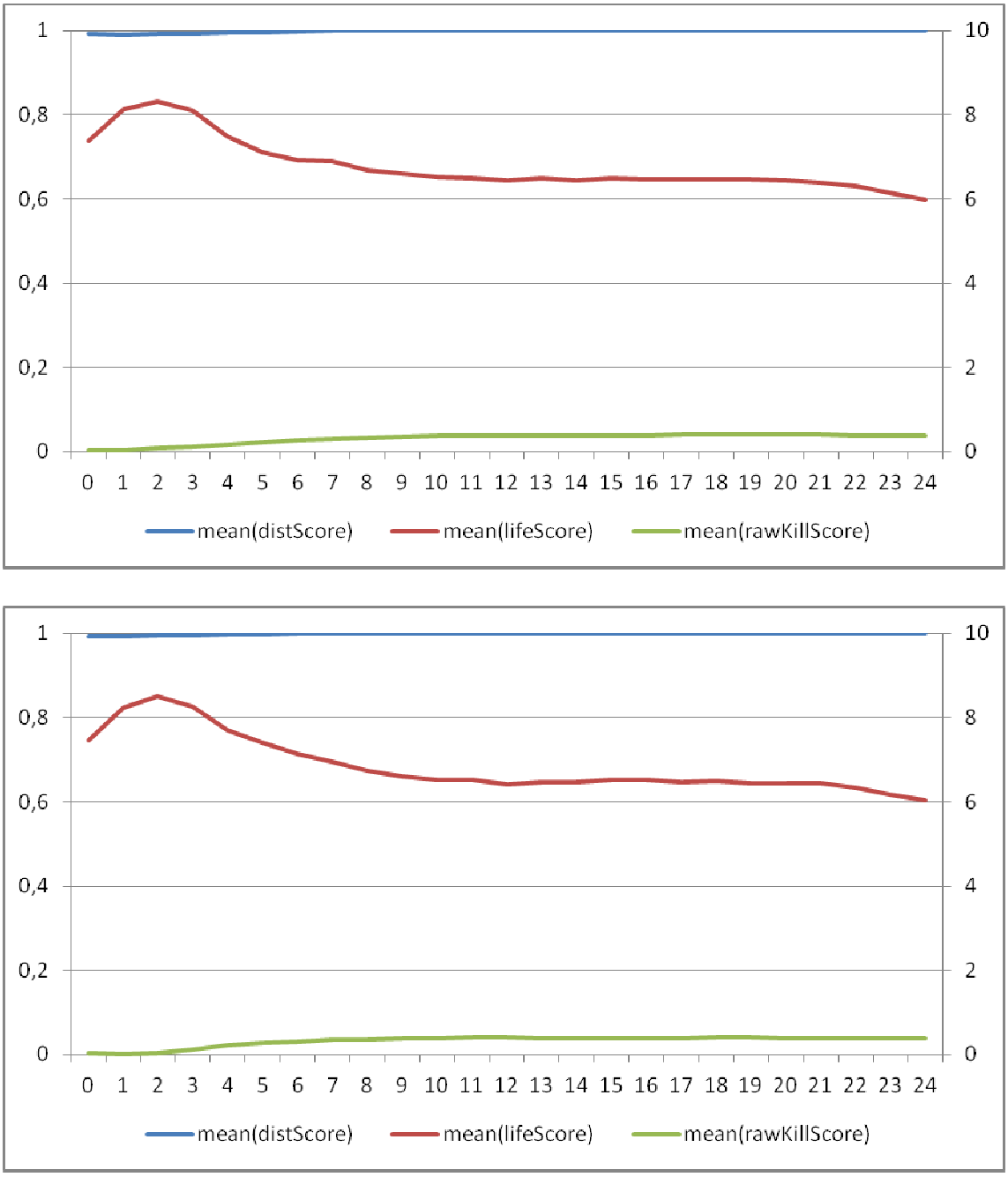
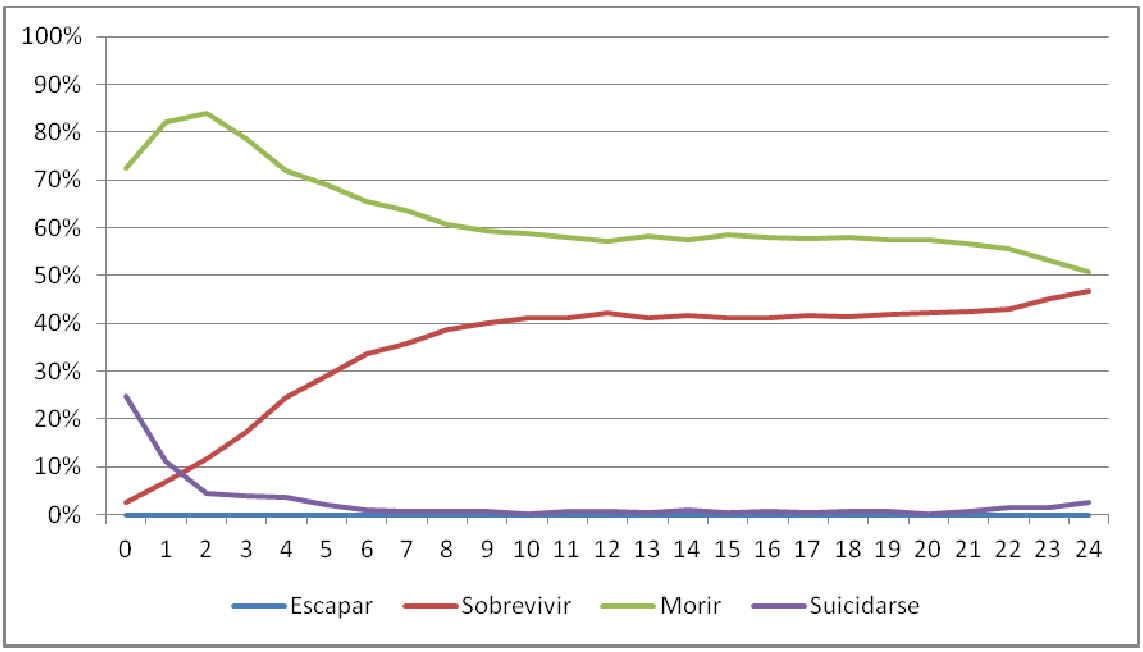


Figura 74. Evolución de los *scores* promedio del escenario básico con 4 supervivientes

4. **Evolución por generación de las tasas promedio de estado final de cada superviviente**



