1202 - MODELADO Y SIMULACIÓN DE SISTEMAS BIOLÓGICOS

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Antonio Cruz Vázquez
i dirigit per
Diego Javier Mostaccio Mancini
Bellaterra, 22 de juny de 2009

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Diego Javier Mostaccio Mancini

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Antonio Cruz Vázquez

I per tal que consti firma la present.

Signat: Diego Javier Mostaccio Mancini

Bellaterra, 22 de juny de 2009

1. Introducción

El objetivo de este capitulo de introducción es dar una visión global del proyecto, tanto a nivel particular como en lo que hace referencia al campo sobre el que va a tratar.

1.1 Motivación

La motivación principal de este proyecto ha sido la de la investigación y desarrollo en un campo que nunca antes había tratado, como es el de los sistemas biológicos utilizando modelos orientados al individuo.

Dentro de dicho campo, me resulta atractivo el poder simular el comportamiento que tienen los peces a la hora de desplazarse y ver como los resultados obtenidos son los mismos que los que se pueden encontrar en el mundo real (según [1] y [2]).

Como conclusión de mis motivaciones, decir que siempre he considerado muy interesante el mundo de la bioinformática y este proyecto me da la oportunidad de desarrollar un simulador basado en ese campo.

1.1.1 Características de los sistemas biológicos orientados al individuo

El presente proyecto tiene como temática principal los sistemas biológicos utilizando modelos orientados al individuo. Dichos sistemas, se basan en el comportamiento de cada uno de sus individuos.

Así pues, las dos principales características de los modelos orientados al individuo son que tienen en cuenta el comportamiento de individuo y que el comportamiento del grupo esta basado en las interacciones entre los individuos que forman el sistema.

Estos sistemas se diferencia de los orientados al grupo, en que estos últimos se basan en el comportamiento grupal de los elementos.

Entrando un poco más en detalle a lo que hace referencia a los sistemas biológicos orientados al individuo, se pueden ver estos como un sistema en el que el comportamiento de cada uno de los individuos que lo forman, provoca que se llegue a un cierto estado grupal.

Dentro de estos casos, se podría ver como el comportamiento unitario de un grupo de personas provoca un cierto comportamiento en las decisiones de donde colocar las salidas de emergencia de un edificio.

También se pueden usar estos sistemas para observar y simular el mundo animal. Por ejemplo se podría ver como un conjunto de individuos de la misma especie acaban formando un grupo cohesionado gracias al comportamiento individualizado de cada uno de sus miembros.

Este proyecto precisamente se va a basar en el estudio, simulación y forma de implementar el algoritmo de un conjunto de animales, concretamente en el de los peces.

1.1.2 Simulador del comportamiento de un banco de peces (fish schools)

En este punto se va a describir en líneas generales el funcionamiento del simulador de fish schools.

El proyecto consistirá en realizar un simulador en tres dimensiones de fish schools que reproduzca el movimiento real de los peces. Se podrá ver como a partir del comportamiento unitario de cada uno de los peces, se llega a conseguir un cierto estado grupal.

Este comportamiento de agrupación de los peces tiene como finalidad el protegerse del ataque de sus enemigos. Partiendo de esta idea, el simulador deberá comportarse de la misma manera que los peces lo hacen. Para ello, habrá que basarse en un estudio realizado por Huth y Wissel [1] y [2] en el que dan las bases del comportamiento de los peces y de la manera en la que acaban agrupándose formando los consiguientes bancos de peces.

En dicho estudio se puede ver como los peces se identifican por su posición y orientación. También se va a poder extraer de ese estudio, que los peces se caracterizan por tener 4 tipos de comportamiento (en cuanto a su movilidad) y que en base a ellos logran formar grupos muy cohesionados.

Dichos comportamientos se basan en la distancia a la que están los peces vecinos respecto a un determinado pez. Según dicha distancia van a existir los comportamientos de repulsión, de orientación paralela, de atracción y de búsqueda.

El comportamiento de repulsión aparece cuando hay el riesgo de que un pez choque con otro debido a que se encuentran muy cerca el uno del otro. En este caso, el pez cambiará su dirección a una que sea perpendicular a la del pez vecino.

En el caso de que la distancia entre un pez y su vecino sea pequeña (pero no tanto como para que choquen), aparecerá la orientación paralela. Esta reacción supone que el pez cambie su dirección igualándola a la que tiene su vecino.

Si la distancia entre el pez y el vecino ya es algo mayor (pero dentro del campo de visión del pez), aparecerá el comportamiento de atracción. Se trata de una reacción en la que el pez se acerca a su vecino haciendo que su dirección apunte a la posición que ocupa el vecino.

El último comportamiento será el de búsqueda, que se dará cuando no existan vecinos lo suficientemente cerca del pez o estos estén en su ángulo muerto de visión. Debido a que el pez no aprecia vecinos dentro de su campo visual, hará un cambio de dirección para intentar encontrarlos.

Como consecuencia de que todos los peces tendrán que comparar sus posiciones con las del resto de vecinos para saber cuales de ellos influenciarán finalmente en su movimiento, habrá n*n-1 comparaciones por cada iteración, que lo podemos considerar como un n^2 y por tanto un algoritmo de orden cuadrático.

A partir de la información extraída de la documentación de referencia, se deberán analizar los diferentes comportamientos de los peces y ver de que manera se puede llegar a pasar de su base teórica a la simulación final en un ordenador. Para ello, se deberá dividir el trabajo en varias fases.

La primera de ellas consistirá en traducir la información existente en la documentación de referencia, a su correspondiente representación matemática. Para ello, habrá que adaptar las explicaciones que aparecen para el modelo en dos dimensiones al correspondiente modelo en tres dimensiones.

Una vez desarrollada la representación matemática del modelo en tres dimensiones, se pasará a la fase correspondiente con implementación del modelo matemático. Seguidamente se realizará la tercera y última fase consistente en el desarrollo e implementación en si del simulador.

Lo primero será definir el espacio de simulación y las características de los peces. Se establecerá un espacio representado en tres dimensiones, cuyo tamaño será limitado pero que podrá ser configurado.

Los peces estarán definidos mediante su identificador, posición y dirección (velocidad). El número de peces también será acotado pero podrá ser configurado. De igual manera, se podrán configurar otros elementos como por ejemplo el número de iteraciones, o el número de peces.

Debido a que la complejidad del algoritmo es cuadrática, se decide pensar alguna manera que permita acelerar el tiempo de ejecución. Una vez analizadas las distintas posibilidades, se decide permitir que se pueda dividir el espacio total de simulación en partes (mediante configuración). De esta manera, la complejidad sigue siendo cuadrática pero con un número menor de elementos a tratar.

Aparte de las opciones de configuración ya comentadas, también se podrá establecer la manera en la que se distribuyan inicialmente los peces, permitiendo así simular varios escenarios.

Una vez definidas la base del simulador y las diferentes opciones de configuración del simulador, se realizará la implementación en lenguaje C++. Dicha implementación estará formada por una parte inicial que consistirá en la configuración del simulador, a partir de la lectura del fichero encargado de ello.

Seguidamente y en función del método de inicialización de los peces que se haya escogido, se realizará la consiguiente creación y colocación de los mismos en los lugares determinados del espacio de simulación.

Una vez colocados los peces, se pasará a lo que es en si el motor de la simulación, consistente en la comparación de cada uno de los peces con el resto y la consiguiente determinación de las nuevas posiciones y direcciones de los peces en función de las reacciones que se produzcan.

Para cada iteración, se escribirán los resultados de las posiciones y de las direcciones de los peces en varios ficheros. También existirá un fichero de estadísticas en donde quedará recogido el grado de homogeneidad de las direcciones y el grado de cohesión del grupo.

Finalmente, esos ficheros generados servirán para representar gráficamente la simulación realizada (mediante otro proceso independiente de este proyecto).

1.2 Objetivos

El objetivo principal será el de obtener un simulador con el que obtengamos los mismos resultados que los aparecidos en la documentación de Huth y Wissel ([1] y [2]) sobre el estudio de los movimientos de los peces.

Para ello, se deberán realizar varios pasos que permitan pasar del estudio de una bibliografía sobre el tema, al desarrollo del simulador. Además, habrá que marcar objetivos parciales para cada una de las fases del proyecto.

El primero de ellos consistirá en la lectura de los trabajos existentes [1] y [2] sobre el comportamiento de los peces. Será importante entender el funcionamiento y la base teórica de sus estudios para posteriormente poderla poner en práctica. También se deberá buscar otra documentación que pueda servir de ayuda para la realización del proyecto.

En este punto, el objetivo a marcar va a ser el asimilar correctamente toda la base teórica que posteriormente habrá que acabar implementando.

El segundo paso será el desarrollo del modelo matemático. Una vez leída y analizada con detenimiento la documentación de referencia, se deberán traducir los conceptos teóricos en fórmulas matemáticas con las que representar el modelo, teniendo en cuenta además, que deben ser fórmulas a utilizar en un espacio tridimensional.

En esta caso, el objetivo a marcar van a ser corroborar que las representaciones matemáticas son equivalentes a las expresiones aparecidas en la documentación de referencia [1] y [2].

Una vez establecido el modelo matemático, se deberá realizar la implementación del mismo. Para ello, habrá que adaptar las fórmulas de un formato teórico, a la representación definitiva que será la que se deberá usar en la implementación del simulador.

En este caso, el objetivo a marcar será nuevamente comprobar que las fórmulas implementadas se corresponden con la realidad, es decir que a partir de dichas fórmulas se obtienen los resultados mostrados en la documentación de referencia [1] y [2].

Finalmente se realizará la implementación del simulador, empezando primero por su análisis y desarrollo. Dentro de la parte de implementación se integrará el sistema matemático implementado previamente y que servirá como elemento fundamental en el correcto funcionamiento del simulador.

Una vez realizada la implementación, habrá que comprobar que los resultados obtenidos con el simulador son los mismos que los que se esperaban inicialmente, es decir los aparecidos en la documentación de referencia [1] y [2].

1.3 Organización de la memoria del proyecto

La memoria estará organizada en 7 puntos más la bibliografía y los anexos que aparecerán al final de la misma.

El primero de los puntos será precisamente este, que servirá de introducción a la misma y en donde aparecerán las características de los sistemas biológicos orientados al individuo, tanto a nivel general como particularizando con el fish schools. También aparecerán las motivaciones y objetivos a cumplir en el proyecto.

A continuación aparecerá la viabilidad y la planificación del proyecto, que servirán para realizar un estudio previo de si se puede llevar a cabo la realización del mismo. En este capitulo habrá también un apartado dedicado al "fish schools" para explicar sus características. Otros apartados que integran este capitulo son el estudio del estado arte y en análisis de riesgos.

El siguiente punto será el correspondiente al análisis funcional y a los requerimientos del proyecto. Aquí se hará la descripción de la funcionalidad del programa, así como de las consideraciones a tener en cuenta para la realización del simulador. Como parte final del capitulo aparecerá el diagrama de bloques del simulador.

A continuación se llegará al diseño. Aquí habrá que realizar un estudio de cómo llevar a cabo el desarrollo. Los apartados que aparecerán en este capitulo serán los

correspondientes a las estructuras de datos, la comunicación entre módulos y el diagrama de clases del simulador.

La implementación será el siguiente punto, y será el lugar en el que se detallará la manera en la que se ha realizado el desarrollo del simulador. Aquí se describirá el algoritmo implementado.

En el sexto punto aparecerán las pruebas a las que se someterá el simulador una vez finalizado. Habrá una primera prueba con la que comparar si los resultados del simulador se corresponden con los aparecidos en la documentación de referencia. A parte de dicha prueba, existirán otras para verificar el correcto funcionamiento del simulador.

Las conclusiones del proyecto serán el séptimo y último punto y nos van a servir para descrobir los objetivos conseguidos. Además se indicarán otros posibles trabajos futuros o variaciones que se podrían desarrollar a partir de la realización de este proyecto.

Finalmente la memoria terminará con la bibliografía utilizada para la realización de este proyecto, además de un apartado extra con los anexos.

2. Viabilidad y planificación

En este segundo capitulo se van a tratar aspectos previos al proyecto como son por ejemplo el estudio de viabilidad o la planificación de las tareas.

2.1 Finalidad del proyecto

La finalidad del proyecto es el desarrollo de un simulador de sistemas biológicos utilizando modelos orientados al individuo.

Gracias a este tipo de proyecto se podrá observar el comportamiento de los diferentes sistemas biológicos de una manera rápida y sin la necesidad de experimentar directamente con ellos. Todos los cálculos se realizarán en una computadora lo suficientemente potente como para simular los diferentes movimientos posibles que pueden desarrollar las especies en estudio.

Los beneficios que se podrán obtener una vez conocido y simulado el comportamiento del modelo a estudiar será el determinar con antelación que pasos o acciones realizar para conseguir que la especie que se vaya a tratar actúe de una manera u otra.

Desde otro punto de vista, el proyecto también va a servir para analizar si los resultados obtenidos con el simulador se ajustan a los existentes en el mundo real.

2.2 Propuesta de proyecto

El presente trabajo se basa en la representación en tres dimensiones de las características del movimiento de los peces en un momento dado. Estas características del movimiento vendrán dadas por la dirección que tome el pez y ésta a su vez vendrá determinada por las reacciones provocadas por los vecinos.

Cada uno de los pasos de la simulación marcarán el movimiento y por tanto nuevas posiciones de los peces. Respecto a las características físicas (tamaño y velocidad) de los peces, estas serán igual para todos.

Habrá 3 tipos de comportamientos, atracción, orientación paralela y repulsión. Cada uno de estos comportamientos vendrán determinados por la distancia y ángulo en la que se encuentren sus peces vecinos. Además de esos 3 tipos de comportamiento, existirá un cuarto (búsqueda) que se dará cuando no hayan vecinos cerca o bien no los pueda ver porque están en su ángulo muerto de visión.

La simulación consistirá en seleccionar 4 vecinos de cada pez (según Huth y Wissel [1] y [2]) y calcular la reacción del pez para cada uno de sus vecinos. A partir de esos cálculos y del peso en que cada uno de sus vecinos repercuta sobre el pez en cuestión, se obtendrá la nueva posición del pez.

Este proyecto sólo se encargará de realizar los cálculos de las posiciones de los peces pero no de la muestra de estos por pantalla.

2.3 Recursos materiales y plataforma de desarrollo

Para realizar el proyecto se dispondrá de un PC convencional que dadas sus limitaciones, no permitirá hacer simulaciones con un gran número de individuos, ya que el tiempo empleado para ello sería demasiado grande. Hay que tener en cuenta que la complejidad del algoritmo del modelo es cuadrática ($O(n^2)$), donde n es el número de peces de la simulación.

Como lenguaje de programación, se utilizará el lenguaje C++, ya que se trata de un lenguaje propicio para trabajar con los peces como si fueran objetos, además de ser bastante accesible y conocido.

El sistema operativo sobre el que se desarrollará y ejecutará el simulador será Windows, debido a que es el más difundido y usado en el mundo.

Como documentación científica de referencia sobre el comportamiento de los peces y que servirá de base para desarrollar tanto el modelo como el simulador, se dispondrá de los trabajos realizados por Andreas Huth y Christian Wissel [1] y [2].

Finalmente se dispondrá de internet para realizar las investigaciones y consultas necesarias para el desarrollo del proyecto.

2.4 Planificación de tareas

FASES	TAREAS	PERIODOS
1	**Investigación inicial y lectura de documentación**	**01-11-08 al 30-11-08**
1.1	Lectura de documentación de referencia de los autores Huth y Wissel	
2	**Estudio de viabilidad del proyecto**	**01-12-08 al 12-12-08**
2.1	Definir la finalidad, propuesta y objetivos del proyecto. También se indicará una primera planificación, así como la metodología o el análisis de riesgos.	
3	**Profundización en la investigación y pre-análisis de cómo desarrollar el proyecto**	**13-12-08 al 07-01-09**
3.1	Asimilación de la documentación de referencia, así como búsqueda de nueva documentación relacionada con el proyecto y que pueda servir de ayuda para el desarrollo del mismo	
3.2	Aumentar conocimientos en los lenguajes de programación a utilizar en el proyecto (C++)	
4	**Análisis funcional y requerimientos para el desarrollo del proyecto**	**08-01-09 al 16-01-09**
4.1	Definir detalladamente la funcionalidad del programa una vez finalizado, estudio de los casos de uso del sistema y diagramas de funcionamiento/bloques	
4.2	Establecer el modelo matemático (fórmulas y cálculos) que implementaremos posteriormente en el desarrollo del simulador	
5	**Diseño del proyecto**	**17-01-09 al 06-02-09**
5.1	Definir el plan de implementación a realizar, así como la estructura modular	
5.2	Realización del algoritmo, poniendo los pasos a seguir para la realización del desarrollo del simulador. Realización del diagrama de flujo, indicando de forma gráfica, los pasos del algoritmo. Realización del pseudocódigo del simulador, que posteriormente pasaremos al lenguaje de programación a utilizar para el desarrollo del mismo.	

6	**Implementación del proyecto**	**07-02-09 al 20-03-09**
6.1	Definir la implementación, la interfase de usuario, así como el prototipo de la primera versión.	
6.2	Desarrollo del simulador en lenguaje C++.	
7	**Evaluación del proyecto**	**21-03-09 al 17-04-09**
7.1	Definir el plan de pruebas a realizar, la metodología de las mismas así como los resultados esperados	
7.2	Realizar las pruebas establecidas consistiendo en primer lugar en pruebas unitarias para posteriormente realizar pruebas globales. Realizaremos finalmente varias simulaciones variando el número de elementos e iteraciones. Las herramientas a utilizar serán el simulador en si realizado en C++.	
8	**Análisis de resultados del proyecto**	**18-04-09 al 08-05-09**
8.1	Definiremos la funcionalidad, los resultados de las pruebas así como la realización del esquema definitivo de la memoria	
8.2	En esta fase, analizaremos los resultados obtenidos en las pruebas finales realizadas en la fase anterior, que nos servirán para determinar el comportamiento final del simulador. Terminaremos de dejar listo las versión definitiva del simulador.	
9	**Finalización del proyecto**	**09-05-09 al 30-06-09**
9.1	Repaso, revisión y finalización de las tarea que pudieran quedar pendiente, así como preparación más exhaustiva de la presentación del proyecto	

2.5 Estudio del estado del arte

Este apartado se va a dividir en 2 partes, correspondientes al nivel de detalle. Primero se empezará tratando los diferentes tipos de simuladores existentes a nivel general, para luego terminar con los simuladores biológicos.

En primer lugar habrá que nombrar algunos de los diferentes tipos de simuladores existentes actualmente, así como algunos de sus productos:

- Simulador de carreras: Se puede conducir un automóvil, motocicleta, camión, etc. Ejemplos: rFactor, GTR, GT Legends.
- Simulador de vuelo o de aviones: Permite dominar el mundo de la aviación y pilotar aviones, helicópteros... Ejemplo: Microsoft Flight Simulator, X-Plane

- Simulador de trenes: Permite controlar un tren. Ejemplo: Microsoft Train Simulator, Trainz , BVE Trainsim .
- Simulador de redes: Permite simular redes. Ejemplo: Omnet++, ns2.
- Simulador biológico: Permite simular elementos vivos, como plantas, animales, personas, etc...

Dentro de estos, se pueden encontrar aquellos simuladores que permiten realizar diagnósticos clínicos sobre pacientes virtuales o bien para practicar casos muy complejos, preparando al médico para cuando se encuentre con una situación real. Un ejemplo de este sería el simulador clínico “Mediteca”. A estos simuladores se les conoce particularmente como simuladores quirúrgicos.

A continuación se abordarán los simuladores biológicos, poniendo ejemplos de diferentes trabajos que tratan este tipo de simuladores.

Como se podrá comprobar la mayoría de los simuladores biológicos están basados en el campo de la medicina. Gracias a ellos, se está consiguiendo un mayor aprendizaje de aquellas situaciones complejas que se pudieran producir o bien como medida de aprendizaje de situaciones reales pero sin la existencia de riesgo sobre el paciente.

Algunos ejemplos de este tipo de simuladores los podemos encontrar a continuación:

- Simuladores para la enseñanza de la cirugía endoscópica en México: [16].
- Desarrollo de modelos biológicos inanimados en urología, con los que se pueda lograr un método de enseñanza que permita desplegar las destrezas quirúrgicas: [17].

Como ejemplo del desarrollo e interés que están suscitando los simuladores biológicos (sobre todo en el campo de la salud), se mostrarán a continuación los diferentes simuladores desarrollados en la facultad de ciencias de la salud de la Universidad de Carabobo (Venezuela): [18].

Dentro de la lista de proyectos aparecidos, se podrían destacar el denominado "simBio", simulador de sistemas biológicos tales como células cardíacas, células epiteliales, y las células β pancreáticas. Está escrito en Java, usa XML y puede resolver ecuaciones diferenciales ordinarias: [19].

Seguidamente se encuentra un trabajo que ya es más propio de la simulación biológica. Se trata de un autómata celular que simula el crecimiento de varias especies de plantas en un terreno: [20].

También cabe destacar la presencia de simuladores y utilidades para la ingeniería química e ingeniería medioambiental: [21].

Otra vertiente relacionada serían los juegos de simulación biológica, en donde el jugador controla ecosistemas donde los organismos pueden evolucionar o bien en los que el jugador toma el papel del animal como si fuera un juego de rol. Algunos de estos juegos serían el "SimLife", el "Darwinbots" o el "Wildlife Tycoon".

2.6 Fish schools

Se define como fish schools el modelo que describe el comportamiento de un banco de peces tomando como base el movimiento de cada individuo y la interacción con el resto del sistema [3].

La característica más relevante del conjunto de fish schools es su alto grado de cohesión en los movimientos grupales y que esta cohesión la tienen sin la presencia de un líder. Estudios relacionados sobre el tema [1] y [2], llegaron a la conclusión que el movimiento conjunto de los peces viene determinado por la elección individual de cada uno de ellos, y que esta depende de la localización física en la que se encuentren sus vecinos.

A continuación se enunciarán las características que determinan los movimientos de los peces en función de sus vecinos.

En primer lugar, se tendrán en cuenta los peces vecinos que influenciarán en la decisión del pez. Dicha elección vendrá dada por aquellos que se encuentren a una distancia próxima y cuya posición sea visible para el pez en cuestión.

Si existen más de 4 peces que cumplen las condiciones de cercanía y grado de visión, el pez escogerá aquellos 4 vecinos que o bien estén más próximos o bien estén más en frente a su posición (según los estudios realizados, parece ser que esta segunda opción es la que genera una mayor aproximación a la realidad). La prioridad por frontalidad se basa en que primero se escogen aquellos peces que están por delante y más enfrente, para luego escoger los que están detrás y más alejados del centro. Para la opción del cálculo por distancia, se escogerán primero los peces que estén más cerca, independientemente de si están por delante o por detrás.

A continuación, se mostrará la relación de distancias y ángulos sobre los que el pez determinará cuales son sus vecinos y que por tanto acabarán influenciando en su movimiento (figura 3.1).

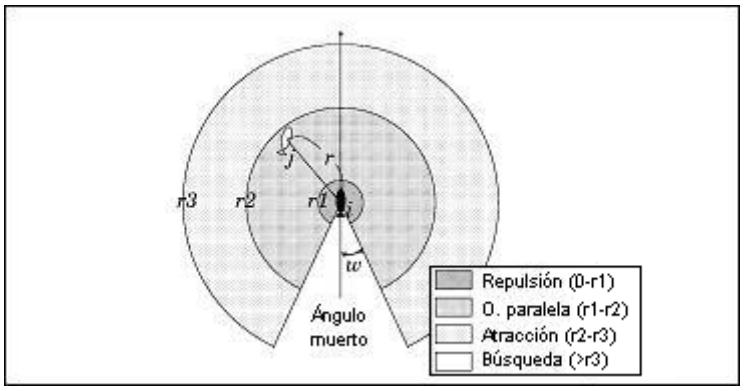


Figura 3.1: Rangos de los tipos de comportamiento

Como se puede apreciar en la figura 3.1, existirá un circulo (el mayor de todos) que marcará la zona de influencia del pez, tanto en distancia como en ángulo de visión.

Dentro del mayor de los círculos habrá otros 2 internos que servirán para dividir la zona de influencia del pez en 3 partes.

Teniendo en cuenta que el pez está situado justo en el centro del circulo, se tendrá como primera zona aquella que está situada a una menor distancia que el radio r1 (tamaño del radio correspondiente a la mitad de un pez) y por tanto muy próxima al pez. Esta área es el llamada de repulsión ya que ante la gran proximidad de un pez vecino, el pez tiende a apartarse para no colisionar. Para ello, el pez realizará un giro que le permita estar perpendicular a la dirección del vecino con el que podría colisionar.

La siguiente zona en distancia, será la formada entre el radio r1 y el r2 (tamaño del radio correspondiente al doble del tamaño de un pez). Esta área es llamada de orientación paralela, ya que el pez tiende a ponerse en una posición paralela al pez vecino que esta localizado en esa zona. Así pues, el pez toma el mismo ángulo que su vecino.

La última zona de influencia y por tanto la más alejada, será la formada entre el radio r2 y el radio r3 (tamaño del radio correspondiente a 5 veces el tamaño del pez). Esta área es llamada de atracción, ya que el pez tiende a acercarse a su vecino. Por tanto el pez cambiará su dirección para ir en busca de su vecino.

Finalmente estará la reacción de búsqueda que se encuentra situada más allá del radio r3 y que se dará cuando el pez no perciba vecinos a su alrededor o bien porque están demasiado lejos o bien porque se encuentran en su ángulo muerto de visión (situado justo detrás y con un ángulo de 30 grados). En este caso, el pez lo que hará será modificar su actual dirección con la intención de localizar a otros peces.

En función de las influencias de los vecinos de cada uno de los peces, se obtendrá la nueva reacción del pez y consecuentemente su nueva dirección y posición. El cálculo final de la dirección vendrá dado como la media de las direcciones resultantes de las reacciones habidas con los diferentes vecinos.

Seguidamente se va a mostrar como es el modelo 2D del fish schools.

Cada pez estará representado por dos valores, uno que indicará su posición y otro que mostrará su orientación.

La posición vendrá representada por los valores de las coordenadas X e Y, mientras que la orientación corresponderá con el ángulo que marque la dirección que tiene el pez en ese momento.

A continuación se mostrará como se calculan las diferentes reacciones para el modelo en dos dimensiones ([1] y [2]).

En primer lugar se indicará la nomenclatura utilizada en las fórmulas:

$\boldsymbol{v}^{o}_{i}, \boldsymbol{x}^{o}_{i}$: Orientación y posición del pez.

$\boldsymbol{v}^{o}_{j}, \boldsymbol{x}^{o}_{j}$: Orientación y posición de su vecino.

Y seguidamente se pasará a analizar las reacciones:

- Repulsión:
  Para evitar que un pez colisione con otro, realiza un giro que le permita estar perpendicular respecto a la dirección de su vecino. Esto se traduce en realizar un giro que sitúe su orientación con un ángulo de 90 grados respecto a la orientación del pez con el que pude chocar. De las posibles opciones para conseguir su objetivo, el pez escogerá aquella que le suponga dar un menor giro.

  Por tanto, la reacción de repulsión se transformará en un ángulo de rotación β_{ij} determinado por:

$$\beta_{ij} = \min \{\angle (\boldsymbol{v}^{o}_{i}, \boldsymbol{v}^{o}_{j}) \pm 90°\}$$

En la figura 3.1 se puede apreciar la reacción de repulsión entre el pez **i** y su vecino **j** dando como resultado la nueva dirección β_{ij} para el pez **i**.

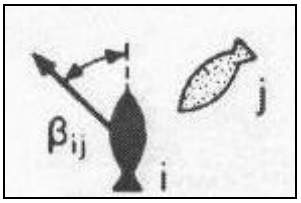


Figura 3.1: Reacción de repulsión para el modelo en dos dimensiones [2]

- Orientación paralela:
  En el caso de que exista algún vecino que esté a una cierta distancia próxima al pez, éste pasará a desplazarse en la misma dirección que lo hace su vecino.

