TÍTULO (DESCRIPCIÓN CORTA DEL PROYECTO. ENTRE 8 Y 12 PALABRAS)

Mauricio Castaño Uribe
EAFIT
Colombia
mcastanou@eafit.edu.co

José Miguel Gil Valencia
EAFIT
Colombia
jmgilv@eafit.edu.co

Mauricio Toro
Universidad Exabit
Colombia
mtorobe@eafit.edu.co

RESUMEN
Hasta la fecha de hoy el número de abejas en el mundo ha disminuido considerablemente y sigue en decadencia, esto es algo que afecta la mayoría de los ecosistemas en el planeta. Por lo cual se ha tomado la decisión de estudiar estos animales, su papel en los ecosistemas y su importancia para la vida de la tierra. Partiendo de una situación en la cual las abejas no existan y se tenga que producir abejas robots, cual sería un algoritmo de baja complejidad y codicioso que evalúe a las abejas en su entorno y evite un choque entre ellas para hacer un correcto funcionamiento.

Palabras clave
-Distancia
-Choque
-Lista

Palabras clave de la clasificación de la ACM
-Software and its engineering
-Mathematics of computing
-Security and privacy
-Trees
-Enumeration

1. INTRODUCCIÓN

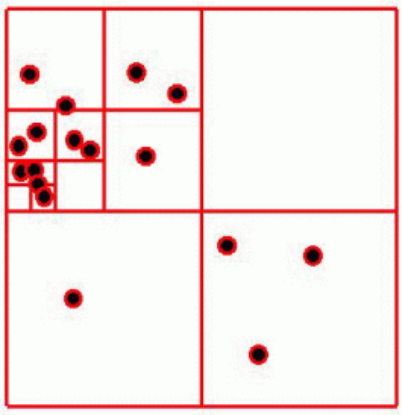
La situación ambiental del planeta está pasando por un momento crítico donde las decisiones que se tomen en estos momentos marcaran el futuro del planeta. Por lo cual en este proyecto se tocará una posible alternativa de mejorar el ambiente mediante las abejas robots. Pero estas necesitan de una buena programación debido a que van a estar por todas partes y se tienen que estar monitoreando frecuentemente para evitar situaciones de peligro, en este caso se realizara un algoritmo que ubique a las abejas y evite que se choquen entre ellas mismas.

2. PROBLEMA

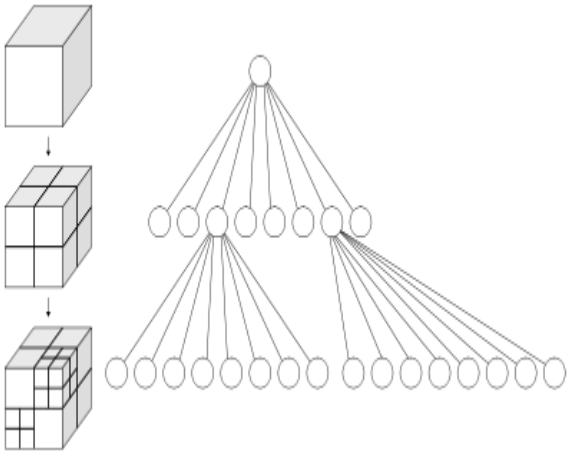
Se piensa construir pequeñas maquinas que logren reemplazar a un gran número de las abejas para polinizar los campos y evitar los grandes cambios que causaría la extinción de dicha especie en la vida de nuestro planeta. La cuestión es que, al manejar un gran número de abejas, debemos de controlar sus movimientos y posiciones para todo momento evitando las colisiones entre ellas.

3. TRABAJOS RELACIONADOS

3.1 QUADTREE Es una forma de estructura la cual se representa con un árbol. Haciendo que cada vez, recursivamente, se generen cuatro subdivisiones y sean del mismo tamaño. Esta estructura se usa para evitar los choques de una manera efectiva, primero se toma un nodo el cual es el plano general y de este nodo salen otros cuatro más pequeño, haciendo que todos los objetos queden separados en áreas más pequeñas y se puedan llevar a casos muy pequeños donde se comparan dos objetos cercanos. [1] [6]

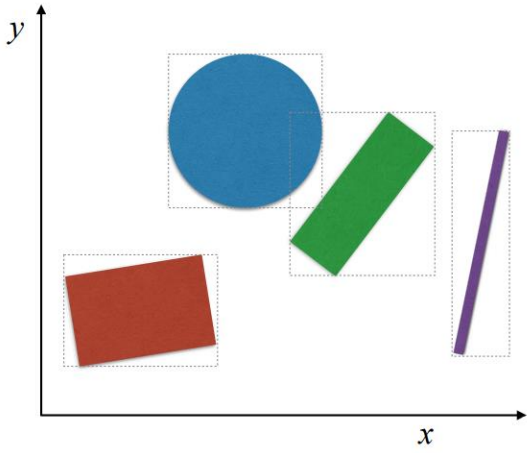


3.2 OCTREE Es un algoritmo con estructura en modo de árbol donde cada nodo tiene 8 sub-nodos, fue creado por Donald Meagher en 1980, es utilizado para particionar el espacio tridimensional, dividiéndolo recursivamente en ocho octantes cada uno representado por un nodo, cada nodo representa un cubo tridimensional y cada sub-nodo es una octava parte del cubo original. [2]