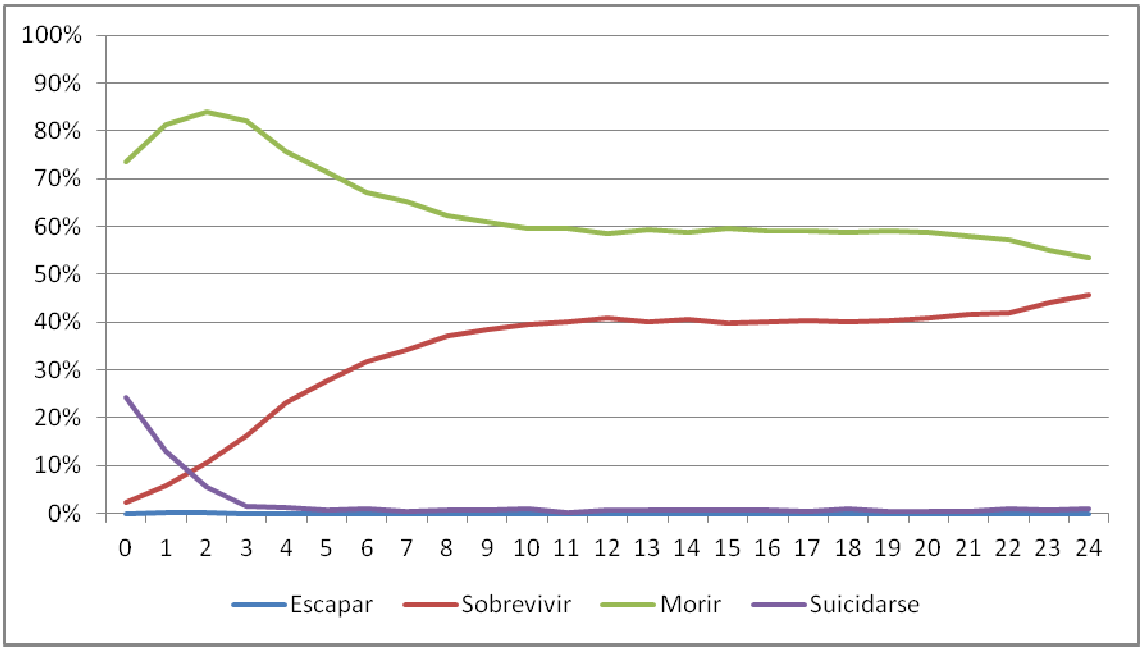


Figura 75. Evolución de las tasas promedio de estado final de superviviente del escenario básico con 4 supervivientes

7. Discusión

7.1 Resumen de resultados

En total, se han llevado a cabo 11 procesos evolutivos. La Tabla 7 recopila los resultados de todos los experimentos en cuanto al valor del *fitness* del mejor individuo de la evolución.

Tabla 7. Resumen del promedio y desviación típica del *fitness* de los mejores individuos encontrados

	1 superviviente	**2 supervivientes**	**4 supervivientes**
Básico	1'7614 (0'3098)	2'5853 (0'4472)	2'3457 (0'5712)
Escape	1'0922 (0'0687)	1'4077 (0'407)	-
Matanza	1'6966 (0'2851)	2'7631 (0'3685)	-
Supervivencia	2'728 (0'5846)	2'6385 (0'6081)	-
Combinado	1'7795 (0'327)	1'7126 (0'129)	-

La Tabla 8 muestra las tasas de estado final de superviviente, promediando para todos los supervivientes del equipo.

Tabla 8. Resumen de las tasas de estado final de superviviente de los mejores individuos encontrados

	1 superviviente				**2 supervivientes**			
	Escapar	**Sobrevivir**	**Morir**	**Suicidarse**	**Escapar**	**Sobrevivir**	**Morir**	**Suicidarse**
Básico	0%	94%	5'9%	0'1%	0'45%	26'25%	73'3%	0%
Escape	0%	100%	0%	0%	0'03%	99'97%	0%	0%
Matanza	0%	93'5%	6'5%	0%	0%	15'75%	84'25%	0%
Supervivencia	0%	17'9%	82'1%	0%	0%	24%	76%	0%
Combinado	0%	93'1%	6'9%	0%	0%	98%	2%	0%

	4 supervivientes			
	Escapar	Sobrevivir	Morir	Suicidarse
Básico	0%	53’22%	44’28%	2’5%

Los resultados de estas tablas serán analizados en las siguientes subsecciones.

7.2 Aprendizaje

La primera observación destacable que se puede realizar es la incapacidad por parte del algoritmo de GP para encontrar comportamientos en los supervivientes que les permitan escapar eficazmente del mapa en la versión actual de la plataforma ni siquiera en el escenario específico de Escape, donde no hay zombis que amenacen la existencia de nuestros supervivientes. No se ha encontrado ningún argumento suficientemente sólido que permita explicar el porqué de este cambio sobre versiones anteriores, en las que sí se llegaron a obtener expresiones del estilo *(IfElse ExitKnown (GoTo NearestExit) (GoTo NearestBoundary))* que permitían al superviviente encontrar la salida en la mayor parte de las ocasiones, o quedándose muy cerca de la misma al final del tiempo estipulado para la simulación. En la Figura 76 se muestra la evolución de la cantidad de nodos por tipo entre todos los individuos de la población en el escenario de Escape para un superviviente, donde se aprecia cómo desaparecen de la población rápidamente y casi por completo las funciones *NearestExit* y *ExitKnown*. Este hecho, unido a que la tasa de mutación aplicada es muy baja, descarta prácticamente la posibilidad que se pueda llegar a alcanzar una mejor solución con mayor cantidad de generaciones. La ausencia de éxito tal vez sea simplemente debida al azar dado que no se ha repetido ni éste ni ninguno de los otros experimentos con semillas aleatorias distintas para comprobar la consistencia de las soluciones a las que el algoritmo converge o la rapidez con que dichas soluciones son obtenidas.