  Así pues el pez girará el siguiente ángulo (β_{ij}):

$$\beta_{ij} = \angle (v^{\circ}_{i}, v^{\circ}_{j})$$

  En la figura 3.2 aparece el pez **i**, su vecino **j** y el ángulo (β_{ij}) resultante de la acción de orientación paralela.

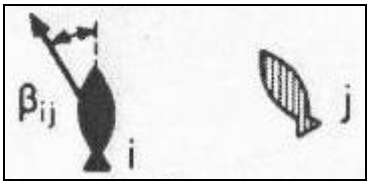


Figura 3.2: Reacción de orientación paralela para el modelo en dos dimensiones [2]

- Atracción:
  Cuando un pez está alejado del resto del grupo, este tiende a desplazarse hacia el grupo.

  De esta manera, el nuevo ángulo será el resultado de la diferencia de posición entre su vecino y el propio pez (β_{ij}):

$$\beta_{ij} = \angle (v^{\circ}_{i}, x_j - x_i)$$

En la figura 3.3 se puede apreciar la reacción de atracción entre el pez **i** y su vecino **j**. Como consecuencia de dicha reacción, se producirá el cambio de dirección β_{ij} para el pez **i**.

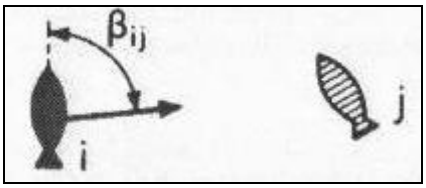


Figura 3.3: Reacción de atracción para el modelo en dos dimensiones [2]

- Búsqueda:
  Si finalmente el vecino está demasiado lejos o en el ángulo muerto, el pez al no percibir ningún vecino realizará una acción de búsqueda para intentar encontrar a algún vecino.

  Así pues, el pez realizará un giro con un ángulo aleatorio (β_{ij}):

$$\beta_{ij} = \textbf{Aleatorio ([-180° , 180°])}$$

En líneas generales, el cálculo de la reacción final del pez vendrá determinada por la media de los ángulos de giro provocados por los diferentes vecinos, según la reacción aplicada para cada uno de ellos.

En el ejemplo de la figura 3.4 se puede ver el caso en que un pez (**i**) tiene dos vecinos (**j=1** y **j=2**) y como la reacción final se calcularía como la media de los ángulos provocados por los dos vecinos.

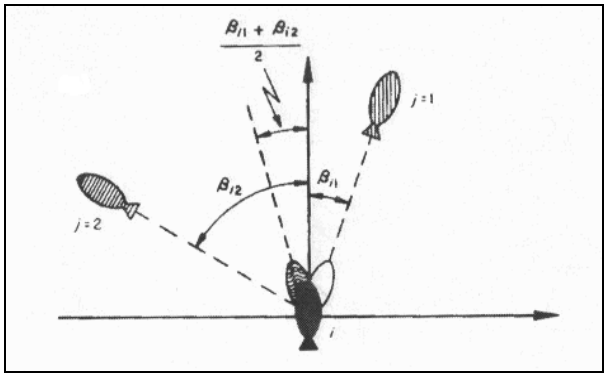


Figura 3.4: Cálculo de la reacción final del pez para el modelo en dos dimensiones [2]

Para poder caracterizar a los fish schools, se dispondrá de tres índices:

- Polarización *(polarization)*: Promedio de la desviación de los peces respecto al movimiento del grupo. A menor valor, mayor homogeneidad en las direcciones de los peces.
- Extensión *(expanse)*: Grado de cercanía de los peces al centro de masas. Indica cuanto de cerca están todos los peces entre si. A menor valor, menor distancia respecto al centro de masas y por tanto mayor cohesión.
- NND *(Nearest Neighbor Distance)*: Promedio de las distancias respecto al vecino más cercano. Nos sirve para calcular el grado de cohesión en el grupo.

2.7 Análisis de riesgo

En este apartado, se van a tratar los posibles imprevistos que pudieran aparecer, así como las medidas a adoptar para hacerles frente.

El principal riesgo serían los imprevistos que se podrían tener al desarrollar el proyecto en un lenguaje concreto de programación, ya que podría darse el caso de tener problemas para conseguir algunos de los objetivos marcados debido a sus limitaciones y características.

Para evitar este riesgo, se procederá a realizar un estudio de todas los posibles elementos a necesitar para realizar la implementación, para posteriormente comprobar si el lenguaje en cuestión las tiene. En caso de que las tenga, se procederá normalmente con la implementación en ese lenguaje. Si por el contrario dicho lenguaje tuviera ciertas limitaciones, se debería realizar el estudio de ver que lenguaje ofrece las herramientas que se necesitan para la realización del proyecto.

Otro riesgo que se podría presentar, sería el de que por algún motivo inesperado no se consiguieran los resultados esperados, retrasando con ello la planificación establecida. Para este caso, la solución consistiría en comprobar paso por paso donde puede estar el error, intentando en lo posible una vez solucionado el problema, volver a los tiempos planificados en un principio.

Finalmente se tendrán que contemplar los riesgos informáticos como pueden ser la pérdida fortuita de datos, que en este caso podría ser el código fuente del simulador. Para minimizar el impacto de los mismos, se procederá a realizar copias periódicas de la información en diferentes dispositivos de almacenamiento.

2.8 Viabilidad del proyecto

En este punto se desglosará la viabilidad del proyecto. Para ello, se procede a su división en 3 puntos, según esta sea a nivel de recursos, de tiempo o de conocimientos disponibles:

1. Viabilidad dependiente del tiempo disponible: Según la planificación existente para la realización de este proyecto (2.4), se puede afirmar que se dispone del suficiente tiempo como para llevarlo a cabo. En principio se iría realizando y siguiendo de manera progresiva para detectar posibles dificultades a tiempo y para que de esta manera, se pueda disponer de cierto margen al final para acabar lo que pudiera quedar pendiente.
2. Viabilidad dependiente de los recursos científicos y de investigación disponibles: Se dispone de la documentación necesaria para el desarrollo del proyecto. Concretamente se trata de unos trabajos realizados por dos científicos en los que se muestra de manera científica y matemática el desplazamiento de los peces [1] y [2]. A partir de esta documentación y con la consiguiente investigación, se podría llevar a cabo el desarrollo teórico del proyecto.
3. Viabilidad dependiente de los recursos hardware-software disponibles: Finalmente, en lo que se refiere al hardware, se dispondrá de un PC con el que realizar el proyecto. Dicho PC se utilizará para llevar a cabo el desarrollo, las pruebas y la ejecución del mismo. Para la parte del software, se usará el lenguaje de programación C++.

Una vez analizados los 3 aspectos, podemos concluir afirmando que el proyecto es considerado viable.

3. Análisis funcional y requerimientos

En este capitulo se mostrarán los requerimientos, así como las consideraciones generales a tener en cuenta para el desarrollo del proyecto.

3.1 Requerimientos

Los requerimientos del simulador son los siguientes:

- Realizar un simulador biológico orientado al individuo que simule el movimiento de los peces.
- Los resultados obtenidos con el simulador deberán ser los mismos que los existentes en la documentación de referencia [1] y [2].
- Se configurarán elementos del simulador como por ejemplo el número de iteraciones del simulador, el número de peces o las dimensiones del espacio de simulación. También se podrá establecer la manera en la que los peces se distribuirán inicialmente por el elemento de configuración. Se usará un fichero para indicar dicha configuración.
- El espacio de simulación será en tres dimensiones y el tamaño del mismo será finito.
- El simulador estará compuesto por un conjunto limitado de peces.
- Los atributos de los peces serán su posición y su dirección, que estarán representados mediante vectores. El vector correspondiente a la dirección estará normalizado para poder ser visualizado con un entorno gráfico.
- El comportamiento de los peces será el definido en la documentación [1] y [2].
- El modelo original de representación será en dos dimensiones [1] y [2] y tendrá que ser por tanto transformado a tres dimensiones.
- Se generarán tantas iteraciones como se deseen.
- Para cada una de las iteraciones del simulador, se escribirán en un conjunto de ficheros los resultados de las posiciones y velocidades de los peces. También se irán guardando las estadísticas del simulador (polarización y grado de cohesión).

3.2 Análisis funcional

En la realización del análisis funcional se van a tener en cuenta tanto los requerimientos establecidos, como posibles mejoras que puedan servir de valor agregado para el proyecto.

Así pues, se va a desarrollar un simulador de fish schools basado en el comportamiento unitario que tienen los peces al desplazarse. Para ello, se dispondrá de un espacio de simulación y de un conjunto de peces con las características indicadas en los requerimientos. Los detalles a tener en cuenta del espacio de simulación aparecerán en el punto 3.2.1, mientras que las correspondientes a los peces se encontrarán en el punto 3.2.2.

El simulador podrá configurarse para poder realizar la simulación de varios escenarios. Los detalles de configuración del simulador serán tratados en el punto 3.2.4.

Los peces van a tener 2 atributos con los que se podrán controlar sus movimientos. El primero de ellos será la posición y servirá para saber el lugar que ocupa el pez. El segundo atributo será el correspondiente a la dirección, con el que se podrá saber la orientación que lleva el pez. En función de dichos valores y de los que tengan los vecinos de alrededor, se podrán calcular las nuevas posiciones y direcciones de los peces.

Las características de los movimientos que tendrán, serán las mismas que las indicadas en la documentación de referencia [1] y [2]. Se tendrá que tener en cuenta que según los requerimientos, el modelo a desarrollar será en tres dimensiones, mientras que el modelo original aparecido en la documentación de referencia [1] y [2] está en dos dimensiones. Por tanto, será necesario realizar un estudio de cómo realizar la transformación del modelo 2D al 3D. Dicha transformación se encuentra detallada en el punto 3.2.3.

Una vez calculados los nuevos valores de los peces, se deberán escribir en diferentes ficheros para que posteriormente puedan ser visualizados en un entorno

grafico (independiente a este proyecto). Habrá un fichero de posiciones, uno de velocidades (direcciones) y uno de estadísticas para cada iteración.

3.2.1 Análisis de consideraciones a tener en cuenta con el espacio de simulación

Debido a que las dimensiones del espacio van a ser finitas, se va a tener que decidir que acciones tomar cuando los peces lleguen a esos limites.

Se han analizado varias opciones, como por ejemplo que si un pez sale de los limites de alguno de los tres planos, aparezca por el lado contrario del mismo plano, pero al final se ha decidido que cuando un pez alcance algunos de los limites del espacio de simulación se produzca un efecto de reflexión en su dirección.

Otro aspecto importante a tener en cuenta va a ser como conseguir buenos tiempos de ejecución. Para ello, se va a partir de la situación básica del simulador consistente en un único espacio en donde están distribuidos todos los peces.

El hecho de que haya un único espacio, implica que no se sepa la distancia a la que se encuentra un vecino de un determinado pez y por tanto se tenga que comparar cada pez con cada uno de los vecinos, para determinar cuales de estos intervienen en la reacción final del pez. Este tratamiento supondría una complejidad de orden cuadrático ($O(n^2)$). Como consecuencia, se analiza si podría existir algún tipo de variante para reducir el tiempo de ejecución.

Partiendo de esa idea inicial, se pensó en realizar algún cambio en la estructura interna que hiciera que el simulador fuera más rápido. Esa modificación pasaba por dividir el espacio de simulación en subgrupos para que así el número de comprobaciones entre peces se redujera a solo aquellos que ocupan la misma zona, junto a aquellos otros que estuviesen en cuadrantes limítrofes.

De esta manera, sólo se estarán haciendo comparaciones con los vecinos que realmente si pueden afectar a la reacción final (ya que se encuentran cerca del pez),

consiguiendo además que el orden cuadrático sea en función del número de elementos que componen cada uno de las particiones y por tanto que el tiempo de ejecución sea menor.

3.2.2 Análisis de consideraciones a tener en cuenta con los peces

En este subapartado se van a analizar la manera en que se guardarán las posiciones de los peces. Hay que tener en cuenta que la representación final de las mismas se realizará mediante números naturales (un pez estará en una posición concreta, no podrá estar parte si y parte no), pero en cambio internamente si que se deberá pensar la forma en la que tratar los valores.

Se ha escogido finalmente el guardar las posiciones internamente en formato decimal para no perder la precisión en los cálculos y consecuentemente reflejar de una manera más real la posición actual del pez. La manera en la que se pasará del valor interno decimal al valor entero de representación final será mediante el redondeo del mismo.

Otra consideración a tener en cuenta en el simulador será el trato que se le de a los peces que se encuentren tapados por otros. En este caso se ha decido que si un pez esta tapado por otro, no se tenga en cuenta. Así, si más de un vecino se encuentra en exactamente el mismo ángulo de visión de un determinado pez, sólo se escogerá aquel que esté a una distancia menor de este.

3.2.3 Adaptación del modelo original fish schools de dos dimensiones al de tres dimensiones

En este punto se tratará la manera de adaptar el modelo en 2 dimensiones visto en el apartado 2.6 (fish schools) al de 3 dimensiones.

Para el cado del modelo en 3D (al igual que en el caso del 2D) se seguirán teniendo como atributos del pez la posición y la orientación, pero en esta ocasión se hará de diferente manera el tratamiento de los mismos.

Para el caso de la posición, se utilizará un vector de tres coordenadas (x,y,z) mientas que para el caso de la orientación se usará un vector tridimensional en donde estará guardada la orientación que va a tener el pez (también llamado vector director).

Seguidamente se mostrará el funcionamiento de las reacciones para el modelo en tres dimensiones.

- Repulsión:
  Para la versión en 3D también se tendrá que buscar la dirección perpendicular al vecino para que el pez no choque, pero en este caso será algo bastante más complicado que para 2D, ya que aparece un tercer plano a tratar (coordenada Z). Esta tercera coordenada implicará tener múltiples direcciones perpendiculares, por lo que se deberá encontrar un vector director cuyo nuevo valor sea además aquel que suponga el menor giro posible para el pez.

- Orientación paralela:
  En este caso, al igual que en la versión 2D, el pez adquirirá la dirección que lleve el vecino. Esto supondrá que el vector director del pez pase a contener los mismos valores que el del vecino.

- Atracción:
  Como en el caso del método de dos dimensiones, se deberá calcular la nueva dirección como la diferencia de posiciones entre el vecino y el pez. Por tanto, la nueva dirección del vector director será la diferencia entre el contenido del vector de posiciones del vecino y el vector de posiciones del pez.

- Búsqueda:
  Para este último caso, también se deberá buscar una nueva dirección con lo que el vector director pasará a tener unos nuevos valores que serán totalmente aleatorios.

Para realizar el cálculo de la reacción final del pez en tres dimensiones, se deberá sumar (y posteriormente normalizar) el contenido de los vectores directores resultantes para cada una de las reacciones provocadas por los vecinos.

Una vez que se tenga la nueva dirección, se calculará consiguiente la posición que acabará ocupando el pez.

3.2.4 Detalle de las posibilidades de configuración del simulador

En este punto se detallarán los elementos del simulador que podrán ser configurados:

- Número de peces: Se indicará el número de peces con los que se realizará la simulación. Hay que tener en cuenta que a mayor número de peces el tiempo de proceso será mayor (ya que la complejidad es cuadrática y por tanto el crecimiento de tiempo también será notorio), aunque se dispondrá de una simulación más rica en cuanto a resultados. Si por el contrario utilizamos pocos peces, el resultado será justo lo contrario.

- Resolución del espacio de simulación: Se indicarán las dimensiones y por tanto las coordenadas de las posiciones por las que podrán moverse los peces. Debido a que es una simulación en tres dimensiones, se deberá indicar el valor para las tres coordenadas. Deberán ser valores entre 1 y n.

- Número de iteraciones: Número de pasos de simulación que se quieren simular. Evidentemente, con un mayor número de iteraciones y por tanto de tiempo, se podrá ver como progresivamente los peces tienden a juntarse. Si por el contrario, se lanza el proceso con pocas iteraciones, es posible que los resultados sean confusos por no dar tiempo a que los peces se encuentren entre si para acabar juntándose finalmente entre ellos. En todo caso, lo que si que se podrán observar serán patrones de comportamiento de los peces que lleven a la cohesión del grupo.

- Disposición inicial de los peces: Se podrá elegir la manera en que los peces se sitúen inicialmente tanto en posición como para ciertos casos en velocidad (dirección).

  Las maneras posibles serán las siguientes:

  - Distribución aleatoria en cualquier posición: Los peces se colocarán aleatoriamente distribuidos a lo largo del espacio de simulación que se

ha dimensionado anteriormente. La velocidad (dirección) también se calculará de forma aleatoria.

- Distribución agrupada en una determinada zona según porcentaje sobre el espacio total: Este método se utilizará para que el grupo de peces se limite a una cierta zona, delimitado por el porcentaje a ocupar respecto el total. Para ello se determinará el porcentaje que se quiere ocupar sobre el total de la dimensión y la localización que tendrá dentro del espacio de simulación. De esta manera por ejemplo, se podrán tener concentrados a todos los peces alrededor del centro del espacio de simulación o en una esquina y ver en esos casos como se comportan. Tanto el cálculo concreto de la posición como la velocidad se calcularán aleatoriamente, pero siempre teniendo en cuenta la configuración establecida. Los valores del porcentaje y de la situación se indicarán en el fichero de configuración del simulador.
- Distribución agrupada en una determinada zona según separación entre peces: Este método se utilizará para que el grupo de peces se limite a una cierta zona siguiendo además una distribución en la que la separación entre peces sea una concreta. Para ello se determinará la separación que se quiere que haya entre cada pez y la localización que tendrá dentro del espacio de simulación. De esta manera se podrá obligar a que inicialmente exista una cierta reacción entre los peces que desemboque posteriormente en un determinado comportamiento del grupo. Tanto la posición como la velocidad se calcularán aleatoriamente, pero siempre teniendo en cuenta la configuración establecida. Los valores de la separación entre peces y de la situación se indicarán en el fichero de configuración.
- Determinada por fichero de entrada: Se utilizará esta opción para aquellos casos en los que no se quiere que se calculen posiciones ni velocidades aleatoriamente, sino que lo que se quiere es ver el comportamiento del simulador para una determinada situación inicial. El fichero de entrada será un fichero de texto en donde se indicará en su primera línea el dimensionado del modelo a cargar, mientras que en las siguientes se indicará tanto la posición como la velocidad de cada

uno de los peces. El nombre y directorio del fichero a cargar se informarán mediante el fichero de configuración del simulador.

- Número de vecinos: Se indicará el número de vecinos que el pez tenga en cuenta para determinar su reacción. Por defecto el valor será 4, ya que es el propicio y correcto (según los estudios de Huth y Wissel [1] y [2]).

- Método de elección de vecinos: A la hora de escoger los vecinos que se tendrán en cuenta para el cálculo de la reacción, se podrá hacer de dos manera distintas. En principio y por defecto se tendrá en cuenta la prioridad de la frontalidad, es decir la de escoger aquellos vecinos que estén más en frente de un determinado pez. La otra opción de la que se dispone es la de que se haga la elección mediante la prioridad de distancia, es decir de escoger aquellos vecinos que estén más cerca de un determinado pez.

- Dimensionado de los radios de actuación y del ángulo muerto: Se podrán configurar tanto el tamaño de los radios de actuación como del ángulo muerto. En un principio y por defecto serán de 0,5 para la repulsión, 2 para la orientación paralela y 5 para la atracción. En el caso del ángulo muerto será de 30.

- Tamaño de los peces: Se indicará el tamaño que tendrán los peces. En un principio el valor será de 3.

- Partición del espacio: Se podrá indicar como se quiere que internamente se distribuyan los peces para conseguir mejorar el tiempo de ejecución del simulador. Inicialmente y por defecto habrá un único espacio en donde estarán todos los peces distribuidos. Para casos en los que haya muchos peces y el espacio de simulación sea grande se deberán hacer particiones del espacio para que el tiempo de ejecución no sea demasiado elevado, tal y como ha quedado explicado en el apartado 3.2.1. Para realizar las particiones, se tendrá que indicar el número de partes en las que se quiere dividir el espacio de simulación. Se tendrá que hacer para cada una de las dimensiones del espacio (x, y, z). De esta manera, cada una de las particiones actuará de forma más o menos independiente con lo que se conseguirán mejores tiempos de ejecución.

3.3 Criterios en la parametrización de la simulación

Los criterios a tener en cuenta vendrán dados por la velocidad y por la complejidad de la simulación.

Si se quiere realizar la simulación con un gran número de peces, será necesario un ordenador bastante potente para que el tiempo de simulación no sea demasiado elevado (debido a la complejidad cuadrática de la simulación).

En todo caso, el hecho de tener un equipo potente permitirá realizar los cálculos de una manera más rápida y por tanto acelerar el tiempo de la simulación.

Si por el contrario, se dispone de un equipo no demasiado potente, se tendrán que hacer simulaciones con pocos elementos, o bien esperar bastante rato a que termine. En esta caso, la velocidad también se resentirá ya que será mayor el tiempo que tardará el simulador en realizar los cálculos.

3.4 Herramientas prácticas a utilizar

Para la realización del proyecto se va a utilizar una herramienta que va a servir para la obtención de los resultados de las pruebas mediante un fichero de video.

La herramienta consistirá en un script en unix que genere una imagen (fichero de tipo .png) con la representación de la posición de los peces (marcados como puntos) sobre el plano cartesiano tridimensional para cada una de las iteraciones (mediante la herramienta “gnuplot”). Es decir, si se tiene una simulación en la que hay 100 iteraciones, el script en unix generará 100 imágenes de tipo .png mostrando la distribución de los peces sobre el plano para cada una de esas iteraciones. Una vez que el script ha creado esas imágenes, las convertirá a tipo video (mediante la herramienta mencoder), generando un fichero .avi en donde se podrá ver la simulación como si fuera una animación.

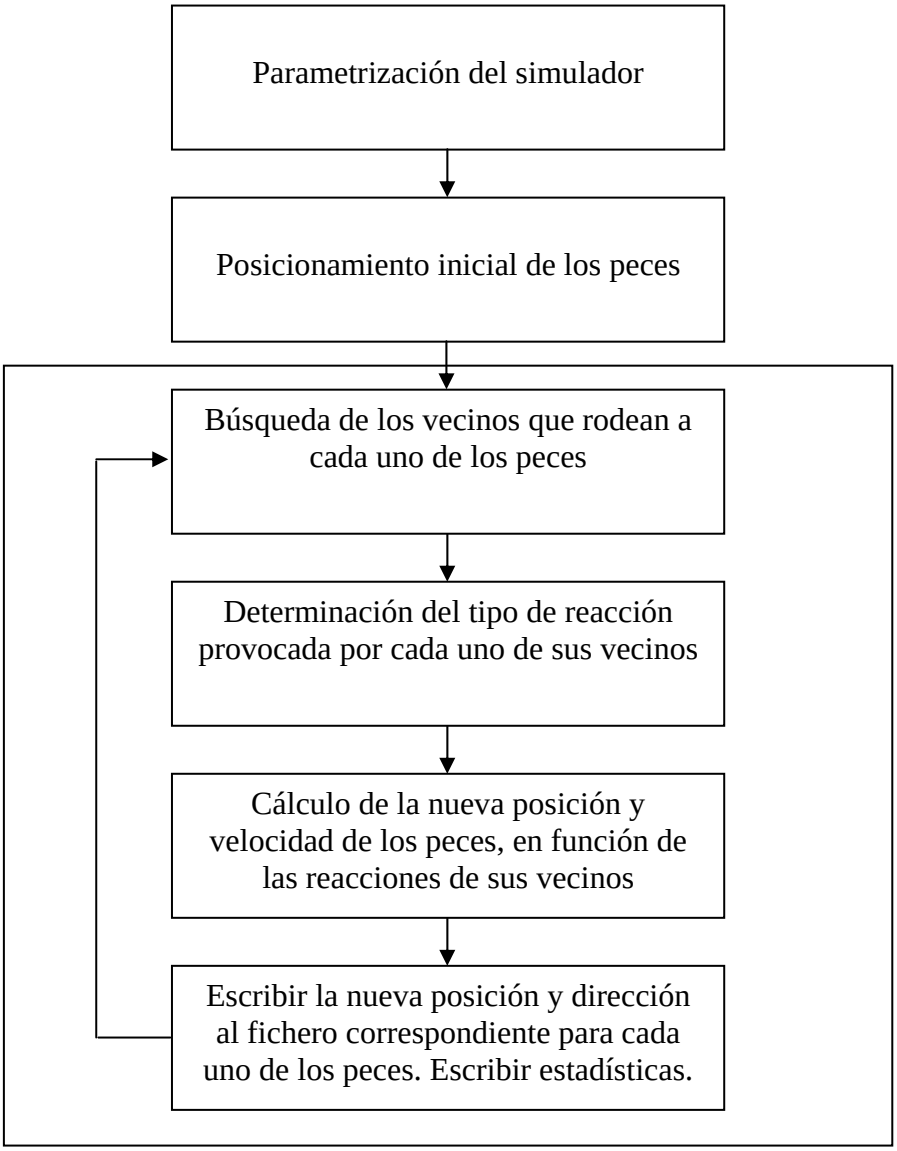
3.5 Estudios de los casos de uso del sistema

El hecho de que el propósito del proyecto sea orientado a temas de investigación más que al uso del mismo por agentes externos, hace que la interacción entre el usuario (actor) y el sistema se limite a la configuración de la simulación. En todo caso, será importante dicha configuración porque de ella dependerá el buen funcionamiento del simulador.

Así pues, el único caso de uso del sistema será la parametrización inicial por parte del usuario de los valores que se vayan a emplear para la ejecución de la simulación.

Una vez introducidos esos valores, el programa funcionará autónomamente realizando la simulación, sin la necesidad de interacción con el usuario.

3.6 Diagrama de bloques



4. Diseño

En este capitulo se indicarán todas los detalles correspondientes al diseño del simulador. Entre ellos, aparecerán las estructuras de datos utilizadas y la estructura modular.

4.1 Estructura modular

La estructura modular se podrá dividir en 3 partes:

1. Inicializador del simulador:
    - Creación de la matriz que servirá como divisor en partes del espacio de simulación: Se creará una estructura estática tridimensional para particionar el espacio de simulación en un determinado número de partes y de esta manera poder minimizar el tiempo necesario para el cálculo de vecinos de un determinado pez.
    - Cálculo de las posiciones iniciales (aleatorias / condicionadas): Módulo encargado de calcular las posiciones y direcciones iniciales de todos los peces. Podrán ser o bien aleatorias (escogiéndose también que porcentaje de distribución y en que posición del espacio), o bien siguiendo algún patrón para comprobar como actúa el simulador.
    - Posicionamiento en el cubo correspondiente del espacio de simulación según la posición en la que se encuentre el pez: Se crearán elementos tipo “pez” que serán referenciados por la posición de la matriz que les correspondan.
2. Simulador:
    - Obtención de los vecinos de un determinado pez: A partir de cada una de las listas y para cada uno de los peces, calculará la distancia y seleccionará los peces que sean vecinos (en principio hasta 4) según la prioridad elegida (en principio de frontalidad). El módulo

devolverá una lista con la identificación de los 4 vecinos seleccionados.

- Obtener la reacción para cada uno de los vecinos: Según la posición que ocupe cada uno de los peces vecinos, se obtendrá una reacción asociada a ese pez que se traducirá en una nueva velocidad (dirección) para el pez origen.
- Cálculo de la reacción final del pez: A partir de todas las afectaciones de los diferentes vecinos, se calcularán los nuevos valores de dirección y posición, teniendo en cuenta que serán fórmulas para tratar coordenadas en tres dimensiones.