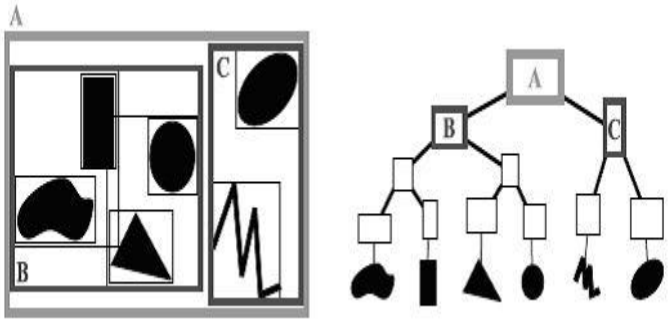


3.3 SWEEP AND PRUNE Es un algoritmo utilizado para la detección de posibles colisiones para limitar el número de solidos que deben verificarse, lo que hace es que guarda la coordenadas x_min y x_max, y_min y y_max, z_min y z_max, ya luego si os solidos se mueven debe de ir

actualizando los datos y si en algún momento existe una opción de posible colisión se manda a revisar utilizando algún otro algoritmo más preciso.[3]



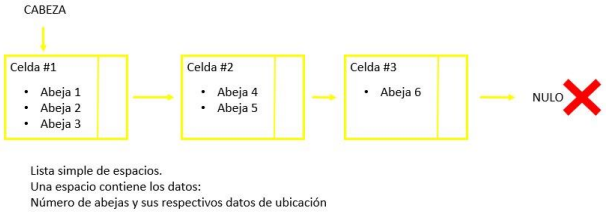
3.4 Boulding Volumen Hierarchy Este algoritmo permite detectar colisiones de una manera efectiva, ya que parte de figuras geométricas que encierran a los nodos. Está basado en la lógica de conjuntos, ya que si un objeto está en un rectángulo y ese rectángulo está en otro más grande, y existe un rectángulo con el cual no se toque, entonces los nodos contenidos en los rectángulos diferente no se van a tocar. [4]



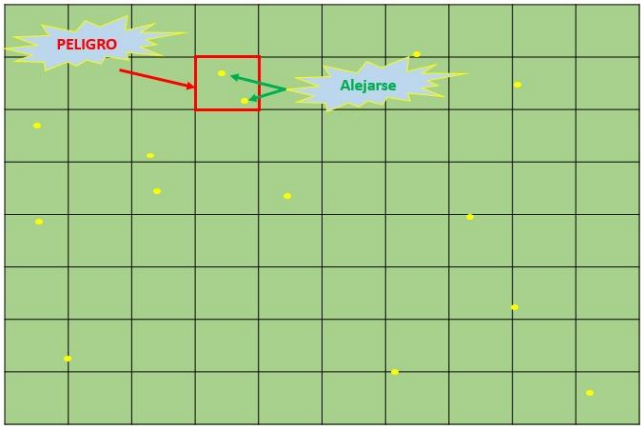
4. Título de la primera estructura de datos diseñada

La estructura de datos que vamos a usar es la de Spatial Hashing, nuestro algoritmo dividirá el espacio en cubos de cierto espacio de modo que la diagonal entre dos vértices opuestos sea igual a la distancia de alerta de colisión, de esta forma cualesquiera dos abejas que se encuentren en ese cubo llamarán a una alerta de colisión, estas serán revisadas en las tablas de hash asignadas a cada uno de los cubos en los que se dividió el espacio.

4.1 Operaciones de la estructura de datos



Lista simple de espacios.
Una espacio contiene los datos:
Número de abejas y sus respectivos datos de ubicación