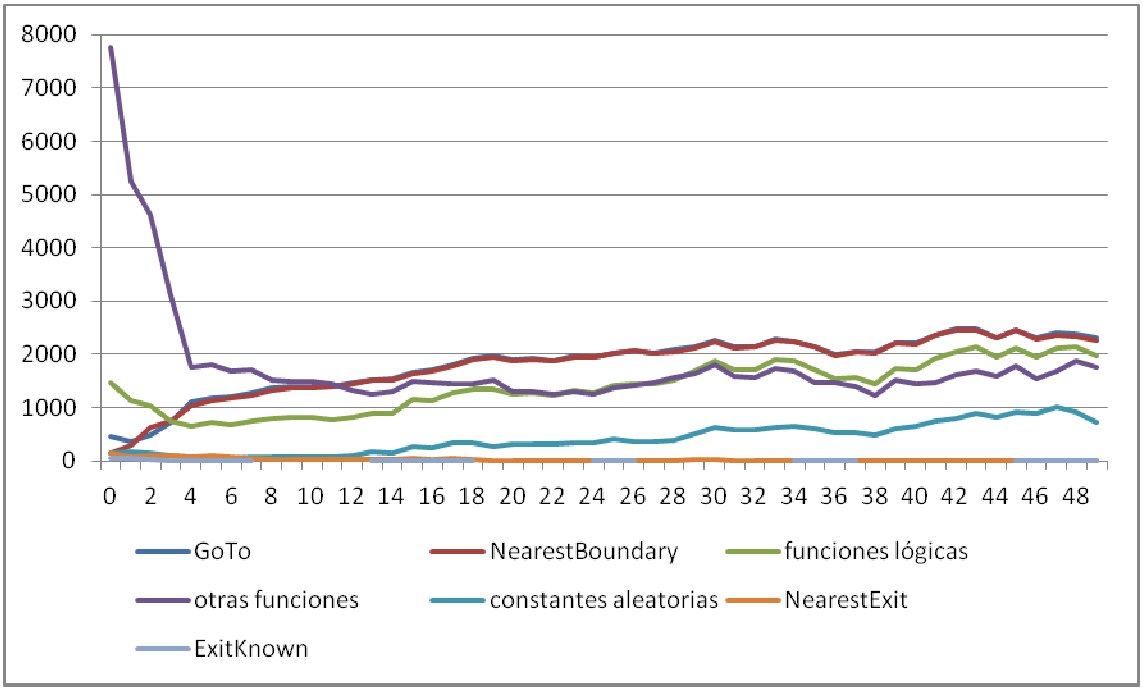


Figura 76. Evolución de la cantidad de nodos por tipo en el escenario de escape para 1 superviviente

Sin embargo, aunque no llegan a derivar soluciones para ir hacia la salida cuando ésta está a la vista, sí que llegan a aprender comportamientos sencillos para explorar intensamente el mapa (e.g.: *(GoTo NearestBoundary)* en el escenario de Escape), los cuales les permiten aproximarse bastante a la salida en algún momento de la simulación y obtener de manera consistente buenos *scores* de distancia, especialmente cuando ésta es el factor más relevante (e.g.: escenarios de Escape), e incluso cayendo ocasionalmente en una casilla de salida (e.g.: Escenarios Básico y Escape, ambos de 2 supervivientes).

Por otra parte, los supervivientes sí que consiguen evitar, con elevada eficacia, y para la totalidad de escenarios, comportamientos con poco sentido semántico que les obliguen a forzar acciones de espera, o con demasiada cautela que les obliguen igualmente a pasar demasiado tiempo inactivos y, en consecuencia, verse forzados a cometer suicidio.

Al margen del escenario de escape, donde es imposible que los supervivientes mueran, otro punto muy positivo que se aprecia de los comportamientos aprendidos son las tasas alcanzadas de supervivencia, llegando a la cota del 98% cuando 2 supervivientes se juntan para desarrollar un comportamiento combinado utilizando los subcomportamientos aprendidos en escenarios más sencillos.

Por último, se puede decir de los aprendizajes realizados por los supervivientes que hacen cierto el dicho popular “la mejor defensa es un buen ataque”. Todos los comportamientos satisfactorios obtenidos para la supervivencia constan esencialmente de acciones de ataque (e.g.: *(Attack (AgentLocation NearestEnemy))*), aunque sólo cuando el superviviente siente cierta amenaza (e.g.: *(IfElse EnemiesInAttackRange)*…), siendo especialmente adecuado el comportamiento del escenario Combinado para 1 superviviente, *(IfElse (Not HaveIAmmo)*

Survive Kill), que chequea primero si tiene munición para ver si le merece la pena ir al ataque aunque sienta una amenaza próxima.

7.3 Impacto del tamaño del equipo

Mientras que en ciertos escenarios, Escape, Supervivencia y Combinado, el efecto de tener varios supervivientes es muy leve, en otros escenarios, Básico y Matanza, la presencia de más supervivientes parece dificultar el aprendizaje de comportamientos de supervivencia en todos los miembros del equipo. Tampoco se ha encontrado explicación para este curioso efecto, que debiera ser contrastado con posterioridad a este trabajo con ejecuciones adicionales modificando las semillas aleatorias.

Incluso en los escenarios donde afecta de manera leve, la forma en qué lo hace difiere mucho entre unos casos y otros. En el caso del escenario de Escape, los supervivientes parecen entorpecerse el uno al otro pasando de un muy buen valor de *fitness*, 1'0922, de alta consistencia (i.e. baja desviación), 0'0687, a otro peor, 1'4077, y mucho más volátil (i.e. alta desviación) 0'407[65]. En cambio, en el escenario de Supervivencia el mayor tamaño del grupo es positivo, ya que los supervivientes sobreviven casi un 6% más (17'9% → 24%), como lo es en el escenario Combinado, donde una mayor capacidad de fuego por parte de los supervivientes les sugiere ir directamente al ataque sin más, mejorando sus valores de *fitness* tanto en promedio, 1'7795 → 1'7126, como en desviación típica, 0'327 → 0'129, y aumentando también su tasa de supervivencia, 93'1% → 98%.

Curiosamente, para el escenario Básico, el único en el que se han podido llegar a probar las tres diferentes configuraciones de tamaño de equipo, aunque los resultados siempre son peores cuando hay más de un superviviente presente, en el caso de haber 4 parece algo mejor que sólo con 2, tal vez porque se compensa la pobre calidad de los comportamientos de parte del equipo con la de otros que, al menos, sí que dotan a sus correspondientes supervivientes con capacidad de disparo, elevando así la probabilidad de supervivencia de los más débiles.

7.4 Impacto de la combinación de comportamientos simples

Sin duda, el desglose del aprendizaje en escenarios sencillos y la elección de una representación apropiada que permitiera la subsunción de patrones simples en otros más complejos, han aportado buenos resultados a este trabajo.

Sin embargo, la utilidad de esta subsunción de comportamientos respecto al escenario básico varía fuertemente entre la configuración de 1 superviviente y la de 2. Cuando hay un único superviviente, la mejora podríamos decir que es de tipo estructural, ya que podemos ver cómo

[65] En realidad estos valores debieran ser algo mayores puesto que los scores de muertes están normalizados por el mejor individuo asesino de la población, el cual es peor que el propio individuo completo

se construye un comportamiento compacto de alto nivel de manera modular empleando comportamientos sencillos de más bajo nivel, pese a que los resultados sean algo peores. En el caso de 2 supervivientes, lo que conseguimos es que emerja cierta forma de cooperación indirecta. El equipo de asesinos evolucionado allí hace que funcionen mejor en equipo que de manera individual. Tristemente, y pese a que dispusimos de medios a los supervivientes para la cooperación, ésta es la más sofisticada pauta de comportamiento cooperativo que se obtuvo en la experimentación.

Por cuestiones de tiempo no ha sido posible aplicar con 4 supervivientes la misma metodología que para 2.