3. Comunes:
    - Cálculo y gestión del cuadrante (cubo) de localización de cada pez: A partir de la posición que ocupe el pez y de la definición de la matriz, se calculará la posición que le corresponderá dentro de la misma. A continuación, se gestionará la lista de peces por zona, eliminando el pez de la zona que ocupe y añadiéndolo en la nueva (siempre y cuando haya un cambio de zona del pez).
    - Cálculo del grado de cohesión y del grado de dirección: Se calculará el grado de cohesión del fish schools, para determinar lo juntos o no que están los peces entre si y el grado de similitud de sus direcciónes (polarización).
    - Envío a fichero de los nuevos valores (dirección, posición e identificador del pez) y grado de cohesión (extensión/*expanse*) y polarización: Se enviarán a los correspondientes ficheros los datos de los peces y las estadísticas de la simulación para cada iteración.

La comunicación entre módulos será mediante envío y recepción de parámetros. Cada uno de los módulos enviará los parámetros para un determinado pez y el resultado del cálculo será devuelto para que en función de este, se actúe de una manera u otra. Se trabajará normalmente con listas, que serán enviadas y devueltas por los módulos una vez realizadas las operaciones correspondientes.

4.2 Estructuras de datos

La estructura general se fundamenta en tener un vector de listas dinámicas y que estas a su vez tengan elementos de tipo “pez”. Este tipo de estructura está destinada a que se pueda dividir el espacio en tantas partes como se quieran y de esta forma, agilizar el tiempo de ejecución. Cada una de las posiciones del vector representará a cada una de las particiones en las que se ha dividido el espacio de simulación (se indiciará en el fichero de configuración).

Una vez definido el vector (con un tamaño igual al número de particiones), serán referenciadas de cada una de sus posiciones las listas dinámicas de elementos de tipo pez que indicaran que peces están asociados a una parte concreta del espacio de simulación.

Es decir, todos los peces que estén en la misma lista dinámica de una determinada posición del vector serán peces que compartan la misma partición (ya que están próximos entre si) y que se tendrán que tener en cuenta para ver si son vecinos. A parte de estos posibles vecinos, también se tendrá que buscar en cuadrantes colindantes por si algún otro pez pudiera ser candidato a vecino.

El hecho de particionar el espacio de simulación, supondrá que se tenga que comparar cada uno de los peces con menos vecinos de alrededor ya que solo se fijarán en los que estén próximos a el.

Si no se realiza ningún tipo de partición, es decir que se trata todo el espacio de simulación de forma unitaria, existirá una única posición de vector y en esa posición estará la lista con todos los peces.

En la figura 4.1 se muestra un vector de 5 posiciones (serían 5 particiones del espacio) y de cada una de las posiciones es referenciada la lista de peces que tendrá tantos elementos como peces haya en cada partición.

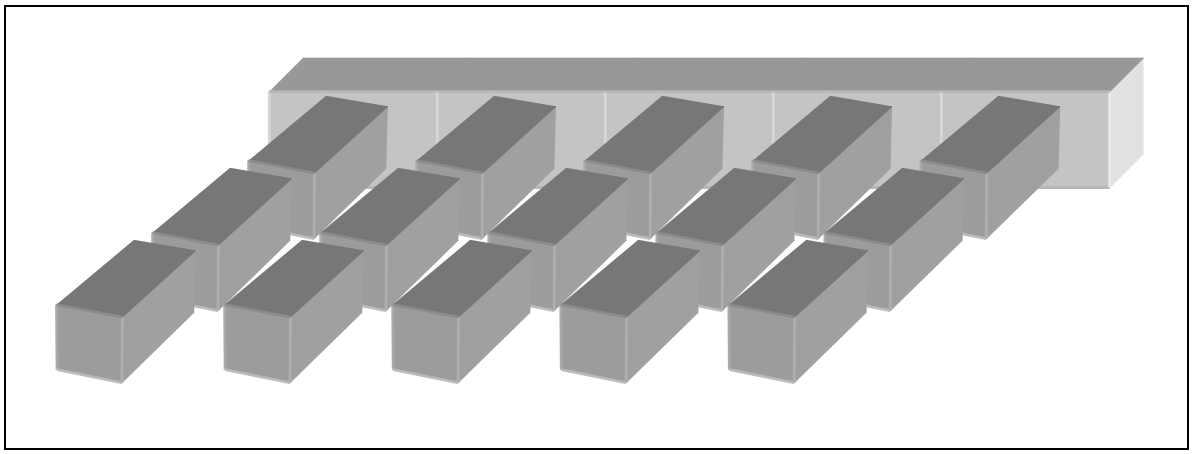


Figura 4.1: Esquema de la estructura de datos del simulador

Para llevar a cabo la estructura de los datos, se pasará a describir a continuación las clases de que está compuesto el simulador.

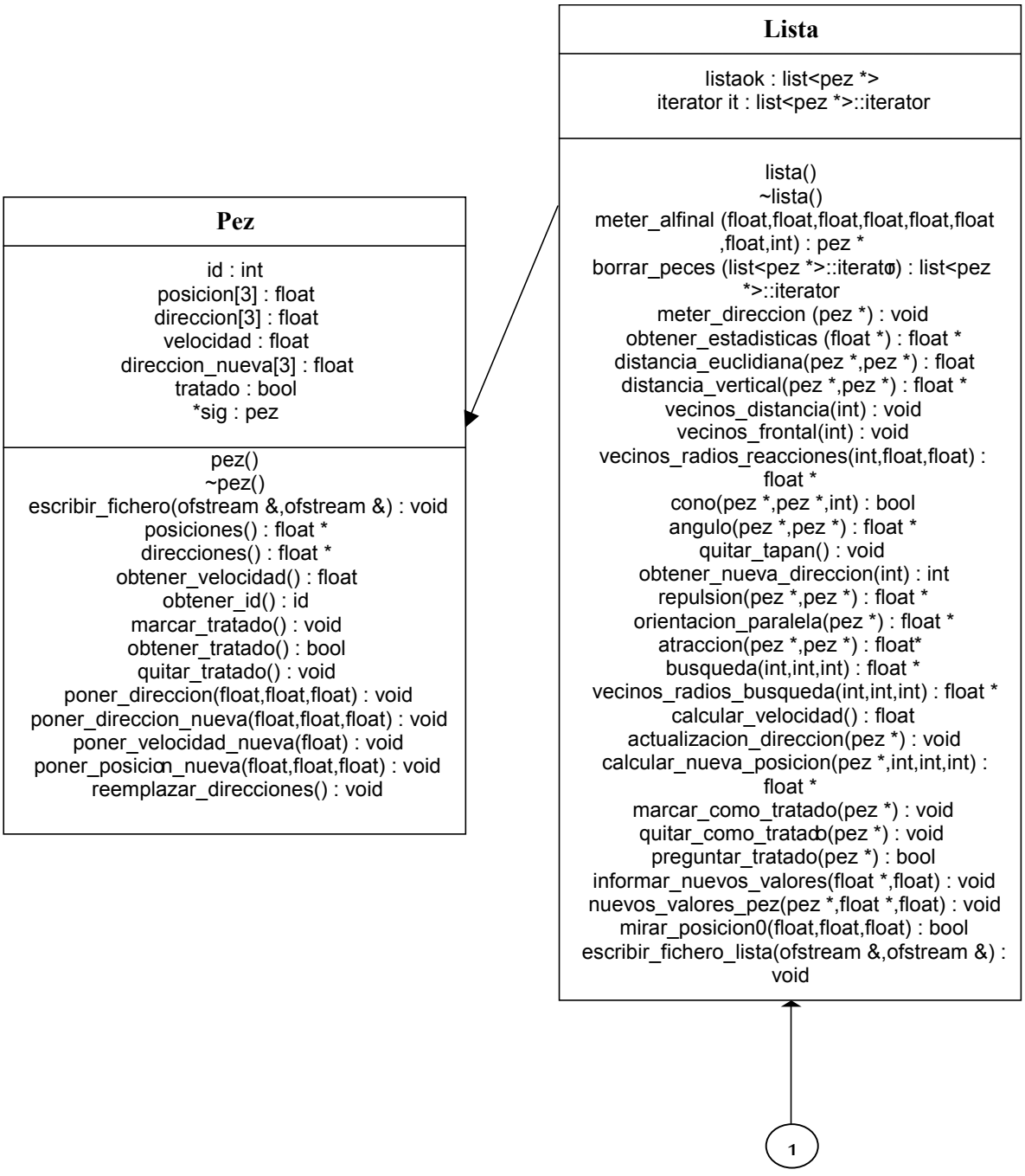
Se dispondrá de la clase pez (se llamará pez) que contendrá la información de todos los peces. Cada elemento de tipo pez contendrá su posición, velocidad e identificador, así como otros atributos auxiliares.

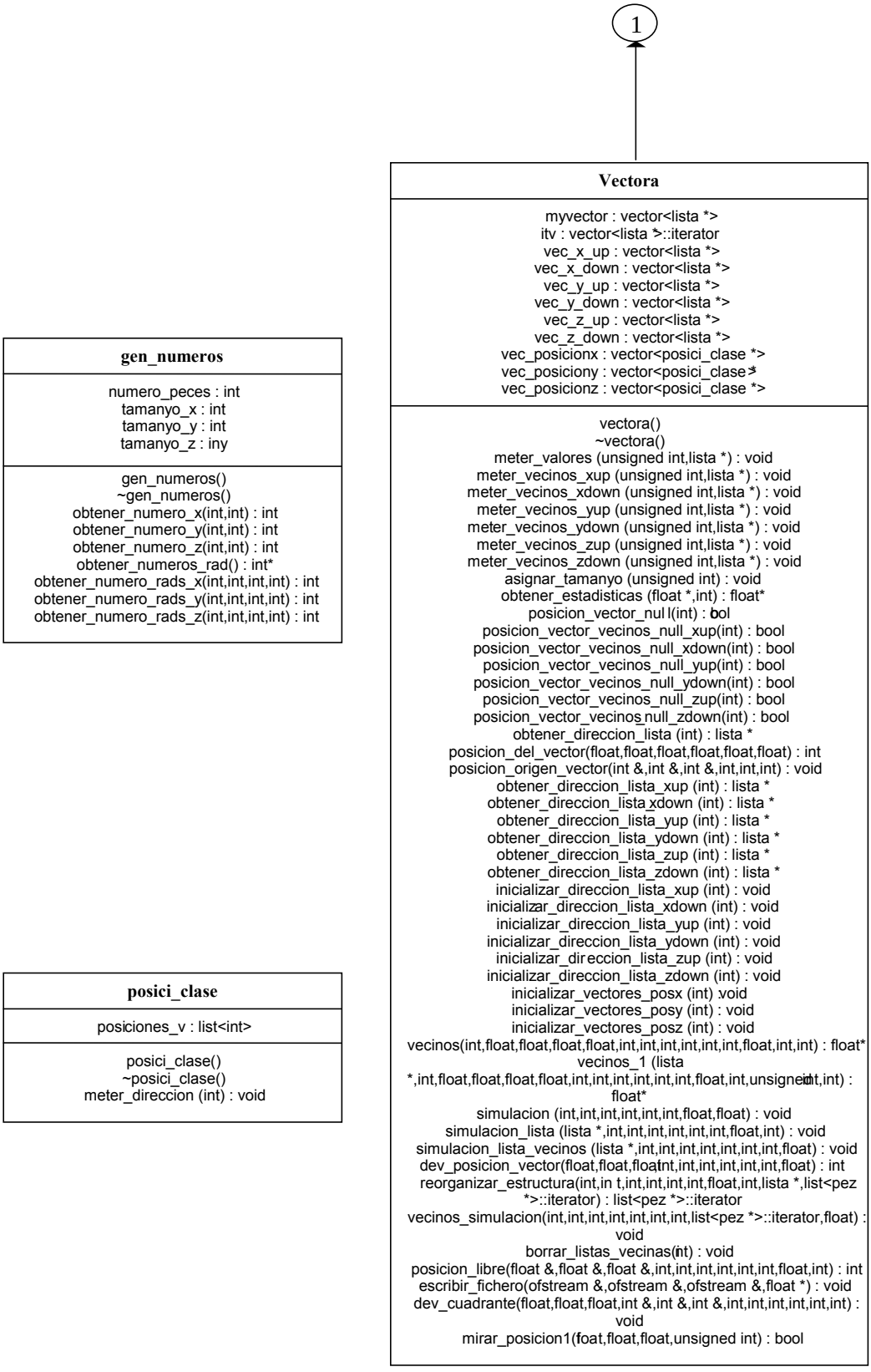
La segunda clase definirá la lista que se ha visto anteriormente, y tendrá el nombre de “lista”. Esta clase se encargará de gestionar las listas de peces que se creen. La lista que se defina en esta clase será de elementos de tipo pez.

La tercera clase definirá lo que será el vector (clase llamada vectora), y gestionará el vector de particiones y los punteros de las listas.

Finalmente se dispondrá de 2 clases secundarias que servirán por un lado para generar números aleatorios (gen_numeros) y una segunda para controlar si un pez que ocupa una nueva posición ya está siendo ocupada por otro pez. Esta segunda será principalmente una lista en donde se añadirán las posiciones de los peces para que cuando se vaya a recolocar otro se controle si ya había sido ocupada esa posición.

4.3 Diagrama de clases




1
Vectora
myvector : vector<lista *>
itv : vector<lista *>::iterator
vec_x_up : vector<lista *>
vec_x_down : vector<lista *>
vec_y_up : vector<lista *>
vec_y_down : vector<lista *>
vec_z_up : vector<lista *>
vec_z_down : vector<lista *>
vec_posicionx : vector<posici_clase *>
vec_posiciony : vector<posici_clase *>
vec_posicionz : vector<posici_clase *>
vectora()
~vectora()
meter_valores (unsigned int,lista *) : void
meter_vecinos_xup (unsigned int,lista *) : void
meter_vecinos_xdown (unsigned int,lista *) : void
meter_vecinos_yup (unsigned int,lista *) : void
meter_vecinos_ydown (unsigned int,lista *) : void
meter_vecinos_zup (unsigned int,lista *) : void
meter_vecinos_zdown (unsigned int,lista *) : void
asignar_tamanyo (unsigned int) : void
obtener_estadisticas (float *,int) : float*
posicion_vector_null (int) : bool
posicion_vector_vecinos_null_xup(int) : bool
posicion_vector_vecinos_null_xdown(int) : bool
posicion_vector_vecinos_null_yup(int) : bool
posicion_vector_vecinos_null_ydown(int) : bool
posicion_vector_vecinos_null_zup(int) : bool
posicion_vector_vecinos_null_zdown(int) : bool
obtener_direccion_lista (int) : lista *
posicion_del_vector(float,float,float,float,float,float) : int
posicion_origen_vector(int &,int &,int,int,int) : void
obtener_direccion_lista_xup (int) : lista *
obtener_direccion_lista_xdown (int) : lista *
obtener_direccion_lista_yup (int) : lista *
obtener_direccion_lista_ydown (int) : lista *
obtener_direccion_lista_zup (int) : lista *
obtener_direccion_lista_zdown (int) : lista *
inicializar_direccion_lista_xup (int) : void
inicializar_direccion_lista_xdown (int) : void
inicializar_direccion_lista_yup (int) : void
inicializar_direccion_lista_ydown (int) : void
inicializar_direccion_lista_zup (int) : void
inicializar_direccion_lista_zdown (int) : void
inicializar_vectores_posx (int) : void
inicializar_vectores_posy (int) : void
inicializar_vectores_posz (int) : void
vecinos(int,float,float,float,float,int,int,int,int,int,int,float,int,int) : float*
vecinos_1 (lista *,int,float,float,float,float,int,int,int,int,int,int,float,int,unsigned int,int) : float*
simulacion (int,int,int,int,int,int,float,float) : void
simulacion_lista (lista *,int,int,int,int,int,int,float,int) : void
simulacion_lista_vecinos (lista *,int,int,int,int,int,int,int,float) : void
dev_posicion_vector(float,float,float,int,int,int,int,int,int,float) : int
reorganizar_estructura(int,int,int,int,int,int,float,int,lista *,list<pez *>::iterator) : list<pez *>::iterator
vecinos_simulacion(int,int,int,int,int,int,int,int,list<pez *>::iterator,float) : void
borrar_listas_vecinas(int) : void
posicion_libre(float &,float &,float &,int,int,int,int,int,int,float,int) : int
escribir_fichero(ofstream &,ofstream &,ofstream &,float *) : void
dev_cuadrante(float,float,float,int &,int &,int &,int,int,int,int,int,int) : void
mirar_posicion1(float,float,float,unsigned int) : bool
gen_numeros
numero_peces : int
tamanyo_x : int
tamanyo_y : int
tamanyo_z : iny
gen_numeros()
~gen_numeros()
obtener_numero_x(int,int) : int
obtener_numero_y(int,int) : int
obtener_numero_z(int,int) : int
obtener_numeros_rad() : int*
obtener_numero_rads_x(int,int,int,int) : int
obtener_numero_rads_y(int,int,int,int) : int
obtener_numero_rads_z(int,int,int,int) : int
posici_clase
posiciones_v : list<int>
posici_clase()
~posici_clase()
meter_direccion (int) : void

5. Implementación

En este capitulo se mostrará la manera en la que se han implementado las reacciones de los peces y el algoritmo del simulador.

5.1 Algoritmo

La implementación del simulador comienza con la carga de un fichero de configuración del cual se lee línea a línea y según el orden de esta, se guarda el contenido en la variable correspondiente.

A continuación, se inicializarán el generador de números aleatorios y las estructuras dependientes del tamaño del espacio de simulación sobre las que se moverán los peces.

En función del método de colocación de los peces inicialmente (definido en el fichero de simulación), se pasarán a realizar unas acciones u otras:

- Para el caso de definir las posiciones de los peces a partir de un fichero, se realizará un bucle que ira leyendo las posiciones y velocidades de los peces definidos en el fichero. Se irán introduciendo los peces en las listas en su correspondiente posición según el lugar que ocupen en el espacio de simulación.
- Para el caso de definir las posiciones de los peces de forma aleatoria tanto para todo el espacio de simulación como en parte de el, se deberán crear tantos elementos de tipo “pez” como se haya indicado en el fichero de configuración. Para ello se crearán elementos de tipo “pez” con sus posiciones y velocidades aleatorias, que seguidamente serán incluidos en las listas en su correspondiente posición.
- Para el caso de definir las posiciones de los peces de forma aleatoria pero separadas por un determinado espacio, se configurará lo que será la definición de las posiciones que ocuparan los peces para que entre ellos

esté el espacio definido en el fichero de configuración. Una vez configurado, se crearán los peces y se les darán las posiciones que les correspondan según la definición anterior. A medida que se vayan creando los elementos de tipo “pez”, se irán introduciendo en las listas en su correspondiente posición según el lugar que ocupen del espacio de simulación.

Para los 3 métodos, se realizará finalmente la acción de guardar en unas listas los peces que puedan ser vecinos de otros peces que estén situados en cuadrantes colindantes.

Si no ha habido ningún problema en la creación de peces anterior, se pasará a realizar el cálculo de las reacciones de los peces con sus respectivos vecinos y que supondrán los cambio en sus velocidades. Para ello se realizará lo siguiente:

- Para cada posición del vector y para cada elemento de la lista se buscarán los vecinos de cada uno de los peces que conforman dicha lista. Primero se mirarán los vecinos que pudieran estar en particiones contiguas y luego los peces que ocupen su misma posición del vector (es decir su mismo cuadrante). Una vez que se tienen todos, se quitarán los vecinos que están en el ángulo muerto del pez. A continuación, se tendrán que coger aquellos peces cuya distancia esté dentro del radio de acción. De todos ellos, se quitarán los que están tapados. Con el resto y en función del método de prioridad (frontal o por distancia), se deberá calcular cuales de ellos son los más prioritarios, teniendo en cuenta el número de vecinos a escoger, que en principio será de 4.
- Una vez escogidos los vecinos, se calcularán las reacciones para cada uno de ellos en función de la posición que ocupen sus vecinos. Así pues, habrá reacción de repulsión, de orientación paralela o de atracción. El resultado de cada una de ellas se normalizará y se sumará para dar la nueva velocidad del pez. En el caso de que no haya peces cerca, la reacción del pez será la de búsqueda. Dicha reacción supondrá el cálculo aleatorio de un nuevo vector de velocidades. Finalmente se pasarán las

nuevas velocidades de los peces a sus correspondientes atributos de la clase “pez”.

- Otra funcionalidad que se aprovechará para calcular serán las estadísticas finales de grado de misma dirección y de cohesión.

A continuación se acabarán de hacer cálculos con los datos obtenidos que serán mostrados posteriormente en las estadísticas.

Seguidamente se escribirán las nuevas posiciones y direcciones (velocidades) junto con el identificador de los peces en los ficheros correspondientes. También se escribirá la primera línea del fichero de estadísticas con los primeros valores obtenidos.

En este punto comenzaría en si el simulador, formado por un bucle que se repetirá tantas veces como se haya indicado en el fichero de configuración. Para cada iteración se realizarán las siguientes acciones:

- Para cada posición del vector y para cada elemento de la lista se reemplazará la nueva dirección del pez (velocidad) por la antigua, se calculará la nueva posición del pez (a partir de su nueva velocidad) teniendo en cuenta que no este ocupada previamente, se informará a los atributos del pez de su nueva posición y finalmente se reorganizará la estructura, moviendo los elementos pez (en el caso en que corresponda) de la lista de la posición del vector en la que estaban a la nueva posición (en caso de que su posición suponga cambio de partición) y marcando como tratado a ese pez para no volver a calcular su posición. Si no hay particiones en la configuración inicial, no supondrá ningún cambio en la estructura.
- A continuación, se inicializarán los vectores de vecinos.
- También se inicializarán los indicadores de peces tratados, para que en la nueva iteración se recalcule todo de nuevo. Finalmente se guardarán las posiciones que se van ocupando por los peces, para que posteriores peces no ocupen la misma posición.

- Seguidamente se pasará de nuevo al cálculo de las reacciones de los peces con sus respectivos vecinos, de la misma manera que se ha indicado anteriormente.
- Como último paso, se enviarán al fichero los nuevos valores de las posiciones, de las direcciones y de las estadísticas.

5.2 Implementación de las reacciones de los peces

A continuación se describirá la implementación a llevar a cabo de las fórmulas correspondientes a las reacciones de los peces.

Para la implementación en 3D se utilizarán dos vectores con los que se definirán la posición y la dirección del pez.

Se tendrá que tener en cuenta que el hecho de trabajar en un espacio tridimensional hará que la representación de los radios mediante circunferencias en el sistema de 2 dimensiones pasen a ser esferas para el modelo en 3 dimensiones. Consecuentemente, el ángulo muerto pasará a estar representado como un cono.

Seguidamente se va a mencionar la nomenclatura de las variables que aparecerán en las fórmulas:

$\mathbf{P}_i\ (x_i, y_i, z_i)$: Vector de posiciones del pez.

$\boldsymbol{\nu}_i\ (\nu_x, \nu_y, \nu_z)$: Vector de velocidades / direcciones del pez (vector director).

Para el caso de los vectores de los vecinos serán $\mathbf{P}_j$ y $\boldsymbol{\nu}_j$.

$\boldsymbol{\nu}_{ij}$: Vector que representa la reacción del pez i-ésimo sobre el vecino j-ésimo.

Y a continuación el detalle para cada una de las reacciones:

Repulsión

El pez deberá tener una dirección perpendicular al vecino, con la realización del menor giro posible.

Para conseguir el vector perpendicular a ν_j, se deberán poner los vectores ν_j y ν_i en el origen del sistema de coordenadas. A continuación será necesario encontrar el plano que contiene el origen de coordenadas y es perpendicular a ν_j y una vez encontrado el plano, buscar el vector ν_{ij} que permita que el ángulo de rotación sea el mínimo.

Como consecuencia, se van a obtener 3 valores (x_{ij},y_{ij},z_{ij}) que van a ser la relación de perpendicularidad entre el pez y el vecino para cada una de las tres coordenadas.

De esta manera, los nuevos valores del vector de velocidad del pez serán los correspondientes a las variables x_{ij}, y_{ij} y z_{ij} obtenidas de la siguiente fórmula:

$$\begin{cases} x_{ij} = x_i - \dfrac{v_{x_i} \cdot v_{x_j} + v_{y_i} \cdot v_{y_j} + v_{z_i} \cdot v_{z_j}}{v_{x_j}^2 + v_{y_j}^2 + v_{z_j}^2} \cdot x_j \\ y_{ij} = y_i - \dfrac{v_{x_i} \cdot v_{x_j} + v_{y_i} \cdot v_{y_j} + v_{z_i} \cdot v_{z_j}}{v_{x_j}^2 + v_{y_j}^2 + v_{z_j}^2} \cdot y_j \\ z_{ij} = z_i - \dfrac{v_{x_i} \cdot v_{x_j} + v_{y_i} \cdot v_{y_j} + v_{z_i} \cdot v_{z_j}}{v_{x_j}^2 + v_{y_j}^2 + v_{z_j}^2} \cdot z_j \end{cases}$$

Orientación paralela

El pez origen cambiará las coordenadas de su vector de velocidad por las de su vecino.

Así pues, el vector ν_{ij} que representa la reacción del pez i-ésimo sobre el vecino j-ésimo pasará a tener la dirección que lleve el vecino j-ésimo:

$$\vec{V}_{ij} = \vec{V}_j$$

Atracción

El pez origen cambiará las coordenadas de su vector de velocidad por las de la diferencia entre la posición de su vecino y la suya, siendo $\mathbf{P}_j$ y $\mathbf{P}_i$ las posiciones del vecino j-énesimo y del pez i-ésimo. Consecuentemente el vector ν_{ij} que representa la reacción del pez i-ésimo sobre el vecino j-ésimo pasará a tener como nueva dirección, la diferencia entre los valores del vector de posiciones del vecino respecto a los valores del vector de posiciones del pez:

$$\vec{V}_{ij} = \vec{P}_j - \vec{P}_i$$

Búsqueda

El pez origen cambiará las coordenadas de su vector de velocidad por otras que se obtendrán de maneara aleatoria.

Reacción Final

La reacción final del pez vendrá dada por el sumatorio de los ν_{ij} obtenidos anteriormente según la reacción dada por los vecinos. Así pues, se realizará la suma de los valores obtenidos en los vectores que representan las reacciones de los peces i-ésimos sobre los vecinos j-ésimos:

$$\vec{V}_i = \sum_{j=1}^{4} \vec{V}_{ij}$$

6. Pruebas

En este punto se van a realizar un conjunto de pruebas con el fin de comprobar el comportamiento del simulador.

Las pruebas se dividirán en dos partes. La primera consistirá en verificar el correcto funcionamiento del sistema, mientras que la segunda serán pruebas a nivel de software.

Dentro de las pruebas de verificación del sistema se van a encontrar tres tipos:

- Verificar que el resultado del modulado de cada reacción es correcto:
    - Repulsión
    - Orientación paralela
    - Atracción
    - Búsqueda
    - Comportamiento correcto de los vecinos tapados (para todas las reacciones)
- Comprobar que el comportamiento emergente de la interacción de las reacciones es correcto:
    - Comprobar que los resultados obtenidos con el simulador sean los mismos que los aparecidos en la documentación de referencia [1] y [2].
- Experimentos adicionales que demuestren robustez del sistema y facilidades para los usuarios.
    - Pruebas de porcentaje de actuación de cada una de las reacciones respecto al total.
    - Pruebas generales de comportamiento, según disposición inicial de los peces:
        - Distribución aleatoria a través de todo el espacio de simulación.
        - Distribución aleatoria en el centro del espacio de simulación por porcentaje.
        - Distribución aleatoria en una esquina del espacio de simulación por porcentaje.

        - Distribución aleatoria en el centro del espacio de simulación por distancia entre peces.
        - Distribución aleatoria en un lado del espacio de simulación por distancia entre peces.
    - Pruebas de cambio de configuración:
        - Aumentar al doble y reducir a la mitad el tamaño de los radios (respecto a los valores por defecto).
        - Aumentar al doble y reducir a la mitad el número de vecinos a escoger (respecto a los valores por defecto).
        - Cambiar el método de elección de vecinos a prioridad por distancia (en lugar de prioridad por frontalidad).

En cuanto a las pruebas a nivel de software, se realizará una comparativa de tiempos de ejecución del simulador, en función del número de particiones en las que se divida el espacio de configuración.