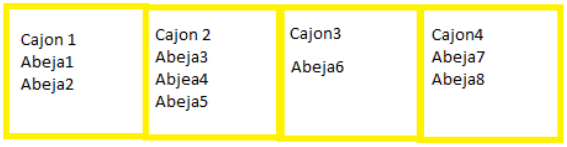


4.2 Criterios de diseño de la estructura de datos

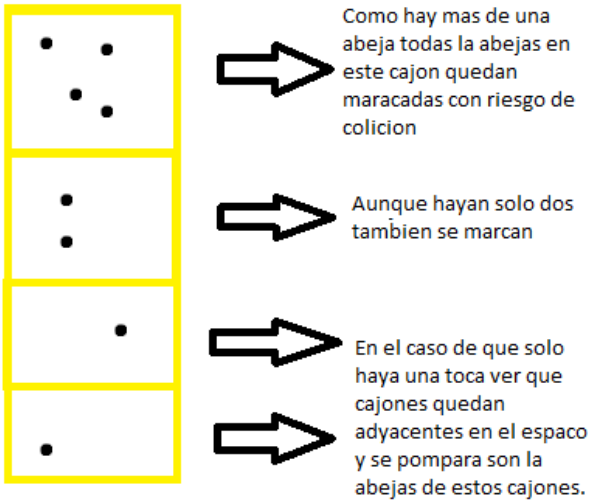
Al principio estábamos convencidos de que el octree iba a ser la estructura de datos que usaríamos, pero luego al seguir investigando diferentes estructuras de datos que nos sirvieran para identificar colisiones en espacios 3D vimos que el Spatial Hashing era una estructuras de datos mucho más eficaz en tiempo y en espacio que los demás que habíamos encontrado, leímos varios blogs y paginas[8][10] de persona que relataban que en sus trabajos compararon varios algoritmos con estructura de datos Spatial Hashing y este claramente les saco mejores resultados que las demás que usaron, aunque el quad-tree y el octree pueden llegar a ser más óptimos en ciertas circunstancias en que pueden eliminar un muy gran espacio en el que no se encuentren abejas pensamos que en otro tipo de situaciones no tan especificas Spatial Hashing nos sirve mucho más para cuando halla abejas ubicadas por la mayoría del espacio monitoreado ya que nos evita revisar el problema con fuerza bruta y no permite comparar las abejas en las tablas de hash ahorrándonos así mucho tiempo para un gran número de abejas.[9]

5.1 Operaciones de la estructura de datos

Primero usamos una función para asignarle a cada abeja un celda en nuestras tablas de hash dependiendo de sus coordenadas.



Ya cuando todas las abejas hayan sido guardadas el algoritmo revisara cada uno de los cajones para ver el número de abejas que contiene cada uno:



Y por ultimo cuando el algoritmo haya revisado todos los cajones y comparado las abejas correspondientes este agregara las abejas marcadas a un arraylist y guardara esos datos en un archivo .txt

5.2 Criterios de diseño de la estructura de datos

Utilizamos las tablas de hash debido a que esta función nos ayuda a organizar la abejas para poder compararlas con mayor facilidad, esto es mucho mas eficiente que comparar todas las abejas con todas, además tener las abejas organizadas es de gran utilidad en la eficiencia de tiempo ya que en nuestro algoritmo todas las abejas que correspondan a la misma celda en nuestra tabla será marcada, nuestro algoritmo también crea una celda caba vez que se lee una abeja a la cual no le corresponda ya una celda, de esta manera evitamos crear un gran numero de celdas que ocupen el espacio completo y solo creamos celdas que ocupen el espacio donde están las abejas.

5.3 Análisis de la complejidad

El método leer tiene complejidad O(n)

El método de guardad tiene complejidad O(n)

El método colisiones tiene complejidad O(n^2) en el peor de lo casos donde todas las abejas este muy separadas entre si y toque compararlas por cajones adyacentes.

El método meterEnCajones tiene complejidad O(n)

Y todas estas complejidades donde n es igual al número de abejas.

6 CONCLUSIONES

Con este trabajo podemos ver como los problemas con un gran numero de datos podemos resolverlos pensando en cómo organizar la información del problema en una estructura de datos para asi manejarla mejor y llegar a una solución en un tiempo optimo y sin gastar excesiva memoria que si los hiciéramos por métodos disitntos a este.

REFERENCIAS

1.https://es.wikipedia.org/wiki/Octree

2.https://moodle201516.ua.es/moodle/pluginfile.php/12368/mod_resource/content/3/vii-07-colisiones.pdf

3.https://en.wikipedia.org/wiki/Sweep_and_prune

4.https://codexlingua.wordpress.com/2010/06/16/gameprogramming-introduccion-a-la-deteccion-decolisiones-en-tiempo-real/

5.https://uva.onlinejudge.org/contests/284-285275bb/m.html

6.https://bensprogstuff.wordpress.com/2012/09/23/broad-phase-collision-detection-spatial-indexing

7./https://www.acm.org/publications/class-2012

8.https://www.gamedev.net/forums/topic/661021-quad-trees-vs-r-trees-vs-spacial-hashmaps/

9.http://zufallsgenerator.github.io/2014/01/26/visually-comparing-algorithms/

10.https://gamedev.stackexchange.com/questions/69776/when-is-a-quadtree-preferable-over-spatial-hashing