7.5 Problemas encontrados

7.5.1 *Bloating*

Como ya se vio en el capítulo 2.3.5, el problema del crecimiento desmesurado de árboles conocido como *bloating* es frecuente en GP, y la existencia de árboles como el de la Figura 36 prueban que se ha producido dicho efecto.

Dado que hemos desglosado el escenario completo en escenarios simples con el objetivo de encontrar en éstos, consecuentemente, comportamientos simples para combinarlos en el escenario completo, posiblemente habría sido positivo limitar mucho más la profundidad máxima (que estaba en el valor por defecto, 17) u optar por medidas más sofisticadas que añadieran algún tipo de penalización al *fitness* en función de la complejidad del árbol a evaluar.

7.5.2 Redundancia

En la línea del problema del *bloating*, nos hemos encontrado con el problema de la redundancia de árboles, es decir, árboles aparentemente distintos que, tras llevar a cabo un proceso de simplificación de sus expresiones, lógicas resultan ser idénticos.

Sin embargo, la redundancia no sólo es achacable al subconjunto de funciones de expresiones lógicas, sino que también es un problema de tipo semántico. Dado que no hacemos comprobaciones semánticas adicionales sobre la estructura del árbol en base al estado interno conocido del superviviente, es posible, por ejemplo, derivar subárboles como (*GoTo NearestExit*) o (*Attack (AgentLocation NearestEnemy*)), los cuales son totalmente equivalentes a un simple nodo *Wait* si el superviviente no conoce ninguna salida o no tiene enemigos a la vista, respectivamente. De manera similar, (*Attack NearestBoundary*) o (*GoTo NearestBoundary*), por ejemplo, son subárboles equivalentes cuando no hay enemigos en la casilla frontera más próxima.

También existe redundancia por el propio conjunto de funciones empleadas. Por ejemplo, (*Avoid RandBoundary*) o (*GoTo RandBoundary*) representan acciones completamente idénticas en cualquier caso. Otras como (*LookLeft*) o (*LookRight*), pese a que la acción física realizada

sobre el entorno es distinta, en realidad, dado que no existen actualmente más funciones en el conjunto completo que tengan en cuenta la posición relativa del superviviente con respecto a su entorno, y que incluso la propia orientación inicial del superviviente es aleatoria, su efecto es totalmente equivalente siempre que aparezca sólo una de ellas en el árbol.

Esta redundancia implica, entre otras cosas, una ampliación del espacio de búsqueda explorado respecto al espacio real de soluciones del problema, puesto que estamos considerando como distintos individuos que son el mismo en realidad. Esta característica, la presencia de genotipos redundantes que se manifiestan en un único fenotipo, no es algo malo, en general, en los algoritmos evolutivos, más allá de la pérdida de tiempo en evaluar el *fitness* de soluciones ya evaluadas. Sin embargo, como se verá en el siguiente apartado, es posible que en nuestro caso sí que sea un problema.

7.5.3 Ruido

Left4Sim es un entorno dinámico y aleatorio. Un mismo comportamiento puede suponer sobrevivir hasta el final de una simulación, y morir a los pocos turnos de la siguiente. Dado que precisamente nuestro objetivo está en determinar si el comportamiento hallado permite sobrevivir o escapar con ciertas garantías a un superviviente, cualquier medida de *fitness* que diseñemos va a tener que afrontar este indeterminismo o ruido.

Optamos por la solución posiblemente más frecuente en este tipo de problemas: Repetir y promediar. Así pues, en la última versión de los experimentos hasta 25 simulaciones por evaluación de *fitness* fueron necesarias para tener una idea aproximada del rendimiento del comportamiento de un equipo de supervivientes cualquiera. 100 se consideró como límite mínimo para determinar el mejor individuo de la evolución y 1000 para demostrar finalmente la calidad del mejor de todos ellos.

Incluso así, el *fitness,* y por consiguiente, el algoritmo de GP, no fueron capaces de encontrar soluciones que entendemos que son mejores. Para demostrar este hecho diseñamos manualmente el individuo cuyo árbol se muestra en la Figura 77. El comportamiento del superviviente representado hace uso de los ítems del entorno (i.e. botiquines y cajas de munición) según los necesita mientras busca la salida, priorizando el movimiento hacia ella una vez la ha divisado.