Hay que tener en cuenta, que los indicadores que aparecerán en las gráficas representan lo siguiente:

- P (x,y,z) – V (x,y,z): Vector de posición y velocidad para un determinado pez. El vector de posición muestra la posición del pez expresado en tres dimensiones (x,y,z). El vector de velocidad, indica la dirección/sentido sobre el plano que tiene el pez (x,y,z).
- En cuanto a la polarización (polarization) y a la extensión (expanse) representan el grado de homogeneidad y cohesión del grupo (tal y como se define en el apartado “fish schools” del capitulo 2).

6.1 Pruebas de verificación del sistema

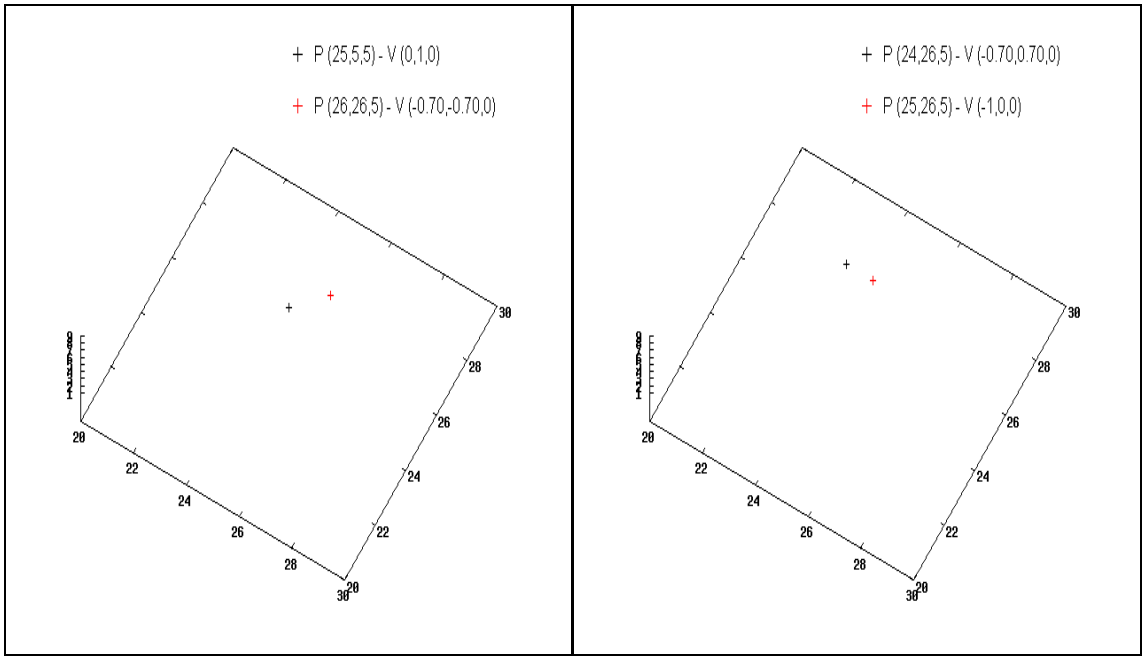
6.1.1 Verificación del modulado de las reacciones

6.1.1.1 Repulsión

En esta prueba se va a verificar el correcto funcionamiento de la reacción de repulsión.

Para ello se va a partir de una situación inicial con dos peces que se encuentran a una distancia menor a r1. Teniendo en cuenta que el tamaño del pez es de 3 y el valor de r1 es 0.5 se puede comprobar que la distancia entre ambos peces es menor de 1.5.

Las siguientes 2 gráficas mostrarán la situación inicial y la que queda una vez producida la reacción de repulsión. Se podrá comprobar como ha influido la reacción de repulsión en los valores tanto del vector de posiciones como de velocidades:



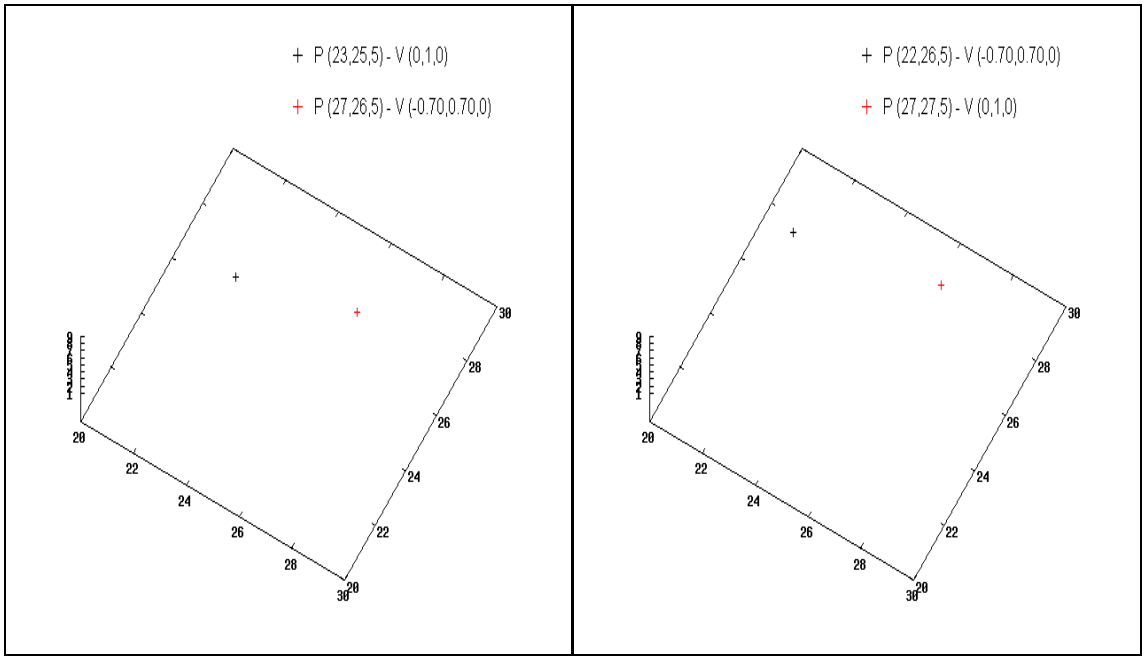
Comprobando los resultados obtenidos se puede ver que ambos peces han tomado correctamente una dirección perpendicular respecto a la que tenía su vecino. De esta manera ambos evitarán chocar con su respectivo vecino.

6.1.1.2 Orientación paralela

En esta prueba se va a verificar el correcto funcionamiento de la reacción de orientación paralela.

Para ello se va a partir de una situación inicial con dos peces que se encuentran a una distancia entre r1 y r2. Teniendo en cuenta que el tamaño del pez es de 3 y el valor de r1 es 0.5 y de r2 es de 2, se puede comprobar que la distancia entre ambos peces esta entre 1.5 y 6.

Con las siguientes 2 gráficas se va a mostrar la situación inicial y la situación que queda una vez producida la reacción de orientación paralela. También se podrá comprobar como ha influido la reacción de orientación paralela en los valores tanto del vector de posiciones como del vector de velocidades:



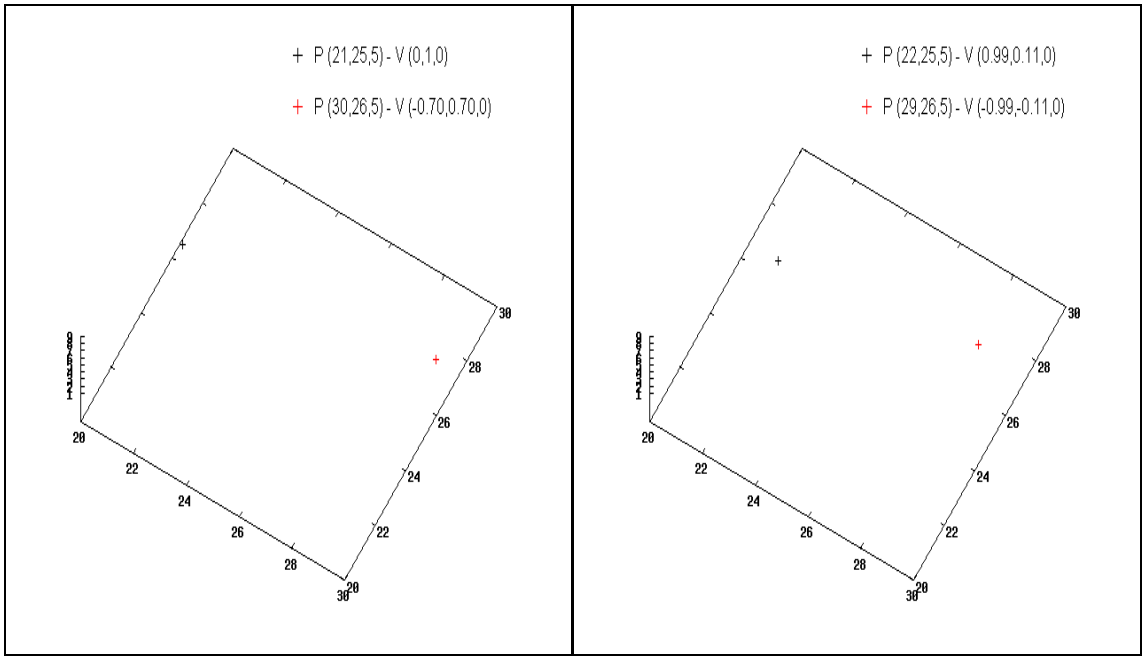
Viendo los resultados obtenidos, se puede apreciar que ambos peces han copiado correctamente la velocidad que tenía su respectivo vecino para orientarse en la misma dirección.

6.1.1.3 Atracción

En esta prueba se va a verificar el correcto funcionamiento de la reacción de atracción.

Para ello se va a partir de una situación inicial con dos peces que se encuentran a una distancia entre r2 y r3. Teniendo en cuenta que el tamaño del pez es de 3 y el valor de r2 es 2 y de r3 es de 5, se puede comprobar que la distancia entre ambos peces esta entre 6 y 15.

En las próximas 2 gráficas se mostrará la situación inicial y la situación que queda una vez producida la reacción de atracción. Además se podrá apreciar como ha influido la reacción de atracción en los valores tanto del vector de posiciones como del vector de velocidades:



A partir de los resultados mostrados en las gráficas, se puede ver que las nuevas direcciones de ambos peces son correctamente el resultado de las diferencias de posición entre ellos. De esta manera, ambos peces tienden a acercarse.

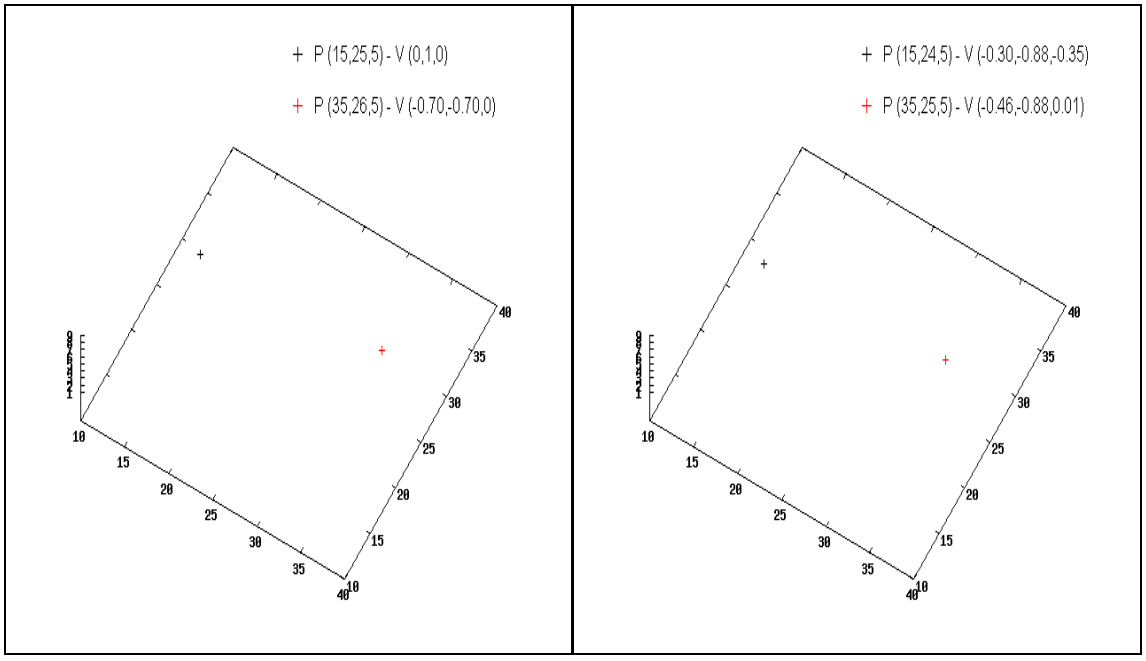
6.1.1.4 Búsqueda

En esta prueba se va a verificar el correcto funcionamiento de la reacción de búsqueda.

Para ello se va a partir de una situación inicial con dos peces que se encuentran a una distancia superior a r3. Teniendo en cuenta que el tamaño del pez es de 3 y el valor de r3 es de 5, se puede comprobar que la distancia entre ambos peces es superior a 15.

En las siguientes 2 gráficas se va a mostrar la situación inicial y la situación que queda una vez producida la reacción de búsqueda.

Se podrá comprobar también como ha influido la reacción de búsqueda en los valores tanto del vector de posiciones como del vector de velocidades:



A continuación se van a analizar los resultados obtenidos. Como se puede ver, las nuevas direcciones de ambos peces son correctamente nuevos valores aleatorios que se corresponden con un cambio en la dirección que llevaba el pez en busca de vecinos en otras direcciones.

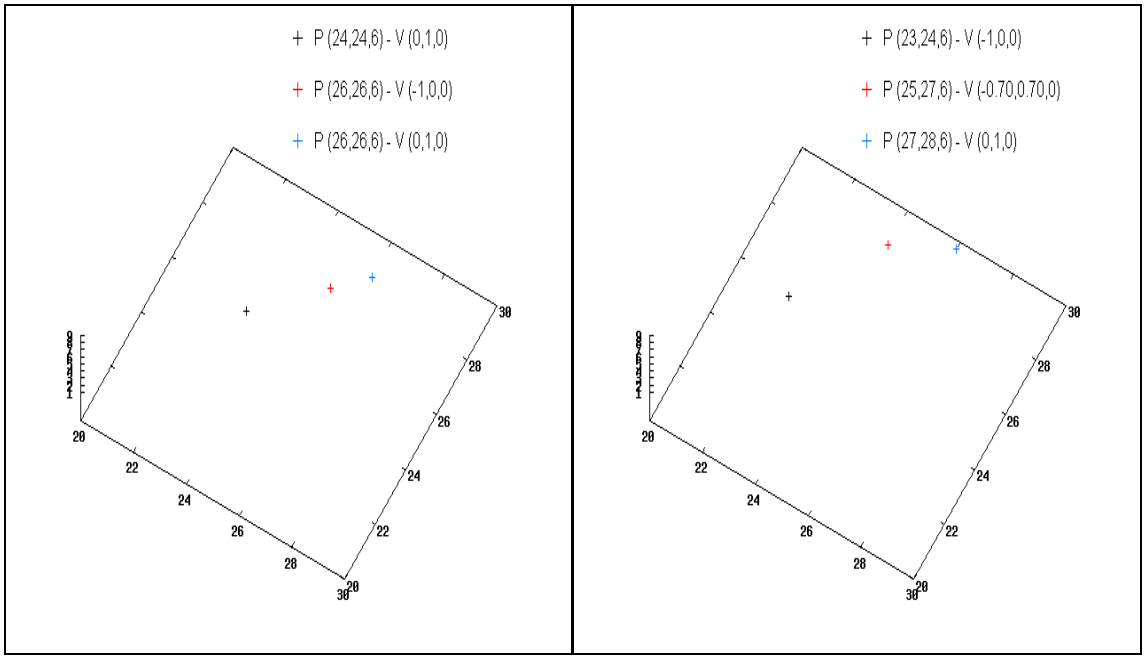
6.1.1.5 Caso en el que un pez esté siendo tapado por otro

En esta prueba se va a verificar el correcto funcionamiento de los casos en los que existen vecinos tapados y por tanto no han de tenerse en cuenta.

Para ello se va a partir de una situación inicial con tres peces que se encuentran a una distancia tal, que su reacción será la de orientación paralela. La prueba consiste en ver que dirección cogerá el pez origen, si solo tiene en cuenta la del pez que ve, o también coge la del pez que esta tapado.

En las siguientes 2 gráficas se van a mostrar la situación inicial y la situación que queda una vez producida la reacción de orientación paralela.

Se podrá comprobar si ha influenciado o no el pez tapado:



A continuación se van a analizar los resultados obtenidos. Se tomará como pez de referencia el que tiene como posición inicial la (24,24,6). Se puede ver que tiene dos peces vecinos, pero que uno de ellos está en el mismo ángulo de visión y por detrás, con lo que está tapado. En la primera iteración se puede observar como sólo ha tenido en cuenta el pez que ve, ya que ha copiado sólo su dirección (-1,0,0), con lo que se puede considerar la prueba es correcta.

6.1.2 Comprobación del comportamiento emergente de la interacción de las reacciones

Comprobación de los resultados obtenidos con el simulador respeto a los aparecidos en la documentación de referencia ([1] y [2])

Esta prueba va a servir para validar si los resultados del simulador se corresponden con los aparecidos en la documentación y por tanto dar como correcto el funcionamiento del mismo.

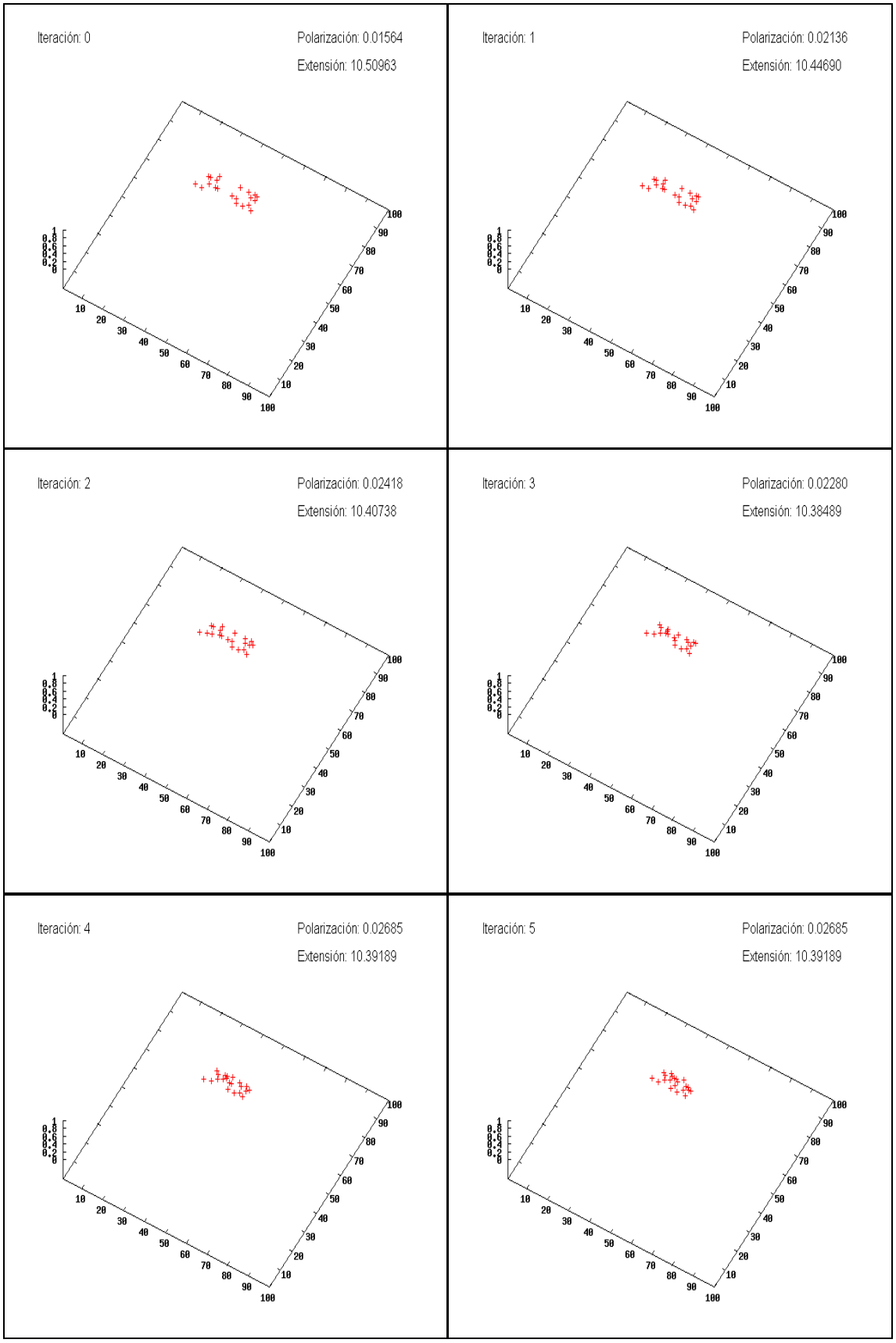
Lo que se pretende es verificar como los peces tienden a agruparse. Para ello, se partirá de dos grupos independientes de peces que se encuentran a una distancia próxima entre ellos y que mediante el conjunto de reacciones propias de los peces acabarán haciendo que los dos grupos se vayan acercando entre ellos hasta finalmente acabar unidos.

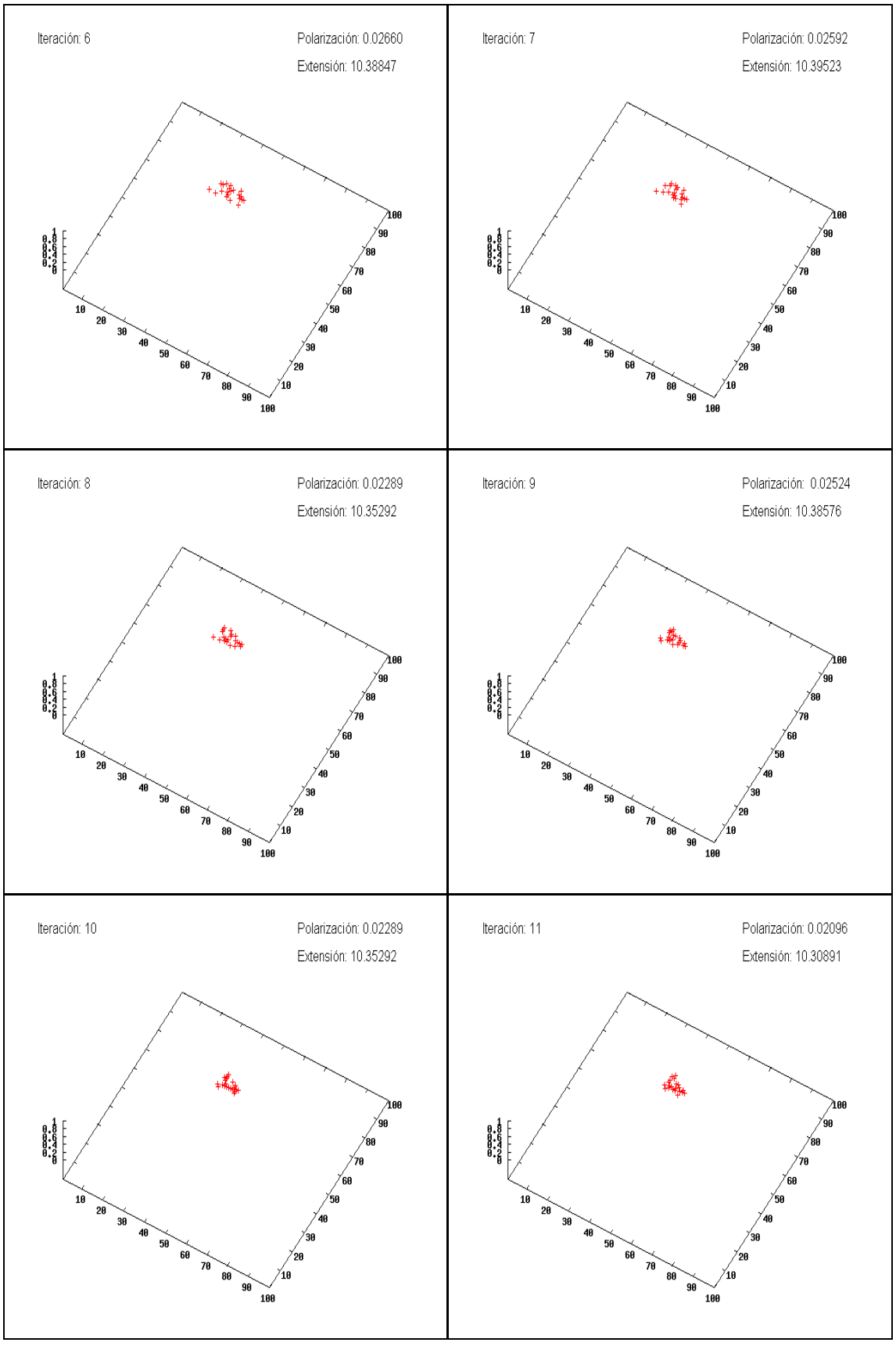
En un principio se realizará la simulación (mostrando las graficas de los resultados obtenidos) y posteriormente se incluirá el gráfico aparecido en la documentación [1] y [2]. De esta manera se podrán comparar ambos resultados.

Los datos de la simulación son:

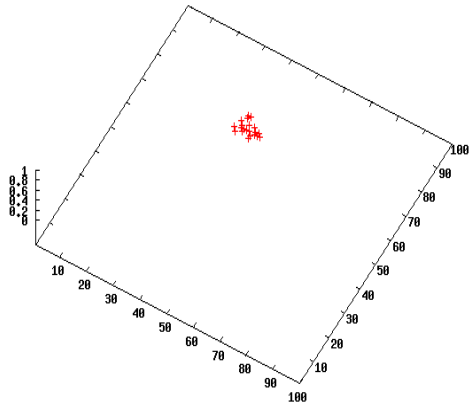
- 21 peces, 100x100x1 de resolución y 60 iteraciones

En el siguiente conjunto de gráficas se muestra la evolución de la simulación desde la posición inicial hasta la iteración 60:

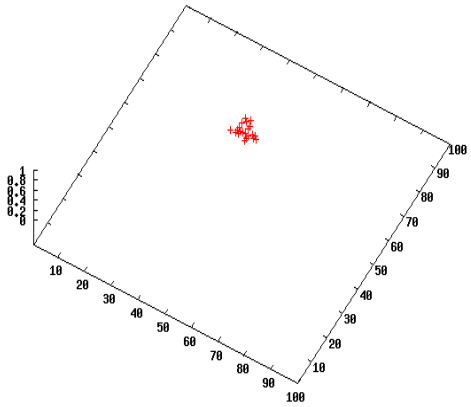



Iteración: 6
Polarización: 0.02660
Extensión: 10.38847
Iteración: 7
Polarización: 0.02592
Extensión: 10.39523
Iteración: 8
Polarización: 0.02289
Extensión: 10.35292
Iteración: 9
Polarización: 0.02524
Extensión: 10.38576
Iteración: 10
Polarización: 0.02289
Extensión: 10.35292
Iteración: 11
Polarización: 0.02096
Extensión: 10.30891

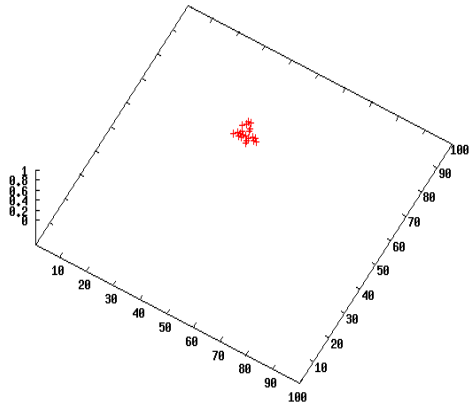
Iteración: 12 Polarización: 0.00894
Extensión: 10.26485



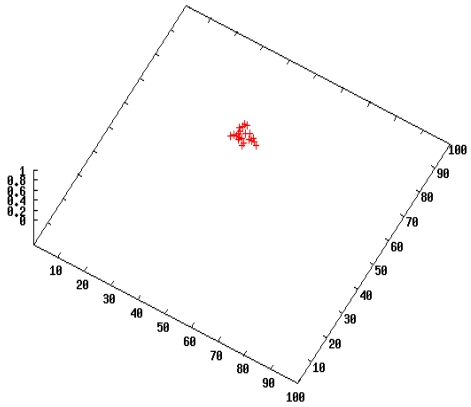
Iteración: 13 Polarización: 0.01976
Extensión: 10.30467



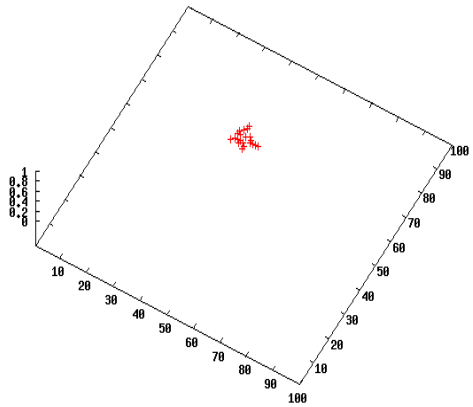
Iteración: 14 Polarización: 0.01328
Extensión: 10.29995



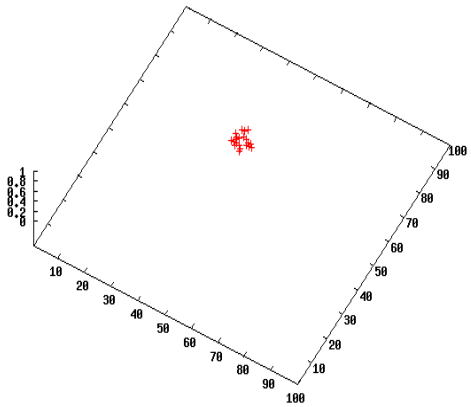
Iteración: 15 Polarización: 0.00894
Extensión: 10.26485



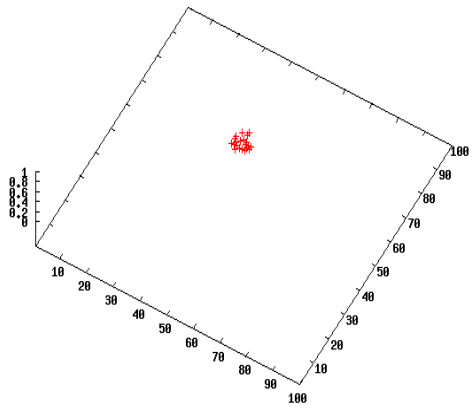
Iteración: 16 Polarización: 0.00848
Extensión: 9.99379



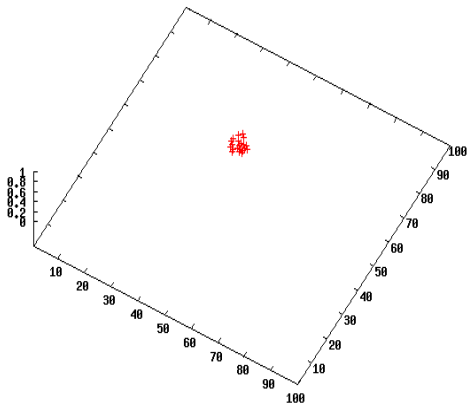
Iteración: 17 Polarización: 0.00783
Extensión: 10.19890



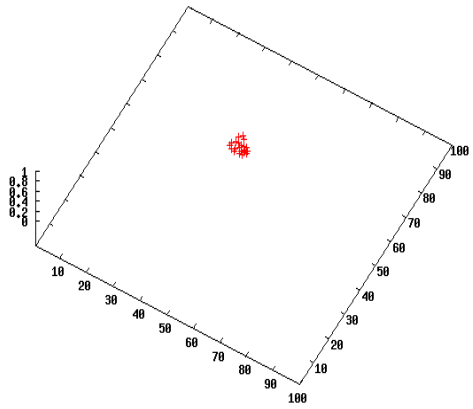
Iteración: 18 Polarización: 0.00703
Extensión: 10.12937



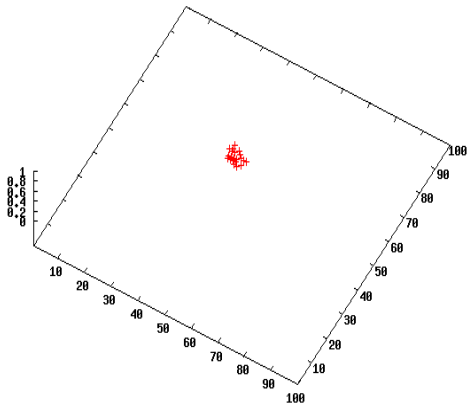
Iteración: 19 Polarización: 0.00767
Extensión: 10.06998



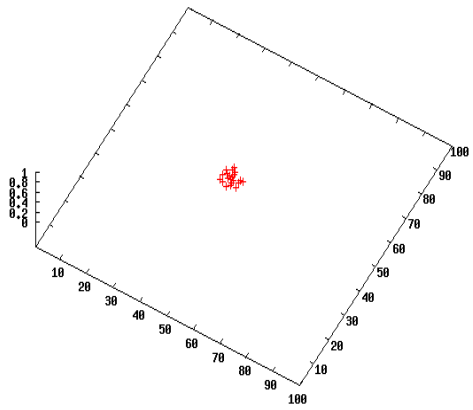
Iteración: 20 Polarización: 0.00781
Extensión: 9.72543



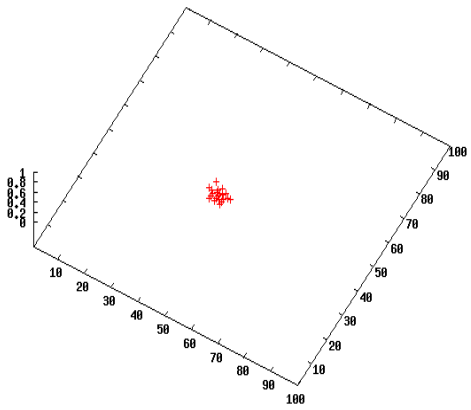
Iteración: 24 Polarización: 0.00748
Extensión: 9.53982

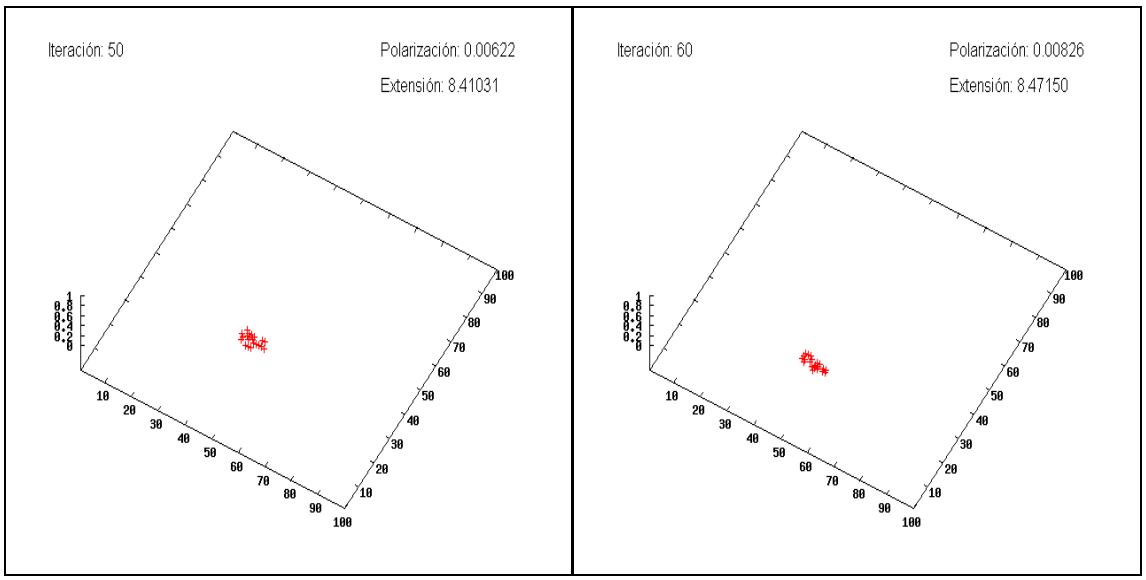


Iteración: 32 Polarización: 0.00637
Extensión: 9.23216



Iteración: 40 Polarización: 0.01259
Extensión: 8.79983





Analizando los resultados se puede comprobar como se partió de una posición inicial en la que dos grupos separados tienen sus direcciones propicias para que finalmente acaben juntándose. Pocas iteraciones posteriores se pude comprobar como dichos grupos se juntan en uno único tendiendo todo el grupo a continuar en la misma dirección que traían ambos grupos. Por tanto se puede considerar como correcto el resultado de la prueba.

A continuación se mostrará la gráfica (figura 6.1) de los resultados aparecidos en la documentación de referencia para esta prueba. Se puede apreciar la evolución de la simulación mediante los 3 pasos existentes (a, b y c):

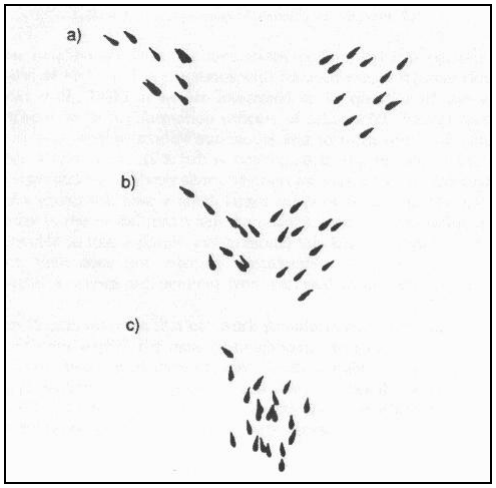


Figura 6.1: Simulación de la unión de dos grupos de peces

Comparando los resultados del simulador con los de la documentación, se puede llegar a la conclusión de que el funcionamiento del simulador es correcto ya que se obtienen los mismos resultados que los existentes en la documentación de referencia.

6.1.3 Experimentos adicionales

6.1.3.1 Pruebas de porcentaje de actuación de cada una de las reacciones respecto al total

Partiendo de una simulación con una distribución como la que aparece en la figura 6.2a (1000 peces, 500x500x20 de resolución), se podrá ver la evolución del grado de actuación de las reacciones a lo largo de la ejecución de la simulación hasta llegar al final de la misma (figura 6.2b).

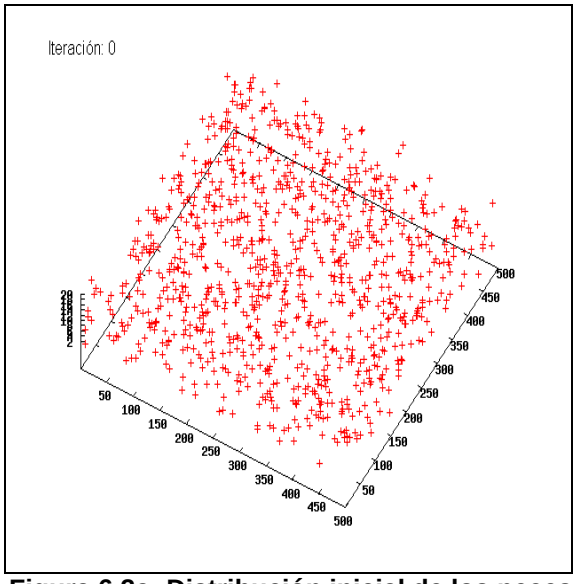


Figura 6.2a: Distribución inicial de los peces

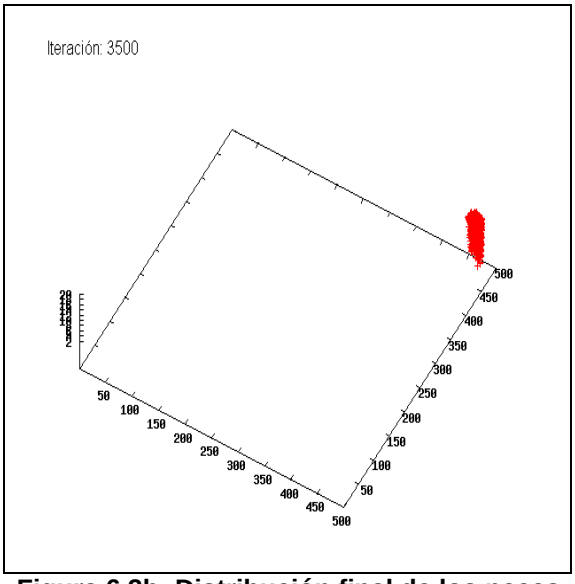


Figura 6.2b: Distribución final de los peces

El resultado para cada una de las iteraciones es el siguiente:

Número de Iteración	Repulsión [%]	Orientación Paralela [%]	Atracción [%]	Búsqueda [%]
0	0,106	8,855	82,185	8,855
1	0	12,777	80,376	6,847
2	0,091	17,425	75,751	6,733
3	0,174	22,125	70,862	6,838
4	0,168	28,943	64,975	5,914
5	0,367	38,452	54,949	6,232
6	0,435	41,512	51,326	6,727
7	0,463	47,974	46,314	5,249
8	0,604	50,396	43,186	5,814
9	0,517	56,056	37,851	5,576
10	0,910	58,806	35,262	5,022
25	4,855	80,450	11,061	3,633
50	5,559	82,712	10,099	1,630
75	6,090	86,346	6,390	1,174
100	7,886	86,180	5,159	0,775
250	16,794	78,874	4,154	0,178
500	23,597	73,233	3,094	0,075
750	23,886	69,554	6,535	0,025
1000	29,325	66,375	4,300	0
2000	26,175	71,325	2,500	0
3000	28,400	60,075	11,525	0
3500	25,150	68,100	6,750	0

Una vez obtenidos los resultados, se puede comprobar como inicialmente y debido a la gran separación entre peces, el mayor porcentaje de actuación es el de atracción.A medida que los peces se van uniendo, el mayor porcentaje pasa a ser el de orientación paralela ya que este está presente en distancias más pequeñas entre peces. Finalmente, en el momento en el que el grupo pasa a estar muy cohesionado, las reacciones que más se dan son la de orientación paralela y la de repulsión, debido a la pequeña distancia existente entre peces.

6.1.3.2 Pruebas de comportamiento, según disposición inicial de los peces

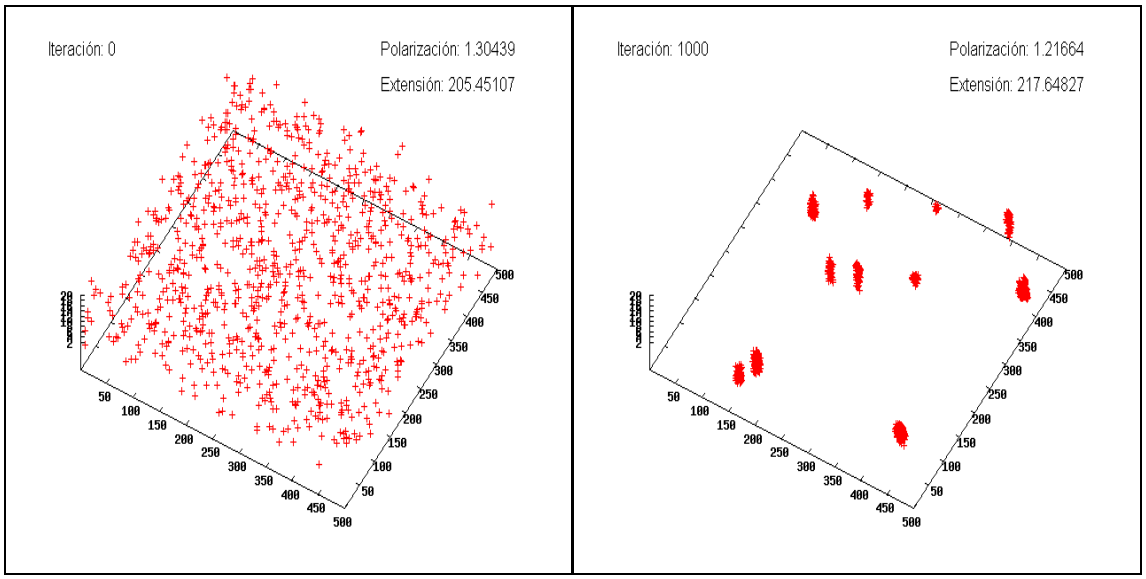
Distribución aleatoria a través de todo el espacio de simulación

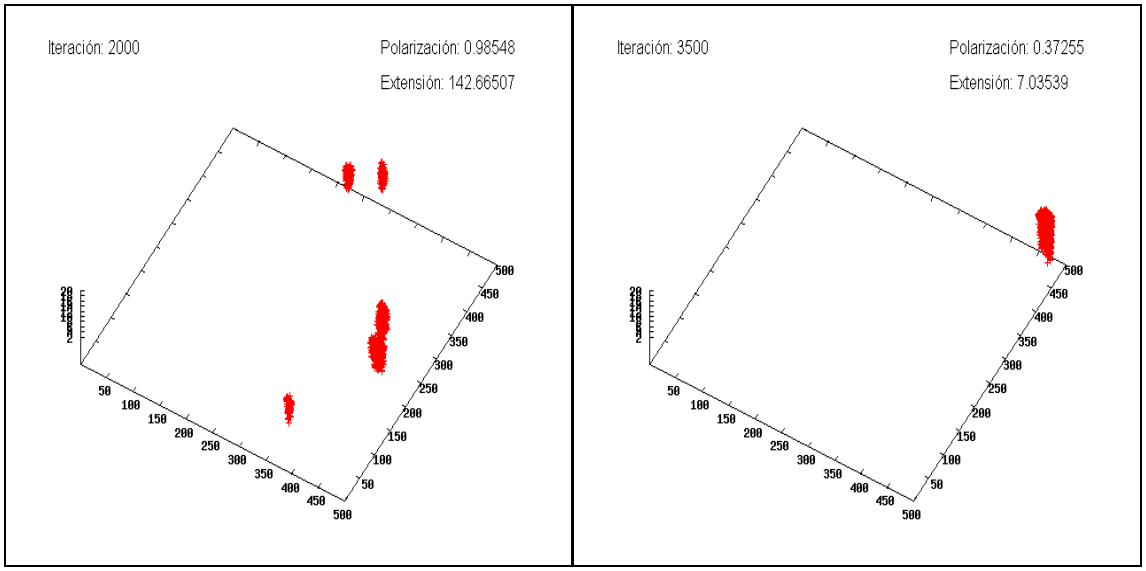
En esta prueba se va a verificar el correcto comportamiento para un caso genérico de simulación en la que la distribución de los peces inicial sea aleatoria y distribuida por todo el espacio de simulación.

Los datos de la simulación son:

- 1000 peces, 500x500x20 de resolución y 3500 iteraciones

En el siguiente conjunto de gráficas se va a mostrar la evolución de la simulación desde la posición inicial hasta la iteración 3500:





Analizando los resultados obtenidos, se puede comprobar que se ha pasado de una situación inicial en la que los peces estaban completamente repartidos por todo el espacio de simulación (extensión/*expanse*: 205) a una situación después de 3500 iteraciones en la que todos los peces están juntos (extensión/*expanse*: 7). En las iteraciones intermedias se puede ver como los peces se van juntando en grupos hasta que finalmente queda un único grupo con todos ellos. Por tanto, se puede considerar como correcto el resultado de la prueba.

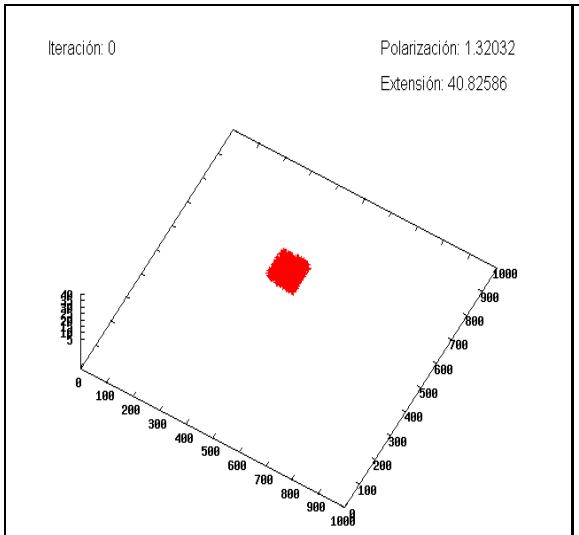
Distribución aleatoria en el centro del espacio de simulación por porcentaje

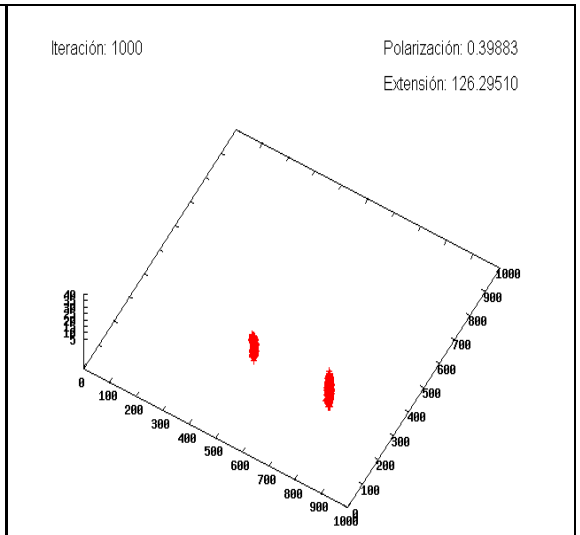
En esta prueba se va a verificar el correcto comportamiento para un caso genérico de simulación en la que la distribución de los peces este agrupada en un espacio del 10% sobre el total y que esté situada en el centro.

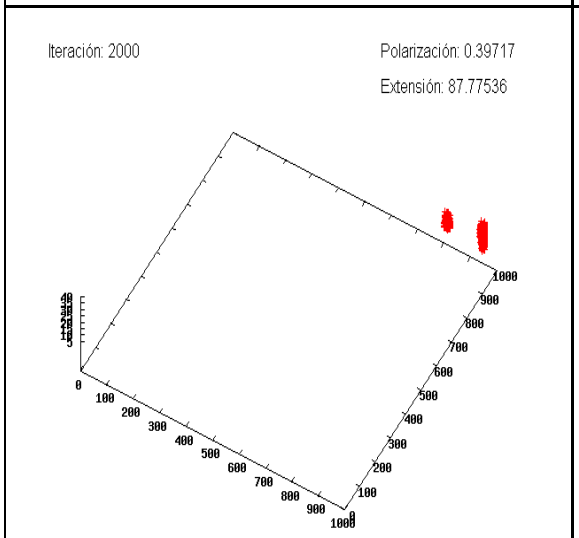
Los datos de la simulación son:

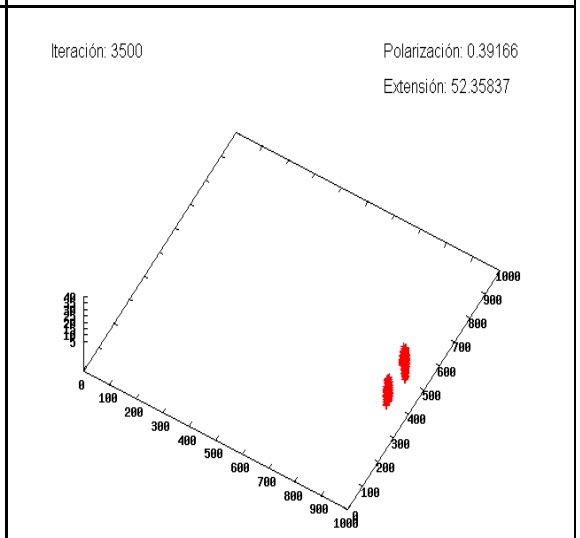
- 1000 peces, 1000x1000x40 de resolución y 4500 iteraciones
- Porcentaje 10 (Distribución de los puntos del 10% del espacio), Posición 50x50x50 (La posición será en el centro).

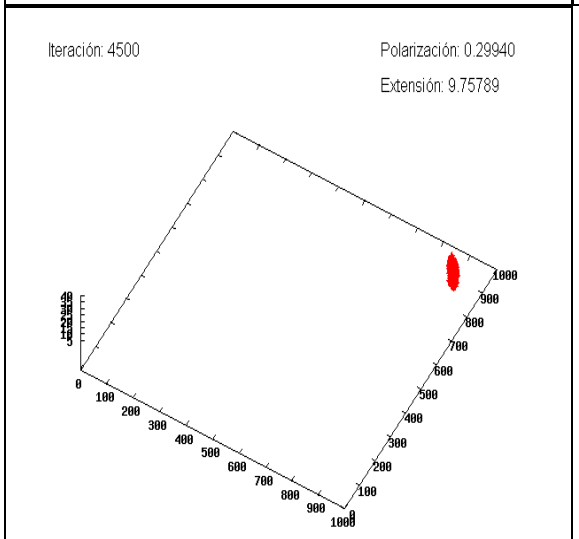
Con el siguiente conjunto de gráficas se mostrará la evolución de la simulación desde la posición inicial hasta la iteración 4500:











Viendo los resultados de la prueba, se puede comprobar que en la posición inicial los peces estaban agrupados en el centro. Después de algunas iteraciones, ese grupo se ha ido separando hasta quedar 2 grupos. En dichos grupos ha existido una fuerte cohesión y se han ido moviendo hasta casi juntarse entre ellos. Como se puede comprobar, finalmente en la iteración 4500 se acaba formando un único grupo, dando un grado de polarización bastante bajo (0.29) con lo que se puede considerar como correcto el resultado de la prueba.