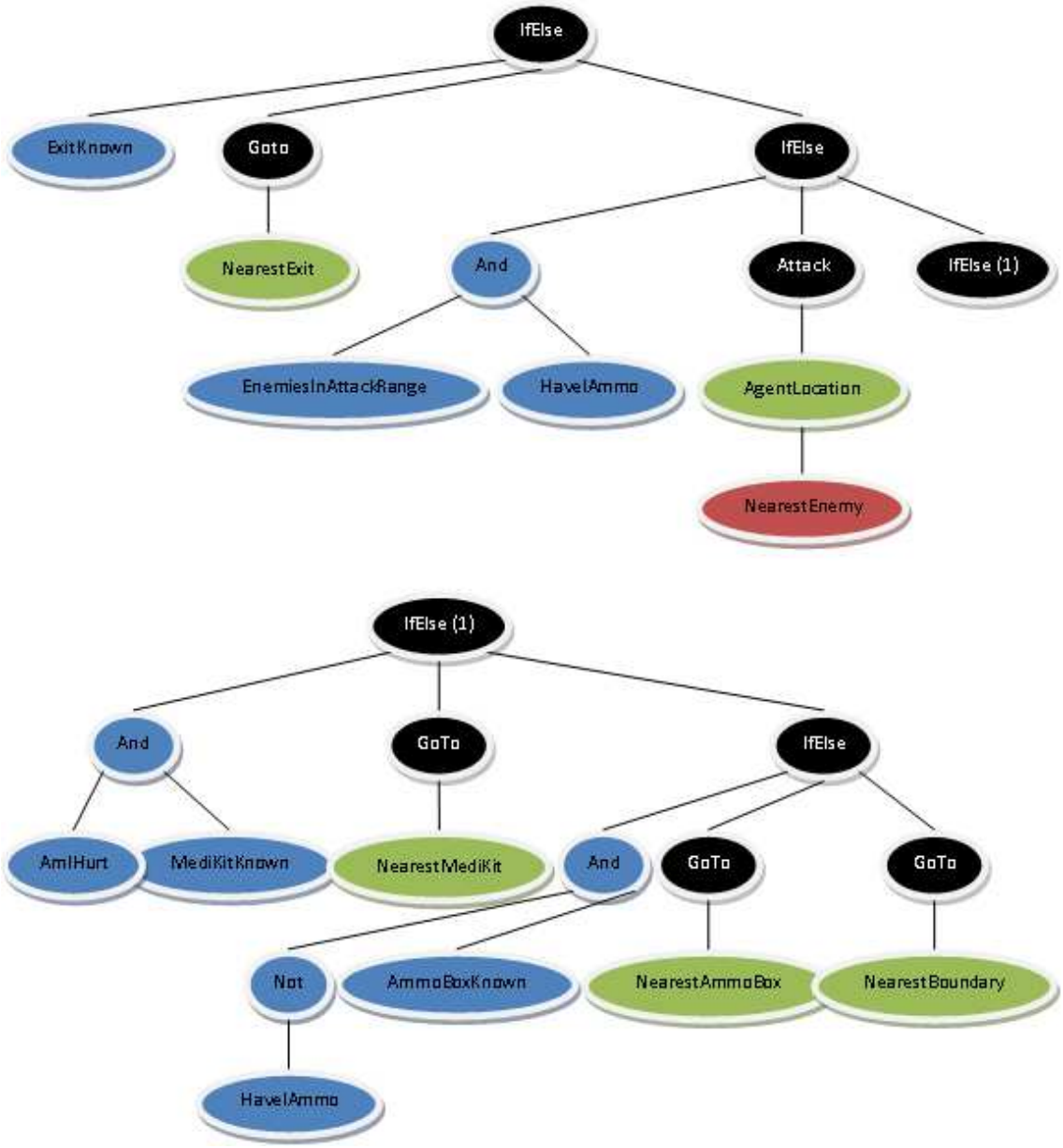


Figura 77. Árbol complejo diseñado manualmente. El superviviente representado usa los ítems del entorno según los necesita en su búsqueda de la salida, priorizando el movimiento hacia ella si ésta se encuentra a la vista

Este relativamente complejo árbol de profundidad = 7 diseñado por un usuario se evaluó 1000 veces de manera similar al mejor individuo encontrado tras una evolución completa. El histograma de los valores de *fitness* y estado final de superviviente encontrados para este individuo, así como para los mejores individuos resultantes del escenario básico y combinado para 1 superviviente, se detallan respectivamente en la Figura 78 y Figura 79.

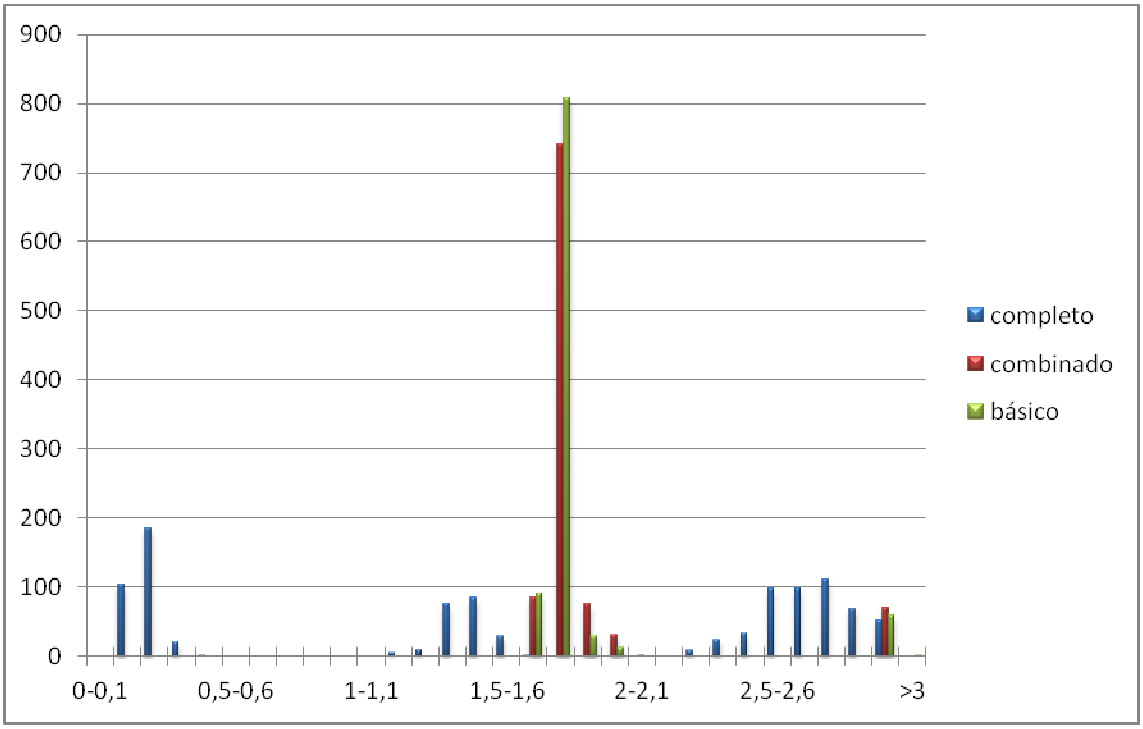


Figura 78. Histograma del *fitness* tras 1000 simulaciones del mejor individuo encontrado en los escenarios básico y combinado para 1 superviviente frente a un individuo "completo" diseñado a mano

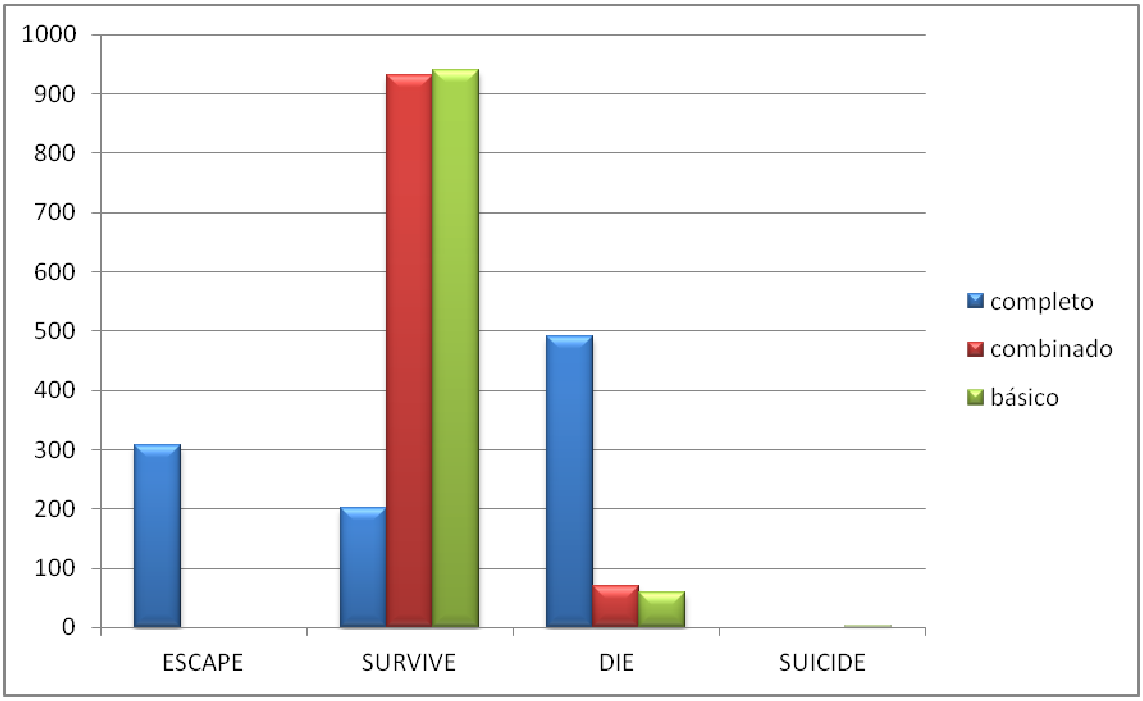


Figura 79. Histograma del estado final de superviviente tras 1000 simulaciones del mejor individuo encontrado en los escenarios básico y combinado para 1 superviviente frente a un individuo "completo" diseñado a mano