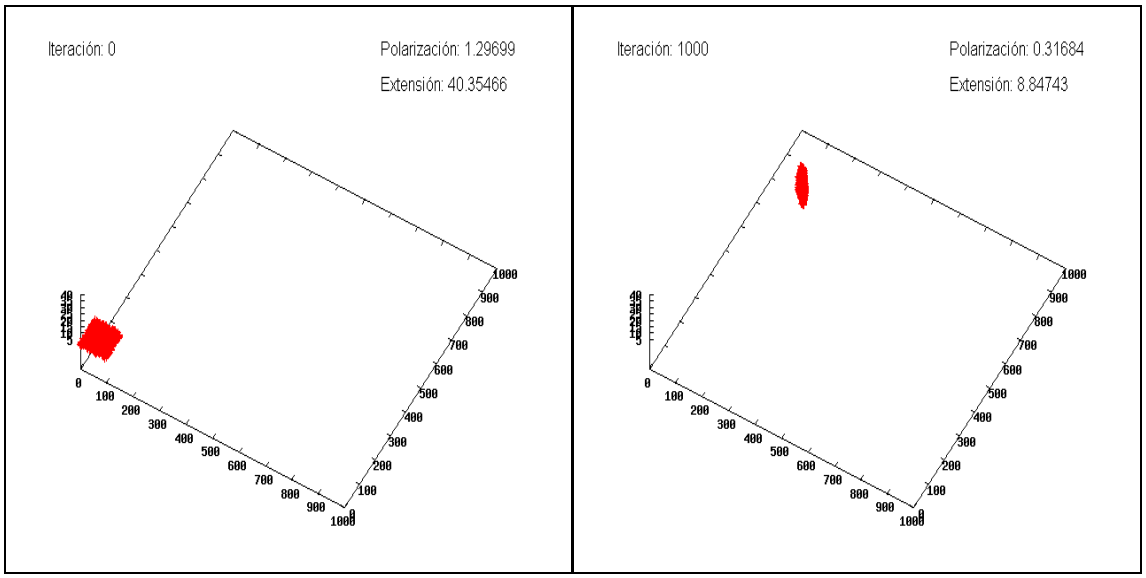
Distribución aleatoria en una esquina del espacio de simulación por porcentaje

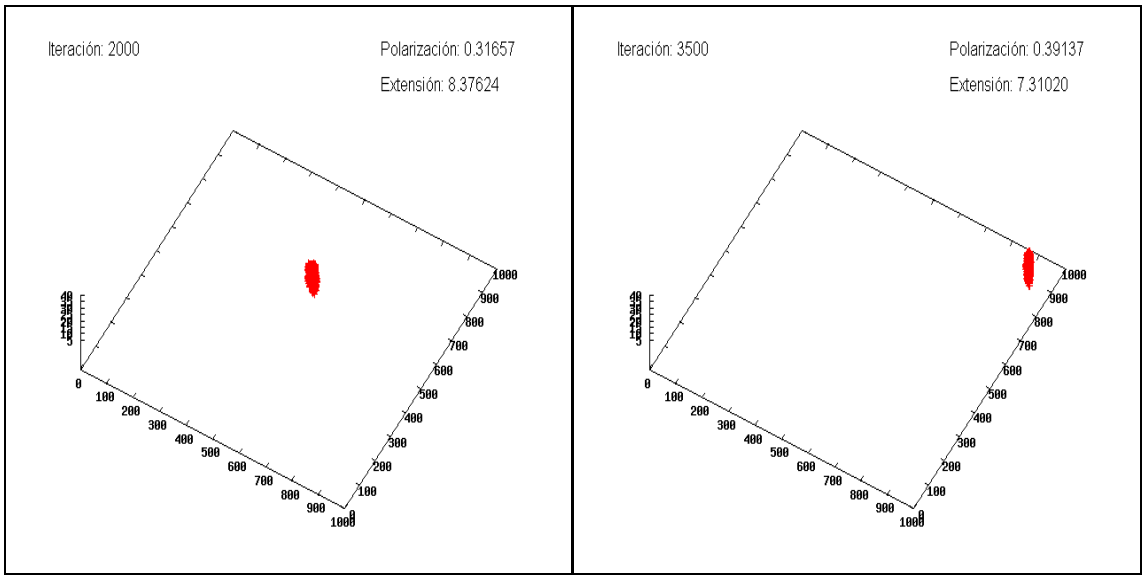
En esta prueba se va a verificar el correcto comportamiento para un caso genérico de simulación en la que la distribución de los peces este agrupada en un espacio del 10% sobre el total y que esté situada en una esquina.

Los datos de la simulación son:

- 1000 peces, 1000x1000x40 de resolución y 3500 iteraciones
- Porcentaje 10 (Distribución de los puntos del 10% del total del espacio), Posición 5x5x5 (La posición será en una esquina).

Mediante el siguiente conjunto de gráficas se va a mostrar la evolución de la simulación desde la posición inicial hasta la iteración 3500:





A continuación se van a analizar los resultados obtenidos. Como se puede comprobar, en la posición inicial los peces estaban agrupados en una esquina, aunque sus direcciones eran bastante desiguales entre ellos (Polarización: 1,29). El hecho de producirse una reflexión de las direcciones de los peces en la esquina y por tanto igualar las direcciones, permite que pocas iteraciones después exista una cohesión bastante fuerte (en la iteración 1000 la polarización ha bajado a 0.31 y la extensión a 8.84). En el resto de iteraciones hasta la final, se consigue que el grupo siga totalmente junto. Por tanto se puede considerar como correcto el resultado de la prueba.

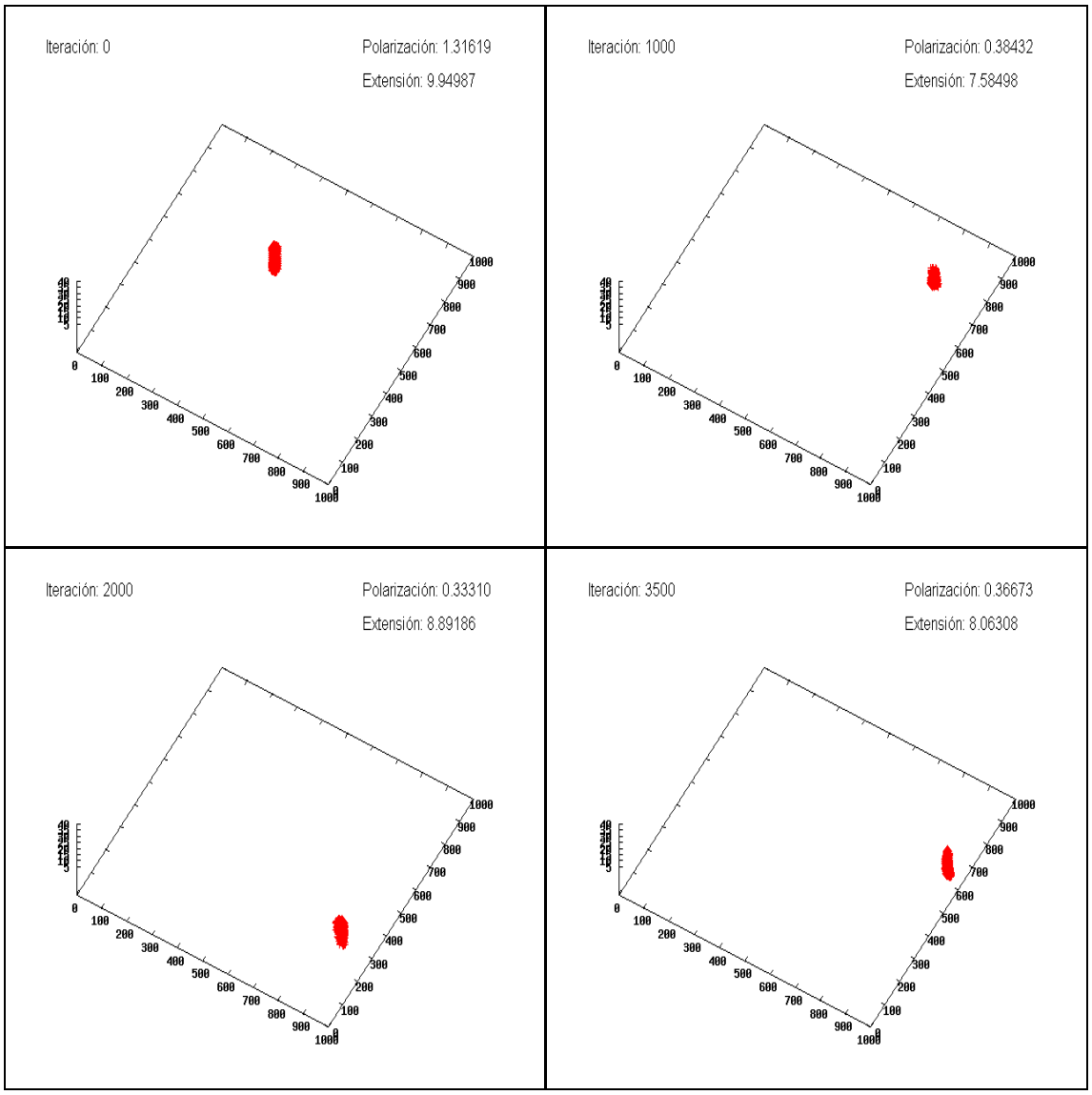
Distribución aleatoria en el centro del espacio de simulación por distancia entre peces

En esta prueba se va a verificar el correcto comportamiento para un caso genérico de simulación en la que la distribución de los peces este agrupada con una distancia entre peces de valor 2 y que esté situada en el centro.

Los datos de la simulación son:

- 1000 peces, 1000x1000x40 de resolución y 3500 iteraciones
- Distancia entre peces de valor 2 (Distancia entre un pez y otro consecutivo), Posición 50x50x50 (La posición será en el centro).

En el siguiente conjunto de gráficas mostrará la evolución de la simulación desde la posición inicial hasta la iteración 3500:



Analizando los resultados obtenidos se puede comprobar que en la posición inicial los peces estaban agrupados en el centro, aunque sus direcciones eran bastante desiguales entre ellos (Polarización: 1,31). El hecho de que la distancia entre peces sea de 2, provoca que haya una reacción de orientación paralela y por tanto igualar sus direcciones. Esto permite que pocas iteraciones después exista una cohesión bastante fuerte (en la iteración 1000 la polarización ha bajado a 0.38 y la extensión a 7.58). En el resto de iteraciones hasta la final, se consigue que el grupo siga

totalmente junto, por lo que el resultado de la prueba se puede considerar como satisfactorio.

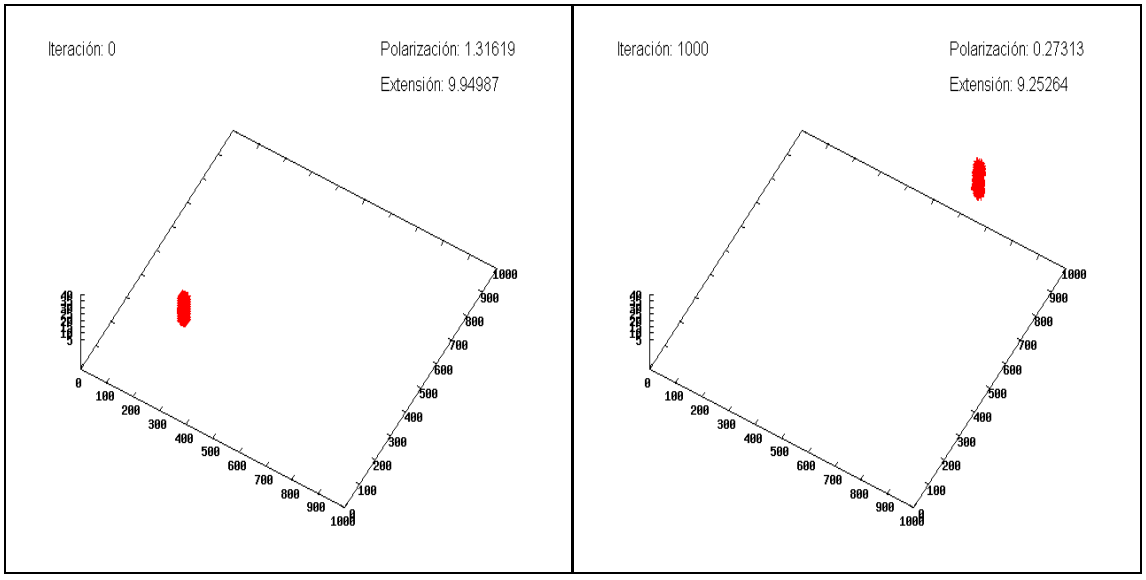
Distribución aleatoria en un lado del espacio de simulación por distancia entre peces

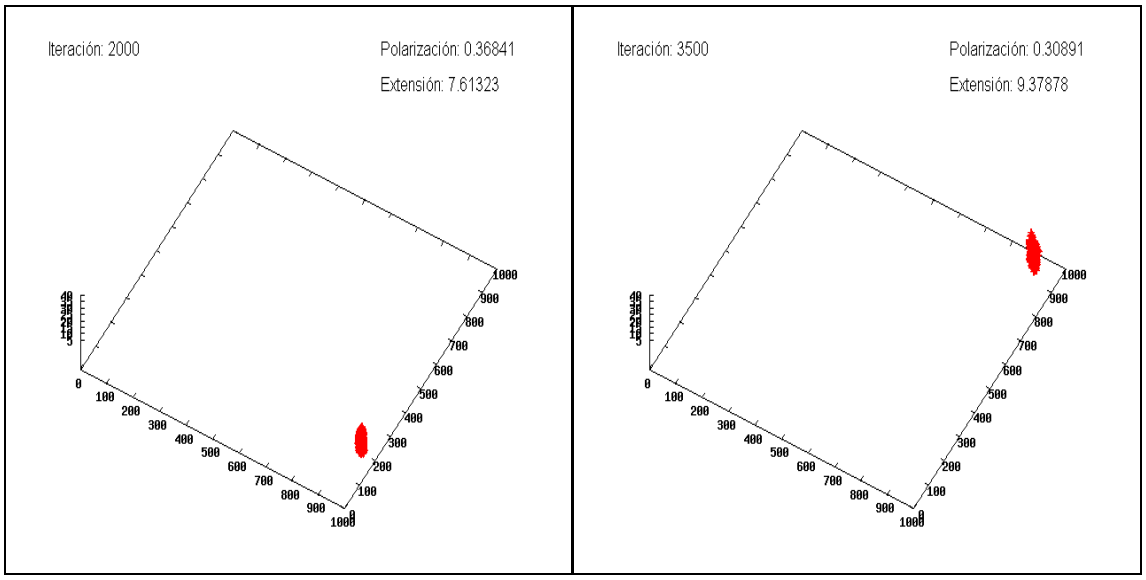
En esta prueba se va a verificar el correcto comportamiento para un caso genérico de simulación en la que la distribución de los peces este agrupada con una distancia entre peces de valor 2 y que esté a un lado del espacio de simulación.

Los datos de la simulación son:

- 1000 peces, 1000x1000x40 de resolución y 3500 iteraciones
- Distancia entre peces de valor 2 (Distancia entre un pez y otro consecutivo), Posición 25x25x25 (La posición será en un lado).

Mediante el siguiente conjunto de gráficas se mostrará la evolución de la simulación desde la posición inicial hasta la iteración 3500:





Viendo los resultados de la prueba se puede comprobar que en la posición inicial estaban los peces agrupados en un lado, aunque sus direcciones eran bastante desiguales entre ellos (Polarización: 1,31). El hecho de que la distancia entre peces sea de 2, provoca que haya una reacción de orientación paralela y por tanto igualar sus direcciones. Esto permite que pocas iteraciones después aparezca una cohesión bastante fuerte (en la iteración 1000 la polarización ha bajado a 0.27 y la extensión a 9.25). En el resto de iteraciones hasta la final, se consigue que el grupo siga totalmente junto. Por tanto se puede considerar el resultado de la prueba como satisfactorio.

6.1.3.3 Pruebas de cambio de configuración

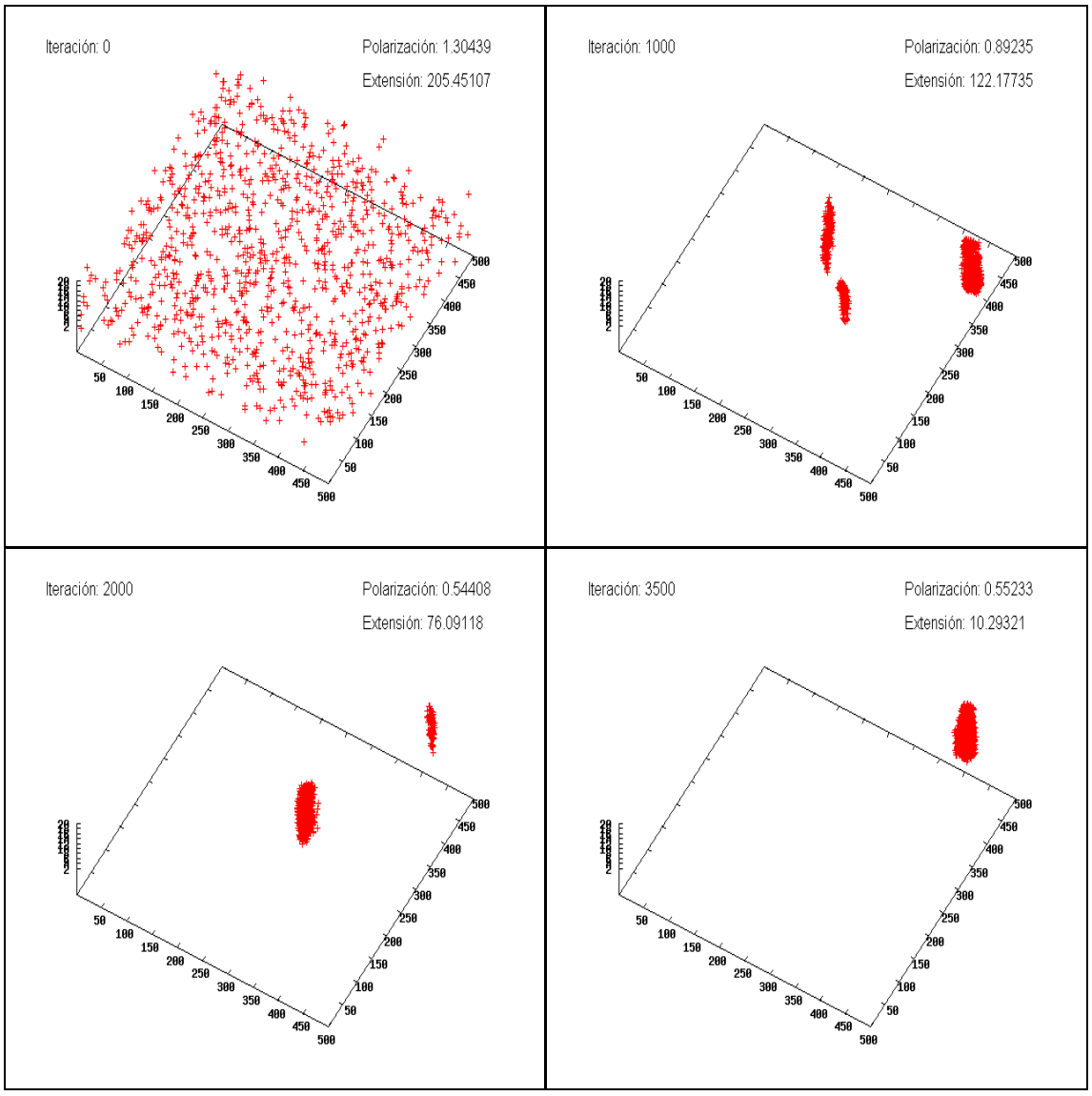
Aumentar al doble y reducir a la mitad el tamaño de los radios

En esta prueba se va a comprobar que implicaciones tiene cambiar el tamaño de los radios de acción de los peces. Para ello, se van a realizar dos simulaciones, una aumentado al doble el tamaño de los 3 radios y una segunda reduciéndolos a la mitad (respecto a los tamaños originales).

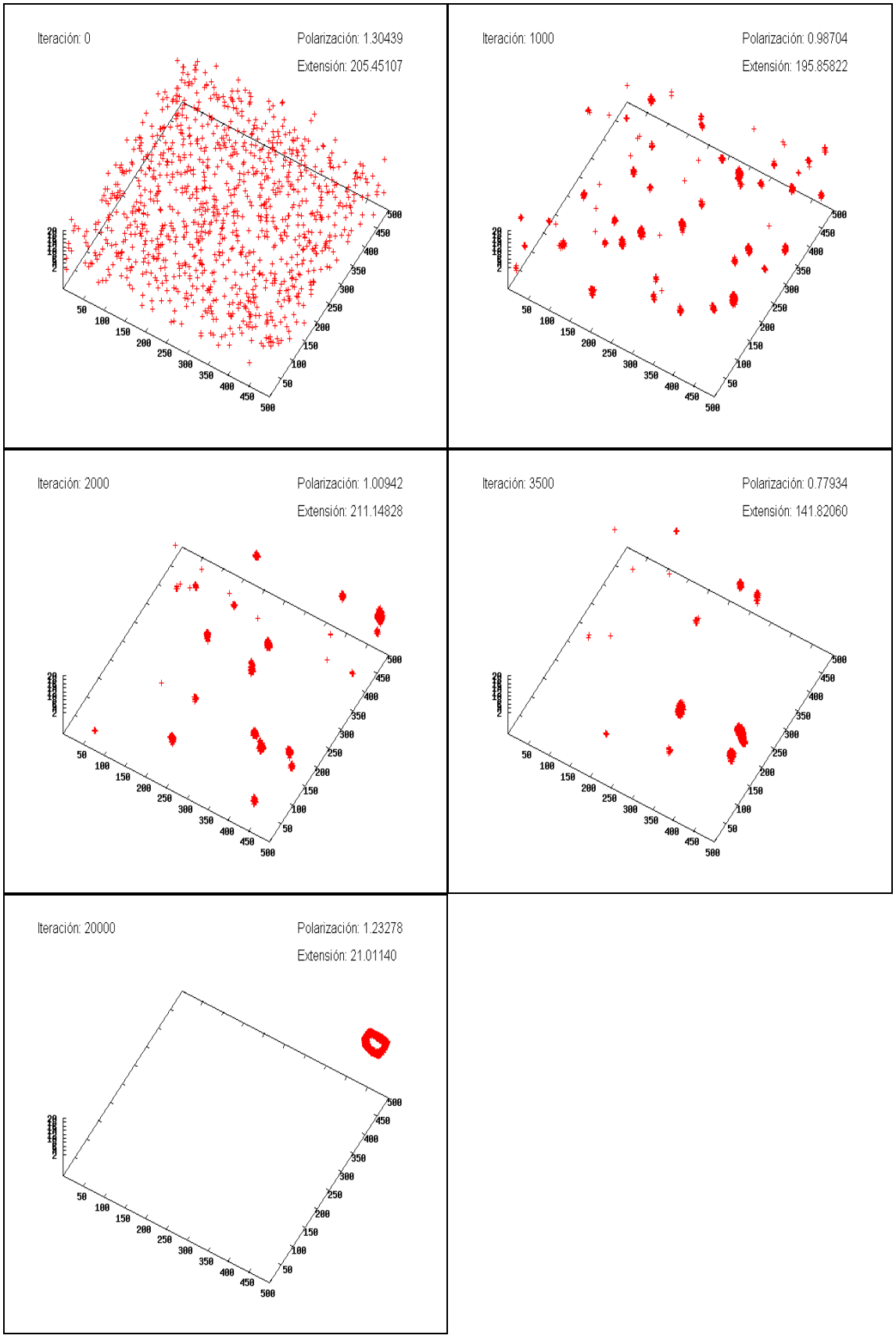
Los datos de la 2 simulaciones son:

- 1000 peces, 500x500x20 de resolución y 3500 iteraciones (20000 para el caso de la mitad)
- La distribución de los peces será aleatoria en todo el espacio de simulación.

En el siguiente conjunto de gráficas se va a mostrar la evolución de la simulación desde la posición inicial hasta la iteración 3500 para el caso de que los radios sean el doble de lo que lo son habitualmente:



Mediante el siguiente conjunto de gráficas se mostrará la evolución de la simulación desde la posición inicial hasta la iteración 20000 para el caso de que los radios sean la mitad de lo que lo son habitualmente:



Como se puede observar, para el caso de la simulación con el doble de radio se obtienen bastante buenos resultados, en cambio para la simulación en la que el tamaño del radio es la mitad de lo estándar, los resultados son bastante peores. Después de las 3500 iteraciones se puede comprobar que existen bastantes peces sueltos y en cambio para el caso del doble de radio, se puede observar que se consigue llegar a tener un único grupo.

Habrá que esperar hasta llegar a la iteración 20000 (en el caso de la mitad del radio) para que los resultados lleguen a ser algo mejores. Como conclusión se podría decir que el hecho de que los radios sean tan pequeños, hace que se tarde más en producir las reacciones de orientación paralela y sobre todo de atracción y en cambio la que predomina es la de búsqueda. Además, el hecho de que el tamaño de los radios no sea el que se establece como correcto, hace que los resultados que se obtengan no sean los esperados.

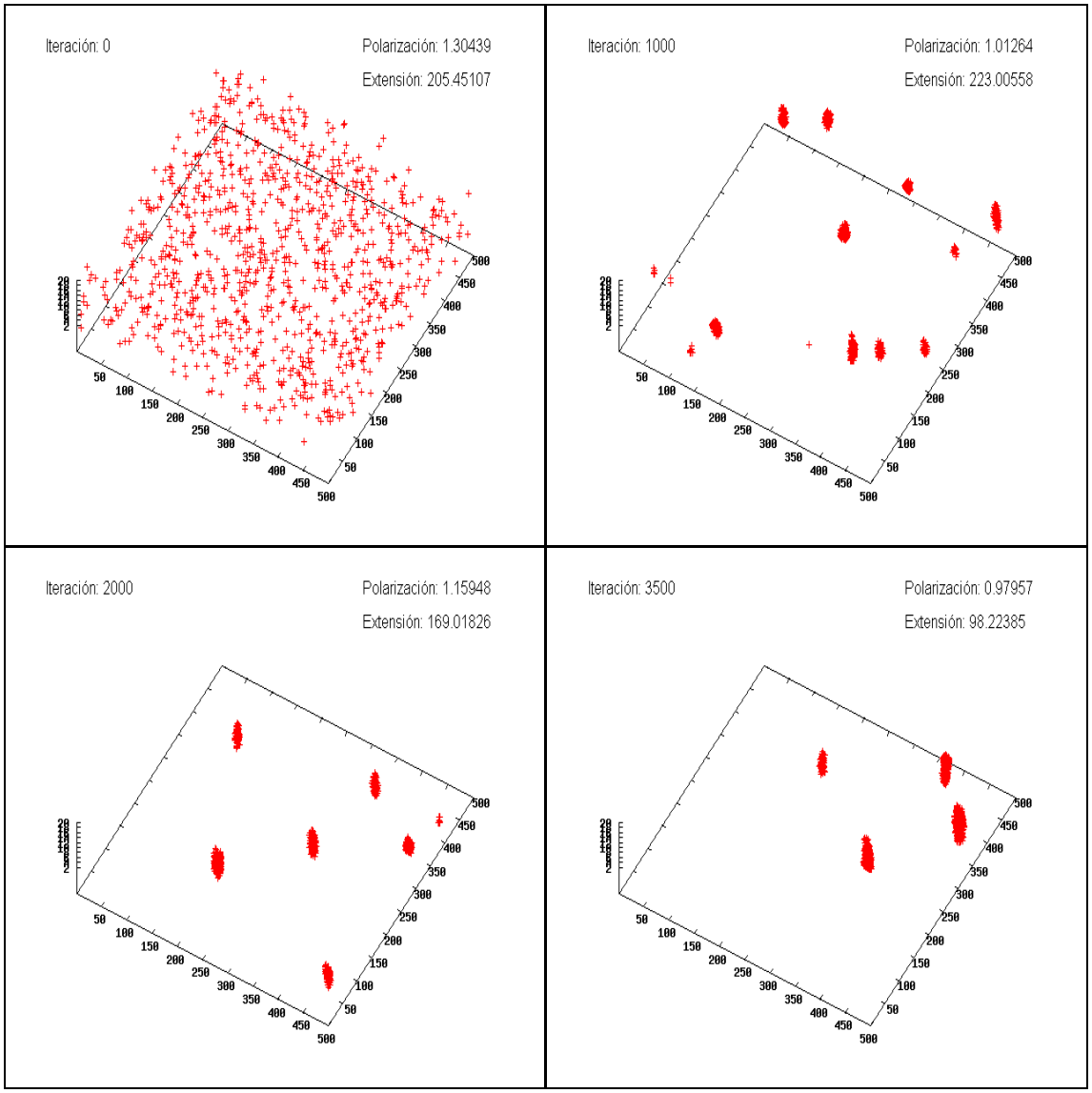
Aumentar al doble y reducir a la mitad el número de vecinos a escoger

En esta prueba se va a comprobar que implicaciones tiene cambiar el número de vecinos a escoger para el cálculo de la reacción final de los peces. Para ello, se realizarán dos simulaciones, una aumentado al doble el número de vecinos y una segunda reduciéndolo a la mitad (respecto a los tamaños originales).