En las gráficas podemos observar las diferencias existentes en el desempeño del comportamiento de los individuos evolucionados, muy similar entre sí, con respecto al diseñado a mano. La principal de ellas es que el individuo completo llega a escapar en algo más

del 30% de las ocasiones frente a ninguna de los evolucionados. Por el contrario, dicho individuo completo muere en casi el 50% de las veces, frente a un escaso 6-7% de sus congéneres evolucionados. Dicho de otra manera, el individuo completo alcanza un buen (i.e. escapar) o aceptable (i.e. sobrevivir) comportamiento en algo más del 50% de las veces, frente al 93-94% de resultados exclusivamente aceptables conseguidos por los evolucionados.

Sin embargo, y como ya se describió previamente (ver capítulo 5.1), nuestro objetivo principal era el de evolucionar (i.e. ser capaces de aprender o generar) comportamientos para escapar, siendo la supervivencia menor en importancia, por lo que, lo siguiente que analizamos es si nuestra función de *fitness* es capaz de discernir entre estos 3 comportamientos cuál es el mejor. En la Tabla 9 se muestran los valores del promedio y la desviación típica del *fitness* de los individuos evolucionados y se contrastan con los del individuo completo.

Tabla 9. Resumen del promedio y desviación típica del *fitness* de los mejores individuos encontrados en los escenarios básico y combinado para 1 superviviente frente a un individuo "completo" diseñado a mano

	1 superviviente
Básico	1'7614[66] (0'3098)
Combinado	1'7795[66] (0'327)
Completo	1'6176 (1'0833)

A la vista de estos resultados podemos afirmar que nuestra función de *fitness* es capaz, tras una suficiente cantidad de repeticiones, de identificar qué comportamiento es mejor según nuestro criterio. Sin embargo, nuestro algoritmo de GP no ha sido capaz de converger hacia soluciones complejas similares a las del individuo completo, y esto puede ser debido a la alta variabilidad observada de los valores de *fitness* (i.e. 25 simulaciones quizás sigan siendo pocas), lo cual, unido a la alta redundancia de árboles simples ha provocado que se converja rápidamente (i.e. apenas 20 generaciones para el individuo del escenario básico o 5 en el combinado, como se veía en la Figura 23 y Figura 43, respectivamente) a soluciones subóptimas de este tipo, más fáciles de alcanzar, y, dada la alta redundancia existente, con mayor probabilidad de obtener un buen resultado promedio para las 25 simulaciones en alguna de sus formas redundantes.

[66] En realidad estos valores debieran ser algo mayores puesto que los *scores* de muertes están normalizados por el mejor individuo asesino de la población, el cual es peor que el propio individuo completo

8. Conclusiones

8.1 Contribuciones

Al cierre de este trabajo hemos desarrollado una plataforma de simulación muy similar al entorno de un videojuego FPS, excepto en lo relativo al aspecto gráfico, y hemos dado los primeros pasos hacia la obtención automática de patrones de comportamiento interesantes mediante una representación de árboles para los agentes del entorno.

Asimismo hemos tomado consciencia de la evolución del mundo del videojuego desde sus inicios hasta los tiempos actuales desde la perspectiva de la IA de los NPCs. Se han visto cuáles son las principales áreas de interés sobre las que los investigadores académicos de IA de videojuegos han depositado sus esfuerzos así como cuáles son las principales incorporaciones que los diseñadores comerciales de IA de videojuegos han introducido en éstos. Y lo que es más, se ha profundizado en alguna de los posibles motivos por los que, a la postre, la IA académica de videojuegos ha tenido tan poca repercusión hasta la fecha en títulos comerciales, pese a que, cada vez más, las diferencias entre ambos colectivos están disminuyendo y fruto de ello son algunas de las ideas innovadoras en videojuegos que hemos visto.

Desde un punto de vista puramente académico, hemos aplicado la GP en un área donde existe un enorme interés por la aplicación de técnicas de IA, como son los videojuegos, y bajo un entorno poco convencional como son los zombis vs humanos.

Acabamos el presente trabajo de fin de máster sin haber alcanzado todos los resultados que inicialmente nos habíamos planteado, pero sí habiendo conseguido profundizar tanto en el área de la simulación, los sistemas multiagente, la computación evolutiva, todo ello en el área de la programación de videojuegos. En cuanto a los dos colectivos primeros, tal vez esto sólo sea el comienzo de algo que puede ser interesante para la comunidad, en cuanto a los otros este trabajo nació y, de momento, seguirá como un trabajo de aplicación más que innovación, proponiendo un nuevo problema para su estudio y búsqueda de soluciones.

8.2 Trabajo futuro

Entre todas las posibles líneas de trabajo futuro, a continuación se describen algunas de las más prometedoras y que a mi juicio podrían originar trabajos de investigación relevantes, o, al menos, de interés, para la comunidad:

- Obtención de resultados con el simulador actual:
  - Revisión de la función de *fitness* (e.g.: remodelación para disminuir la alta variabilidad, enfoque multiobjetivo, incorporación de *scores* de equipo, modificación de la norma del *score* de muertes, etc.)
  - Repetición de los experimentos con distintas semillas aleatorias para comprobar la consistencia de las soluciones obtenidas así como la velocidad de la convergencia hacia las mismas
  - Pruebas con poblaciones más grandes
  - Pruebas con otros operadores genéticos[67]
  - Refinamiento del conjunto de funciones (e.g.: nuevas funciones de posición relativa, nuevas funciones cooperativas, nuevas funciones de información del entorno, etc.)
  - Reducción de la redundancia de árboles (e.g.: simplificación de árboles mediante expresiones regulares y/o LISP[68], penalización al *fitness* por complejidad del árbol, eliminación de ramas no usadas, etc.)
  - Pruebas con mapas con obstáculos
  - Pruebas con equipos de supervivientes más grandes y estudio de otras posibles formas de representación (e.g.: un único superviviente por individuo, un único árbol por equipo de supervivientes con funciones específicas de equipo, etc.) y evolución (e.g.: coevolución de distintas subespecies de superviviente, *clustering* con lógica difusa de comportamientos individuales para los escenarios simples, otras formas de combinación de comportamientos individuales en el escenario combinado, etc.) de los mismos
  - Pruebas con otros objetivos (e.g.: ajuste de parámetros de entorno para adecuar las tasas de escape/supervivencia a los valores deseados, modelización y adaptación a jugadores humanos, etc.)