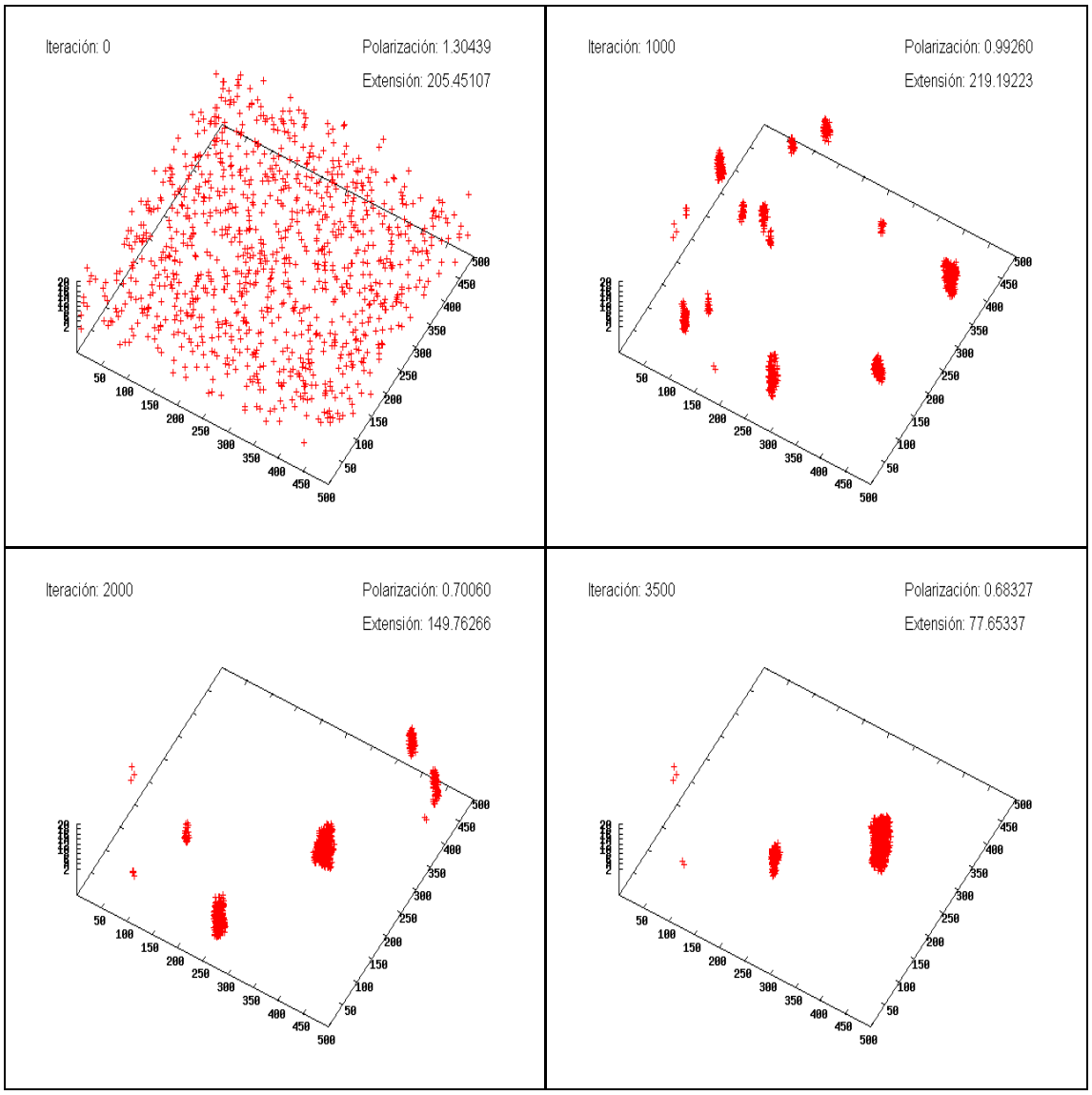
Los datos de la 2 simulaciones son:

- 1000 peces, 500x500x20 de resolución y 3500 iteraciones
- La distribución de los peces será aleatoria en todo el espacio de simulación.

En las siguientes gráficas se mostrará la evolución de la simulación desde la posición inicial hasta la iteración 3500 para el caso de que el número de vecinos sea el doble de lo que lo son habitualmente:



Mediante el siguiente conjunto de gráficas se va a mostrar la evolución de la simulación desde la posición inicial hasta la iteración 3500 para el caso de que el número de vecinos sea la mitad de lo que lo son habitualmente:



Viendo el resultado de las pruebas, se puede apreciar que en ambos casos no se consigue llegar a resultados demasiado buenos. En todo caso, parece que para el caso de la mitad de peces se obtienen mejores resultados que para el caso de cuando es el doble.

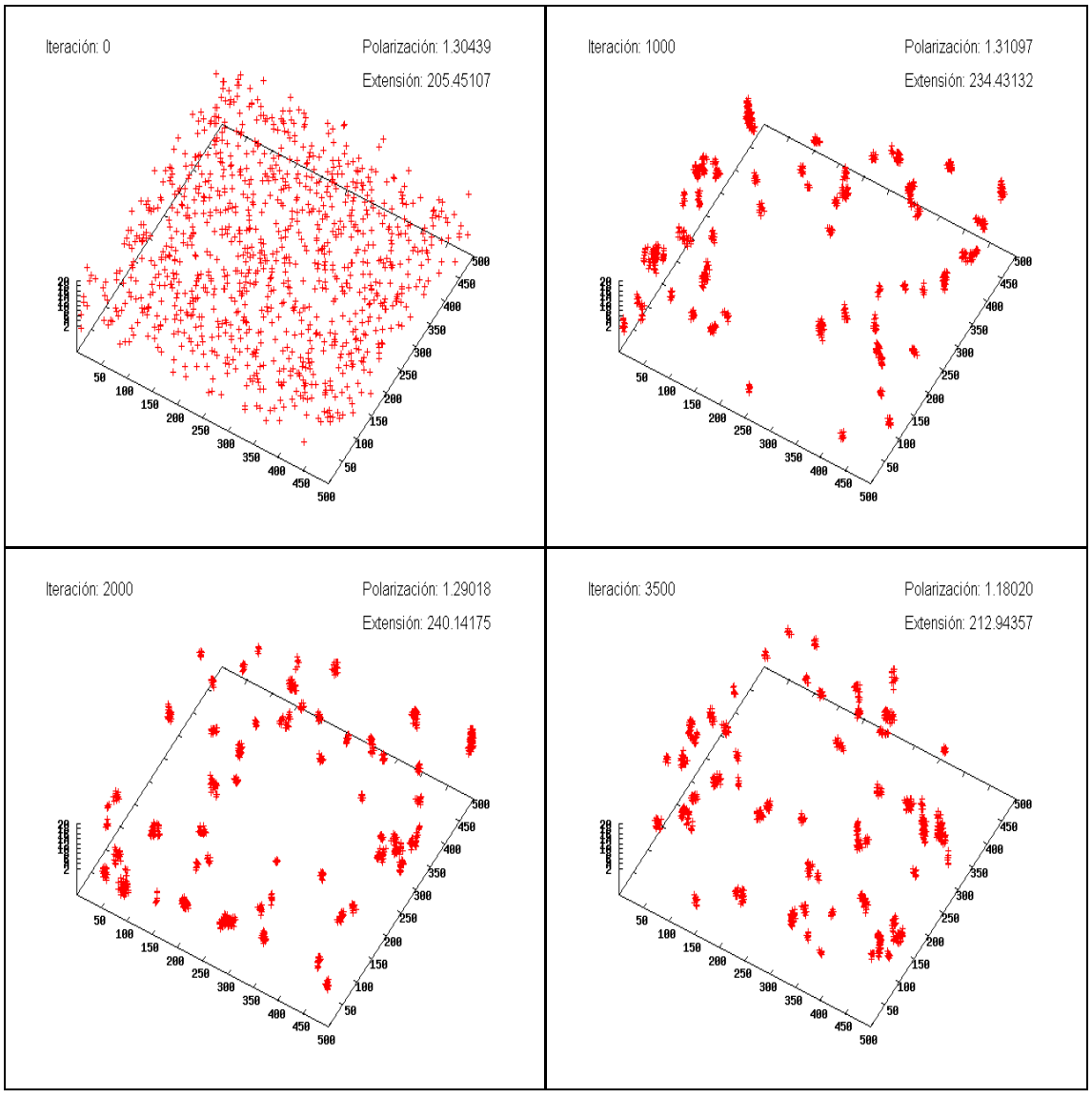
Cambiar el método de elección de vecinos a prioridad por distancia (en lugar de prioridad por frontalidad)

En esta prueba se va a comprobar como se comporta la elección de vecinos por el criterio de la prioridad por distancia.

Los datos de la simulaciones son:

- 1000 peces, 500x500x20 de resolución y 3500 iteraciones
- La distribución de los peces será aleatoria en todo el espacio de simulación.

En el siguiente conjunto de gráficas se va a mostrar la evolución de la simulación desde la posición inicial hasta la iteración 3500:



Al comprobar los resultados obtenidos se podría decir que son más bien malos si se comparan con los que proporciona el criterio de elección por frontalidad.

6.2 Pruebas a nivel de software

Comparativa de tiempos de ejecución en función del número de particiones

En esta prueba se va a comprobar la diferencia de velocidad en la ejecución del simulador para el caso de que se particione el espacio de simulación respecto a que no se haga. Para ello, se realizarán 14 simulaciones, una primera con una única partición (la utilizada por defecto) y el resto con un número concreto de particiones.

Los datos de las 14 simulaciones son:

- 10000 peces, 1000x1000x100 de resolución y 100 iteraciones
- La distribución de los peces será aleatoria en todo el espacio de simulación.

En la siguiente tabla se mostrarán los tiempos de ejecución para cada simulación:

Núm. prueba	Simulación	Tiempo (seg.)
1	1x1x1	1663
2	2x2x2	361
3	3x3x3	199
4	4x4x4	139
5	5x5x5	103
6	6x6x6	82
7	7x7x7	70
8	8x8x8	65
9	9x9x9	65
10	10x10x10	63
11	15x15x15	61
12	20x20x20	60
13	25x25x25	61
14	30x30x30	63

A continuación se podrá ver una gráfica con la evolución del tiempo de ejecución respecto al número de particiones del espacio de simulación:

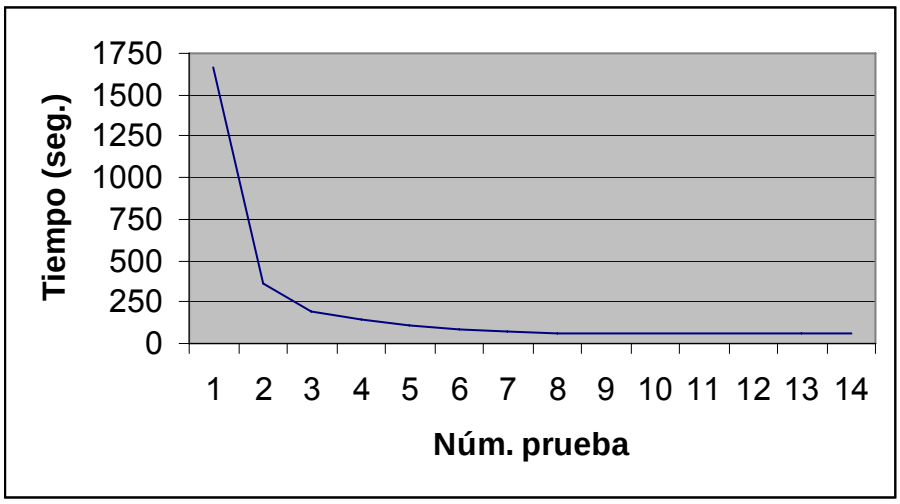


Figura 6.3: Gráfica comparativa del tiempo de ejecución respecto al número de particiones

Por tanto, se deduce que claramente el hecho de particionar mejora notablemente el tiempo de ejecución del simulador y que en este caso a mayor número de particiones, mejores resultados.

7. Conclusiones y líneas de futuro

En este último punto, aparecerán las conclusiones del proyecto, así como las líneas de futuro y modificaciones que se podrían hacer a partir de este proyecto.

7.1 Conclusiones y resultados

Como conclusión, se puede decir que se ha cumplido satisfactoriamente con los objetivos marcados para este proyecto. Además, se ha conseguido llegar al objetivo cumpliendo con la planificación establecida inicialmente.

Se ha conseguido desarrollar un simulador que reproduce de manera bastante fiel, el sistema de desplazamiento que tienen los peces y que tiene como característica el hecho de hacerlo de forma cohesionada y desde el modelado del individuo.

A nivel detallado, los resultados concretos de este trabajo son:

- Se han adquirido conocimientos en el campo de los sistemas biológicos orientados al individuo y más concretamente en el ámbito de los peces (fish schools).
- Se ha desarrollado un simulador de fish schools que consigue simular el comportamiento de los peces al desplazarse.
- Se ha podido verificar que con el simulador se han obtenido los mismos resultados que los que aparecen en la investigación realizada por Huth y Wissel [1] y [2], según se ha demostrado en la parte de pruebas de este documento.
- Se han adquirido nuevos conocimientos en el lenguaje usado para el desarrollo del simulador (C++), como por ejemplo con las clases ya implementadas en el lenguaje para trabajar con listas y vectores.
- Se ha aumentado el conocimientos en el campo de la trigonometría y de los vectores, debido a la necesidad de realizar bastantes cálculos con vectores y elementos trigonométricos para el desarrollo del simulador.

7.2 Posibles trabajos futuros

En este segundo apartado se van a tratar los posibles proyectos futuros que se podrían desarrollan en base al trabajo realizado.

- Se podría hacer una versión del mismo simulador pero para el entorno UNIX o incluso el desarrollo en algún otro lenguaje.

- Un proyecto interesante sería adaptar el actual simulador para que pudiese ser ejecutado de manera distribuida usando varios ordenadores. De esta manera, se conseguiría realizar gran cantidad de cálculos de una manera bastante más rápida de lo que se hace normalmente con un único ordenador.

  Se tendría que tener en cuenta el como dividir el espacio total de simulación en los diferentes equipos y la comunicación entre ellos.

- Otra opción sería la de desarrollar otros proyectos relacionados con los sistemas biológicos orientados al individuo. Por ejemplo, y siguiendo con la línea del presente trabajo, se podrían simular el comportamiento de otros animales como pueden ser los pájaros. A partir de documentación relacionada con el comportamiento de los pájaros, se debería realizar el mismo procedimiento que se ha llevado a cabo para la realización de este proyecto.

  Siguiendo con la misma línea pero basado en las personas, se podrían desarrollar simuladores que a partir del análisis del comportamiento de un cierto grupo de individuos, permitieran tomar decisiones que precisamente ayudaran a las personas. Como ejemplo, se podría mencionar el caso de donde poner las salidas de emergencia de un edificio, en función del comportamiento que toman los individuos cuando se produce un incendio.

Bibliografía

[1] The Simulation of the Movement of Fish Schools (Andreas Huth y Christian Wissel, 1992)

[2] The simulation of fish schools in comparison with experimental data (Andreas Huth y Christian Wissel, 1994)

[3] La ciencia (e ingeniería) computacional, 18 de mayo de 2009 http://www.cytedgrid.org/documentos/un_modelo_de_colaboracion_cientifica.pdf

[4] ¿Qué es un banco de peces?, 25 de mayo de 2009, http://www.laprensa.com.ni/archivo/2006/febrero/11/cabito/conocimiento/

[5] Definición de vectores, 12 de abril de 2009, http://tochtli.fisica.uson.mx/electro/vectores/definici%C3%B3n_de_vectores.htm

[6] Geometría Analítica, http://concurso.cnice.mec.es/cnice2006/material098/geometria/geoweb/geoana2.htm

[7] Ministerio de Educación, Política Social y Deporte, GEOMETRÍA MÉTRICA, 15 de abril de 2009, http://descartes.cnice.mec.es/materiales_didacticos/Geometria_metrica_d3/distrec1.htm

[8] Ministerio de Educación, Política Social y Deporte, GEOMETRÍA MÉTRICA, 15 de abril de 2009, http://www.arandurape.edu.py/Ministerio_de_espana/Descartes/Bach_CNST_2/Geometria_metrica_d3/distrec.htm

[9] Cálculo superior, 26 de abril de 2009
http://www.cidse.itcr.ac.cr/cursos-linea/Algebra-Lineal/algebra-vectorial-geova-walter/node3.html

[10] Distancias y ángulos, 12 de abril de 2009,
http://thales.cica.es/rd/Recursos/rd99/ed99-0543-04/Distancia.html#3.%20Distancia%20entre%20puntos,%20rectas%20y

[11] Geometría del triángulo, 8 de abril de 2009,
http://www.juntadeandalucia.es/averroes/iesarroyo/matematicas/materiales/3eso/geometria/teoriatriangulo/triangulo.htm

[12] VITUTOR, Distancia entre rectas y planos, 12 de abril de 2009,
http://www.vitutor.com/analitica/distancias/distancias.html

[13] Resolución de problemas,10 de abril de 2009,
http://www.lopezdearenas.com/trigonometria/problemas.htm

[14] cplusplus.com,20 de marzo de 2009,
http://www.cplusplus.com/

[15] Curso de C++, 25 de marzo de 2009,
http://www.conclase.net/c/curso/index.php?cap=903e

[16] http://www.medigraphic.com/pdfs/h-gea/gg-2006/gg063i.pdf

[17] http://www.imbiomed.com.mx/1/1/articulos.php?method=showDetail&id_articulo=45672&id_seccion=38&id_ejemplar=4625&id_revista=8

[18] http://www.investigacion.fcs.uc.edu.ve/simuladores.htm

[19] http://www.sim-bio.org/index.html

[20] http://recursos.cnice.mec.es/biosfera/blog/2006/06/proyecto-siveace-un-simulador-biolgico.html

[21] http://www.ellaboratorio.8k.com/computador.htm

Anexo 1: Detalle de las clases utilizadas

1. Detalle de los métodos y principales atributos

Clase PEZ

Atributos principales:

- int id: Identificador del pez.
- float posicion[3]: Posición que ocupa el pez.
- float direccion[3]: Dirección (velocidad) del pez.

Métodos:

- Pez: Constructor de la clase. Se crea el elemento pez con sus atributos de posición y velocidad.
- escribir_fichero: Escribe en los ficheros de posiciones y velocidades (direcciones) fichero de estadísticas los valores correspondientes a cada pez y para cada iteración.
- posiciones: Obtiene la posición de un determinado pez.
- direcciones: Obtiene la dirección (velocidad) de un determinado pez.
- obtener_velocidad: Obtiene la velocidad (rapidez de movimiento) de un determinado pez.
- obtener_id: Obtiene el identificador del pez.
- marcar_tratado: Pone el flag de tratado a un determinado pez para indicar que ya se ha actualizado sus nuevas posiciones y velocidades.
- obtener_tratado: Obtiene el flag de si el pez ha sido actualizado o no.
- quitar_tratado: Pone a “false” el flag de tratado para un determinado pez.
- poner_dirección: Pone la nueva dirección (velocidad) para un determinado pez.
- poner_dirección_nueva: Pone la nueva dirección (velocidad) temporal que posteriormente reemplazará a la dirección (velocidad) definitiva para un determinado pez.
- poner_velocidad_nueva: Pone la nueva velocidad (rapidez de movimiento) temporal que posteriormente reemplazará a la velocidad definitiva para un determinado pez.

- poner_posicion_nueva: Pone la nueva posición temporal que posteriormente reemplazará a la posición definitiva para un determinado pez.
- reemplazar_direcciones: Pone como dirección (velocidad) actual la que figuraba como temporal para un determinado pez.
- ~pez: Destructor de la clase pez.

Clase LISTA

Atributos principales:

- list<pez *> listaok: Lista de elementos de tipo pez que será parte de la base de la estructura de almacenamiento.
- list<pez *>::iterator it: Puntero a la lista de elementos de tipo pez.

Métodos:

- Lista: Constructor de la clase lista.
- meter_alfinal: Crea y mete al final de la lista un elemento de tipo pez.
- borrar_peces: Borra un determinado pez de la lista actual, que previamente lo habremos copiado a su nueva posición de la lista.
- meter_dirección: Pone la dirección del pez al final de la lista.
- obtener_estadísticas: Calcula las estadísticas de grado de misma dirección y de extensión de los peces (separación entre ellos).
- distancia_euclidiana: Calcula la distancia euclidiana entre dos peces.
- distancia_vertical: Calcula la distancia a la que esta un pez respecto a la posición frontal de otro.
- vecinos_distancia: Calcula los vecinos definitivos que se van a escoger en función de la prioridad de distancia entre un pez y el resto.
- vecinos_frontal: Calcula los vecinos definitivos que se van a escoger en función de la prioridad de frontalidad entre un pez y el resto.
- vecinos_radios_reacciones: Calcula que reacción tendrá el pez en función de la posición de sus vecinos.
- cono: Determina que pez esta respecto a otro en el cono de ángulo muerto.
- angulo: Calcula el ángulo (expresado en valores vectoriales) de un pez respecto a otro y que nos servirá para saber si algún pez tapa a otro.

- quitar_tapan: Quita de la lista de vecinos aquellos que son tapados por otros para no tenerlos en cuenta.
- obtener_nueva_dirección: Obtiene una nueva dirección aleatoria para los elementos de la lista (peces) en el caso de que tengan una reacción de búsqueda.
- Repulsión: Calcula la nueva dirección (velocidad) que tendrá el pez para el caso de repulsión.
- orientacion_paralela: Calcula la nueva dirección (velocidad) que tendrá el pez para el caso de orientación paralela.
- atracción: Calcula la nueva dirección (velocidad) que tendrá el pez para el caso de atracción.
- búsqueda: Calcula la nueva dirección (velocidad) que tendrá el pez para el caso de busqueda.
- vecinos_radios_búsqueda: Método que llama al método busca para calcular la nueva dirección en caso de que el pez necesite buscar a otros peces.
- calcular_velocidad: Calcula la nueva velocidad (rapidez de movimiento).
- actualizacion_dirección: Mátodo que llama a reemplazar_direcciones para poner como dirección (velocidad) final la que era temporal.
- calcular_nueva_posicion: Calcula la nueva posición de cada uno de los peces en función de la nueva dirección (velocidad).
- marcar_como_tratado: Marca el flag de tratado para un determinado pez para saber que pez ya se ha recalculado su nueva posición en la lista.
- quitar_como_tratado: Inicializa el flag de tratado cada vez que acabemos una iteración.
- preguntar_tratado: Llama al método de la clase pez para saber si un pez ya ha sido tratado.
- informar_nuevos_valores: Método que actualiza la nueva dirección y velocidad.
- nuevos_valores_pez: Método llamado desde informar_nuevos_valores para actualizar la dirección y la velocidad.
- mirar_posicion0: Comprueba si una posición esta ya ocupada previamente en el proceso de distribución inicial de los peces.

- escribir_fichero_lista: Llama al método escribir_fichero de la clase pez para enviar a fichero las posiciones y velocidades(direcciones) de cada pez.
- ~lista: Destructor de la clase lista.

Clase VECTORA

Atributos principales:

- vector<lista *> myvector: Vector de elementos de tipo lista que será parte de la base de la estructura de almacenamiento.
- vector<lista *>::iterator itv: Puntero al vector de elementos de tipo lista.

Métodos:

- vectora: Constructor de la clase vectora.
- meter_valores: Método para asignar una lista a una posición del vector (correspondiente a un cuadrante), pasándole el apuntador de la lista.
- meter_vecinos_xup: Método para asignar una lista a una posición del vector de vecinos (correspondiente a un cuadrante) que ocupan la parte superior de la coordenada X.
- meter_vecinos_xdown: Método para asignar una lista a una posición del vector de vecinos (correspondiente a un cuadrante) que ocupan la parte inferior de la coordenada X.
- meter_vecinos_yup: Método para asignar una lista a una posición del vector de vecinos (correspondiente a un cuadrante) que ocupan la parte superior de la coordenada Y.
- meter_vecinos_ydown: Método para asignar una lista a una posición del vector de vecinos (correspondiente a un cuadrante) que ocupan la parte inferior de la coordenada Y.
- meter_vecinos_zup: Método para asignar una lista a una posición del vector de vecinos (correspondiente a un cuadrante) que ocupan la parte superior de la coordenada Z.
- meter_vecinos_zdown: Método para asignar una lista a una posición del vector de vecinos (correspondiente a un cuadrante) que ocupan la parte inferior de la coordenada Z.

- asignar_tamanyo: Asigna tamaño a las posiciones de vectores de vecinos, correspondiente al total de partición.
- obtener_estadísticas: Método que calcula las estadísticas de grado de similitud de dirección y de distancia entre peces a nivel de vector, es decir, el proceso recorre el vector de posiciones, para acceder a las listas y posteriormente a los atributos de los peces.
- posicion_vector_null: Devuelve si una posición del vector está vacía.
- posicion_vector_vecinos_null_xup: Devuelve si una posición del vector de vecinos de X superior está vacía.
- posicion_vector_vecinos_null_xdown: Devuelve si una posición del vector de vecinos de X inferior está vacía.
- posicion_vector_vecinos_null_yup: Devuelve si una posición del vector de vecinos de Y superior está vacía.
- posicion_vector_vecinos_null_ydown: Devuelve si una posición del vector de vecinos de X inferior está vacía.
- posicion_vector_vecinos_null_zup: Devuelve si una posición del vector de vecinos de Z superior está vacía.
- posicion_vector_vecinos_null_zdown: Devuelve si una posición del vector de vecinos de Z inferior está vacía.
- obtener_direccion_lista: Devuelve la dirección del puntero de la lista que se encuentra en una determinada posición del vector.
- posicion_del_vector: Método que calcula la posición final del vector a partir a partir de las coordenadas X,Y y Z de lo que sería la matriz.
- posicion_origen_vector: Método que calcula las coordenadas X,Y y Z de la matriz, a partir de una posición del vector dada.
- obtener_direccion_lista_xup: Devuelve la dirección del puntero de la lista que se encuentra en una determinada posición del vector de vecinos de X superior.
- obtener_direccion_lista_xdown: Devuelve la dirección del puntero de la lista que se encuentra en una determinada posición del vector de vecinos de X inferior.

- obtener_direccion_lista_yup: Devuelve la dirección del puntero de la lista que se encuentra en una determinada posición del vector de vecinos de Y superior.
- obtener_direccion_lista_ydown: Devuelve la dirección del puntero de la lista que se encuentra en una determinada posición del vector de vecinos de Y superior.
- obtener_direccion_lista_zup: Devuelve la dirección del puntero de la lista que se encuentra en una determinada posición del vector de vecinos de Z superior.
- obtener_direccion_lista_zdown: Devuelve la dirección del puntero de la lista que se encuentra en una determinada posición del vector de vecinos de Z inferior.
- inicializar_direccion_lista_xup: Inicializa la posición del vector de la lista que se encontraba en una determinada posición del vector de vecinos de X superior.
- inicializar_direccion_lista_xdown: Inicializa la posición del vector de la lista que se encontraba en una determinada posición del vector de vecinos de X inferior.
- inicializar_direccion_lista_yup: Inicializa la posición del vector de la lista que se encontraba en una determinada posición del vector de vecinos de Y superior.
- inicializar_direccion_lista_ydown: Inicializa la posición del vector de la lista que se encontraba en una determinada posición del vector de vecinos de Y inferior.
- inicializar_direccion_lista_zup: Inicializa la posición del vector de la lista que se encontraba en una determinada posición del vector de vecinos de Z superior.
- inicializar_direccion_lista_zdown: Inicializa la posición del vector de la lista que se encontraba en una determinada posición del vector de vecinos de Z inferior.
- inicializar_vectores_posx: Inicializa el vector correspondiente de controlar que no se sitúen más de un pez en la misma posición, para la coordenada X.