- Mejora del entorno *Left4Sim*:
  - Posibilidad de control de supervivientes por jugadores humanos
  - Reducción de la complejidad temporal de la plataforma[69]
  - Aumento del nivel de paralelismo del proceso evolutivo a varias máquinas

[67] Se hicieron algunas pruebas con distintas tasas de mutación pero no lo suficientemente completas ni analizadas como para ser incluidas en el trabajo

[68] Se empezó a trabajar en ello pero no hubo tiempo para concluir el reductor LISP de expresiones

[69] Los tiempos de ejecución crecen desmesuradamente por factores como el nº de supervivientes, la amplitud de propagación y memoria de estímulos, el número máximo de pasos de simulación o el tamaño del mapa

    - Mejora del sistema de estímulos de los zombis (e.g. bloqueo por obstáculos, percepción por sentidos direccionales VS percepción por sentidos no direccionales, etc.)
    - Introducción de distintas armas con diferentes características
    - Introducción de zombis especiales

- Migración de las ideas exploradas al entorno *Pogamut*
- Otros estilos de FPS
- Otros géneros de videojuegos

9. Bibliografía

1. http://es.wikipedia.org/wiki/Software#Clasificaci.C3.B3n_del_software. [En línea]

2. **Darwin, Charles.** *On the Origin of Species by Means of Natural Selection, or the Preservation of Favored Races in the Struggle for Life.* London : John Murray, 1859.

3. **Holland, J. H.** *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* s.l. : University of Michigan Press, 1975.

4. **Goldberg, D.** Genetic Algorithms in Search, Optimization, and Machine Learning. s.l. : Addison-Wesley, 1989.

5. **Eiben, A.E. y Smith, J.E.** *Natural Computing Series. Introduction to Evolutionary Computing.* s.l. : Springer, 2007. ISBN: 978-3-540-40184-1.

6. **Koza, J. R.** *Genetic Programming: On the programming of Computers by Means of Natural Selection.* Cambrige, MA : MIT Press, 1992. ISBN-10: 0-262-11170-5.

7. **Poli, R., Langdon, W. B. y McPhee, N. F.** *A Field Guide to Genetic Programming.* s.l. : Lulu Enterprises, 2008. ISBN: 978-1-4092-0073-4.

8. **Fogel, L., Owens, A. y Walsh, M.** *Artificial Intelligence through Simulated Evolution.* New York : John Wiley & Sons, 1966. ISBN: 0-471-33250-X.

9. **Schwefel, H.-P.** *Evolutionsstrategie und numerishe Optimierung, Ph. D. thesis, Technical University of Berlin.* Berlin : s.n., 1975.

10. *Size control via size fair genetic operators in the PushGP genetic programming system.* **Crawford-Marks, Raphael y Spector, L.** s.l. : Morgan Kaufmann Publishers, 2002. GECCO '02 Proceedings of the Genetic and Evolutionary Computation Conference . págs. 733-739. ISBN: 1-55860-878-8.

11. *Size fair and homologous tree genetic programming crossovers.* **Langdon, W. B.** 1-2, Abril de 2000, Genetic Programming and Evolvable Machines, Vol. 1, págs. 95–119. ISSN 1389-2576.

12. *A comparison of bloat control methods for genetic programming.* **Luke, Sean y Panait, Liviu.** [ed.] MIT Press Cambridge. 3, Septiembre de 2006, Evolutionary Computation, Vol. 14, págs. 309-344.

13. *Strongly Typed Genetic Programming.* **Montana, D.** 1995. Evolutionary computation, 3(2). págs. 199–230.

14. *Grammatical Evolution.* **O'Neill, M. y Ryan, C.** 2001. IEEE Transactions on Evolutionary Computation, 5(4). págs. 349–358.

15. *Genetic Programming for Pedestrians.* **Banzhaf, W.** University of Illinois at Urbana-Champaign : Morgan Kaufmann, 1993. Proceedings of the 5th International Conference on Genetic Algorithms (ICGA-93). pág. 628.

16. **Miller, J. y Thomson, P.** Cartesian Genetic Programming. *Genetic Programming, Lecture Notes in Computer Science.* Heidelberg : Springer-Verlag, 2000, Vol. 1802, págs. 121–132.

17. *An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach.* **Miller, J.** 1999. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999). Vol. 2, págs. 1135–1142.

18. **Charles, Darryl, y otros.** Contemporary video game AI. *Biologically Inspired Artificial Intelligence for Computer Games.* 2008, págs. 1-11.

19. http://en.wikipedia.org/wiki/Video_game#History. [En línea]

20. [En línea] http://aigamedev.com/open/highlights/top-ai-games/.

21. *Human-level AI's Killer Application: Interactive Computer Games.* **Laird, John y VanLent, Michael.** ISSN: 0738-4602, 2000, Artificial Intelligence Magazine, Vol. 22(2), págs. 15-26.

22. *A Taxonomy of Video Games and AI.* **Gunn, E.A.A., Craenen, B.G.W. y Hart, E.** 2009. Adaptive and Emergent Behaviour and Complex Systems (AISB 2009).

23. *A survey on the need and use of AI in game agents.* **Yildirim, Sule y Stene, Sindre Berg.** 2008. Proceedings of the 2008 Spring simulation multiconference (SpringSim '08). págs. 124-131. ISBN:1-56555-319-5.

24. *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents.* **Franklin, Stan y Graesser, .Art.** Berlin : Springer Verlag, 1997. ECAI '96 Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages. págs. 21-35. ISBN:3-540-62507-0 .

25. **Rabin, S.** *AI game programming wisdom.* Hingham, MA : Charles River Media, 2002.

26. *Research directions for ai in computer games.* **Fairclough, Chris, y otros.** 2001. Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science. págs. 333-344.

27. *Game AI Revisited.* **Yannakakis, Geogios N.** New York, NY : ACM, 2012. Proceedings of the 9th conference on Computing Frontiers (CF '12). págs. 285-292. ISBN: 978-1-4503-1215-8.

28. *A Turing Test for Computer Game Bots.* **Hingston, P.** Edith Cowan Univ., Perth, WA, Australia : s.n., 2009. IEEE Transactions on Computational Intelligence and AI in Games. Vols. 1 , Issue: 3, págs. 169 - 186.

29. *Combining self organizing maps and multilayer perceptrons to learn bot-behaviour for a commercial game.* **Thurau, C., Bauckhage, C. y Sagerer, G.** 2003. GAME-ON. págs. 119–123.

30. *Learning human-like Movement Behavior for Computer Games.* **Thurau, C., Bauckhage, C. y Sagerer, G.** 2004. Proceedings of the International Conference on the Simulation of Adaptive Behavior. págs. 315-323.

31. *It Knows What You're Going To Do - Adding Anticipation to a Quakebot.* **Laird, John E.** New York : ACM, 2001. Proceedings of the fifth international conference on Autonomous agents (AGENTS '01). págs. 385-392 . ISBN:1-58113-326-X.

32. *Chuck Norris rocks!* **Cuadrado, Daniel y Saez, Yago.** 2009. CIG'09 Proceedings of the 5th international conference on Computational Intelligence and Games. págs. 69-74. ISBN: 978-1-4244-4814-2.

33. *Evolving the Cooperative Behaviour in UnrealTM Bots.* **Mora, A.M., y otros.** 2010. IEEE Symposium on Computational Intelligence and Games (CIG 2010). págs. 241-248. ISBN: 978-1-4244-6297-1.

34. *Evolving bot AI in Unreal.* **Mora, A., y otros.** s.l. : Springer-Verlag, 2010. EvoApplicatons'10 Proceedings of the 2010 international conference on Applications of Evolutionary Computation. Vol. I, págs. 171-180. ISBN:3-642-12238-8 978-3-642-12238-5.

35. *Agent Smith - Towards an Evolutionary Rule-Based Agent for Interactive Dynamic Games.* **Small, R.** 2009. IEEE Congress on Evolutionary Computation 2009 (CEC '09). págs. 660-666.

36. *Evolving Team Behaviours in Environments of Varying Difficulty.* **Doherty, Darren y O'Riordan, Colm.** 4, s.l. : Kluwer Academic Publishers Norwell, Abril de 2007, Artificial Intelligence Review, Vol. 27, págs. 223-244.

37. *Evolving Agent-Based Team Tactics for Combative Computer Games.* **Doherty, Darren y O'Riordan, Colm.** Belfast, Ireland : s.n., 2006. Proceedings of the 17th Irish Artificial Intelligence and Cognitive Science Conference. págs. 52-61.

38. *Evolving Multi-modal Behavior in NPCs.* **Schrum, Jacob y Miikkulainen, Risto.** 2009. IEEE Symposium on Computational Intelligence and Games (CIG 2009). págs. 325-332. ISBN: 978-1-4244-4815-9.

39. *Gathering and utilising domain knowledge in commercial computer games.* **Bakkes, Sander y Spronck, Pieter.** 2006. Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2006). págs. 41-48. ISSN: 1568-7805.

40. *Coevolution of role-based cooperation in Multi-Agent systems.* **Yong, C. y Miikkulainen, R.** 2010. IEEE Transactions on Autonomous Mental Development.

41. *Evolving Multimodal Networks for Multitask Games.* **Schrum, Jacob y Miikkulainen, Risto.** 2011. 2011 IEEE Conference on Computational Intelligence and Games (CIG). págs. 102-109 .

42. *Evolving Behaviour Trees for the Commercial Game DEFCON.* **Lim, Chong-U, Baumgarten, Robin y Colton, Simon.** s.l. : Springer-Verlag, 2010. EvoApplicatons'10 Proceedings of the 2010

international conference on Applications of Evolutionary Computation. Vol. I, págs. 100-110. ISBN: 3-642-12238-8 978-3-642-12238-5.

43. *Robust player imitation using multiobjective evolution.* **Hoorn, Niels van, y otros.** 2009. IEEE Congress on Evolutionary Computation (CEC '09). págs. 652–659. ISBN: 978-1-4244-2958-5.

44. *Learning to Overtake in TORCS.* **Loiacono, Daniele, y otros.** 2010. 2010 IEEE Congress on Evolutionary Computation (CEC). págs. 1-8. ISBN: 4244-6909-3.

45. *Using a Training Camp with Genetic Programming to Evolve Ms Pac-Man Agents.* **Alhejali, Atif M. y Lucas, Simon M.** 2011. IEEE Conference on Computational Intelligence and Games (CIG 2011). págs. 118-125. ISBN: 978-1-4577-0010-1.

46. *Towards Conscious-like Behavior in Computer Game Characters.* **Arrabales, Raúl, Ledezma, Agapito y Sanchis, Araceli.** 2009. CIG'09 Proceedings of the 5th international conference on Computational Intelligence and Games. págs. 217-224. ISBN: 978-1-4244-4814-2.

47. *A Model for Reliable Adaptive Game Intelligence.* **Spronck, Pieter.** 2005. Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation and Learning in Computer Games.

48. *Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation.* **Takagi, Hideyuki.** 9, 2001, Proceedings of the IEEE, Vol. 89, págs. 1275-1296. ISSN: 0018-9219.

49. *Real-Time Neuroevolution in the NERO Video Game.* **Stanley, Kenneth O., Bryant, Bobby D. y Miikkulainen, Risto.** 6, s.l. : IEEE Press Piscataway, 2005, IEEE Transactions on Evolutionary Computation, Vol. 9, págs. 653–668. ISSN: 1089-778.

50. **Gemrot, J., y otros.** Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. *Lecture Notes in Computer Science, Agents for Games and Simulations.* s.l. : Springer-Verlag, 2009, Vol. 5920, págs. 1-15.

51. *MASON: A Multi-Agent Simulation Environment.* **Luke, Sean, y otros.** 7, San Diego, CA : s.n., Julio de 2005, Simulation: Transactions of the society for Modeling and Simulation International, Vol. 81, págs. 517-527.

52. *Gamebots - A 3D Virtual World Test-Bed For Multi-Agents Research.* **Adobbati, Rogelio, y otros.** 2001. Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS.

53. *Multiagent Systems: A Survey from a Machine Learning Perspective.* **Stone, Peter y Veloso, Manuela M.** 3, 2000, Autonomous Robots, Vol. 8, págs. 345-383. ISSN: 1573-7527.

54. **Benda, M., Jagannathan, V. y Dodhiawala, R.** *On optimal cooperation of knowledge sources - an empirical investigation.* Boeing Advanced Technology Center, Boeing Computing Services. Seattle, Washington : s.n., 1986. Technical Report. BCS–G2010–28.

55. *Genre and game studies: toward a critical approach to video game genres.* **Apperley, Thomas H.** 1, s.l. : Sage Publications, Inc., Marzo de 2006, Simulation and Gaming - Symposium: Video games: Issues in research and learning, part 2, Vol. 37, págs. 6-23. ISSN:1046-8781.