- inicializar_vectores_posy: Inicializa el vector correspondiente de controlar que no se sitúen más de un pez en la misma posición, para la coordenada Y.
- inicializar_vectores_posz: Inicializa el vector correspondiente de controlar que no se sitúen más de un pez en la misma posición, para la coordenada Z.
- vecinos: Método inicial del cálculo de reacciones del pez a partir de la situación de sus vecinos y que lo hace a partir de llamar al método vecinos_1. También hace cálculos que aparecerán posteriormente en las estadísticas.
- vecinos_1: Método principal que calcula las nuevas direcciones (velocidades) de cada pez, basándose en los vecinos del mismo cuadrante, así como los que pudiera haber en otros cuadrantes. Una vez pre-seleccionados los vecinos, llama a otras funciones que eligen los vecinos definitivos y calculan la reacción final en función de la situación de cada pez vecino.
- Simulacion: Método principal de las simulaciones, encargado de calcular las nuevas posiciones a partir de las velocidades calculadas anteriormente, reorganizar la estructura de datos moviendo los peces a las listas correspondientes según su nueva posición y finalmente se encargará también de limpiar las listas de vecinos de la anterior iteración.
- simulacion_lista: Método llamado desde Simulacion y que se encarga de la actualización de las nuevas posiciones y de reestructurar la organización de peces, moviendo los mismos a la lista de la posición del vector que le corresponda según sus nuevas coordenadas.
- simulacion_lista_vecinos: Método que limpia las listas de vecinas anteriores para ser usadas nuevamente en la nueva iteración. También controla que las nuevas posiciones que se acaban de calcular no estén repetidas mediante una lista de posiciones.
- dev_posicion_vector: Obtiene la posición del vector concreta que ocuparía un pez con unas coordenadas de posición concretas.
- reorganizar_estructura: Método que reposiciona a los peces según su nueva posición. En función de sus nuevas coordenadas y en el caso de que supongan un cambio de cuadrante, el método borra el pez que cuelga de la lista de la posición concreta del vector, para añadirse a la lista que cuelga de la posición del vector correspondiente a las nuevas coordenadas del pez.

- vecinos_simulacion: Método que controla que las nuevas posiciones de los peces no sean solapadas por otro pez. Esto se realiza mediante la creación de una lista en donde se irán añadiendo las nuevas coordenadas de los peces, para que posteriores peces no ocupen esas mismas posiciones.
- borrar_listas_vecinas: Método para borrar el contenido de las listas de vecinos, para que en la siguiente iteración se recalculen de nuevo.
- posicion_libre: Método para comprobar si la nueva posición de un pez está ocupada y en caso de estarlo, busca una contigua a la misma.
- escribir_fichero: Proceso encargado de escribir en fichero los valores de estadísticas de grado de similitud de dirección y de separación entre peces.
- dev_cuadrante: Devuelve el cuadrante al que le pertenece un pez según la definición original del numero de partes en las que dividimos el espacio de simulación y según las coordenadas de posición que tenga el pez.
- mirar_posicion1: Comprueba si una posición esta ya ocupada en el proceso de calculo de nuevos posiciones en las diversas iteraciones que se vayan produciendo para cada uno de los peces.
- ~vectora: Destructor de la clase vectora

Clase POSICI_CLASE

Atributos principales:

- list<int> posiciones_v: Lista de enteros para guardar las coordenadas de las posiciones.

Métodos:

- posici_clase: Constructor de la clase posici_clase.
- meter_dirección: Método para introducir elementos en la lista de posiciones ocupadas y así detectar solape de posiciones en el espacio de simulación.
- ~posici_clase: destructor de la clase posici_clase.

Clase GEN_NUMEROS

Atributos principales:

- int numero_peces: Número de peces de la simulación.
- int tamanyo_x,tamanyo_y,tamanyo_z: Dimensiones del espacio de simulación.

Métodos:

- gen_numeros: Constructor de la clase gen_numeros que nos servirá para generar la semilla de generación de números aleatorios.
- obtener_numero_x: Obtiene un valor aleatorio para la coordenada X de la posición del pez.
- obtener_numero_y: Obtiene un valor aleatorio para la coordenada Y de la posición del pez.
- obtener_numero_z: Obtiene un valor aleatorio para la coordenada Z de la posición del pez.
- obtener_numeros_rad: Obtiene la configuración de las posiciones iniciales de los peces en el caso de que se use el método de inicialización mediante separación en distancia entre ellos.
- obtener_numero_rads_x: Obtiene un valor aleatorio para la coordenada X de la posición del pez en el caso de que el método de inicialización de peces sea mediante separación en distancia entre ellos.
- obtener_numero_rads_y: Obtiene un valor aleatorio para la coordenada Y de la posición del pez en el caso de que el método de inicialización de peces sea mediante separación en distancia entre ellos.
- obtener_numero_rads_z: Obtiene un valor aleatorio para la coordenada Z de la posición del pez en el caso de que el método de inicialización de peces sea mediante separación en distancia entre ellos.
- ~gen_numeros: destructor de la clase gen_numeros

2. Algoritmo basado en las llamadas a los módulos principales

En este apartado se va a detallar a que módulos se llama a lo largo de los pasos que constituye el algoritmo. Hay que tener en cuenta, las siguiente nomenclaturas:

- P->: Método de la clase PEZ
- L->: Método de la clase LISTA
- V->: Método de la clase VECTORA
- G->: Método de la clase GEN_NUMEROS

- Lectura del fichero de configuración
    - Según el método de inicialización de los peces definido en la configuración, hacer las correspondientes acciones:
        - Para el caso de definir las posiciones de los peces a partir de un fichero no tendrá ninguna función especial, sino que comparte las comunes para los 4 métodos.
        - Para el caso de definir las posiciones de los peces de forma aleatoria tanto para todo el espacio de simulación como en parte de el:
          G->obtener_numero_x
          G->obtener_numero_y
          G->obtener_numero_z
          V->posicion_libre
          L->obtener_nueva_dirección
        - Para el caso de definir las posiciones de los peces de forma aleatoria pero separadas por un determinado espacio:
          G->obtener_numeros_rad
          G-> obtener_numero_rads_x
          G-> obtener_numero_rads_y
          G-> obtener_numero_rads_z
        - Funciones comunes para todos los tipos de inicialización:
          V->posicion_libre
          V->posicion_vector_null
          V->meter_valores
          L->meter_alfinal
          V->obtener_direccion_lista
          V->dev_cuadrante
          V->posicion_vector_vecinos_null_xup
          V->meter_vecinos_xup

V->obtener_direccion_lista_xup
V->posicion_vector_vecinos_null_xdown
V->meter_vecinos_xdown
V->obtener_direccion_lista_xdown
V->posicion_vector_vecinos_null_yup
V->meter_vecinos_yup
V->obtener_direccion_lista_yup
V->posicion_vector_vecinos_null_ydown
V->meter_vecinos_ydown
V->obtener_direccion_lista_ydown
V->posicion_vector_vecinos_null_zup
V->meter_vecinos_zup
V->obtener_direccion_lista_zup
V->posicion_vector_vecinos_null_zdown
V->meter_vecinos_zdown
V->obtener_direccion_lista_zdown
L->meter_dirección

V->vecinos
V->obtener_estadisticas
V->escribir_fichero

Repetir tantas veces como iteraciones se quieran hacer
V->simulación
V->vecinos
V->obtener_estadísticas
V->escribir_fichero

V->simulacion
Para cada posición del vector:
V->simulacion_lista
V->borrar_listas_vecinas
V->simulacion_lista_vecinos

V->simulacion_lista
Para cada posición de la lista:
L-> actualizacion_dirección
L->calcular_nueva_posicion
V->posicion_libre
V->poner_posicion_nueva
V->reorganizar_estructura

V->borrar_listas_vecinas
V->inicializar_direccion_lista_xup
V-> inicializar_direccion_lista_xdown
V-> inicializar_direccion_lista_yup
V-> inicializar_direccion_lista_ydown
V-> inicializar_direccion_lista_zup
V-> inicializar_direccion_lista_zdown

V->simulacion_lista_vecinos
Para cada posición de la lista:
    L->quitar_como_tratado
    V->vecinos_simulacion

V->vecinos
Para cada posición del vector:
    V->vecinos_1

V->vecinos_1
Para cada vecino:
    L-> distancia_euclidiana
    L->cono
    L->meter_dirección
    L->quitar_tapan
    L-> vecinos_frontal / L-> vecinos_distancia
    L-> vecinos_radios_reacciones / L->vecinos_radios_busqueda
    L-> calcular_velocidad
    L->informar_nuevos_valores

L-> vecinos_frontal
    L-> distancia_vertical

L-> vecinos_distancia
    L-> distancia_euclidiana

L-> vecinos_radios_reacciones
    L-> distancia_euclidiana
    L-> repulsión
    L-> orientacion_paralela
    L-> atraccion

L->vecinos_radios_busqueda
    L-> busqueda

V->obtener_estadisticas
Para cada posición del vector:
    L->obtener_estadísticas

V->escribir_fichero
Para cada posición del vector:
    L-> escribir_fichero_lista

L-> escribir_fichero_lista
Para cada posición de la lista:
    P-> escribir_fichero

Anexo 2: Manual de usuario

Con este manual se va a poder instalar, configurar y ejecutar el simulador.

1. Instalación

Lo primero a indicar en este manual de usuario van a ser los elementos que forman este simulador. Por un lado estará el simulador en si (ejecutable) y por otro lado habrá un fichero de configuración.

Ambos ficheros deben estar en la misma carpeta, cuyo nombre es indiferente para el correcto funcionamiento del simulador.

El fichero ejecutable tendrá como nombre “simufish.exe” y necesitará obligatoriamente de la presencia del fichero de configuración, ya que de este leerá los parámetros sobre los que ejecutar el simulador. El nombre del fichero de configuración tendrá el nombre de “simufish.cfg”

2. Configuración

A continuación se detallará el contenido del fichero de configuración, que dependiendo de los valores que tenga hará que se defina la manera en la que se ejecute la simulación.

Entre otras cosas, se podrá establecer la configuración de la distribución de los peces inicialmente, el número de iteraciones o por ejemplo el número de peces.

También se podrán definir elementos más internos de la configuración como son la distancia entre radios (que en principio no debería modificar si se quieren obtener los resultados esperados) o el número de vecinos a tener en cuenta en la reacción final del pez.

Una vez visto la utilidad del fichero de configuración, se pasará a mostrar un ejemplo del mismo:

```
numero_peces: 1000
numero_iteraciones: 350
resolucion: 500 500 20
particion: 1 1 1
numero_vecinos: 4
tipo_vecinos: 0
tamano_pez: 3
radios: 0.5 2 5
angulo_muerto: 30
metodo_colocacion: 0
porcentaje_colocacion: 10 10 10
separacion_peces: 2
lugar_colocacion: 50 50 50
direct_salida: C:\Simufish\out\
fich_entrada: C:\fichero_in\fichero1.txt
```

Hay que tener en cuenta que todos los parámetros tienen que aparecer en el fichero (la definición del parámetro) y en el orden en que aparecen en el fichero de configuración del ejemplo.

A continuación se realizará la explicación de todos los parámetros que incluye:

- numero_peces: Indica el número de peces que va a tener el simulador. Deberá ser un número entero mayor que 0. Se ha de informa obligatoriamente.
- numero_iteraciones: Mediante este parámetro se informará del número de iteraciones que se van a producir en la simulación. Si no se quiere que haya iteraciones y por tanto mostrar solo la posición inicial, se deberá poner un 0. Si por el contrario se quiere hacer la simulación normalmente, habrá que poner un número entero mayor que 0. Se ha de informa obligatoriamente.

- resolución: La resolución indicará la extensión que tendrá el espacio de simulación sobre el que se moverán los peces. Al tratarse de un espacio en 3 dimensiones, se deberá indicar el tamaño para las tres planos (x, y, z). Se deberán indicar en el fichero separados con un espacio y en el orden anterior. Así pues, en el fichero de ejemplo se puede ver que la resolución será de 500 para la coordenada 'x', 500 para la coordenada 'y' y 20 para coordenada 'z'. Para las tres coordenadas deberán ser valores enteros mayores que 0 y deberán ser informados obligatoriamente.

- particion: Se indicará el número de partes en las que se quiere particionar el espacio de simulación (definido con el anterior parámetro). Si se quiere tratar el espacio de simulación como un todo y de forma única, se deberá definir el valor 1 para los tres planos (x, y, z). Si por el contrario se quiere dividir el espacio en varias partes se deberá indicar en cuantas se desea hacer para cada uno de los planos. Así por ejemplo, si se pone el valor "3 2 1" indicaría que se quiere dividir el espacio de simulación en 3 partes para el plano 'x', en 2 partes para el plano 'y' dejarlo en una parte para el plano 'z'. Para las tres coordenadas deberán ser valores enteros mayores que 0 y deberán estar informados obligatoriamente.

- numero_vecinos: Mediante este parámetro podremos definir el número de vecinos que queremos que se tengan en cuenta para la reacción final del pez. En principio el valor debería ser de 4 (que se corresponde con el valor que más se ajusta a la forma de actuar de los peces), pero podría variarse para ver el nuevo comportamiento del simulador. Deberá ser un número entero mayor que 0. Se ha de informa obligatoriamente.

- tipo_vecinos: Se informará la manera en la que escoger la prioridad a la hora de seleccionar a los vecinos. Habrá dos posibles valores, 0 para el caso de la prioridad frontal y 1 para el caso de la prioridad por distancia. Por tanto, si existe un mayor número de posibles vecinos de los indicados en el anterior parámetro (numero_vecinos), el simulador se quedará con aquellos que o bien estén más en frente del pez origen (valor 0, prioridad frontal) o bien estén a una distancia menor del pez origen (valor 1, prioridad por distancia). Por defecto y como valor que más se aproxima al comportamiento de los peces, habrá que poner como método el de la prioridad frontal, o lo que es lo mismo "tipo_vecinos: 0". Por

tanto, los dos valores posibles con los que el simulador funcione correctamente serán el “0” y el “1”. Se ha de informar obligatoriamente.

- tamano_pez: Mediante este parámetro se indicará el tamaño del pez (expresado como longitud y en la misma magnitud que el resto de parámetros). En principio, un valor con el que se van a conseguir buenos resultados es el de 3. Deberá ser un número entero mayor que 0. Se ha de informa obligatoriamente.

- radios: Mediante este parámetro se definirá el tamaño de los radios que a su vez implicará que se seleccione un determinado tipo de reacción. Así pues, habrá tres diferentes radios con tres diferentes tipos de reacciones. El primero de ellos será el r1 y marcará la distancia en la que un pez tiene una reacción de repulsión respecto a sus vecinos. El segundo radio será el r2 y marcará la distancia en la que un pez tiene una reacción de orientación paralela respecto a sus vecinos. Finalmente estará el tercer radio r3 que servirá tanto para delimitar a los vecinos de un pez como para marcar la distancia en la que un pez tiene una reacción de atracción. Los tres valores por defecto y que en principio no deberían tocarse (ya que supone el comportamiento normal de los peces) serían 0.5 para r1, 2 para r2 y 5 para r3. Sólo será necesario modificar estos valores si se quiere probar el comportamiento del simulador en diferentes circunstancias a las normales. Se deberá separar cada valor del radio mediante un espacio y el orden será de r1, r2 y r3. Deberá ser un número mayor que 0. Se ha de informa obligatoriamente.

- angulo_muerto: Mediante este parámetro se definirá el valor que tendrá el ángulo muerto de visión del pez. Dicho ángulo se encuentra localizado justo detrás de donde se encuentra el pez y correspondería a la zona en la que el pez no puede ver lo que hay. En principio, el valor por defecto y que no debería tocarse (ya que supone el comportamiento normal de los peces) es de 30 (es decir 30 grados). Sólo se modificará este valor si se quiere probar el comportamiento del simulador en diferentes circunstancias a las normales. Deberá ser un número mayor que 0. Se ha de informa obligatoriamente.

- metodo_colocacion: Se indicará la manera en que inicialmente se colocarán los peces sobre el espacio de simulación.

Habrá 4 diferentes posibilidades que serán:

- Posicionamiento aleatorio distribuido por todo el especio de simulación: Este tipo de inicialización se basará en el hecho de que todos los peces estarán distribuidos a lo largo de todo el espacio de simulación. Será la opción más genérica de inicialización y en principio la que se usará por defecto.

  Un ejemplo de este tipo de inicialización sería el siguiente:

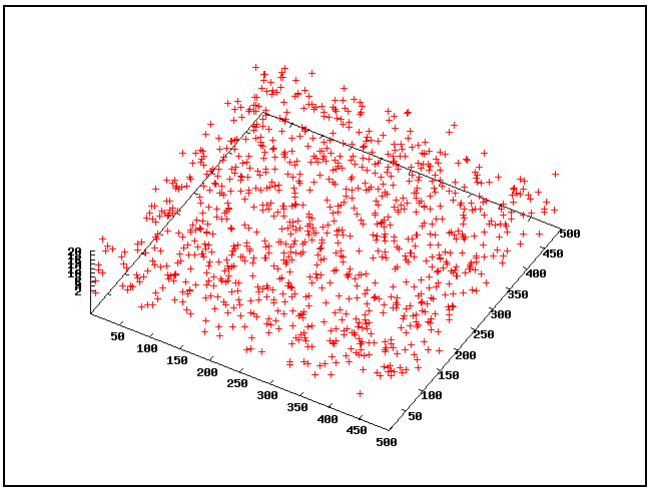


Figura anex.1: Distribución inicial aleatoria

- Posicionamiento aleatorio distribuido en un determinado porcentaje sobre el total del espacio de simulación y situado en una zona concreta de ese espacio: En este caso, el propósito de este tipo de inicialización es concentrar los peces en una determinada zona, en lugar de que estén distribuidos por todo el espacio de simulación.

  Para ello, habrá que basarse en dos variables (parámetros):

  - porcentaje_colocacion: Marca que porcentaje del espacio de simulación estará ocupado por peces o lo que es lo mismo, servirá para indicar si se quiere que los peces estén juntos o separados entre sí. Estará compuesto por 3 valores que representarán el porcentaje para los coordenadas ‘x’, ‘y’ y ‘z’. Así por ejemplo, si este parámetro tiene el valor “50 50 50” querrá decir que todos los peces estarán agrupados ocupando la mitad del espacio de simulación, mientras que si tienen el valor del fichero del ejemplo (“10 10 10”) querrá decir que los peces estarán distribuidos a lo largo del 10%

del espacio de simulación y que consecuentemente estarán muy juntos entre sí. Para las tres coordenadas deberán ser valores enteros mayores que 0 y deberán ser informados obligatoriamente.

- lugar_colocacion: Indica el lugar en el que se situará el grupo de peces respecto al total del espacio de simulación. Se definirá para las tres coordenadas, así por ejemplo si se quiere que los peces estén situados en el centro del espacio de simulación se deberá poner “50 50 50” (o lo que es lo mismo, que estén repartidos alrededor de la posición que representa el 50% del espacio de simulación para las tres coordenadas). Para las tres coordenadas deberán ser valores enteros mayores que 0 y deberán ser informados obligatoriamente.

Este tipo de inicialización servirá para probar como se comporta la simulación en casos particulares de posicionamiento y distribución de los peces.

Unos ejemplos de este tipo de inicialización serían los siguientes:

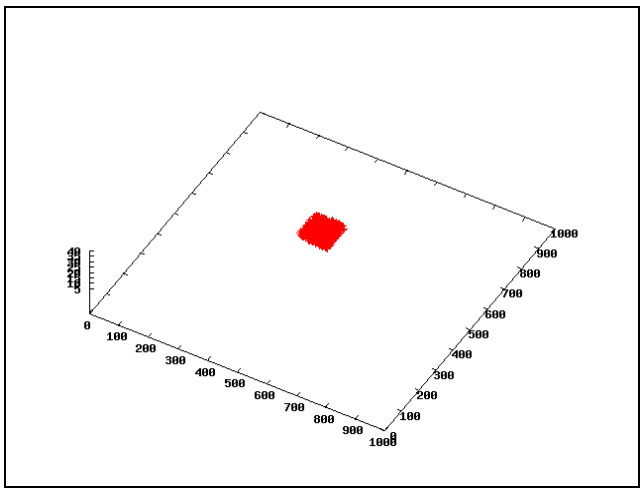


Figura anex.2: Distribución inicial aleatoria por porcentaje en el centro

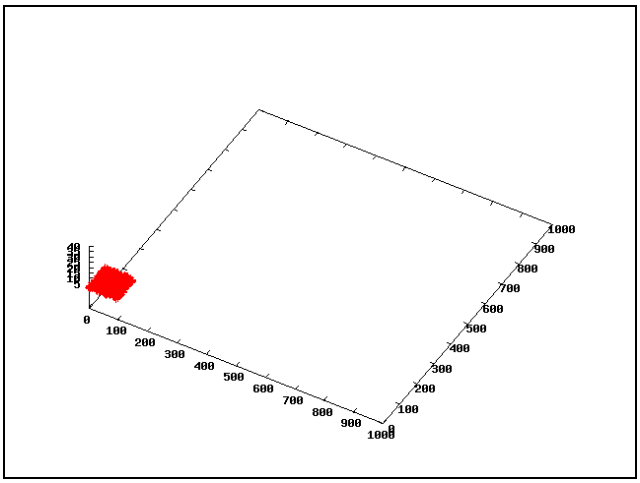


Figura anex.3: Distribución inicial aleatoria por porcentaje en una esquina

- Posicionamiento aleatorio distribuido en función de la distancia entre peces y situado en una zona concreta del espacio de simulación: Para este tipo de inicialización, los peces se distribuirán en el espacio de simulación en función de la distancia que se quiere que haya entre ellos. La finalidad será provocar inicialmente una determinada reacción de los peces en función precisamente de esa distancia que se va a indicar en el siguiente parámetro:
    - separacion_peces: Distancia entre un pez y su vecino más próximo. Así pues, si como en el ejemplo se informa con el valor '2', indicará que entre un pez y su vecino existe una distancia de 2 posiciones. Según lo explicado en la memoria, esa distancia de 2 implicará que haya una acción de orientación paralela y que por tanto los peces tiendan a juntarse. Será un valor entero mayor que 0 y deberá estar informado obligatoriamente.

  Además de la separación entre peces, también se deberá indicar el lugar en el que se quiere situar al conjunto de los peces:
    - lugar_colocacion: Este parámetro actuará de la misma manera que lo hace en el tipo de inicialización anterior (posicionamiento aleatorio distribuido en un determinado porcentaje sobre el total del espacio de simulación y situado en una zona concreta del espacio de simulación), es decir, se indicará la posición en la que se

colocarán los peces mediante 3 coordenadas. Sus valores deberán ser enteros mayores que 0 y deberán ser informados obligatoriamente.

Este tipo de inicialización servirá para probar como se comporta la simulación en casos particulares de posicionamiento y distribución de los peces, partiendo de una reacción inicial provocada por la distancia existente entre ellos.

Un ejemplo de este tipo de inicialización serían los siguientes:

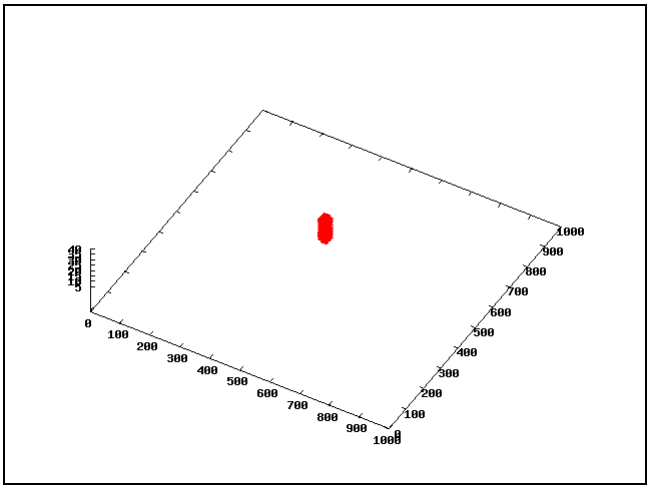


Figura anex.4: Distribución inicial aleatoria por separación entre peces en centro

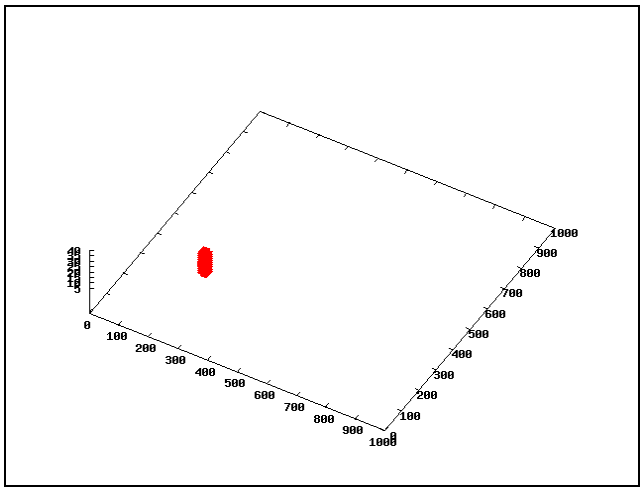


Figura anex.5: Distribución inicial aleatoria por separación entre peces en esquina

- Posicionamiento a partir de la definición de los peces mediante fichero de entrada: Gracias a este tipo de parametrización, se podrá definir el

conjunto de peces que van a aparecer en la simulación, indicando su posición y orientación inicial. Al indicar este tipo de inicialización, se deberá informar el nombre del fichero del que se leerán los datos de los peces a simular:

- fich_entrada: Se indicará la ruta y nombre del fichero que contendrá la definición de los peces a cargar en el simulador. Sólo se tendrá en cuenta este valor para el caso en el que el método de colocación sea 3 (situación inicial de peces leída de fichero). Por tanto, este parámetro será solo obligatorio en el caso de que se tenga "metodo_colocacion: 3" y opcional para el resto de métodos de colocación. Se deberá poner la ruta entera (ej. C:\fichero_in\fichero1.txt).

  Un ejemplo de fichero de entrada con la definición de dos peces sería el siguiente:

```
10 2 1
1 1 1 0 1 0
10 2 1 -1 1 0
```

  La primera fila del fichero indicará el tamaño del espacio de simulación definido en el fichero (para las coordenadas 'x', 'y' y 'z'). Así por ejemplo, para el caso de "10 2 1", querrá decir que habrá 10 posiciones de tamaño para la coordenada 'x', 2 para la 'y' y una única para la 'z'. Estos valores se extraerán de la definición de los peces que aparecen en las líneas posteriores del fichero. De esta manera, desde la línea 2 hasta el final del fichero se definirán los peces. Las tres primeras posiciones, indicarán la posición del pez, mientras que las tres ultimas harán referencia a la dirección del mismo. Para el caso del primer pez, se puede ver que la posición es la (1 1 1), mientras que la dirección es la (0 1 0). Si se observa la posición que ocupará el segundo pez (10 2 1) se podrá calcular el tamaño del espacio de simulación que se acabará informando en la primera línea del fichero (10-1+1) (2-1+1) (1-1+1) = (10 2 1).

Una vez visto los 4 tipos, se pasará a definir los valores a poner para seleccionar cada uno de ellos: 0: Posicionamiento aleatorio distribuido por todo el especio de simulación, 1: Posicionamiento aleatorio distribuido en un determinado porcentaje sobre el total del espacio de simulación y situado en una zona concreta del espacio de simulación, 2: Posicionamiento aleatorio distribuido en función de la distancia entre peces y situado en una zona concreta del espacio de simulación y 3: Posicionamiento a partir de la definición de los peces mediante fichero de entrada.

Se ha de informar obligatoriamente.

- direct_salida: Este parámetro indicará el directorio en donde se generarán los ficheros de posiciones, de velocidades y el de estadísticas. Deberá estar informado obligatoriamente y acabado con el carácter de barra invertida "\". (ej. C:\Simufish\out\).

3. Ejecución

Una vez que ya se tiene el fichero de configuración preparado, se pasará a la ejecución del simulador, que consistirá en hacer doble clic en el fichero "simufish.exe".

Signat: Antonio Cruz Vázquez

Bellaterra, 22 de juny de 2009

En la presente memoria se ha recogido de forma escrita el conjunto de fases que se han llevado a cabo en la realización del proyecto consistente en un simulador de sistemas biológicos utilizando modelos orientados al individuo. Concretamente, el sistema biológico representado ha consistido en el movimiento y comportamiento de cohesión que poseen los peces.

En la present memòria s'ha recollit de forma escrita el conjunt de fases que s'han portat a terme en la realització del projecte consistent en un simulador de sistemes biològics utilitzant models orientats a l'individu. Concretament, el sistema biològic representat ha consistit en el moviment i comportament de cohesió que posseïxen els peixos.

In this report has been written the phases that have been done at the project realization consisting of a simulator of biological systems using the individual-oriented models. Specifically, the biological system represented has consisted in the movement and cohesion behavior that fishes have.